EU GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

EGE UNIVERSITY

PhD THESIS

CONSTRUCTING GRAPH THEORETICAL STRUCTURES USING META-HEURISTIC ALGORITHMS

Züleyha AKUSTA DAĞDEVİREN

Supervisor : Prof. Dr. M. Serdar KORUKOĞLU

International Computer Department

Presentation Date : 01.12.2017

Bornova-İZMİR
2017

EGE UNIVERSITY GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

(PHD THESIS)

CONSTRUCTING GRAPH THEORETICAL STRUCTURES USING META-HEURISTIC ALGORITHMS

Züleyha AKUSTA DAĞDEVİREN

Supervisor: Prof. Dr. M. Serdar KORUKOĞLU

International Computer Department

Presentation Date: 01.12.2017

BORNOVA-İZMİR
2017

Züleyha AKUSTA DAĞDEVİREN tarafından Doktora Tezi olarak sunulan “Constructing Graph Theoretical Structures Using Meta-Heuristic Algorithms” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 01/12/2017 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	: Prof. Dr. M. Serdar KORUKOĞLU	[illegible]
Raportör Üye	: Prof. Dr. Aybars UĞUR	[illegible]
Üye	: Prof. Dr. Bülent TAVLI	[illegible]
Üye	: Doç. Dr. Doğan AYDIN	[illegible]
Üye	: Doç. Dr. Geylani KARDAŞ	[illegible]

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum "Constructing Graph Theoretical Structures Using Meta-Heuristic Algorithms" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

01 / 12 / 2017

[illegible]

Züleyha AKUSTA DAĞDEVİREN

ÖZET

ÜST-SEZGİSEL ALGORİTMALAR KULLANILARAK ÇİZGE TEORİK YAPILARIN OLUŞTURULMASI

AKUSTA DAĞDEVİREN, Züleyha

Doktora Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Prof. Dr. M. Serdar KORUKOĞLU
Aralık 2017, 94 sayfa

Çizge teorik yapıların kullanımı sayesinde çeşitli ağlar üzerinde pek çok önemli işlem gerçekleştirilebilmektedir. Bu yapılardan biri olan hakim kümenin telsiz duyarga ağlarında kümeleme, saldırı tespiti ve omurga oluşturma; telsiz örgü ağlarında ağ geçitlerinin yerleştirilmesi; internet üzerinde bilgi geri getirimi için çok sayıdaki dökümanın özetlenmesi ve sorgu seçilmesi gibi önemli uygulamaları bulunmaktadır.

En küçük ağırlıklı bağlı hakim kümenin (EABHK) bulunması NP-Zor bir problemdir. Bundan dolayı yakınsama algoritmaları ve üst-sezgisel algoritmalar polinom zamanda etkili sonuçlar verebilmektedir. Literatürde bu konu ile ilgili çeşitli çalışmalar yapılmış olsa da üst-sezgisel algoritmalar kullanılarak yönsüz çizgeler için EABHK bulunmasıyla ilgili bir çalışma yapılmamıştır. Bu tez çalışmasında EABHK problemi için iki farklı üst-sezgisel algoritma önerilmiştir. Bu algoritmalar Hibrit Genetik Algoritma (HGA) ve Popülasyon Tabanlı Tekrarlı Açgözlü (PTTA) Algoritmadır. HGA, genetik arama ile açgözlü sezgisel yaklaşımı birleştiren bir kararlı-durum algoritmasıdır. PTTA algoritma her bir bireye bozma ve açgözlü bir şekilde yeniden yapılandırma süreçleri uygulayarak popülasyonu iyileştirmektedir. Önerilen algoritmaların performansları diğer açgözlü sezgisel ve kaba kuvvet algoritmaları ile karşılaştırılmıştır. Önerilen algoritmalar çözüm kalitesi ve uygulama süresi açısından çok iyi performans göstermiştir.

Anahtar sözcükler: En küçük ağırlıklı bağlı hakim küme, hibrit genetik algoritma, popülasyon tabanlı tekrarlı açgözlü algoritma, yönsüz çizgeler, üst-sezgisel algoritmalar.

ABSTRACT

CONSTRUCTING GRAPH THEORETICAL STRUCTURES USING META-HEURISTIC ALGORITHMS

AKUSTA DAĞDEVİREN, Züleyha

PhD in International Computer Department.
Supervisor: Prof. Dr. M. Serdar KORUKOĞLU
December 2017, 94 pages

Through the use of graph theoretical structures, many important operations can be performed on various networks. Dominating set which is one of these structures, has many important applications such as clustering, intrusion detection and backbone formation in wireless sensor networks; placement of gateways in wireless mesh networks; summarizing multiple documents and selecting queries for information retrieval on the internet.

Finding the minimum weighted connected dominating set (MWCDS) is an NP-Hard problem. Hence, approximation algorithms and meta-heuristic algorithms can give effective results in polynomial time. Although there are numerous studies related to this subject in the literature, there is no study about finding the MWCDS for undirected graphs using meta-heuristic algorithms. In this thesis study, two different meta-heuristic algorithms are proposed for the MWCDS problem. These algorithms are Hybrid Genetic Algorithm (HGA) and Population-Based Iterated Greedy (PBIG) Algorithm. HGA is a steady-state algorithm that combines a genetic search with a greedy heuristic approach. PBIG algorithm improves the population by applying a deconstruction process and a reconstruction process to each individual in a greedy way. The performances of the proposed algorithms are compared with other greedy heuristics and brute force algorithms. The proposed algorithms performed very well in terms of solution quality and execution time.

Keywords: Minimum weighted connected dominating set, hybrid genetic algorithm, population-based iterated greedy algorithm, undirected graphs, meta-heuristic algorithms.

ACKNOWLEDGEMENT

I would like to thank my advisor Prof. Dr. M. Serdar KORUKOĞLU for his valuable knowledge, expertise and guidance. I would like to thank my master thesis advisor Assoc. Prof. Dr. M. Gökhan CİNSDİKİCİ for his support and motivation. I also want to thank Prof. Dr. Aybars UĞUR and Assoc. Prof. Dr. Doğan AYDIN for their suggestions and comments in thesis monitoring comittee meetings.

I also thank Scientific and Technical Research Council of Turkey (TUBITAK) for the BIDEB 2211-C program for the PhD scholarship.

Lastly, I would like to thank all my family for their endless love and encouragement. Especially to my beloved mother for the support of every aspect of my life, to my daughter for always cheering me up and to my son who will be born soon. I would like to express appreciation to my husband for his endless support and patience.

LIST OF CONTENTS

LIST OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF SYMBOLS AND ABBREVIATIONS

Symbols	Explanation
C	Chromosome
C_i	*i*th bit of chromosome
depth(u)	Depth value of node *u*
E	Edge set
G	Graph
G'	Subgraph
low(u)	Low value of node *u*
n	number of vertices
m	number of edges
parent(u)	Parent of node *u*
V	Vertex set
V'	Vertex induced subgraph
$W_T(V)$	Total weight of vertices in *V* set
$\Gamma(v)$	Open neighborhood of node *v*
$\Gamma[v]$	Closed neighborhood of node *v*
$\Gamma(v)_c$	Open neighborhood of node *v* with color *c*
$\Gamma[v]_c$	Closed neighborhood of node *v* with color *c*

LIST OF SYMBOLS AND ABBREVIATIONS (continued)

Abbreviations

ACO	Ant Colony Optimization
BF	Brute force
CDS	Connected Dominating Set
CPU	Central Processing Unit
DFS	Depth-First Search
DS	Dominating Set
GCDS	Geometric Connected Dominating Set
GD	Greedy Degree
GR	Greedy Ratio
GW	GreedyWeight
HGA	Hybrid Genetic Algorithm
IG	Iterated Greedy
ILS	Iterated Local Search
MCDS	Minimum Connected Dominating Set
MDS	Minimum Dominating Set
MWCDS	Minimum Weight Connected Dominating Set
MWDS	Minimum Weight Dominating Set

LIST OF SYMBOLS AND ABBREVIATIONS (continued)

Abbreviations

PBIG	Population-Based Iterated Greedy
PMBGA	Probabilistic Model-Building Genetic Algorithm
SLS	Subsidary Local Search
T-BF	Time-limited Brute Force
UDG	Unit Disk Graph
UG	Undirected Graph
WCDS	Weighted Connected Dominating Set
WDS	Weighted Dominating Set
WSAN	Wireless Sensor and Ad hoc Network

1. INTRODUCTION

In this section, minimum weight connected dominating set problem will be introduced, then wireless ad hoc and sensor networks with their applications will be given. Following this, hybrid genetic algorithm and population-based iterated greedy algorithm concepts will be explained. Finally, the organization and the contributions of the thesis will be presented.

1.1 Minimum Weight Connected Dominating Set Problem

Finding the dominating set (DS) and its variants in graphs is a very popular graph theoretical problem which has many application areas such as clustering, intrusion detection and backbone formation in wireless ad hoc and sensor networks (WASNs) (Chen and Liestman, 2002) (Subhadrabandhu et al., 2004) (Wu and Li, 1999), placement of gateways in wireless mesh networks (Aoun et al., 2006), wavelength division multiplexing deployment in optical networks (Houmaidi and Bassiouni, 2003), information retrieval to summarize multiple documents (Shen and Li, 2010) and query selection to obtain data from web (Wu et al., 2006).

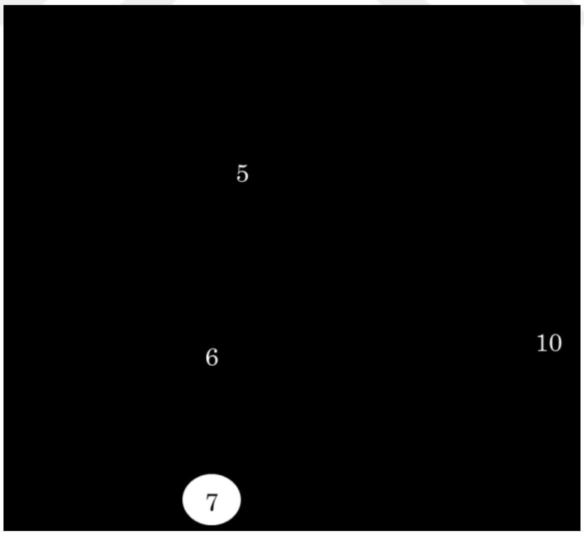


Figure 1.1. An Example DS

A DS is a set of nodes S such that every node in the network graph G is a neighbor of at least one element of S. The Minimum Dominating Set (MDS) problem is to find the S with minimum cardinality for a given network graph. More formally, MDS problem is finding a subset including vertices (nodes) $S \subseteq V$ where each node in $V \setminus S$ is a neighbor of at least one node in S for a given undirected graph $G(V,E)$ where E is the set of bidirectional edges (links) and V is the set of vertices. The nodes in S set are called dominators and the other nodes are called as dominatees or ordinary nodes. MDS problem is in NP-Hard complexity class (Cormen et al., 2009). Therefore, optimum solutions cannot be guaranteed in polynomial time. Heuristic and approximation algorithms can be applied to obtain near optimal solutions. Hence, various studies have been proposed to deal with this problem and many research studies are ongoing. Unless P=NP can be proved, it is anticipated that the popularity of the problem will continue.

An example graph including 10 nodes with one of its MDS S={5, 6, 10} having 3 elements is given in Figure 1.1. Please notice that {2, 3, 5, 6, 8, 9, 10}, {2, 3, 4, 6, 9, 10}, {1, 3, 6, 8, 10}, {5, 7, 8, 9} are DSs with 7, 6, 5 and 4 elements, respectively. Since the cardinalities of these sets are greater than 3 we cannot identify these sets as MDS. Other alternative MDS sets are {5, 6, 8} and {5, 6, 10}. Clustering a WASN is a very popular application of MDS problem. The members of MDS are cluster heads and other nodes are cluster member nodes. In other words, dominators are cluster heads and ordinary nodes are cluster members. By sending messages from one cluster to another rapidly and hierarchically, the cluster heads will oversee routing within and through the clusters. So, the aim of MDS problem in WASNs is to find the smallest set of efficient cluster heads.

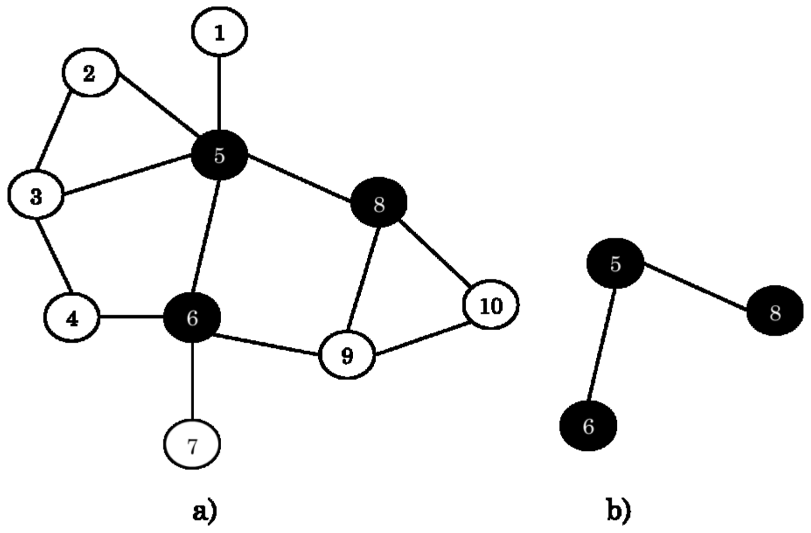


Figure 1.2. a) An Example CDS b) An Example Induced Subgraph

A connected DS is a subset of nodes where the elements of DS are connected through each other. More formally, if S is a DS and each node pair

$(v_i,v_j) \in S$ has at least one path including only nodes in *S*, then we can call S as a CDS. An alternative formal definition of CDS can be made by using induced subgraph concept. An example CDS is given in Figure 1.2.a. A vertex induced subgraph by $I \subseteq V$ is $G'=(I, E')$ in which *E'* is the set of edges $\{(v_x, v_y): ((v_x, v_y) \in E) \wedge (v_x \in I) \wedge (v_y \in I)\}$. An example vertex induced subgraph *G'* from *G* such that $I=\{5, 6, 8\}$ is given in Figure 1.2.b. A CDS *S* can be defined as a DS whose induced subgraph $G'=(V'=S, E)$ is connected. Finding Minimum CDS (MCDS) is an NP-Hard problem similar to MDS (Cormen et al., 2009).

An example graph with MCDS $S=\{5, 6, 8\}$ is given in Figure 1.2.a. $A=\{2, 3, 4, 6, 8, 9\}$ and $B=\{2, 5, 6, 9, 10\}$ are CDSs with 6 and 5 elements, respectively. Since the MCDS has 3 elements, A and B are CDS but they are not MCDS. Other alternative MCDS is {5, 6, 9}. CDSs provide many advantages in network applications such as ease of broadcasting and constructing virtual backbones (Subhadrabandhu et al., 2004). When CDS is used as a backbone in WASNs, the data collected by the dominators can be relayed through CDS. For CDS backbones with small size, the number of transmitted messages through the backbone is smaller which results in energy-efficient operation.

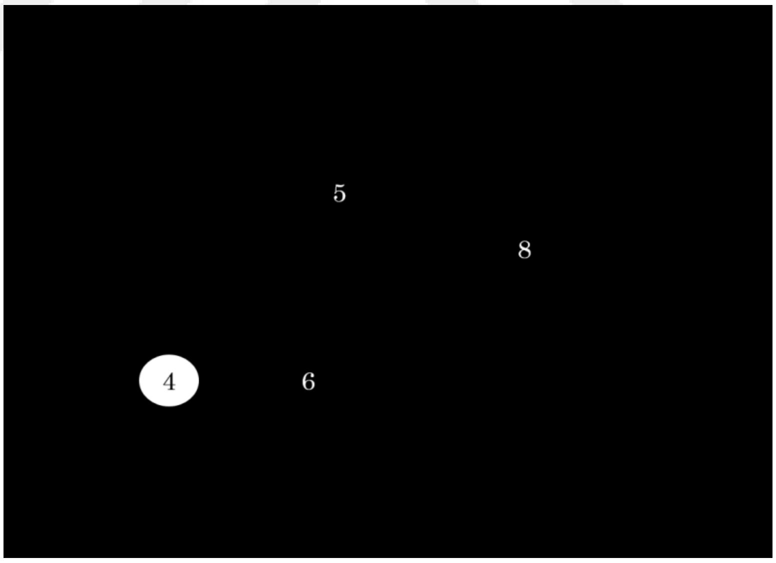


Figure 1.3. An Example WCDS

Generally, WASNs compose of battery-powered nodes which we will describe them in the following section in detail. Since the network lifetime depends on the lifetime of nodes, the energy efficient operation is of paramount importance. Transmission is the dominant factor of the energy consumption (Karl and Willig, 2005). Since the nodes in S are responsible to carry network traffic, they may run out of their batteries very earlier than other nodes. This situation may result catastrophes such that the network traffic can be significantly reduced when a dominator node connecting many other nodes fails. An important solution to this problem is choosing the nodes having high energies as dominators. A minimum weighted connected dominating set (MWCDS) backbone has been applied to overcome this problem (Wang et al., 2005). Different from CDS problem, MWCDS problem aims to minimize the total weight of the CDS. A formal definition of MWCDS problem can be formally defined as follows. The MWCDS problem is finding a CDS with minimum weight $W_T(V) = \sum_{v_i \in D} w(v_i)$ where w is a function $w{:}V \rightarrow R^+$ and D is the set of dominators.

An example MWCDS is given in Figure 1.3. Each node is numbered within the circle with its node id. The numbers near to each node are its energy in joule and its weight which is calculated as 1/energy. The set {5, 6, 8} is the only MWCDS in this example which has 0.37 cost (weight) in total. The set {5, 6, 9} is WCDS but not MWCDS because the total cost of this set is 0.40.

1.2 Wireless Ad Hoc and Sensor Networks

WASNs are composed of ten to thousands of tiny sensor nodes which are low cost and low power hardware (Akyildiz et al., 2002). Since these nodes can be equipped with wireless transceivers, networked sensors can be realized. These nodes generally have read access memory, read only memory, digital to analog converter, analog to digital converter, universal asynchronous receiver transmitter, interrupt controller and counter. Short range radio frequency, infrared, optical and other transmission techniques can be applied on them. The sensors can be interacted with the environment to sense the heat, light, acceleration and chemical materials. We can summarize some general properties of the networked sensor nodes as follows:

- Generally, sensor nodes are small to be deployed easily to the environment. An example sensor node is given in Figure 1.4. They use low power as their hardware and software technologies permits.

- Sensor nodes can process concurrently such as they can sense from the environment at the same time they can execute a scheduled operation.

- Sensor nodes are designed as simple and low cost devices. Unlike general purpose personal computers, generally sensor nodes are special purpose devices.

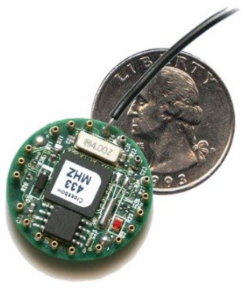


Figure 1.4. An Example Sensor Node (https://www.comsys.rwth-aachen.de/fileadmin/_migrated/pics/mica2dot.jpg)

Components of a sensor node can be seen in Figure 1.5. There are five components of this design which are central processing unit (CPU), sensors, low power transceiver, memory and power supply (Karl and Willig, 2005).

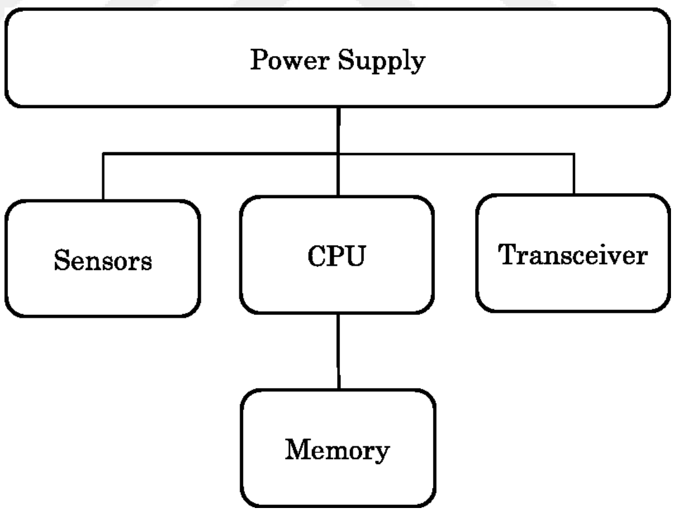


Figure 1.5. Components of a Sensor Node

WASNs are composed of sensor nodes that are self-organizing. An example WASN is given in Figure 1.6 (Karl and Willig, 2005). As seen in this figure, the data collected by a sensor node is relayed in a multi-hop manner over some other sensor nodes to the sink node. The sink node is a gateway node collecting data from other nodes and forwarding data to users. The sink node can send data through Internet or a satellite network. The user can send configuration data to the sensor network, thus the links between users and the sink node are bidirectional.

WASNs can be embedded in the environment to sense various data, thus they have lots of application types (Ning, 2005). One of the most important applications is the habitat monitoring. In great duck island application, the lifecycle of Storm Petre bird in great duck island is monitored by the researchers from University of California Berkeley and Intel Research Laboratory (Mainwaring et al., 2002). The PODS Project developed in Hawai University aims to investigate the endangered plant species (Biagioni and Bridges, 2002). The other popular application type of WASNs is healthcare applications. Schwiebert et al. (Schwiebert et al., 2001) used micro sensors to construct a prosthesis for blind people. Other types of applications are remote patient monitoring and drug use (Akyildiz et al., 2002). Also, WASNs can be used in home and office applications. Srivastava et al.'s kindergarden application aims to interact with children to teach various topics (Srivastava et al., 2001).

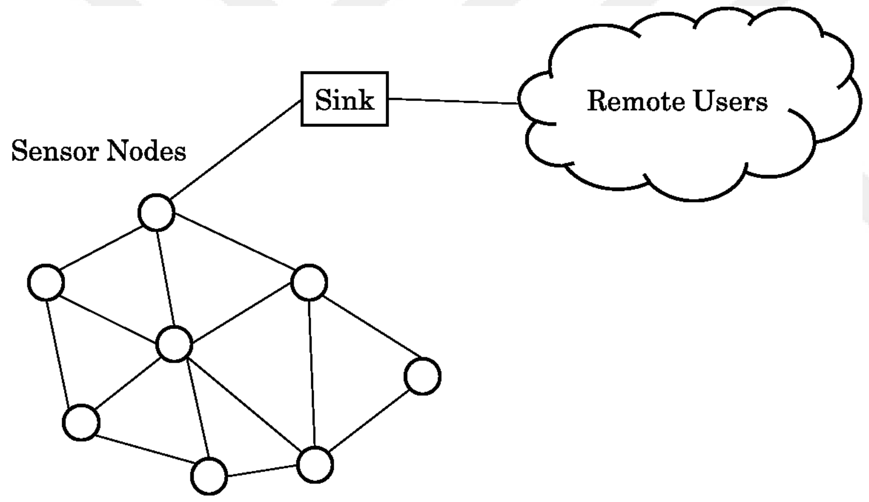


Figure 1.6. An Example Sensor Network

1.3 Hybrid Genetic Algorithms

Genetic algorithms (GAs) are optimization techniques that are inspired by the principle of evolution through genetic process (McCall, 2005). The concept of GA was proposed by John Holland (McCall, 2005). GA works on a population of chromosomes artificially created strings. These strings are usually in binary form and represent solution of problems. Each chromosome has a fitness value which shows how good the solution is. A GA starts with randomly produced chromosomes, selects chromosomes according to their fitness values and combines chromosomes to generate new offspring. The combination process is iterated until a stopping criterion is reached.

As aforementioned, in a GA, initially a random population is generated and it is evolved with time. Each chromosome is scored with its fitness value. Chromosomes with higher fitness values are preferred over the other ones. Besides chromosomes with the worst fitness values may be removed from the population. To diversify the population, offspring are produced from parent chromosomes. Elitism method provides the selection of the parents as chromosomes with the two highest fitness values. In roulette wheel method, the chromosomes with higher fitness values get higher probability for being selected as a parent. Child chromosome can be produced via single point crossover operator in which a random point p in the chromosome data [0,n] is chosen, the first part of the child chromosome [0,p] is produced by copying the [0,p] of the first parent chromosome and the second part of first part of the child chromosome (p,n] is produced by copying the (p,n] of the second parent chromosome. After a child chromosome is produced, a mutation can be applied. Algorithm continues until termination condition is met. An example algorithm is given in Algorithm 1.1.

Algorithm 1.1: An Example GA

```
1. Generate an initial random population of solutions.
2. Evaluate the fitness of all individuals.
3. while termination condition not met or generations not run out
4.     Select the parents by the roulette wheel method to produce new individuals.
5.     Apply single point crossover.
6.     Apply mutation.
7.     Evaluate fitness of new individuals.
8.     Generate a new population by inserting new individuals or replace low-fit parents with them.
9. end while
```

Hybrid GAs (HGAs) are based on genetic and other search methods that can complement each other to achieve an optimization goal (El-Mihoub et al., 2006). A local search method can be integrated with a GA to enhance search capabilities.

A local search method that is able to find local optimum can be integrated with a GA to perform local and global search efficiently. This efficiency can be declared as in terms of solution quality and the time needed to finish entire operation. An efficient search method increases the fitness values of chromosomes. This enhancement leads to reduce the standard deviation of the members of the population. In this case, the hybrid algorithm can be still efficient even the population size is small. In pure GAs, mutation and crossover operations may result illegal solutions. When a problem specific search algorithm is integrated to a GA, these illegal solutions are prevented. For example, an intelligent search method can be used with a GA to recover infeasible solutions (Konak and Smith, 1999). Sometimes the calculation of fitness values can be complex and time consuming (Jin, 2005). In this situation, an approximation based search method can be used to estimate the fitness values. Crossover and mutation operations can be replaced by problem specific methods to increase the quality of search operation. For example, in probabilistic model-building genetic algorithms (PMBGA) the crossover and mutation operations of a pure GA is replaced with a model that is based on the estimation of promising solutions (Pelikan et al., 1999). Compact genetic algorithm, population-based incremental learning, univariate marginal distribution algorithm and bivariate marginal distribution algorithms are examples of PMBGA. An efficient technique can be incorporated within a GA to optimize control parameters to improve the search performance.

1.4 Population-Based Iterated Greedy Algorithms

An iterated local search (ILS) has three main components (Gonzalez, 2007). Firstly, to find local optimum *c*, a subsidiary local search (SLS) procedure is used. Secondly, to escape from local optimum, a perturbation procedure is executed. Finally, to decide whether the procedure is continued with *c*, an acceptance condition is used. An example ILS algorithm is given in Algorithm 1.2. ILS algorithm is a very promising technique for solving various hard combinatorial problems such as traveling salesman problem (Samet, 1990).

Greedy selection is a very widely used algorithmic technique that provides many solutions to the well-known problems. Greedy selection is also at the heart of iterated greedy (IG) algorithms which is a variant of ILS (Gonzalez, 2007). IG differs from ILS that perturbation and local search phases are changed with constructive and destructive search. In a destruction phase, solutions are removed randomly or according to a heuristic. After this phase has accomplished, a partial solution is generated from a solution. In a construction phase, a new solution is added according to a greedy heuristic. Similar to ILS, IG has an acceptance condition to decide whether the new solution will be used in further iterations. Population-based IG (PBIG) is a variant of IG in which the algorithm works on population of candidate solutions. IG and its variants are very efficient techniques and provide state-of-the-art performance for flow-shop scheduling (Linde et al., 1980) and set covering (Xiang et al., 1994; Gray et al., 1980) problems.

Algorithm 1.2: An Example ILS.

```
1. Construct initial candidate solution c.
2. Execute SLS on c.
3. while termination condition is not met
4.     t ← c.
5.     Execute perturbation on c.
6.     Execute SLS on c based on acceptance condition.
7.     Keep c or revert c ← t.
8. end while
```

1.5 Contributions of the Thesis

The contributions of this thesis are listed as follows:

- We proposed a hybrid genetic algorithm which incorporates a greedy heuristic with a genetic approach to solve MWCDS problem. The algorithm runs on a population of solutions and aim to improve the solution quality by applying cross over, mutation, repair and minimization operations sequentially and iteratively.

- We proposed a population-based iterated greedy algorithm which executes iteratively by applying destruction and construction phases. This strategy can generally improve the solution quality by preventing getting stuck in a local minimum solution.

- Proposed algorithms are the first population-based optimization algorithms for MWCDS problem on undirected graphs.

- We analyzed the time complexities of HGA and PBIG algorithms. We provided implementations of the proposed algorithms, greedy heuristics and brute force algorithms in Java. Greedy heuristics are Greedy Ratio (GR), Greedy Weight (GW) and Greedy Degree (GD). Brute force algorithms are a pure implementation named as Brute Force (BF), and a time-limited version of BF (T-BF).

- We used two datasets of graphs such that the first dataset is a popular dataset used by the researchers and the second dataset is generated in this thesis. The reason why we generated the second dataset is some of the graphs in first dataset are unconnected.

- From our performance evaluations, we obtained that GR has the best performance in terms of MWCDS weight among the other greedy heuristics (GD and GW). These greedy heuristics have very close execution times. Our proposed algorithms have significantly better weight performance than GR for nearly all problem instances. Additionally, our proposed algorithms can find optimum results same as BF for small-size problem instances, at the same time our algorithms run very faster than BF. Moreover, when we compared the performances of our algorithms and T-BF, we found that our algorithms outperform T-BF in terms of both WCDS weight and execution time.

- Our measurement results taken from the proposed algorithms show that PBIG performs better in terms of WCDS weight and HGA is faster.

- The materials given in thesis are published in the following publication: *Zuleyha Akusta Dagdeviren, Dogan Aydin, Muhammed Cinsdikici, Two Population-based Optimization Algorithms for Minimum Weight Connected Dominating Set Problem, Applied Soft Computing, 59, pp. 644-658, Elsevier* (Dagdeviren et al., 2017).

1.6 Organization of the Thesis

The rest of the thesis is organized as follows:

In Section 2, related work is given. In this section, centralized dominating set algorithms, centralized connected dominating set algorithms and distributed algorithms for both dominating set and connected dominating set problems are mentioned and compared with proposed algorithms. Some important algorithms are explained in detail. At the end of this section, algorithms are summarized and listed in a table.

Proposed algorithms HGA and PBIG are described in Section 3. Firstly, background information related to the proposed algorithms are given. After that, the steps of the algorithms are explained in detail. Examples are given to show the operations of the proposed algorithms. Lastly, time complexities of the algorithms are analyzed in this section.

In Section 4, extensive performance evaluations of the proposed algorithms with their counterparts are given. This section is divided into Evaluation of Small-Size Problem Instances, Evaluation of Moderate-Size Problem Instances and Evaluation of Large-Size Problem Instances. The last section is Section 5 in which conclusions are drawn by summarizing main findings.

2. RELATED WORK

This section reviews the existing studies as centralized dominating set algorithms, centralized connected dominating set algorithms and distributed algorithms. At the end of the section, a summary of the literaure is given.

2.1 Centralized Dominating Set Algorithms

A greedy heuristic algorithm for set-covering problem is proposed by Chavatal (1979). The algorithm continues until all points are covered. For each set $|P_j| / C_j$ ratio is calculated where C_j is the cost and $|P_j|$ is the number of the points covered. The set which has the minimum ratio is selected in each step. We apply this heuristic in our proposed algorithms, because this heuristic can be used in constructing a weighted dominating set. For the minimum weighted dominating set problem, the approximation ratio of this heuristic becomes O(log $W_T(S)$) where $W_T(S)$ is the total weight of the optimum solution set S. The algorithm is given in Algorithm 2.1.

Algorithm 2.1: Chavatal's Algorithm

```
1. input: Sets of Points {$S_0$, … ,$S_n$}, Costs of Sets {$C_0$, …,$C_n$}
2. initially
3. $P_j$ : the number of points covered by set $S_j$.
4. $C_j$ : the cost of set $S_j$.
5. repeat
6.    For each set $S_j$ calculate the weight ratio of $R_j$ as $C_j$/ $|P_j|$.
7.    Choose the set $Sj$ with the minimum $Rj$ and cover the points in S$j$.
8. until all points are not covered
9. Return chosen sets
```

Algorithm 2.2: Potluri and Singh's Main Algorithm

```
1. input: $p_c$ (probability of crossover),
2. Generate initial population and set as $P$
3. $f \leftarrow$ fitness of best member of $P$
4. $m \leftarrow$ Best member of $P$
5. $iteration\_count \leftarrow 0$
6. while $iteration\_count <$ MAX do
7.     if $p < p_c$ then
8.       Select $b_1$ and $b_2$ by binary tournament
9.       $X \leftarrow$ crossover($b_1$, $b_2$)
10.      $X \leftarrow$ mutate($X$)
11.    else
12.      Generate $X$ randomly
13.    end if
14.    $X \leftarrow$ Repair_Procedure($X$)    // see Algorithm 2.3
15.    $X \leftarrow$ Minimize_Procedure($X$)   // see Algorithm 2.4
16.    if $X$ is unique then
17.      Remove the worst member of population $P$
18.      Add $X$ to the population $P$
19.      if fitness of $X < f$ then
20.        $f \leftarrow$ Fitness of $X$
21.        $m \leftarrow X$
22.      end if
23.      $iteration\_count \leftarrow iteration\_count + 1$
24.    end if
25. end while
26. Return m
```

Jovanovic et al. (2010) propose an ant colony optimization (ACO) algorithm to apply for the minimum weight dominating set problem. Their algorithm is compared with the greedy algorithm for different edge densities, weight distributions and node counts. The obtained results present that the algorithm is better than the greedy approach. Potluri and Singh (2013) propose hybrid metaheuristic algorithms for minimum weight dominating set problem on undirected graphs. The main algorithm is given in Algorithm 2.2. In this algorithm, firstly an initial population of chromosomes is generated. Then, for *MAX* number of iteration times the following operations are applied.

Algorithm 2.3: Potluri and Singh's Repair Procedure

```
1. input: $p_h$ (probability of repair)
2. $T \leftarrow V \setminus X$
3. if $p < p_h$ then
4.    while $X$ is not a dominating set do
5.       $maximum \leftarrow 0$
6.       for $t \in T$ do
7.          if $maximum < W(t) / w(t)$ then  // w(t) is the
            weight of node t. W(t) is the total weight of the
            dominatee neighbors of node t.
8.             $maximum \leftarrow W(t) / w(t)$
9.             $v \leftarrow t$
10.         end if
11.      end for
12.      $X \leftarrow X \cup v$
13.      $T \leftarrow T \setminus v$
14.   end while
15. else
16.   while $X$ is not a dominating set do
17.      v ← Select randomly from $T$
18.      $X \leftarrow X \cup v$
19.      $T := T \setminus v$
20.   end while
21. end if
22. Return $X$
```

A new chromosome is generated either by applying crossover to two chromosomes that are selected by binary tournament selection, or generating randomly. The generated chromosome is repaired to provide a weighted dominating set from this solution. After that, the chromosome is minimized in order to remove the redundant nodes in minimum weighted dominating set. If the generated chromosome does not exist in the population, the worst member of the population is removed and the generated chromosome is added to the population.

At the end of the algorithm, the chromosome having the maximum fitness value is returned as solution.

Algorithm 2.4: Potluri and Singh's Minimize Procedure

```
1. input: $p_r$ (probability of deletion)
2. $S$ is the set of dominators where the neighbors of each
element of $S$ is covered by other dominators.
3. while $S \neq \emptyset$ do
4.    if $p < p_r$ then
5.        $r \leftarrow \arg\max_{t \in S} w(t) / d(t)$   // d($t$): degree of node $t$
6.    else
7.        $r \leftarrow$ Select randomly from $S$
8.    end if
9.    $X \leftarrow X \setminus r$
10.   recalculate $S$
11. end while
12. Return $X$
```

Potluri and Singh's Repair Procedure is given in Algorithm 2.3. In this algorithm, a dominating set is constructed from a partial solution. The algorithm either repairs the partial solution by adding the node having the maximum weight ratio to the partial solution, or by adding a random node to the partial solution. This operation iteratively continues until the partial solution becomes a full solution, in another word, a dominating set. Although a dominating set is constructed after repairing the solution, redundant nodes may exist in the dominating set.

Potluri and Singh's Minimize Procedure is given in Algorithm 2.4. Redundant nodes are removed in this procedure. A redundant dominator is a dominator whose all ordinary nodes are covered by other dominators. In this case, if this redundant dominator is removed from the dominating set, the remaining dominators still constitute a dominating set. In Minimize Procedure given in Algorithm 2.4, firstly redundant nodes are identified. Then, the algorithm either removes the redundant dominator having the greater w(t)/d(t) ratio or removes a redundant dominator randomly. After each removal, the set of redundant dominators are recalculated. If there are no redundant nodes, the algorithm quits from the operation. Otherwise, the operation continues iteratively.

Algorithm 2.5: Bouamama et al.'s Main Algorithm

```
1. input: population_size, probability, determinism_rate
2. Generate initial population w.r.t input population size and set as $S$ // see Algorithm 2.6
3. while termination condition is not true do
4.     $S_1 \leftarrow \emptyset$
5.     for each $C_i \in S$ do
6.         $C_i^m \leftarrow$ PartiallyDestroy($C_i$, probability)
7.         $C_i^n \leftarrow$ GreedyWeightedVertexCover($C_i^m$, determinism_rate) // see Algorithm 2.7
8.         $S_1 \leftarrow S_1 \cup \{ C_i^n \}$
9.     end for
10.    $S \leftarrow$ Accept($S$,$S_1$)
11. end while
12. Return argmin{ weight($Ci$) | $Ci \in S$, $i$=1, . . . , population_size}
```

Nitash and Singh (2014) propose an artificial bee colony algorithm for solving minimum weight dominating set problem for undirected weighted graphs. The results of the algorithm show that their algorithm performs better than other metaheuristics in literature. Bouamama et al. (2012) propose a population-based iterated greedy (PBIG) algorithm for tackling the minimum weight vertex cover problem on vertex-weighted undirected graphs. The aim of vertex cover problem is to find a set of vertices C such that every edge in graph is incident to at least one vertex in C. Vertex cover problem resembles the dominating set problem and it has various applications such as clustering, backbone formation and link monitoring in ad hoc networks. Bouamama et al.'s Main Algorithm is given in Algorithm 2.5. The algorithm starts with generating an initial population. This population generation algorithm is given in Algorithm 2.6. The algorithm proceeds by partially destroying and reconstruction of each solution in the population. These new solutions are added to a new population. After that, the new population and the old population are merged. These operations are iteratively executed until the termination condition is satisfied. At the end of the algorithm, the solution having the minimum weight is returned. Bouamama et al. assess the performance of their algorithm on all benchmark instances that have been considered by Shyu et al. (2004).

Algorithm 2.6: Bouamama et al.'s GenerateInitial Population Procedure

```
1. inputs: population_size
2. $S \leftarrow \emptyset$
3. $C_0 \leftarrow \emptyset$
4. for i=1, . . . , population_size do
5.     $C_i$ ← GreedyWeightedVertexCover($C_0$, determinism_rate) // see Algorithm 2.7
6.     $S \leftarrow S \cup \{C_i\}$
7. end for
8. return $S=\{C_1, C_2, . . . , C_{population\_size}\}$
```

The GreedyWeightedVertexCover Procedure of Bouamama et al.'s Vertex Cover Algorithm is given in Algorithm 2.7. In this algorithm, firstly the set of nodes which are incident to uncovered edges are identified. While this set is not empty, the node having the smallest weight or the node having the second smallest weight is removed from this set and added to the vertex cover.

After vertex cover is constructed, the redundant nodes are removed from the vertex cover. A redundant node is a node which is in vertex cover and whose all edges are covered by other nodes. While redundant node set is not empty, the node having the greatest weight or the node having the second greatest weight is removed from the redundant node set. A randomized version of PBIG algorithm is proposed by Bouamama and Blum (2015) for constructing minimum weight dominating sets.

Zhu et al. (2012) propose the first polynomial time approximation algorithm that achieves a (1+ε)-approximation for any ε>0 to solve the minimum weighted dominating set problem with smooth weights on unit disk graphs. A hybrid self-adaptive evolutionary algorithm is proposed by Lin (2016) for formulating the minimum weight dominating set.

Although the mentioned approaches in this chapter can be used to construct weighted dominating sets in WSNs, they do not provide a backbone structure since WCDS formation is not maintained.

Algorithm 2.7: Bouamama et al.'s GreedyWeighted VertexCover Procedure

```
1. Inputs: solution, determinism_rate
2. $U \leftarrow$ Set of nodes that are not in vertex cover. Each node in this set is incident to at least one uncovered edge.
3. while $U \neq \emptyset$ do
4.    $v_1 \leftarrow \text{argmin}\{\text{weight}(v) \mid v \in U\}$
5.    $v_2 \leftarrow \text{argmin}\{\text{weight}(v) \mid v \in U \setminus \{v_1\}\}$
6.    probability ← generate random probability
7.    if probability ≤ determinisim_rate then $v_b \leftarrow v_1$
8.    else $v_b \leftarrow v_2$
9.    end if
10.   $solution \leftarrow solution \cup \{v_b\}$
11.   Recalculate $U$
12. end while
13. $R \leftarrow$ Set of nodes that are in vertex cover. All edges of each node in this set should be covered by two nodes.
14. while $R \neq \emptyset$ do
15.   $v_1 \leftarrow \text{argmax}\{\text{weight}(v) \mid v \in R\}$
16.   $v_2 \leftarrow \text{argmax}\{\text{weight}(v) \mid v \in R \setminus \{v_1\}\}$
17.   probability ← generate random probability
18.   if probability ≤ determinisim_rate then $v_b \leftarrow v_1$
19.   else $v_b \leftarrow v_2$
20.   end if
21.   $solution \leftarrow solution \setminus \{v_b\}$
22.   Recalculate $R$
23. end while
24. Return solution
```

2.2 Centralized Connected Dominating Set Algorithms

CDS is a very popular graph theoretic structure for WASNs since it can be very useful in backbone formation operation. Unit disk graph (UDG) is used in WSANs when the transmission range of a node is assumed to be an ideal circle.

A genetic algorithm for power aware minimum CDS (MCDS) problem in WASNs is proposed by Kamali and Safarnourollah (2006). Their algorithm is used for if there is any power aware CDS with size *k* on a UDG. A genetic algorithm is proposed by More and Mangalwede (2013) for CDS problem in WSANs modeled as UDG. Zou et al. (2009) propose two approximation algorithms for MWDS and MWCDS problems on a unit disk graph. A (4 +ε)-approximation algorithm for an MWDS based on a dynamic programming algorithm for a Min-Weight Chromatic Disk Cover is presented. They also propose a (1+ε)-approximation algorithm for the connecting part by showing a polynomial-time approximation scheme for a Node-Weighted Steiner Tree problem when the given terminal set is c-local and thus obtain a (5+ε)-approximation algorithm for an MWCDS.

Khalil and Ozdemir (2015a) propose a genetic algorithm to construct MCDS on UDGs. Another study (Khalil and Ozdemir, 2015b) is similar the previous one. The only difference between them is the consumed energy while nodes are sending messages to sink is not minimized; instead the data reliability between the nodes in CDS and the dominatees is aimed. The aim of this study is minimizing the number of nodes in CDS and the consumed energy while sending messages to sink.

The algorithms mentioned so far in this section failed to model the real wireless transmission that is not an ideal circle in most cases such as interference and noise are present (Khun et al., 2003). Our proposed algorithms run on UG model which can capture the features of real wireless transmission better than the UDG model.

Tree Growing Algorithm is proposed for CDS problem by Guha and Kuller (1998). Their algorithm is an efficient approximation algorithm constructing a CDS on UGs. The nodes are given colors as BLACK, GRAY and WHITE where BLACK is a dominator node, GRAY is an ordinary node with having at least one BLACK neighbor and WHITE is an ordinary node without having a BLACK neighbor. The algorithm uses a heuristic which considers the number of WHITE neighbors. All nodes in graph are WHITE initially. The node having the maximum number of WHITE neighbors is selected as dominator. Then its color becomes BLACK and its neighbors are colored GRAY. To provide connectivity between BLACK nodes, the following selections are made from GRAY nodes. The selection criterion is again the maximum number of WHITE neighbors. The selected nodes are colored BLACK and its neighbors are colored GRAY. These steps repeat until all the nodes are dominated which means there is no WHITE node in the graph. The algorithm is given in Algorithm 2.8.

Algorithm 2.8: Guha's Connected Dominating Set Algorithm

```
1. Definition of scan operation: Let node $u$ be a GRAY node. First color node $u$ BLACK and then color WHITE neighbors of node $u$ GRAY.
2. Definition of the gain of a scan operation: Number of nodes which are colored GRAY
3. Color all nodes WHITE
4. Color the node $v$ having the greatest degree BLACK and color node $v$'s neighbors GRAY
5. while there are WHITE nodes do
6.     Apply the scan operation having the largest gain
7. end while
```

A memetic algorithm for the minimum weighted edge dominating set problem is proposed by Abdel-Aziz et al. (2013). An edge (*u*, *v*) in *G* dominates itself and any other edge adjacent to (*u*, *v*). An edge dominating set is a set of edges which dominates all the other edges in the graph *G*. In this algorithm, three fitness functions are used and a search method is developed. In our study, we aim to construct node weighted connected dominating set instead of edge weighted dominating set since the main goal is to select backbone nodes with high energy.

Alkhalifah and Wainwright (2004) propose a genetic algorithm that can be applied to different graph theoretical problems such as geometric CDS (GCDS) problem for wireless networks. Their approach is called as the nearest four neighbors heuristic that can be applied on traveling tourist man, capacitated kcenter and capacitated p-median problems. Although this algorithm performs well in constructing GCDS, the main drawback of this study is the nodes are not weighted thus energy-efficient backbone construction is not maintained.

Another ACO algorithm with pheromone correction strategy is proposed by Jovanovic and Tuba (2013) for constructing MCDSs. Their algorithm is simple one-step ACO method based on greedy heuristic. This heuristic is based on pheromone correction strategy. Yu et al. (2016) propose an algorithm for formulating CDSs in cognitive radio networks. Liu et al. (2016) propose the first constant factor approximation algorithm for constructing minimum-sized partial CDSs in growth-bounded graphs. A 5-approximation algorithm to solve CDS problem for WSANs is proposed by Al-Nabhan et al. (2016). He et al. (2011) propose a genetic algorithm for constructing a reliable minimum CDS in probabilistic wireless networks. A genetic algorithm for load-balanced CDS construction in WSANs is proposed by He et al. (2012). Gendron et al. (2014) propose benders decomposition, branch-and-cut and hybrid algorithms for MCDS problem.

The algorithms mentioned in this chapter generally aim to minimize the size of CDS; the nodes are unweighted so that energy-efficient backbone formation is omitted.

2.3 Distributed Algorithms

Wang et al. (2005) propose a distributed low-cost backbone formation algorithm for WASNs. The algorithm has two phases. In the first, a weighted maximal independent set algorithm is constructed. This algorithm is based on Chatterjee's algorithm (Chatterjee et al., 2002). First phase of Wang et al.'s algorithm is given in Algorithm 2.9. Initially, all nodes are WHITE. The nodes having the lowest weight among their neighbors send *IamDominator* message to their neighbors and become a *PossibleDominator.* If a node receives an *IamDominator* message to one of its neighbors, it colors itself GRAY and it sends an *IamDominatee* message. When a WHITE node receives an *IamDominatee* message, it deletes the sender of this message from the list of WHITE nodes and sends an *IamDominator* message and becomes a *PossibleDominator* if it has the minimum weight among its WHITE neighbors. At the second step of the first phase *PossibleDominator* nodes learn the weights of its neighbors at most 2 hops away from them. Then, each *PossibleDominator* node applies a set cover based method to cover itself and its one-hop neighbors. If a *PossibleDominator* finds that its one-hop neighbors including itself can be covered by other nodes with less total weight it quits becoming a *PossibleDominator*. Otherwise, this *PossibleDominator* node becomes a *Dominator*. At the end of the first phase, *Dominator* nodes constitute a DS but not definitely a CDS.

At the second phase, dominator nodes are connected by selecting new dominator nodes. In this phase, dominator nodes which are at most 2 hops away learn the costs of the paths between each other. During this operation, ordinary nodes relay the messages of the dominators. The cost between two dominator nodes is found by summing the weights of ordinary nodes between these dominators. After dominator nodes learn the costs, they execute a distributed minimum spanning tree algorithm. The second phase of Wang et al.'s algorithm is given in Algorithm 2.10.

Algorithm 2.9: First Phase of Wang et al.'s Algorithm

```
1. Find a minimal maximal independent ($I$) set given given by Chaterjee. Let dominators be the elements of $I$.
2. Each dominator node $d$ executes a set cover algorithm on neighborhood sets (each neighborhood set is identified by the id of that node) of the graph $G'=(V',E')$ in which $V'$ is the set of all nodes at most 2 hops away from node $v$. $E'$ is the set of edges connecting nodes in $V'$ excluding the edges between nodes $x$ and $y$ where $x$ and $y$ are two hops away from $d$.
3. if node $d$ is not included in the solution set
4.     node $d$ quits from dominating set
5.     send a message to each node in the solution set to inform that they are dominators.
6. end if
```

Algorithm 2.10: Second Phase of Wang et al.'s Algorithm

1. A graph $H=(X,Y)$ is constructed where X is the set of dominator nodes and Y is the set of edges between dominator nodes which are at most 2 hops away from each other and only minimum cost paths between each other are included in Y.

2. Dominator nodes run a distributed minimum spanning tree algorithm on H.

An intelligent algorithm based on distributed learning automata is proposed by Torkestani and Meybodi (2010a) for constructing backbone in wireless ad hoc networks. Torkestani and Meybodi (2010b) propose another distributed learning automata approach for clustering WSANs. An algorithm based on learning automata is proposed by Torkestani and Meybodi (2012) for finding MWCDS in stochastic graphs. Ramalakshmi and Radhakrishnan (2015) study on WDS based routing for ad hoc communications in emergency and rescue scenarios. For a detailed survey of these studies please refer to Yu et al (2013). Since we focus on the design and implementation of central algorithms in this thesis, we omit distributed approaches.

2.4 Summary of Algorithms

The algorithms given in Section 2 are summarized in Table 2.1. The algorithms are sorted with respect to the date. Type of the algorithms, target problems, the graphs that algorithms running on and the locality types of algorithms (centralized/distributed) are given.

Table 2.1. Summary of the studies in literature

Paper	**Type of the Algorithm**	**Target Problem**	**Graph Type**	**Locality Type**
Chavatal, 1979	Approximation Algorithm	Minimum Weight Set Cover	Designed for Set Data Structure. Can be used in Node Weighted Undirected Graph.	Centralized
Guha and Khuller, 1998	Approximation Algorithm	Minimum Weight Connected Dominating Set	Node Weighted Undirected Graph	Centralized
Alkhalifah and Wainwright , 2004	Genetic Algorithm	Minimum Connected Dominating Set	Undirected Graph	Centralized
Shyu et al., 2004	Ant Colony Algorithm	Minimum Weight Vertex Cover	Node Weighted Undirected Graph	Centralized
Wang et al., 2005	Approximation Algorithm	Minimum Weight Connected Dominating Set	Node Weighted Unit Disk Graph	Distributed
Kamali and Safarnourol lah, 2006	Genetic Algorithm	Minimum Connected Dominating Set	Unit Disk Graph	Centralized
Zou et al., 2009	Approximation Algorithm	Minimum Weight Dominating Set, Minimum Weight Connected Dominating Set	Node Weighted Unit Disk Graph	Centralized
Jovanovic et al., 2010	Ant Colony Algorithm	Minimum Weight Dominating Set	Node Weighted Undirected Graph	Centralized
Torkestani and Meybodi, 2010a	Learning Automata	Minimum Connected Dominating Set	Undirected Graph	Distributed

Paper	Type of the Algorithm	Target Problem	Graph Type	Locality Type
Torkestani and Meybodi, 2010b	Learning Automata	Minimum Weight Connected Dominating Set	Weighted Undirected Graph	Distributed
He et al., 2011	Genetic Algorithm	Minimum Connected Dominating Set	Probabilistic Network	Centralized
He et al., 2012	Genetic Algorithm	Connected Dominating Set	Undirected Graph	Centralized
Torkestani, 2012	Learning Automata	Minimum Weight Connected Dominating Set	Stochastic Graph	Distributed
Bouamama et al., 2012	Population-based Iterated Greedy Algorithm	Minimum Weight Vertex Cover	Node Weighted Undirected Graph	Centralized
Zhu et al., 2012	Approximation Algorithm	Minimum Weight Dominating Set	Node Weighted Unit Disk Graph	Centralized
Jovanovic and Tuba, 2013	Ant Colony Algorithm	Minimum Connected Dominating Set	Undirected Graph	Centralized
Abdel-Aziz et al., 2013	Genetic Algorithm	Minimum Edge Weighted Dominating Set	Edge Weighted Undirected Graph	Centralized
More and Mangalwede, 2013	Genetic Algorithm	Minimum Connected Dominating Set	Unit Disk Graph	Centralized
Potluri and Singh, 2013	Hybrid Genetic Algorithm and Ant Colony Algorithm	Minimum Weight Dominating Set	Node Weighted Unit Disk Graph	Centralized
Gendron et al., 2014	Branch and Cut And Hybrid Algorithm	Minimum Connected Dominating Set	Undirected Graph	Centralized
Nitash and Singh, A., 2014	Ant Colony Algorithm	Minimum Weight Dominating Set	Node Weighted Undirected Graph	Centralized

Paper	**Type of the Algorithm**	**Target Problem**	**Graph Type**	**Locality Type**
He et al., 2015	Multi-objective Genetic Algorithm	Minimum Connected Dominating Set	Probabilistic Network	Centralized
Ramalakshmi and Radhakrishnan, 2015	Heuristic Algorithm	Minimum Weight Connected Dominating Set	Node Weighted Undirected Graph	Distributed
Khalil and Ozdemir, 2015a	Genetic Algorithm	Minimum Connected Dominating Set	Unit Disk Graph	Centralized
Khalil and Ozdemir, 2015b	Genetic Algorithm	Minimum Connected Dominating Set	Unit Disk Graph	Centralized
Bouamama and Blum, 2015	Randomized Population-based Iterated Greedy Algorithm	Minimum Weight Dominating Set	Node Weighted Undirected Graph	Centralized
Al-Nabhan, 2016	Approximation Algorithm	3-Connected Dominating Set	Unit Disk Graph	Centralized
Lin, 2016	Hybrid Evolutionary Algorithm	Minimum Weight Dominating Set	Node Weighted Undirected Graph	Centralized
Liu et al., 2016	Approximation Algorithm	Minimum Partial Connected Dominating Set	Growth-Bounded Graph	Centralized
Yu et al., 2016	Heuristic Algorithm	Connected Dominating Set	Unit Disk Graph	Centralized

3. PROPOSED ALGORITHMS

In this section, the preliminaries are given firstly. Then, the proposed two meta-heuristic algorithms are given in detail. The complexity analysis of the proposed algorithms is explained in the last subsection.

3.1 Preliminaries

If the removal of a vertex disconnects a graph, we call that vertex as a cut vertex. An example UG is given in Figure 3.1 where node 4 is the cut vertex. When node 4 is removed from the graph, there will be two disconnected components as: {1, 2, 3} and {5, 6, 7, 8}.

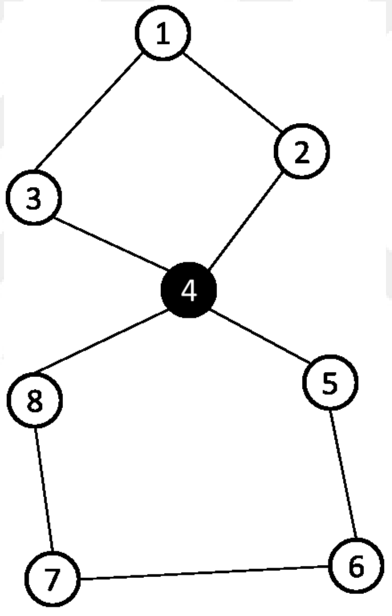


Figure 3.1. An Example UG with a Cut Vertex

Cut vertices can be detected by Hopcroft and Tarjan's linear time algorithm (Hopcroft and Tarjan, 1974) which is based on depth-first search (DFS) algorithm. The algorithm starts from a root node. In this algorithm each node v is associated with parent, depth and low values. The parent(v) of node v is the node that is visited just before node v. The depth(v) of node v is found as depth(v)=depth(parent(v))+1 where the depth value of the root is 1. The depth value of node v indicates the distance of node v to the root in the DFS tree. The low(v) of node v is found as minimum{depth(v), the low values of children of node v, the depth values of node v's neighbors excluding node v's children and parent(v)}. The low value of node v indicates the minimum depth value of the node that is reachable from subtree of node *v*. The main steps of the Hopcroft and Tarjan's cut vertex detection algorithm is given in Algorithm 3.1.

Algorithm 3.1: Tarjan's Cut Vertex Detection Algorithm

```
1. DFS is executed.
2. The root node is a cut vertex if it has more than one child in
   the DFS tree.
3. The node $v$ (which is not a root node) is a cut vertex if at least
   one of its children $u$ has the low($u$) $\geq$ depth($v$) property.
```

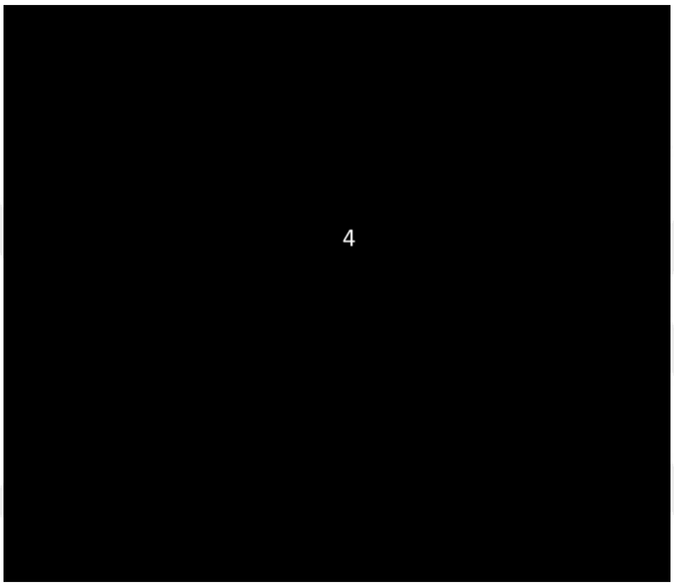


Figure 3.2. Example Operation of Cut Vertex Detection Algorithm

This algorithm can be implemented with modifications to the standard DFS algorithm and it has O(V+E) time complexity. An example operation of the cut vertex detection algorithm is given in Figure 3.2. Node 1 is the root node which is shown with double circles. Parent-child relationships are depicted with directed edges. Other edges (dashed edges) are not included in DFS tree but they belong to the graph. The visiting order of nodes in DFS tree is 1, 2, 4, 3, 5, 6, 7 and 8. Since node 8 is directly connected to node 4 and these nodes do not have a parent-child relationship, the low value of node 8 is 3. Because node 7's, node 6's and node 5's subtrees include node 8, their low values are 3. Since the depth value of node 1 is 1, the low value of node 3 is 1. Because node 2's and node 4's subtrees include node 3, the low values of node 2 and node 4 are 1. Since the depth value of node 1 is 1, its low value is equal to 1. Node 4 is a cut vertex in this graph because node 5's low value is 3 which is equal to the node 4's depth value.

Moreover, node 4 is the only cut vertex in this graph, because this condition cannot be satisfied by another node.

In this thesis, colors are used to define the states of the nodes. The color of node u is defined with color(u). A dominator node u is BLACK. A dominatee node without having a BLACK neighbor is WHITE. Other dominatee nodes (dominatee nodes having at least one BLACK neighbor) is GRAY. As aforementioned, a CDS is constructed when there is no WHITE node and the induced subgraph of BLACK nodes are connected. $\Gamma(u)_k$ is defined as $\Gamma(u)_k = \{t \in \Gamma(u): color(t) = k$ and is used to indicate the node u's open neighborhood with color k. Node u's closed neighborhood with color k is defined as $\Gamma[u]_k = \{t \in \Gamma(u) \cup \{u\}: color(t) = k\}$.

Greedy ratio (GR), greedy weight (GW) and greedy degree (GD) heuristics are implemented for MWCDS formation in this thesis. In these algorithms, a WHITE node is chosen at the first step and GRAY nodes are chosen at the following steps. In this manner, the algorithm resembles the Tree Growing Algorithm. The algorithms terminate when there are no WHITE nodes left. The selection policies of algorithms are different from each other. In GR, the node which has the smallest weight ratio according to the Chavatal's heuristic is selected. In Equation 1, the heuristic function of GR (f_{GR}:$V \rightarrow R$) for node u is given.

$$f_{GR}(u) = \frac{w(u)}{W_T(\Gamma[u]_{\text{WHITE}})} \qquad (1)$$

In Equation 1, the heuristic function of GR (f_{GR}:$V \rightarrow R$) for node u is given. The chosen node ($chosen_{GR}$) by GR is given in Equation 2.

$$chosen_{GR} = arg\ min_{u \in V} f_{GR}(u) \qquad (2)$$

In GD, the node u which has the maximum number of WHITE neighbors is selected. In Equation 3, the heuristic function of GD (f_{GD}:$V \rightarrow R$) for node u is given. Equation 4 gives the chosen node ($chosen_{GD}$) by GD.

$$f_{GD}(u) = |\Gamma\ (u)_{WHITE}| \tag{3}$$

$$chosen_{GD} = arg\ max_{u \in V} f_{GD}(u) \tag{4}$$

In GW, the node u which has the minimum weight among neighbors is selected. The heuristic function of GW (f_{GD}:$V \rightarrow R$) for node u is given in Equation 5. In Equation 6, the chosen node by GW is given.

$$f_{GW}(u) = w(u) \tag{5}$$

$$chosen_{GW} = arg\ min_{u \in V} f_{GW}(u) \tag{6}$$

Figure 3.3, Figure 3.4 and Figure 3.5 show example operations of GR, GD and GW respectively. Among these figures, GR has the best performance and it produces a WCDS having total weight equals to 23. GW has the worst performance and its WCDS has 42 total weight.

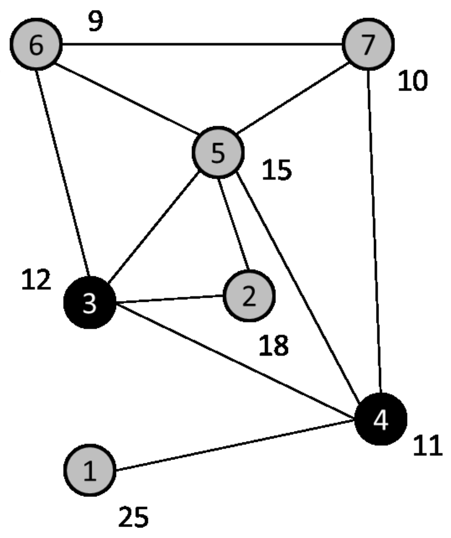


Figure 3.3. Example Operation of GR Heuristic

In Figure 3.3, GR selects node 4 having 11/73=0.151 weight ratio firstly. Node 4 is colored BLACK and its neighbors node 1, node 3, node 5 and node 7 are colored GRAY. Secondly, node 3 is selected by GR since its weight ratio is 12/(18+9)=0.144 that is smaller than the weight ratio of other GRAY nodes. The

WHITE neighbors of node 3 is node 2 and node 6, so they are colored GRAY. After this step, no more WHITE nodes left, so the GR algorithm finishes.

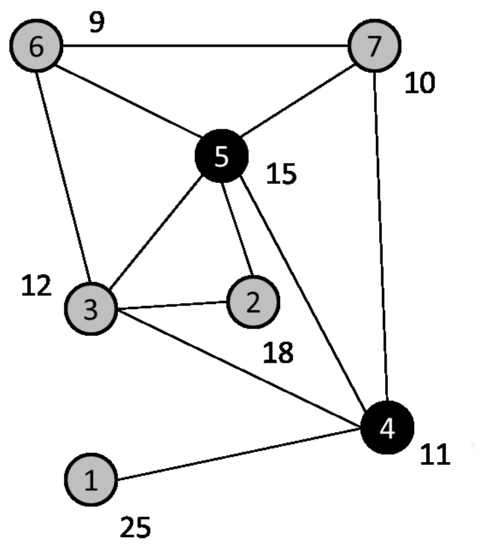


Figure 3.4. Example Operation of GD Heuristic

In GD algorithm, node 5 is chosen at the first step. The reason of this selection is that node 5 has 5 WHITE neighbors which is the maximum WHITE neighbor count, among other nodes. Node 5 is colored BLACK and its WHITE neighbors 2, 3, 4, 6 and 7 are colored GRAY. At the second step, node 4 is chosen among GRAY nodes. The WHITE degree of node 4 is 1, on the other side other GRAY nodes do not have any WHITE neighbor, so their WHITE degree is 0. Node 4 is colored BLACK and node 1 is colored GRAY. After this operation, the WHITE nodes are consumed, so the algorithm terminates.

Node 6 is selected in GW algorithm at the first step in Figure 3.5. The weight of node 6 which is the minimum among other nodes’ weights is 9. Node 6 is colored BLACK and its WHITE neighbors 3, 5 and 7 are colored GRAY. At the second step, node 7 is chosen which has the minimum weight among GRAY nodes. Node 7 is colored BLACK and node 4 is colored GRAY. At the third step, GW selects node 4 whose weight equals to 11. Node 4 is colored BLACK and node 1 that is the WHITE neighbor of node 4 is colored GRAY. At the last step, node 3 is selected by GW. Node 3 is colored BLACK and its WHITE neighbor node 2 is colored GRAY. The algorithm finishes after this selection.

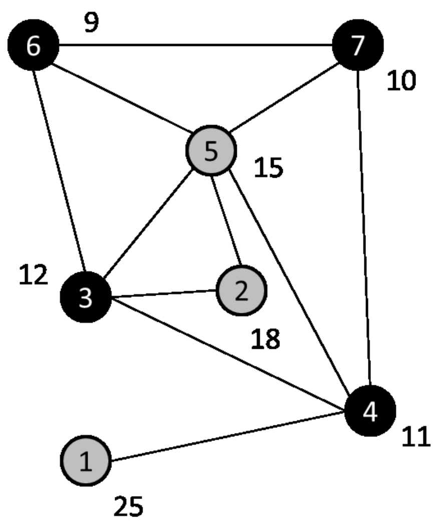


Figure 3.5. Example Operation of GW Heuristic

As aforementioned, GR has the best performance among other heuristics in Figure 3.3, Figure 3.4 and Figure 3.5. Although these figures are useful to show the operation of algorithms and to obtain a general overview about their performances, we cannot derive a general result. We present extensive simulations in the following section to show the performances of these algorithms in detail. From our measurements given in Section 4, we found that GR achieves the best results in terms of weight ratio. Moreover, the runtime of these heuristics are very similar. Because of these reasons, GR is used as the greedy heuristic in our proposed algorithms.

3.2 Description of Hybrid Genetic Algorithm

We have proposed a steady-state Hybrid Genetic Algorithm (HGA) that uses a genetic algorithm with a greedy heuristic to solve minimum weight connected dominating set problem. HGA is a genetic algorithm to improve the solution quality which is produced by greedy heuristic.

The chromosome used for graph problems is simply a set of vertices, in the mathematical sense of a set. In our HGA, the chromosomes are represented with a bit vector C of the length n where n is the number of vertices in the graph. This representation provides:

- Every chromosome is same length,
- There is no need ordering among the vertices,
- There are no duplicated vertices.

C_i represents the i^{th} bit of the chromosome. Initially all bit values of the chromosome are 0. C_i is 1 if and only if node *i* is dominator. When C_i is set to 1, the color of node *i* is set to BLACK and the colors of WHITE neighbors of node *i* are set to GRAY. When C_i is set to 0, the color of node *i* is set to BLACK if node *i* does not have any BLACK neighbor, otherwise its color is set to GRAY. Then, the colors of GRAY neighbors of node *i* are set to WHITE if and only if their only BLACK neighbor is node *i*. If any GRAY neighbor of node *i* has another BLACK neighbor other than node *i*, the color of this neighbor does not change.

Algorithm 3.2: HGA_MWCDS(max_iteration, pop_size, p_c, p_u, p_r, p_m)

```
4.  P ← GenerateInitialPopulation(pop_size, $p_r$, $p_m$)
5.  CV ← FindCutVertices()
6.  while max_iteration > 0
7.          p ← RandNum()
8.          if p < $p_c$
9.                  $p_1$ ← BinaryTournamentSelection()
10.                 $p_2$ ← BinaryTournamentSelection()
11.                 C ← FitnessBasedCrossover($p_1$, $p_2$)
12.                 C ← Mutation(C, $p_u$)
13.         else
14.                 C ← GenerateRandomChromosome()
15.         end if
16.         C ← Repair(C, $p_r$)
17.         C ← Minimize(C, CV, $p_m$)
18.         if $C \notin P$
19.                 $P \leftarrow P \cup C$
20.                 Remove($P_{pop\_size-1}$)    // remove the worst member
21.         end if
22.         max_iteration ← max_iteration-1
23. end while
24. Return w($P_0$)        // the weight of the best chromosome
```

Our proposed algorithm HGA_MWCDS is given in Algorithm 3.2 and starts with generating the initial population. The first member of the population is generated by GR heuristic. Then, the algorithm detects the cut vertices of the graph by calling the *FindCutVertices* function that uses Depth First Search (DFS) algorithm. These cut vertices are used in the minimization process. CDS should contain cut vertices as dominator. If cut vertices do not take place in DS, this DS can never provide connectivity so it cannot be a CDS. The following operations repeat for *max_iteration* times. A probability value p is generated by *RandNum* function that generates a random value in [0, 1] interval.

The algorithm continues with generating new chromosomes. Producing new chromosomes is done in two ways regarding to p_c probability. In one of these ways, two parents are chosen by binary tournament selection method and crossover of these parents is done. In step 6 and 7, the chromosomes with a better fitness value are selected for crossover. In step 8, *FitnessBasedCrossover* function is called to generate new chromosome. This function uses a fitness based crossover technique which is given in (Beasley and Chu, 1996). In this technique, the crossover operation is executed as follows. Let f_1 and f_2 are the fitness values of parents P_1 and P_2 respectively. Let C is child generated by the crossover and all P_1, P_2 and C are bit string with length n where n is the number of nodes in the graph. For all i=1 to n

(i) If $P_1[i] = P_2[i]$ then $C[i] \leftarrow P_1[i]$

(ii) Else if $P_1[i] \neq P_2$[i] then $p = f_1 / (f_1+f_2)$

$C[i] \leftarrow P_1[i]$ with probability p

$C[i] \leftarrow P_2[i]$ with probability 1-p

Then, the generated chromosome is mutated by *Mutation* function. This function mutates each node i in the graph by applying $C_i \leftarrow (C_i+1) \bmod 2$ operation regarding to p_u mutation probability.

In the other way, to ensure randomness in the population, a new chromosome is produced by generating randomly with a 1-p_c probability. Then, the new chromosome is sent to *Repair* procedure to check it if it is a CDS or not and if it does not provide a CDS, it is repaired by this procedure. After that, it becomes a CDS and sent to *Minimize* procedure to remove redundant dominators. Before inserting new member to the population, it is checked to prevent duplication. Then, the worst member is removed from the population. When all iterations are completed, the weight of the best member is returned as the final solution.

Algorithm 3.3: GenerateInitialPopulation(pop_size p_r, p_m)

```
1.  $M \leftarrow \emptyset$
2.  $P \leftarrow \emptyset$
3.  $CV \leftarrow FindCutVertices()$
4.  for i←1 to pop_ size
5.      $M \leftarrow GenerateRandomChromosome()$
6.      if $CheckCDS(M)$ is false
7.          $M \leftarrow Repair(M, p_r)$
8.      end if
9.      $M \leftarrow Minimize(M, CV, p_m)$
10.     if $M \notin P$
11.         $P \leftarrow P \cup M$
12.         $M \leftarrow \emptyset$
13.     end if
14. end for
15. Return $P$
```

GenerateInitialPopulation algorithm which is explained in Algorithm 3.3 produces members by generating chromosome data randomly. Then the produced member is checked whether it is a CDS or not. If it does not provide a solution or it is not a CDS in other words, it is repaired by applying *Repair* procedure. Thereafter, *Minimize* procedure is applied to remove redundant nodes from the solution. After these operations, if a unique solution is obtained it is added to the initial population. If a unique solution is not obtained, these steps repeat for a maximum number of trials. After a maximum number of trials, if a unique solution cannot be obtained, the size of the initial population is updated as the current size. This situation only occurs while generating the initial population for small size problem instances due to the small number of nodes.

Algorithm 3.4: Greedy_MWCDS()

```
1.  $v \leftarrow VertexWithNthLowestRatio(1, \text{WHITE})$
2.  $M \leftarrow M \cup v$
3.  color $(v) \leftarrow$ BLACK
4.  color $(\Gamma\ (v)_{WHITE}) \leftarrow$ GRAY
5.  while $\exists\ i \in V$: color($i$) = WHITE
6.      $v \leftarrow VertexWithNthLowestRatio(1, \text{GRAY})$
7.      $M \leftarrow M \cup v$
8.      color $(v) \leftarrow$ BLACK
9.      color $(\Gamma\ (v)_{WHITE}) \leftarrow$ GRAY
10. end while
11. Return $M$
```

As mentioned before, the first member of the initial population is obtained by *Greedy_MWCDS* algorithm that uses GR heuristic. The detailed steps of the algorithm are given in Algorithm 3.4. This algorithm selects the nodes regarding to their weight ratio. The weight ratio of node v is calculated as $\frac{w(v)}{w_T(\Gamma[\mathrm{v}]_{\mathrm{WHITE}})}$. Firstly, the algorithm selects the node with the minimum weight ratio by *VertexWithNthLowestRatio* algorithm which takes order n and color as parameters. It sorts all nodes having that color in ascending order regarding to their weight ratios and returns the node id in desired order that means n^{th} node id. The color of the selected node is set to BLACK and its WHITE neighbors are colored as GRAY. Then, the second and the following nodes are selected from GRAY nodes to provide connectivity between dominators. These operations continue until all nodes are GRAY or BLACK in other words there is no WHITE node.

Repair procedure that is given in Algorithm 3.5 is a heuristic to repair the chromosome with a probability p_r. A chromosome is repaired by *RepairWithRatio* that uses GR heuristic or by *RepairRandomly* that adds nodes randomly until it is a CDS.

Algorithm 3.5: Repair(C, p_r)

```
1. p ← RandNum()
2. if p < p_r
3.     Return RepairWithRatio(C)
4. else
5.     Return RepairRandomly(C)
6. end if
```

Algorithm 3.6 gives the detailed steps of *RepairWithRatio* procedure that takes a chromosome as a parameter. In the first step, the algorithm detects the BLACK nodes. If there are no WHITE nodes left in the graph, it means that all nodes are dominated. Then, the connectivity of the dominators is checked by *CheckCDS* algorithm. If the dominators are connected also, the chromosome is already a solution, so there is no need to repair it. If there is any WHITE node, the one having the minimum weight ratio among WHITE nodes is selected by *VertexWithNthLowestRatio* algorithm. *VertexWithNthLowestRatio* is designed as a generic procedure, so to get the lowest weight ratio “1” is given as a parameter. Then, the color of the selected node is set to BLACK and its WHITE neighbors are colored as GRAY. While the BLACK nodes do not construct a CDS, following nodes are selected from GRAY nodes to provide connectivity. The selected node's color is set to BLACK and its WHITE neighbors' color are set to GRAY. These steps continue until BLACK nodes provide a CDS.

Algorithm 3.6: RepairWithRatio(*C*)

```
1.  $B \leftarrow \{v: v \in V \land \text{color}(v) = \text{BLACK}\}$
2.  if $\nexists v \in V$ : color($v$) = WHITE
3.     if CheckCDS($B$) is true
4.        Return $C$
5.     end if
6.  else if $\forall v \in V$ : color($v$) = WHITE
7.     $v_{best} \leftarrow$ VertexWithNthLowestRatio(1, WHITE)
8.     $C_{v_{best}} \leftarrow 1$
9.     color ($v_{best}$) $\leftarrow$ BLACK
10.    color ($\Gamma(v_{best})_{WHITE}$) $\leftarrow$ GRAY
11.    $B \leftarrow B \cup v_{best}$
12. end if
13. while CheckCDS($B$) is false
14.    $v_{best} \leftarrow$ VertexWithNthLowestRatio(1, GRAY)
15.    $C_{v_{best}} \leftarrow 1$
16.    color ($v_{best}$) $\leftarrow$ BLACK
17.    color ($\Gamma(v_{best})_{WHITE}$) $\leftarrow$ GRAY
18.    $B \leftarrow B \cup v_{best}$
19. end while
20. Return $C$
```

Algorithm 3.7: RepairRandomly(*C*)

```
1.  $B \leftarrow \{v: v \in V \land \text{color}(v) = \text{BLACK}\}$
2.  if $\nexists v \in V$ : color($v$) = WHITE
3.     if CheckCDS($B$) is true
4.        Return $M$
5.     end if
6.  else if $\forall v \in V$ : color($v$) = WHITE
7.     $p \leftarrow \lfloor RandNum() \times (\text{graphSize} - 1) \rfloor$
8.     $C_{v_p} \leftarrow 1$
9.     color ($v_p$) $\leftarrow$ BLACK
10.    color ($\Gamma(v_p)_{WHITE}$) $\leftarrow$ GRAY
11.    $B \leftarrow B \cup v_p$
12. end if
13. while CheckCDS($B$) is false
14.    nodeList $\leftarrow$ GetNodeListByColor(GRAY)
15.    $p \leftarrow$ GetRandomNode(nodeList)
16.    $C_{v_p} \leftarrow 1$
17.    color ($v_p$) $\leftarrow$ BLACK
18.    color ($\Gamma(v_p)_{WHITE}$) $\leftarrow$ GRAY
19.    $B \leftarrow B \cup v_p$
20. end while
21. Return $C$
```

RepairRandomly algorithm is given in Algorithm 3.7. As similar to *RepairWithRatio*, if there is no WHITE node in the graph, the BLACK nodes are checked whether they construct a CDS or not. If they do not provide a solution and there exists any WHITE node, a random value p between [0, n-1] is generated where n is the graph size. The bit of chromosome at the index same with p value is set to 1. It means that, this node is selected as dominator. So, the color of the selected node is set to BLACK and its WHITE neighbors' colors are set to GRAY. While the BLACK nodes do not provide a CDS, the list of GRAY nodes is detected by *GetNodeListByColor* function. Then, a random node is selected from that list by *GetRandomNode* function. This random node's color is set to BLACK and the colors of its WHITE neighbors are set to GRAY.

Algorithm 3.8: Minimize(C, CV, p_m)

```
1.  $R \leftarrow$ FindRedundantInChromosome ($C$, $CV$)
2.  while $R$ is not empty
3.      $p \leftarrow$ RandNum()
4.      if $p < p_m$
5.          $id \leftarrow$ GetNodeWithMaxWeight($R$)
6.      else
7.          $id \leftarrow$ GetRandomNode($R$)
8.      end if
9.      color ($v_{id}$) $\leftarrow$ GRAY
10.     $R \leftarrow$ FindRedundantInChromosome($C$, $CV$)
11. end while
12. Return $C$
```

Minimize algorithm is given in Algorithm 3.8. It takes chromosome, cut vertices and minimization probability p_m as parameters. Cut vertices must be protected not to break the connectivity. At the first step of the algorithm, redundant nodes are detected by *FindRedundantInChromosome* function that is given in Algorithm 3.9. While there is any redundant node in chromosome, the following operations repeat. Among redundant nodes, a random node determined by *GetRandomNode* or node with the maximum weight ratio determined by *GetNodeWithMaxWeight* is selected regarding to p_m value. Then, the selected node is colored as GRAY. The colors of its neighbor do not change. Because it is a redundant node such that this node has no neighbor which is dominated by only that node. The list of the redundant nodes is updated after removing that node from CDS.

FindRedundantInChromosome algorithm determines redundant nodes among the BLACK nodes in chromosome. At the first step of the algorithm, the cut vertices in CDS are determined as BLACK cut vertices (*BCV*) by *FindCutVerticesInCDS* function. For all BLACK nodes, the algorithm checks three conditions to determine if it is a redundant dominator or not.

Algorithm 3.9: FindRedundantInChromosome(*C*)

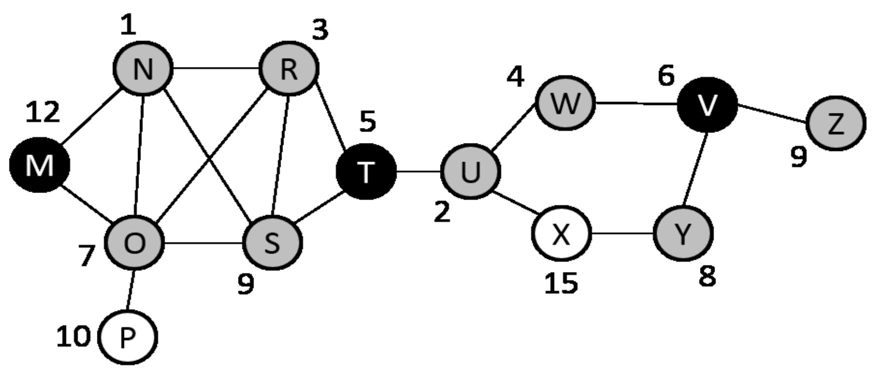
```
1. $BCV \leftarrow FindCutVerticesInCDS()$
2. $R \leftarrow \emptyset$
3. for $v \in V$: color($v$) = BLACK
4.     if $v \notin CV$ and $v \notin BCV$ and $\nexists\, i : i \in \Gamma(v) \wedge$
   color($i$)=WHITE $\wedge\ \Gamma(i)_{BLACK} = \{v\}$
5.         $R \leftarrow R \cup v$
6.     end if
7. end for
8. Return $R$
```

A dominator is redundant if:

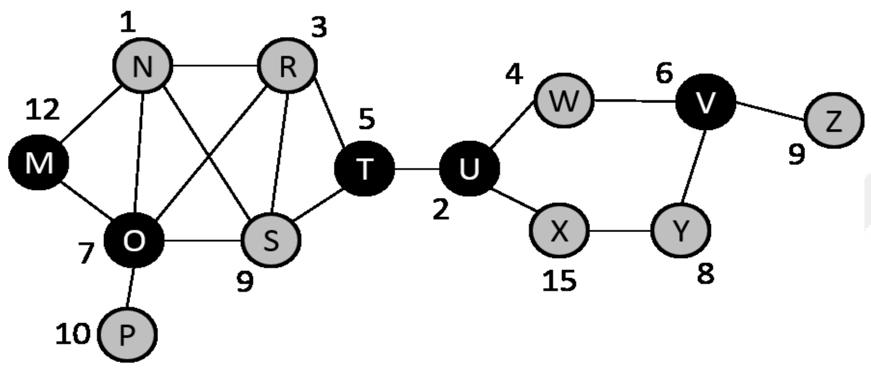
- It is not in cut vertex set ($v \notin CV$),
- It is not in black cut vertex set ($v \notin BCV$)
- It does not have any neighbor that is dominated by only that node ($\nexists\, i : i \in \Gamma(v) \wedge$ color(i)=WHITE $\wedge\ \Gamma(i)_{BLACK} = \{v\}$)

If any BLACK node satisfies these three conditions, it is a redundant dominator and the algorithm adds that node to the redundant dominator list *R*. An example operation to explain the detailed steps of *Repair* and *Minimize* procedures is given in Figure 3.6. In Figure 3.6.a, a randomly generated chromosome is shown as a graph. Nodes *M*, *T* and *V* are the initial dominators and *N*, *O*, *R*, *S*, *U*, *V*, *W*, *Y* and *Z* are the dominatees.

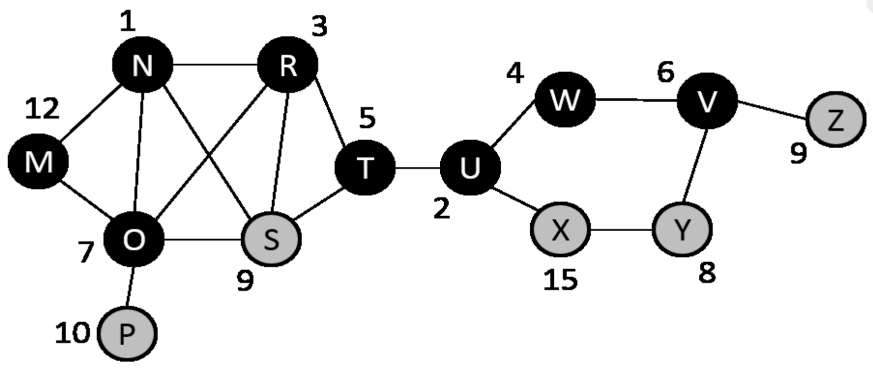
Since the colors of nodes *P* and *X* are not dominated by any dominator, this chromosome is sent to *Repair* procedure which starts by selecting a node which has any WHITE neighbor to complete covering all the nodes in the graph. Node *U* is selected between nodes *O*, *U* and *Y* whose weight ratios are 7/10, 2/15 and 8/15, respectively. Then, node O which is the only node having WHITE neighbor is selected. The colors of *O* and *U* are set to BLACK and their WHITE neighbors *P* and *X* are colored as GRAY. The updated graph is given in Figure 3.6.b. This graph shows a WDS but it is not a WCDS. Node *N* having the lowest weight from GRAY nodes is selected. Then, nodes *R* and *W* are selected. After these operations, a CDS including some redundant dominators is constructed as shown in Figure 3.6.c. *Minimize* function firstly removes node *M*, then node *N* as they are redundant dominators. Figure 3.6.d shows the final network with CDS.



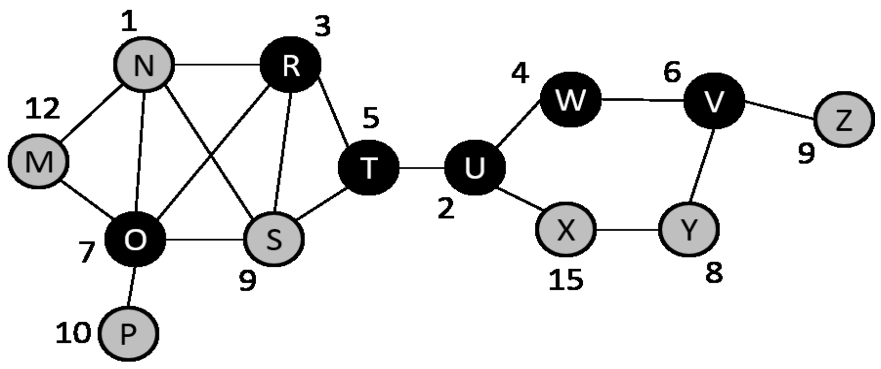
(a)



(b)



(c)



(d)

Figure 3.6. a) Initial network b) Insertion of nodes *U* and *O* c) Nodes *N*, *R* and *W* are inserted d) Redundant dominators *M* and *N* are removed.

3.3 Description of Population-based Iterated Greedy Algorithm

We proposed a Population-based Iterated Greedy (PBIG) algorithm for minimum weight connected dominating set problem. The strategy of IG algorithms is that they have a deconstruction process and a reconstruction process. They destroy the current solution and then apply a greedy heuristic to repair that solution. The steps of the proposed PBIG algorithm are given in Algorithm 3.10.

Algorithm 3.10: PBIG_MWCDS(max_iteration, pop_size, p, α)

```
1.  $P \leftarrow$ GenerateInitialPopulation_PBIG(pop_size, $\alpha$)
2.  $CV \leftarrow$ FindCutVertices()
3.  while max_iteration > 0
4.      $P_{new} \leftarrow \emptyset$
5.      for $M \in P$
6.          $M_{dest} \leftarrow$ RemoveRedundantAndDestroyPartially($M$, $p$, $CV$)
7.          $M_{new} \leftarrow$ Greedy_MWCDS_PBIG($M_{dest}$, $\alpha$)
8.          if $M_{new} \notin P$
9.              $P_{new} \leftarrow P_{new} \cup M_{new}$
10.         end if
11.     end for
12.     $P \leftarrow$ Accept($P$,$P_{new}$)
13.     max_iteration ← max_iteration-1
14. end while
15. Return w($P_0$)   // weight of first member (best member's weight)
```

The algorithm generates initial population *P* by *GenerateInitial Population_PBIG* algorithm which is given in Algorithm 3.11. Initial population consists of the candidate solutions that are generated by *Greedy_MWCDS_PBIG* algorithm that is given in Algorithm 3.12. Then, the cut vertices of the graph are determined by *FindCutVertices* function which is used in the proposed HGA also. These cut vertices are necessary for redundant nodes elimination and destruction phases of the *RemoveRedundantAndDestroyPartially* algorithm that is given in Algorithm 3.13. Until the maximum iteration counter comes to end, the following steps repeat. For each member of the population, *RemoveRedundant AndDestroyPartially* algorithm is executed. After removing redundant dominators of the member and destroying it partially, it may not be a CDS. So, this new individual is sent to *Greedy_MWCDS_PBIG* algorithm again to be repaired. If it is not generated before, it is added to the new population P_{new}. Thus, the new individuals obtained from the initial population in that way construct P_{new}. Then, the new and the initial populations are sent to *Accept* method that adds the members in the new population to the initial population and sorts the combined population in ascending order according to their weights. The new population is generated by taking the first individuals up to the initial population size at the last step of the while loop. After all iterations of the algorithm are completed, the weight of the best member is returned.

Algorithm 3.11: GenerateInitialPopulation_PBIG(pop_size, α)

```
1. $M \leftarrow \emptyset$
2. $P \leftarrow \emptyset$
3. for i←1 to pop_ size
4.     $M \leftarrow Greedy\_MWCDS\_PBIG(M, \alpha)$
5.     $P \leftarrow P \cup M$
6.     $M \leftarrow \emptyset$
7. end for
8. Return $P$
```

Algorithm 3.12: Greedy_MWCDS_PBIG(M, α)

```
1. if $\nexists v \in V$ : color($v$) = WHITE
2.         Return $M$
3. else if $\forall v \in V$ : color($v$) = WHITE
4.     $v_{best1} \leftarrow VertexWithNthLowestRatio(1, WHITE)$
5.     $v_{best2} \leftarrow VertexWithNthLowestRatio(2, WHITE)$
6.     $\alpha_0 \leftarrow RandNum()$
7.     if $\alpha_0 \leq \alpha$ then $v \leftarrow v_{best1}$
8.     else $v \leftarrow v_{best2}$
9.     end if
10.    $M \leftarrow M \cup v$
11.    color ($v$) ← BLACK
12.    color ($\Gamma(v)_{WHITE}$) ← GRAY
13. end if
14. while $\exists i \in V$: color($i$) = WHITE
15.    $v_{best1} \leftarrow VertexWithNthLowestRatio(1, GRAY)$
16.    $v_{best2} \leftarrow VertexWithNthLowestRatio(2, GRAY)$
17.    $\alpha_0 \leftarrow RandNum()$
18.    if $\alpha_0 \leq \alpha$ then $v \leftarrow v_{best1}$
19.    else $v \leftarrow v_{best2}$
20.    end if
21.    $M \leftarrow M \cup v$
22.    color ($v$) ← BLACK
23.    color ($\Gamma(v)_{WHITE}$) ← GRAY
24. end while
25. Return $M$
```

GenerateInitialPopulation_PBIG algorithm that is given in Algorithm 3.11 constructs the initial population with generating each member by *Greedy_MWCDS_PBIG* algorithm. The algorithm takes population size and determinism rate α that is used in *Greedy_MWCDS_PBIG* as parameters.

Algorithm 3.13: RemoveRedundantAndDestroyPartially(*M*, *p*, *CV*)

```
1.  $BCV \leftarrow FindCutVerticesInCDS()$
2.  $M_{new} \leftarrow \emptyset$
3.  for $v \in V$: color($v$) = BLACK
4.      $p' \leftarrow RandNum()$
        // Remove redundant nodes
5.      if $v \notin CV$ and $v \notin BCV$ and $\nexists i : i \in \Gamma(v) \wedge$ color($i$)=WHITE
    $\wedge \Gamma(i)_{BLACK} = \{v\}$
6.          color($v$) $\leftarrow$ GRAY
7.          $BCV \leftarrow FindCutVerticesInCDS()$
        // Destroy partially
8.      else if $p' \leq p$ and $v \notin CV$ and $v \notin BCV$
9.          for $j \in \Gamma(v)$
10.             if color($j$) = GRAY and $|\Gamma(j)_{BLACK}|$=1
11.                 color($j$) $\leftarrow$ WHITE
12.             end if
13.         end for
14.         color($v$) $\leftarrow$ GRAY
15.         $BCV \leftarrow FindCutVerticesInCDS()$
16.     else $M_{new} \leftarrow M_{new} \cup v$
17.     end if
18. end for
19. Return $M_{new}$
```

The detailed steps of the *Greedy_MWCDS_PBIG* algorithm are given in Algorithm 3.12. This algorithm generates each individual for the initial population and repairs partial solutions. It takes two parameters member *M* and determinism rate *α*. *M* is an empty set while generating the member of the initial population or is a partial solution while repairing.

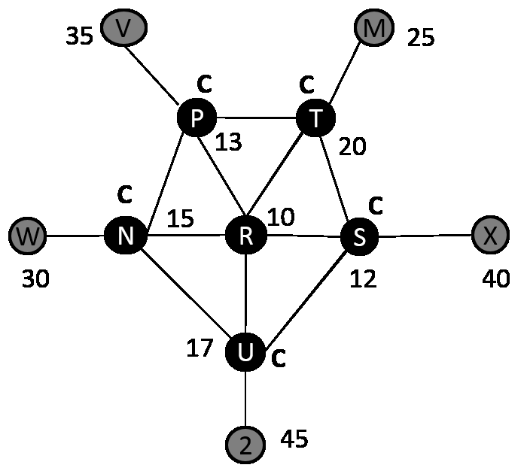
At the first step of the algorithm, WHITE node presence is checked. Because if there is no WHITE node in the graph, it means all nodes are covered and the input member is already a CDS. Thereafter, the first two nodes having the first and second lowest costs are determined by *VertexWithNthLowestRatio*. The cost of a node is calculated as the weight ratio that is given in the description of HGA algorithm. The selection from these two nodes is determined regarding to *α* determinism rate. This procedure provides to prevent selecting the same individual at each time. In other words, it prevents getting stuck in a local minimum solution that is the individual with the lowest weight ratio for that case. After selecting the node, it is added to the solution set *M* which is being generated at that time. The color of the selected node is set to BLACK and its WHITE neighbors are colored GRAY.

While there exits WHITE neighbor in the graph that means all nodes are not covered, the second and the following nodes must be selected from GRAY nodes to construct a “connected” dominating set. The first two nodes having the first and second lowest weight ratios are selected as before, but unlike the previous one they are selected from GRAY nodes. One of them is chosen according to the determinism rate and it is colored BLACK. The colors of its WHITE neighbors are set to GRAY. At the end of the while loop, the individual generated by this way is returned as a CDS solution.

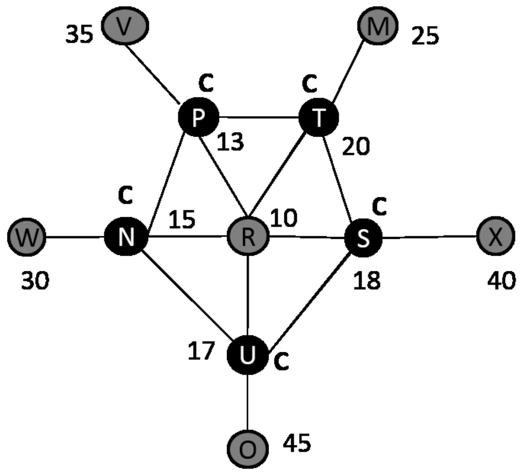
The members generated by *Greedy_MWCDS_PBIG* algorithm may include redundant dominators. The determination of these redundant nodes is provided by *RemoveRedundantAndDestroyPartially* algorithm that is given in Algorithm 3.13. As the name implies, this algorithm removes redundant nodes from the input member *M* and destroys *M* partially regarding to destruction degree *p*. The redundant nodes are determined by the same rules given in the description of the HGA.

The followings steps are executed for each BLACK node. If a dominator is not cut vertex for both the whole graph and the subgraph constituted by BLACK nodes and there does not exist any WHITE neighbor dominated by only that dominator, it is redundant. The color of the dominator is changed as GRAY. Because the BLACK nodes have changed, the BLACK cut vertices have changed also. So, the array *BCV* holding the cut vertices in BLACK nodes must be updated. In step 7, *FindCutVerticesInCDS* algorithm that detects the nodes which connect BLACK nodes is called and the returned array is assigned to *BCV*. There is no need to change the colors of the redundant node *v*’s neighbors, because there does not exist any neighbor that is dominated by only node *v*.

If any node *v* is not redundant and is not cut vertex for both graph and dominators subgraph, it can be destroyed which means it can be removed from the solution set regarding to destruction degree *p*. To remove it, the neighbors of node *v* dominated by only node *v* must be colored WHITE. These nodes’ color was GRAY because they were dominated by node *v*. So, to determine these neighbors, their color and BLACK neighbor count are checked in line 10. After neighbors are updated, node *v* is colored GRAY in line 14. The BLACK cut vertices array is updated by *FindCutVerticesInCDS* algorithm.



(a)



(b)

Figure 3.7. a) Redundant node example b) Redundant node remove (Node *R* is removed)

Remove redundant dominator example is given in Figure 3.7. In this figure, *C* indicates that this node is in *CV* set since it is a cut vertex. BLACK nodes are the dominators generated by GR heuristic and nodes *N*, *P*, *R*, *S*, *T* and *U* are cut vertices in Figure 3.7.a. In this example node *R* is a redundant dominator because it is not a cut vertex and is not a BLACK cut vertex (not in *BCV* set) that if it is removed other dominators are still connected. Also, it does not have any neighbor which is dominated by only that node. After removing node *R*, the final version of the graph is given in Figure 3.7.b.

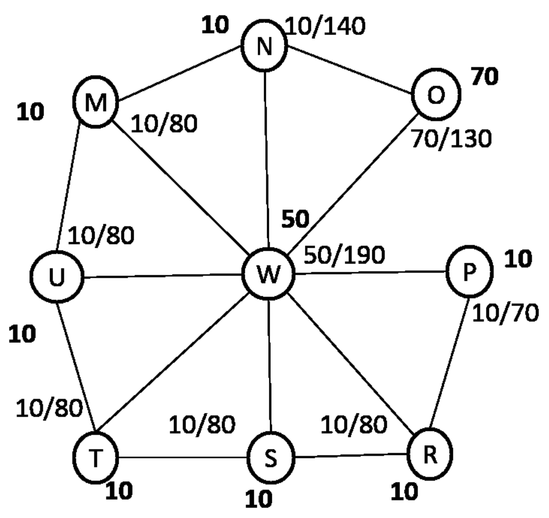


Figure 3.8. An example weighted graph

An example initial graph with weight ratio of each node is given in Figure 3.8 to clearify the node selections while producing WCDS by *Greedy_MWCDS_PBIG*. Firstly, node *N* is selected because it has the minimum weight ratio. After that, among its neighbors (*M*, *O* and *W*) the best two neighbors are taken (*M* and *W* has the same weigth ratio 10/10 and 50/50, respectively) and node *M* is selected regarding to determinism rate *α*. Then nodes *U*, *T*, *S* and *R* are selected respectively. This state of the graph which provides a WCDS is shown in Figure 3.9.a.

An example partially destroy operation is given in Figure 3.9. In this figure, *BC* indicates BLACK cut vertices that are in *BCV* set. Node *N* is selected for removing depending on destruction degree *p*. After removing node *N*, the graph is updated as Figure 3.9.b. Since the WCDS is destroyed, it is sent to *Greedy_MWCDS_PBIG* procedure to be repaired. Nodes *N* or *W* must be inserted to repair the solution. Node *W* with 50/70 weight ratio is inserted regarding to determinism rate. The constructed WCDS is given in Figure 3.9.c. After node *W* is inserted, all the other dominators *M*, *R*, *S*, *T* and *U* become redundant. So, these nodes quit from *BCV* set. They are removed from the WCDS as shown in Figure 3.9.d. After these operations, the total cost of WCDS reduces 60 to 50 and the MWCDS is constructed.

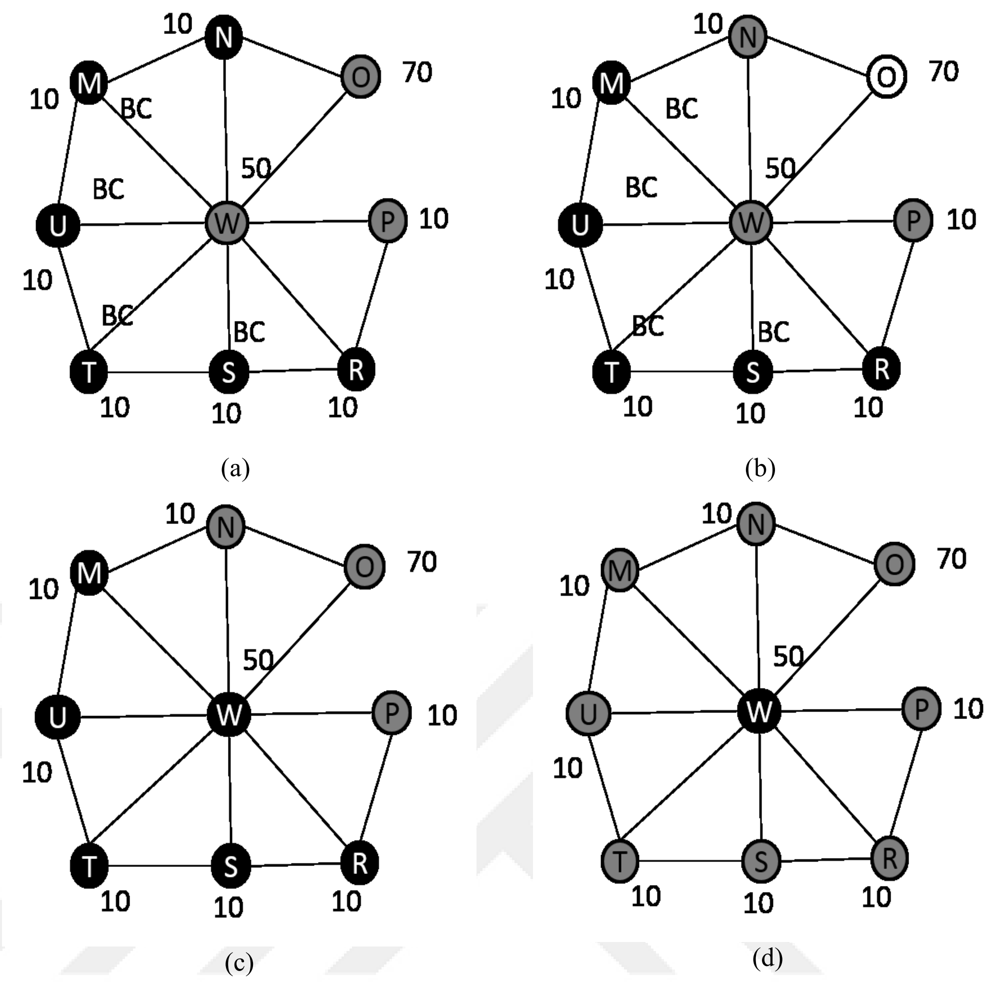


Figure 3.9. a) An example WCDS b) Node *N* is removed c) Node *W* is inserted d) Nodes *M*, *U*, *T*, *S* and *R* are removed.

3.4 Complexity Analysis of the Proposed Algorithms

In this section, the time complexity analyses of the proposed algorithms are given. Each proposed algorithm is analysed line by line.

3.4.1 Complexity Analysis of HGA

Lemma 1. *The running time complexity of Greedy_MWCDS algorithm (Algorithm 3.4) is same with Guha and Khuller's algorithm and it is equal to O(m) where m is the number of edges.*

Proof. Similar to Guha and Khuller's algorithm, our algorithm selects the vertex with minimum weight ratio in each step. To construct a CDS, it selects the second and the following nodes with the minimum ratio among the GRAY nodes.

The total time complexity of *Greedy_MWCDS* algorithm is same with Guha and Khuller's algorithm that is *O(m)*. □

Lemma 2. *The running time complexity of RepairWithRatio algorithm (Algorithm 3.6) is O(n^2+nm) where n is the number of vertices and m is the number of edges.*

Proof. Line 1 is executed in *O(n)* time. Line 2 checks if there does not exist WHITE nodes in *O(n)* time. Line 3 performs in *O(n+m)* time where *CheckCDS* is called. Line 4 performs in constant time. Line 6 controls whether all nodes are WHITE or not in *O(n)*. Line 7 is performed in *O(n)* time. Line 8 and line 9 are executed in *O(1)* time separately. Line 10 performs in *O(n)*. Line 11 is executed in constant time. The while loop between line 13 and 19 repeats for *O(n+m)*. In the while loop, line 14 and 17 execute in *O(n)*; line 15, 16 and 18 execute in *O(1)* time separately. Hence, the while loop is completed in *O(n^2+nm)* time. Line 20 perform in constant time. Consequently, the total time complexity of *RepairWithRatio* algorithm is *O(n^2+nm)*. □

Lemma 3. *The running time complexity of RepairRandomly algorithm (Algorithm 3.7) is O(n^2+nm) where n is the number of vertices and m is the number of edges.*

Proof. Line 1 and line 2 are performed in *O(n)* time separately. Line 3 controls BLACK nodes whether they construct a CDS or not by calling *CheckCDS* that is executed in *O(n+m)* time. Line 4 is executed in *O(1)* time. Line 6 is performed in *O(n)*. Line 7, 8 and 9 are performed in *O(1)* time separately. Line 10 is executed in *O(n)* time. Line 11 is performed in constant time. The while loop between line 13 and 20 repeats for *O(n+m)* times. Line 14 is executed in *O(n)* time. Line 15, 16 and 17 execute in *O(1)* time separately. Line 18 performs in *O(n)* time. Line 19 is executed in constant time. Line 21 is executed in *O(1)* time. Consequently, the total time complexity of *RepairRandomly* algorithm is *O(n^2+nm)*. □

Lemma 4. *The running time complexity of Repair algorithm (Algorithm 3.5) is O(n^2+nm) where n is the number of vertices and m is the number of edges.*

Proof. Line 1 is executed in constant time. If control in line 2 performs in *O(1)* time. Line 3 is executed in *O(n^2+nm)* from Lemma 2. Else, line 5 is executed that is in $O(n^2+nm)$ time from Lemma 3. Hence, the total time complexity of *Repair* algorithm is $O(n^2+nm)$. □

Lemma 5. *The running time complexity of FindRedundantInChromosome algorithm (Algorithm 3.9) is $O(n^3)$ where n is the number of vertices.*

Proof. Line 1 is executed in $O(n+m)$ where *FindCutVertexInCDS* algorithm is called that finds the cut vertices in CDS by DFS algorithm. Line 2 performs in $O(1)$ time. The for loop in lines 3 and 7 repeats for n times at the worst case. The if statement between line 4 and 6 checks $v \notin CV$ and $v \notin BCV$ statements which can be executed in $O(1)$ time if the CV and BCV values are stored for each vertex. The BLACK neighbor control in the same statement can be performed in $O(n)$ time for each vertex that leads to $O(n^2)$ time at the worst case. Hence, the total time complexity of the line 4 is $O(n^2)$. Line 5 is performed in constant time. Line 8 executes in $O(1)$. Thus, the total time complexity of *FindRedundant InChromosome* algorithm is $O(n^3)$. □

Lemma 6. *The running time complexity of Minimize algorithm (Algorithm 3.8) is $O(n^4)$ where n is the number of vertices.*

Proof. Line 1 is executed in $O(n^3)$ time from Lemma 5. The while loop between line 2 and 11 repeats for R times where R can be n at the worst case. Line 3 and 4 perform in $O(1)$ time separately. Line 5 selects the node with maximum weight from R set and it is executed in $O(n)$ time. Line 7 performs in $O(1)$ time. Line 9 is executed in constant time. Line 10 performs in $O(n^3)$ time from Lemma 5. Line 12 executes in constant time. Consequently, the total time complexity of *Minimize* algorithm is $O(n^4)$ time. □

Lemma 7. *The running time complexity of GenerateInitialPopulation algorithm (Algorithm 3.3) is $O(n^4s)$ where n is the number of vertices and s is the population size.*

Proof. Line 1 and 2 are performed in $O(1)$. Line 3 executes in $O(n+m)$ where *FindCutVertices* algorithm detects the cut vertices by DFS algorithm that performs in $O(n+m)$ time. The for loop between line 4 and 14 executes for s times. *GenerateRandomChromosome* algorithm generates each bit of the chromosome randomly so line 5 performs in $O(n)$. Line 6 calls *CheckCDS* which controls the connectivity between BLACK nodes in $O(n+m)$ time. Line 7 performs in $O(n^2+nm)$ from Lemma 4. Line 9 executes in $O(n^4)$ time from Lemma 6. Line 10 checks the uniqueness of the generated member in $O(ns)$ time. Line 11 and 12 perform in $O(1)$ time separately. Line 15 executes in constant time. Hence, the total time complexity of *GenerateInitialPopulation* algorithm is $O(n^4s)$. □

Theorem 1. *The running time complexity of HGA_MWCDS algorithm (Algorithm 3.2) is* $O(n^4(s+ I_{max})+ sn\, I_{max})$ *where* I_{max} *is the number of maximum iteration, s is the population size and n is the number of vertices.*

Proof. Line 1 generates the initial population in $O(n^4s)$ time complexity from Lemma 7. Line 2 performs in $O(n+m)$ where *FindCutVertices* finds the cut vertices in the graph. The while loop between lines 3 and 20 repeats $O(I_{max})$ times. Line 4 performs in $O(1)$ time. The if control in line 5 performs in $O(1)$ time. In line 6 and 7, the parents are selected by *BinaryTournamentSelection* algorithm that performs in $O(1)$. In line 8, fitness based crossover operation is performed in constant time. Line 9 performs in $O(n)$ where *Mutation* algorithm changing each bit of the chromosome regarding to mutation probability p_u is called. Lines 10-11 are executed in $O(n)$ time because *GenerateRandomChromosome* algorithm generates each bit of the chromosome randomly. Line 13 is performed in $O(n^2+nm)$ from Lemma 4. Line 14 is executed in $O(n^4)$ time from Lemma 6. Line 15 checks whether the generated individual exists or not in the population and it is performed in $O(ns)$ time complexity. Line 16 performs in $O(1)$ time. Line 17 removes the worst member in constant time. Line 19 decreases the iteration counter in $O(1)$ time. Line 21 performs in constant time. Consequently, the total time complexity of *HGA_MWCDS* algorithm yields $O(n^4(s+ I_{max})+ sn\, I_{max})$. □

3.4.2 Complexity Analysis of PBIG Algorithm

Lemma 8. *The running time complexity of Greedy_MWCDS_PBIG algorithm (Algorithm 3.12) is same with Guha and Khuller's algorithm and it is equal to O(m) where m is the number of edges.*

Proof. Guha and Khuller's algorithm selects the vertex with the minimum weight ratio in each step. Similarly, our algorithm generates a random probability and selects a vertex among the vertices having the best two weight ratios according to this probability in each step. Since, the complexities of generating a random number and calculating the weight ratio are in $O(1)$, the total complexity of *Greedy_MWCDS_PBIG* algorithm is same with Guha and Khuller's algorithm that is *O(m)*. □

<u>**Lemma 9**</u>**.** *The running time complexity of GenerateInitialPopulation_PBIG algorithm (Algorithm 3.11) is O(ms) where m is the number of edges and s is the population size.*

Proof. Line 1 and line 2 are executed in *O(1)*. In line 4 *Greedy_MWCDS_PBIG* algorithm is called so it is performed in *O(m)* time from Lemma 8. Line 5 and line 6 are executed in constant time. The for loop between the lines 3 and 7 is executed for *s* times. Hence, the total complexity of *GenerateInitialPopulation_PBIG* algorithm is *O(ms)*. □

<u>**Lemma 10**</u>**.** *The running time complexity of RemoveRedundantAnd DestroyPartially algorithm (Algorithm 3.13) is* $O(n^3)$ *where n is the number of vertices.*

Proof. Line 1 is performed in *O(n+m)*, because *FindCutVertexInCDS* finds the cut vertices in CDS by DFS algorithm which executes in *O(n+m)* time. Line 2 is executed in *O(1)* time. Line 4 generates random number between (0, 1) and is performed in *O(1)*. In line 5, $v \notin CV$ and $v \notin BCV$ controls can be executed in *O(1)* time if the *CV* and *BCV* values are stored for each vertex. In the same line, the BLACK neighbor control can be executed in *O(n)* time for each vertex which leads to $O(n^2)$ time at the worst case. Hence, the total time complexity of the line 5 is $O(n^2)$. Line 6 colors *v* vertices in constant time. Line 7 performs in *O(n+m)* like line 1 for the same reason. Line 8 is performed in *O(1)* time. The if control in lines 10-11 is executed in *O(1)*. The for loop in lines 9-13 runs *O(n)* times. Line 14 and 16 are performed in constant time. Line 15 is executed in *O(n+m)*. Lines 3 to 18 repeat for each BLACK vertices and there can be *O(n)* BLACK vertices. Hence, these lines repeat for O(*n*) times at worst case. The total time complexity yields $O(n)(O(n^2)+O(n+m))$ and *m* is in $O(n^2)$. Consequently the run time complexity of *RemoveRedundantAndDestroyPartially* algorithm is in $O(n^3)$.□

<u>**Theorem 2**</u>**.** *The running time complexity of PBIG_MWCDS algorithm (Algorithm 3.10) is* $O(I_{max}(n^3+ns)+ms)$ *where* I_{max} *is the number of maximum iteration, s is the population size, n is the number of vertices and m is the number of edges.*

Proof. Line 1 generates the initial population in *O(ms)* time complexity from Lemma 9. Line 2 executes in *O(n+m)* where *FindCutVertices* algorithm detects the cut vertices by DFS algorithm that performs in *O(n+m)* time. The while loop between lines 3 and 14 repeats $O(I_{max})$ times. Line 4 performs in *O(1)* time. The lines 5 to 11 are executed for each member in population so these lines repeat for *s* times. Line 6 is performed in $O(n^3)$ from Lemma 10. Line 7 is executed in *O(m)* time from Lemma 8. In line 8, the uniqueness control of each generated individual is performed that is in *O(ns)* time complexity. Line 9 adds new member to the population in constant time. Line 12 merges the sorted populations and takes the first *s* members in *O(s)* time. The total time complexity of *PBIG_MWCDS* is $O(I_{max}(n^3+ns)+ms)$.□

4. PERFORMANCE EVALUATIONS

We implemented the proposed algorithms with their counterparts in Java language. The PC used in our implementations has 8 Gigabytes of memory and Intel Core i7 4500U 1.80 GHz processor. As the counterparts of our algorithms, a brute force algorithm and three greedy algorithms are implemented. The implemented Brute force (BF) algorithm always finds the optimal solution by running all possible subset of solutions. Thus, 2^n possible subsets of nodes are considered.

Although BF produces solutions for small scale datasets within hours, its execution times become unacceptable long for moderate and large scale datasets. To solve the time problem in moderate and large scale datasets, a time-limited version of BF which is named as T-BF is implemented. In T-BF algorithm, the CPU execution time of the algorithm is bounded by the two times of the CPU execution time of PBIG algorithm for each (node, edge) combination. We choose PBIG instead of HGA to limit the CPU execution time of the T-BF algorithm since generally PBIG takes longer time than HGA especially for large and moderate datasets. Since T-BF is not permitted to finish its execution, it cannot always find optimum solutions. We apply t-test between PBIG and BF to compare the WCDS solution qualities of the proposed algorithms. We do not apply t-test for other algorithms because we obtain optimal solutions from HGA for small-size instances and we obtain better solutions from HGA and PBIG than T-BF for all medium and large scale datasets.

In our measurements two types of datasets are used to benchmark the proposed algorithms with their counterparts. The first dataset is proposed by (Shyu et al., 2004) and the second dataset is proposed by us by randomly generating graphs. The first dataset consists of node weighted and undirected graphs. The first dataset is divided into Type 1 and Type 2 graphs according to their distributions of node weights. For Type 1 instances, the weights of vertices are uniformly and randomly given between 20 and 120 and for Type 2 instances, the weights of the vertices are uniformly distributed between 1 and $d(v)^2$ where $d(v)$ is the degree of vertex v. The first dataset is also divided into three subgroups according to node counts in the graphs. This categorization is made by diving into small, moderate and large problem instances. In small instances, the node counts are {10, 15, 20, 25}. In moderate instances, the node counts are {50, 100, 150, 200, 250, 300}. In large instances, the node counts are {500, 800, 1000}. Besides, to measure the effect of network density, edge counts are varied. 10 different instances are generated for each node and edge count combination.

Table 4.1. Implementation parameters

Implementation Parameters		
PC Configuration	8 GB memory, Intel Core i7 4500U 1.80 GHz processor	
Language	Java	
Implemented Algorithms	HGA, PBIG, BF, T-BF, GR, GD, GW	
Datasets	Shyu's dataset (Type 1 and Type 2), Our dataset	

Properties of Type 1 Instances	**Node Counts**	Small: {10, 15, 20, 25}
		Moderate: {50, 100, 150, 200, 250, 300}
		Large: {500, 800, 1000}

	Node Weights	Randomly between [20, 120]

Properties of Type 2 Instances	**Node Counts**	Small: {10, 15, 20, 25}
		Moderate: {50, 100, 150, 200, 250, 300}
		Large: {500, 800, 1000}

	Node Weights	Randomly between [1, degree(v)^2]

Properties of Our Dataset	**Node Counts**	Small: {10, 15, 20, 25}
		Moderate: {50, 100, 150, 200, 250, 300}
		Large: {250, 500, 750, 1000}

	Node Weights	Randomly between [20, 120]
Population Size	100	
Maximum Iterations	200	
Solution Evaluations	20,000	

Even though the first dataset is a popular dataset used by researchers (Jovanovic et al., 2010; Bouamama et al., 2012; Potluri and Singh, 2013), some of the graphs generated in this dataset are unconnected. This situation brings an important problem to our experiment setups, because it is obvious that a MWCDS cannot be found on an unconnected graph. To overcome this issue, we generated a new dataset in which all graphs are connected. Our dataset is divided into small, moderate and large scale instances according to their sizes. In small size instances, there are {10, 15, 20, 25} nodes. In medium size instances, there are {50, 100, 200, 250} nodes. In large problem instances, there are {250, 500, 750, 1000} nodes. $2n$, $4n$, $6n$, $8n$ and $10n$ edges are randomly generated for each node count n. For each node and edge count combination, 10 instances are generated and the weights of the nodes are given between 20 and 120.

An initial population having 100 members is used in our algorithms. These members are executed for 200 iterations (*max_iteration*). Because of this, the proposed algorithms are executed on 20,000 generations. In other words, maximum 20,000 solutions are evaluated for each algorithm. In some cases, the initial population cannot have 100 members, because the number of all solutions are smaller than 100. In these cases, the *max_iteration* is set to 20,000 / (population size). By applying this operation, equal number of solutions is generated for reach run. The determinism rate α (used for greedy selection) is set to 0.5 and the partially destroy probability p is set to 0.5 for PBIG. The probability p_c (used for crossover) is set to 0.9, the probability p_u (used for mutation) is set to 0.005, the probability p_r (used for repair) is set to 0.7 and the probability p_m (used for minimization) is set to 0.6 for HGA. We choose these values after making experiments with various values. These parameters are summarized in Tables 4.1, 4.2 and 4.3.

Table 4.2. HGA parameters

Parameter		**Value**
p_c	Crossover probability	0.9
p_u	Mutation probability	0.005
p_r	Repair probability	0.7
p_m	Minimization probability	0.6

Table 4.3. PBIG parameters

Parameter		Value
p	Partially destroy probability	0.5
α	Determinism rate	0.5

The results of small-size problem instances are given in Tables 4.4-4.9 and Figures 4.1-4.6. The results of moderate size problem instances are given in Tables 4.10-4.15 and Figures 4.7-4.12. The results of large size problem size instances are given in Tables 4.16-4.19 and Figures 4.13-4.16. In these tables, *n* is the node count, *m* is the edge count, *# of con.* is the connected graph count, *weight* is the total weight of the MWCDS and *time* is the execution time of the algorithm.

When we investigate the figures given in Figures 4.1-4.16 we see that generally the weight and time results of the algorithms increase when we increase the node count and fix the edge count as a constant factor of the node count. The reason of this situation is the run times of the algorithms depend on the node count and the algorithms produce more dominators when node count is increased. When we fix the node count and increase the edge count, the weight results of the algorithms generally decrease since nodes can be covered by less number of dominators in dense networks. The time results of the algorithms except BF are similar when we increase the edge count and fix the node count, the time results of BF increase in this case. The time results of T-BF is omitted in Figures 4.7b, 4.8b, 4.9b, 4.10b, 4.11b, 4.12b, 4.13b, 4.14b, 4.15b and 4.16b since they are always equal to two times of PBIG's run times for moderate and large size problem instances.

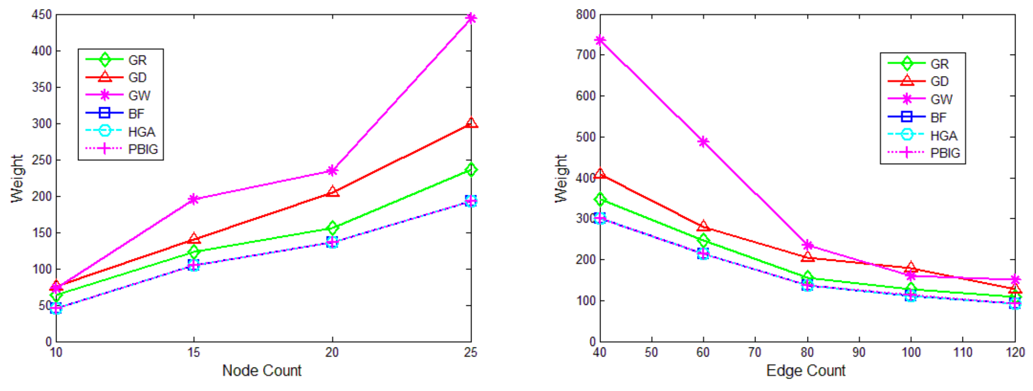
The obtained results show us that GR has the best weight performance among greedy heuristics. Because of this, we use GR as the greedy heuristic in our proposed algorithms. As expected, GR executes faster than our proposed algorithms since GR quits after finding the first solution whereas our algorithms iteratively execute to search for better candidates. On the other side, we obtain significantly better weight performances from our proposed algorithms. In the following sections we will investigate the performances of the algorithms in detail.

4.1 Evaluation of Small-Size Problem Instances

Measurements of greedy heuristics taken in small-size problem instances are given in Tables 4.4-4.9 and their related figures are given in Figures 4.1-4.6. GR has the best weight performance and GD obtains the worst weight performance when we compare the total weight of the WCDS solutions.

Table 4.4. Weight results of small size problem instances (Shyu's dataset -Type 1)

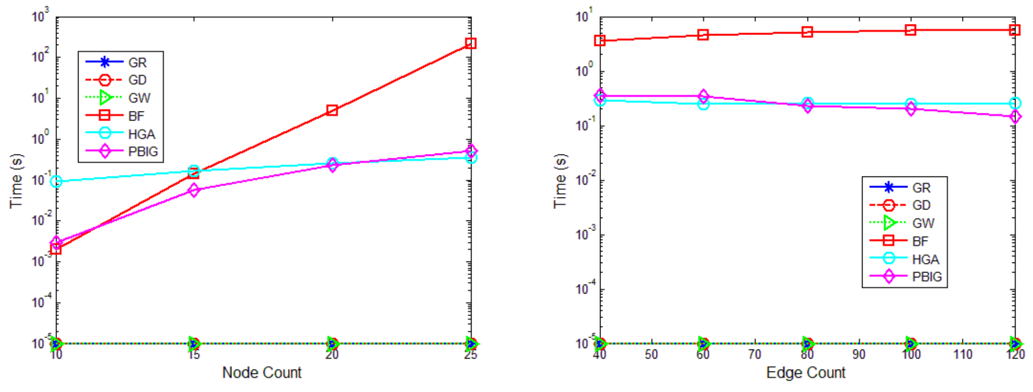
n	m	#of con.	GR	GD	GW	BF	HGA	PBIG	Rel. Error	Stat. Sign.
10	10	2	341.0	341.0	534.0	341.0	341.0	341.0	0	=
	20	10	180.0	188.8	241.7	157.2	157.2	157.2	0	=
	30	10	100.1	118.0	167.8	85.9	85.9	85.9	0	=
	40	10	63.6	75.3	73.1	45.0	45.0	45.8	1.778	≈
15	20	3	461.7	491.3	864.3	409.7	409.7	409.7	0	=
	40	10	191.7	207.4	207.4	163.0	163.0	163.0	0	=
	60	10	122.8	140.2	195.5	104.3	104.3	104.3	0	=
	80	10	74.1	119.0	91.1	67.6	67.6	67.6	0	=
	100	10	34.9	58.0	37.1	25.7	25.7	25.7	0	=
20	20	0	-	-	-	-	-	-	-	-
	40	9	347.8	407.8	735.0	300.4	300.4	300.4	0	=
	60	10	245.7	279.3	487.8	213.7	213.7	213.7	0	=
	80	10	155.7	204.3	235.5	135.8	135.8	135.8	0	=
	100	10	127.3	179.6	159.8	111.4	111.4	112.9	1.346	≈
	120	10	109.8	126.6	151.0	92.1	92.1	92.1	0	=
25	40	4	577.8	597.8	1191.5	508.3	508.3	508.3	0	=
	80	10	255.5	353.6	474.9	228.0	228.0	228.0	0	=
	100	10	235.8	300.0	443.5	193.4	193.4	193.4	0	=
	150	10	130.2	227.7	229.6	120.0	120.0	120.0	0	=
	200	10	93.7	133.4	145.6	80.3	80.3	80.3	0	=



(a) (b)

Figure 4.1 Weight results of small size problem instances (Shyu's dataset – Type 1) a) Weight results versus node count (m=4n) b) Weight results versus edge count (n=20)

We found 71.27% as the biggest weight performance difference between GR and GD and this value is obtained from (n=20, m=120) in Table 4.4. The performance difference of GR and GW reaches up to 52.3%. GR finds optimal solutions for 50 out of 52 problem samples.



(a) (b)

Figure 4.2 Time results of small size problem instances (Shyu's dataset – Type 1) a) Time results versus node count (m=4n) b) Time results versus edge count (n=20)

Table 4.5. Time results of small size problem instances (Shyu's dataset -Type 1)

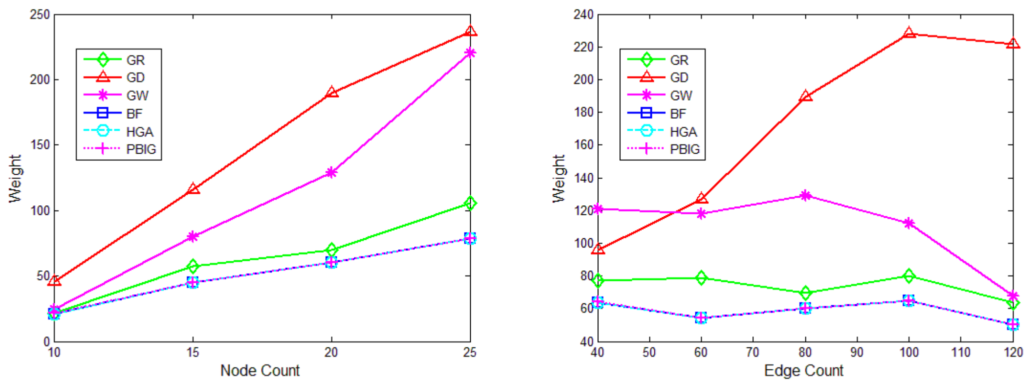
n	m	#of con.	GR*	GD*	GW*	BF	HGA	PBIG
10	10	2	0.000	0.000	0.000	0.005	0.133	0.047
	20	10	0.000	0.000	0.000	0.002	0.103	0.029
	30	10	0.000	0.000	0.000	0.003	0.097	0.008
	40	10	0.000	0.000	0.000	0.002	0.091	0.003
15	20	3	0.000	0.000	0.000	0.069	0.192	0.048
	40	10	0.000	0.000	0.000	0.107	0.161	0.192
	60	10	0.000	0.000	0.000	0.144	0.164	0.058
	80	10	0.000	0.000	0.000	0.126	0.171	0.015
	100	10	0.000	0.000	0.000	0.126	0.154	0.005
20	20	0	-	-	-	-	-	-
	40	9	0.000	0.000	0.000	3.549	0.290	0.360
	60	10	0.000	0.000	0.000	4.584	0.253	0.344
	80	10	0.000	0.000	0.000	5.081	0.259	0.228
	100	10	0.000	0.000	0.000	5.498	0.251	0.206
	120	10	0.000	0.000	0.000	5.661	0.262	0.145
25	40	4	0.000	0.000	0.000	111.652	0.457	0.420
	80	10	0.000	0.000	0.000	183.172	0.367	0.522
	100	10	0.000	0.000	0.000	210.263	0.357	0.531
	150	10	0.000	0.000	0.000	235.622	0.368	0.437
	200	10	0.000	0.000	0.000	243.002	0.397	0.096
* These values are less than 10^{-4}								

The problem instances in which GR cannot find optimum solutions are (*n*=10, *m*=10 of Shyu's Type 2 dataset) in Table 4.6 and (*n*=10, *m*=40 of Our Dataset) in Table 4.8. In the first instance, GR and GD produce 28 and 26.7 total weight, respectively. We obtain the 4.87% relative error between them. In the second instance, GR and GD produce 62.7 and 62.2 weight results where the relative error between them is 0.8%.

Table 4.6. Weight results of small size problem instances (Shyu's dataset -Type 2)

n	m	#of con.	GR	GD	GW	BF	HGA	PBIG	Relative Error	Stat. Sign.
10	10	2	28.0	26.7	32.0	26.7	26.7	26.7	0	=
	20	10	23.2	39.6	38.2	21.3	21.3	21.4	0.469	≈
	30	10	29.0	54.5	34.2	25.5	25.5	25.6	0.392	≈
	40	10	21.8	45.4	24.4	21.1	21.1	21.2	0.474	≈
15	20	3	40.2	41.0	45.5	33.8	33.8	33.8	0	=
	40	10	60.6	79.9	101.1	48.3	48.3	49.2	1.863	≈
	60	10	57.3	115.6	80.4	45.1	45.1	45.1	0	=
	80	10	39.9	128.3	44.0	37.9	37.9	37.9	0	=
	100	10	17.7	76.7	17.7	17.0	17.0	17.0	0	=
20	20	0	-	-	-	-	-	-	-	-
	40	9	77.1	95.6	120.9	63.6	63.6	64.3	1.101	≈
	60	10	78.8	126.9	118.3	54.1	54.1	54.1	0	=
	80	10	69.6	189.7	129.1	60.2	60.2	60.2	0	=
	100	10	80.0	228.3	112.5	64.8	64.8	64.8	0	=
	120	10	63.7	221.7	67.5	50.1	50.1	50.1	0	=
25	40	4	93.8	115.8	150.8	82.5	82.5	82.5	0	=
	80	10	103.2	198.0	164.5	80.0	80.0	80.0	0	=
	100	10	105.4	236.2	220.6	78.8	78.8	78.8	0	=
	150	10	97.6	287.9	204.6	84.2	84.2	84.2	0	=
	200	10	125.4	305.4	176.5	105.0	105.0	105.0	0	=

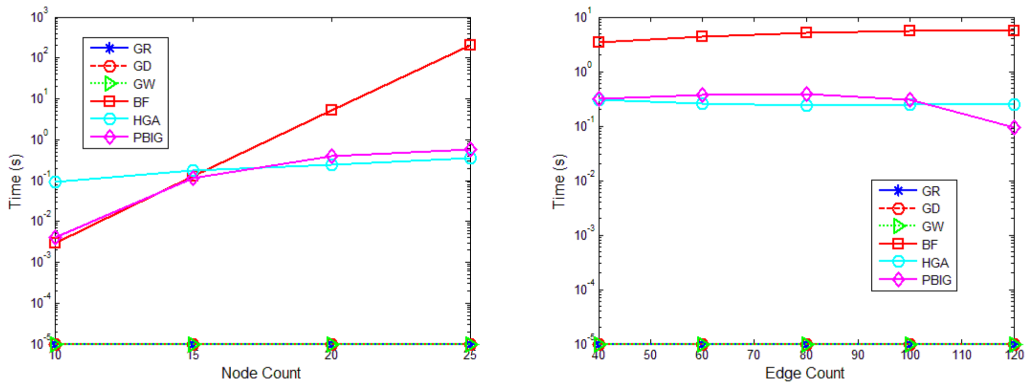
The t-test results given in Tables 4.4, 4.6 and 4.8 show that we cannot find any significant difference between the performance results of GR and GD. The CPU time of these heuristics given in Tables 4.5, 4.7 and 4.9 are close to each other and they are generally less than 0.0005 s.



(a) (b)

Figure 4.3 Weight results of small size problem instances (Shyu's dataset – Type 2) a) Weight results versus node count (m=4n) b) Weight results versus edge count (n=20)

Our proposed algorithms produce better WCDS solutions than GR except for only (n=10, m=10) problem instance in Table 4.4. In this combination, GR finds the optimum result. PBIG produces optimum results for nearly 80% of instances. HGA can find optimum results for all combinations. For 98.15% of the combinations, our algorithms perform better solutions than other greedy heuristics. The maximum difference equals to 31.3% and it is obtained in Table 4.6 for (n=20, m=60) problem instance.



(a) (b)

Figure 4.4 Time results of small size problem instances (Shyu's dataset – Type 2) a) Time results versus node count (m=4n) b) Time results versus edge count (n=20)

Table 4.7. Time results of small size problem instances (Shyu's dataset -Type 2)

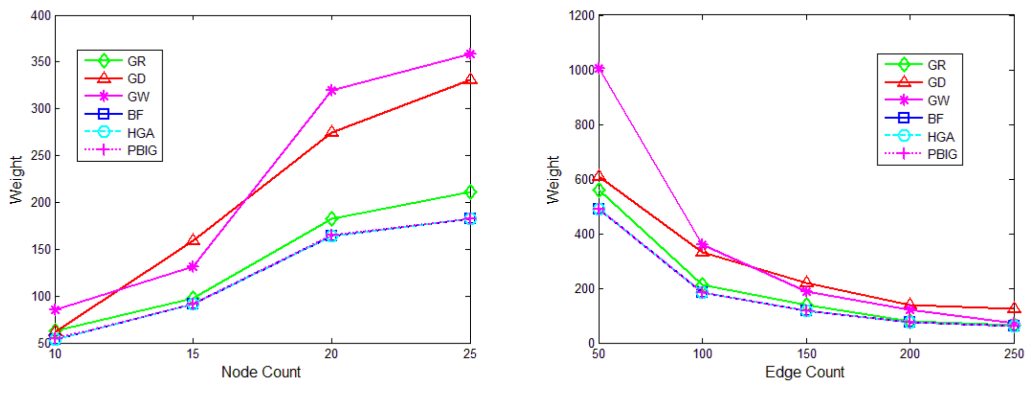
n	m	#of con.	GR*	GD*	GW*	BF	HGA	PBIG
10	10	2	0.000	0.000	0.000	0.002	0.179	0.004
	20	10	0.000	0.000	0.000	0.002	0.096	0.028
	30	10	0.000	0.000	0.000	0.003	0.095	0.013
	40	10	0.000	0.000	0.000	0.003	0.092	0.004
15	20	3	0.000	0.000	0.000	0.071	0.191	0.131
	40	10	0.000	0.000	0.000	0.109	0.162	0.265
	60	10	0.000	0.000	0.000	0.126	0.175	0.112
	80	10	0.000	0.000	0.000	0.126	0.160	0.014
	100	10	0.000	0.000	0.000	0.125	0.155	0.005
20	20	0	-	-	-	-	-	-
	40	9	0.000	0.000	0.000	3.400	0.305	0.320
	60	10	0.000	0.000	0.000	4.454	0.260	0.374
	80	10	0.000	0.000	0.000	5.182	0.240	0.391
	100	10	0.000	0.000	0.000	5.523	0.246	0.303
	120	10	0.000	0.000	0.000	5.645	0.244	0.095
25	40	4	0.000	0.000	0.000	112.309	0.483	0.400
	80	10	0.000	0.000	0.000	184.949	0.380	0.552
	100	10	0.000	0.000	0.000	209.702	0.349	0.561
	150	10	0.000	0.000	0.000	235.587	0.359	0.495
	200	10	0.000	0.000	0.000	243.215	0.396	0.162
* These values are less than 10^{-4}								

For 41 out of 52 small-size problem combinations, PBIG achieves optimum results. For the other 11 out of 52 combinations, PBIG achieves nearly optimum WCDS solutions. As an example, PBIG produces WCDS with 21.4 total weight in (*n*=10, *m*=20) problem instance where 21.3 is the optimum weight. For 11 different results, the average relative error equals to 1.73%.

Table 4.8. Weight results of small size problem instances (Our dataset)

n	m	GR	GD	GW	BF	HGA	PBIG	Relative Error	Stat. Sign.
10	20	144.4	162.1	204.9	122.7	122.7	122.7	0	=
	40	62.7	62.2	85.4	53.5	53.5	54.6	2.056	≈
15	30	273.0	322.2	416.1	250.1	250.1	250.1	0	=
	60	97.8	158.5	131.4	91.1	91.1	91.1	0	=
	90	70.5	92.6	78.3	62.5	62.5	66.6	6.56	≈
20	40	378.1	410.9	657.0	329.8	329.8	329.8	0	=
	80	182.6	274.5	319.2	164.0	164.0	164.7	0.427	≈
	120	89.9	156.5	100.3	79.5	79.5	79.5	0	=
	160	62.0	106.2	67.0	55.5	55.5	56.9	2.523	≈
25	50	559.1	609.2	1005.2	489.4	489.4	489.4	0	=
	100	211.5	330.7	358.1	182.9	182.9	182.9	0	=
	150	137.4	218.4	189.1	117.7	117.7	117.7	0	=
	200	80.3	137.0	120.0	75.2	75.2	75.2	0	=
	250	64.8	126.3	70.9	60.8	60.8	60.8	0	=

We obtain no significant difference when we apply t-test to the weight results of PBIG and the optimum weight results. Additionally, HGA produces optimum results for all small-size problem instances. These results show us that the performances of our algorithms are outstanding.



(a) (b)

Figure 4.5 Weight results of small size problem instances (Our dataset) a) Weight results versus node count (m=4n) b) Weight results versus edge count (n=25)

Table 4.9. Time results of small size problem instances (Our dataset)

n	m	GR*	GD*	GW*	BF	HGA	PBIG
10	20	0.000	0.000	0.000	0.003	0.035	0.021
	40	0.000	0.000	0.000	0.002	0.025	0.003
15	30	0.000	0.000	0.000	0.091	0.052	0.210
	60	0.000	0.000	0.000	0.124	0.048	0.060
	90	0.000	0.000	0.000	0.124	0.046	0.007
20	40	0.000	0.000	0.000	3.539	0.098	0.368
	80	0.000	0.000	0.000	5.067	0.096	0.362
	120	0.000	0.000	0.000	5.504	0.078	0.071
	160	0.000	0.000	0.000	5.588	0.073	0.014
25	50	0.000	0.000	0.000	135.025	0.149	0.543
	100	0.000	0.000	0.000	203.123	0.110	0.509
	150	0.000	0.000	0.000	226.934	0.117	0.438
	200	0.000	0.000	0.000	236.863	0.118	0.127
	250	0.000	0.000	0.000	239.601	0.110	0.030
* These values are less than 10^{-4}							

The run time results of the algorithms show that our proposed algorithms are significantly faster than BF. Even though BF generally executes faster for *n*=10 and *n*=15, the execution times of BF exponentially increase. This situation causes that the time performance difference between PBIG and BF reaches up to 7986.7 for (*n*=25, *m*=250) combination in Table 4.8 where PBIG performs optimum at the same time. For the same problem instance, the performance difference between HGA and BF is 2187.2. From these results, we can claim that our proposed algorithms are far more fast than scalable than BF.

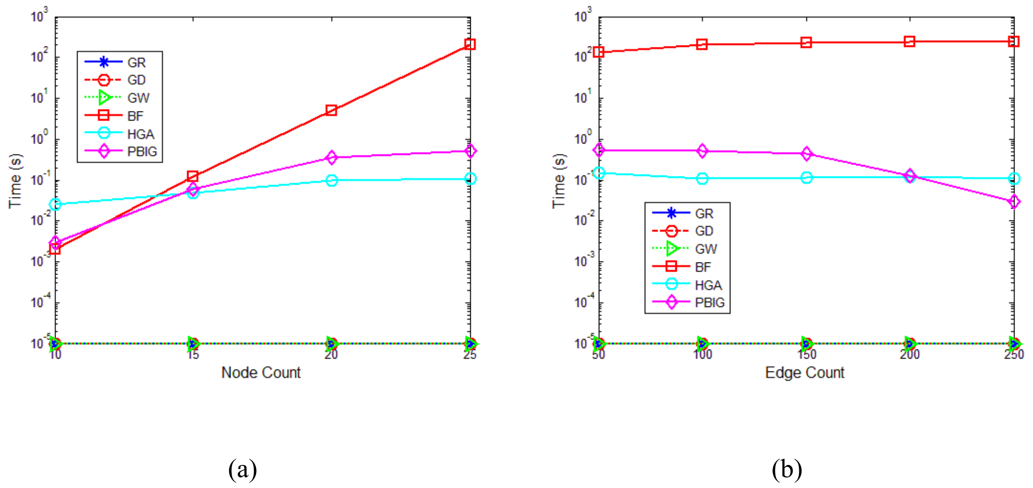


Figure 4.6 Time results of small size problem instances (Our dataset) a) Time results versus node count (m=4n) b) Time results versus edge count (n=25)

For 41 out of 52 combinations, HGA and PBIG provide the optimum results. For most of these 41 problem instances, PBIG runs faster than HGA. For other 11 out of 52 combinations, the performance of HGA is better than PBIG. For all small-size problem instances, HGA finds optimal solutions which can be seen in Tables 4.4, 4.6 and 4.8. The average execution times of HGA and PBIG are similar and they equal to 0.198 s and 0.206 s, respectively.

4.2. Evaluation of Moderate-Size Problem Instances

Measurements of greedy heuristics taken in 69 moderate-size problem instances are given in Tables 4.10-4.15 and Figures 4.7-4.12.

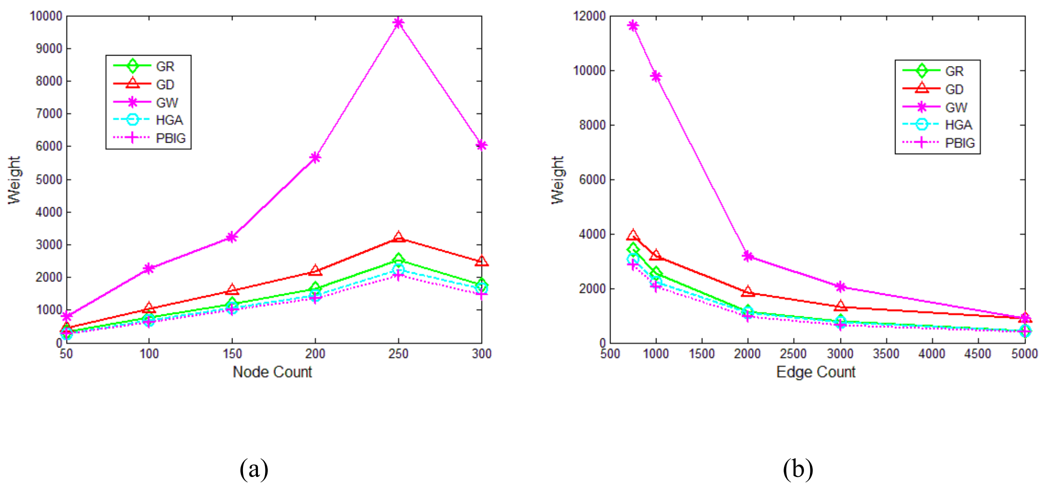


Figure 4.7 Weight results of moderate size problem instances (Shyu’s dataset – Type 1) a) Weight results versus node count (m≈5n) b) Weight results versus edge count (n=250)

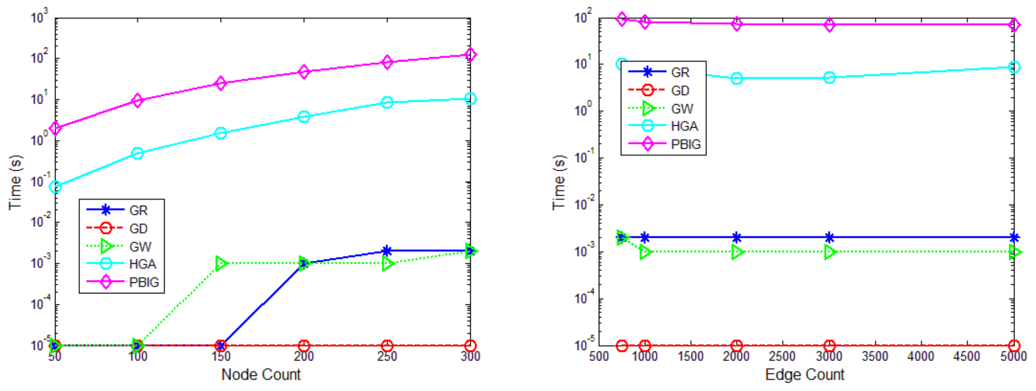
Table 4.10. Weight results of moderate size problem instances (Shyu's dataset – Type 1)

n	m	# of con.	GR	GD	GW	T-BF	HGA	PBIG
50	50	0	-	-	-	-	-	-
	100	4	1007.5	1119.5	2728.5	NF	900.0	900.0
	250	10	337.2	464.2	816.5	493.8*	291.3	290.6
	500	10	160.6	290.9	301.3	237.9	141.6	141.2
	750	10	105.4	182.3	167.2	110.9	87.7	87.6
	1000	10	59.9	138.3	80.7	73.5	51.4	51.4
100	100	0	-	-	-	-	-	-
	250	2	1649.0	1877.0	5018.0	NF	1408.0	1362.5
	500	10	768.2	1046.2	2259.1	NF	687.4	641.9
	750	10	483.8	725.2	1015.7	NF	439.8	410.4
	1000	10	373.2	591.9	710.8	661.0^{+}	336.9	308.6
	2000	10	158.9	314.3	253.9	272.9	144.4	141.9
150	150	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	6	1962.7	2387.8	7728.0	NF	1752.0	1661.3
	750	10	1179.6	1607.5	3243.4	NF	1070.2	1013.4
	1000	10	855.0	1290.7	2141.4	NF	761.4	701.6
	2000	10	416.3	708.4	806.1	NF	392.3	345.0
	3000	10	271.5	530.4	539.0	565.8	256.2	234.1
200	250	0	-	-	-	-	-	-
	500	4	3411.0	3865.8	12264.8	NF	3056.75	2966.3
	750	10	2173.2	2757.8	7384.5	NF	1917.5	1811.0
	1000	9	1647.6	2168.9	5661.2	NF	1451.4	1345.6
	2000	10	745.8	1287.5	1732.7	NF	698.0	627.1
	3000	10	497.1	891.3	1051.9	NF	478.0	417.7
250	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	7	3405.3	3921.6	11631.9	NF	3068.1	2850.3
	1000	9	2534.3	3190.4	9780.0	NF	2227.8	2056.8

n	m	# of con.	GR	GD	GW	T-BF	HGA	PBIG
250	2000	10	1151.8	1841.3	3189.1	NF	1091.9	976.3
	3000	10	780.9	1327.2	2053.3	NF	752.2	656.9
	5000	10	447.8	886.8	899.2	NF	439.7	394.9
300	300	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	2	5003.0	5819.5	17622.0	NF	4449.5	4293.5
	1000	9	3671.3	4364.4	15645.1	NF	3315.4	3111.0
	2000	10	1780.4	2464.7	6035.1	NF	1639.4	1472.6
	3000	10	1111.5	1800.0	2846.1	NF	1083.7	945.3
	5000	10	662.3	1195.8	1411.5	NF	646.0	564.2

*n=50, m=250 Brute Force found only 4 of 10 graphs, + n=100, m=1000 Brute Force found only 4 of 10 graphs

Again GR has the best performance among the implemented greedy heuristics for all moderate-size problem instances. The performance difference between GR and GD reaches up to 87.26%. We obtain this result from (*n*=50, *m*=750) combination in Shyu's Type 2 dataset which is given in Table 4.12. GW performs up to 76.8% worse than GR where this difference is obtained from (*n*=300, *m*=1000) combination. The run time results are close to each other and the maximum performance difference between them is 0.002 s.



(a) (b)

Figure 4.8 Time results of moderate size problem instances (Shyu's dataset – Type 1) a) Time results versus node count ($m \approx 5n$) b) Time results versus edge count (n=250)

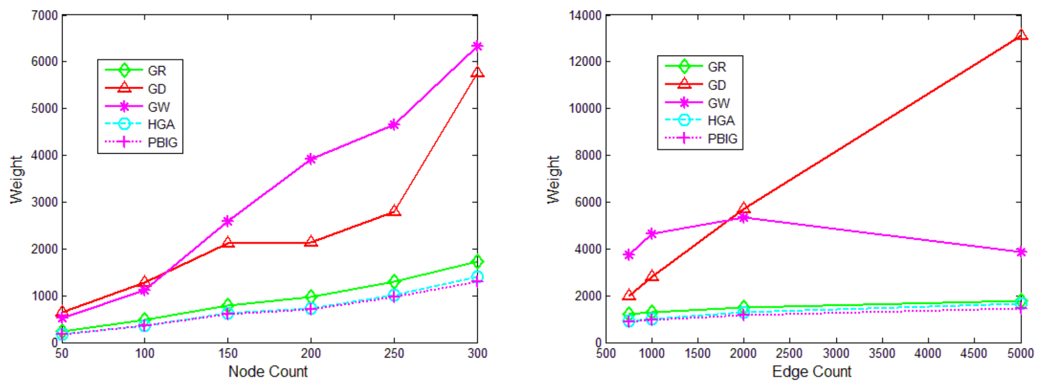
Table 4.11. Time results of moderate size problem instances (Shyu's dataset – Type 1)

n	m	# of con.	GR*	GD*	GW*	T-BF**	HGA	PBIG
50	50	0	-	-	-	-	-	-
	100	4	0.001	0.001	0.000	3.462	0.160	1.731
	250	10	0.000	0.000	0.000	3.983	0.076	1.991
	500	10	0.000	0.000	0.000	4.408	0.094	2.204
	750	10	0.000	0.000	0.000	3.545	0.113	1.772
	1000	10	0.000	0.000	0.000	0.356	0.134	0.178
100	100	0	-	-	-	-	-	-
	250	2	0.000	0.000	0.000	16151.0	0.713	8.076
	500	10	0.000	0.000	0.000	18.697	0.483	9.348
	750	10	0.000	0.000	0.000	18.437	0.412	9.218
	1000	10	0.000	0.000	0.000	19.528	0.463	9.764
	2000	10	0.000	0.000	0.000	21.664	0.942	10.832
150	150	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	6	0.000	0.001	0.000	50.864	2.202	25.432
	750	10	0.000	0.001	0.000	48.985	1.537	24.492
	1000	10	0.000	0.000	0.000	42.338	1.296	21.169
	2000	10	0.000	0.000	0.000	47.171	1.535	23.585
	3000	10	0.000	0.000	0.000	45.568	2.518	22.784
200	250	0	-	-	-	-	-	-
	500	4	0.001	0.001	0.000	97.776	5.854	48.888
	750	10	0.001	0.001	0.000	101.605	4.881	50.802
	1000	9	0.001	0.001	0.000	94.333	3.693	47.166
	2000	10	0.001	0.001	0.000	83.854	2.765	41.927
	3000	10	0.001	0.001	0.000	81.510	3.614	40.755
250	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-

n	m	# of con.	GR*	GD*	GW*	T-BF**	HGA	PBIG
250	750	7	0.002	0.002	0.000	190.136	10.401	95.068
	1000	9	0.002	0.001	0.000	164.699	8.489	82.349
	2000	10	0.002	0.001	0.000	150.110	5.072	75.055
	3000	10	0.002	0.001	0.000	143.539	5.284	71.769
	5000	10	0.002	0.001	0.000	142.146	8.735	71.073
300	300	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	2	0.003	0.003	0.001	313.279	18.848	156.640
	1000	9	0.003	0.003	0.000	312.629	16.753	156.314
	2000	10	0.002	0.002	0.000	250.434	10.330	125.217
	3000	10	0.003	0.002	0.000	222.202	8.265	111.101
	5000	10	0.003	0.002	0.000	218.030	11.360	109.015

* The 0.000 values in the related columns are less than 10^{-4}
** Time limit that is two times of PBIG's run time

Our algorithms produce significantly better weight results than GR for all 69 combinations. The performance difference between PBIG and GR reaches up to 28.2% and this result is obtained in Table 4.12 from (*n*=200, *m*=500) problem instance of Shyu's Type 2 dataset.



(a) (b)

Figure 4.9 Weight results of moderate size problem instances (Shyu's dataset – Type 2) a) Weight results versus node count ($m \approx 5n$) b) Weight results versus edge count (n=250)

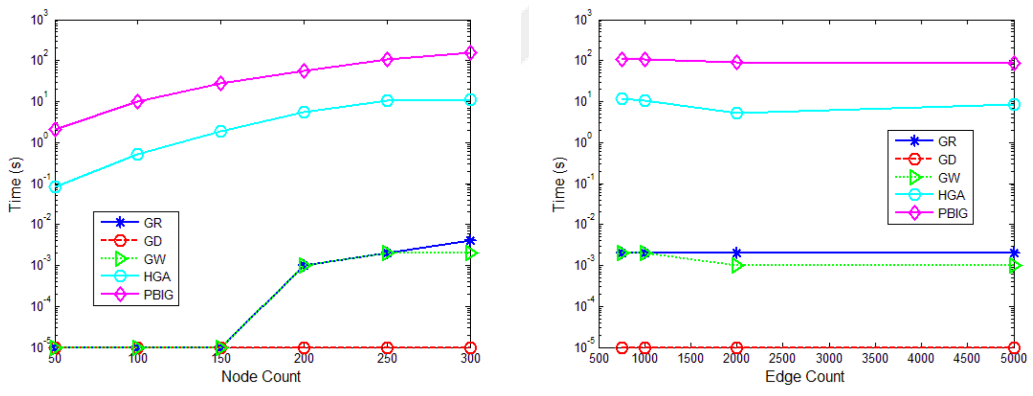
Table 4.12. Weight results of moderate size problem instances (Shyu's dataset – Type 2)

n	m	# of con.	GR	GD	GW	T-BF	HGA	PBIG
50	50	0	-	-	-	-	-	-
	100	4	213.5	280.0	384.0	NF	166.0	166.0
	250	10	238.7	642.7	516.5	418.4	182.8	182.8
	500	10	230.0	1083.7	466.4	465.2	204.4	204.4
	750	10	239.4	1879.4	369.8	612.8	215.3	215.3
100	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	7	401.7	647.7	1065.4	NF	306.1	307.4
	500	10	485.8	1286.1	1117.7	NF	364.4	348.9
	750	10	542.0	1747.1	1283.2	NF	464.5	432.2
150	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	9	629.2	1317.2	2059.8	NF	490.0	479.1
	750	10	789.4	2109.2	2599.0	NF	622.5	596.9
200	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	3	924.3	1380.7	2580.7	NF	678.7	664.0
	750	10	969.4	2131.0	3909.7	NF	733.2	709.0
250	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	6	1188.0	1987.0	3747.8	NF	924.7	896.5
	1000	9	1303.3	2797.0	4649.8	NF	1014.6	967.8
	2000	10	1492.2	5717.6	5360.7	NF	1272.9	1167.8
	5000	10	1780.4	13116.9	3860.7	NF	1666.7	1471.9
300	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	1	1251.0	1821.0	4191.0	NF	999.0	981.0

n	m	# of con.	GR	GD	GW	T-BF	HGA	PBIG
300	1000	9	1395.1	2691.9	6010.6	NF	1092.6	1058.7
	2000	10	1727.2	5768.0	6329.0	NF	1395.6	1294.7
	5000	10	2010.2	14004.9	4796.8	NF	1934.2	1622.4

Even though T-BF is given two times more time than PBIG, T-BF cannot find any solution for 53 out of 69 moderate-size combinations. On the other side, our algorithms produce WCDS for all these problem instances.

In Tables 4.10, 4.12, 4.16 and 4.18, NF stands for "Not Found". The results taken are found by averaging the results of ten instances for each combination. The weight results are the average of 6 (*n*, *m*) instances instead of 10 instances because T-BF can only find solutions for 6 problem instances.



(a) (b)

Figure 4.10 Time results of moderate size problem instances (Shyu's dataset – Type 2) a) Time results versus node count ($m \approx 5n$) b) Time results versus edge count (n=250)

PBIG produces up to 2.29 times better weight results than T-BF for (n=100, m=800) combination in Table 4.14. HGA finds up to 2.11 times better weight results than T-BF for (n=100, m=1000) combination in the same table. Our proposed algorithms perform up to 2.85 times (for (n=50, m=750) problem instance in Table 4.12) better than T-BF for the other 10 combinations. T-BF cannot produce any solution for large-size problem combinations.

Table 4.13. Time results of moderate size problem instances (Shyu's dataset – Type 2)

n	m	# of con.	GR*	GD*	GW*	T-BF**	HGA	PBIG
50	50	0	-	-	-	-	-	-
	100	4	0.000	0.000	0.000	3.441	0.181	1.720
	250	10	0.000	0.000	0.000	4.201	0.082	2.100
	500	10	0.000	0.000	0.000	5.050	0.091	2.525
	750	10	0.000	0.000	0.000	2.834	0.114	1.417
100	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	7	0.000	0.000	0.000	20.533	0.801	10.266
	500	10	0.000	0.000	0.000	20.002	0.513	10.001
	750	10	0.000	0.000	0.000	22.102	0.419	1.1051
150	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	9	0.001	0.001	0.000	52.513	2.363	26.256
	750	10	0.000	0.000	0.000	55.639	1.838	27.819
200	50	0	-	-	-	-	-	-
	100	0	-	-	-	-	-	-
	250	0	-	-	-	-	-	-
	500	3	0.002	0.001	0.001	107.096	6.675	53.548
	750	10	0.001	0.001	0.000	109.209	5.453	54.605
250	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	6	0.002	0.002	0.000	215.133	12.071	107.566
	1000	9	0.002	0.002	0.000	211.656	10.657	105.828
	2000	10	0.002	0.001	0.000	186.403	5.354	93.202
	5000	10	0.002	0.001	0.000	171.986	8.574	85.993
300	250	0	-	-	-	-	-	-
	500	0	-	-	-	-	-	-
	750	1	0.004	0.003	0.000	366.842	22.292	183.421

n	m	# of con.	GR*	GD*	GW*	T-BF**	HGA	PBIG
300	1000	9	0.003	0.002	0.000	350.722	20.359	175.361
	2000	10	0.004	0.002	0.000	316.873	11.242	158.436
	5000	10	0.003	0.002	0.000	263.854	11.472	131.927
* The 0.000 values in the related columns are less than 10^{-4} ** Time limit that is two times of PBIG's run time								

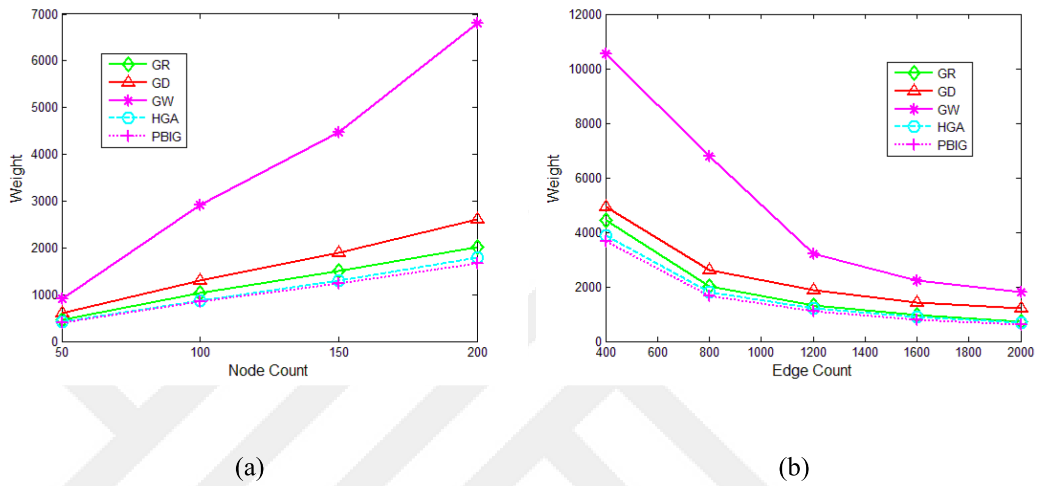


Figure 4.11 Weight results of moderate size problem instances (Our dataset) a) Weight results versus node count (m=4n) b) Weight results versus edge count (n=200)

For 5 problem instances, we obtain same results from proposed algorithms. The average weight value of HGA is 306.1 and the average weight value of PBIG is 307.4 for (n=100, m=250) problem instance in Table 4.10.

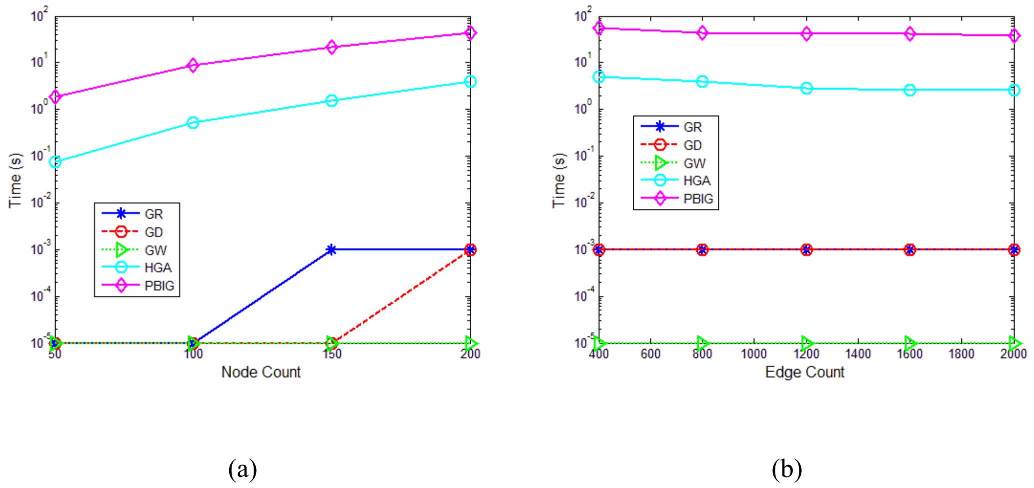


Figure 4.12 Time results of moderate size problem instances (Our dataset) a) Time results versus node count (m=4n) b) Time results versus edge count (n=200)

PBIG is better than HGA in terms of WCDS weight for the other 63 problem instances. PBIG performs up to 16% better than HGA in terms WCDS weight for (*n*=300, *m*=5000) problem instance in Table 4.12. On the other hand, PBIG is slower than HGA for all problem instances. HGA is 28 times faster than PBIG for (*n*=50, *m*=500) problem instance in Table 4.12.

Table 4.14. Weight results of moderate size problem instances (Our dataset)

n	m	GR	GD	GW	T-BF	HGA	PBIG
50	100	1068.4	1161.3	2121.3	NF	908.1	905.0
	200	459.2	594.9	904.9	658.0[α]	412.8	407.5
	300	307.6	440.5	638.2	402.3[β]	263.5	257.1
	400	219.0	340.7	356.2	289.4	186.0	183.0
	500	164.0	297.1	260.3	227.7	148.5	147.4
100	200	2149.7	2310.8	5494.5	NF	1856.5	1804.7
	400	1023.4	1305.9	2913.3	NF	873.2	845.9
	600	671.6	895.4	1846.1	NF	599.2	559.8
	800	502.2	750.3	1173.6	945.8[φ]	450.0	412.7
	1000	353.3	615.2	742.5	714.7[ψ]	338.9	314.8
150	300	3274.8	3718.6	7311.5	NF	2927.6	2817.0
	600	1499.7	1901.9	4466.5	NF	1300.7	1237.0
	900	993.3	1398.6	2435.8	NF	878.8	826.2
	1200	742.7	1048.7	1997.1	NF	694.3	623.1
	1500	543.1	902.4	1414.2	NF	508.5	460.4
200	400	4452.9	4927.9	10547.9	NF	3884.6	3683.5
	800	2021.7	2612.5	6796.9	NF	1797.2	1674.8
	1200	1323.6	1890.5	3219.5	NF	1193.5	1115.5
	1600	951.7	1413.0	2227.0	NF	891.1	784.7
	2000	723.5	1196.5	1803.6	NF	694.7	614.6

α *n*=50, *m*=200 Brute Force found only 3 of 10 graphs, **β** *n*=50, *m*=300 Brute Force found only 9 of 10 graphs, **φ** *n*=100, *m*=800 Brute Force found only 1 of 10 graphs, **ψ** *n*=100, *m*=1000 Brute Force found only 2 of 10 graphs

Table 4.15. Time results of moderate size problem instances (Our dataset)

n	m	GR*	GD*	GW*	T-BF**	HGA	PBIG
50	100	0.000	0.000	0.000	3.745	0.114	1.873
	200	0.000	0.000	0.000	3.766	0.075	1.883
	300	0.000	0.000	0.000	4.040	0.073	2.020
	400	0.000	0.000	0.000	4.288	0.079	2.144
	500	0.000	0.000	0.000	4.327	0.095	2.163
100	200	0.000	0.000	0.000	17.785	0.682	8.892
	400	0.000	0.000	0.000	17.567	0.518	8.783
	600	0.000	0.000	0.000	17.166	0.428	8.583
	800	0.000	0.000	0.000	17.181	0.401	8.590
	1000	0.000	0.000	0.000	17.728	0.439	8.864
150	300	0.001	0.000	0.000	51.873	2.166	25.937
	600	0.001	0.000	0.000	42.917	1.567	21.458
	900	0.001	0.000	0.000	43.095	1.289	21.547
	1200	0.000	0.000	0.000	43.462	1.182	21.731
	1500	0.001	0.000	0.000	42.925	1.209	21.462
200	400	0.001	0.001	0.000	109.691	4.976	54.845
	800	0.001	0.001	0.000	89.036	3.900	44.518
	1200	0.001	0.001	0.000	84.292	2.835	42.146
	1600	0.001	0.001	0.000	83.360	2.592	41.680
	2000	0.001	0.001	0.000	77.772	2.622	38.886

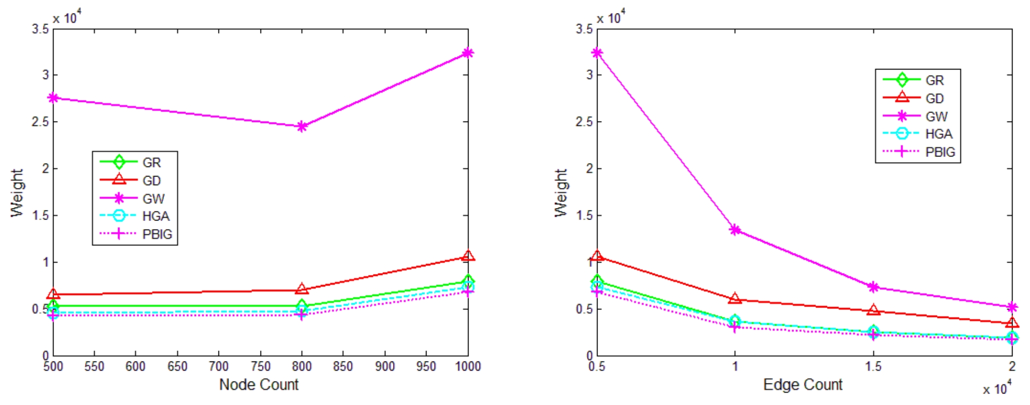
* The 0.000 values in the related columns are less than 10^{-4}
** Time limit that is two times of PBIG's run time

4.3. Evaluation of Large-Size Problem Instances

We present the measurements of greedy heuristics taken in 29 large-size problem instances in Tables 4.16-4.19 and Figures 4.13-4.16. The best weight performance is achieved by GR for all 29 instances. Its performance is up to 80.8% better than GW and this result can be seen in (*n*=500, *m*=2000) combination in Table 4.16.

Table 4.16. Weight results of large size problem instances (Shyu's dataset – Type 1)

n	m	GR	GD	GW	T-BF	HGA	PBIG
500	500	-	-	-	-	-	-
	1000	-	-	-	-	-	-
	2000	5297	6546	27581	NF	4579	4239
	5000	1803	2714	5911	NF	1803	1576
	10000	922	1925	2752	NF	922	868
800	500	-	-	-	-	-	-
	1000	-	-	-	-	-	-
	2000	-	-	-	-	-	-
	5000	5223	6976	24521	NF	4740	4334
	10000	2527	4244	6972	NF	2459	2081
1000	1000	-	-	-	-	-	-
	5000	7900	10599	32379	NF	7319	6762
	10000	3623	5946	13500	NF	3596	3013
	15000	2530	4752	7276	NF	2483	2178
	20000	1895	3432	5126	NF	1895	1658



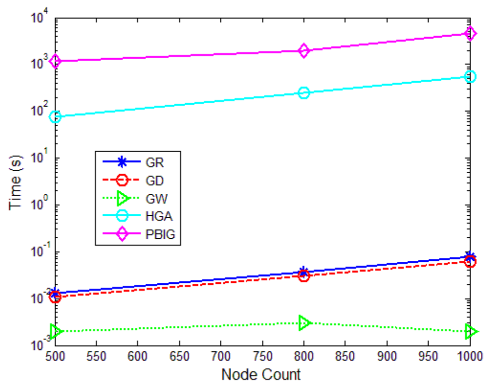
(a) (b)

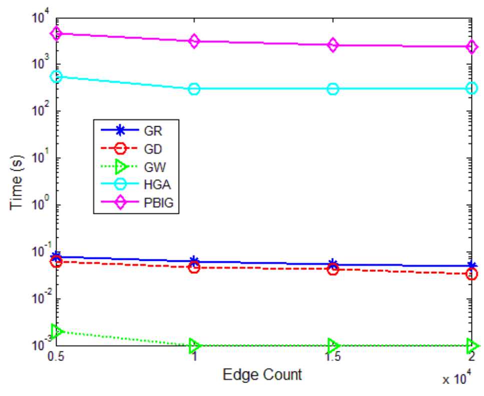
Figure 4.13 Weight results of large size problem instances (Shyu's dataset – Type 1) a) Weight results versus node count (m≈4n) b) Weight results versus edge count (n=1000)

Table 4.17. Time results of large size problem instances (Shyu's dataset – Type 1)

n	m	GR	GD	GW*	T-BF**	HGA	PBIG
500	500	-	-	-	-	-	-
	1000	-	-	-	-	-	-
	2000	0.013	0.011	0.002	583.485	74.495	1166.970
	5000	0.010	0.007	0.001	904.560	33.905	452.280
	10000	0.007	0.006	0.000	796.998	45.099	398.499
800	500	-	-	-	-	-	-
	1000	-	-	-	-	-	-
	2000	-	-	-	-	-	-
	5000	0.037	0.031	0.003	3845.316	246.808	1922.658
	10000	0.030	0.021	0.001	2931.058	131.813	1465.529
1000	1000	-	-	-	-	-	-
	5000	0.080	0.063	0.002	9070.406	540.864	4535.203
	10000	0.061	0.046	0.001	6243.744	296.014	3121.872
	15000	0.053	0.043	0.001	5312.758	295.591	2656.379
	20000	0.049	0.034	0.001	4789.570	308.906	2394.785

* The 0.000 value in the related column is less than 10^{-3}
**Time limit that is two times of PBIG's run time





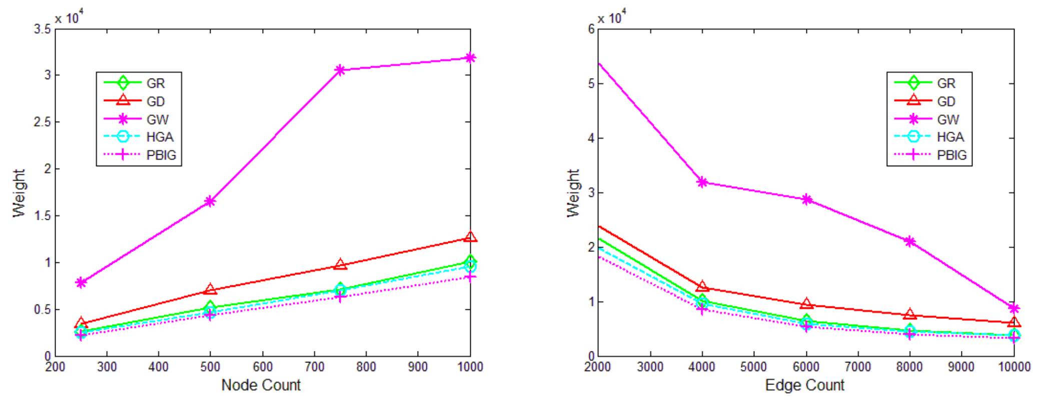
(a) (b)

Figure 4.14 Time results of large size problem instances (Shyu's dataset – Type 1) a) Time results versus node count (m≈4n) b) Time results versus edge count (n=1000)

As aforementioned GR achieves better than GD and the performance difference between these two algorithms reaches up to 52.1%. This result can be seen in (*n*=500, *m*=1000) problem sample of Shyu’s Type 1 dataset given in Table 4.16. In terms of execution times, GW has the best performance and GR has the worst performance on the average.

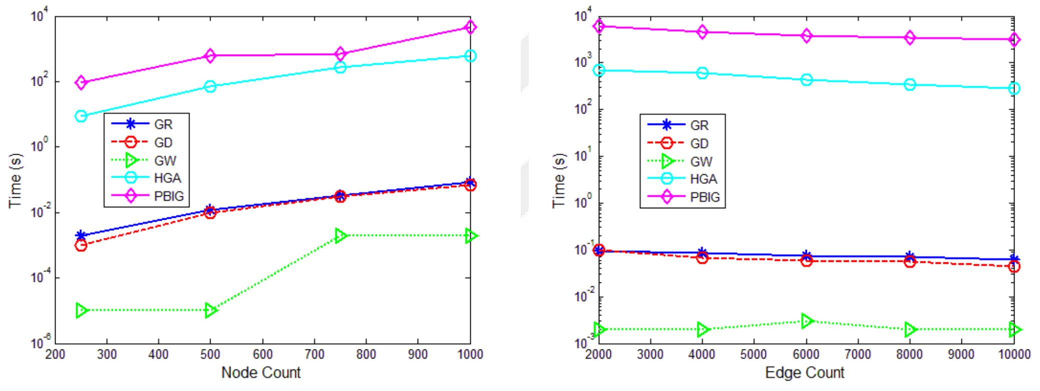
Table 4.18. Weight results of large size problem instances (Our dataset)

n	**m**	**GR**	**GD**	**GW**	**T-BF**	**HGA**	**PBIG**
250	500	5123.4	5554.2	10720.2	NF	4716.1	4585.0
	1000	2604.9	3411.2	7871.8	NF	2481.8	2228.0
	1500	1704.4	2368.3	5374.0	NF	1548.1	1384.0
	2000	1240.7	2073.8	4827.7	NF	1104.9	1062.4
	2500	1010.8	1437.4	2535.5	NF	960.3	822.2
500	1000	10229.1	11257.2	27955.9	NF	9444.3	8859.4
	2000	5164.3	7030.6	16516.7	NF	4693.7	4393.7
	3000	3446.5	5759.9	13742.6	NF	3256.9	2915.8
	4000	2552.6	3819.0	3819.1	NF	2517.9	2164.3
	5000	1977.0	3051.0	6763.3	NF	1766.5	1552.7
750	1500	16786.8	18807.5	52193.1	NF	15491.2	14298.5
	3000	7153.5	9694.3	30528.0	NF	6979.4	6250.9
	4500	4878.3	7518.4	18476.1	NF	4878.3	4383.5
	6000	3728.7	5730.6	12627.7	NF	3602.2	3226.9
	7500	2823.6	4751.1	8258.3	NF	2823.6	2435.0
1000	2000	21477.5	23785.9	53644.9	NF	19786.5	18235.3
	4000	10127.6	12624.5	31832.3	NF	9532.0	8475.9
	6000	6401.5	9340.0	28727.0	NF	5938.7	5341.9
	8000	4701.8	7498.9	21034.6	NF	4557.0	3983.5
	10000	3801.5	6024.2	8608.8	NF	3755.9	3188.9



(a) (b)

Figure 4.15 Weight results of large size problem instances (Our dataset) a) Weight results versus node count (m=4n) b) Weight results versus edge count (n=1000)



(a) (b)

Figure 4.16 Time results of large size problem instances (Our dataset) a) Time results versus node count (m=4n) b) Time results versus edge count (n=1000)

We take the maximum run time difference between GW and other heuristics from (n=1000, m=2000) problem sample in Table 4.19. In this sample GR and GD find the solution in 0.096 s, GW finds the solution in 0.1 s.

We obtain better solutions from proposed solutions in terms of WCDS weight for all 29 large-size combinations. For (n=500, m=5000) problem instance in Table 4.18, the weight performance of PBIG and GR reaches up to 21.46 %.

Table 4.19. Time results of large size problem instances (Our dataset)

n	m	GR	GD	GW*	T-BF**	HGA	PBIG
250	500	0.018	0.004	0.002	221.436	9.891	110.718
	1000	0.002	0.001	0.000	180.712	9.072	90.356
	1500	0.002	0.002	0.000	144.138	5.988	72.069
	2000	0.002	0.001	0.001	159.186	4.935	79.593
	2500	0.001	0.001	0.001	124.650	4.891	62.325
500	1000	0.013	0.013	0.001	1590.280	85.877	795.140
	2000	0.012	0.010	0.000	1242.566	68.812	621.283
	3000	0.011	0.009	0.001	1025.832	49.512	512.916
	4000	0.011	0.008	0.000	935.362	37.736	467.681
	5000	0.010	0.006	0.001	726.876	34.345	363.438
750	1500	0.043	0.043	0.002	4787.364	305.155	2393.682
	3000	0.032	0.030	0.002	3436.760	263.961	1718.380
	4500	0.030	0.025	0.001	3077.078	199.290	1538.539
	6000	0.029	0.022	0.001	3049.036	143.775	1524.518
	7500	0.027	0.020	0.001	2903.990	118.980	1451.995
1000	2000	0.096	0.100	0.002	12105.298	706.465	6052.649
	4000	0.087	0.068	0.002	9302.974	609.619	4651.487
	6000	0.073	0.058	0.003	7597.726	429.499	3798.863
	8000	0.070	0.056	0.002	6943.852	345.449	3471.926
	10000	0.063	0.044	0.002	6226.756	280.598	3113.378

* The 0.000 values in the related column are less than 10^{-3}
** Time limit that is two times of PBIG's run time

PBIG performs better than HGA for all problem instances where the performance difference reaches up to 16% for *(n*=1000, *m*=10000) combination in Table 4.16. On the other side, HGA is faster than PBIG for all problem instances. For (*n*=250, *m*=1000) in Table 4.19 problem instance, HGA is 16 times faster than PBIG.

5. CONCLUSION

We provided two population-based MWCDS optimization algorithms for undirected graphs in this thesis. Firstly, we defined MWCDS problem and its variants in detail. We showed that MWCDS has been applied to overcome backbone formation problem for WASNs where nodes with higher energies are aimed to include in backbone sets. We described WASNs and mentioned one of the most important problems in WASNs as energy conservation. We listed their various applications such as habitat monitoring, healthcare monitoring and office applications. We realized that energy-efficient backbone construction is vital to prolong the application lifetime and MWCDS is a very suitable structure to meet the energy requirements. After reviewing the literature, we found that although there are various algorithms for DS and CDS constructions, we could not find a population-based MWCDS approach which can iteratively refine the MWCDS solution quality. The motivation of this thesis arose from this fact.

The first contribution of this thesis is HGA which is a hybrid genetic algorithm and uses a greedy heuristic to solve MWCDS problem. Hybrid genetic algorithms are based on genetic and other search methods that can complement each other to achieve an optimization objective. This algorithm improves the solution quality produced by greedy heuristic. The chromosomes used in this problem are sets of vertices where each chromosome C_i is represented with a bit vector. If node *i* is a dominator, C_i is set to 1 otherwise C_i equals to 0 showing that node i is a dominatee. The first member of the HGA is generated by the GR heuristic. HGA runs a DFS based algorithm to detect cut vertices which are used in minimization process.

After above operations are achieved, HGA executes the following operations repeatedly. The algorithm generates a new chromosome in two ways. In the first way, two parents are chosen by binary tournament selection and these parents produce a new offspring by applying a crossover operation. In this crossover operation, two parents P_1 and P_2 having f_1 and f_2 fitness values are used to generate a new chromosome. For each gene, *Ci* is set to $P_1[i]$ with a probability of $f_1/(f_1+f_2)$ and C_i is set to $P_2[i]$ otherwise. After a new chromosome is generated, a mutation operation is applied on this chromosome. This operation is achieved by applying $C_i \leftarrow (C_i+1) \bmod 2$ operation according to a predefined priority. The generated chromosome is repaired to provide that it is definitely a CDS. The redundant dominators are removed from the chromosome and it is added to the population if it is not included in the population. After all iterations are finished, best member is returned as the final solution.

The second contribution of this thesis is PBIG which is based on iterated greedy strategy. This strategy aims to improve the solution quality by iteratively applying deconstruction process and a reconstruction process based on a greedy heuristic. At the first step, the algorithm generates an initial population of solutions by applying a construction process on each member. This construction process can produce a new member and can repair a partial solution. It starts with checking of WHITE node presence. If there is no WHITE node in the graph, all nodes are covered and the input is already a CDS. Thereafter, the algorithm determines first two nodes having the lowest cost and selects one of them with a predefined probability. This procedure provides to prevent getting stuck in a local minimum solution. The selected node is set to BLACK and its WHITE neighbors are colored GRAY. These operations are executed similarly while there is WHITE node in the graph. At the end of this operation, the solution is returned.

After construction process is applied, cut vertices are determined for redundant node elimination and destruction phases. This operation is followed by the below operations which are iteratively executed. The redundant dominators of each member are removed and each member is partially destroyed. The solution is repaired by applying the construction process and it is added to new population. In this manner, the newly generated members are added to the new population. Then, the new and initial populations are merged by sorting the combined population in ascending order according to the weights of solutions and taking the first individuals up to the initial population size. Lastly, when all iterations of the PBIG algorithm are completed, the best member is returned.

We analyzed the running time complexities of HGA and PBIG algorithms. We found that the running time complexity of HGA_MWCDS algorithm

(Algorithm 3.2) is $O(n^4(s+ I_{max})+ sn\, I_{max})$ where I_{max} is the number of maximum iteration, s is the population size and n is the number of vertices. We also found that the running time complexity of PBIG_MWCDS algorithm (Algorithm 3.10) is $O(I_{max}(n^3+ns)+ms)$ where m is the number of edges. To further analyze algorithms, we provide implementations of our proposed algorithms with their counterparts; greedy heuristics and brute force algorithms. GR, GW and GD are the greedy heuristics that are implemented. The implemented brute force algorithm BF always finds the best solution by running all possible subset of solutions. Since BF algorithm can execute for small scale datasets within hours, BF runs unacceptably long for moderate and large scale datasets. To overcome this problem, a time-limited version of BF named as T-BF is implemented.

We applied t-tests between our proposed algorithms to compare the solution qualities. We used two types of datasets to benchmark the algorithms. Although the first dataset is a popular dataset, some of the graphs in this dataset are unconnected. This situation causes a significant problem that MWCDS cannot be found on an unconnected graph. Because of that reason, the second dataset is proposed in this thesis by randomly generating undirected connected graphs. Our generated dataset is divided into small, moderate and large scale instances with respect to their node counts. In small, medium and large problem instances have {10, 15, 20, 25} nodes, {50, 100, 200, 250} nodes and {250, 500, 750, 1000} nodes, respectively. For each node count, $2n$, $4n$, $6n$, $8n$ and $10n$ edges are randomly generated. We generated 10 instances by randomly assigning the weights of the nodes between 20 and 120 for each node and edge count combination.

In our performance evaluation study, we investigate the measurements taken from greedy heuristics. We found that GR has the best performance in terms of WCDS weight and GD has the worst performance in small-size and moderate-size problem instances. The performance difference between GR and GD reaches 71.27% and 87.26% in small-size and medium-size problem instances, respectively. The CPU times of the greedy heuristics in small-size and moderate-size problem instances are generally close to each other. In large-size problem instances, again GR has the best performance; GW has the worst performance where GR performs up to 80.8% better than GW. On the other hand, GW runs faster than other greedy heuristics while GR performs slower than other heuristics in large-size problem instances.

We evaluated the performance results of our algorithms and GR. As expected, the running times of GR are lower than our algorithms, since GR exits after finding the first solution whereas our proposed algorithms iteratively searches for better candidates. On the other hand, our algorithms perform significantly better than GR in terms of WCDS weight. In small-size problem instances, our algorithms produce better solutions for 98.15% of the combinations. In moderate-size problem instances, the performance difference between our algorithms and GR reaches up to 28.2%. In large-size problem instances, we found better solutions for all combinations where the weight performance difference reaches up to 21.46%.

We compared the performances of our proposed algorithms with BF. Since BF executes unacceptable long, we obtained the performances of BF only in small-size problem instances. HGA produces same results with BF for all small-size problem instances. From our t-tests, we obtained no significant difference between PBIG and BF. Moreover, our algorithms outperform BF that the time performance difference between PBIG and BF reaches up to 7986.7.

We evaluated the performance values of our proposed algorithms and T-BF. T-BF cannot produce any solution for most of moderate-size problem instances whereas our algorithms produce solutions for these instances. T-BF cannot produce any solution for large-size problem instances. However, our algorithms provide solutions for all instances.

When we compared the performances of our proposed algorithms HGA and PBIG, both of them provide optimum results for 41 out of 52 small-size problem instances. For other small-size problem instances, HGA performs better than PBIG. The execution times of these algorithms are similar for small-size problem instances. In moderate-size and large-size problem instances, PBIG performs better than HGA in terms of WCDS weight for most of the problem instances where the performance difference reaches up to 16%. On the other hand, PBIG is slower than HGA for all moderate-size and large-size problem instances. These results show that PBIG performs better in terms of MWCDS weight and HGA is faster.

In future, we are planning to study on node and edge weighted versions of MWCDS problem. Edge weights can represent wireless link qualities such as received signal strength indicator or link quality indicator. Another interesting future problem is designing Steiner Tree based approaches which can provide connection between unconnected dominators to construct WCDSs having low total cost.

REFERENCES

Abdel-Aziz, S. N., Hedar, A., Sewisy, A. A., 2013, Memetic Algorithm with Filtering Scheme for the Minimum Weighted Edge Dominating Set Problem, International Journal of Advanced Research in Artificial Intelligence, Vol. 2, No. 8.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y. and Cayirci, E., 2002, A Survey on Sensor Networks. IEEE Communications Magazine, 40(8):102-114 pp.

Alkhalifah, Y. and Wainwright, R. L., 2004, A genetic algorithm applied to graph problems involving subsets of vertices, in: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 303-308.

Al-Nabhan, N., Zhang, B., Cheng, X., Al-Rodhaan, M. and Al-Dhelaan, A., 2016, Three connected dominating set algorithms for wireless sensor networks. International Journal of Sensor Networks, 21 (1) (January 2016), 53-66.

Aoun, B., Boutaba, R., Iraqi, Y. and Kenward, G., 2006, Gateway placement optimization in wireless mesh networks with QOS constraints, IEEE Journal on Selected Areas in Communications, pp.2127–2136.

Biagioni E. and Bridges, K., 2002, The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species, In Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, 16:3.

Beasley, J. E. and Chu, P.C., 1996, A genetic algorithm for the set covering problem, European Journal of Operational Research, 94 (October (2)) (1996), 392-404.

Benedettini, S., Blum, C. and Roli, A. 2010 A Randomized Iterated Greedy Algorithm for the Founder Sequence Reconstruction Problem. LION 2010: 37-51.

Bouamama, S., Blum, C. and Boukerram, A., 2012, A Population-based Iterated Greedy Algorithm for the Minimum Weight Vertex Cover Problem, Applied Soft Computing, (12) 1632-1639.

REFERENCES (continued)

Bouamama, S. and Blum, C., 2015, A Randomized Population-based Iterated Greedy Algorithm for the Minimum Weight Dominating Set Problem, 6th International Conference on Information and Communication Systems (ICICS).

Chatterjee M., Das, S. K., Turgut, D., 2002, WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks, Cluster Computing, v.5 n.2, pp.193-204.

Chen, Y. P. and Liestman, A. L., 2002, Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks, in: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '02), ACM, New York, NY, USA, 2002, pp. 165-172.

Chvatal, V., 1979, A Greedy Heuristic for the Set-Covering Problem, Mathematics of Operations Research, 4(3):233–235.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., 2009, *Introduction to Algorithms*, Third Edition (3rd ed.), The MIT Press.

Dagdeviren, Z.A., Aydin, D. and Cinsdikici, M., 2017, Two population-based optimization algorithms for minimum weight connected dominating set, Applied Soft Computing, 59:644-658.

El-Mihoub, T. A., Hopgood, A. A., Nolle, L. and Battersby, A., 2006, Hybrid Genetic Algorithms: A Review. Engineering Letters 13 (2), 124-137.

Fanjul-Peyro L. and Ruiz, R. 2010, Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research 207(1): 55-69.

Gendron, B., Lucena, A., Salles da Cunha, A. and Simonetti, L., 2014, Benders Decomposition, Branch-and-Cut, and Hybrid Algorithms for the Minimum Connected Dominating Set Problem, INFORMS Journal on Computing 26(4) (2014), 645-657.

Gonzalez, T. F., 2007, *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*, Chapman & Hall/CRC.

REFERENCES (continued)

Gray, R. M., Kieffer, J. C., and Linde, Y., 1980, Locally optimal block quantizer design, Information and Control, 45, 178.

Guha, S. and Khuller, S., 1998, Approximation Algorithms for Connected Dominating Sets, Algorithmica, (20) 374-387.

Hand, D. J., Smyth, P., and Mannila, H., 2001, Principles of Data Mining. MIT Press, Cambridge, MA, USA.

He, J., Cai, Z., Ji, S., Beyah, R. and Pan, Y., 2011, A genetic algorithm for constructing a reliable MCDS in probabilistic wireless networks, in: Proceedings of the 6th International Conference on Wireless Algorithms, Systems, and Applications (WASA 2011), LNCS 6843, pp. 96–107.

He, J., Ji, S., Yan, M., Pan, Y. and Li, Y., 2012, Load-balanced CDS construction in wireless sensor networks via genetic algorithm, International Journal of Sensor Networks, 11 (3) (April 2012), 166-178.

He, J. S., Ji, S., Beyah, R., Xie, Y. and Li, Y., 2015, Constructing load-balanced virtual backbones in probabilistic wireless sensor networks via multi-objective genetic algorithm, Transactions on Emerging Telecommunications Technologies, 26 (2) (February 2015), 147-163.

Hoos H. H. and Stützle, T., 2005, Stochastic Local Search: Foundations and Applications, Morgan Kaufmann Publishers.

Hopcroft, J. and Tarjan, R., 1974, Efficient planarity testing, Journal of the ACM (JACM), 21(4), 549-568.

Houmaidi, M. E. and Bassiouni, M. A., 2003, K-weighted minimum dominating sets for sparse wavelength converters placement under non-uniform traffic, in: Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03), pp. 56–61.

Jin, Y., 2005, A comprehensive survey of fitness approximation in evolutionary computation, Soft Computing, vol. 9, pp. 3-12.

REFERENCES (continued)

Jovanovic, R., Tuba, M. and Simian, D., 2010, Ant Colony Optimization Applied to Minimum Weight Dominating Set Problem, in: Proceedings of the 12th International conference on Automatic control, modelling and simulation. pp. 322–326. ACMOS'10, World Scientific and Engineering Academy and Society, Stevens Point, Wisconsin, USA.

Jovanovic, R. and Tuba, M., 2013, Ant Colony Optimization Algorithm with Pheromone Correction Strategy for the Minimum Connected Dominating Set Problem, Computer Science and Information Systems (ComSIS), 10 (1).

Kamali, S. and Safarnourollah, V., 2006, A Genetic Algorithm for Power Aware Minimum Connected Dominating Set Problem in Wireless Ad-Hoc Networks, Concordia University.

Karl, H. and Willig, A., 2005, Protocols and Architectures for Wireless Sensor Networks, Wiley 2005, ISBN 978-0-470-09510-2, pp. I-XXV, 1-497.

Khalil, E. and Ozdemir, S., 2015a, Prolonging stability period of CDS based wireless sensor networks, in: Proceedings of the 11th International Wireless Communications and Mobile Computing Conference (IWCMC), Dubrovnik, Croatia.

Khalil, E. and Ozdemir, S., 2015b, CDS based reliable topology control in wireless sensor networks, in: Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC), Hammamet, Tunisia.

Khun, F., Wattenhofer, R., and Zollinger, A., 2003, Ad-hoc networks beyond unit disk graphs, in: Proceedings of the 2003 joint workshop on Foundations of mobile computing(DIALM-POMC'03), ACM, New York, USA, pp.69-78.

Klein, P., and Ravi, P., 1995, A nearly best-possible approximation algorithm for node-weighted Steiner trees, Journal of Algorithms*,* 19(1):104-115pp.

Konak A. and Smith, A. E., 1999 "A hybrid genetic algorithm approach for backbone design of communication networks," in the 1999 Congress on Evolutionary Computation. Washington D.C, USA: IEEE, pp. 1817-1823.

REFERENCES (continued)

Linde, Y., Buzo, A., and Gray, R. M., 1980, An algorithm for vector quantizer design, IEEE Transactions on Communications., 28(1), 84.

Lin, G., 2016, A hybrid self-adaptive evolutionary algorithm for the minimum weight dominating set problem, International Journal of Wireless and Mobile Computing, 11 (1), (January 2016), 54-61.

Liu, X., Wang, W., Kim, D., Yang, Z., Tokuta, A. O. and Jiang, Y., 2016, The first constant factor approximation for minimum partial connected dominating set problem in growth-bounded graphs, Wireless Networks, 22 (2), 553-562.

Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D. and Anderson, J., 2002, Wireless Sensor Networks for Habitat Monitoring. In Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA.

McCall, J., 2005, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics, Volume 184, Issue 1, Pages 205-222, ISSN 0377 0427, http://dx.doi.org/10.1016/j.cam.2004.07.034.

More, V. and Mangalwede, S.R., 2013, A Genetic Algorithm for Solving Connected Dominating Set Problem in Wireless Ad-Hoc Network, International Journal of Computer and Communication Engineering Research (IJCCER), 1 (2).

Ning X., 2005, A Survey of Sensor Network Applications, IEEE Communications Magazine, 5(5): 774-788pp.

Nitash, C. G and Singh, A., 2014, An artificial bee colony algorithm for minimum weight dominating set, in: Proceedings of the IEEE Symposium on Swarm Intelligence (SIS), pp. 1-7.

Pelikan, M., Goldberg, D. E. and Lobo, F., 1999, A survey of optimization by building and using probabilistic models, IlliGAL.

Potluri A. and Singh, A., 2013, Hybrid metaheuristic algorithms for minimum weight dominating set, Applied Soft Computing, 13 (1) 76-88.

REFERENCES (continued)

Ramalakshmi, R. and Radhakrishnan, S., 2015, Weighted dominating set based routing for ad hoc communications in emergency and rescue scenarios, Wireless Networks, 21 (2), 499-512.

Ruiz, R. and Stützle T., 2007, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3): 2033-2049.

Samet, H., 1990, Applications of Spatial Data Structures, Addison-Wesley, Reading, MA.

Schwiebert, L., Gupta, S. K. S. and Weinmann, J., 2001**,** Research Challenges in Wireless Networks of Biomedical Sensors. In Proceedings of Mobile Computing and Networking, 151-165pp.

Shen, C. and Li, T., 2010, Multi-document summarization via the minimum dominating set, in: Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), August 2010, pp. 984–992.

Shyu, S. J., Yin, P., Lin and B. M. T., 2004, An ant colony optimization algorithm for the minimum weight vertex cover problem, Annals of Operation Research, (131), 283-304.

Singh, A. and Gupta, A. K., 2006, A hybrid heuristic for the minimum weight vertex cover problem, Asia-Pacific Journal of Operational Research, 23 (June (2)), 73-285.

Srivastava, M. B., Muntz, R. R. and Potkonjak, M., 2001**,** Smart Kindergarten: Sensorbased Wireless Networks for Smart Developmental Problem-solving Enviroments, In Proceedings of Mobile Computing and Networking, 132-138pp.

Subhadrabandhu, D., Sarkar, S. and Anjum, F., 2004, Efficacy of misuse detection in adhoc networks, in: Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004, pp. 97–107.

Torkestani, J. A. and Meybodi, M. R., 2010a, An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata, Computer Networks, 54 (5), 826-843.

REFERENCES (continued)

Torkestani, J. A. and Meybodi, M. R., 2010b, Clustering the wireless ad hoc networks: A distributed learning automata approach, Journal of Parallel and Distributed Computing, 70 (4), 394-405.

Torkestani, J. A. and Meybodi, M. R., 2012, Finding minimum weight connected dominating set in stochastic graph based on learning automata, Information Sciences, 200, 57-77.

Wang, Y., Wang, W. Z. and Li, X. -Y., 2005, Distributed low-cost backbone formation for wireless ad hoc networks, in: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '05), ACM, New York, NY, USA, pp. 2-13.

Wu, J. and Li, H., 1999, On calculating connected dominating set for efficient routing in ad hoc wireless networks, in: *Proceedings of the 3rd international workshop on discrete algorithms and methods for mobile computing and comm.* (DIALM '99), ACM, New York, USA, pp. 7-14.

Wu, P., Wen, J. -R., Liu, H. and Ma, W. -Y., 2006, Query selection techniques for efficient crawling of structured web sources, in: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), pp. 47.

Xiang, Z. and Joy, G., 1994, Color image quantization by agglomerative clustering, IEEE Computer Graphics and Applications, 14(3), 44.

Yu, J., Wang, N., Wang, G. and Yu, D., 2013, Connected dominating sets in wireless ad hoc and sensor networks - A comprehensive survey, Computer Communications, 36 (2), 121-134.

Yu, J., Li, W., Cheng, X., Atiquzzaman, M., Wang, H. and Feng, L., 2016, Connected dominating set construction in cognitive radio networks, Personal and Ubiquitous Computing, 20 (5), 757-769.

Zhu, X., Wang, W., Shan, S., Wang, Z. and Wu, W., 2012, A PTAS for the minimum weighted dominating set problem with smooth weights on unit disk graphs, Journal of Combinatorial Opt., 23 (4) (May 2012), 443-450.

Zou, F., Wang, Y., Xu, X., Li, X., Du, H., Wan, P. and Wu, W., 2011, New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs, Theoretical Computer Science 412(3), 198-208.

CURRICULUM VITAE

Züleyha AKUSTA DAĞDEVİREN

Address: International Computer Institute, Izmir/TURKEY.
Telefon: 05436914983
E-mail: zuleyhaakusta@gmail.com

Personal Information
Nationality: Turkish
Birth Date: 20.02.1987
Birth Place: Izmir

Education
September 2012- present, Ph.D. in Information Technologies,
International Computer Institute, Ege University
September 2010 – June 2012, M.Sc. in Information Technologies,
International Computer Institute, Ege University
September 2005 – June 2010, B.Sc. in Computer Engineering,
Computer Engineering Department, Izmir Institute of Technology

Foreign Languages
Turkish: First Language
English: Advanced

Programming Languages
C/C++, Java, C#, Matlab, MIPS, Assembly.

Publications

Dagdeviren, Z.A., Aydin, D. and Cinsdikici, M., 2017, Two population-based optimization algorithms for minimum weight connected dominating set, Applied Soft Computing, 59:644-658.

Dagdeviren, Z.A., Oguz, K. and Cinsdikici, M., 2015, Automatic Registration of Structural Brain MR Images to MNI Image Space, In proceedings of the

23th Signal Processing and Communications Applications Conference (SIU), Malatya, Turkey, pp. 359–362.

Dagdeviren, Z.A., Oguz, K. and Cinsdikici, M.G., 2014, Three Techniques for Automatic Extraction of Corpus Callosum in Structural Midsagittal Brain MR Images: Valley Matching, Evolutionary Corpus Callosum Detection and Hybrid Method, Engineering Applications of Artificial Intelligence, 31:101–115.

Akusta, Z. and Kardas, G., 2011, A Case Study on the Development of Electronic Barter Systems using Software Agents, In proceedings of the 5th Turkish National Software Engineering Symposium (UYMS 2011), September 26-28, Ankara, Turkey, pp. 123-126.

Thesis

Dagdeviren, Z. A., Registration of the Structural MR Images of the Patients to MNI Image Space, M.Sc. Thesis, Ege University, Izmir, Turkey, 2012 (in Turkish).

Akusta, Z., Development of a B2B E-Commerce Software by Using Database Technology, B.Sc. Thesis, Izmir Institute of Technology, Izmir, Turkey, 2010 (in English).

Other Academic Activities

TUBITAK 2211-C Schoolarship

Reviewers of Journals: Applied Soft Computing

IET Image Processing