A Study of Algorithms for Minimum Dominating Set and its Generalizations

A thesis submitted during 2012 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in Department of Computer and Information Sciences

by

Anupama Potluri

Department of Computer and Information Sciences

School of Mathematics and Computer/Information Sciences

University of Hyderabad
P.O. Central University, Gachibowli
Hyderabad – 500046
Andhra Pradesh, India

September 2012

CERTIFICATE

This is to certify that the thesis entitled **"A Study of Algorithms for Minimum Dominating Set and its Generalizations"** submitted by **Anupama Potluri** bearing **Reg. No. 07MCPC23** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by her under my supervision and guidance.

The thesis has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

(Dr. Atul Negi)
Advisor
Department of Computer and Information Sciences
School of Mathematics and Computer/Information Sciences
University of Hyderabad
Hyderabad – 500046, India

Head
Department of Computer &
Information Sciences
University of Hyderabad
Hyderabad – 500046, India

Dean
School of Mathematics &
Computer/Information Sciences
University of Hyderabad
Hyderabad – 500046, India

DECLARATION

I, **Anupama Potluri**, hereby declare that this thesis entitled **"A Study of Algorithms for Minimum Dominating Set and its Generalizations"** submitted by me under the guidance and supervision of **Dr. Atul Negi** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Date : **Name: Anupama Potluri**

Signature of the Student:

Reg. No.: 07MCPC23

Abstract

Algorithms for dominating sets of minimum cardinality have recently attracted intense research attention due to their use in many diverse fields such as wireless networks, information retrieval, query selection in web databases etc.. It is well known that computing dominating sets of minimum cardinality is $\mathcal{NP}$-hard. Many exact and approximation (centralized and distributed) algorithms and Polynomial-Time Approximation Schemes (PTAS) have been proposed recently for solving the dominating set problems. There has also been a great interest in distributed algorithms that compute dominating sets in wireless networks which optimize message complexity (in terms of communication rounds needed). However, many of these algorithms work only for polynomially bounded growth graphs such as Unit Disk Graphs (UDGs), which are used to model wireless networks. Other application areas cannot be modeled with such graphs and/or do not have the need for distributed algorithms.

In this thesis, we conducted an empirical study of the Minimum Independent Dominating Set (MIDS) problem to understand the practical significance of the PTAS algorithm. We compared the solutions returned by the PTAS with those of various heuristics for MIDS. For the graph instances we studied, the solutions returned by the PTAS were, at most, only as good as those returned by the best heuristic. In many instances, the PTAS returned solutions worse than those of the best heuristic. The time taken by the PTAS was prohibitively high even for small graphs. It is impractical for large and high density graphs as it uses an exact algorithm to compute the local optimal dominating set. Thus, we conclude that, while the existence of a PTAS is of theoretical interest, it is not practical to use in reality.

Metaheuristics are known to perform better than heuristics and approximation algorithms. However, to the best of our knowledge there have been very few attempts to apply metaheuristics to the problem of dominating sets so far. We study the performance of Hybrid Genetic Algorithms (HGA) and Hybrid Ant-Colony Optimization (ACO) Algorithms for various dominating set problems, i.e., the Minimum Dominating Set (MDS), Minimum Weight Dominating Set (MWDS) and Minimum Capacitated Dominating Set (CAPMDS) problems. The heuristics we studied for MDS and MWDS problems are better than the only known metaheuristic algorithms for these problems in literature. We are the first to propose metaheuristic algorithms for the CAPMDS problem. We show that our proposed metaheuristic algorithms return a better cardinality than the best known heuristics for all instances studied. This obviously comes at a cost of more time. For example, it takes, in the worst case, $\simeq 4-6$ hours for a 1000 node graph. However, metaheuristics have a role to play where minimization of cardinality is of utmost importance such as optical converter placement in WDM networks. We observed that the ACO algorithms work best when combined with local search minimization heuristic. A simple state-transition rule based simply on the pheromone deposit on the nodes is found to be sufficient. Adding a heuristic component to the state-transition rule increases computational complexity without reducing the cardinality of the dominating set found. The use of a pre-processing step helps in reducing the time required to compute the solution and in improving the solution for the instances where the search space is large.

The second part of this thesis is the novel application of dominating sets to the problem of image analysis such as skeleton and clustering of images. Images can be modeled as grid graphs. It is well known that computing MIDS and MDS is $\mathcal{NP}$-hard for grid graphs also. We extended the graph morphological operators proposed recently by Cousty et al., on the complete lattice of a grid, to use structuring elements. We use these operators to compute the distance transform of the image and show that it is identical to the standard algorithm using point sets. We are the first to apply graph morphological operators to standard graph problems such as computation of MIDS and MDS. We show that our algorithms perform as well as the best heuristics for these problems. A novel hierarchical clustering scheme based on an overlay graph of the dominating nodes is proposed. In the overlay graph, the dominating nodes form the vertices of the graph and edges are added between dominating nodes if they or their dominated nodes are neighbors of each other. The algorithm halts either after a fixed number of iterations or a single cluster is formed or there are only disconnected components in the overlay graph. We illustrated this algorithm on some sample images as part of this thesis.

In conclusion, our empirical study of the MIDS problem demonstrated that the PTAS is of theoretical interest but is not practical to use. Our proposed metaheuristic algorithms for the various dominating set problems are superior to the best heuristics in all instances. We propose a novel application of dominating sets to image analysis and propose morphological algorithms for the MDS and MIDS problems.

*To my parents, **P.Nagabhushanam and L.Lakshmikanthamma**, without whose support and encouragement, this would not have been possible.*

Acknowledgements

There are primarily two people who started me off on this journey, forcibly in the beginning, and continued to nag me until I was done. These are my father, **P.Nagabhushanam**, and my advisor, **Atul Negi**. I guess I may not even have had the motivation to do PhD if not for their persistent efforts. I would like to thank them even if I grumbled and fought with my father and resisted him a lot. My mother, **L.Lakshmikanthamma**, is probably one of the most unconventional mothers who feels that it is alright to be single as long as you do a PhD. I am proud to have such parents and I hope they are happy that I finally managed to get to this stage. I wish to thank Atul who taught me the ropes of the academic world and without whose encouragement and initial inputs, there would have been no start to this PhD. I thank my other advisors/collaborators – **Alok and Chakravarthy** – for their help and ideas in making this thesis possible. Alok ensured that I am on track with my work and scolded me on multiple occasions for not giving top priority to my PhD. Chakravarthy started seeing dominating sets in his dreams and decided to get rid of them by making me learn mathematical morphology. I am grateful to him for introducing me to a totally new and challenging field.

I lived through some really difficult times throughout the period of my PhD work – both on the professional and personal fronts. I thank my colleagues who stood by me and encouraged me to the hilt and supported me in so many various ways. I would like to thank my **"tea gang" – Alok, Vineet, Sobha, Bhavani, R P Lal, Subba Rao, Wilson, Udgata and Rukma** – for the many wonderful moments of laughter. I thank my **"lunch gang" – Atul, Chakravarthy, Bapi, Bhavani and Sobha** – for even more laughter. I want to thank Sobha and R P Lal especially here; Sobha, for always being there as a sounding board for my ideas and also bringing lunch for me every day when I decided to stay on campus to finish my work; R P Lal, for willingly being dragged into so many paper reading sessions with me as I wrestled with the approximation algorithms papers which would have been near impossible to do on my own.

I thank **Prof.Arun Pujari** for the many discussions he held with me about approximation algorithms, listened to my ideas on clustering and educated me, in his inimitable style, about the clustering concepts. **Prof.Sriram Pemmaraju** was, fortunately for me, on a sabbatical in IIIT, Hyderabad when I was starting off seriously on my PhD work. In his typical gentle manner, he showed me how ignorant I was in graph theory and the concept of NP-Completeness. I learnt a lot from him in the semester I spent time with him reading the approximation and distributed approximation papers. My knowledge and understanding of algorithms grew tremendously in this period. I am deeply indebted to him for his time and patience.

I thank my sisters **Anuradha** and **Aravinda** and brothers-in-law, **Nageswara Rao** and **Bhyrava Prasad** and my dear nephew, **Manasvi Lingam** for their support when I fell ill in this period. I am fortunate to have them for any help any time in my life.

I wish to thank the **Head and Dean** for granting me leave to pursue my PhD and also for reducing my work load once I was back from leave to facilitate early completion.

Anupama Potluri

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The White Rabbit put on his spectacles. "Where shall I begin, please your Majesty?" he asked.

"Begin at the beginning," the King said gravely, "and go on till you come to the end: then stop."

Alice's Adventures in Wonderland, Lewis Carroll

Dominating Sets have been extensively studied for a long time and there is a wealth of literature documenting the work [49, 50]. Finding a Minimum Dominating Set and its various generalizations is known to be $\mathcal{NP}$-hard [75], including for Unit Disk Graphs (UDGs) (which are used to model wireless networks) and Grid Graphs (which are used to model images) [17]. Here, we review some important applications to understand the widespread use and enormous scope of applicability of various kinds of dominating sets.

Dominating sets find tremendous use in wireless networks, which leading to a great surge of interest in them, in the recent times. Many types of dominating sets such as Minimum Dominating Set (MDS), Minimum Independent Dominating Set (MIDS), Minimum Connected Dominating Set (MCDS) have been used extensively in clustering of wireless networks. Routing with the dominating set as the backbone has been used to reduce flooding. Clustering lends a hierarchical structure to wireless networks and makes them more manageable, especially in ad hoc and sensor networks [15, 33]. Clustering and routing using dominating sets reduce the energy dissipated in the network prolonging the network lifetime. This is extremely important for sensor networks and wireless ad hoc networks.

In addition to routing and clustering in wireless networks, minimum weight dominating set (MWDS) has been used in gateway placement in wireless mesh networks such that the number of gateways is minimized while also satisfying any QoS requirements [4]. It has been used to reduce the number of full wavelength converters during deployment of Wavelength Division Multiplexing (WDM) all-optical networks to reduce the cost of the deployment and overcome technological limitations [55]. Another use of weighted dominating set is to determine the nodes which actively participate in intrusion detection in ad hoc networks [97]. The nodes in the dominating set actively sniff the packets traversing the ad hoc network and detect anomalous behavior. MWDS has most recently been used in information retrieval for multi-document summarization [87]. In this work, the sentences in the documents are represented by nodes and an edge exists between nodes if the sentences are similar. The sentences corresponding to the minimum dominating set of the resulting graph form a summary of all the documents. MWDS has also been used in query selection techniques for harvesting data records from web databases [104]. The information contained in the hidden web databases is accessible through the various queries on the databases. In [104], the structured web database is modeled as an attribute-value graph. The extraction of an efficient set of queries that extract information from the hidden database is shown to be equivalent to MWDS.

Finding Minimum Capacitated Dominating Set (CAPMDS) is the problem of allocating network centers [5] such that there is a balanced load on each center. In the case of MDS, there is no limit on the number of nodes that are serviced by a dominating node. However, every node has finite resources such as battery power or bandwidth in a wireless node. If too many nodes are covered by a single node, the node may die out or communication may fail due to lack of available bandwidth. The capacity of a node is modeled as a value associated with each node which represents the limit on the number of nodes that can be dominated by that node. This problem has many other applications such as facility location, distributed databases and distributed data structures [5].

Minimization of the dominating sets leads to many advantages such as reducing the number of nodes that form the backbone for routing and therefore the amount of communication needed in the network, which translates into energy efficiency. In information retrieval, the minimality leads to a more efficient search as the number of items to be searched against will be reduced. In full-wavelength optical converters, even a small reduction in cardinality can lead to a huge savings in cost. Since the WDM network design and deployment is done once, any solution that reduces cardinality while being practical is acceptable. Thus, having seen that minimization of dominating sets is immensely useful, we focus on the problems themselves and the various approaches proposed in the literature for solving them.

1.1 Algorithms for Dominating Sets

Most of the algorithms in the literature for Minimum Dominating Set and its generalizations have focused on approximation algorithms. Some of the approximation algorithms focus on the cardinality of the solution; the distributed algorithms focus on the communication complexity of the algorithms, in addition to the cardinality. A number of these algorithms are applicable only to UDGs or other limited types of graphs. Many of them do not discuss the time complexity of the algorithm itself.

The distributed approximation and Polynomial Time Approximation Schemes (PTASs) [43, 44, 56, 60, 61, 77, 84, 101, 105] have been proposed for various dominating set problems in the context of wireless networks. The aim of the distributed approximation algorithms has been to reduce the communication complexity such that the energy expenditure of the network is reduced, leading to a longer network lifetime.

All PTAS algorithms are applicable only to Bounded Independence Graphs (BIGs), of which, Unit Disk Graphs (UDGs) are a special case. It was observed by us, as part of the work for this thesis, that the PTAS for MIDS is not very practical to use in high density graphs as it takes a prohibitively long time to run. The PTAS algorithms proposed for various dominating set problems use an exact algorithm for the local neighborhood which becomes extremely costly as the number of nodes in the local neighborhood grows. Further, these algorithms are not useful for other applications such as optical networks or information retrieval etc., where the topology is more likely to be a general graph. Moreover, these applications also neither need distributed algorithms nor is it essential that the algorithms run in real-time.

We need other techniques to solve the problem of minimizing the dominating sets which work for general graphs and complete in practical time. Metaheuristics have been used to solve many $\mathcal{NP}$-hard problems and found to be highly effective. Our literature survey of the dominating sets made us realise that there has been hardly any attempt to solve these problems using metaheuristics. The very few metaheuristic algorithms for dominating sets in literature, we felt, could be improved upon significantly by better choice of algorithms. We also felt that hybridization with the right heuristics can help improve the solution quality.

1.2 Metaheuristics for $\mathcal{NP}$-Hard Problems

Metaheuristics such as Genetic Algorithms (GA), Ant-Colony Optimization (ACO) Algorithms, Aritificial Bee Colony (ABC) Algorithms, Tabu Search and others have been proposed primarily to solve $\mathcal{NP}$-Hard problems. These algorithms have the characteristic of being applicable to any problem, independent of domain. However, it has been proved that combining them with problem specific heuristics significantly improves the quality of the solution returned by them. Dominating set problems are subset-selection problems. Metaheuristics have been applied to many subset selection problems [93] such as maximum clique [94], minimum vertex cover [89, 90], leaf-constrained minimum spanning tree [91] and others. They are proved to compute solutions superior to the best known heuristics for the respective problems.

Genetic Algorithms are based on the evolutionary processes in nature. The algorithm starts off by constructing an initial population whose members are changed from generation to generation. The concept of *fitness* is used to determine the quality of the members of the population. As generations progress, the more "fit" members are retained and others are discarded. The generation of a new member is dependent on the parent selection policy, the way they are recombined and the mutation policy. The member may also be optimized further by the use of a local search algorithm before it is inserted into the population. The algorithm halts after running for a specified number of generations or a certain number of generations without improvement in the quality of the best solution.

Ant-Colony Optimization is based on the stygmergic behavior of ants to converge to a single better path to the food source, if multiple different paths are available. Ants deposit pheromone on the path they traverse and probabilistically follow a path with more pheromone deposit. As more ants traverse a specific path, the pheromone deposit on it increases until all ants follow the better path. This behavior has been modeled and improved over time and applied to many optimization problems. The basic ACO algorithm is dependent on the state-transition rule that determines the path an ant follows and the pheromone updation rule that determines by how much amount the pheromone on a path is increased or decreased. By controlling the parameters associated with these rules and combining heuristics in the state-transition rule, ACO algorithms have been found to be highly effective in finding to optimal solutions. They are further improved by the addition of a local search technique to improve the solution constructed by the ant by its random walk.

In this thesis, we study the performance of the genetic algorithms and ant-colony optimization algorithms for the dominating set problems. While the metaheuristics improve the cardinality of the solutions at the cost of more time, we find that the time taken is $\simeq 4-6$ hours in the worst case for 1000 node graphs. This is still practical enough for problems such as the optical converter placement in WDM networks that are solved only once. They are also practical for offline algorithms such as multi-document summarization that may be run once a day or less.

1.3 Morphological Algorithms for Dominating Sets and their Application to Image Analysis

Mathematical morphology, a non-linear, set-based approach to image processing, developed originally by Matheron and Serra[85] provides a number of powerful tools for image analysis. Developments in morphology have taken mainly two paths. The first is extending the morphological operations developed for binary images to a number of domains such as gray scale images[95], graphs[9, 52], fuzzy sets[8, 78], color images[18, 82], simplical complex spaces [27] etc. The second path is the development of new operations such as opening and closing distributions[14], pattern spectra[66], area morphology[2], watersheds[21, 72], etc.

Graph-theoretic approaches have found increasing applications in image analysis from the turn of this century. Some notable contributions are in segmentation[34, 88] and computer vision[35]. Classical graph problems such as minimum spanning tree (MST), cuts and partitioning, and sub-graph isomorphism[11] have been applied to such image analysis tasks. Though the morphological operators were extended to graphs in

[52], recently, Cousty, et al. [22] have proposed a different set of graph morphological operators that are shown to be dilation and erosion operators.

While dominating sets have been used for many purposes, as shown previously, they have not been applied before to image analysis. We have taken the first steps in this direction and shown that dominating sets can be used effectively for representing the skeleton of images and in clustering of images. We extend and use the graph morphological operators proposed in [22] to construct algorithms for standard graph problems such as computation of MDS and MIDS. This is, to our knowledge, the first attempt to construct morphological algorithms for standard graph-theoretic problems. Since morphological algorithms are highly amenable to parallel processing, these algorithms can help in parallel computation of MDS and MIDS.

1.4 Thesis Contributions

In this thesis, we conducted the *first empirical study* of various exact, PTAS and heuristic algorithms for MIDS. We find that the exact algorithm proposed by Liu and Song [64] has the better asymptotic complexity, but takes more time than an intelligent enumeration algorithm, in practice, on small graphs. The PTAS is useful only in small and low density graphs and even in these, returns a cardinality matched by the best heuristic for MIDS, the maximum degree heuristic. The results of this study are published in *Proceedings of IC3-2011.*

This thesis presents the *first serious attempt to apply the metaheuristic algorithms* to the problem of dominating sets. This attempt was inspired by the fact that many of the approximation algorithms are not applicable to general graphs and our empirical results show that the PTAS for MIDS is not so useful in practice even for UDGs. We find that our proposed metaheuristic algorithms for MDS and MWDS are superior to the metaheuristic algorithms in literature. They also improve upon the results of the best heuristics. In fact, since the heuristic for MDS is also the optimal approximation algorithm, the improvement in results using metaheuristics is quite useful for applications where minimization of cardinality is crucial. The work on MDS is published in *Proceedings of SEMCCO-2011.* The work regarding MWDS has been has been accepted for publication in *Applied Soft Computing.*

We also propose simple greedy heuristics for the problem of CAPMDS and compare the performance of these heuristics on UDGs and general graphs with uniform and variable capacity. We propose the *first ever metaheuristic solutions for the CAPMDS problem.* These use the best proposed heuristic for the pre-processing step to guide the solutions towards more promising search space. A well-designed minimization heuristic is used to reduce cardinality of the solutions generated by the metaheuristics. It is seen that the metaheuristics perform really well when the capacity and degree of graphs is low. More work needs to be done to reduce cardinality when the degree is high and capacity is low, by improving the coverage algorithm used to determine the neighbors that are covered by a dominating node. We confirm that the use of a pre-processing step works only where the search space is large and is not effective otherwise in improving the quality of the solution. We also found that using the heuristic component in the state-transition rule does not improve the quality of the solution in any of the problems considered. The work regarding comparison of the greedy heuristic and its variants has been published in *Proceedings of IC3-2012* and the work relating to the metaheuristics is under review in the journal *Swarm and Evolutionary Computation.*

We propose the *application of dominating sets to image analysis*, specifically, the use of dominating sets as the *skeleton of the image and in clustering of images.* Since images can be modeled as grid graphs and morphology plays an important part in image analysis, we proposed novel morphological algorithms to compute MDS and MIDS. In fact, this is the *first attempt to apply graph morphological operators to standard graph problems* such as computing dominating sets. For this purpose, we extended the graph morphological operators proposed by Cousty et al. [22] to use structuring elements. Finally, we propose a *new hierarchical clustering scheme* based on the *overlay graph of dominating nodes* and illustrate it on some sample images. This, to our knowledge, is the first time dominating sets have been used in hierarchical clustering. This work has been accepted for publication in the *International Workshop on Combinatorial Image Analysis (IWCIA-2012).*

1.5 Organization of the Thesis

There are three parts to this thesis. In the **first part** of the thesis, we introduce the various concepts used in the rest of the thesis. Chapter 2, Preliminaries, Foundational

Concepts and Related Work chapter, introduces the basic concepts of independent sets, dominating sets, matching in graphs and bounded-independence graphs. It also introduces the various concepts regarding genetic and ant-colony optimization algorithms, which are needed to explain the metaheuristic algorithms in literature and our own proposed hybrid metaheuristic algorithms. We also introduce the concepts of mathematical morphology and the algorithms for skeleton and distance transform using morphology of point sets in this chapter. We end with the review of some of the literature regarding the approximation algorithms and Polynomial Time Approximation Schemes (PTASs) for various generalizations of dominating sets.

In Chapter 3, we present the results of our empirical study of PTAS and various greedy heuristics for MIDS. This gave us an insight into the working of PTAS and exact algorithms which resulted in our exploration of metaheuristic algorithms.

The **second part** of the thesis consists of two chapters: 4 and 5. The proposed metaheuristic solutions for MDS and MWDS are presented in Chapter 4. We studied many variants of the greedy heuristic for MWDS. The metaheuristic algorithms in literature, against which we compared our algorithms, are also described in this chapter. The comparison of the cardinality returned by the proposed metaheuristics with the heuristics and other metaheuristic solutions is presented next. We propose a greedy heuristic for the CAPMDS problem and its two variants in Chapter 5. We compare these variants on both general graphs and UDGs to determine the better algorithm. The metaheuristic algorithms proposed for MDS and MWDS in Chapter 4 are extended to CAPMDS. These are, then, compared against the best heuristic for different types of graphs with uniform and variable capacity on the nodes.

The **third part** of the thesis deals with the morphological algorithms to compute dominating sets and is presented in Chapter 6. We also present the algorithms to compute distance transform using the extended graph morphological operators, skeleton using MIDS and the hierarchical clustering of images using dominating sets in this chapter.

The thesis concludes with Chapter 7 that summarizes the results and presents directions for future work.

Chapter 2

Preliminaries, Foundational Concepts and Related Work

"When I use a word," Humpty Dumpty said in rather a scornful tone, "it means just what I choose it to mean – neither more nor less."

"The question is," said Alice, "whether you can make words mean so many different things."

Through the Looking Glass, Lewis Carroll

In this chapter, we first introduce some basic definitions of various graph theoretic concepts such as independent sets, different types of dominating sets and matching and some known complexity results. We also define specialized types of graphs that are encountered in this thesis.

Dominating sets have been studied for many years now and there is a wealth of literature on dominating sets [49, 50]. Recently, many algorithms have been proposed for clustering using different generalizations of dominating sets such as Independent Dominating Sets, Connected Dominating Sets and so on [15, 33]. Many algorithms that reduce communication complexity have also been proposed as it is important that the energy dissipated for communication in wireless networks be minimal. We review the state-of-the-art literature of approximation algorithms for dominating sets in Section 2.3.1.

While there is a lot of literature exploring the exact and approximation algorithms for various dominating sets, there have been very few attempts to apply metaheuristics to this problem. In this thesis, we propose and study metaheuristic algorithms for dominating sets and evaluate their effectiveness. We review the concepts of the meta-heursitics used viz., Genetic Algorithms (GA) and Ant-Colony Optimization (ACO) algorithms, in Sections 2.2.1 and 2.2.2.

We use graph morphology to propose novel algorithms to compute dominating sets and show their use in image analysis in Chapter 6. In Section 2.2.3, we introduce basic mathematical morphology concepts and review the concepts of distance transform and skeleton of images.

2.1 Preliminaries

In this section, we start with definitions of the basic graph theoretic concepts relating to dominating sets and the known complexity results relating to them. We also discuss some of the types of graphs such as Unit Disk Graphs (UDGs) which are used to model wireless networks. Many of the approximation algorithms in literature are limited to such graphs.

2.1.1 Definitions and Known Complexity Results

A graph $G = (V, E)$ is a finite set of vertices (nodes) and E is the finite set of edges. An edge is an unordered pair of vertices, (u, v), where u and v are distinct elements of V. An example graph is given in Fig. 2.1, where circles represent vertices and lines represent edges. We use this graph to illustrate the various concepts introduced in this chapter.

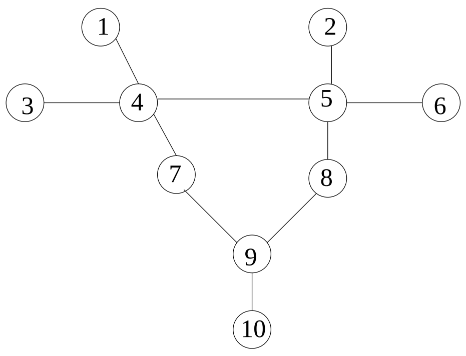


Figure 2.1: An Example Graph.

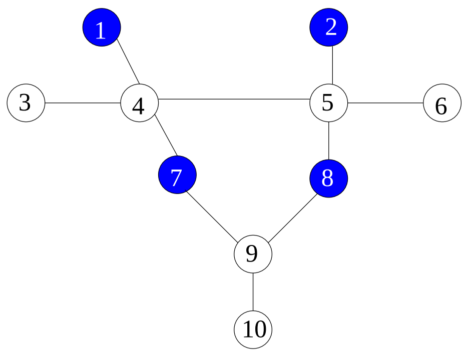


Figure 2.2: Nodes {1, 2, 7, 8} represent an Independent Set of the graph of Fig. 2.1.

Neighborhood: The neighborhood of a node $v \in V$ is $N(v) = \{u : (u, v) \in E\}$, i.e., the set of all nodes which are *adjacent* to it. This is also called the *open* neighborhood of a node v. The *closed* neighborhood of a node $N[v] = N(v) \cup v$. The number of neighbors of a node v is called the *degree*, $d(v) = \mid N(v) \mid$, of the node v. The minimum degree in the graph is represented by δ and the maximum degree by Δ. A k-hop neighborhood of a node $N_k(v)$ is the set of nodes that are, at most, k hops away from the node. In other words the open neighborhood is a special case of the k-hop neighborhood, with $k = 1, N_1(v)$. The open neighborhood of a set $S \subset V$ is $N(S) = \cup_{s \in S} N(s)$. Similarly, the closed neighborhood is $N[S] = \{\cup_{s \in S} N(s)\} \cup S$.

Independent Sets: Given a graph $G = (V, E)$, an **independent set** of the graph is a subset $S \subset V$ such that no two nodes of S are adjacent to each other. The nodes {1, 2, 7, 8} form an independent set for the example graph as shown in Fig. 2.2.

Definition 1. *A **maximal independent set (MIS)** is an independent set that cannot be extended without violating the independence property of the set. A **maximum independent set (MaxIS)** is a maximal independent set with the maximum cardinality.*

A maximum independent set for the example graph is given in Fig. 2.3 and consists of the nodes {1, 2, 3, 6, 7, 8, 10}.

Bipartite Graph: A *bipartite graph* is a graph in which the nodes of the graph can be divided into two disjoint sets $U \subset V$ and $W \subset V$ such that for all edges $\{(u, w) \in E \mid u \in U, w \in W\}$. An example bipartite graph is given in Fig. 2.4.

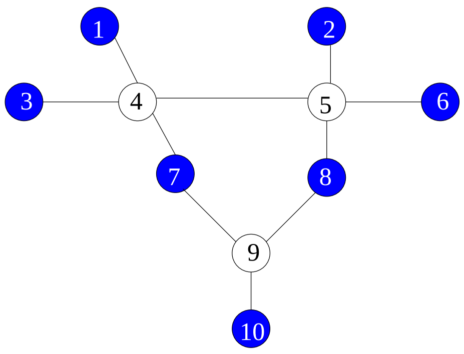


Figure 2.3: Nodes {1, 2, 3, 6, 7, 8, 10} form a Maximal Independent Set of Fig. 2.1 which also happens to have the *maximum* cardinality. In other words, the *Maximum* Independent Set.

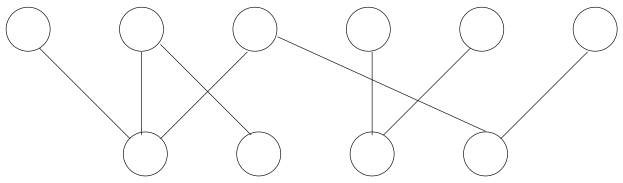


Figure 2.4: An example *Bipartite* graph.

2.1.2 Dominating Sets

A **dominating set** is a subset $S \subseteq V$ such that every node $v \in V$ is either a member of S or is adjacent to a node $u \in S$. A node $v \in S$ is called a *dominating node* whereas a node $v \in V \setminus S$ is called a *dominated node*, if it is adjacent to a dominating node. An example of a dominating set is given in Fig. 2.5, where the dominating nodes {1, 3, 5, 7, 10} are the shaded nodes. It can be seen from the figure that all the other nodes are adjacent to at least one of the nodes in the dominating set. Every maximal independent set is, by definition, a dominating set. Since it also satisfies the property of independence, it is also called an *independent dominating set*.

Definition 2. *A dominating set of minimum cardinality is called the* ***Minimum Dominating Set (MDS).*** *Its cardinality, γ, is called the* ***domination number*** *of the graph.*

Definition 3. *A maximal independent set of minimum cardinality is called the* ***Minimum Independent Dominating Set (MIDS).*** *The cardinality of MIDS, i, is called the* ***independent domination number*** *of the graph.*

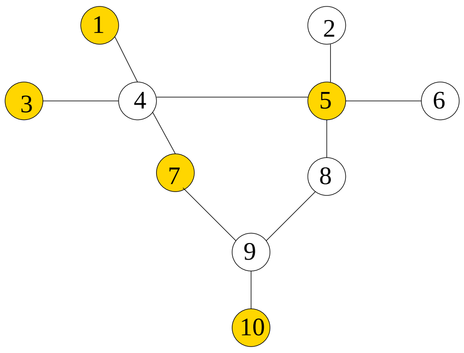


Figure 2.5: A Dominating Set of the graph in Fig. 2.1 where the dominating nodes are $\{1, 3, 5, 7, 10\}$.

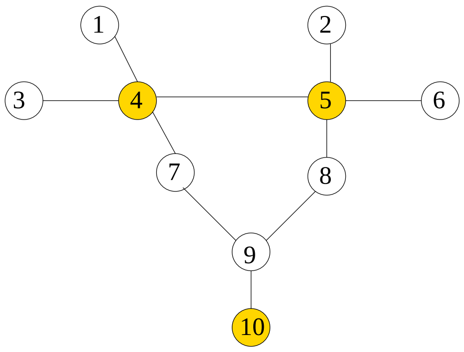


Figure 2.6: A Minimum Dominating Set of the graph of Fig. 2.1 consisting of nodes $\{4, 5, 10\}$.

The MDS and MIDS of the example graph are given in Figs. 2.6 and 2.7 respectively.

A dominating set, whose members induce a connected subgraph of G, is called a **connnected dominating set.** It is also called a *strongly* connected dominating set. An example of a strongly connected dominating set is given in Fig. 2.8. If all the dominating nodes are connected to each other through a single non-dominating node as in Fig. 2.9, it called a *weakly* connected dominating set.

Definition 4. *A connected dominating set of minimum cardinality is called the* ***Minimum Connected Dominating Set (MCDS).***

A *weighted graph* is a graph in which a non-negative weight function $w : V \longrightarrow \mathbb{R}^+$ is associated with the vertices.

Definition 5. *A* ***Minimum Weight Dominating Set (MWDS)*** *is a dominating set, D, such that $\sum_{v \in D} w(v)$ is minimized.*

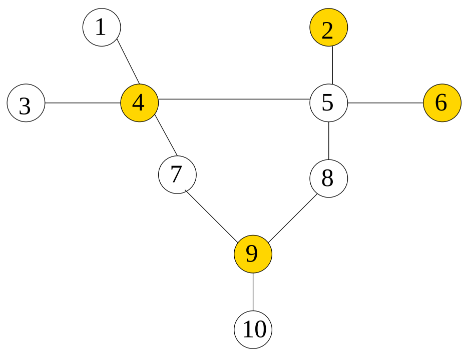


Figure 2.7: A Minimum Independent Dominating Set of the graph of Fig. 2.1, with nodes {2, 4, 6, 9} forming the MIDS.

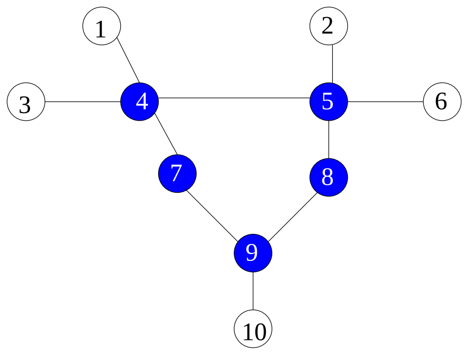


Figure 2.8: A *Strongly* Connected Dominating Set of the graph in Fig. 2.1 with the nodes {4, 5, 7, 8, 9} forming the dominating set.

In some graphs, every node $v \in V$ has a *capacity* denoted by $cap(v) \geq 1$, which limits the number of neighbors that the node can dominate. A capacitated dominating set is a subset $S \subseteq V$ such that the number of nodes dominated by a node does not exceed its capacity. It is formally defined as follows: there is a mapping $\psi : (V \setminus S) \rightarrow S$, $\psi(u) \in N(u), \forall u \in (V \setminus S)$ and $| \{u \ : \ \psi(u) = v\} | \leq \ cap(v), \ \forall v \in S$.

Definition 6. *A* ***Minimum Capacitated Dominating Set (CAPMDS)*** *is a capacitated dominating set of minimum cardinality.*

2.1.3 Bounded Independence Graphs

A graph $G = (V, E)$ is said to be a Bounded-Independence Graph (BIG) if the size of the maximum independent set of the r-hop neighborhood of a node v, $N_r(v)$, is bounded by a function $f(r), \forall r \geq 0$. If the function $f(r)$ is a polynomial, $O(r^c)$, where

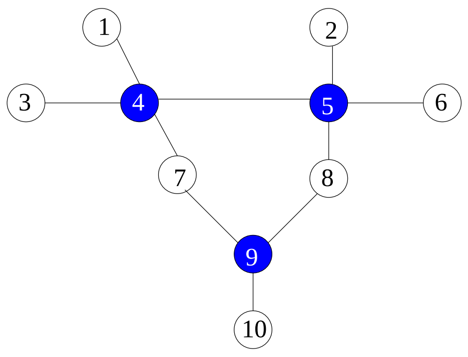


Figure 2.9: A *Weakly* Connected Dominating Set of the graph in Fig. 2.1. The nodes $\{4, 5, 9\}$ are the dominating nodes and they are connected through the non-dominating nodes $\{7, 8\}$.

$c \geq 1$, such graphs are called as **polynomially bounded growth graphs** [77, 83].

Definition 7. *Consider n equal-sized circles laid on a plane. Each circle represents a node in the graph and an edge exists between nodes if the circles intersect. Such an intersection graph is called a* ***Unit Disk Graph (UDG)*** *[17].*

UDGs, which are one form of BIGs, are typically used to model wireless communication networks. UDGs are not planar graphs. An example UDG is given in Fig. 2.10.

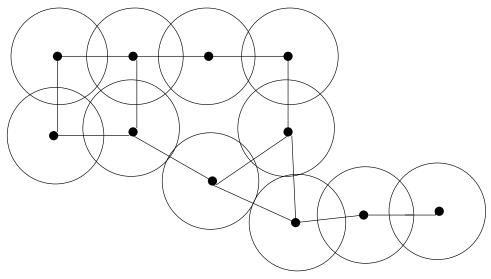


Figure 2.10: An example Unit Disk Graph.

Definition 8. *A* ***grid graph*** *is a unit disk graph in which all centers have integer coordinates and the radius of the disk is 1/2 [17]. It can also be defined as a subset of the complete grid graph, where the grid can be a square or a rectangle.*

The intersection of the grid lines represents a node and edges are represented by the horizontal and vertical grid lines. The grid graph is a planar graph. An example grid graph is given in Fig. 2.11.

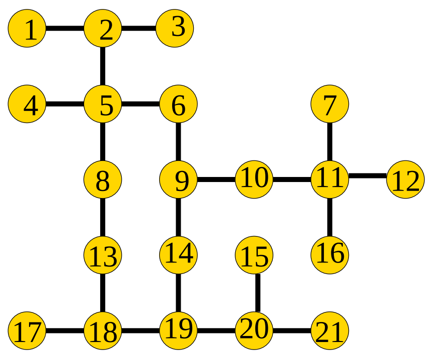


Figure 2.11: An example Grid Graph.

Finding the Minimum Dominating Set and its various generalizations is $\mathcal{NP}$**-hard [75], including on *Unit Disk Graphs* and *Grid Graphs* [17].**

2.1.4 Matching in Graphs

Given a graph $G = (V, E)$, a matching $M \subset E$ of G is the set of pairwise non-adjacent edges, i.e., no two edges of M share a vertex. A *maximal matching*, (Fig. 2.12) is a matching which cannot be extended without violating the property of non-adjacency.

Definition 9. *A maximal matching with the largest cardinality is called* ***Maximum Matching****, seen in Fig. 2.13.*

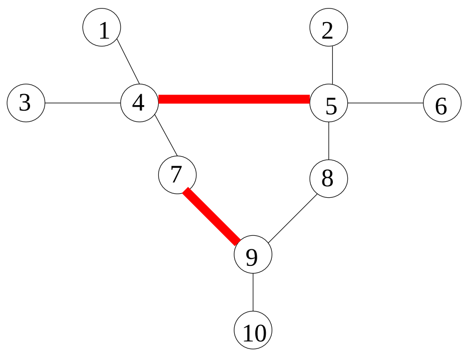


Figure 2.12: The thick edges represent the Maximal Matching of the graph in Fig. 2.1.

Definition 10. *A* ***perfect matching*** *is the matching which has every vertex of the graph covered by the edges of the matching. Every perfect matching is a maximum matching.*

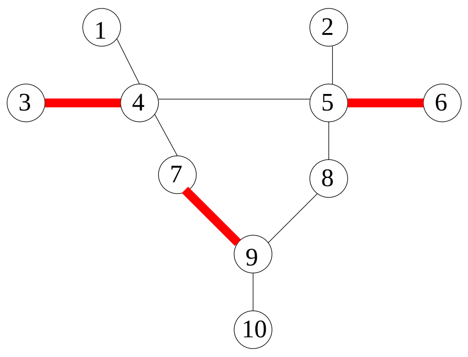


Figure 2.13: The thick edges represent the *Maximum* Matching of the graph in Fig. 2.1.

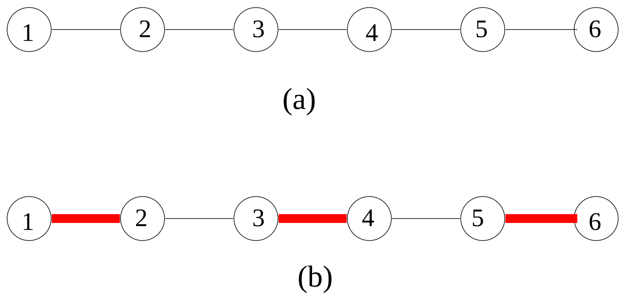


Figure 2.14: (a) An example graph and (b) its *Perfect* Matching represented by the thick edges.

An example graph and its perfect matching are given in Figs. 2.14(a) and 2.14(b).

The Maximum Matching of a bipartite graph is the integral max-flow of the bipartite graph [19].

2.2 Foundational Concepts

We introduce the basic concepts of Genetic Algorithms and Ant-Colony Optimization Algorithms, the two metaheuristic techniques used in our study, in this section. We end this section with the concepts of mathematical morphology and the standard morphological algorithms to compute distance transform and skeleton of images.

2.2.1 Genetic Algorithms

Genetic Algorithms (GA) were introduced by Holland [54] based on the evolutionary processes in nature. The genetic algorithms are primarily iterative algorithms that halt after a specified number of iterations or after a time constraint is reached or if there is no improvement in the quality of the best solution for a specified number of consecutive iterations. Each iteration is called a *generation*. In each generation, a population consisting of a specified number of individuals, N, is generated. To generate the population of the next generation from the current generation, parents are selected based on some criteria. New members of the population, created through recombination of chosen parents and possible mutation, replace the older generation. Each individual is characterised by a *fitness*, which determines the quality of the solution represented by this member.

We can characterise a genetic algorithm by the representation scheme chosen for the problem, the method by which the initial population is generated, population replacement scheme or survivor selection, parent selection mechanism, recombination or crossover technique and mutation. Each of these components has many methods proposed in literature. Depending on the domain of the problem, the right method for each of these components has to be chosen. In the rest of this section, we briefly describe the various methods proposed in literature for each of these components, with special emphasis on those that are more appropriate to subset-selection problems such as the dominating set.

2.2.1.1 Representation

Many problems, especialy subset selection problems, can be coded with binary representation. In this, the *chromosome*, which represents a sample solution, is encoded as a binary string. In subset selection problems, a value of '1' indicates that particular member of the set is chosen to be part of the subset whereas a '0' indicates its absence. Binary encoding for subset selection problems results in fixed length chromosomes in comparison to variable length chromosomes resulting from encoding a subset directly. For other problems which deal with integer and real numbered values, other representations exist. A problem, such as the Travelling Salesman Problem (TSP), where the order of the subset is important, uses a permutation representation.

2.2.1.2 Fitness

The fitness of a member of the population is a measure of how good a member is with respect to the objective function of the problem. In dominating sets, minimization of cardinality is the main objective. Fitness is, therefore, defined as the cardinality of the member. The smaller the cardinality of the member, the better its fitness.

2.2.1.3 Initial Population Generation

The most common method for generating initial population is to generate many random solutions for the problem. When the search space is very large, a purely random choice for the initial population may not result in a good solution within the practical time constraints. In such cases, we may initialize the population with members obtained from standard greedy heuristics for the problem or some other method which gives a better fitness for the initial population. This is called *seeding* the initial population.

2.2.1.4 Population Replacement

There are two major types of population replacement or survivor selection methods proposed in literature: generational replacement and steady-state replacement [25]. In the former, N new members are generated and replace the entire previous generation, whereas in the latter, a few of the worst members, λ, are replaced by an equal number of new members. There is a slight variation of generational replacement, where the k best members generated so far are retained across generations in an attempt to ensure they survive. This is called *elitism*.

2.2.1.5 Parent Selection

Parent selection is an important part of the genetic algorithm as it plays an important role in the generation of new members and their fitness. Some of the techniques for parent selection as given in [32] are :

1. **Fitness Proportionate Selection:** This is also called the Roulette Wheel Selection. Each member of the population, i, is selected with a probability $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where f_i is the fitness of i. This has some drawbacks such as premature convergence where some members dominate the whole population too early and slow improvement of performance once the fitness of most members of the population is similar.

2. **Ranking Selection:** Ranking selection was put forth to overcome some of the issues with fitness proportionate selection. In this method, the members are ranked according to their fitness. The selection is based on the roulette wheel of rank rather than on the fitness value itself. In Linear Rank Selection, the worst fitness is usually given a rank of 1 and the best fitness a rank of N , where N is the population size. The expected value of a member, $p_i = min + (max - min)\frac{rank(i)-1}{N-1}$. The values normally used for min and max are $1 \leq max \leq 2$ and $min = 2 - max$ [73].

3. **Tournament Selection:** In tournament selection, k individuals of a population are picked randomly to participate in a tournament. The winner is the member which has the highest fitness. This becomes one of the parents used for recombination. Another tournament selects the second parent. The recombination of these results in a new member of the next generation. This process is repeated until N or λ members are generated for a generational GA or steady-state GA respectively.

   If $k = 2$, it is called a binary tournament. In this, two individuals are chosen at random. The better of the two individuals is chosen as a parent with a probability p_{better}, where $0.5 < p_{better} \leq 1$. This is proven to have the same properties as the linear ranking selection [45]. It is one of the most popular selection techniques in genetic algorithms as it is easy to use.

2.2.1.6 Recombination or Crossover

Recombination or crossover is the reproduction of new members by combining characteristics of two parents. This is expected to lead to an improvement in the quality of the solution from one generation to the next. There are many possible crossover techniques based on the representation scheme chosen. For binary representation, the following are the most commonly used crossover techniques:

1. **Single-point Crossover:** A random position, r, in the binary string is taken and the parents are split at this point. The bits 1 to r of parent p_1 are combined with the bits $r + 1$ to l of parent p_2, where l represents the length of the string and vice versa, creating two new children.

2. **Multi-point Crossover:** In this the binary string is broken into more than two parts and the parts from the two parents are interspliced to form the children.

3. **Uniform Crossover:** In uniform crossover, the bits are chosen from each parent with a defined probability. Thus, if the probability for parent p_i is pr_i and parent p_j is $pr_j = 1 - pr_i$, if the generated probability $p < pr_i$, the bit is chosen from p_i. Otherwise, it is chosen from p_j. This is the more popular crossover technique in use.

There are many other types of crossover for the integer, floating point and permutation representations.

2.2.1.7 Mutation

Mutation schemes depend on the representation chosen for the problem. In binary representations, the standard mutation method is the *bit-flip* method. Each bit is allowed to “flip” with the probability defined for mutation, p_m. The mutation operators defined are different for the other representations: random resetting and creep mutation for integer representations, uniform mutation and non-uniform mutation with a fixed distribution for floating point representations and swap mutation, insert mutation, scramble mutation and inversion mutation for permutation representations [32].

2.2.2 Ant-Colony Optimization Algorithms

The Ant-Colony Optimization (ACO) algorithm [28, 29, 30] is based on the intelligent foraging behavior of ants and is one of the swarm intelligence techniques proposed for solving combinatorial optimization problems. In real life, ants deposit a chemical called pheromone on paths and an ant tends to probabilistically choose a path that has more pheromone deposited on it. A path with less pheromone is still followed by a few ants to facilitate path exploration. The pheromone evaporates with time which helps in path exploration. Experiments with ants using a double bridge path to the food source have shown that, initially, the ants follow both paths to the food source equally. But, after a point of time, they converge to a single path due to stochastic differences in the number of ants that follow a particular path [26, 47]. As the pheromone deposit on one path increases due to the small difference in the number of ants following a path, more and more ants follow this path and eventually converge on to this path. In other experiments with ants, it was discovered that once the ants start following a longer path, even if a shorter path is introduced, they do not change the path as the pheromone evaporation is too slow to help discover the new path [47]. This shows that ants can get trapped into bad paths by following only the pheromone trail.

This stygmergic behavior of the ants, to converge to a path, was modeled to solve combinatorial optimization problems in an algorithm called the Ant System [28]. This was further developed into the Ant-Colony Optimization algorithm. The ACO algorithm has been applied to many subset selection problems ([10, 36, 53, 62, 89, 93, 94]), bounded capacity problems such as Capacitated Arc Routing Problem (CARP) [81] and Capacitated Minimum Spanning Tree Problem (CMST) [79] and multi-objective optimization problems [1].

The Ant System was first applied to the Travelling Salesman Problem [28, 29]. The basic ACO algorithm for any problem is given in Algorithm 1. The algorithm iterates for a certain number of times or until there is no improvement in the quality of the solution for a given number of successive iterations. In each iteration, a maximum number of ants are used to explore the solution space through a random walk of the graph. The next node/edge to be added to the solution is based on a probabilistic state-transition rule. This state-transition rule is based on the pheromone deposit of the nodes/edges in the given graph. It may also, optionally, include other heuristic

Algorithm 1: ***General ACO Algorithm***

```
for $I := 1 \rightarrow MAX - ITER$ do
  for $A := 1 \rightarrow MAX - ANTS$ do
    Choose the next node/edge to add to the solution using the probabilistic
    state-transition rule.
    Add node/edge chosen to the solution
  end for
  Enhance Solution using Local Search (Optional)
  Update_Pheromone
end for
```

information specific to a problem. At the end of exploration by all ants, the pheromone deposit on the nodes and/or edges in the graph is updated. During updation, the pheromone is evaporated on all nodes/edges. In addition, pheromone is deposited on the nodes/edges that are part of a solution so that these nodes/edges are preferred by ants in future.

There are many variations of the ACO algorithm based on the state-transition and pheromone updation rules used [30]. We will describe each of these variations briefly, using the Travelling Salesman Problem (TSP), in the rest of this section. The objective in TSP is to construct a tour of the graph such that each city in the tour is visited exactly once and the cost of the tour is minimized.

2.2.2.1 Ant System

In the Ant System algorithm [29], the state-transition rule is based on the pheromone trail on the edges and heuristic information. The heuristic information used is $\eta_{ij} = 1/d_{ij}$ where d_{ij} represents the distance between cities i and j. The probability with which ant k at city i chooses the next city j to visit is given by:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}^{\alpha}\eta_{il}^{\beta}}, \; if \; j \in \mathcal{N}_i^k \tag{2.1}$$

where $\mathcal{N}_i^k$ is the set of neighbors of i that have not already been visited by the ant k. The parameters α and β determine the relative importance of pheromone and heuristic in the selection of the next city. As pointed out earlier, too much importance to the pheromone trail can trap the algorithm in a local optimum. The fine tuning of these parameters can help in improving the quality of the solution returned by the ACO algorithm.

The pheromone evaporation is given by

$$\tau_{ij} = (1 - \rho)\tau_{ij} \tag{2.2}$$

where ρ is the pheromone evaporation rate. This is done for all the edges of the graph. Each ant, k, deposits pheromone on the edges that constitute the tour constructed by it, T^k, using the formula:

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^k \tag{2.3}$$

where $\Delta\tau_{ij}^k$ is the amount of pheromone deposited which is defined based on the cost of the solution, C^k, found by the ant k:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C^k & \text{if edge } (i,j) \text{ belongs to } T^k \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

2.2.2.2 Elitist Ant System

The only difference between the Ant System and Elitist Ant System (EAS) [28] is that, in addition to the ants of each iteration, the *best-so-far* ant also deposits additional pheromone on the edges of its tour, T^{bs}. EAS also uses an additional parameter, e, that is the weight given to the *best-so-far* ant. The pheromone update rule now changes from Eqn. 2.3 to the following:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k + e\Delta\tau_{ij}^{bs} \tag{2.5}$$

where $\Delta\tau_{ij}^k$ is as defined in 2.4 and $\Delta\tau_{ij}^{bs}$ is defined as:

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \text{if edge } (i,j) \text{ belongs to } T^{bs} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

2.2.2.3 Rank-Based Ant System

The Rank-Based Ant System was proposed by Bullnheimer et al. in 1999 [13]. In this, the ants are ranked based on the length of the tours constructed by them. Unlike in the Ant System, only $w - 1$ ants are allowed to deposit pheromone on the edges based on their tours. The best ant, ranked 1, is given a weight of w and its deposit is multiplied by w. The pheromone deposited by an ant ranked r is given a weight of $w - r$. The pheromone deposited is given by:

$$\tau_{ij} = \tau_{ij} + \sum_{r=1}^{w-1}(w - r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs} \tag{2.7}$$

2.2.2.4 Ant Colony System

The Ant Colony System (ACS), which is the most commonly used form of ACO algorithms, differs from the Ant System in three ways. Firstly, when an ant k on node i wants to move to the next node j, the node j is chosen using a *pseudorandom proportional* rule. It defines a random variable, q, uniformly distributed in [0, 1] and a parameter $q_0, 0 \leq q_0 \leq 1$ such that if $q \leq q_0$, it always chooses the node with the maximum pheromone-heuristic value. Otherwise, it chooses a node based on the standard probabilistic rule. The state-transition rule is therefore given as:

$$p_{ij}^{k} = \begin{cases} 1 & \text{if } q < q_0 \text{ and } j = \underset{l \in \mathcal{N}_i^k}{\operatorname{argmax}}\ \tau_{il}\eta_{il}^{\beta} \\ 0 & \text{if } q < q_0 \text{ and } j \neq \underset{l \in \mathcal{N}_i^k}{\operatorname{argmax}}\ \tau_{il}\eta_{il}^{\beta} \\ \frac{\tau_{il}\eta_{il}^{\beta}}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}\eta_{il}^{\beta}} & \text{if } q \geq q_0 \end{cases} \tag{2.8}$$

This allows the system to select the best edges more often. Fine-tuning q_0 allows more exploration of other paths by the ant.

Secondly, it has two pheromone update rules: a global update rule and a local update rule. The global update rule is followed only by the *best-so-far* ant and is used to deposit additional pheromone only on the edges that constitute the tour of the *best-so-far* ant. Further, the pheromone deposited is weighted by the evaporation rate leading to a weighted average of the new and old pheromone values. The global update rule is defined as:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_{ij}^{bs} \tag{2.9}$$

Thirdly, the ACS defines a local update rule that reduces the pheromone deposit on an edge that an ant crosses as it constructs the tour. This is done to encourage more exploration of new paths. The local update rule is given below:

$$\tau_{ij} = (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{2.10}$$

where τ_0 is the initial pheromone on all the edges and $\xi, 0 < \xi < 1$ is a parameter that controls the relative importance of current versus the initial pheromone on an edge.

2.2.2.5 $\mathcal{MAX} - \mathcal{MIN}$ Ant System

Another very popular variant of the ACO algorithms is the $\mathcal{MAX} - \mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [96]. As in the ACS, only the best ants deposit pheromone on the edges that are part of their tours. Here, the best ant may be the best of an iteration or *best-so-far* as defined in other variants. If the same edges appear in the tours of many ants, the pheromone deposit on these edges may increase without limit. This may result in loss of exploration and the solution returned may be a local minimum. Hence, to encourage exploration, the $\mathcal{MM}$AS limits the maximum value of pheromone, τ_{max} on any node. Since any edge that is never used will drop down to a pheromone value of 0 as the pheromone evaporates, $\mathcal{MM}$AS also sets a minimum on the pheromone value, τ_{min}. This ensures that every edge has a non-zero probability of being selected for exploration by an ant. The edges are initialized with the maximum pheromone in the beginning and re-initialized if there is no improvement in the solution over a specified number of consecutive iterations.

2.2.2.6 Other Enhancements to the ACO Algorithm

ACO algorithms have been enhanced by combining them with local search [94] or with other metaheuristic techniques like the tabu search [31, 59], tournament selection of genetic algorithms [53] or using multi-pheromone based solutions [1, 98]. A pre-processing step has also been used to improve the solution and reduce the time taken [92].

In [94], Solnon and Fenet study ACO for solving the maximum clique problem. They use the $\mathcal{MM}$AS system with pheromone deposit on both edges and nodes to study which is the better option. They found that in most of the instances, adding a local search component to the edge based pheromone resulted in improving the average solution in many instances. The local search component used was the (2, 1)-exchange procedure where, given certain properties of three vertices v_i, v_j and v_k, it replaces one vertex with the other two, thus increasing the size of the clique. This local search is repeated until no such exchanges are possible.

Solnon, in [92], uses a pre-processing step to enhance both the quality of the solution as well as reduce the time taken to arrive at the solution. In this paper, a set of solutions which are expected to represent the “sample search space” are generated using a local search heuristic. A pre-defined number of “best” solutions from this set are used to initialize the pheromone trails. It is shown that doing so allows the ACO algorithm to achieve the diversification without the expensive operations related to pheromone updation. This helps reduce the time taken while ensuring that it has no detrimental effect on the quality of the solution. In some cases, it is actually seen that this improves the solution.

In [53], the ACO algorithm for computing MDS is enhanced with the use of a tournament selection to select the next node to be added to the dominating set. The tournament selection is used with a certain probability. At other times, it uses the standard state-transition rule. Only a certain number of admissible nodes, w_{size}, participate in the tournament. If the number of eligible nodes is less than w_{size}, all of them participate. A measure is devised to determine the desirability of a node to be added to the dominating set. This is based on the pheromone content of the nodes and the number of additional uncovered nodes that will be covered by selecting a node. The node with the best such measure is treated as the winner of the tournament and added to the dominating set. This, however, seems similar to the ACS where the best node is picked with probability q_0.

In [74], a specialized pheromone trail has been devised for the car-sequencing problem, which is basically the order of the cars on an assembly line. The pheromone is dependent not just on the neighbors in the graph but on multiple neighbors. Thus, the global and local pheromone update rules are modified to take into consideration the pheromone trail on these s_{max} neighbors. In [31], three local search improvement operators are defined with new evaluation functions to reduce the number of colors needed for the graph coloring problem. One of the local improvement operators uses the tabu search method.

Recent advances in hybridization are combining exact algorithms with the ACO algorithms to obtain more optimal solutions. One of the earliest algorithms using this technique of hybridization is the problem of Capacitated Minimum Spanning Tree (CMST) problem [79]. The authors use the solution for the Capacitated Vehicle Routing Problem (CVRP) to form clusters. Within each cluster, they apply a standard local search algorithm that gives the upper bound on the CMST solution. Then, they apply the Prim's algorithm to find the optimal CMST.

The algorithm in [59] combines Prim's, ACO and tabu search for the problem of finding k-minimum spanning tree (k-MST), i.e., a spanning tree that has k or less edges and whose weight is minimal. They observe that the tabu search method is quite good at finding local solutions but poor at exploration whereas ACO is good at exploration but not as good at minimizing in the local context. On the basis of this observation, they combine the two methods to use the strengths of both to obtain a solution better than either method. The three steps in the algorithm are: a) find an initial solution by using Prim's algorithm until a k-subtree is found; b) use tabu search to explore the local search space and construct a k-MST and terminate when the tabu tenure is longer than the allowed maximum; c) apply the ACO to diversify the search. The algorithm is repeated until the limit on the computational time is reached.

2.2.3 Mathematical Morphology

In this section, we describe the basic concepts of mathematical morphology with images represented as sets of points. We also introduce the image analysis tasks of distance transform, medial axis and skeleton.

Recently, Cousty et al. have introduced graph morphological operators [22] that are easily applied to graph problems. These are explained in detail in 6.1.

2.2.3.1 Notion of Lattice

To define the basic operations of mathematical morphology, we need a notion of a complete lattice. A *lattice* $(\mathfrak{L}, \leq)$ is a set $\mathfrak{L}$ with an ordering relationship, $\leq$, on its elements which is reflexive, $\forall x \in \mathfrak{L}, x \leq x$, anti-symmetric, $x \leq y$ and $y \leq x \Rightarrow x = y$ and transitive, $x \leq y$ and $y \leq z \Rightarrow x \leq z$.

Some Properties of Morphological Operations: Many morphological operators preserve the ordering structure of the lattice. Some of the basic properties of morphological operations are [76, 86]:

1. An operator, Φ, is *increasing* if $\forall x, y \in (\mathfrak{L}, \leq), x \leq y \Rightarrow \Phi(x) \leq \Phi(y)$.

2. An operator, ψ is *idempotent* if $\forall x \in (\mathfrak{L}, \leq), \psi(\psi(x)) = \psi(x)$.

3. An operator, γ, is *anti-extensive* if $\forall x \in (\mathfrak{L}, \leq), \gamma(x) \leq x$.
4. An operator, ϕ, is *extensive* if $\forall x \in (\mathfrak{L}, \leq), \phi(x) \geq x$.

Two basic operations of mathematical morphology are *erosion* which is anti-extensive and *dilation* which is extensive.

2.2.3.2 Dilation and Erosion

Given an image, the shape and structure of the image can be probed using smaller sets of different size, shape and orientation. These smaller sets, B, are called **structuring elements (SE)** and are fundamental to mathematical morphology. Dilation and Erosion are typically defined with respect to structuring elements.

The **dilation**, δ, of the given image X by the SE B is:

$$\delta_B(X) = X \oplus B = \{x \mid \hat{B_x} \cap X \neq \phi\} \tag{2.11}$$

where $\hat{B_x} = \{-x \mid x \in B\}$ is the reflection of B.

The **erosion**, ϵ, of X by B is:

$$\epsilon_B(X) = X \ominus B = \{x \mid B_x \subseteq X\} \tag{2.12}$$

A more fundamental operation of morphology is the **Hit-or-Miss Transform** (HMT). Given B, a set of two structuring elements, B_1 and B_2, the HMT is :

$$X \circledast B = (X \ominus B_1) \cap (X^c \ominus B_2) \tag{2.13}$$

where X^c is the complement of X. If X is taken as the foreground and X^c as the background, in HMT, one SE fits completely in the foreground and the other completely in the background.

2.2.3.3 Morphological Filters

The dilation and erosion operators can be composed to define *morphological filters.* A *Morphological filter* is an operator that is *increasing* and *idempotent.* This is unlike the filters in signal processing where a filter can be defined arbitrarily. Two of the most important morphological filters are the operations of *opening* which is anti-extensive and *closing* which is extensive.

Opening is erosion followed by dilation :

$$\gamma_B(X) = X \circ B = (X \ominus B) \oplus B \tag{2.14}$$

Closing is dilation followed by erosion:

$$\phi_B(X) = X \bullet B = (X \oplus B) \ominus B \tag{2.15}$$

2.2.3.4 Distance Transform and Skeleton

Image analysis relating to size and shape uses tranformations such as distance transform, medial axis and skeleton to represent and reconstruct original images. We look at these three operations in more detail in this section. We need the concepts of *ultimate erosion* and *maximal balls* to understand the notions of distance transform, medial axis and skeleton.

Ultimate Erosion operator is defined as iterative erosion until a further erosion either achieves idempotence, i.e., there is no change in the residual image, or results in an empty set.

Maximal balls are defined as balls that have a maximum radius r_{max} such that the ball fits within the given image boundaries. For every point $x \in X$ of an image, we can define the maximal ball at that point $B_{r_{max}}(x)$.

Distance Transform: The *Distance Transform* of an image, X, is defined as the distance of every pixel $x \in X$ to the nearest point $y \in X^c$. The distance of a pixel $x \in X$ from y is $d(x, y)$ with the value being ∞ for all y to which there is no path from x. There are many concepts of distance in images such as the Euclidean distance, the Manhattan distance and the chessboard distance. The distance transform is normally computed for binary images. A simple algorithm to compute distance transform is to iteratively perform erosion using a structuring element and label pixels of the image with an iteration number of erosion that removed them [39].

Skeleton in Discrete Spaces: The skeleton of an image was described by Serra [85] as:

1. If x is a point of the skeleton, $S(X)$, of a set X and $B_{r_{max}}(x)$ is the maximal ball centered at x, then, there does not exist a larger disk centered at any other point in X that also contains $B_{r_{max}}(x)$ and included completely in X.

2. The maximal ball $B_{r_{max}}(x)$ hits the boundary of X at two or more places.

From this, we understand that the *Skeleton* of an image X, as the center points of the maximal balls contained in the image. The Skeleton also has the additional properties of being '*thin*' such that one more erosion will result in an empty set (*ultimate erosion*). It is central to the image X, i.e., it consists of pixels which are equidistant from the boundaries of the image. A skeleton that is equidistant is also termed as the *medial axis* of the image. While there is no requirement that the skeleton of an image be continuous, a medial axis has the property that it is continuous.

The definition of the skeleton above is for continuous spaces and is computationally complex. The requirement of a continuous skeleton for arbitrary shapes cannot even be calculated. When it comes to discrete spaces, the algorithms become simpler through the notion of ultimate erosion and opening [46, 85]. Using these notions, the skeleton of a set A, $S(A)$ is:

$$S(A) = \bigcup_{k=0}^{K} S_k(A) \tag{2.16}$$

where

$$S_k(A) = \bigcup_{k=0}^{K} \{(A \ominus kB) \ \setminus \ [(A \ominus B) \circ B]\} \tag{2.17}$$

B is the structuring element and kB represents k successive erosions of set A with B. K represents the ultimate erosion iteration value.

The reconstruction of the set A using the skeleton is given by:

$$A = \bigcup_{k=0}^{K} (S_k(A) \oplus kB) \tag{2.18}$$

$S_k(A) \oplus B$ represents dilation with the structuring element B.

Since the image can be reconstructed given the skeleton and the structuring element, it is applied to image coding and compression. Storing the distance of each skeleton point from the boundary of the image can be used as a representation of the shape of the image.

2.3 Related Work

The time you enjoy wasting is not wasted time.
Bertrand Russell

Most of the recent work on MDS and its generalizations focuses on exact, approximation, distributed approximation and Polynomial Time Approximation Schemes (PTAS). Tremendous interest from the theoretical community has seen the development of many exact algorithms that reduce the complexity to less than $O(2^n)$, the brute-force algorithmic complexity for computing dominating sets. Some of the exact algorithms proposed for dominating set problems are for Minimum Independent Dominating Set [12, 41, 64], Minimum Dominating Set for general graphs in [99] and for certain classes of graphs in [40] and for Minimum Capacitated Dominating Set [23].

This section reviews some of the latest approximation, distributed approximation and PTAS algorithms. The few metaheuristic algorithms in literature are reviewed in the chapter corresponding to that particular problem.

2.3.1 Approximation and Distributed Approximation Algorithms for Dominating Sets

There have been many approximation algorithms proposed for different dominating set problems [5, 58] (for capacitated domination), [3] (for independent domination), [24, 101, 106] (for weighted domination). The greedy algorithm for computing Minimum Dominating Set is also proven to be the optimal approximation algorithm [102] and is described in detail in Section 4.1. Distributed approximation algorithms, that have minimal communication complexity, have been proposed for various dominating set problems, [60, 61, 84]. In fact, the distributed algorithm in [84] is proven to have optimal communication complexity of $O(log^*n)$ communication rounds to compute the Maximal Independent Set, where n is the number of nodes in the graph.

A Polynomial-Time Approximation Scheme (PTAS) has been proposed for the first time for the Minimum Dominating Set by Nieberg and Hurink [77] for polynomially bounded growth graphs such as Unit Disk Graphs (UDGs), quasi disk graphs and so on. This has later been adapted for the Minimum Independent Dominating Set (MIDS) problem [56], Minimum Weight Dominating Set (MWDS) problem for UDGs with smooth weights [105] and the Minimum Connected Dominating Set (MCDS) problem [43].

In the rest of this section, we will describe some of these schemes in more detail. The algorithms which have been implemented and compared against other heuristics and metaheuristics are described in the respective chapters later in this thesis.

2.3.1.1 Approximation Algorithm for Minimum Independent Dominating Set

An approximation algorithm for Minimum Independent Dominating Set (MIDS) is defined for bounded degree graphs in [3]. We call this algorithm *Alimonti-MIDS* in the rest of this section. A graph is said to have a bounded degree B, if every node in the graph has at most degree B. If every node in the graph has degree exactly B, it is called a B-regular graph. A 3-regular graph is called a *cubic graph* and a graph which has nodes with degree 3 or less is called an *at-most cubic graph*.

Results of *Alimonti-MIDS* algorithm: The approximation ratios achieved by the algorithms in this paper are 1.923 for cubic graphs, 2 for at-most cubic and 4-regular graphs, $\frac{(B^2-2B+2)(B+1)}{B^2+1}$ for B-regular graphs where $B \geq 5$ and $\frac{(B^2-B+1)(B+1)}{B^2+1}$ for graphs of bounded degree $B \geq 4$.

Algorithm Description: The algorithm for at-most cubic graphs, which we call *Alimonti-cubic-MIDS*, starts off by selecting a node with degree 3, adding it to MIDS and removing its neighborhood from the graph. Then, it selects the next node with degree 3 and so on. When there are no degree 3 nodes left, the algorithm computes the optimal MIDS for the residual graph. MIDS for a graph with at most degree 2 can be found in polynomial time. It is seen, sometimes, that when a node with degree 3 is added into MIDS, it can leave a lot of isolated nodes. This is because the neighbors of the 3-degree node may all have neighbors which are not connected to any other node as shown in Fig. 2.15. All such nodes become isolated and have to be added to MIDS. This increases the cardinality of MIDS. To overcome this issue, the *Alimonti-cubic-MIDS* algorithm applies a local search to check whether replacing a node with its independent neighborhood actually reduces the cardinality of MIDS. If it does, it removes the node and adds its independent neighborhood into the MIDS. After this

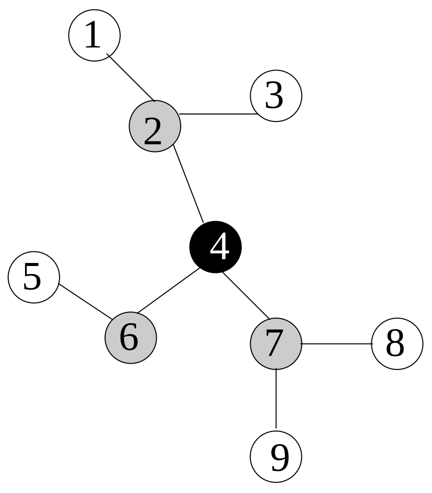


Figure 2.15: Example of an at-most cubic graph illustrating that the selection of the 3-degree node may result in many isolated nodes that need to be added to MIDS, increasing the cardinality. In the figure, choosing the black node to be a dominating node, results in the non-shaded nodes becoming isolated.

swapping step is performed, the graph has at most degree 2. The optimal MIDS of this graph is computed and its union with the previously computed MIDS gives the final MIDS of the at-most cubic graph.

The algorithm for a general bounded degree graph of degree B follows a similar procedure. All nodes with degree B are added to the dominating set as in the algorithm for at-most cubic graphs. If, after removing the dominating nodes and their neighborhood, the resultant graph is an at-most cubic graph, then the *Alimonti-cubic-MIDS* algorithm described earlier is directly applied. Otherwise, the graph is a bounded degree graph with degree at most $B - 1$. The algorithm iterates by choosing nodes with degree $B - 1$ and so on until the residual graph is an at-most cubic graph. The union of the dominating sets found in all iterations forms the final dominating set of the original bounded degree graph.

2.3.1.2 Distributed Algorithm for Maximal Independent Set

The distributed algorithm for Maximal Independent Set (MIS) plays a very important role in distributed algorithms for all generalizations of dominating sets. Algorithms for distributed capacitated dominating set [60], distributed connected dominating set [43] etc. construct an MIS as part of the algorithm and the communication complexity of these algorithms is limited by that of MIS. If an efficient MIS algorithm is used, all of these algorithms become efficient. We describe the distributed algorithm for constructing MIS in [61] in the rest of this section. We call this algorithm *Kuhn-Dist-MIS*.

Kuhn-Dist-MIS algorithm has three phases: the **first phase** consists of constructing a r-hop independent set, which is called a sparse independent set. For a polynomially bounded growth graph, $r = \log \Delta$, where Δ represents the maximum degree of the graph. The steps in the first phase are as follows:

1. Set state of all nodes to active.
2. Each node, v, selects an active neighbor with the minimum ID, u, represented by $d(v) = u$.
3. Similarly, for all neighboring nodes w for which $d(w) = v$, the node v selects the node with the minimum ID, called $p(v) = w'$.
4. If a node v does not have $d(v)$ or $p(v)$, it becomes passive.
5. A subgraph induced by v, $d(v)$ and $p(v)$ is constructed with edges from v to $d(v)$ and $p(v)$. This graph, therefore, has at most 2 edges and algorithms that return the MIS, I, of such graphs in $O(log^*n)$ rounds are used to calculate it.
6. Any node that does not belong to I becomes passive.
7. Any node that has no active neighbors joins the MIS being constructed, S.
8. Repeat steps 2-7 until there are no active nodes left.

In the paper, the authors prove that running this algorithm for $O(\log \Delta)$ iterations results in an independent set where every node is at most $O(\log \Delta)$ hops away from a node in the independent set for polynomially bounded growth graphs.

In the **second phase**, the set, S, constructed in the first phase, is reduced to a 3-hop independent set. This is done as follows:

1. Repeat until S is a 3-hop MIS

    (a) Each node in S adds nodes from its 4-hop neighborhood, $\hat{S}_u$, such that every node in its 3-hop neighborhood has a neighbor in $S \cup \hat{S}_u$ without violating the independence property of S.

    (b) The graph induced by the union of all $\hat{S}_u$, $\mathcal{G}$, is constructed.

    (c) Since each node u adds nodes independent of other nodes, the union may not preserve the independence property. Hence, the MIS of the graph $\mathcal{G}$ is calculated and this is added to S to give the final 3-hop MIS, i.e., $S = MIS(\mathcal{G}) \cup S$.

The **third phase** consists of constructing the cluster graph induced by the clusters formed by the nodes of S. This graph, $\mathcal{G}_{S'}$ is constructed by having edges between the nodes of S if the nodes covered by them are adjacent to each other.

1. The nodes in the cluster graph are now colored. Each node in the original graph v is colored using its cluster's color.

2. A lexicographic ordering is defined based on an ordering of the colors and for nodes of the same color, based on the ID of the nodes.

3. Any node which has the smallest lexicographic order can join the MIS and sends a JOIN message. On reception of this message, all its neighbors simply mark themselves COVERED.

4. A node which received a COVERED message from all neighbors with a lower lexicographic order can declare itself part of MIS without waiting for other nodes' messages.

The entire algorithm composed of all three phases is proven by the authors to take a total of $O(\log \Delta \cdot \log^* n)$ time to compute a maximal independent set.

2.3.1.3 Approximation Algorithms for Capacitated Dominating Set

Bar-Ilan et al. [5] present many approximation algorithms for the problems of network center allocation such that the allocation does not exceed a uniform capacity L on the nodes. The centers are nothing but the dominating nodes. In particular, they consider two types of center allocation problems with load balancing: first, where the number of centers is fixed, say k, and the problem is to minimize the maximum distance between any center and an assigned node. The second problem is to fix a bound ρ on the maximum distance of a node from a dominating node and minimizing the number of centers, given the capacity. The latter is called the ρ-dominating set by Bar-Ilan et al. A greedy algorithm is presented by them for this problem where the capacity on all the nodes is uniform. They prove that this is an approximation algorithm with a ratio of $\lceil ln\ N \rceil$ where N is the total number of nodes in the graph.

Approximation Algorithm for ρ-dominating set: The algorithm constructs a flow graph with dominating nodes forming one partition and the rest of the nodes, called *unassigned* nodes, in the second partition. Now, two new vertices s and t are added to the graph. s is connected to every dominating node and t to the rest of the nodes. If the uniform capacity of each node is c, the edge from s is given a weight of $c - 1$ and every edge from dominating nodes to other nodes as well as every edge coming into t have a weight of 1. This is, in essence, a bipartite graph.

In each iteration, an unassigned node, v is chosen and added to the dominating partition. The integral max-flow algorithm is run on the flow graph. Every node w, whose flow $f(u, w) > 0$, chooses u as its dominating node. The max-flow algorithm returns the maximum matching in a bipartite graph. Hence, the unassigned nodes, $X(C)$, represent the minimum number of nodes that will remain unassigned if this node v is added to the dominating set. This is repeated for every unassigned node $v \in V \setminus C$. The node, v_{min}, that results in the least number of unassigned members is added to the dominating set. This whole process is then iterated until there are no more unassigned nodes in the graph ($X(C) = 0$). As stated earlier, this algorithm is proven to return a cardinality not more than $OPT \times \ln N$ where OPT is the cardinality of the optimal solution. This is given in Algorithm 2.

We now analyze the computational complexity of the algorithm in the worst case. In the worst case, in the *while* loop, only one neighbor is assigned to the new dominating node. Thus, in every iteration, $X(C) := X(C) - 2$. This gives a total of $O(V)$ iterations. Since in each iteration, we loop for every $v \in V \setminus C$, this has a complexity of $O(V)$. For each iteration in the *for* loop, we construct the max-flow algorithm whose complexity is given as $O(V^3)$ or $O(V^2E)$ or $O(VE^2)$ depending on the algorithm used. If $E = V^2$,

Algorithm 2: ***Algorithm ρ-dominating set($G = (V, E), \rho, c$)***

```
$C := \phi$
$X(C) := |V|$
while $X(C) > 0$ do
  $U_{min} := |V \setminus C|$
  $v_{min} := -1$
  for all $v \in V \setminus C$ do
    $C := C \cup v$
    $G' = ConstructFlowGraph(C, V \setminus C, \rho, c - 1, 1, 1)$
    $MaxFlow(G')$
    $U$ := unassigned nodes after max-flow assigns nodes
    if $|U| < U_{min}$ then
      $U_{min} := |U|$
      $v_{min} := v$
    end if
    $C := C \setminus v$
  end for
  $C := C \cup v_{min}$
  $X(C) := U_{min}$
end while
```

the best case algorithm is $O(V^3)$. We assume a complexity of $O(V^3)$. Thus, the worst case complexity of this algorithm comes to $O(V^5)$.

Distributed Approximation Algorithm: Distributed approximation algorithms have been presented by Kuhn and Moscibroda [60] for the same problem, where they prove that, in general graphs, even with uniform capacity, the problem is non-local in nature. They present a local approximation algorithm for the case where the capacity may be violated by some parameter $\epsilon > 0$. They also present a constant factor distributed approximation algorithm for the special case of Bounded-Independence Graphs (BIGs) with uniform capacity. This is done in two phases: in the first phase, a Maximal Independent Set (MIS) is found. It is assumed that this MIS algorithm returns clusters of the form (u_i, C_i) where u_i is the center and C_i is the cluster associated with it, i.e., it is the set of all neighbors of u_i which have been assigned to this cluster. Since, in a capacitated dominating set, we also need to ensure that capacity constraints are not violated, for every u_i whose capacity is violated, the algorithm executes the second phase. The second phase of the algorithm recursively computes an MIS on the subgraph induced by this cluster such that a capacitated dominating set of size at most $O(\mid C_i \mid /cap)$ is computed.

2.3.2 PTAS for Minimum Dominating Set and other generalizations

Nieberg and Hurink proposed a PTAS for Minimum Dominating Set (MDS) for the first time without the use of any geometric information for polynomially bounded growth graphs [77]. In this algorithm, the graph is divided into subsets of graphs that are 2-separated, i.e., if node n_i belongs to subset i and n_j belongs to subset j, the distance between these nodes $\mid d(n_i, n_j) \mid > 2$. The optimal dominating set, D_i, is calculated for each subset S_i. $\bigcup_{i=1}^{k} D_i$ gives the dominating set of the whole graph.

The efficient division of the graph into subsets is the most important part of the algorithm. This is achieved as follows: the optimal minimum dominating sets of r-hop ($D_r[v]$) and $(r+2)$-hop ($D_{r+2}[v]$) closed neighborhood of a node v are computed. If the cardinality of the dominating sets satisfies the inequality, $\mid D_{r+2}[v] \mid > (1+\epsilon) \cdot \mid D_r[v] \mid$, the neighborhood is expanded. The expansion stops whenever this inequality is violated and the subset of the graph represented by the closed $(r+2)$-hop neighborhood of v is removed from the graph.

Given a graph $G = (V, E)$, the algorithm works as follows:

1. Set $D = \phi$
2. Choose a vertex $v \in V$.
3. The radius of the neighborhood under consideration $r = 0$.
4. Repeat until $V = \phi$
   (a) While $\mid D_{r+2}[v] \mid > (1+\epsilon) \cdot \mid D_r[v] \mid$, increment r. $\mid D_r[v] \mid$ and $\mid D_{r+2}[v] \mid$ represent the cardinality of the optimal dominating set of the r and $(r+2)$-hop neighborhood of node v.
   (b) $D = D \cup D_{r+2}[v]$
   (c) Remove the $(r+2)$-hop neighborhood of (v) from the graph.

Note that the dominating nodes of a r-hop neighborhood of a node v can be from the $r+1$-hop neighborhood, i.e., $D_r[v] \subset N_{r+1}[v]$.

The algorithm works in polynomial time as the time taken to compute the optimal solution of r-hop neighborhood of a node v is given by $n^{O(p(r))}$, where $p(r)$ is a polynomial in r and the property of bounded growth graphs such as unit disk graphs.

The PTAS for MIDS is very similar except that it has an additional step added to preserve the independence property. This is described in more detail in Chapter 3.

PTAS for Minimum Connected Dominating Set: The algorithm for Minimum Connected Dominating Set (MCDS) [43] is, once again, quite similar to the PTAS for MDS. In this, however, the halting condition for expansion of neighborhood is given by $\mid MaxIS(N[N_r[V]] \setminus N_r[v]) \mid \leq \epsilon \cdot \mid MaxIS(N_r[v]) \mid$, where $MaxIS$ is the *Maximum Independent Set*. Whenever this condition becomes true, the MCDS of the $(r+4)$-hop neighborhood of v is calculated and added to the MCDS of the graph, D, being computed. Then, the $(r+2)$-hop neighborhood is removed from the graph. Unlike in MDS, in this algorithm, only a subset of neighborhood is removed so that the MCDS computed is on overlapping areas which ensures the *connected* property of the dominating set.

A distributed algorithm is also presented in this paper for MCDS that initially constructs the Maximal Independent Set of the graph to divide the graph into clusters. The cluster graph induced by these clusters is constructed and colored. For each unique color, the centralized algorithm is run in parallel on all clusters of that color. The algorithm is iterated for all the colors in the cluster graph. The union of the MCDS of all these clusters gives the final MCDS of the graph.

PTAS for Minimum Weighted Dominating Set with Smooth Weights: Given a graph $G = (V, E)$, where the nodes have weights $w(v)$ associated with each node v. The weights are said to be smooth if the ratio of the weights of adjacent nodes never exceeds a given constant C, i.e., $max_{(u,v)\in E} \frac{w(u)}{w(v)} \leq C$. In [105], the algorithm begins by choosing a node $u \in V$ such that $w(u) > w(v), \forall v \in V$. The stopping criterion for the expansion of neighborhood is $w(D(N_{r+2}[u])) \leq (1+\epsilon) \cdot w(D(N_r[u]))$. At this point, the weight of the dominating set W is updated as $W = W + w(D(N_{r+2}[u]))$ and the neighborhood $N_{r+2}[u]$ is removed from the graph. This is repeated until there are no nodes left in the graph.

2.3.3 Summary

In the recent times, there have been many attempts to reduce the computational complexity of the exact algorithms for dominating sets of minimum cardinality. There have also been many approximation and PTAS algorithms proposed for different generalizations of minimum dominating set. While these are of great theoretical interest, many of these algorithms are difficult to implement in practice. No empirical study had ever been done to verify how practical a PTAS is or the quality of the solution returned by it, as compared to the heuristics. Our first step was to implement and experiment with algorithms for one dominating set problem to understand the practical issues. In the next chapter, we describe the various algorithms studied, the experimental results and conclusions drawn from our empirical study of the Minimum Independent Dominating Set (MIDS) problem.

Chapter 3

An Empirical Study of Algorithms for MIDS

I never guess. It is a capital mistake to theorize before one has data.
Sir Arthur Conan Doyle

A survey of the clustering algorithms for wireless networks [15, 33] shows that the initial clustering schemes for wireless networks were primarily independent dominating set (IDS) schemes such as in [6, 37, 63] etc. There is a renewed interest in IDS based schemes in the recent times, in the context of wireless sensor networks (WSNs) [80] and wireless sensor and actor networks (WSANs) [69]. These are useful in determining where to place actors or design topologies that lead to energy-efficiency.

A clustering scheme that leads to a minimum independent dominating set (MIDS) would minimize the number of clusters such that the cluster heads do not interfere with each other. It is proven that computing a MIDS is an NP-hard problem [75], including on Unit Disk Graphs (UDGs) [17]. It is also proven that it is very hard to approximate it for general graphs [48]. Exact algorithms that improve the running time for MIDS are still being proposed in literature [12, 41, 64]. So far, only one PTAS has been proposed for computing MIDS [56] and that too for polynomially bounded growth graphs such as unit disk graphs, quasi-disk graphs and coverage-area graphs. An approximation algorithm for MIDS was discussed in Section 2.3.1.1.

A simple result on approximation for UDGs is that, any maximal independent set is a constant approximation of MIDS [67]. This is easily derived since the maximum number of mutually independent nodes in the neighborhood of a node is 5 for a UDG.

If, each node v of the optimal MIDS, $\overline{D}$, is replaced by the maximum independent set of its neighborhood, the resultant dominating set $D = 5 \times \overline{D}$.

While a lot of IDS-based schemes have been proposed, no attempt has been made by anyone so far to determine the quality of the solution returned by the heuristics. To the best of our knowledge, we are also the first to implement the exact algorithms for MIDS and compare the actual run time of the two algorithms. We also compare the cardinality of MIDS and the time taken to compute it by the PTAS with that of heuristics. This study has led us to gain valuable insights into these algorithms and their validity for real-life situations. For example, we studied the intelligent enumeration algorithm and the algorithm using graph matching proposed in [64] for computing the exact MIDS. While the asymptotic complexity of the enumeration algorithm may be worse than that of [64], in practice, it works better for small graphs. So, we use the intelligent enumeration exact algorithm for computing the local exact MIDS required as part of PTAS implementation.

The rest of this chapter is organized as follows: we review three of the more important and general heuristics for MIDS in Section 3.1, the exact algorithm proposed in [64] in Section 3.2, the PTAS for MIDS in Section 3.3, comparison of the exact algorithms in Section 3.4, comparison of PTAS and heuristics in Section 3.5 and conclude with some observations and recommendations in Section 3.6.

3.1 IDS based Clustering Schemes

The lowest ID clustering algorithm for IDS, described in [63] works as follows: every node in the graph is aware of all its 1-hop neighbors. A node which has the lowest ID amongst all its 1-hop neighbors declares itself as a dominating node. All its neighbors then join this cluster and become dominated. If a node finds that its lowest ID neighbor has actually joined another cluster as a member and it has the lowest ID amongst the rest of the neighbors, it declares itself as a dominating node.

In k-hop clustering [37], as the name indicates, the nodes of a cluster are at most k hops away from their dominating node rather than one hop away. In k-hop clustering, the lowest ID algorithm is generalized to k hops. They also propose an algorithm based on the connectivity of a node. The node with higher connectivity or, in other words, degree, in its k-hop neighborhood becomes the dominating node. If two neighbors have the same degree, the lowest ID node becomes the dominating node.

In [69], the authors propose positioning of mobile actors of a wireless sensor and actor network (WSAN) such that the number of sensors within the range of an actor is maximized. The paper proposes a heuristic that returns a k-hop IDS. The dominating nodes of the IDS represent the positions where the actors must be placed to achieve maximum coverage. A node is called a border node if it is k hops away from a dominator node. A node is chosen as a dominator in the following way: every node computes a suitability value, S_i, based on its degree and its distance to the nearest border node of another dominator. This suitability value is then compared to a random value, R. If $R < S_i$, then, the node becomes a dominator and advertises this fact to its k-hop neighbors, which declare themselves as dominated.

3.2 Exact Algorithm for Computing MIDS

We implemented the exact MIDS algorithm proposed by Liu and Song [64]. In this algorithm, a maximal matching, M, has to be found in the given graph. $V_m \subseteq V$ is the set of nodes that are covered by M. The set $I = V \setminus V_m$ is an independent set. The minimum independent dominating set, $\overline{D}$, satisfies the relationship $\overline{D} \cap I = I \setminus N(\overline{D} \cap V_m)$. We can obtain a candidate for an independent dominating set by using the formula $D_s = S \cup (I \setminus N(S))$, where $S \subseteq V_m$ is a subset enumeration of nodes in V_m. We enumerate all possible subsets of V_m. The D_s, which is an IDS with the minimum cardinality, is the MIDS. This is shown in Algorithm 3.

In the worst case, M is a perfect matching and $\mid M \mid = \frac{V}{2}$. When enumerating all possible subsets of V_m, for every edge, $e = (u, v) \in M$, we exclude the combination that contains both nodes of an edge as we are looking for an independent dominating set. Thus, for every edge, $e \in M$, there are, at most, 3 possible enumerations. Therefore, the running time of this algorithm, in the worst case, is $3^{\frac{|V|}{2}}$.

3.3 PTAS for MIDS

There is, so far, only one PTAS proposed for finding MIDS [56]. The primary idea behind this scheme is to divide the given graph into sub-graphs which are separated by

Algorithm 3: ***Exact-MIDS(*** $G = (V, E), \overline{D}$***)***

```
$\overline{D} := \phi$
$| \overline{D} | := | V |$
Find a maximal matching $M$ of $G$
$V_m \subseteq V$ is the set of nodes covered by $M$
$I := V \setminus V_m$
for all enumerated $S \subseteq V_m$ do
    $D_s := S \cup (I \setminus N(S))$ where $N(S)$ is the open neighborhood of $S$
    if ($Independent(D_s)$ and $Dominating(D_s)$ and $(| \overline{D} | > | D_s |)$) then
        $| \overline{D} | := | D_s |$
        $\overline{D} := D_s$
    end if
end for
```

at least 2 hops and find locally optimal solutions that satisfy a bound on the cardinality. The algorithm finds a local MIDS which satisfies the following property :$| D_{i+3} | \leq (1+\epsilon) \cdot | D_i |$, where i is initialized to 0. D_i represents the exact MIDS of the closed i-hop neighborhood, $N_i[v]$, of a node v and D_{i+3} represents the exact MIDS of the closed $(i+3)$-hop neighborhood, $N_{i+3}[v]$, of the given node, v. These are the locally optimal dominating sets. The nodes that dominate $N_i[v]$ can be from the closed neighborhood of the set, i.e., $N[N_i[v]]$. As long as the bound on the cardinality of the dominating sets, is not satisfied, the neighborhood is expanded by incrementing i. When the inequality, given above, is satisfied, the neighborhood $N_{i+3}[v]$ is removed from the graph and the algorithm is repeated on the rest of the graph. The union of $D_{i+3}(v)$ of each iteration constitutes the dominating set of the graph. Since the dominating nodes can be from the neighborhood of the subgraph for which the dominating set is computed, the final union of the local dominating sets may violate the independence property of the dominating set. The second phase of the algorithm repairs the independence property of the dominating set. This yields the global MIDS such that it is at most $(1+\epsilon)$ times the optimal solution.

3.4 Comparison of Exact Algorithms for Computing MIDS

Since the PTAS algorithm requires computing the exact MIDS of the local neighborhood of nodes, we explored the literature for exact exponential algorithms that compute MIDS. We implemented their scheme [64] as well as an intelligent subset enumeration algorithm. We found that the latter works well in practice for small graphs.

The underlying principle of the intelligent enumeration algorithm is efficient subset enumeration as given in [65] where all possible subsets are enumerated in increasing order of cardinality. It generates a bitmap $B = \{b_0 b_1 ... b_{N-1}\}$ where if bit b_i is set, the i^{th} element of the set is present in the subset and absent otherwise. We use this bitmap to construct a subset of nodes, $\overline{D} \in V$, where $\overline{D}$ is a candidate for an IDS. If the subset so constructed is, indeed, an IDS, the algorithm terminates since the first such set found will be the IDS with minimum cardinality. In fact, this algorithm can be made more efficient by starting enumeration from the lower bound on MIDS, if the graph has a proven lower bound. The algorithm is given in Algorithm 4.

Algorithm 4: ***Intelligent-Enumeration-MIDS***$(G = (V, E), \overline{D})$

```
for $i = 1$ to $|V|$ do
  $\overline{D} = \phi$
  for all $bitmap$ with $i$ bits set do
    Construct $\overline{D} \subseteq V$ such that node $u_j \in \overline{D}$ if $j^{th}$ bit is set
    if ($Independent(\overline{D})$ and $Dominating(\overline{D})$) then
      return $\overline{D}$
    end if
  end for
end for
```

3.4.1 Experimentation and Analysis

The intelligent enumeration algorithm has the advantage of simplicity of implementation. To examine how it performs as compared to [64], we ran the two algorithms on grid topologies of sizes ranging from 3×3 to 7×7 (we could not get the results of [64] for grid of size 7×7 as it was taking prohibitively long time). We found that the intelligent enumeration algorithm works well in practice for these graphs as shown in the running

Table 3.1: Time taken and Worst-case Enumerations for the Exact Algorithm by Liu and Song and Intelligent Enumeration. i is the Independence Domination Number of the given graph.

Grid Size	i	**Liu and Song**[64]		**Intelligent Enumeration**[65]	
		Time (sec)	$3^{\frac{\lvert V\rvert}{2}}$	**Time (sec)**	$\binom{N}{r}$, $r = \lvert \overline{D} \rvert$
3×3	3	0	140	0	125
4×4	4	0	6560	0	2380
5×5	7	11	920500	1	710930
6×6	10	39225	$3.8742e + 08$	254	$3.87e + 08$
7×7	12	Unknown	$4.892e + 11$	141242	$1.3167e + 11$

time of the two algorithms in Table 3.1. We also show the number of enumerations that need to be tested for the two algorithms in the table. The value $\binom{n}{r}$ given for intelligent enumeration includes only the highest four $\binom{n}{k}$ terms as these overwhelm the other terms. For [64], we show the value $3^{\frac{|V|}{2}}$ because we found that for all grid topologies, the maximal matching includes all nodes for a graph with even number of nodes and one less for graphs with odd number of nodes. Thus, for grid topologies, this algorithm always has the worst-case running time. We found the time taken to run, even for sizes as small as 49 nodes, is quite prohibitive for exact algorithms. So, we did not try for higher grid sizes. While the number of combinations tested seems almost equal for the two algorithms, intelligent enumeration exits as soon as a solution is found. Hence, it does not go through all the enumerations listed. For [64], however, all enumerations have to be tested before we can declare the MIDS. The other reason for the superior performance of intelligent enumeration is as follows: as can be seen from the Algorithms in 3 and 4, while both of them check candidate sets for domination and independence, Algorithm 3 of [64] has to do a lot more computation. Specifically, once it forms the subset S, it has to union it with $\{I \setminus N(S)\}$. If this is an independent and dominating set, it also has to check whether this has a smaller cardinality than the one encountered so far and update, if necessary. At the same time, unless it goes through all $3^{\frac{|V_m|}{2}}$ enumerations, it cannot determine the MIDS. If V_m is a much smaller set of nodes than V, then, the number of enumerations is drastically reduced and the algorithm will run fast. We also found that intelligent enumeration worked better than [64] for a UDG of 50 nodes.

Based on these results, we went ahead with the implementation of the PTAS algorithm by using intelligent enumeration for finding the exact local MIDS. Since we expect that the number of nodes on which the local MIDS needs to be calculated will be small, it makes more sense to use this than the one in [64].

3.5 Comparison of PTAS and Heuristics for Computing MIDS

The most popular clustering schemes in literature that produce an independent dominating set are the lowest ID clustering and the highest degree node clustering. The algorithm proposed in [69] is similar to the heuristic explained in the next section. We implemented all three heuristics to compare the performance with respect to the cardinality returned by them. The results of the centralized algorithms in our implementation are the best case results for the distributed algorithms. If the distributed algorithms do not implement a time out, then, they will arrive at the same solution as the centralized algorithm after $O(n)$ communication rounds, in the worst case.

Algorithm 5: ***Greedy Heuristic for Computing MIDS***$(G = (V, E), \overline{D})$

```
$\overline{D} := \phi$
**while** $V \neq \phi$ **do**
  Pick $v \in V$
  **while** $\exists u$ such that $((v, x) \in E$ **and** $(x, y) \in E$ **and** $(y, u) \in E)$ **and** $u \neq v$
  **and** $(v, u) \notin E$ **do**
    find $u$ with the highest degree
    $\overline{D} := \overline{D} \cup \{v\}$
    $V := V \setminus N[v]$
    $v := u$
  **end while**
  $\overline{D} := \overline{D} \cup \{v\}$
  $V := V \setminus N[v]$
**end while**
```

3.5.1 Inter-Dominator 3-hop Distance Heuristic

In this algorithm, we always pick dominators that are 3-hops apart. This allows the neighbors of both nodes to be disjoint and hence result in maximum coverage with minimum number of dominators. We call this 3HD heuristic in the rest of the chapter. The algorithm can be described as follows: we pick any node $v \in V$ of the graph $G = (V, E)$ and add it to the dominating set $\overline{D}$ and remove the closed neighborhood of v, $N[v]$, from the graph. We, then, find u, a 3-hop neighbor of v with the highest degree and add it to the dominating set. We repeat this algorithm until either V is empty or we cannot find the next candidate node. If no candidate node is found because the current candidate node has no neighbors, we arbitrarily pick any node from the remaining nodes of the graph and repeat the algorithm stated above. The pseudo-code of this scheme is given in Algorithm 5.

3.5.2 Experimental Design

All the programs were written in C and executed on a server having Intel(R) Xeon(TM) CPU 3.00GHz dual processor quad core with 8GB RAM running Linux version 2.6.9-22.ELsmp. We downloaded the unit disk graph topologies of 50 and 100 nodes included in [68]. There are 300 topologies for each graph size. We calculated the average of the cardinality returned by PTAS and the heuristics on all the instances for both 50 and 100 node graphs. We also used the topology generator included in this software to generate 300 instances of graphs with 200-1000 nodes. We used these for comparing cardinaltiy returned by the heuristics.

In addition, we also use the BRITE topology generator [71] to generate general graphs, specifically, the Waxman Router model topologies. Since the PTAS is applicable only to UDGs, we compared only the heuristics for general graphs. We generated graphs with 50, 100, 250, 500, 800 and 1000 nodes, each having 20 instances. The results were averaged over the 20 instances for each heuristic for all graph sizes.

We developed a Grid Graph generator software that generates incomplete grid graphs with the specified number of edges. We used this software to generate grid graphs having 50, 100, 250, 500, 750 and 1000 edges. Each of these had 20 instances generated.

Grid Topologies: The results are averaged over the 20 instances for the heuristics and the PTAS with $\epsilon = 4$. With a smaller ϵ value of 2, the PTAS was taking prohibitively large time for completion. For example, some topologies of 250 nodes did not complete in 4 hours. Hence, we did not run the PTAS with smaller ϵ values on these graph sizes. We present the results of PTAS with $\epsilon = 2$ for 50 and 100 edge graphs.

Unit Disk Graph Topologies: We ran the PTAS along with heuristics on UDG instances of 50 and 100 nodes each. We found that for small graphs, the heuristics return a cardinality similar to or better than the PTAS while taking less time. For larger graphs, we were unable to run the PTAS because it uses exact algorithm until the condition $| D_{i+3} | \leq | D_i |$ is met to stop expanding the neighborhood. We found that the 5-hop or the 6-hop neighborhood of a node has nearly 60 nodes which becomes too large for the exact algorithm to finish in practical time. Hence, for larger graphs of 200-1000 nodes and general graphs, we only compared the heuristics and present those results here.

3.5.3 Discussion of Results

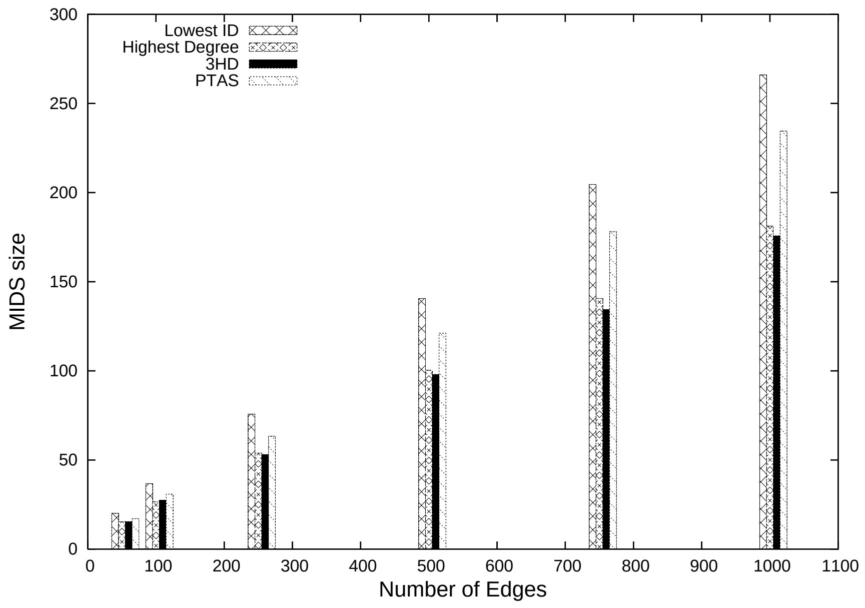


Figure 3.1: Average size of the MIDS returned by PTAS with $\epsilon = 4$, Lowest ID, Highest Degree and 3HD on Grid topologies for each graph size.

Grid Topologies: In these topologies the 3HD heuristic performs slightly better than the highest degree heuristic as observed in figure 3.1. This is because when the highest degree node is chosen in grid topologies, we found that it leaves many other nodes isolated, all of which have to be added to the IDS increasing its cardinality. This was also the observation that was used in the approximation algorithm for MIDS described in Section 2.3.1.1. The lowest ID does not work well at all because it neither separates the dominators nor does it cover as many neighbors as possible when selecting dominators.

The PTAS performance is better than only lowest ID heuristic with $\epsilon = 4$. When we compare the performance of PTAS with $\epsilon = 2$ for 50 and 100 edge grid graphs, we get the average MIDS size of 15 for 50 edge graphs and 29 for 100 edge graphs as compared to 17.05 and 30.85 with $\epsilon = 4$. The time taken with $\epsilon = 4$ is in milliseconds whereas it is $1s$ and $56s$ respectively for 50 and 100 edge graphs with $\epsilon = 2$. Considering that the highest degree and 3HD heuristics return an average cardinality of 15.35 and 15.4 respectively for 50 edges and 26.5 and 27.4 for 100 edges, we can see that the PTAS does not improve upon the heuristics as the graph size increases but takes much more time. Thus, we conclude that PTAS is not practical to use.

Unit Disk Graph Topologies: We ran the PTAS and heuristics for the unit disk graph topologies with 50 and 100 nodes. The cardinality returned by the algorithms is shown in Figure 3.2. We used $\epsilon = 6$ for 100 node graphs as even with $\epsilon = 4$, the PTAS was taking a lot of time to terminate. As seen later in this section, we observed that the quality of the solution is not affected by increasing ϵ, whereas the time taken is reduced quite drastically.

We can observe from figure 3.2 that the highest degree node heuristic performs best amongst all the algorithms studied, including the PTAS. The 3HD heuristic is better than the lowest ID which is the worst heuristic for minimizing cardinality. The 3HD heuristic gives results comparable to PTAS and maximum degree for 50 nodes. The comparison of the time taken by the PTAS and heuristics is given in Figure 3.3. The time taken by the PTAS is approximately $160s$, with $\epsilon = 2$. The time taken by the heuristics, which is in the order of milliseconds. As with grid graphs, the maximum degree heuristic and 3HD heuristic return cardinality better than or comparable to the

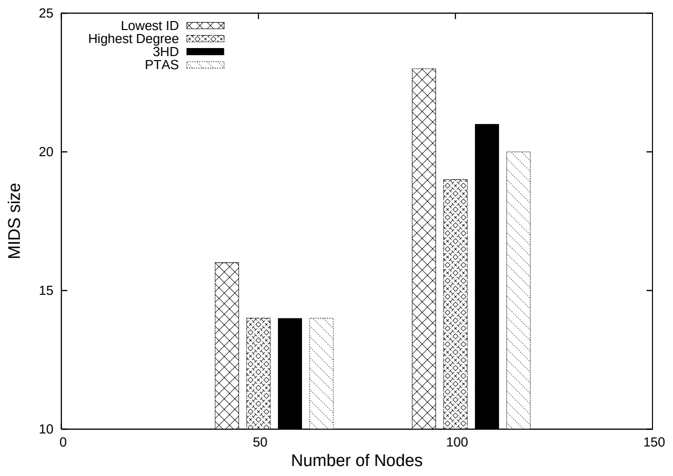


Figure 3.2: Average size of the MIDS of 300 UDG topologies, each of size 50 and 100, returned by PTAS with $\epsilon = 2$ and $\epsilon = 6$ respectively, Lowest ID, Highest Degree and 3HD.

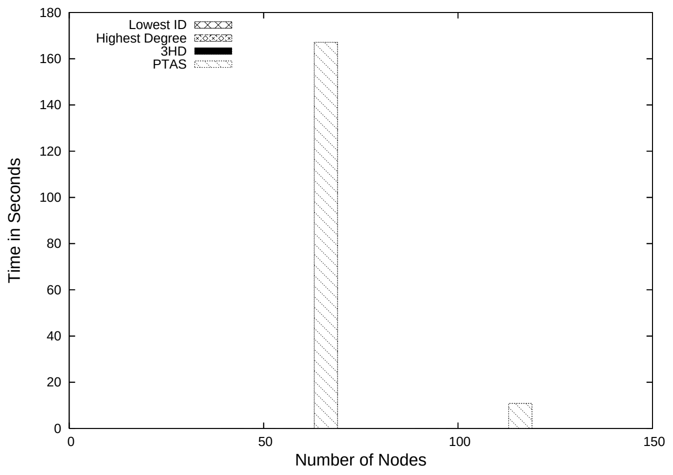


Figure 3.3: Average time taken to compute MIDS returned by PTAS with $\epsilon = 2$ for a UDG of 50 nodes and $\epsilon = 6$ for a UDG of 100 nodes, Lowest ID, Highest Degree and 3HD on 300 topologies for each graph size. **Note:** Heuristics takes an average time in the order of milli/microseconds.

PTAS in much less time. Therefore, we conclude that PTAS is not practical to use with UDGs also.

In figure 3.4, we can, once again, see that the highest degree heuristic is consistently better than 3HD and lowest ID for all graph sizes. As stated earlier, the results are averaged over 300 instances for each graph size. So, we can conclude that using highest degree heuristic is best for minimizing the cardinality of the MIDS. Lowest ID and highest degree heuristics are both highly amenable for a distributed algorithm. The lowest ID heuristic results in a high cardinality and therefore is not recommended for MIDS.

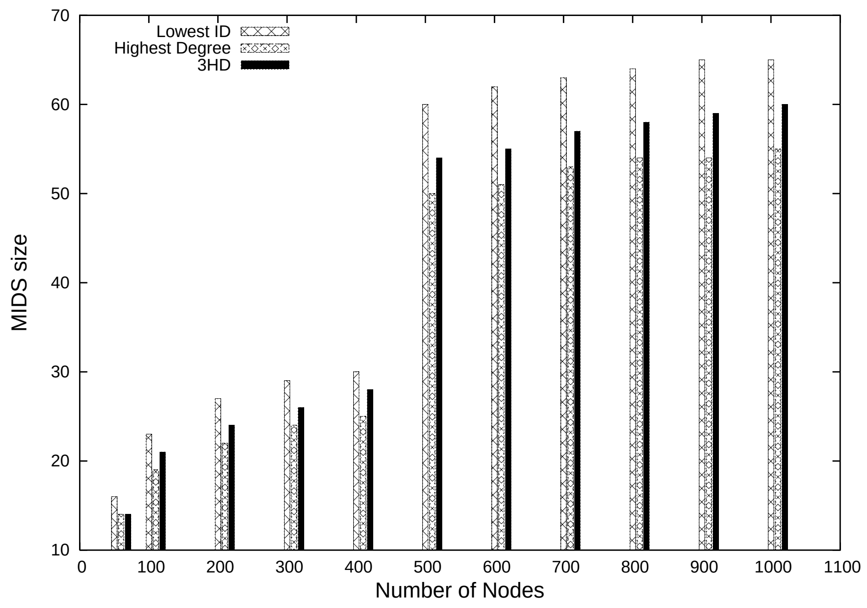


Figure 3.4: Average size of the MIDS returned by Lowest ID, Highest Degree and 3HD on 300 UDG topologies each for graph sizes of 50-1000 nodes.

Waxman Router Model Topologies: In figure 3.5, we present the cardinality of MIDS returned by the three heuristics. We cannot use PTAS for these graphs as these are not polynomially bounded growth graphs. Once again, we see that the highest degree heuristic returns the best cardinality, followed by 3HD heuristic and the worst is lowest ID. In fact, we can observe that the difference in cardinality between these three heuristics is quite large for general graphs such as the Waxman Router model topologies. This is easily explained by the fact that a node with the highest degree covers the maximum number of uncovered nodes. Thus, we need fewer nodes to cover all the nodes in the graph. In 3HD heuristic, we start at some arbitrary point and proceed to choose nodes which are 3 hops away from the previously chosen node. This ensures that we do not choose 2-hop neighbors to ensure maximum independence but does not take into account the degree of the remaining nodes in the graph. Lowest ID does not result in either a hop separation between the dominators nor is it guaranteed to cover the maximum number of nodes amongst the remaining nodes, both of which try to minimize the cardinality. Hence, it performs really badly.

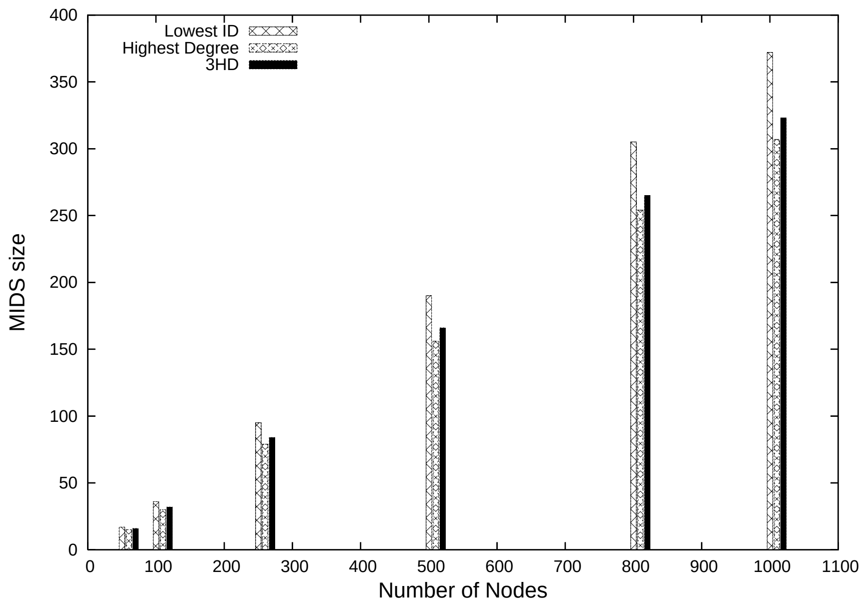


Figure 3.5: Average size of the MIDS returned by Lowest ID, Highest Degree and 3HD on 20 BRITE topologies for each graph size.

3.6 Some Important Observations and Recommendations

Exact Algorithms: The asymptotic complexity of the intelligent enumeration algorithm is, in the worst case, $O(2^n)$ compared to that of [64] which is $O(\sqrt{3}^n)$. However, in practice, for small graphs, the former works better than the latter. For large graphs, neither is practical. Thus, it is better to use intelligent enumeration over other exact algorithms, in practice, when an exact algorithm for small graphs is needed.

PTAS: We found that in many instances there was no change at all in the cardinality of the IDS returned by the PTAS when the value of ϵ is changed from 2 all the way to 20. In fact, surprisingly, we found that in some instances, the cardinality returned for $\epsilon = 6$ is less than that for $\epsilon = 4$. The time taken is significantly reduced when ϵ is increased. This is because the stopping criterion for dividing the graph into 2-separated sub-graphs is more easily reached with a higher value of ϵ. As the number of nodes in a neighborhood increases, the exact solution for the neighborhood takes a long time to complete and so the PTAS also takes a prohibitively long time. In fact, with high density graphs, the PTAS is no longer practical. We recommend the use of PTAS in low degree graphs with higher values of ϵ.

Heuristics: As we can see from the results on various topologies, the maximum degree heuristic performs best except for grid topologies where 3HD performs slightly better. When we look at the distributed solutions based on these heuristics, it is easy to see that lowest ID and highest degree are much more easily adapted to be distributed algorithms compared to 3HD. In fact, the algorithm presented in [69], which is similar to 3HD, will under certain circumstances, result in a dominating set that is not even an IDS. Thus, we recommend that the maximum degree heuristic be used for computing MIDS for general graphs and UDGs and 3HD for grid topologies.

3.6.1 Summary

It can be seen from our study of MIDS that the PTAS is not very practical when it comes to large graphs. We also found that it does not improve upon the best heuristic for the graphs we studied.

Metaheuristic techniques are known to improve upon the results of heuristics and approximation algorithms. Our survey of literature for dominating sets shows that there has been hardly any work in applying metaheuristic techniques to the problem of computing dominating sets ([51, 53] for MDS and [57] for MWDS). This led us to propose metaheuristic algorithms for computing dominating sets of minimum cardinality. These are presented in the next two chapters, 4 and 5.

Chapter 4

Metaheuristic Algorithms for MDS and MWDS

I have never met a man so ignorant that I couldn't learn something from him.
Galileo Galilei

In this chapter, we describe our proposed Hybrid Genetic Algorithm (HGA) and Hybrid Ant-Colony Optimization algorithms. We also describe the optimal approximation algorithm for computing MDS first. We, then, present the hybrid genetic algorithm for MDS of Hedar et al., *Hedar-MDS* [51], which is one of the two metaheuristic algorithms for MDS available in literature. The other is the ACO algorithm by Ho et al. [53] presented in Section 2.2.2.6. We implemented the algorithm *Hedar-MDS* and compare the cardinality returned by our proposed hybrid GA and hybrid ACO algorithms with this scheme and the optimal approximation algorithm in Section 4.7.

We investigated many variations of the heuristic for MWDS, which are given in Section 4.3, to determine which of them performs well. The results of these experiments are presented in 4.8. We review the ACO algorithm for MWDS proposed by Raka Jovanovic et al. [57], which we call *Raka-ACO*, in Section 4.4. Our proposed hybrid metaheuristic algorithms, discussed in Sections 4.5 and 4.6, and the heuristics were run on the same data set used in *Raka-ACO*. The results, presented in Section 4.8, show that our algorithms and in fact, most of the heuristic variations experimented with, are all superior to the *Raka-ACO* algorithm.

4.1 Optimal Approximation Algorithm for MDS

The greedy heuristic [102] for finding a minimum dominating set (MDS) returns a solution at most ($ln\Delta \times Opt$) where Δ is the maximum degree of a node in the graph. It is proven to be the optimal approximate solution unless $P = NP$ [102].

Initially, the nodes are all colored WHITE. A node which is in the dominating set is colored BLACK and all dominated nodes are colored GREY. The algorithm works by selecting a non-BLACK node with the maximum WHITE degree and including it in the dominating set. The node is then colored BLACK and all its neighbors are colored GREY. This is repeated until there are no WHITE nodes in the graph.

4.2 Hedar-MDS: Hybrid Genetic Algorithm for Minimum Dominating Set

Recently, the problem of MDS has been solved using a hybrid genetic algorithm [51]. In this paper, the authors use a generational GA with a linear rank selection algorithm to select parents for the standard one-point crossover and uniform mutation to generate members of the new population.

It uses three procedures to reduce the cardinality of the solution computed.

1. **Filtering:** which basically checks if a node can be removed from the dominating set without affecting the domination property.

2. **Local Search:** tries to randomly add nodes to the dominating set proportional to its degree if the generated population member is not a DS; it deletes a node randomly in inverse proportion to its degree if it is a DS. In both cases, if the fitness value increases, it retains the change to the chromosome. However, it does not repeat this procedure until the generated member is a dominating set or the cardinality is reduced to the minimum possible. Instead, it iterates only for a fixed number of iterations.

3. **EliteInspiration:** constructs the intersection of the three best DS solutions found so far and then tries to minimize the cardinality. This is a variation of the *elitism* strategy.

In our experimentation, we found that in some cases, this algorithm does not even find a DS. We modified the algorithm such that it always returns a DS as output. We changed it as follows: after finding the intersection of the best n_{core} members of the DS population, if the resultant solution is not a DS, we keep adding nodes to it using the greedy heuristic until the member is a DS. We, then, applied *Filtering* to reduce the cardinality of this solution. We found that with these changes, the solution returned by their algorithm is better than a straighforward implementation of their paper. The results presented in this thesis are with these changes incorporated.

4.3 Greedy Heuristics for Minimum Weight Dominating Set

We experimented with four different variations of a greedy heuristic to determine the minimum weight dominating set. Just as in the greedy heuristic for MDS in Section 4.1, initially all nodes are colored WHITE. A node that is included in the dominating set is colored BLACK, whereas all nodes adjacent to the dominating node are colored GREY. We halt the algorithm when there are no WHITE nodes in the graph.

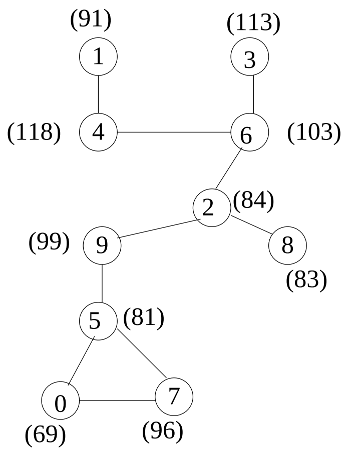


Figure 4.1: Example Graph to illustrate the Heuristics for MWDS.

An example graph is given in Fig. 4.1. Each node is numbered within the circle with its ID and outside the circle with its weight. We use this graph to illustrate the general working of the heuristic. We show how the dominating set is constructed step-by-step using Eqn. 4.1.

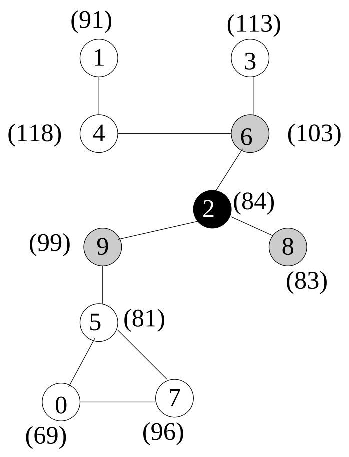


Figure 4.2: Node 2 is selected as the first dominating node as per Eqn. 4.1. It is colored BLACK and all its neighbors GREY.

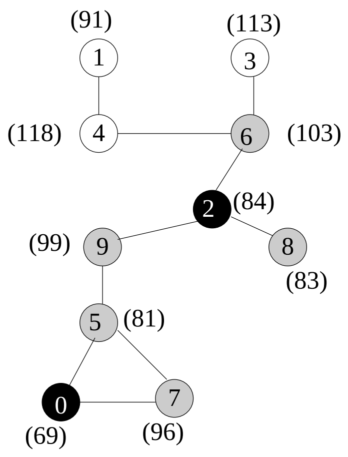


Figure 4.3: Node 0 is selected as the next dominating node as per Eqn. 4.1.

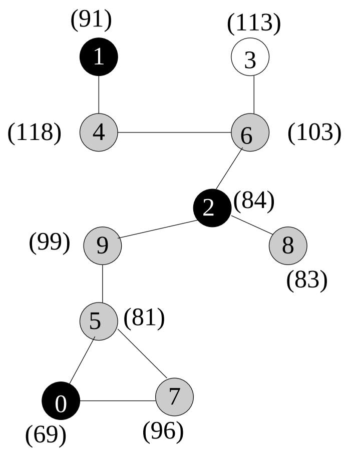


Figure 4.4: Node 1 is selected as the third dominating node as per Eqn. 4.1.

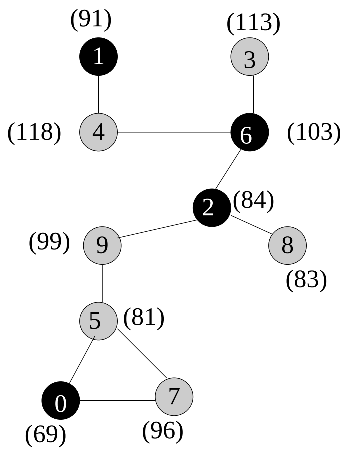


Figure 4.5: Finally, Node 6 is selected as the dominating node as per Eqn. 4.1. Since none of the nodes is WHITE at this point, the algorithm halts.

In the first heuristic, we use an algorithm based on the standard greedy algorithm for the weighted set-cover problem [16]. The dominating set being constructed, D, is initialized to a NULL set in the beginning. The WHITE degree of a node, $d_w(u)$, is the number of WHITE neighbors of a given node u. For computing the minimum weight dominating set, we divide the WHITE degree of the node, $d_w(u)$, by the weight of the node $w(u)$ and add the node with the maximum ratio to the dominating set D. If S represents $V - D$, where D is the set of dominating nodes, then the next node to include in D is selected as follows:

$$v \leftarrow \underset{u \in S}{\operatorname{argmax}} \frac{d_w(u)}{w(u)} \tag{4.1}$$

In the example graph of Fig. 4.1, the node which has the maximum ratio initially is 2 and is colored BLACK. All its adjacent nodes are marked GREY as shown in Fig. 4.2. This process of finding the node with maximum ratio and coloring its neighbors GREY is repeated until there are no WHITE nodes in the graph. For the graph of Fig. 4.1, nodes 0, 1 and 6 are selected in that order after 2. This is illustrated in Figures 4.3-4.5.

In the second heuristic, we divide the sum of the weights of the WHITE neighbors of a node by the weight of the node. The node with the maximum ratio is taken as the node to be included in the dominating set. The weight of the WHITE neighbors of a node u is represented as $W(u) = \sum_{s \in N(u)} w(s)$, where $color(s) = WHITE$.

$$v \leftarrow \underset{u \in S}{\operatorname{argmax}} \frac{W(u)}{w(u)} \tag{4.2}$$

In the third heuristic we calculate the difference of the weight of the WHITE neighbors of a node and its own weight. The node with the maximum difference is chosen as the node to be included in the dominating set.

$$v \leftarrow \underset{u \in S}{\operatorname{argmax}} (W(u) - w(u)) \tag{4.3}$$

In the fourth heuristic, we divide the product of the WHITE degree of a node and the sum of the weights of the WHITE neighbors of a node by the weight of the node. The node with the maximum ratio is taken as the node to be included in the dominating set.

$$v \leftarrow \underset{u \in S}{\operatorname{argmax}} \frac{d_w(u) \times W(u)}{w(u)} \tag{4.4}$$

The first two heuristics differ from the ones presented in [57] in that the latter does not consider nodes colored GREY when choosing a new node to add to the dominating set. This eliminates a large number of promising candidates leading to worse results than our heuristics.

The results of applying these heuristics on the data sets used in [57] are given in Section 4.8. Those data sets consist of two types of graphs, Type I and Type II, which are also described in Section 4.8. We also applied these heuristics on UDG instances we generated using the UDG generator [68]. It can be readily seen from these tables that the first, second and fourth heuristics give similar results. In some cases, one heuristic is better and in others another. The third heuristic is worse than the others in all instances. For Type II instances, the difference in cardinality returned by the third heuristic and the other three is larger.

4.4 Raka-ACO: ACO Algorithm for Minimum Weighted Dominating Set

The only metaheuristic algorithm proposed so far in the literature for the MWDS problem [57] uses an ant-colony system (ACS) algorithm. This algorithm is similar to the solution proposed for minimum weight vertex cover problem by Shyu et al. [89]. The algorithm initially generates a solution which is used to initialize the pheromone values of the nodes in the ACO algorithm. To compute this solution, the given graph is converted into a complete graph with edges having a weight 0, if they are not present in the original graph and 1, otherwise. A value of 1 also represents an uncovered neighbor whereas 0 represents a covered neighbor. Every time a node is added to the MWDS, the weight of all edges incident on it and its neighbors is set to 0, i.e., marking the node and its neighbors as covered. In each iteration, the node with the maximum ratio of the sum of weights of its uncovered neighbors to its weight, η_r, is added to the MWDS.

The state-transition rule determines the probability with which the next node i is chosen by an ant k and is represented by p_i^k. In *Raka-ACO*, it is calculated by the formula $p_i^k = \frac{\tau_i \eta_i^\beta}{\sum_{r \in A_k} \tau_r \eta_r^\beta}$, where A_k represents the set of nodes that are not in the dominating set and η_r^β represents the heuristic component. In addition, if a random variable $q \in (0, 1)$ is greater than a certain threshold q_0, the node with the maximum product of the pheromone and heuristic is always chosen. If $q < q_0$, the nodes are selected based on the probability calculated using the equation for p_i^k. The algorithm uses the global pheromone updating rule of $\tau_i = (1-\omega)\tau_i + \omega\Delta\tau_i$ at the end of one iteration, where ω represents the pheromone evaporation rate and $\Delta\tau_i = \frac{1}{\sum_{j \in D} w(j)}$, D being the best dominating set constructed by an ant in that iteration. For all other nodes, the pheromone is simply evaporated. The algorithm also uses a local pheromone updation rule which is applied after selection of a node i by an ant. The pheromone on node i is updated by $\tau_i = (1-\phi)\tau_i + \phi\tau_0$ where $\phi \in (0,1)$ is a parameter used to adjust the pheromone laid previously on the node and τ_0 is the same as the initial pheromone laid on all the nodes.

4.5 Proposed Hybrid Genetic Algorithm for Dominating Sets

We have used a steady-state genetic algorithm [25] to solve the dominating set problems. It is generally found that the steady-state genetic algorithm is better in such problems as can be seen in the hybrid genetic algorithm for the minimum weight vertex cover problem [90]. We use a similar strategy for the MDS and MWDS problems.

Our hybrid genetic algortihm can be summarised as follows: we start off with an initial population of 100 members generated randomly. To generate a new population member, we do a crossover of two parents which are chosen using the binary tournament selection method. To ensure that there is enough randomness in the population, we generate a random member instead of using crossover with a probability of $1 - p_c$, where p_c is the probability of crossover. We replace the worst member of the previous generation with the new member so generated. If the new member is not a dominating set, we use a greedy heuristic for the specific dominating set problem to repair it. This is done with a probability p_h. In other cases, we repair it by adding nodes randomly until it is a dominating set. Once the new member is a dominating set, we use a minimization heuristic and check for duplication before inserting it into the population. p is a uniform variant in the interval $[0,1]$ in the pseudo code for the genetic algorithm given in Algorithm 6. In the rest of this section, we describe the proposed hybrid genetic algorithm in detail.

Algorithm 6: ***Hybrid Genetic Algorithm for WMDS***

```
Generate Initial Population, $I$
$F$ := fitness of best member of $I$
$b$ := Best member of $I$
while $gen < MAXGEN$ do
  if $(p < p_c)$ then
    Select parents $p_1$ and $p_2$ using binary tournament selection
    $C$ := crossover($p_1$, $p_2$)
    $C$ := mutate($C$)
  else
    Generate $C$ randomly
  end if
  $C$ := Repair($C$)
  $C$ := Minimize($C$)
  if unique($C$) then
    Replace worst member of the population with $C$
    if $f(C) < F$ then
      $F := f(C)$
      $b := C$
    end if
    gen := gen + 1
  end if
end while
return $b$
```

4.5.1 Chromosome Representation

We use the binary representation for the chromosome with a bit vector of length n, where n is the number of nodes in the graph. The bit corresponding to a node is set to 1 if the node is a member of the dominating set and 0 otherwise.

4.5.2 Fitness

Fitness function is the *cardinality* of the dominating set for MDS and the *sum of weights* of all the nodes in the dominating set for MWDS. The aim is to minimize this cardinality for MDS and sum for MWDS.

4.5.3 Crossover

Binary tournament is used to select two parents for crossover where the member with a better fitness is selected with probability p_{better}. We, then, use the fitness-based crossover method proposed by Beasley and Chu [7] to create the child member. Let the parents be p_1 and p_2 and their respective fitness values $f(p_1)$ and $f(p_2)$. The bits in the child are inherited from parent p_1 with probability $\frac{f(p_2)}{f(p_1)+f(p_2)}$ and from parent p_2 with probability $\frac{f(p_1)}{f(p_1)+f(p_2)}$. This ensures that bits are more often inherited from a parent with a better fitness. As stated earlier, crossover is not applied always, but only with probability p_c. In the remaining cases, a new member is generated randomly to diversify the population in order to prevent the algorithm from being trapped in a local minimum.

4.5.4 Mutation

A simple bit flip mutation scheme is applied. With probability p_m, for every bit in the child generated by crossover, if the bit is 0, we flip it to 1 and if it is 1, we flip it to 0.

4.5.5 Repair

We use the repair function whenever we find that a member is not dominating. For MWDS, we use the heuristic of adding nodes with the maximum $\frac{W(u)}{w(u)}$ ratio to the member until it is a dominating set. This ensures that fewer nodes with relatively less weight are added to the dominating set, thus improving the fitness of the member. The corresponding heuristic for MDS is to use a member with the maximum WHITE degree. The heuristic is used with a probability p_h. With probability $(1 - p_h)$, we repair the member by adding nodes randomly to the member. This helps to diversify the population.

The member generated and repaired in this fashion may consist of *redundant nodes* in the final solution. We remove this redundancy using the minimization heuristic which is described in the next section. The pseudo-code of the repair function for MWDS is given in Algorithm 7.

Algorithm 7: ***Repair(C)***

```
$S := V \setminus C$
if ($p < p_h$) then
  while ($C$ is not a dominating set) do
    $max := 0$
    for ($s \in S$) do
      if ($max < \frac{W(s)}{w(s)}$) then
        $max := \frac{W(s)}{w(s)}$
        $v := s$
      end if
    end for
    $C := C \cup v$
    $S := S \setminus v$
  end while
else
  while ($C$ is not a dominating set) do
    $v := random(S)$
    $C := C \cup v$
    $S := S \setminus v$
  end while
end if
return $C$
```

4.5.6 Minimization Heuristic

We first define *redundant nodes*. A redundant node, v, is one whose closed neighborhood, $N[v]$, is dominated by other nodes of the dominating set. In other words, if D is the dominating set and $N[v] \subset N[D \setminus v]$, then v is a redundant node. Let R be the set of such redundant nodes in D. In MWDS, we delete a node which has the maximum ratio of its weight to its degree from the redundant set with probability p_r. This helps to minimize the dominating set as far as possible by taking into consideration the weight contributed by a node against the number of nodes which are affected by its removal. We delete a node randomly from R with probability $1 - p_r$, to increase the diversity of the population. We repeat this operation until there are no more redundant nodes in the member. The minimization function for MWDS is presented in Algorithm 8.

Algorithm 8: ***Minimize(C)***

```
$R := \{v : v \in D \bigwedge N[v] \subset N[D \setminus v]\}$
while $(R \neq \phi)$ do
  if $(p < p_r)$ then
    $v := \underset{u \in R}{\operatorname{argmax}} \frac{w(u)}{d_w(u)}$
  else
    $v := random(R)$
  end if
  $C := C \setminus v$
  recalculate $R$
end while
return $C$
```

The minimization heuristic is same for MDS except that, with probability p_r, the node with the *smallest degree* is removed from the dominating set.

4.5.7 Initial Population Generation

The first member of the initial population is generated using the respective greedy heuristic for MDS or MWDS. Specifically, the heuristic corresponding to Eqn. 4.2 is used for MWDS. The remaining members of the initial population are generated as follows: each node of the graph is included with probability 0.3. The subset, thus generated, is checked for domination. If it is not dominating, the repair procedure is applied to make it dominating. Then, the minimization heuristic is applied to reduce the weight/cardinality of the dominating set. At the end of the minimization, the resultant dominating set is checked for duplication in the population members generated so far. If it is unique, it is added to the population. Otherwise, it is discarded and the procedure is repeated. If a unique member cannot be generated after a given maximum number of trials, the number of members in the initial population is updated to the actual number generated. This needs to be done in small graphs where all possibilities have been exhausted when generating the initial population.

4.6 Proposed Hybrid Ant-Colony Optimization with Local Search for Dominating Sets

Our proposed ACO algorithm is primarily based on the $\mathcal{MAX} - \mathcal{MIN}$ Ant System [96]. It starts off with an initial pheromone value of $\tau_0 = 10.0$ on each node. All the nodes in the graph are colored WHITE initially. An ant performs a random walk of the graph using the state-transition rule to determine which node is to be visited next. Each time a node, u, is selected, it is added to the dominating set and colored BLACK. All the neighbors of this node are colored GREY. The walk stops when there are no WHITE nodes in the graph.

As discussed in Section 2.2.2, two rules define the behavior of ants in the ACO algorithm - the state-transition rule and the pheromone updation rule. The state-transition rule may be based only on the pheromone value [91, 94] or it may be based on both the heuristic and pheromone values [89]. When we experimented using both these methods, we found that neither works really well. We then added the *local search component* of minimization of the dominating set through removal of the *redundant nodes*. This yielded better results with both the methods, though, using the heuristic along with the pheromone made the algorithm slower without improving the quality of the solution. Therefore, we did not use the heuristic in our ACO algorithm. In this respect, our algorithm is similar to the maximum clique ACO algorithm proposed in [94].

In the beginning, each node has an initial pheromone value of τ_0. This pheromone is evaporated with time, based on the pheromone persistence rate, $0 \leq \rho \leq 1$. If the pheromone value falls below a pre-specified threshold, τ_{min}, the value is set to τ_{min}. This is to ensure that there is sufficient exploration of the search space. The state transition rule for an ant in our algorithm is given by:

$$p(v) = \frac{\tau_v}{\sum_{u \in S} \tau_u} \tag{4.5}$$

where $S = V \setminus D$, V is the set of nodes in the graph and D is the dominating set constructed so far. The ant stops its walk once it has constructed a dominating set. The minimization heuristic described in Section 4.5.6 is applied to the dominating set constructed. This is the local search mechanism that is used to reduce the cardinality/weight of the dominating set by removing any redundant nodes from the solution

Algorithm 9: ***ACO-LS***$(G, \overline{D})$

```
Initialize Pheromone trails
$\overline{D} := \phi$
$f := MAXWT$
$F := MAXWT$
if $PP$ then
  ACO_PP$(G, M)$
end if
for $iter = 1$ to $N$ do
  for $i = 1$ to $NumAnts$ do
    $A_i$ = RandomWalk$(G)$
    if fitness$(A_i) < f$ then
      $bestAnt := A_i$
      $f :=$ fitness$(A_i)$
    end if
  end for
  if $f < F$ then
    $F := f$
    $\overline{D} := bestAnt$
  end if
  UpdatePheromone$(G, f, F)$
end for
return $\overline{D}$
```

generated by an ant. At the end of each iteration, after all the ants have generated their solutions, the best solution of that iteration is determined. Then, the pheromone value of every node v that forms part of this solution is updated as per equation 4.6 for MWDS and 4.7 for MDS:

$$\tau_v = \tau_v \rho + \frac{2.0}{5.0 + f - F} \tag{4.6}$$

$$\tau_v = \tau_v \rho + \frac{1.0}{10.0 + f - F} \tag{4.7}$$

where f represents the fitness of the best solution in the given iteration and F represents the fitness of the *best-so-far* ant. For all the nodes that are not part of the best solution, the pheromone value is simply evaporated as follows:

$$\tau_v = \tau_v \rho \tag{4.8}$$

We also experimented by adding a pre-processing step as proposed in [92] to the ACO algorithm for MWDS. In the pre-processing step, we generate $M = 100$ solutions using different maximal independent sets (MIS), as it is well-known that any MIS is a dominating set. For generating an MIS, we select a node in the graph randomly. We add this node to the dominating set, change its color to BLACK and color all its neighbors GREY. We then choose a WHITE node to add to the dominating set and repeat the procedure until there are no more WHITE nodes in the graph. The pheromone value of the nodes that are part of these independent sets is updated using the pheromone updation rule given in equation 4.6. The advantage of using a pre-processing step, in general, is that it reduces time for convergence while retaining the quality of the solution.

We call the proposed hybrid genetic algorithm as HGA, the ACO as ACO-LS and the ACO with pre-processing as ACO-PP-LS. The results for MWDS consist of the weight returned by the various heuristics, HGA, ACO-LS and ACO-LS-PP on the same data set as used in [57]. For MDS, the metaheuristic algorithms are run on unit disk graph instances and Router Waxman model instances generated by us. They are compared against the greedy heuristic and the algorithm presented in [51].

4.7 Experimental Results for MDS

The experiments were done on two different types of graphs - unit disk graphs generated using the UDG topology generator [68] and Waxman Router Topologies [103] using BRITE [71]. The data set consists of 50, 100, 250, 500, 750 and 1000 node graphs. For UDG topologies, we used ranges of 150, 200 and 250 units to study the effect of different degrees of connectivity on the performance of the algorithms. Graphs with nodes 50, 100 and 250 are generated using an area of 1000×1000 units whereas those with nodes 500, 750 and 1000 are generated using an area of 2000×2000 units. In the case of Router Waxman topologies, we used random and heavy-tailed placement [70] which is considered more common for Internet topologies. We varied the degree of connectivity by using $2 \times N$, $4 \times N$ and $8 \times N$ edges for graphs, where N is the number of nodes in the graph. In all cases, the results presented are averaged over 10 instances.

HGA Parameters: HGA has an initial population of 100 members and we ran it for 10,000 generations. This would mean that a total of 10,000 solutions are created as we create one new member of the population in each generation. We use the following values for the various probabilities: when generating random population members both in initial population and a new member in later generations, we use a probability of 0.3 for adding a node into the dominating set. We use the value of $p_c = 0.9$ for crossover, $p_m = 0.02$ for mutation, $p_{better} = 0.8$ to choose a better parent during binary tournament selection, $p_h = 0.2$ for using the heuristic to repair a member. We use random removal of a redundant node with probability $p_r = 0.6$. All of these values were arrived at after extensive experimentation with different values.

ACO Parameters: The hybrid ACO algorithm has 20 ants and is run for 1000 iterations which is a total of 20,000 solutions. We use an initial pheromone value of 10.0 and a minimum threshold on the pheromone value of 0.08. The pheromone persistence rate is $\rho = 0.985$.

Parameters for *Hedar-MDS*: We used 100 generations and 100 members in each generation for a total of 10,000 solutions, as it uses generational GA, the same as in our hybrid GA. The rest of the parameters are as specified in their paper.

4.7.1 Discussion of Results

It can readily be seen from Tables 4.1, 4.2 and 4.3 that the performance of *Hedar-MDs* is worse than even the greedy heuristic in all the graph instances studied. This can be explained by the fact that there is no attempt to force each member of the population to be a dominating set and no attempt to minimize cardinality. The final *EliteInspiration* process seems flawed because in standard elitism, the best n_{core} members are always retained across generations. In their method, they retain them to compute the intersection of the best n_{core} members which is not even guaranteed to be a dominating set.

We observe from Table 4.1 that our hybrid ACO and our HGA perform similarly for most instances in terms of cardinality of the solution. We observe that the time taken by ACO-LS and HGA are similar up to 250 nodes for UDG instances. But, as the number of nodes increases, the ACO-LS algorithm takes twice the time taken by

Table 4.1: Cardinality (γ) of MDS and Time taken in seconds using Heuristic, Hedar-MDS, HGA and ACO-LS for UDG Instances

N	Range	Heuristic	Hedar-MDS		HGA		ACO-LS	
			γ	Time (s)	γ	Time (s)	γ	Time (s)
50	150	13.9	15.4	0.1	**12.9**	0.7	**12.9**	1.1
50	200	10.5	11.3	0.0	**9.4**	0.6	**9.4**	1.0
50	250	8	8.6	0.1	**6.9**	0.5	**6.9**	0.7
100	150	19.4	20.8	0.2	**17**	**2.2**	**17**	2.9
100	200	12.8	13.5	0.2	**10.4**	**1.6**	**10.4**	2.0
100	250	9.1	10.2	0.3	**7.5**	**1.1**	**7.6**	1.6
250	150	22.7	24.8	0.5	**18.7**	**6.0**	**18.1**	9.4
250	200	14.6	15.5	0.8	**11.4**	**3.5**	**11**	5.8
250	250	10.1	11.2	1.0	**8**	**3.0**	**8**	4.4
500	150	75.3	84.7	1.7	67.3	95.4	**64.5**	**83.5**
500	200	48.2	55.4	1.5	41.4	**43.8**	**39.8**	51.9
500	250	34.6	36.9	1.0	27.9	**17.6**	**26.8**	33.3
750	150	82.9	90.4	3.2	72.9	**152.8**	**68.7**	170.2
750	200	51.4	59	2.4	43.9	**54.8**	**41.3**	91.6
750	250	35.9	39.2	2.5	28.7	**24.6**	**27.3**	57.0
1000	150	85.9	94.2	4.4	74.8	**215.1**	**70.3**	264.3
1000	200	53	60	3.4	44.8	**65.9**	**42.5**	135
1000	250	36.7	39.8	4.0	29.8	**35.5**	**28.2**	83.9

HGA. However, we note that the solutions generated by ACO-LS are also twice those generated by the GA. Since the cardinality returned by HGA is comparable to the ACO-LS, it is better to use it for UDG instances.

When it comes to the results on Router Waxman topologies in Tables 4.2 and 4.3, the cardinality returned by HGA and ACO-LS are, once again, comparable. We also observe that for large Router Waxman graphs with higher degree of connectivity, the HGA performs slightly better than the ACO-LS algorithm both in terms of cardinality as well as time. For large Router Waxman instances with smaller degree of connectivity, ACO-LS takes significantly lesser time than HGA. Based on these observations, we can say that the ACO-LS algorithm is the better option for these types of graphs.

Table 4.2: Cardinality (γ) of MDS and Time taken in seconds using Heuristic, Hedar-MDS, HGA and ACO-LS for Small Router Waxman Instances with random and heavy-tailed (ht) placement of nodes

N	**Range**	**Placement**	**Heuristic**	**Hedar-MDS**		**HGA**		**ACO-LS**	
				γ	**Time (s)**	γ	**Time (s)**	γ	**Time (s)**
50	100	ht	13.5	15.1	0.1	**12.1**	**0.6**	**12.1**	1.0
50	100	random	12.4	14.4	0.0	**11.6**	**0.7**	**11.6**	**1.0**
50	200	ht	7.7	10.9	0.1	**7**	**0.5**	**7**	**0.6**
50	200	random	7.3	10.3	0.0	**6.8**	**0.5**	**6.8**	**0.8**
50	400	ht	4.7	6.7	0.2	**4.2**	**0.4**	**4.2**	**0.5**
50	400	random	4.1	6.6	0.1	**3.8**	**0.3**	**3.8**	**0.4**
100	200	ht	26.7	31.8	0.1	**23.6**	**2.4**	**23.7**	3.4
100	200	random	25.8	32.1	0.2	**23.4**	**2.7**	**23.4**	3.3
100	400	ht	16.5	21.5	0.2	**14.8**	**1.8**	**14.5**	2.4
100	400	random	16.1	22.1	0.2	**14.7**	**1.9**	**14.4**	2.4
100	800	ht	9.6	15	0.1	**8.7**	**1.1**	**8.7**	1.6
100	800	random	9.2	15.5	0.2	**8.7**	**1.2**	**8.4**	**1.6**
250	500	ht	64.7	77.1	0.6	59.4	24.6	**58.5**	**23.3**
250	500	random	67.2	78.4	0.6	60.8	25.1	**59.8**	**23.5**
250	1000	ht	42.2	56.4	0.7	38.2	**15.2**	**37.2**	17.4
250	1000	random	41.8	57.9	0.6	38.5	**16.2**	**37.1**	17.4
250	1000	ht	26.1	39.6	0.5	23.7	**7.4**	**23.1**	11.1
250	1000	random	25.3	40.2	0.6	23.2	**8.2**	**22.4**	10.9

Table 4.3: Cardinality (γ) of MDS and Time taken in seconds using Heuristic, Hedar-MDS, HGA and ACO-LS for Large Router Waxman Instances with random and heavy-tailed (ht) placement of nodes

N	**Range**	**Placement**	**Heuristic**	**Hedar-MDS**		**HGA**		**ACO-LS**	
				γ	**Time (s)**	γ	**Time (s)**	γ	**Time (s)**
500	1000	ht	131.4	152.7	2.0	122.2	161.7	**117.3**	**129.4**
500	1000	random	131	157.7	2.0	121.7	162.2	**117.9**	**127.2**
500	2000	ht	83.9	113.5	2.3	78.4	**94.5**	**75.8**	97.9
500	2000	random	84.1	116.7	2.3	79.5	100.7	**76.5**	**99.6**
500	4000	ht	52.3	81.1	1.7	50.3	**43.9**	**49**	61
500	4000	random	50.1	83.8	1.7	48.7	**49.1**	**47.2**	60
750	1500	ht	196.6	230.8	4.2	183.7	520.5	**176.9**	**375**
750	1500	random	195.7	241.5	4.3	185.1	527.3	**178.7**	**372.4**
750	3000	ht	125.6	169.2	4.8	**119.2**	302.7	**119.6**	**288.2**
750	3000	random	127.3	176.4	4.6	**120.8**	319.2	**119**	**290.4**
750	6000	ht	78.2	120.5	4.1	**76.6**	134.9	77.7	**171.2**
750	6000	random	77.7	128.9	4.2	76.4	**140.1**	**75.6**	169.5
1000	2000	ht	268	317	7.5	251.3	1211.4	**244.8**	**853.7**
1000	2000	random	259.8	313.5	7.4	247.6	1212.3	**239.1**	**824.7**
1000	4000	ht	168.5	231.5	7.5	**161.1**	720.9	163.3	**635.7**
1000	4000	random	168	236.9	7.7	**160.7**	735.6	**161.3**	**638.2**
1000	8000	ht	104.1	166.3	8.5	**103**	**300.7**	106.5	359.7
1000	8000	random	104	171.4	8.2	**102.5**	**311.1**	106.7	367.8

4.8 Experimental Results for MWDS

4.8.1 Experimentation

The experiments were done on the same data set as that used in *Raka-ACO*. This data set consists of two types of instances. Type I instances are connected undirected graphs with the vertex weights randomly distributed in the interval [20, 70]. For Type II instances, the weights of the vertices are chosen to be a function of the degree of the node and randomly distributed in the interval $[1, d(v)^2]$. The number of nodes in the graphs varies from 50 to 1000 nodes. The number of edges in the graphs is varied to observe the effect of the degree of connectivity on the results. For each graph of a specific number of nodes and edges, there are 10 instances. The results presented in the tables are the average of running the algorithms on these 10 instances per node and degree combination. For algorithms proposed by us, we also report the standard deviation. Data for *Raka-ACO* is taken from their paper, where no standard deviation is reported. Therefore, it is not included here. The best results are shown in bold in all the tables. Since we run our algorithms once on each of the ten instances and the value of the solution varies from instance to instance, the standard deviation is high for all approaches.

We also generated UDG instances using the topology generator of [68]. The UDG graphs have nodes ranging from 50-1000 nodes with a node's transmission range of 150 and 200 units. The range is changed to change the degree of connectivity for a single graph size. The nodes are distributed randomly in an area of 1000×1000 units. Since the area is limited and constant, as the number of nodes in the graph increases, the average degree of a graph increases. For each graph size and range, we have 10 instances over which the results are averaged.

Proposed HGA Parameters: The hybrid genetic algorithm that we implemented has an initial population of 100 members and we ran it for 20,000 generations. This would mean that a total of 20,000 solutions are created as we create one new member of the population in each generation. We use the following values for the various probabilities: when generating random population members both in initial population and a new member in later generations, we use a probability of 0.3 for adding a node into the dominating set. We use the value of $p_c = 0.9$ for crossover, $p_m = 0.005$ for mutation, $p_{better} = 0.8$ to choose a better candidate during binary tournament selection, $p_h = 0.7$ for using the heuristic to repair a member and finally a value of $p_r = 0.6$ for improving the fitness of a member through minimization. All of these values were arrived at after extensive experimentation with different values.

Proposed ACO-LS parameters: The pheromone value of all the nodes is initialized to 10.0. We use 20 ants and 1000 iterations in our ACO algorithm and always generate 20,000 solutions before terminating. The pheromone persistent rate $\rho = 0.985$. When deleting redundant nodes in the minimization heuristic, we remove the nodes with maximum weight to its absolute degree ratio with probability $p_r = 0.6$ and the rest of the time, we delete the node randomly. When we use pre-processing, we use 100 MIS solutions to update the initial pheromone. We also use one ant per iteration from the heuristic. If this is the best ant found in the iteration, it enhances the pheromone on the nodes that form the solution of the heuristic so these nodes have much higher chance of being part of the final solution. Since the heuristic is deterministic, the pheromone on the same set of nodes can be repeatedly enhanced if it is the best ant of the iteration.

4.8.2 Discussion of Results

Heuristic Results: The results obtained by running the various heuristics described in Section 4.3 are given in tables 4.4–4.6 for Type I instances. The results for Type II instances are presented in tables 4.7–4.9 and those for UDG instances in table 4.16. We can see that the first, second and fourth heuristics give similar results with no clear pattern of a particular heuristic being better for the general graph instances. On the other hand, for UDG instances, the fourth heuristic is found to be best in all instances except for 50 nodes and 150 range. Thus, we can conclude that for UDG instances, this is the best heuristic. Hence, we compare the results of this heuristic with the metaheuristic algorithms for UDGs. For general graph instances, however, we use the second heuristic for comparison with the metaheuristics.

Comparison with *Raka-ACO* [57]: As can be readily observed from tables 4.10–4.12 for Type I instances, for all instances except for 1000 nodes with 1000 edges, one or both of our metaheuristic algorithms does better than *Raka-ACO*. This is despite the fact that 10 ants were used in the *Raka-ACO* with 10,000 iterations which implies a total of 100,000 solutions. We, on the other hand, generate only 20,000 solutions in HGA and ACO-LS. Interestingly, the heuristic itself performs better than *Raka-ACO* in many instances except those with very low degree of connectivity. When the degree of connectivity or the number of nodes in the graph increases, the heuristic performs better than *Raka-ACO*. The *Raka-ACO* algorithm improves upon the heuristics described in [57], but does not perform as well as three out of the four heuristics we have chosen to work with. When it comes to results of Type II instances shown in tables 4.13–4.15, the difference in the performance of *Raka-ACO* on the one hand and the heuristic and our metaheuristic algorithms on the other hand is quite large. In this case, our algorithms give consistently better results than the *Raka-ACO* without exceptions. In fact, since the weight of a node is dependent on the degree of the node, as the degree increases, the performance of the *Raka-ACO* algorithm is much worse than the heuristic and our algorithms. This is exacerbated when the number of nodes in the graph itself is high.

Comparison of Proposed HGA and ACO-LS algorithms: The results of running our algorithms on the UDG instances generated are given in tables 4.16 and 4.17. The results returned by our proposed HGA and ACO-LS algorithms are 7-14% better than the heuristic solution for UDGs.

The following are the highlights of the results when comparing our proposed HGA and ACO-LS/ACO-PP-LS algorithms:

1. For UDG instances, all the proposed metaheuristic algorithms perform in a similar fashion and they are better than the best heuristic for all instances. However, the difference becomes smaller as the degree of connectivity increases. This could be due to the fact that the solution of both the heuristic and the metaheuristics approaches the optimal solution. The standard deviation of our metaheuristic algorithms is also found to be similar.

2. For Type I instances, the ACO-LS and ACO-PP-LS algorithms perform better than the HGA whenever the average degree of the graph is less. As the number of nodes in the graph increases, the ACO-LS variants perform better than the HGA until the average degree becomes really high. While in a 250-node graph, the HGA performs better than the ACO-LS variants when the number of edges is 2000, giving an average degree of 8, for a 500-node graph, it is better than the

ACO-LS variants only for the 10,000 edge instances with an average degree of 20. The difference in solution is not statistically significant. On the other hand, the time taken by ACO-LS and ACO-PP-LS algorithms is half the time taken by the HGA algorithm. So, for Type I instances, we can get better or similar quality solutions as the HGA using ACO-LS algorithm in a much shorter time.

3. For Type II instances, as for Type I instances, ACO-LS algorithms work better than the HGA for a low average degree of a graph. However, unlike in Type I instances, as the number of nodes in the graph and average degree increases, the ACO-LS solutions are worse than the HGA solution. The time taken by ACO variants to compute solutions for Type II instances is very less compared to HGA. Thus, it is better to use the ACO-LS algorithms whenever the solution difference is not statistically significant.

4. Using the pre-processing step in the ACO algorithm does not really help in terms of improving the quality of the solution. In fact, in many instances the results returned by ACO-LS are better than ACO-PP-LS. However, when it comes to the time taken, the pre-processing step helps speed up the algorithm in almost all cases.

5. In the majority of instances, the metaheuristic approach which yielded the best mean also has the least standard deviation. This is especially true for large Type I instances. On the other hand, for large Type II instances, this observation does not hold.

Run-Time Comparison: The time taken to compute the results for our proposed HGA, ACO-LS and ACO-PP-LS algorithms is given in tables 4.18 and 4.19 for Type I instances and in tables 4.20 and 4.21 for Type II instances. As noted above, the time taken by the ACO algorithms is, in some cases, as much as half or 1/3 the time taken by HGA. This makes the ACO a better option to use as the difference in cardinality is not statistically significant in most cases.

Table 4.4: WMDS using Different Heuristics for Small Type I Instances

N	Edges (M)	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u) \times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	50	**578.3**	**35.57**	588	41.48	598.4	52.33	583.9	41.82
50	100	410.5	36.34	407.4	40.17	413.2	23.13	**405.9**	**19.89**
50	250	197.2	20.42	198.8	15.28	208.8	21.22	**195.2**	**12.54**
50	500	105.5	9.89	103.3	11.07	122.4	18.06	**99.9**	**5.88**
50	750	72.9	9.35	72	8.65	82.5	12.52	**67.9**	**5.15**
50	1000	48.1	9.72	47.9	9.63	55.9	9.09	**44.8**	**5.79**
100	100	1168	**55.6**	1176.5	66.9	1178.8	72.9	**1165.7**	56.06
100	250	676.7	36.96	673	33.95	708.4	50.95	**668.7**	**24.57**
100	500	397.3	36.09	**396.6**	36.64	415.7	**24.23**	406.8	39.83
100	750	294.9	**13.54**	**286.1**	21.53	312	23.99	293.4	16.87
100	1000	235.1	**7.59**	234	11.07	258.4	27.44	**230.2**	14.82
100	2000	122.3	8.37	119.6	10.42	146.1	13.79	**115.3**	**7.63**
150	150	1727.6	84.34	1725.2	**74.79**	1748	95.68	**1717.9**	84.56
150	250	1340.5	67.32	1353	67.65	1389.4	**40.05**	**1318.3**	65.14
150	500	830.4	58.44	**826**	**54.08**	890.5	65.17	846.7	65.54
150	750	611.4	**41.6**	617.5	55.94	673	**41.16**	**609.8**	43.97
150	1000	**488.7**	28.48	496.1	**22.91**	547	33.08	493.1	28.64
150	2000	271.5	**10.18**	**266.5**	14.63	311.4	21.67	276.6	17.63
150	3000	191.1	**12.59**	**186.1**	13.19	225.2	21.62	187	19.6

Table 4.5: WMDS using Different Heuristics for Medium Type I Instances

N	Edges (M)	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u) \times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
200	250	**2095.1**	75.6	2112.6	**70.82**	2158.8	79.4	2108.7	76.34
200	500	**1380.8**	56.27	1403.5	57.88	1429.4	56.65	1390.7	**43.38**
200	750	1020.5	**61.08**	**1019.1**	65.51	1087.5	84.96	1039.4	75.69
200	1000	**809.6**	**27.04**	813.4	**27.04**	889	52.99	819.8	37.95
200	2000	**467.5**	**21**	473.9	**20.67**	562.7	25.36	485.6	22.45
200	3000	334.8	19.06	**331.8**	16.7	368.6	21.96	334.9	**14.5**
250	250	2906	**89.12**	2931.6	96.5	2900.7	**89.7**	**2891.3**	94.22
250	500	2040.9	114.18	2014.6	89.38	2058.2	**79.87**	**2012.1**	106.87
250	750	1533.8	72.9	1542.3	61.71	1607.4	64.95	**1510.5**	**48.37**
250	1000	1238.8	51.52	1232.4	69.94	1309.3	**43.79**	**1225.8**	56.54
250	2000	715.8	29.45	706.4	35.81	778.6	32.23	**705.4**	**26.35**
250	3000	**500.4**	**18.72**	**500.2**	20.78	572.7	32.08	521.1	23.25
250	5000	328.4	**13.44**	**323.6**	17.28	392.1	22.01	325.9	16.54
300	300	3481.5	118.14	3504.9	**83.26**	3506.7	105.85	**3465.7**	116.99
300	500	**2697.9**	92.03	2709.7	109.48	2772.8	95.99	2707.3	**85.07**
300	750	2104.3	88.5	2106.3	89.63	2146	**84.8**	**2073.3**	96.55
300	1000	**1688**	60.42	1674.4	55.68	1758.1	**53.63**	1692.2	71.29
300	2000	965.6	51.4	**949.5**	27.81	1060.7	49.84	988.2	**42.34**
300	3000	701	30.61	**693.7**	31.01	773.4	32.75	707.3	**24.17**
300	5000	459.8	21.74	**457.2**	**17.25**	554.5	29.01	472.6	19.73

Table 4.6: WMDS using Different Heuristics for Large Type I Instances

N	**Edges (M)**	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u) \times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
500	500	5820.5	133.1	5840.4	**124.91**	5872.3	**125.05**	**5798.1**	126.7
500	1000	4039.8	121.86	4038.9	123	4141.4	118.88	**4032.6**	**95.44**
500	2000	2484.9	79.59	2478.7	92.24	2585.6	80.38	**2452.2**	**73.56**
500	5000	1177.1	**32.25**	**1171.3**	33.45	1354.6	69.83	1193.5	54.68
500	10000	670.5	**10.65**	**656.1**	25.68	798.7	48.63	666.3	24.24
800	1000	8472.1	157.12	8512.6	**151.07**	8536.3	157.9	**8402.6**	208.45
800	2000	5641.4	165.59	5625.7	**85.19**	5810.3	92.1	**5602.4**	109.62
800	5000	2739.9	71.15	**2735.5**	61.18	3060.3	**52.4**	2798.1	69.17
800	10000	1590.6	36.76	**1553**	40.35	1823.9	60.04	1593.5	**28.76**
1000	1000	11666.3	162.07	11701.4	185.26	11673.9	196.46	**11635.5**	**153.42**
1000	5000	4168.5	98.69	**4114.4**	68.47	4492.1	132.82	4187.8	**58.09**
1000	10000	2379.8	68.02	**2368.1**	**63.41**	2702.5	79.08	2410.8	71.6
1000	15000	**1704**	**27.07**	**1704.3**	39.21	1998.8	40.95	1711.1	32.31
1000	20000	1351.3	26.86	**1322.9**	**22.94**	1593.6	50.12	1367.8	28.68

Table 4.7: WMDS using Different Heuristics for Small Type II Instances

N	**Edges (M)**	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u) \times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	50	**64.9**	**7.58**	**65**	8.22	71.1	11.87	65.8	**7.45**
50	100	100.1	21.11	99.4	22.52	124.8	26.55	**97**	**17.06**
50	250	**165.4**	52.33	166.5	**47.64**	273.6	48.53	167.9	54.84
50	500	205.5	88.26	**201.1**	**75.5**	732.2	337.15	205	81.8
50	750	178.1	93.8	178.1	93.8	919.8	581.66	**175.3**	**90.62**
50	1000	**162.8**	108.66	167.5	**106.73**	847.8	653.35	**162.8**	108.66
100	100	133	17.22	133.7	**16.69**	138	20.55	**131**	17.92
100	250	**229.2**	**24.22**	232.7	25.34	319.1	42.67	235.8	30.86
100	500	**340.8**	63.64	343.2	**60.47**	639.9	100.06	355.8	68.94
100	750	**437.8**	104.62	**438.2**	**98.59**	902.7	243.06	459.5	118.66
100	1000	**470.3**	84.91	480.1	**83.57**	1708	503.28	486.2	137.3
100	2000	**615.4**	**197.06**	632.4	223.03	3099	1124.12	635.4	222.77
150	150	**198**	19.06	**197.9**	**17.65**	208.4	22.87	198.8	18.76
150	250	255.4	28.7	**254.9**	**26.1**	331.4	44.53	263.4	30.86
150	500	**376.8**	39.97	378.9	**35.03**	642.9	116.03	388.2	46.66
150	750	513.5	97.18	**502**	**87.82**	1024.5	206.14	529.2	91.32
150	1000	622.8	109.74	**613.6**	**93.9**	1422.1	369.14	620.3	102.25
150	2000	833.4	233.91	804.1	253.94	2988.8	789.67	**776.8**	**226.88**
150	3000	888.5	268.37	**880.4**	**248.24**	4329.5	1324.29	908.4	291.82

Table 4.8: WMDS using Different Heuristics for Medium Type II Instances

N	**Edges (M)**	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u)\times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
200	250	297.4	26.78	297	**23.62**	345.3	28.25	**291.6**	29.59
200	500	**427**	62.33	**426.6**	54.83	673	**48.91**	435.2	56.09
200	750	**560.5**	74.4	561.7	**66.64**	1038	97.26	567.9	84.21
200	1000	**668.1**	**48.15**	673	66.46	1413.5	149.06	677.5	64.34
200	2000	1004.7	162.63	**959.4**	156.86	3008.8	610.62	995.7	**141.75**
200	3000	1140.8	176.53	1164.2	166.26	5445.9	708.58	**1126.3**	**163.97**
250	250	327.2	23	331.6	**22.24**	339.7	26.11	**325.4**	**22.25**
250	500	**485.7**	44.09	**485.2**	**39.6**	679.7	61.25	495.3	43.85
250	750	635.4	50.61	**632.9**	48.34	1013.1	121.53	661.2	**46.53**
250	1000	**749.7**	73.98	753	**61.65**	1297.8	203.01	788.5	78.44
250	2000	**1154.3**	185.42	**1154.8**	**173.51**	3051.5	438.6	1187.4	196.69
250	3000	1438.8	**225.95**	**1433.9**	295.33	5054.5	1127.1	1467.1	226.64
250	5000	1741.1	448.61	**1628.7**	366.79	8994.6	1486.89	1656.1	**290.92**
300	300	397.9	27.98	401.4	**25.61**	413.4	28.85	**393.2**	27.53
300	500	516	**40.73**	**512.9**	42.04	680.8	81.98	538.2	49.51
300	750	680.7	**57.89**	**675.4**	65.26	989.6	84.9	697.6	80.76
300	1000	828.7	**90.36**	**817.8**	92.48	1365.2	120.86	844.5	97.06
300	2000	1240.6	128.54	**1207.6**	127.13	2687.6	523.75	1248.9	**114.48**
300	3000	1555.3	163.37	**1510.1**	**141.32**	4668.2	791.36	1576.1	167.66
300	5000	1941.1	361.41	**1862.5**	302.84	8027.4	1776.97	1869.2	**278.07**

Table 4.9: WMDS using Different Heuristics for Large Type II Instances

N	**Edges (M)**	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u)\times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
500	500	668.6	31.39	678.5	**26.83**	699.7	36.96	**659.2**	31.98
500	1000	987.5	**45.8**	**985.2**	50.42	1329.9	127.81	1012.1	60.73
500	2000	**1508.1**	132.91	1526.7	**122.58**	2712.6	264.76	1554.1	140.33
500	5000	2668.1	283.36	**2652.5**	**258.04**	8300.6	1051.59	2729.9	325.67
500	10000	3723.3	452.27	**3630.6**	**439.89**	18447.9	2526.29	3704	452.2
800	1000	**1193.1**	52.12	1197	54.93	1403.3	71.6	1198	**47.13**
800	2000	1813	**97.39**	**1804.5**	108.64	2720.4	119.9	1853.5	108.68
800	5000	**3321.5**	231.13	3225.9	**212.11**	7439.3	350.99	3343.2	308.24
800	10000	4725.3	431.67	**4657**	**396.56**	17790	1941.18	4810.9	420.11
1000	1000	1338.5	39.15	1346.7	38.76	1383.5	46.59	**1324.5**	**35.22**
1000	5000	3596.4	**230.69**	**3553.2**	273.23	6911.8	478.41	3674.6	284.24
1000	10000	5432.6	**286.66**	**5368**	363.5	16196.5	1520.76	5541.9	366.72
1000	15000	6857	**461.14**	**6734.3**	497.71	27251.8	3464.45	7030.1	703.76
1000	20000	7785.6	697.26	**7602**	**576.8**	36355.7	3286.4	7824.7	643.22

Table 4.10: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Small Type I Instances

N	**Edges (M)**	**Heuristic**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**	
				Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	50	578.3	539.8	**531.3**	**26.86**	**531.3**	**26.86**	532.6	**26.26**
50	100	410.5	391.9	**371.2**	**22.63**	**371.2**	**22.63**	**371.5**	**22.6**
50	250	197.2	195.3	**175.7**	**8.38**	**176**	**8.41**	**175.7**	**8.38**
50	500	105.5	112.8	**94.9**	**6.3**	**94.9**	**6.3**	**95.2**	**6.58**
50	750	72.9	69.0	**63.1**	**6.15**	**63.1**	**6.15**	**63.2**	**6.12**
50	1000	48.1	44.7	**41.5**	**1.78**	**41.5**	**1.78**	**41.5**	**1.78**
100	100	1168	1087.2	1081.3	52.4	**1066.9**	**47.85**	**1065.4**	**47.85**
100	250	676.7	698.7	**626.2**	36.17	627.2	**30.7**	627.4	**30.84**
100	500	397.3	442.8	**358.3**	19.98	362.5	19.23	363.2	**18.47**
100	750	294.9	313.7	**261.2**	13.85	263.5	15.12	265	**12.8**
100	1000	235.1	247.8	**205.6**	**8.1**	209.2	8.61	208.8	9.24
100	2000	122.3	125.9	**108.2**	3.12	**108.1**	**3.07**	**108.4**	3.47
150	150	1727.6	1630.1	1607	64.49	**1582.8**	**57.9**	1585.2	59.21
150	250	1340.5	1317.7	1238.6	48.56	**1237.2**	45.48	1238.3	**44.26**
150	500	830.4	899.9	**763**	43.02	767.7	45.69	768.6	**42.79**
150	750	611.4	674.4	**558.5**	**27.83**	565	31.16	562.8	33.39
150	1000	488.7	540.7	**438.7**	23.02	446.8	22.46	448.3	**21.31**
150	2000	271.5	293.1	**245.7**	13.94	259.4	9.34	255.6	**9.05**
150	3000	191.1	204.7	**169.2**	10.39	173.4	10.86	175.2	**7.89**

Table 4.11: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Medium Type I Instances

N	**Edges (M)**	**Heuristic**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**	
				Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
200	250	2095.1	2039.2	1962.1	**52.3**	1934.3	**52.62**	**1927**	54.6
200	500	1380.8	1389.4	1266.3	**40.07**	**1259.7**	45.8	1260.8	50.59
200	750	1020.5	1096.2	939.8	49.22	**938.7**	**44.79**	940.1	**44.97**
200	1000	809.6	869.9	**747.8**	**16.41**	751.2	**16.4**	753.7	19.3
200	2000	467.5	524.1	**432.9**	17.46	440.2	**16.52**	444.7	**16.61**
200	3000	334.8	385.7	**308.5**	12.28	309.9	**11.86**	315.2	17.44
250	250	2906	NA	2703.4	80.7	**2655.4**	**72.91**	**2655.4**	74.63
250	500	2040.9	NA	1878.8	77.85	1850.3	**55.49**	**1847.9**	68.49
250	750	1533.8	NA	1421.1	40.16	**1405.2**	36.56	**1405.5**	**34.09**
250	1000	1238.8	NA	1143.4	43.43	1127.1	43.03	**1122.9**	**26.48**
250	2000	715.8	NA	656.6	30.73	**672.8**	**29.18**	676.4	30.88
250	3000	500.4	NA	**469.3**	22.09	474.1	**20.02**	476.3	21.77
250	5000	328.4	NA	**300.5**	**9.65**	310.4	12.18	308.7	11.56
300	300	3481.5	NA	3255.2	74.13	**3198.5**	**63.82**	3205.9	70.39
300	500	2697.9	NA	2509.8	**69.21**	2479.2	80.75	**2473.3**	84.44
300	750	2104.3	NA	1933.9	81.23	**1903.3**	**55.08**	1913.9	64.69
300	1000	1688	NA	1560.1	35.27	**1552.5**	**32.51**	1555.8	36.19
300	2000	965.6	NA	**909.6**	**22.85**	916.8	25.61	916.5	23.08
300	3000	701	NA	**654.9**	**24.44**	667.8	30	670.7	28
300	5000	459.8	NA	**428.3**	**15.07**	437.4	16.59	435.9	16.22

Table 4.12: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Large Type I Instances

N	Edges (M)	Heuristic	Raka-ACO	HGA		ACO-LS		ACO-PP-LS	
				Mean	SD	Mean	SD	Mean	SD
500	500	5820.5	5476.3	5498.3	113.45	5398.3	100.57	**5387.7**	**99.53**
500	1000	4039.8	4069.8	3798.6	92.08	3714.8	103.77	**3698.3**	**85.61**
500	2000	2484.9	2627.5	2338.2	77.75	2277.6	**60.2**	**2275.9**	65.12
500	5000	1177.1	1398.5	1122.7	**30.96**	1115.3	35.79	**1110.2**	41.94
500	10000	670.5	825.7	**641.1**	22.18	652.8	**11.81**	650.9	**11.39**
800	1000	8472.1	8098.9	**8017.7**	**141.01**	8117.6	162.03	8068	178.6
800	2000	5641.4	5739.9	**5318.7**	**130.02**	5389.9	151.14	5389.6	144.43
800	5000	2739.9	3116.5	2633.4	69.07	2616	66.49	**2607.9**	**62.02**
800	10000	1590.6	1923.0	1547.7	45.66	**1525.7**	32.4	1535.3	**31**
1000	1000	11666.3	**10924.4**	11095.2	147.38	11035.5	174.92	11022.9	**129.43**
1000	5000	4168.5	4662.7	**3996.6**	**73.73**	4012	81.91	4029.8	85.9
1000	10000	2379.8	2890.3	2334.7	64.51	2314.9	64.03	**2306.6**	**56.03**
1000	15000	1704	2164.3	1687.5	**28.29**	**1656.3**	44.23	1657.4	40.05
1000	20000	1351.3	1734.3	1337.2	30.97	**1312.8**	**22.52**	1315.8	24.1

Table 4.13: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Small Type II Instances

N	**Edges (M)**	**Heuristic**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**	
				Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	50	64.9	62.3	**60.8**	**5.71**	**60.8**	**5.71**	**60.8**	**5.71**
50	100	100.1	98.4	**90.3**	**17.21**	**90.3**	**17.21**	**90.3**	**17.21**
50	250	165.4	202.4	**146.7**	**40.54**	**146.7**	**40.54**	**146.7**	**40.54**
50	500	205.5	312.9	**179.9**	**63.14**	**179.9**	**63.14**	**179.9**	**63.14**
50	750	178.1	386.3	**171.1**	**91.33**	**171.1**	**91.33**	**171.1**	**91.33**
50	1000	162.8	NA	**146.5**	**97.3**	**146.5**	**97.3**	**146.5**	**97.3**
100	100	133	126.5	124.5	15.26	**123.6**	**14.77**	**123.5**	**14.74**
100	250	229.2	236.6	211.4	**21.76**	**210.2**	**21.95**	**210.4**	22.43
100	500	340.8	404.8	**306**	45.17	307.8	**44.7**	308.4	**44.46**
100	750	437.8	615.1	**385.3**	**82.76**	**385.7**	83.08	**386.3**	**82.64**
100	1000	470.3	697.3	**429.1**	**76.13**	430.3	79.43	430.3	79.43
100	2000	615.4	1193.9	**550.6**	171.77	558.8	**169.7**	559.8	171.82
150	150	198	190.1	186	18.24	**184.7**	**17.99**	**184.9**	18.21
150	250	255.4	253.9	234.9	21.47	**233.2**	21.02	**233.4**	**20.84**
150	500	376.8	443.2	**350**	**36.99**	351.9	38.66	351.9	38.66
150	750	513.5	623.3	455.8	77.8	456.9	**77.96**	**454.7**	**78.06**
150	1000	622.8	825.3	**547.5**	82.12	551.4	83.67	549	**81.68**
150	2000	833.4	1436.4	**720.1**	180.32	725.7	**179.45**	725.7	**179.45**
150	3000	888.5	1751.9	**792.6**	**218.03**	794	220.1	806.2	245.43

Table 4.14: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Medium Type II Instances

N	Edges (M)	Heuristic	Raka-ACO	HGA		ACO-LS		ACO-PP-LS	
				Mean	SD	Mean	SD	Mean	SD
200	250	297.4	293.2	275.1	21.55	**272.6**	20.12	**272.6**	**20.31**
200	500	427	456.5	390.7	**55.66**	**388.6**	56.42	**388.4**	56.64
200	750	560.5	657.9	507	60.41	**501.7**	**50.44**	**501.4**	**50.1**
200	1000	668.1	829.2	**601.1**	52.84	605.9	**47.97**	605.8	49.16
200	2000	1004.7	1626.0	893.5	150.64	**891**	136.47	892.9	**133.15**
200	3000	1140.8	2210.3	**1021.3**	**162.54**	1027	164.38	1034.4	167.73
250	250	327.2	NA	310.1	19.6	**306.5**	**17.89**	**306.7**	**17.98**
250	500	485.7	NA	444	**30.35**	443.8	32.84	**443.2**	32.47
250	750	635.4	NA	578.2	42.33	**573.1**	**40.82**	575.9	42.7
250	1000	749.7	NA	672.8	59.42	**671.8**	**58.56**	675.1	60.79
250	2000	1154.3	NA	**1030.8**	139.83	1033.9	131.02	1031.5	**129.71**
250	3000	1438.8	NA	**1262**	216.63	1288.5	**212.74**	1277	228.16
250	5000	1741.1	NA	**1480.9**	307.17	1493.6	**306.43**	1520.1	349.3
300	300	397.9	NA	375.6	24.79	**371.1**	**23.14**	**371.1**	**23.44**
300	500	516	NA	484.2	39.19	**480.8**	**40.13**	**481.2**	**40.05**
300	750	680.7	NA	623.8	52.76	621.6	**48.76**	**618.3**	**48.9**
300	1000	828.7	NA	751.1	75.91	744.9	77.8	**743.5**	**74.2**
300	2000	1240.6	NA	**1106.7**	116.09	1111.6	114.61	1107.5	**112.03**
300	3000	1555.3	NA	**1382.1**	**125.93**	1422.8	153.78	1415.3	167.5
300	5000	1941.1	NA	**1686.3**	294.44	1712.1	**291.41**	1698.6	300.02

Table 4.15: WMDS using Heuristic, Raka-ACO, HGA, ACO-LS and ACO-PP-LS for Large Type II Instances

N	**Edges (M)**	**Heuristic**	**Raka-ACO**	**HGA**		**ACO-LS**		**ACO-PP-LS**	
				Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
500	500	668.6	651.2	632.9	**29.54**	**627.5**	30.06	**627.3**	30.13
500	1000	987.5	1018.1	919.2	41.71	**913**	**35.69**	**912.6**	36.56
500	2000	1508.1	1871.8	1398.2	131.9	1384.9	**121.03**	**1383.9**	**121.77**
500	5000	2668.1	4299.8	**2393.2**	**222.03**	2459.1	272.38	2468.8	260.35
500	10000	3723.3	8543.5	**3264.9**	**421.38**	3377.9	470.35	3369.4	482.89
800	1000	1193.1	1171.2	1128.2	**48.22**	1126.4	51.56	**1125.1**	50.79
800	2000	1813	1938.7	**1679.2**	**74.7**	1693.7	80.25	1697.9	80.26
800	5000	3321.5	4439.0	**3003.6**	**204.03**	3121.9	227.35	3120.9	229.21
800	10000	4725.3	8951.1	**4268.1**	379.71	4404.1	380.67	4447.9	**371.23**
1000	1000	1338.5	1289.3	1265.2	**30.99**	1259.3	33.44	**1258.6**	34.35
1000	5000	3596.4	4720.1	**3320.1**	221.66	3411.6	228.22	3415.1	**209.28**
1000	10000	5432.6	9407.7	**4947.5**	330.77	5129.1	308.66	5101.9	**306.17**
1000	15000	6857	14433.5	**6267.6**	463.09	6454.6	**445.76**	6470.6	467.53
1000	20000	7785.6	19172.6	**7088.5**	659.71	7297.4	**598.98**	7340.8	604.06

Table 4.16: WMDS using Different Heuristics for Unit Disk Graph Instances

N	Edges (M)	$\frac{W(u)}{w(u)}$		$\frac{d_w(u)}{w(u)}$		$W(u) - w(u)$		$\frac{d_w(u) \times W(u)}{w(u)}$	
		Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	150	425.4	72.09	**428.2**	71.52	463.2	**48.4**	429.6	73.84
50	200	276.8	55.86	271	53.58	323.6	66.49	**264.5**	**46.98**
100	150	496.1	**73.2**	494.5	79.43	598.6	117.38	**484.4**	90.04
100	200	247.9	27.29	246.1	**24.37**	348.2	46.59	**237.5**	25.95
250	150	335.1	63.91	336	66.85	584.7	94.67	**323.8**	**56.41**
250	200	143.5	29.06	142.1	28.12	399.5	71.8	**135.1**	**25.77**
500	150	195.1	50.79	196.1	51.67	656.8	115.04	**183.8**	**47.38**
500	200	77.5	13.21	78	**12.94**	374.9	71.72	**77.4**	15.66
800	150	135.4	**22.45**	133.7	**22.24**	709.8	160.15	**129.5**	24.66
800	200	53.4	13.01	53	12.33	359.4	56.68	**48.2**	**9**
1000	150	139.8	**14.09**	139.3	16.52	639.3	140.73	**136.3**	**14.05**
1000	200	53.9	9.31	54.1	10.26	399.4	59.55	**53.4**	**8.58**

Table 4.17: WMDS using Heuristic, HGA, ACO-LS and ACO-PP-LS for Unit Disk Graph Instances

N	Edges (M)	Heuristic	HGA		ACO-LS		ACO-PP-LS	
			Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
50	150	429.6	**394.3**	59.42	**393.9**	**58.89**	**393.9**	**58.89**
50	200	264.5	**247.8**	**48.06**	**247.8**	**48.06**	**247.8**	**48.06**
100	150	484.4	450.4	**73.23**	**449.7**	**73.71**	**449.7**	**73.71**
100	200	237.5	217.3	**18.56**	**216**	**18.34**	**216**	**18.34**
250	150	323.8	**294.2**	53.42	**294.6**	**52.92**	**294.5**	**52.92**
250	200	135.1	**119.1**	**17.8**	**118.9**	**17.95**	**118.9**	**17.95**
500	150	183.8	172.7	36.91	**170.8**	**35.88**	**170.9**	**35.45**
500	200	77.4	**68.1**	**11.29**	**68**	**11.37**	**68**	**11.37**
800	150	129.5	115.9	18.33	**114.1**	**16.58**	**114**	**16.45**
800	200	48.2	42.6	8.04	**41**	**7.6**	**40.8**	**7.47**
1000	150	136.3	125	16.17	**121.9**	14.12	**121.9**	**13.64**
1000	200	53.4	47.2	6.12	**46**	5.72	**46.2**	**5.55**

Table 4.18: Time taken in seconds to compute WMDS using HGA, ACO-LS and ACO-PP-LS for Small and Medium Type I Instances

N	Edges (M)	HGA	ACO-LS	ACO-PP-LS
50	50	2.7	**1.2**	**1.1**
50	100	2.6	**1**	**0.9**
50	250	2.3	**0.6**	**0.6**
50	500	2	**0.5**	**0.5**
50	750	2.1	**0.3**	**0.3**
50	1000	2.1	**0.3**	**0.3**
100	100	9.6	4.3	**3.9**
100	250	8.4	3.1	**2.8**
100	500	5.8	2.3	**2**
100	750	4.8	**2**	**1.9**
100	1000	4.9	**1.7**	**1.7**
100	2000	4.9	**1.2**	**1.2**
150	150	23.4	9.9	**8.8**
150	250	20.8	8.6	**7.5**
150	500	15.3	6.1	**5.5**
150	750	12.2	**5**	**4.5**
150	1000	11.8	**4.5**	**4**
150	2000	9.2	**3.3**	**3.3**
150	3000	8.3	**3**	**2.8**
200	250	41.7	17.7	**15.3**
200	500	33.4	13.2	**11.5**
200	750	28.1	10.6	**9.2**
200	1000	24.9	9	**8**
200	2000	15.2	**6.5**	**6**
200	3000	14.4	**5.7**	**5.4**
250	250	72.7	32.5	**28.4**
250	500	59.9	25.1	**21.9**
250	750	55	20	**17.4**
250	1000	47.3	17.3	**15.2**
250	2000	26.1	11.6	**10.7**
250	3000	23	**9.5**	**9**
250	5000	21.8	**8.4**	**8.1**

Table 4.19: Time taken in seconds to compute WMDS using HGA, ACO-LS and ACO-PP-LS for Large Type I Instances

N	Edges (M)	HGA	ACO-LS	ACO-PP-LS
300	300	116.3	49.2	**42.2**
300	500	109	41	**35.7**
300	750	93.2	34	**30**
300	1000	80.1	28.2	**24.5**
300	2000	47.6	18.8	**17**
300	3000	37.3	15.4	**14.3**
300	5000	27.4	**12.5**	**12**
500	500	412.3	180.3	**156**
500	1000	359.8	**143.7**	116.4
500	2000	219.2	90.3	**81.7**
500	5000	114.4	51.3	**45.7**
500	10000	64	36.9	**35.8**
800	1000	1459.5	769.8	**709.6**
800	2000	1094.3	572.6	**554.8**
800	5000	551.5	263.5	**237.3**
800	10000	246.1	147.5	**145.4**
1000	1000	2829.6	1320.7	**1189.9**
1000	5000	1152.1	627.7	**600.8**
1000	10000	566.2	308.8	**289.9**
1000	15000	356.3	227.7	**211**
1000	20000	251.1	204.9	**200.3**

Table 4.20: Time taken in seconds to compute WMDS using HGA, ACO-LS and ACO-PP-LS for Small and Medium Type II Instances

N	Edges (M)	HGA	ACO-LS	ACO-PP-LS
50	50	3.8	**1.2**	**1.2**
50	100	3.8	**1.1**	**1.1**
50	250	3.4	**0.8**	**0.7**
50	500	2.8	**0.6**	**0.6**
50	750	2.5	**0.5**	**0.4**
50	1000	1.9	**0.3**	**0.2**
100	100	11	**4.2**	**4**
100	250	11	**3.5**	**3.2**
100	500	10.2	**2.7**	**2.5**
100	750	8.6	**2.4**	**2.3**
100	1000	8.5	**2**	**1.9**
100	2000	7.8	**1.6**	**1.5**
150	150	29.5	**9.1**	**8.6**
150	250	29.1	**8.6**	**7.8**
150	500	24.5	7	**6.2**
150	750	21.8	**5.9**	**5.6**
150	1000	19.7	**5.5**	**5.1**
150	2000	17.5	**4.4**	**4.1**
150	3000	15.4	**3.6**	**3.6**
200	250	53.9	**17.1**	**15.6**
200	500	49.9	**14.6**	**12.8**
200	750	45.9	**12.5**	**11.1**
200	1000	37.3	**11.2**	**10.2**
200	2000	31.9	**8.9**	**8.1**
200	3000	29.9	**7.2**	**7**
250	250	88	**28**	**25.5**
250	500	89.1	**25.4**	**22.8**
250	750	77.1	**23**	**20.6**
250	1000	74.2	**20.8**	**19**
250	2000	56.8	**15.9**	15.1
250	3000	49	**14**	**13.5**
250	5000	46.9	**11.2**	**10.9**

Table 4.21: Time taken in seconds to compute WMDS using HGA, ACO-LS and ACO-PP-LS for Large Type II Instances

N	**Edges (M)**	**HGA**	**ACO-LS**	**ACO-PP-LS**
300	300	142.3	**43**	**39.2**
300	500	143.4	**40**	**35.8**
300	750	124.5	**36.9**	**32.9**
300	1000	113	**33.7**	**30.5**
300	2000	87.7	**24.3**	**22.7**
300	3000	70.9	**23.9**	**20.9**
300	5000	66.7	**18.2**	**17.4**
500	500	522	149.1	**135.4**
500	1000	478.8	137.8	**122.1**
500	2000	354.3	109.9	**98.7**
500	5000	217.5	**91.3**	**90.7**
500	10000	191.1	**60.1**	**60.4**
800	1000	1810.5	498.8	**444.9**
800	2000	1560.5	516.6	**474.7**
800	5000	955.7	413.7	**407.5**
800	10000	653.6	**249.9**	**249.7**
1000	1000	3481.4	931.2	**832.8**
1000	5000	1995.4	**832.6**	**827.4**
1000	10000	1250.5	**546.5**	**548.6**
1000	15000	977.7	**398.7**	**398.9**
1000	20000	817.3	**322.8**	**325.5**

4.9 Summary

There have been very few attempts to apply metaheuristic algorithms to dominating set problems of minimum cardinality. We have implemented a Hybrid steady-state Genetic Algorithm (HGA) with binary tournament selection and fitness-based crossover and bit-flip mutation for both MDS and MWDS problems. Each member generated is improved by applying a minimization procedure to reduce cardinality/weight by removing any redundant nodes in the member. We also implemented an ACO algorithm that uses a state-transition rule based on pheromone concentration on the nodes and combines it with a local search step of minimization used by the HGA. A pre-processing step introduced in ACO for MWDS did not improve the solution but reduced the time taken to compute solutions.

Our algorithms and the greedy heuristics outperform the only metaheuristic algorithms proposed in literature. Our metaheuristics perform $\simeq 5 - 15\%$ better than the heuristics in most instances. The fact that the greedy heuristic for MDS is also the optimal approximation algorithm shows that the metaheuristics are superior to any algorithm that may be proposed for MDS. Therefore, where minimization of cardinality is crucial, the metaheuristics can play a significant role. We found that there is no statistically significant difference between the solutions returned by both our metaheuristic algorithms. The ACO algorithm runs twice as fast as the HGA for large graphs. Therefore it is the better algorithm for practical applications involving large graphs.

Chapter 5

Greedy Heuristics and Metaheuristic Algorithms for CAPMDS

"Well, in our country," said Alice, still panting a little, "you'd generally get to somewhere else – if you ran very fast for a long time, as we've been doing."
"A slow sort of country!" said the Queen. "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"
Through the Looking Glass by Lewis Carroll

Minimum Capacitated Dominating Set (CAPMDS) problem is nothing but what Bar-Ilan et al. call the ρ-dominating set in [5]. As reviewed in Section 2.3.1.3, they propose an approximation algorithm with an approximation ratio of $\ln n$ for graphs with uniform capacity. A distributed approximation algorithm has also been defined for this problem [60] and is also reviewed in Section 2.3.1.3. There have been no metaheuristic algorithms proposed for the CAPMDS problem in literature.

In this chapter, we propose a greedy heuristic and a couple of variants for the CAPMDS problem. The variants are similar to the algorithm proposed in [60]. We also propose metaheuristic algorithms for computing CAPMDS. We experimented extensively with both UDG and general graph instances using both uniform and variable capacity on the nodes.

The chapter is organized as follows: we present the greedy heuristic and its variants in Section 5.1. The metaheuristic algorithms are described in Sections 5.2 and 5.3. We describe our experimental set up and discussion of results in Section 5.4 and end with the summary.

5.1 Proposed Greedy Heuristic and its variants

In this section, we describe three variants of a heuristic for the computation of a capacitated dominating set. These heuristics work for both uniform as well as variable capacity graphs.

5.1.1 Maximum Coverage-Lowest Degree Heuristic

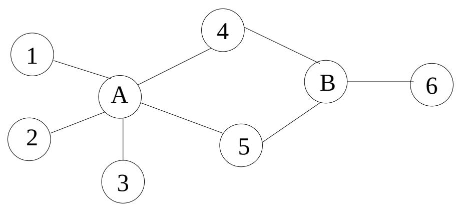
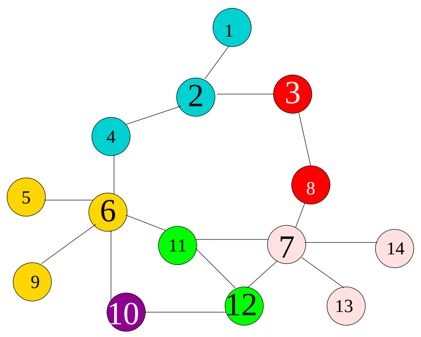


Figure 5.1: Example Topology to illustrate the advantage of choosing a node with lower absolute degree as the dominating node when two nodes have the same $min(c_u, d_u)$ value, where c_u and d_u are the capacity and WHITE degree of node u respectively.

This heuristic is based on the standard heuristic used for the Minimum Dominating Set problem, which is also the optimal approximation algorithm as described in Section 4.1. Our algorithm works as follows: the node, v, with the maximum number of WHITE neighboring nodes that can be covered, d_v, given its capacity, c_v, is determined. This node is added to the dominating set. The $min(d_v, c_v)$ nodes that are covered by this dominating node are chosen randomly from its set of neighbors and colored GREY. This process is repeated until there are no WHITE nodes in the graph.

$$v \leftarrow \underset{u \in V}{\text{argmax}}\ min(c_u, d_u) \tag{5.1}$$

If two nodes have the same number of WHITE nodes that can be covered given their capacities, we can break the tie in many ways: choosing the first node found or choosing the node with the absolute lower degree are two such methods. We experimented with



CapMDS = {2, 3, 6, 7, 10, 12}

Figure 5.2: Example Topology with the CapMDS formed by the MC-LDEG algorithm.

both these methods on various graphs. We found that in general graphs, as the size of the graph increases, the difference in cardinality between these two schemes becomes as large as 50-150 nodes with the latter giving better results. The reason for this difference can be explained as follows: consider the following scenario given in Fig. 5.1. Let each node have a uniform capacity of 3 (excluding itself) and all nodes are WHITE. Nodes A and B have the maximum number of WHITE nodes that can be covered given the capacity. If we choose A as the dominating node, we have a choice of covering nodes 4 and 5 or excluding them. If nodes 4 and 5 are included in the cluster of node A, then, we will need to include 3 more nodes in the dominating set – either B or 6, and the remaining two neighbors of A, say, 1 and 2. On the other hand, if B is chosen as the dominating node, it covers nodes 4, 5 and 6. Now, A will cover nodes 1, 2 and 3. Thus, the cardinality of the dominating set is two, whereas in the other case, it would have been four. We break the tie in the heuristic presented here by selecting the node with the lower absolute degree.

An example topology with the solution returned by this algorithm is given in Fig. 5.2 when the nodes have a uniform capacity = 2. Nodes 2, 3, 4, 6, 7, 8, 10, 11, 12 all have 2 WHITE neighbors in the beginning and the capacity = 2. Thus, any of them can be included in the dominating set. However, of these the nodes with lowest absolute degree are 3, 4, 8, 10. Let us assume that it is the node with the lowest ID that is chosen as the dominating node to break the tie this time. Thus, 3 becomes part of the dominating set. Its neighbors, 2 and 8 are colored GREY. At this point

Algorithm 10: ***Maximum-Coverage Lowest-Degree Heuristic (MC-LDEG)***$(G = (V, E))$

```
$\overline{D} := \phi$
$maxCap := 0$
$maxDegree := 0$
$s := -1$
while $V \neq \phi$ do
  $v := \underset{u \in V}{\text{argmax}} \min(c_u, d_u)$
  if
  $((maxCap < \min(c_v, d_v))$ or $((maxCap = \min(c_v, d_v))$ and $(d_v < maxDegree)))$
  then
    $maxCap := \min(c_v, d_v)$
    $maxDegree := d_v$
    $s := v$
  end if
  $\overline{D} := \overline{D} \cup \{v\}$
  $V := V \setminus s$
end while
return $\overline{D}$
```

of time, the nodes which have $\geq$ 2 WHITE neighbors are 2, 6, 7, 10, 11, 12. The algorithm is repeated in this fashion until there are no WHITE nodes in the graph. The final dominating set is shown with each cluster in a different color (shade) and the dominating nodes in a larger font in Fig. 5.2.

The pseudo-code for this heuristic is given in Algorithm 10. In the rest of the thesis, we refer to this heuristic as MC-LDEG heuristic. The worst-case time complexity of the MC-LDEG algorithm is $O(V^2)$ since in each *while* loop, we need to find the node which covers the maximum number of $\{min(d_u, c_u), u \in V\}$ nodes. This takes $O(V)$ time and the number of iterations in the worst case is $O(V)$. This algorithm has a better time complexity compared to the algorithm in [5], which we show to have a complexity of $O(V^5)$ in Section 2.3.1.3.

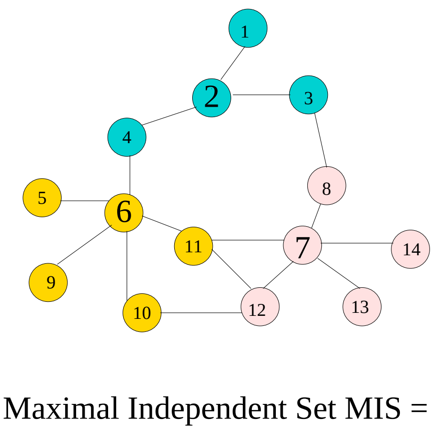
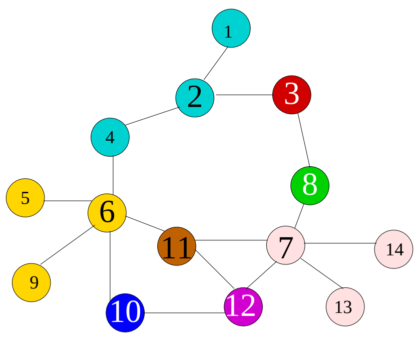


Figure 5.3: Example Topology with disjoint clusters formed using MIS. The MIS nodes are numbered in a bigger font.

5.1.2 Cluster-Subdivision Heuristic

This heuristic is based on the distributed approximation algorithm proposed for Bounded-Independence Graphs (BIGs) with uniform capacity in [60] and described in Section 2.3.1.3. In this heuristic, we first calculate the Maximal Independent Set (MIS) of the given graph which is nothing but a dominating set of the graph exactly as in [60]. The non-dominating nodes in the graph are assigned to different dominating nodes to form clusters, (u_i, C_{u_i}), that are mutually disjoint. u_i is the center of the cluster and C_{u_i} is the set of neighbors of u_i which are part of its cluster. The clusters for the example topology are shown in Fig. 5.3, with the clusters being {2, {1, 3, 4}}, {6, {5, 9, 10, 11}} and {7, {8, 12, 13, 14}}. If the capacity of a dominating node is not exceeded, the cluster is retained as it is and the dominating node is retained in the final capacitated dominating set. If the capacity of the dominating node is exceeded, the MC-LDEG algorithm is run on the cluster, i.e., on the subgraph induced by the dominating node and the nodes covered by it. Thus, we sub-divide the cluster into further clusters such that the capacity is not exceeded for any dominating node in the sub-clusters. We call this the C-SUBD heuristic in the rest of this thesis. The pseudo-code for this heuristic is given in Algorithm 11. The result of running this heuristic on the topology of Fig. 5.3 is shown in Fig. 5.4. Since the nodes in each cluster are not connected to each other except through the cluster head, we end up with a large capacitated dominating set in this case.



CapMDS = {2, 6, 7, 3, 8, 10, 11, 12}

Figure 5.4: Running algorithm of C-SUBD now gives a dominating set whose members are numbered in a bigger font.

Analysis: The MIS has a time complexity of $O(V^2)$. If $|I| = |V|$, each cluster has only one node and therefore the **for** loop takes $O(V)$ time. If $|I| = 1$, all nodes of the graph form a single cluster. The time complexity of the **for** loop in this case is $O(V^2)$ as MC-LDEG runs on the given graph. Therefore, in the worst case the running time for this algorithm is $O(V^2)$.

Algorithm 11: ***Cluster-Subdivision Heuristic (C-SUBD)***$(G = (V, E))$

```
$\overline{D} := \phi$
Compute MIS, $I$ of $G$ and form disjoint Clusters $(v, C_v)$ where $v$ is the
dominating node and $C_v$ is the set of nodes that are dominated by $v$.
for all $v \in I$ do
  if $\mid C_v \mid > cap(v)$ then
    $\overline{D} := \overline{D} \cup$ MC-LDEG$(C_v)$
  else
    $\overline{D} := \overline{D} \cup v$
  end if
end for
```

5.1.3 Relax MIS Heuristic

This heuristic is similar to the C-SUBD heuristic, but differs in the sub-division of a cluster when the capacity of the dominating node is exceeded. As in C-SUBD heuristic,

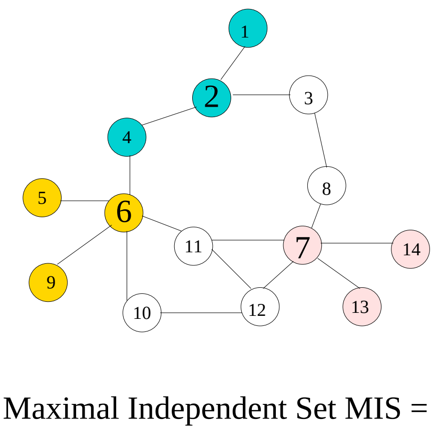


Figure 5.5: The excess nodes of the members of MIS are deleted from the clusters formed and colored WHITE.

we find the MIS of the graph initially. Then, unlike in C-SUBD where each cluster is further sub-divided, we remove the excess nodes covered by any dominating node from its cluster by changing their color back to WHITE as shown in Fig. 5.5. The nodes to be removed from the cluster are selected *randomly.* We, now, run the MC-LDEG algorithm on the entire graph until there are no WHITE nodes in the graph. We call this heuristic the DS-RELAX heuristic in the rest of the thesis. The pseudo-code for this is given in Algorithm 12.

Algorithm 12: ***Relax MIS Heuristic (DS-RELAX)**($G = (V, E)$)*

```
$I := \phi$
Compute MIS, $I$ of $G$ and form Clusters $(v, C_v)$ where $v$ is the dominating
node and $C_v$ is the set of nodes that are dominated by $v$.
for all $v \in I$ do
  if $\mid C_v \mid > cap(v)$ then
    Mark excess nodes of $v$ WHITE
  end if
end for
$I := I \cup$ MC-LDEG($G$)
```

This heuristic differs from C-SUBD heuristic in that, if there are excess nodes in two adjacent clusters which are neighbors of each other, they will be able to form a cluster together. In C-SUBD, such nodes could not combine to form a single cluster. This can

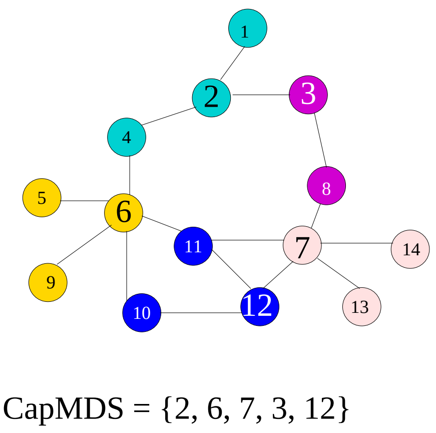


Figure 5.6: Running algorithm DS-RELAX now gives a dominating set whose members are numbered in a bigger font.

be seen in Fig. 5.6. Nodes 10, 11, 12 are removed from their respective clusters whose centers are 6, 7. These are adjacent to each other and can form a single cluster of their own without violating the capacity constraint. This is what happens in DS-RELAX. However, in C-SUBD, since they are part of different clusters, they cannot combine into a single cluster. The same is true for nodes 3, 8.

Analysis: The MIS of the graph takes $O(V^2)$ time. If $|I| = |V|$, then, each cluster has only one node. The **for** loop is executed $|V|$ times and it has $O(1)$ complexity. If $|I| = 1$, all nodes in the same cluster. Thus, marking excess nodes WHITE takes $O(V)$ time. The final step where we run MC-LDEG(G) has a complexity of $O(V^2)$. Therefore, the total time complexity of this algorithm is $O(V^2)$.

It can be readily seen from the experimental results of these three heuristics, in Sections 5.4.1 and 5.4.2 that MC-LDEG is superior to the other two variants for most instances of graphs studied. The metaheuristics can be improved with a heuristic but this needs to be done without a prohibitively high computational cost. Secondly, a deterministic algorithm cannot be used for pre-processing. Thirdly, the algorithm needs to work for graphs with uniform and variable capacity. None of these properties are satisfied by the algorithm in [5]. Thus, we use MC-LDEG for the pre-processing step in the metaheuristic algorithms to improve the cardinality returned by them.

5.2 Hybrid Genetic Algorithm (HGA-CAPMDS)

The hybrid genetic algorithm proposed for CAPMDS is very similar to the algorithm for MDS and MWDS described in Section 4.5 and uses steady-state population replacement with fitness based crossover of parents selected via the binary tournament. There are some slight differences in the way the initial population is generated. In addition, due to capacity constraints on the dominating nodes, the algorithm to determine the nodes covered by the dominating node, redundant node definition and minimization algorithm are all different for CAPMDS. We discuss only these algorithms in this chapter.

Table 5.1: Comparison of the cardinality returned by HGA-CAPMDS algorithm with no seeding (Seed=0.0), with Seeding of 20% (Seed=0.2) and Seeding of 100% (Seed=1.0) and MC-LDEG for some of the general graph instances. The capacity of each node is represented by *cap* and the values compared are 2 and average degree = α

N	Edges	cap	Seed=0.0	Seed=0.2	Seed=1.0	MC-LDEG
250	250	2	92	91	91	109
250	500	2	94	94	93	96
250	500	α	70	70	69	77
500	500	2	185	186	187	218
500	2000	2	191	188	186	197
500	2000	α	90	86	86	89

5.2.1 Initial Population Generation

We create the initial population in two ways. We use the concept of seeding 20% of the initial population with the solutions generated by the heuristic MC-LDEG described in Section 5.1.1. As seen from Table 5.1, for large graphs with higher average degree and large capacity, the solution generated using the GA does not even match that returned by the MC-LDEG heuristic unless we use seeding. We can explain this by the fact that as the degree increases without increase in capacity, there are many ways of selecting the neighbors that are covered. Since there are many feasible solutions to the CAPMDS problem, if we create initial population only by random selection of nodes, the cardinality of the initial population may be large. Thus, the crossover is done over parents which are not good candidates. By using seeding, we are creating population members which are better than randomly generated members in terms of cardinality in most cases. Using such solutions for crossover will help us in minimization since the initial solutions are themselves quite good. Hence, seeding leads to improvement in final solution using the GA. The MC-LDEG algorithm returns a different dominating set each time it runs because we are selecting the neighbors covered by a dominating node randomly. Thus, even if the same node is part of the dominating set, a different set of neighbors may be dominated in different calls to the same algorithm. Therefore, we get many unique solutions using the same algorithm. We use this property of the MC-LDEG algorithm to seed the HGA-CAPMDS. The remaining 80% of the population is created by randomly selecting nodes with a probability of 0.5. If the random bit vector so created is not a dominating set, we add more nodes to the dominating set using the MC-LDEG heuristic over the remaining WHITE nodes of the graph with probability p_h. In other cases, we keep adding nodes randomly to the bit vector until the member is a dominating set.

Algorithm 13: **CoverAdjacentNodes**($G = (V, E), u$)

```
$W := \{w : (u, w) \in E$ and $color(w) = WHITE)\}$
$GB := \{v : (u, v) \in E$ and $(color(v) = GREY$ or $color(v) = BLACK)\}$
while $(c_u \neq 0)$ and $W \neq \phi)$ do
  color($w \in W$) := GREY
  $c_u := c_u - 1$
  $r_w := r_w + 1$
  $W := W \setminus w$
end while
while $(c_u \neq 0)$ and $GB \neq \phi)$ do
  $c_u := c_u - 1$
  $r_v := r_v + 1$ /* $(v \in GB)$ */
  $GB := GB \setminus v$
end while
return
```

5.2.2 Neighborhood Coverage Algorithm

Whenever a node, u, is chosen to be part of the dominating set, it can cover $min(d_u, c_u)$ nodes, where d_u is the WHITE degree of the node and c_u is the capacity of the node.

If $d_u > c_u$, the actual nodes to be covered are chosen randomly in our metaheuristic algorithms. If $d_u < c_u$, the node u has excess capacity which remains unutilized. In such cases, we utilize the residual capacity of node u to cover other already covered neighbors redundantly.

We represent this by using a reference count for each node v, r_v. If $r_v > 1$, then, this node is redundantly covered. This is shown in Algorithm 13. The purpose of this redundant coverage is that we will be able to minimize the cardinality by removing redundant dominating nodes after the construction of the dominating set is complete.

Algorithm 14: **Minimize**(G, C)

```
$R := \{v : (v \in D)$ and $(A[v] \subset A[D \setminus v])\}$
while $(R \neq \phi)$ do
  if $(p < p_r)$ then
    $v := random(R)$
  else
    $U := \phi$
    $U := U \cup \operatorname{argmax}_{u \in R} rc_u$
    if $| U | \geq 1$ then
      $v := \operatorname{argmin}_{u \in U} d_u$
    else
      $v := \operatorname{argmin}_{u \in R} d_u$
    end if
  end if
  $C := C \setminus v$
  recalculate $R$
end while
return $C$
```

5.2.3 Minimization Heuristic

We define a dominating node to be redundant if the node and all its *covered* neighbors, $A[u] \subseteq N[u]$, are covered by other dominating nodes with their given capacity. This is different from MDS and MWDS where we look for the entire neighborhood to be covered by other dominating nodes. The minimization heuristic, as usual, calculates the set of redundant nodes, if any.

When removing redundant nodes from the generated member, the node to be removed is chosen randomly with a probability of p_r. In other cases, we use a heuristic that determines the redundant node with the maximum residual capacity even after covering all its neighbors. Such a node has a very small degree and an excess capacity and hence should be removed first. If the residual capacity of two nodes is the same, then it is better to remove the node with the lower absolute degree from the dominating set. If there is no node with any residual capacity, then, we remove the node with the lowest absolute degree. After the removal of a redundant node, we re-calculate the set of redundant nodes once again. We repeat this until there are no more redundant dominating nodes. This is shown in Algorithm 14. The variable rc_u in this algorithm is the *residual capacity* of the node.

5.3 Hybrid Ant Colony Optimization Algorithm (ACO-LS-CAPMDS)

The ACO-LS-CAPMDS algorithm for CAPMDS is similar to that used in computing MDS and MWDS. For CAPMDS, we experimented to see which state-transition rule would be more effective for the minimization of the capacitated dominating set. In the next section, we describe the various state-transition rules considered and present results of these experiments. We also studied the effect of using the pre-processing step and found that this is essential in CAPMDS in some of the instances where the search space is large. The discussion regarding use of pre-processing for improvement of cardinality is given in Section 5.3.3. In Section 5.3.2, the pheromone updation rule is specified. The final ACO-LS-CAPMDS algorithm is given in Section 5.3.4.

5.3.1 State Transition Rule

The probabilistic state-transition rule in ACO assigns the probability p_i^k, with which an ant k chooses a node i out of all the nodes A_k accessible to that ant. There are many state-transition rules that have been proposed in the literature and reviewed in Section 2.2.2. We give below some of the methods we have tested and the results thereof are presented in Table 5.2.

In equation 5.2 we use only the pheromone deposit, τ_i of node i, to determine which node is to be added next to the dominating set. In equation 5.3, we use a probability q_0 with which we always select the node with the maximum pheromone deposit, as done in the Ant Colony System. For the rest of the cases, we choose the node based on the proportional contribution of this node to the total pheromone deposit in the system.

$$p_i^k = \frac{\tau_i}{\sum_{i \in A_k} \tau_i} \tag{5.2}$$

$$p_i^k = \begin{cases} 1 & \text{if } q < q_0 \text{ and } i = \text{argmax}_{j \in A_k} \tau_j \\ 0 & \text{if } q < q_0 \text{ and } i \neq \text{argmax}_{j \in A_k} \tau_j \\ \frac{\tau_i}{\sum_{i \in A_k} \tau_i} & \text{if } q \geq q_0 \end{cases} \tag{5.3}$$

Equation 5.4 is similar to equation 5.2 except that it adds a heuristic component, η_i for node i, to the transition rule. The heuristic component used by us for the computation of capacitated dominating set is $min(c_i, d_i)$ where c_i is the capacity and d_i is the WHITE degree of the node i. The node with the maximum such value is more likely to be selected to be part of the dominating set than a node with a smaller value. The parameters α and β in equation 5.4 control the relative significance of the pheromone and heuristic components. Similar to the equation 5.3, we select the node with the maximum value in equation 5.5 with a probability of q_0. In other cases, the node is chosen based on the proportional contribution of the node to the total pheromone and heuristic components of the system.

$$p_i^k = \frac{\tau_i^\alpha \eta_i^\beta}{\sum_{i \in A_k} \tau_i^\alpha \eta_i^\beta} \tag{5.4}$$

$$p_i^k = \begin{cases} 1 & \text{if } q < q_0 \text{ and } i = \text{argmax}_{j \in A_k} (\tau_j^\alpha \eta_j^\beta) \\ 0 & \text{if } q < q_0 \text{ and } i \neq \text{argmax}_{j \in A_k} (\tau_j^\alpha \eta_j^\beta) \\ \frac{\tau_i^\alpha \eta_i^\beta}{\sum_{i \in A_k} \tau_i^\alpha \eta_i^\beta} & \text{if } q \geq q_0 \end{cases} \tag{5.5}$$

The results of using each of these state-transition rules for computing the CAPMDS of the graphs are presented in Table 5.2. The column titled $PP - ITER$ represents a pre-processing step that has been applied to improve the solution returned by the proposed ACO-LS-CAPMDS algorithm and is described in detail in Section 5.3.3. Here, we only discuss the results of using each of the equations 5.2-5.5.

Table 5.2: Cardinality (γ) and Time taken in seconds for computing CAPMDS using different state-transition rules. When $\eta = F$ and $q_0 = 0$, it is as per Eqn. 5.2, whereas with $\eta = T$ and $q_0 = 0$, it as per Eqn. 5.4. When $\eta = F$ and $q_0 > 0$, it is as per Eqn. 5.3 whereas $\eta = T$ and $q_0 > 0$, it is as per Eqn. 5.5

N	Edges	η	PP-ITER	q_0	γ	Time(s)
250	250	F	0	0.8	89	22.75
		F	0	0.3	88.25	35
		F	250	0.8	91.75	27.5
		F	250	0.3	88.75	34.5
		F	250	0.0	88.5	39.25
		T	0	0.8	90.25	23
		T	0	0.3	89.75	32
		T	0	0.0	91.75	50.25
		T	250	0.8	99	27.75
		T	250	0.3	96.75	33.75
		T	250	0.0	92.5	80.25
500	500	F	0	0.8	180.5	81.75
		F	0	0.3	178.25	127
		F	250	0.8	188	106
		F	250	0.3	178.25	140.75
		F	250	0.0	176	172
		T	0	0.8	184	84.5
		T	0	0.3	181.5	112
		T	0	0.0	191.5	211.75
		T	250	0.8	198.25	110.25
		T	250	0.3	194.25	131
		T	250	0.0	190	372.25

We use values of $\alpha = \beta = 1$ in equations 5.4 and 5.5. We experimented with different values such as $\alpha = 2, \beta = 1$, $\alpha = 1, \beta = 2$ and $\alpha = 4, \beta = 2$. The results did not vary significantly or were worse than those presented here. The column η in Table 5.2 has value F to represent that the heuristic was not used in the state-transition rule, corresponding to the equations 5.2 and 5.3. When the value in this column is T, it represents the usage of the heuristic component as in equations 5.4 and 5.5. We can see from Table 5.2 that the algorithm works best when using only the pheromone rather than when using the heuristic along with the pheromone. We can also see that using q_0 probability for selecting the node with the maximum τ or $\tau^\alpha \eta^\beta$ value is actually leading to a worse solution many times. It gets worse with the increase in the value of q_0. Therefore, we chose to use the simple state-transition rule of equation 5.2 in our proposed hybrid ACO-LS-CAPMDS algorithm.

5.3.2 Pheromone Updating Rule

The dominating set with the best cardinality in an iteration is used to update the pheromone values of the nodes in the dominating set. The formula used to update the pheromone for nodes which are part of the best solution is given below:

$$\tau_i = \tau_i \times \rho + \frac{2.0}{1.0 + f - F} \tag{5.6}$$

τ_i is the pheromone deposit on node i, ρ represents the pheromone persistence factor, f is the fitness of the best solution in the current iteration and F is the fitness of the best solution so far. For nodes which are not part of the solution, we just evaporate the pheromone value on that node by using the persistence factor as follows:

$$\tau_i = \tau_i \times \rho \tag{5.7}$$

If the pheromone value of any node reduces below a specified minimum pheromone value, τ_{min}, the pheromone value is set to τ_{min} to ensure that this node still has a small probability of getting selected. We do not use any explicit maximum value on the pheromone value.

5.3.3 Improvement using Pre-processing

With the above algorithm, the solutions returned were still not good enough for those cases where the degree is large and capacity is low. There are too many feasible solutions, depending on which neighbors are covered by a dominating node. Secondly, the time taken to compute the solution was also high. We felt that, unlike in MWDS, the solution can be improved by using a pre-processing step. The pre-processing step is to generate solutions using the MC-LDEG algorithm and use these to initialize the pheromone trails to bias the ants towards nodes selected by this heuristic.

Therefore, we tested the ACO-LS-CAPMDS algorithm with and without the pre-processing step. Some of the results are presented in Table 5.3 as well as in Table

Table 5.3: Comparison of the cardinality returned by ACO-LS-CAPMDS algorithm with no pre-procesing (PP-ITER=0), with pre-processing (PP-ITER=250) and MC-LDEG for some of the general graph instances. Capacity of the node is represented by *cap* and values compared are 2 and average degree = α

N	Edges	cap	PP-ITER=0	PP-ITER=250	MC-LDEG
250	250	2	88	89	109
250	500	2	96	94	96
250	500	α	68	66	77
500	500	2	180	176	218
500	2000	2	199	191	197
500	2000	α	91	86	89

5.2. The column called $PP-ITER$ in Table 5.2 shows whether pre-processing was used or not. If it is 0, it was not used and in the other case, we used a total of 250 solutions generated using the MC-LDEG heuristic for pre-processing. We can observe from Table 5.3, that without pre-processing, the solution returned by the ACO-LS-CAPMDS is worse than that returned by MC-LDEG for large graphs with higher degree of connectivity. In general, the solution is improved by using pre-processing as can readily be observed from Tables 5.3 and 5.2. Based on these observations, we introduced the pre-processing step in the ACO-LS-CAPMDS with 250 solutions generated using MC-LDEG. These solutions are used to update the pheromone on the nodes of the solution using the updation formula in Eqn. 5.6. We also found that with the introduction of pre-processing, the time taken for finding the solution is reduced as also reported in [92]. Thus, all results reported for ACO-LS-CAPMDS are with pre-processing included.

5.3.4 Proposed Hybrid ACO-LS-CAPMDS Algorithm

The final ACO-LS-CAPMDS algorithm is given in Algorithm 15. It uses the same node coverage algorithm of Section 5.2.2. The local search component used is the minimization heuristic described in Section 5.2.3. As was the case with MDS and MWDS problems, the local search component was very critical to reduce the cardinality of the solution returned.

Algorithm 15: **ACO-LS-CAPMDS($G = (V, E), \overline{D}$)**

```
Initialize Pheromone levels $\tau_0$ on all nodes
$F :=\mid V \mid$
for $i = 1$ to $PP\_ITER$ do
  $S := MC\_LDEG(G)$
  $UpdatePheromone(S)$
  if $\mid S \mid< F$ then
    $F :=\mid S \mid$
    $\overline{D} := S$
  end if
end for
for $iter = 1$ to $ITER$ do
  $f :=\mid V \mid$
  $D := \phi$
  for $j = 1$ to $N_{ants}$ do
    $S := \phi$
    while $S \neq DominatingSet$ do
      For each node $n$, calculate $p_n = \frac{\tau_n}{\sum_{n \in A_j} \tau_n}$
      Generate $p$
      if $p < p_n$ for some node $n$ then
        $S := S \cup n$
        $CoverAdjacentNodes(G, n)$
      end if
    end while
    $Minimize(G, S)$
    if $\mid S \mid< f$ then
      $f :=\mid S \mid$
      $D := S$
    end if
  end for
  if $(f < F)$ then
    $F := f$
    $\overline{D} := D$
  end if
  $UpdatePheromone(D)$
end for
return $\overline{D}$
```

5.4 Experimentation and Discussion of Results

We have done extensive experimentation with both Unit Disk Graphs (UDGs) generated using the topology generator provided in [68] and general graph instances which have been taken from the Type I instances in [57]. Both have been modified to have capacity for nodes. We experimented with six different graph sizes - 50, 100, 250, 500, 800 and 1000 nodes. In the case of UDGs, we used two different ranges - 150 and 200 units - in an area of 1000×1000 units to study the effect of degree of connectivity on the solutions. In the case of general graph instances, we experimented with different number of edges for each graph size. For every nodes/range combination in UDGs and nodes/edges combination in general graphs, the results presented are an average of the runs of the algorithm over 10 different instances.

Capacitated dominating set finds use in wireless networks where the bandwidth may be the capacity of a node. The ratio of bandwidth of 802.11 (WiFi) interface to that of 802.16 (WiMAX) interface or a 3G/4G mobile interface is approximately 2:5. In addition, many of the instances have a maximum degree of 2 or 5. Hence, we have chosen the ratio of 2:5 for variable capacity of nodes in a graph. We use (2, 5) or $(\alpha/5, \alpha/2)$ as the capacities of the nodes, where α is the average degree of the graph. We also tested our algorithms assuming that the capacity can be a value in the interval $[1..\alpha]$ as an additional test case. In the case of uniform capacity, we generated three different uniform capacities, viz., 2, 5 and average degree reperesented by α. If the capacity exceeds maximum degree, the CAPMDS problem reduces to MDS. This is another reason for the capacities chosen in our experiments.

HGA-CAPMDS Parameters: For the HGA-CAPMDS, we used the following parameters: the probability used for random selection of a node during initial population generation is 0.5, crossover probability $p_c = 0.9$, probability of selecting the better parent after binary tournament selection $p_{better} = 0.8$, probability of using the heuristic to repair a random population member to be a dominating set is $p_h = 0.6$, the probability with which we minimize the dominating set by removing redundant nodes randomly is $p_r = 0.6$. In all other cases, we use a heuristic to remove the redundant nodes as described in Section 5.2.3. Seeding of the initial population with the solutions from the heuristic is 20%. The mutation probability, $p_m = 0.01$ for uniform capacity experiments and $p_m = 0.002$ for variable capacity experiments. We ran the HGA-CAPMDS for 10,000 generations.

ACO-LS-CAPMDS parameters: For the ACO-LS-CAPMDS, we use 20 ants and 1000 iterations, for a total of 20,000 solutions. We use pre-processing with 250 solutions from the MC-LDEG heuristic. Pheromone persistence factor is set to $\rho = 0.985$ for uniform capacity experiments and $\rho = 0.995$ for variable capacity experiments. Minimum pheromone value for every node is set to $\tau_{min} = 0.08$. The minimization procedure uses random removal of redundant nodes with $p_r = 0.4$ in variable capacity experiments and $p_r = 0.6$ for uniform capacity experiments. We also use one ant solution in every iteration to be the MC-LDEG solution for variable capacity experiments. No such ant is used in uniform capacity experiments.

We present the results of applying all the heuristics, described in this chapter, in tables 5.4 – 5.7. The comparison of the best heuristic (MC-LDEG) with the HGA-CAPMDS and ACO-LS-CAPMDS algorithms is given in tables 5.8 – 5.11.

5.4.1 Uniform Capacity Results for Heuristics

Table 5.4 lists the results of the heuristics for uniform capacity of 2, 5 and average degree (α) on UDG graphs. Table 5.5 lists the results for similar experimentation on general graph instances. Given below are some of the observations on the results obtained.

1. In UDGs with node capacity = 2, the C-SUBD heuristic performs better than both the DS-RELAX and MC-LDEG heuristics as the graph size increases. Also, the difference in the cardinality increases as the degree of connectivity increases. DS-RELAX heuristic is the worst of the three for UDG instances for this capacity.

   For capacity = 5, MC-LDEG performs best of all three except for range=200 and nodes=800, 1000. Thus, when the degree of connectivity is very high, C-SUBD does slightly better than MC-LDEG for this capacity. DS-RELAX is also better than C-SUBD up to graphs of size 250 nodes. Beyond this size C-SUBD performs better than DS-RELAX.

For capacity = α, the C-SUBD heuristic performs worse than the other two heuristics in all cases. MC-LDEG is the best heuristic in this case.

2. For general graphs, all the three heuristics return the same solution when the average degree is equal to the capacity. MC-LDEG performs better than the other two heuristics for all other instances. It becomes significantly better as the graph size, degree of connectivity and capacity increase. DS-RELAX is always better than C-SUBD for general graphs. C-SUBD heuristic is significantly worse than the other two heuristics for general graphs as the size of the graph and degree of connectivity increase.

Table 5.4: Cardinality (γ) of CAPMDS using C-SUBD, DS-RELAX and MC-LDEG for UDG Instances with a uniform capacity of 2, 5, average degree=α for every node.

N	**Range**	**C-SUBD**			**DS-RELAX**			**MC-LDEG**		
		2	**5**	α	**2**	**5**	α	**2**	**5**	α
50	150	25	17.8	21.2	23.5	17.8	20.6	**21.1**	**15.6**	**17.9**
50	200	22.6	15.1	15.1	22.5	14.8	14.8	**20.4**	**12.7**	**12.7**
100	150	45.8	30.6	30.6	44.8	28.9	28.9	**41.2**	**23.3**	**23.3**
100	200	42.1	26.3	19.5	44.6	24.4	18.2	**41.4**	**21**	**14.1**
250	150	**99.7**	60.7	33.8	111.4	54.4	31.9	104	**48.4**	**25.1**
250	200	**94.2**	53.8	21.4	110.8	51	20	106.4	**47.8**	**15.3**
500	150	**184.5**	104.8	37.6	223.5	102.5	35	212.7	**95.7**	**26.3**
500	200	**178.1**	96.7	23	219.5	99	21.1	213.5	**95.4**	**15.7**
800	150	**286.1**	155.8	37.8	352.9	159.7	35.9	343.4	**152.7**	**27.4**
800	200	**278.1**	**146.3**	24.6	350.4	157.4	22.7	344.3	152	**16.3**
1000	150	**352.3**	**190.4**	39.2	438.2	199.4	36.6	428.4	**190.8**	**27.2**
1000	200	**343.9**	**180.3**	24	437.3	196.4	22.5	430.5	190.6	**17**

5.4.2 Variable Capacity Results for Heuristics

Table 5.6 lists the results obtained by the heuristics for UDGs with variable capacity of (2, 5), ($\alpha/5$, $\alpha/2$) and in the range [1..α]. Table 5.7 compares the cardinality returned by the various heuristics with variable capacity on general graph instances. Given below are some of the observations on the results obtained.

1. The MC-LDEG heuristic performs best among the three for all variable capacity instances in both UDG and general graphs.

2. When the variable capacity of nodes is (2, 5) in UDG graphs, the DS-RELAX heuristic does better than C-SUBD for graphs of sizes up to 250 nodes. With higher capacities such as $\alpha/2$ and $\alpha/5$, DS-RELAX does better for all sizes of graphs. For capacity in the closed interval $[1..\alpha]$, DS-RELAX and C-SUBD perform similarly.

3. For general graphs, with all variable capacities experimented with, the DS-RELAX heuristic performs better than C-SUBD heuristic. The difference becomes significant as the size of the graph as well as the average degree of connectivity increase.

Analysis: In C-SUBD algorithm, we run MC-LDEG in the subgraph induced by the cluster whenever the capacity, c, of the cluster center is exceeded. The cluster center, u_i, may be one of the candidates selected by the MC-LDEG algorithm. c neighbors of u_i will be part of this new cluster. If most of the other nodes in the old cluster, which are not part of this new cluster, are independent of each other, each of them becomes a cluster of its own. This can be seen in Fig.5.4. Thus, when there is no bound on the maximum independent set of the neighborhood of a node, the number of clusters can become quite high in this algorithm. Since UDGs are polynomially bounded growth graphs, whereas, general graphs are not, this algorithm works very well for UDGs and not for general graphs.

Table 5.5: Cardinality (γ) of CAPMDS using C-SUBD, DS-RELAX and MC-LDEG for General Graph Instances with a uniform capacity of 2, 5 and average degree=α for every node.

N	**Edges**	**C-SUBD**			**DS-RELAX**			**MC-LDEG**		
		2	**5**	α	**2**	**5**	α	**2**	**5**	α
50	100	26.7	17.9	18.6	23.2	17.9	18.6	**20.3**	**14.2**	**15.2**
50	250	26.8	18.4	11.2	22.3	13.8	11.2	**19.9**	**10.1**	**7**
50	500	22.3	14.1	7.4	21.7	11.6	7.1	**19.3**	**9.5**	**4.2**
100	100	**43.8**	43.4	**43.8**	**43.8**	43.4	**43.8**	**43.6**	**39.5**	**43.6**
100	250	56.3	35.6	35.6	46.3	34.9	34.9	**39.3**	**26.1**	**26.1**
100	500	60.6	36.3	24.1	44.4	27.8	23.7	**39.6**	**21.1**	**15.3**
250	250	**109**	108.6	**109**	**109**	108.6	**109**	**108.8**	**100**	**108.8**
250	500	133.5	91.4	95.6	116.8	91.3	94.8	**99.4**	**71.1**	**77.7**
250	1000	155.9	93	71.1	112.8	76.9	69.1	**98.3**	**55.8**	**45.4**
500	500	**216.4**	216.2	**216.4**	**216.4**	216.2	**216.4**	**216.5**	**201.2**	**216.5**
500	1000	271.1	185.7	195.5	237.2	185.5	192.5	**199.2**	**141.2**	**153.4**
500	2000	316.1	185.2	141.7	225.7	153.5	138.2	**197**	**109.4**	**91.2**
800	1000	380	332.5	380	374.5	332.5	374.5	**330.8**	**281.3**	**330.8**
800	2000	460.8	286.6	286.6	370.5	280.9	280.9	**317.1**	**209.4**	**209.4**
800	5000	550.4	354.9	182.4	358.4	214.7	173.9	**313**	**156.8**	**104.3**
1000	1000	**431.8**	431.4	**431.8**	**431.8**	431.4	**431.8**	**431.7**	**399.5**	**431.7**
1000	5000	670.5	412.5	255.4	450.7	287	245.5	**391.6**	**205.7**	**152.6**
1000	10000	722.4	513.3	164	444.2	238.9	157.9	**391.3**	**187.7**	**88.7**

Table 5.6: Cardinality (γ) of CAPMDS using C-SUBD, DS-RELAX and MC-LDEG for UDG Instances with a VARIABLE capacity of (2, 5), ($\alpha/5$, $\alpha/2$) and [1..α] for every node.

N	**Range**	**C-SUBD**			**DS-RELAX**			**MC-LDEG**		
		(2,5)	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**
50	150	20.8	**31.2**	23.6	20.6	33.4	24.6	**16.1**	**31**	**19.1**
50	200	16.6	23.4	17.6	17.1	24.2	18.9	**12.7**	**21.1**	**13.7**
100	150	35	43	33.3	35	41.9	34	**25.8**	**37.4**	**25.6**
100	200	28.5	28	21.9	26.9	26.7	22.2	**21.3**	**21**	**15.2**
250	150	63.8	53.5	40	59.9	48.1	40.5	**49.7**	**37.6**	**26.8**
250	200	56.4	32.5	26	55.1	30.2	26.1	**47.6**	**22.6**	**16.9**
500	150	107.5	56.6	45.5	108	52.5	46.4	**95.8**	**39.2**	**30**
500	200	99.1	34.6	27.4	102.7	32.4	27.8	**95.2**	**24.2**	**17.6**
800	150	159.6	60.2	47.4	164.2	55.7	48.3	**151.7**	**40.4**	**29.7**
800	200	**151.2**	37	28.8	158.6	33.9	29.5	**151.7**	**24.3**	**18**
1000	150	194.6	62.8	48.1	201.3	57.6	49.7	**189.2**	**41**	**29.8**
1000	200	**185.2**	35.3	29.6	196.9	33.4	28.5	189	**25.1**	**18.4**

Table 5.7: Cardinality (γ) of CAPMDS using C-SUBD, DS-RELAX and MC-LDEG for General Graph Instances with a VARIABLE capacity of (2, 5), ($\alpha/5$, $\alpha/2$) and [1..α] for the nodes.

N	**Edges**	**C-SUBD**			**DS-RELAX**			**MC-LDEG**		
		(2,5)	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**
50	100	19.8	30.1	23.4	19.5	26.1	21.5	**15.1**	**21.7**	**16.3**
50	250	20.4	21.7	16.1	14.8	16	13.2	**10.3**	**10.5**	**8.6**
50	500	15.9	12.9	8	12.2	9.9	7.4	**9.8**	**6**	**4.5**
100	100	43.5	64.4	52	43.5	**62.9**	51.5	**41.6**	63.9	**47.6**
100	250	45.3	63.8	50.3	39.4	51.9	41.7	**26.8**	**42.6**	**28.5**
100	500	49.1	49.2	42	31.1	31.6	29.2	**21.4**	**21.4**	**16.2**
250	250	108.7	161.4	132.6	108.7	**156.4**	131.6	**103.3**	**156.4**	**113.4**
250	500	115.8	155.8	127.7	106.3	130.7	114.6	**75.4**	**107.6**	**82.2**
250	1000	125.7	150.7	107	87.3	97.2	82.5	**57.6**	**65.6**	**49.5**
500	500	216.3	324.6	274.3	216.3	316.4	272.4	**209.4**	**314**	**231.2**
500	1000	234.4	310.1	268.9	215.9	264.5	237	**153.1**	**214**	**166.2**
500	2000	260.3	290.4	231.4	175.3	190.7	168.7	**115**	**131.1**	**98.6**
800	1000	357.5	543	461.1	355.8	510	441.1	**298.7**	**496.7**	**356.6**
800	2000	371.9	527.5	409.5	324.1	413.2	340.3	**222.4**	**339.6**	**228.7**
800	5000	469.6	434.8	341.4	244.1	230.6	211.1	**162.8**	**146.6**	**109.3**
1000	1000	431.7	647.3	543.1	431.7	630.5	538.5	**413**	**628.7**	**457.5**
1000	5000	540.3	539.6	444.6	320.3	319.8	297.5	**213.1**	**213.2**	**164.7**
1000	10000	615.2	440.4	354.7	261.2	203	192.3	**191.4**	**118.4**	**92.9**

5.4.3 Uniform Capacity Results for Metaheuristics

Table 5.8 lists the results of the MC-LDEG heuristic, HGA-CAPMDS and ACO-LS-CAPMDS algorithms for uniform capacity of 2, 5 and average degree (α) on UDG graphs. Table 5.9 lists the results on general graph instances. The difference in the performance between the two metaheuristics is not statistically significant. Given below are some of the observations on the results obtained.

Unit Disk Graphs Since we kept the area constant while increasing the number of nodes from 50 to 1000 nodes, when generating UDG instances, this has the effect of increasing average degree as the number of nodes increases. We also experiment with a range of 200 to increase average degree for a specific graph size and understand the effect of increasing connectivity on the solutions generated by the various algorithms.

We observed that the percentage improvement of the metaheuristics over the greedy heuristic remains constant across all graph sizes. It is $\simeq 15-18\%$ with uniform capacity of 2 and average degree (α). With capacity 5, however, the improvement drops to 3-5% for all graph sizes from 250 nodes onwards.

The constant improvement seems to be related to the fixed cardinality of the maximum independent set of the neighborhood of a node in UDGs. Every MIS is a constant approximation of the MDS in UDGs. Similarly, Kuhn and Moscibroda in [60] show that their algorithm is a constant approximation algorithm for capacitated dominating set for bounded independence graphs with uniform capacity. We can observe from table 5.8 that the cardinality of the dominating set doubles for every doubling of the graph size. Thus, for capacity 2 and range=150, the heuristic returns a cardinality of 21.1 for 50 nodes and 41.2 for 100 nodes and so on. The cardinality returned by the metaheuristics similarly doubles and therefore results in constant improvement over the heuristic.

General Graphs Interestingly, as in UDGs, the improvement of cardinality over the heuristic remains constant for all graph sizes. The average degree of a node remains constant in the instances studied. This seems to be the reason for the constant improvement in these instances. In low degree graphs, the metaheuristics once again return an improvement of $\simeq 18\%$ over the greedy heuristic as in UDGs for capacity $= 2$. When capacity $= 5$, in these graphs, the problem reduces to finding a minimum dominating set as capacity 5 is larger than the maximum degree. Thus, the greedy heuristic reduces to the optimal approximation algorithm and the improvement is only $\simeq 10\%$ in this case.

When the degree of connectivity increases, the percentage improvement over the heuristic drops to $\simeq 5-10\%$. This is true for all the capacities tested. When the capacity remains small but degree of connectivity increases, not only do we need to explore different nodes being part of the dominating set but also which neighbors are dominated by these nodes. In other words, it seems that the choice of which neighbors are dominated by the dominating node has an impact on the cardinality. Thus, we conclude that randomly choosing the neighbors that are dominated may not be the best coverage algorithm for such instances.

5.4.4 Variable Capacity Results for Metaheuristics

Table 5.10 compares the results obtained by the MC-LDEG heuristic, HGA-CAPMDS and ACO-LS-CAPMDS algorithms for UDG instances with variable capacity of (2, 5), $(\alpha/5,\ \alpha/2)$ and in the range $[1..\alpha]$. Table 5.11 compares the cardinality returned by these algorithms on general graph instances. As with uniform capacity instances, there is no statistically significant difference between the two metaheuristics.

Unit Disk Graphs We can see from the table 5.10 that the improvement in cardinality returned by the metaheuristic algorithms for variable capacity graph instances is less than in the case of uniform capacity graph instances and is $\simeq 8-12\%$ over the greedy heuristic. When using variable capacities (2, 5) for the nodes, the improvement of cardinality drops from $\simeq 12\%$ to $\simeq 2\%$ as the graph size increases.

General Graphs For low degree graphs, as in uniform capacity, the improvement in cardinality is $\simeq 15\%$ for the metaheuristic algorithms when using variable capacity of (2, 5) or $(\alpha/5,\ \alpha/2)$. The improvement is constant across all graph sizes. When using capacities in the closed interval $[1..\alpha]$, the improvement drops as the graph size increases, from $\simeq 15\%$ to $\simeq 10\%$. As with uniform capacity, as the average degree increases, the percentage improvement over the heuristic drops to between $\simeq 2-5\%$.

Table 5.8: Cardinality (γ) of CAPMDS using MC-LDEG, ACO and HGA for UDG Instances with a uniform capacity of 2, 5, average degree=α for every node.

N	**Range**	**MC-LDEG**			**ACO**			**HGA**		
		2	**5**	α	**2**	**5**	α	**2**	**5**	α
50	150	21.1	15.6	17.9	**17.7**	**13**	**14.7**	**18.1**	**13**	**14.7**
50	200	20.4	12.7	12.7	**17.5**	**10.4**	**10.4**	**17.7**	**10.5**	**10.5**
100	150	41.2	23.3	23.3	**36.6**	**19.9**	**19.9**	**36.4**	20.6	20.6
100	200	41.4	21	14.1	**36**	**18.8**	**11.9**	**35.8**	**18.7**	**11.8**
250	150	104	48.4	25.1	93.2	**46.5**	**20.6**	**92.5**	**46.4**	21.8
250	200	106.4	47.8	15.3	**92.2**	45.9	**12.7**	**91.9**	**44.9**	**13**
500	150	212.7	95.7	26.3	188.6	93.3	**21.3**	**187.4**	**91.2**	22.7
500	200	213.5	95.4	15.7	187.3	91.8	**13**	**186.5**	**90.4**	13.8
800	150	343.4	152.7	27.4	303.4	149.5	**22.1**	**301.1**	**145.4**	24
800	200	344.3	152	16.3	**301.6**	146.2	**13.3**	**301.1**	**144**	14.5
1000	150	428.4	190.8	27.2	379.5	186.4	**22.5**	**377.6**	**181.5**	23.9
1000	200	430.5	190.6	17	**377.6**	182	**13.1**	**377.3**	**179.8**	14.5

Table 5.9: Cardinality (γ) of CAPMDS using MC-LDEG, ACO and HGA for General Graph Instances. The nodes have uniform capacity of 2, 5, average degree=α.

N	**Edges**	**MC-LDEG**			**ACO**			**HGA**		
		2	**5**	α	**2**	**5**	α	**2**	**5**	α
50	100	20.3	14.2	15.2	**17.2**	**12.5**	**12.9**	**17.5**	**12.1**	**12.7**
50	250	19.9	10.1	7	**17.1**	**9**	**6.7**	**17.2**	**9.1**	**6.5**
50	500	19.3	9.5	4.2	**17**	**9**	**4**	**17**	**9**	6.5
100	100	43.6	39.5	43.6	**35.5**	**35.4**	**35.5**	**35.5**	**35**	**35.5**
100	250	39.3	26.1	26.1	**36.6**	**21.6**	**21.6**	**36**	**22.4**	**22.4**
100	500	39.6	21.1	15.3	**36.2**	**19.5**	**14.1**	**35.3**	**19.1**	**14.1**
250	250	108.8	100	108.8	**88.8**	**88.3**	**88.8**	91	89.5	91
250	500	99.4	71.1	77.7	95.5	**62.7**	**65.8**	**93.5**	64.4	68.5
250	1000	98.3	55.8	45.4	93.3	54.2	**41.3**	**92.2**	**52.7**	42.4
500	500	216.5	201.2	216.5	**176.4**	**176.3**	**176.4**	185.9	181.5	185.9
500	1000	199.2	141.2	153.4	194.8	**126**	**137.4**	**191.2**	131.9	140.2
500	2000	197	109.4	91.2	189.3	**107.7**	**87.7**	**186.9**	**107.3**	88.6
800	1000	330.8	281.3	330.8	**303.6**	**254.4**	**303.6**	**303.3**	264.9	**303.3**
800	2000	317.1	209.4	209.4	308.7	**200.7**	**200.7**	**307**	**200.8**	**200.8**
800	5000	313	156.8	104.3	301.9	**152.5**	**101.6**	**298**	154.2	**101.9**
1000	1000	431.7	399.5	431.7	**355.2**	**353**	**355.2**	377.5	370.3	377.5
1000	5000	391.6	205.7	152.6	380	**200.8**	**149.6**	**375.6**	202.3	150.8
1000	10000	391.3	187.7	88.7	378.3	**183.4**	**85.6**	**372.1**	**184.1**	**86.4**

Table 5.10: Cardinality (γ) of CAPMDS using MC-LDEG, ACO and HGA for UDG Instances with a VARIABLE capacity of (2, 5), ($\alpha/5$, $\alpha/2$) and [1..α] for the nodes.

N	**Range**	**MC-LDEG**			**ACO**			**HGA**		
		(2,5)	**($\alpha/5$,$\alpha/2$)**	**[1..α]**	**(2,5)**	**($\alpha/5$,$\alpha/2$)**	**[1..α]**	**(2,5)**	**($\alpha/5$,$\alpha/2$)**	**[1..α]**
50	150	16.1	31	19.1	**15.1**	**25.4**	**17**	**14.7**	**25.6**	**17.2**
50	200	12.7	21.1	13.7	**11.4**	**17.9**	**12.4**	**11.4**	**18.1**	**12.5**
100	150	25.8	37.4	25.6	**22.7**	**33.1**	**23**	**23.4**	**33.2**	**23.3**
100	200	21.3	21	15.2	**19.3**	**19.3**	**13.7**	**19.2**	**19.4**	**13.8**
250	150	49.7	37.6	26.8	**46.3**	**34.9**	**24.6**	**47**	**35.7**	**24.3**
250	200	47.6	22.6	16.9	**45.4**	**20.7**	**15.2**	**45.1**	**21.4**	**15**
500	150	95.8	39.2	30	**92.1**	**35.8**	**26.6**	**92.2**	37	**26.8**
500	200	95.2	24.2	17.6	**91.9**	**21.4**	**16**	**90.3**	**22.3**	**15.8**
800	150	151.7	40.4	29.7	148.2	**36.8**	**26.9**	**146.1**	38	**27.7**
800	200	151.7	24.3	18	148.4	**21.9**	**16.6**	**144.7**	23.1	**16.6**
1000	150	189.2	41	29.8	185.6	**37.5**	**27.3**	**183**	38.6	28.4
1000	200	189	25.1	18.4	185.8	**22.4**	**16.6**	**180.9**	23.1	**16.5**

Table 5.11: Cardinality (γ) of CAPMDS using MC-LDEG, ACO and HGA for General Graph Instances with a VARIABLE capacity of (2, 5), (α/5, α/2) and [1..α] for the nodes.

N	**Edges**	**MC-LDEG**			**ACO**			**HGA**		
		(2,5)	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**	**(2,5)**	**(α/5,α/2)**	**[1..α]**
50	100	15.1	21.7	16.3	**13.6**	**18.5**	**14.3**	**13.2**	**18.9**	**14.5**
50	250	10.3	10.5	8.6	**9.6**	**9.4**	**8.1**	**9.2**	**9.6**	**7.8**
50	500	9.8	6	4.5	**9**	**5.2**	**4.2**	**9**	**5.1**	**3.9**
100	100	41.6	63.9	47.6	**35.2**	**51.4**	**40.2**	**35.4**	**51.7**	**40.7**
100	250	26.8	42.6	28.5	**25.2**	**38.2**	**26.1**	**24.7**	**38.4**	**26**
100	500	21.4	21.4	**16.2**	**19.9**	**20**	**15.6**	**20**	**19.6**	**15.5**
250	250	103.3	156.4	113.4	**87.7**	**132.7**	**99.1**	90.7	**132.9**	102.6
250	500	75.4	107.6	82.2	**71.3**	**101.6**	**78.3**	**72.1**	**102**	**77.7**
250	1000	57.6	65.6	49.5	**55.7**	**61.9**	**46.8**	**56.1**	63.1	**47.4**
500	500	209.4	314	231.2	**176.9**	**270.3**	**208.2**	185.3	**270**	212.3
500	1000	153.1	214	166.2	**146.3**	**205.5**	**162.7**	**146.8**	**206.1**	**162**
500	2000	115	131.1	98.6	**111**	**125.1**	**94.3**	112.4	127.2	96.2
800	1000	298.7	496.7	356.6	**274.4**	**437.9**	343	279.4	**437.4**	**339.2**
800	2000	222.4	339.6	228.7	**219.9**	**328.6**	**222.8**	**219**	329.6	225.3
800	5000	162.8	146.6	109.3	**157.2**	**140.4**	**105.5**	159	142.7	107.1
1000	1000	413	628.7	457.5	**362.4**	**546**	**421.4**	374	**545.25**	428.3
1000	5000	213.1	213.2	164.7	**206.3**	**206.7**	**157.6**	209.5	208.3	160.3
1000	10000	191.4	118.4	92.9	**186.4**	**114**	**88.4**	188	115.5	90.3

5.4.5 Comparison of Time Taken by the Metaheuristics

The execution time of the metaheuristics is given in tables 5.12-5.15. The HGA runs for 10,000 iterations for a total of 10,000 solutions and the ACO-LS for 1000 iterations. In each iteration, 20 ants are used for a total of 20,000 solutions. Thus, ACO-LS generates twice as many solutions as HGA. The following are the observations about the time taken by the two algorithms:

Uniform Capacity: The HGA takes less time than ACO-LS in all cases for unit disk graphs. For general graphs, the HGA takes less time than ACO-LS only for smaller graphs. As the graph size increases, ACO-LS is faster than HGA. In some instances, ACO-LS is twice as fast as HGA. The pre-processing step in ACO-LS certainly contributes towards its faster running time.

Variable Capacity: For UDGs, unlike in uniform capacity, HGA takes more time than ACO-LS as the graph size increases. For general graphs, the HGA is faster than ACO-LS except for 800 and 1000 node graphs with higher degree of connectivity. Overall, the ACO-LS algorithm seems to perform better as it generates slightly better solutions than the HGA in less time for many instances. This is despite the fact that it generates twice as many solutions as HGA.

Table 5.12: Time Taken (s) to compute CAPMDS for ACO and HGA for UDG Instances with a uniform capacity of 2, 5, average degree=α for every node.

N	**Range**	**Capacity = 2**		**Capacity = 5**		**Capacity = α**	
		ACO	**HGA**	**ACO**	**HGA**	**ACO**	**HGA**
50	150	3.5	1.6	2	1.5	3.3	1.1
50	200	3.7	1.3	2.4	1.1	2.5	1
100	150	13	4.4	7.1	3.2	6.9	3.4
100	200	11.5	3.9	7.8	4	4.9	3
250	150	56	25.2	40.6	31	19.2	10.4
250	200	49	23.8	31.8	25.9	14.4	10
500	150	194.1	112.6	129.5	132.2	58	30.7
500	200	177.6	112.3	111.2	96.3	47.9	23.9
800	150	464	315.2	303.1	322.1	126.3	68.5
800	200	442.9	309.1	275.7	207.8	105.4	59.1
1000	150	704.1	520.4	454	461.6	174.6	102.9
1000	200	682.3	511.7	421.9	330.5	150.4	68.6

Table 5.13: Time Taken (s) to compute CAPMDS for ACO and HGA for General Graph Instances with a uniform capacity of 2, 5, average degree=α for every node.

N	Range	Capacity = 2		Capacity = 5		Capacity = α	
		ACO	HGA	ACO	HGA	ACO	HGA
50	100	3.5	1.4	2	1.5	2.6	1.3
50	250	3.2	1.5	2.4	1.1	1.9	1
50	500	3	1.4	2.1	1.3	1.3	1.2
100	100	7.6	5	7.1	5	7.4	5.2
100	250	12.8	5.1	8	3.7	8	3.8
100	500	11	4.6	7.8	3.9	5.4	3.6
250	250	39.1	45.2	38.6	41.7	39.1	45.4
250	500	87.1	51.2	47.5	34.6	54.8	32
250	1000	73.7	42.3	47.2	37.1	35.7	32.7
500	500	174.4	306.1	180.7	288	174.4	305.8
500	1000	458.8	354.2	246.7	258.9	281.1	219
500	2000	353.8	280.1	244.8	270.2	182.7	242.6
800	1000	1107.3	1169.6	506.6	1300	1109	1167.4
800	2000	1446.3	1397.6	852.5	1066.7	853.1	1071.4
800	5000	780.2	903.7	594.4	832.2	444.4	712.3
1000	1000	888.7	2075.8	922.6	2049	888.6	2069
1000	5000	1628.3	1906.8	1205.9	1821.5	920.1	1674.9
1000	10000	933.5	1349.4	710.4	1305.7	535.7	964.8

Table 5.14: Time Taken (s) to compute CAPMDS for ACO and HGA for UDG Instances with a variable capacity of (2, 5), ($\alpha/5$,$\alpha/2$) and [1..α].

N	**Range**	**Capacity = (2, 5)**		**Capacity = ($\alpha/5$,$\alpha/2$)**		**Capacity = [1..α]**	
		ACO	**HGA**	**ACO**	**HGA**	**ACO**	**HGA**
50	150	3	1.3	4.4	1.7	3.8	1.3
50	200	3	1.2	3.6	1.6	3	1.3
100	150	9.7	5.6	11.5	5.7	9.9	4.6
100	200	8.7	4.8	9	4.5	7.7	4.2
250	150	44.9	38.8	44.8	30.7	39.6	23
250	200	38	36.9	31.8	23.1	29.5	17.2
500	150	175.6	190.9	135.9	129.8	144.9	95
500	200	162	170.2	101.5	103.5	95.5	64.4
800	150	477	622.3	312.6	351.4	290.7	256.4
800	200	454.7	557	242.8	278.5	229.4	105.7
1000	150	780.2	1108.7	465.1	529.1	434.6	347.5
1000	200	745.8	1039.7	355.4	430.8	343.2	203.8

Table 5.15: Time Taken (s) to compute CAPMDS for ACO and HGA for General Graph Instances with a variable capacity of (2, 5), ($\alpha/5$,$\alpha/2$) and [1..α].

N	**Edges**	**Capacity = (2, 5)**		**Capacity = (α/5,α/2)**		**Capacity = [1..α]**	
		ACO	**HGA**	**ACO**	**HGA**	**ACO**	**HGA**
50	100	3.3	1.2	3.7	1.6	3.2	1.3
50	250	2.1	1.8	2.4	1.5	2.7	1
50	500	2.1	1.4	2	1.2	1.8	1
100	100	9.8	4.8	17.2	5.3	12.2	5.5
100	250	11	5	13	6	10.8	5.5
100	500	8.7	4.4	8.7	5.1	8.4	4.1
250	250	72.5	43.1	110.5	49.8	81.6	50.7
250	500	84.9	61.6	91	57	86.2	55.3
250	1000	63.7	49	63.4	48.9	63.7	43
500	500	422.3	286.6	546.5	331.7	455.5	349.5
500	1000	485.1	358.6	467.6	394.8	509.1	391.8
500	2000	341.2	319.9	332.4	318.1	332	297.3
800	1000	1586.7	1352.9	1625.7	1229.9	1731.5	1469.4
800	2000	1645.5	1510.6	1377	1380.1	1577.4	1484.6
800	5000	804.2	964.9	793.1	924.8	770.9	824.4
1000	1000	2650	2010	3065.29	2428.88	2848.7	2567.7
1000	5000	1703.4	2060.3	1681.8	1962.9	1708.2	1877.6
1000	10000	1038.2	1506.8	997.9	1183.1	959	1077.6

5.5 Summary

While there have been some approximation algorithms proposed for CAPMDS, so far, there has been no study of it using metaheuristics. Since the approximation algorithms have high computational complexity, we have proposed a greedy heuristic, MC-LDEG, for use in the hybrid metaheuristic algorithms. We have proposed variants of our heuristic, C-SUBD and DS-RELAX, along the lines of the distributed approximation algorithm proposed in [60]. A comparison of the performance of the three variants shows that the algorithm we call MC-LDEG is superior to the other two for general graph instances and UDGs with medium and high node capacities. C-SUBD is the best algorithm for UDGs when capacity is low. It is also the worst for general graph instances. This behavior can be explained based on the bounded independence property of UDGs.

We have proposed two metaheuristics for computing minimum capacitated dominating set. We use MC-LDEG to seed the GA and introduce a pre-processing step for the ACO. We find that the proposed hybrid GA and hybrid ACO algorithms benefit from this additional step. The percentage improvement of metaheuristics over MC-LDEG is constant for both uniform and variable capacity graphs across all graph sizes. The metaheuristics return a cardinality which is approximately 18% better than MC-LDEG for low degree graphs and approximately 10% for high degree graphs. As capacity increases, the problem tends to be more of finding MDS. Since the MC-LDEG heuristic is derived from the optimal approximation algorithm for MDS, it performs quite well in these instances and the metaheuristics do not improve too much on the heuristic. For the instances where the degree is large and capacity is small, the choice of the neighboring nodes that are dominated by a dominating node has a significant impact on the cardinality of the CAPMDS returned. The metaheuristics are not doing as well as expected here. We conclude that we may need to direct the search by also considering the information about coverage of the neighbors to improve the solutions. The time taken by ACO-LS is less than that for HGA in some instances despite the fact that it generates twice as many solutions as HGA.

In future, we wish to explore methods to improve the performance of the metaheuristic algorithms for the instances where they have not reduced the cardinality significantly from that returned by the greedy heuristic.

Chapter 6

Morphological Algorithms for Dominating Sets with Applications to Image Analysis

Do not fear to be eccentric in opinion, for every opinion now accepted was once eccentric.

Bertrand Russell

Images are modeled as *grid graphs* which are a special subset of *unit disk graphs*. Dominating set problems are an active area of work on UDGs, as already shown in Chapter 2. In this chapter, we explore, for the first time, the application of mathematical morphology to dominating set problems on grid graphs. We propose novel *morphological algorithms* for Minimum Dominating Sets (MDS) and Minimum Independent Dominating Sets (MIDS). We also show that dominating set problems on graphs may be used to advantage in image analysis tasks such as skeletons and clustering. We propose a new hierarchical clustering scheme based on an overlay graph of dominating sets.

The chapter is organized as follows: the graph morphological operators proposed by Cousty et al. are discussed in Section 6.1. We extend these operators to use structuring elements and define dilations and erosions using these extended operators in Section 6.2. The algorithms for MDS and MIDS are described in Section 6.3. The MDS and MIDS algorithms presented here are compared with the best heuristics on grid graphs which were generated using our grid graph topology generator. These results are presented in

Section 6.4. The application of the extended operators to distance transform is given in Section 6.5. The application of dominating sets to image analysis is discussed in Section 6.6.

6.1 Graph Morphological Operators by Cousty et al.

Cousty et al. [22] have recently defined basic operators which are used to derive a set of edges from a given set of vertices and vice versa. These are shown to be dilations and erosions. The workspace considered in this thesis, as in [22], is the complete grid graph shown in Fig. 6.1. The powerset of the vertices, edges and subgraphs of this workspace are represented by $\mathcal{G}^\bullet$, $\mathcal{G}^\times$ and $\mathcal{G}$ respectively. $\mathcal{G}$ forms the complete lattice.

Briefly, two operators are defined from $\mathcal{G}^\times \rightarrow \mathcal{G}^\bullet$ which return vertices given a set of edges and two operators are defined from $\mathcal{G}^\bullet \rightarrow \mathcal{G}^\times$ which return edges given a set of vertices. Given a graph $X = (X^\bullet, X^\times)$:

1. $\delta^\bullet(X^\times) : \mathcal{G}^\times \rightarrow \mathcal{G}^\bullet$, returns the set of all vertices that belong to the edges of the given graph.
$$\delta^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \exists e_{x,y} \in X^\times\} \tag{6.1}$$

2. $\epsilon^\times(X^\bullet) : \mathcal{G}^\bullet \rightarrow \mathcal{G}^\times$, returns the set of all edges for which both ends are in $X^\bullet$
$$\epsilon^\times(X^\bullet) = \{e_{x,y} \in \mathbb{G}^\times \mid x \in X^\bullet \textit{ and } y \in X^\bullet\} \tag{6.2}$$

3. $\epsilon^\bullet(X^\times) : \mathcal{G}^\times \rightarrow \mathcal{G}^\bullet$ returns those vertices for which all the edges of the workspace are in $X^\times$
$$\epsilon^\bullet(X^\times) = \{x \in \mathbb{G}^\bullet \mid \forall e_{x,y} \in \mathbb{G}^\times, e_{x,y} \in X^\times\} \tag{6.3}$$

4. $\delta^\times(X^\bullet) : \mathcal{G}^\bullet \rightarrow \mathcal{G}^\times$, returns all the edges for which at least one end is in $X^\bullet$
$$\delta^\times(X^\bullet) = \{e_{x,y} \in \mathbb{G}^\times \mid x \in X^\bullet \textit{ or } y \in X^\bullet\} \tag{6.4}$$

These can then be combined to form vertex dilation and erosion as well as edge dilation and erosion operators. Combining the vertex and edge operations leads to graph dilation and erosion. In addition, they define opening and closing, filters and granulometries as in classical mathematical morphology using these operators.

Vertex Dilation and Erosion: Vertex dilation is defined as $\delta = \delta^\bullet \circ \delta^\times$, where $\circ$ represents the composition of the operators and not opening. Similarly, vertex erosion is given by $\epsilon = \epsilon^\bullet \circ \epsilon^\times$.

Edge Dilation and Erosion: Edge dilation is given by $\Delta = \delta^\times \circ \delta^\bullet$ and edge erosion by $\varepsilon = \epsilon^\times \circ \epsilon^\bullet$.

Graph Dilation and Erosion: Graph dilation is obtained by composition of the vertex and edge dilation operators ($\delta \circledvee \Delta(X)$) and graph erosion by combining the vertex and edge erosion operators ($\epsilon \circledvee \varepsilon(X)$).

Opening and Closing: The opening on vertices is defined as $\gamma_1 = \delta \circ \epsilon$ and closing as $\phi_1 = \epsilon \circ \delta$. Similarly the opening on edges is given by $\Gamma_1 = \Delta \circ \varepsilon$ and closing by $\Phi_1 = \varepsilon \circ \Delta$. Composing these operators gives the opening and closing on any given graph: $\gamma_1 \circledvee \Gamma_1(X) = (\gamma_1(X^\bullet), \Gamma_1(X^\times))$ and $\phi_1 \circledvee \Phi_1(X) = (\phi_1(X^\bullet), \Phi_1(X^\times))$.

6.2 Dilations and Erosions with Structuring Elements

In this section, we extend the operators defined in the previous section to incorporate structuring elements and then, in later sections, apply them to MDS and MIDS problems. We define graph morphological operators with structuring elements on the same workspace and lattice as in [22]. Following the conventional definition, a structuring element on graphs is a *small* graph $B = (B^\bullet, B^\times)$. It may be used to identify, isolate or operate on graphs and sub-graphs that are isomorphic to the structuring element.

We define $\mathcal{B}$ to be the set of all possible 1-hop structuring elements of $\mathbb{G}$. By 1-hop, we mean that a structuring element is composed of a node and the nodes adjacent to it in the grid. Thus, there are a total of 15 1-hop structuring elements – one with four edges, 4 with three edges, 6 with two edges and 4 with a single edge. These are shown in Fig. 6.3. $\mathcal{B}$ is an ordered set of structuring elements where the structuring elements are ordered in descending order of the number of edges in them. Thus, $B_1 < B_2 \Rightarrow |B_1^\times| < |B_2^\times|$.

As in the case of morphological operators defined on binary images, we translate the origin of the structuring element to each node in $\mathbb{G}^\bullet$ when defining the erosion and dilation operators. Given a structuring element, $B \in \mathcal{B}$ and a graph $X = (X^\bullet, X^\times)$, $B_x = (B_x^\bullet, B_x^\times)$ is B translated to $x \in \mathbb{G}^\bullet$.

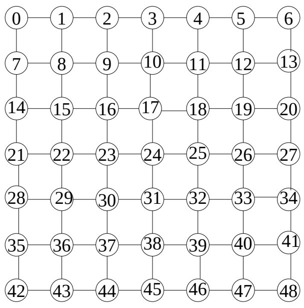


Figure 6.1: The Complete Grid, which forms the workspace, $\mathbb{G}$. The subgraphs of this grid graph form the input topologies for the algorithms presented in this thesis.

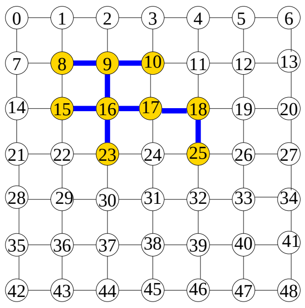


Figure 6.2: Topology to illustrate the results of application of extended morphological operators that use structuring elements of Fig. 6.3.

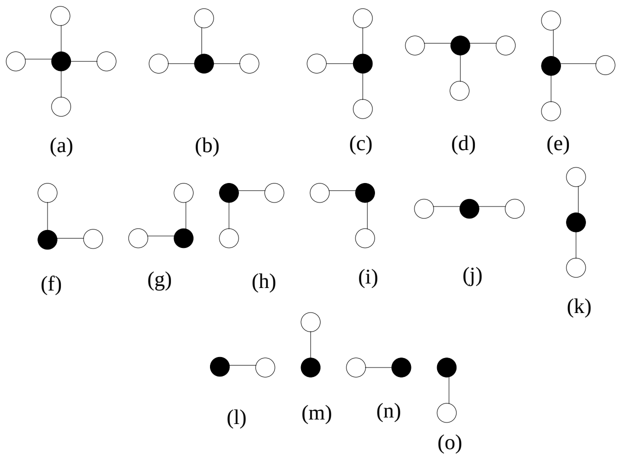


Figure 6.3: The 15 structuring elements of a grid graph in descending order of the number of edges in the structuring element. The black node represents the origin of the structuring element.

1. $\delta_B^{\bullet} : \mathcal{G}^{\times} \rightarrow \mathcal{G}^{\bullet}$ is such that:

$$\delta_B^{\bullet}(X^{\times}) = \{x \in \mathbb{G}^{\bullet} \mid \exists e_{x,y} \in (X^{\times} \cap B_x^{\times})\} \tag{6.5}$$

2. $\delta_B^{\times} : \mathcal{G}^{\bullet} \rightarrow \mathcal{G}^{\times}$ is such that:

$$\delta_B^{\times}(X^{\bullet}) = \{e_{x,y} \in B_x^{\times} \mid x \in (X^{\bullet} \cap B_x^{\bullet}) \textbf{ OR } y \in (X^{\bullet} \cap B_x^{\bullet})\} \tag{6.6}$$

3. $\epsilon_B^{\bullet} : \mathcal{G}^{\times} \rightarrow \mathcal{G}^{\bullet}$ is such that:

$$\epsilon_B^{\bullet}(X^{\times}) = \{x \in \mathbb{G}^{\bullet} \mid \forall e_{x,y} \in B_x^{\times}, e_{x,y} \in (X^{\times} \cap B_x^{\times})\} \tag{6.7}$$

4. $\epsilon_B^{\times} : \mathcal{G}^{\bullet} \rightarrow \mathcal{G}^{\times}$ is such that:

$$\epsilon_B^{\times}(X^{\bullet}) = \{e_{x,y} \in \mathbb{G}^{\times} \mid x \in (X^{\bullet} \cap B_x^{\bullet}) \textbf{ AND } y \in (X^{\bullet} \cap B_x^{\bullet})\} \qquad (6.8)$$

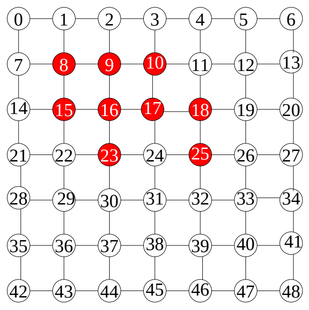
Operator $\delta_B^{\bullet}$ returns the set of all vertices that belong to the intersection of the edges of the given graph with those of the structuring element. The result of applying the structuring element given in Fig. 6.3(i) on the topology in Fig. 6.2 is shown in Fig. 6.4(b). We use the same structuring element with the other operators to illustrate the results of applying them on a given graph. The operator $\delta_B^{\times}$ returns the set of all edges, at least one end point of which is a member of the intersection of the vertices of the given graph and the vertices of the structuring element. The result is shown in Fig. 6.4(d). Operator $\epsilon_B^{\bullet}$ returns only those vertices where all the edges of the vertex in the graph are also the edges of the structuring element. $\epsilon_B^{\times}$ returns the set of edges where both the end points of the edge are in the intersection of the vertices of the given graph with those of the structuring element. The results for $\epsilon_B^{\bullet}$ and $\epsilon_B^{\times}$ are shown in Figs. 6.4(f) and 6.4(h).

As shown in [22], the operators $\delta_B^{\bullet}$ and $\delta_B^{\times}$ are dilations, $\epsilon_B^{\bullet}$ and $\epsilon_B^{\times}$ are erosions, when using structuring elements that comprise of both horizontal and vertical edges. If we compose the dilation and erosion operators to act on $\mathcal{G}^{\bullet}$ and $\mathcal{G}^{\times}$ with these structuring elements, we get the vertex and edge dilations and erosions using structuring elements. In fact, the operators defined in equations 6.5–6.8 are identical to the equations 6.1–6.4, when the structuring element is the 4-edge SE of Fig. 6.3(a). Thus, these equations are generalizations of the operators defined by Cousty et al..

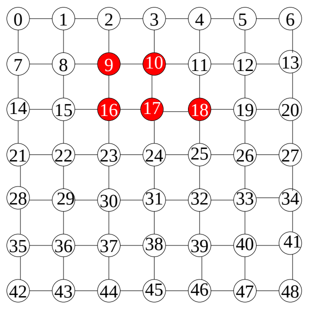
Vertex Dilation and Erosion : We define Vertex Dilation, δ_B and Vertex Erosion, ϵ_B that act on $\mathcal{G}^{\bullet}$ (i.e., $\mathcal{G}^{\bullet} \rightarrow \mathcal{G}^{\bullet}$) by $\delta_B = \delta_B^{\bullet} \circ \delta_B^{\times}$ and $\epsilon_B = \epsilon_B^{\bullet} \circ \epsilon_B^{\times}$.

Edge Dilation and Erosion : We define Edge Dilation, Δ_B and Edge Erosion, ε_B that act on $\mathcal{G}^{\times}$ (i.e., $\mathcal{G}^{\times} \rightarrow \mathcal{G}^{\times}$) by $\Delta_B = \delta_B^{\times} \circ \delta_B^{\bullet}$ and $\varepsilon_B = \epsilon_B^{\times} \circ \epsilon_B^{\bullet}$.

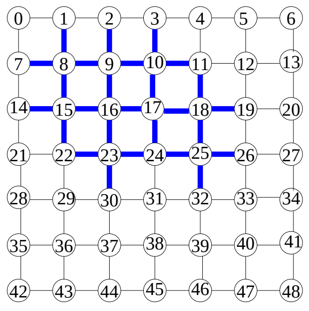
Graph Dilation and Erosion : We can now define the graph dilation and erosion operators using structuring elements by composing the dilation and erosion operators of vertices and edges together. Thus, the graph dilation operator, $X \oplus B$ or $\delta \oplus \Delta_B(X)$ is defined as $(\delta_B(X^{\bullet}), \Delta_B(X^{\times}))$. Similarly, the graph erosion operator, $X \ominus B$ or



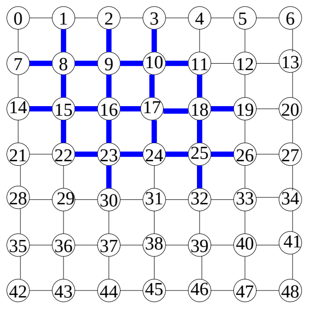
(a) $\delta^{\bullet}$



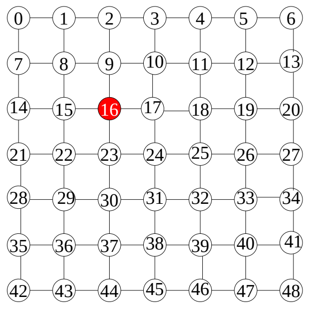
(b) $\delta^{\bullet}_{B}$



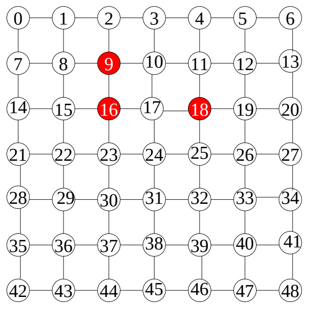
(c) $\delta^{\times}$



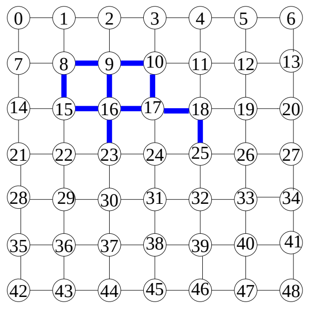
(d) $\delta^{\times}_{B}$



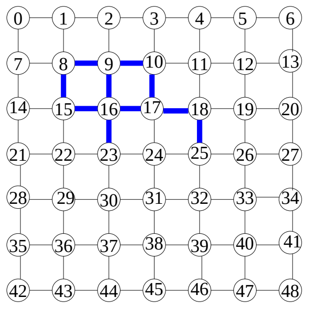
(e) $\epsilon^{\bullet}$



(f) $\epsilon^{\bullet}_{B}$

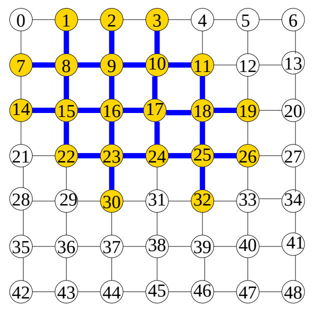


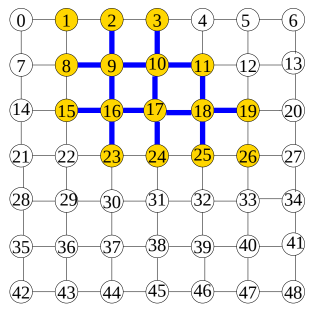
(g) $\epsilon^{\times}$

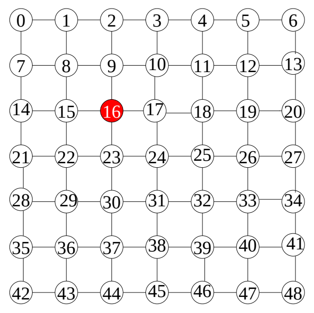


(h) $\epsilon^{\times}_{B}$

Figure 6.4: Result of applying Graph Morphological Operators on the graph in Fig. 6.2 without structuring elements (SEs) and with the SE of Fig. 6.3(i).


(a) $\delta \circledvee \Delta(X)$


(b) $\delta \oplus \Delta_B(X)$


(c) $\epsilon \circledwedge \varepsilon(X)$

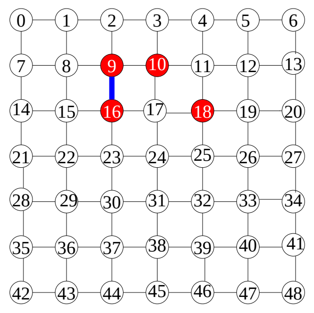

(d) $\epsilon \ominus \varepsilon_B(X)$

Figure 6.5: Graph dilation and erosion for the graph in Fig. 6.2 without SEs and with SE of Fig. 6.3(i).

$\epsilon \ominus \varepsilon_B(X)$ is defined as $(\epsilon_B(X^\bullet), \varepsilon_B(X^\times))$. For the graph given in Fig. 6.2 we computed the dilation and erosion with and without structuring elements. The results are shown in Fig. 6.5.

We applied our operators to compute the distance transform on images, which is presented in Section 6.4. We apply these operators, as well as those defined by Cousty, et al. to compute dominating sets. These algorithms are described in the next section.

6.3 Algorithms for MIDS and MDS using Graph Morphological Operators

As pointed out in Chapter 2, computing MDS and MIDS on grid graphs is $\mathcal{NP}$-hard. We propose heuristics for these two problems using the extended graph morphological operators. These are described in the next two sub-sections.

6.3.1 MIDS Algorithm

The MIDS algorithm $MIDS(X, Y^{\bullet})$, for a given graph $X = (X^{\bullet}, X^{\times})$ with the dominating set returned $Y^{\bullet}$ is as follows:

1. For each structuring element $B \in \mathcal{B}$ do the following:
   (a) $Z_B = \epsilon^{\bullet}_B(X^{\times})$
   (b) For each node $v \in Z_B$ do the following:
      i. If $B_v \not\subset X$ where B_v is the structuring element B translated to the node v, then remove v from Z_B and go to Step 1b
      ii. Construct the set of the dominating node, v, and its dominated nodes. This can be given as $\delta^{\bullet} \circ \alpha_{B_v}(X^{\times})$ where, $\alpha_{B_v}(X^{\times}) = \delta^{\times}_{B_v} \circ \epsilon^{\bullet}_{B_v}$, where the subscript v indicates that all the operations are performed only at node v. The result is the set of edges that are incident on the node v. $\delta^{\bullet}$ operator from [22] returns all nodes incident on these edges, which are nothing but the dominating and the dominated nodes.
      iii. The set of all edges incident on the dominating and dominated nodes is given by $\Delta \circ \alpha_{B_v}(X)$, where Δ is the edge dilation operator from [22]. We define $\beta_{B_v}(X) = (\delta^{\bullet} \circ \alpha_{B_v}(X), \Delta \circ \alpha_{B_v}(X))$ to be the subgraph of the dominating and dominated nodes and all edges incident on them.
      iv. We need to remove the subgraph represented by $\beta_{B_v}(X)$ from the given graph X by defining the operator $X = \psi_{B_v}(X) = (X \setminus \beta_{B_v}(X) = (X^{\bullet} \setminus \delta^{\bullet} \circ \alpha_{B_v}(X), X^{\times} \setminus \Delta \circ \alpha_{B_v}(X))$.
      v. Update Z_B by removing all nodes that are not in $\psi_{B_v}(X)$
   (c) $Y^{\bullet} = Y^{\bullet} \cup Z_B$
2. This leaves isolated nodes, if any. These are added to the dominating set to get the final dominating set: $Y^{\bullet} = Y^{\bullet} \cup X^{\bullet}$.

This is given in algorithmic format in Algorithm 16. The entire algorithm is illustrated in Figs. 6.6–6.17. An example topology is given in Fig. 6.6. Applying step 1a using the structuring element of Fig.6.3(a) results in $Z_B = \{23, 31\}$. Let us assume that the first element in $Z_B = 31$. Since $B_v \subset X$, step 1(b)ii is applied to get the

Algorithm 16: $MIDS(X = (X^\bullet, X^\times), Y^\bullet)$

```
$Y^\bullet := \phi$
for $B \in \mathcal{B}$ do
  $Z_B := \epsilon^\bullet_B(X^\times)$
  for $v \in Z_B$ do
    if $B_v \not\subset X$ then
      continue
    end if
    $Y^\bullet := Y^\bullet \cup v$
    $\alpha_{B_v}(X^\times) = \delta^\times_{B_v} \circ \epsilon^\bullet_{B_v}$
    $\beta_{B_v}(X) = (\delta^\bullet \circ \alpha_{B_v}(X), \Delta \circ \alpha_{B_v}(X))$
    $X = \psi_{B_v}(X) = (X \setminus \beta_{B_v}(X) = (X^\bullet \setminus \delta^\bullet \circ \alpha_{B_v}(X), X^\times \setminus \Delta \circ \alpha_{B_v}(X))$
    $Z_B := Z_B \cap X^\bullet$
  end for
end for
$Y^\bullet := Y^\bullet \cup X^\bullet$
return
```

dominating and its dominated nodes shown in Fig. 6.8. We now get all edges incident on these nodes using step 1(b)iii and the results is shown in Fig. 6.9. We remove this subgraph from the given graph in step 1(b)iv giving the graph of Fig. 6.10. At this point, we go back to step 1b and the next node in $Z_B = 23$. However, at this point, $B_v \not\subset X$. Since there are no more elements in Z_B, we go to the next structuring element in step 1, i.e., Fig. 6.3(b). Applying step 1a results in $Z_B = \{44, 46\}$. The process described above is repeated and shown in Figs. 6.12–6.13. The algorithm is repeated using other structuring elements until only an isolated node is left. Since it does not match any of the structuring elements, the algorithm eventually executes step 2 and adds this isolated node to MIDS.

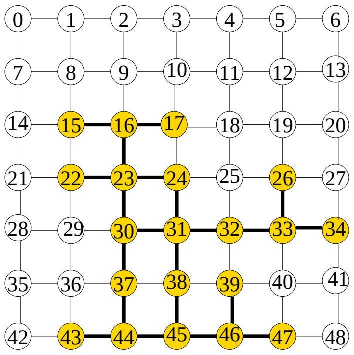


Figure 6.6: Topology to illustrate the working of MIDS algorithm

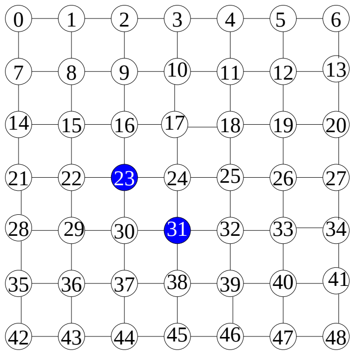


Figure 6.7: Applying $\epsilon^{\bullet}_{B}(X^{\times})$ with the structuring element of Fig.6.3(a) results in nodes 23, 31.

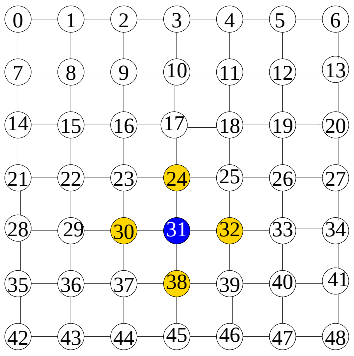


Figure 6.8: The dominating node 31 and its dominated nodes after step 1(b)ii of MIDS algorithm.

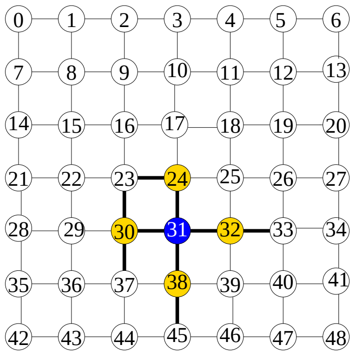


Figure 6.9: The dominating node 31, its dominated nodes and all edges incident on them after step 1(b)iii of MIDS algorithm.

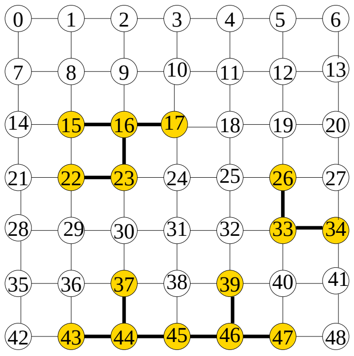


Figure 6.10: Result of removal of the subgraph constructed in Fig. 6.9 from the graph in Fig. 6.6 - Step 1(b)iv of MIDS algorithm.

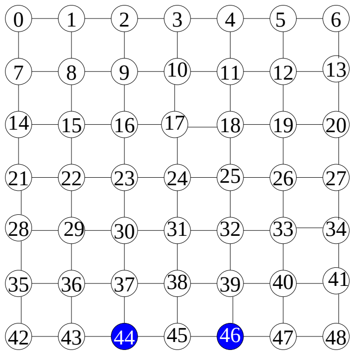


Figure 6.11: Applying $\epsilon^{\bullet}_{B}(X^{\times})$ with the structuring element of Fig.6.3(b) results in nodes 44, 46.

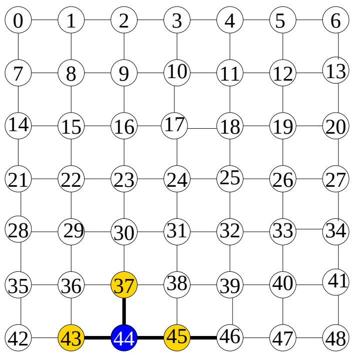


Figure 6.12: Result of executing steps 1(b)ii–1(b)iii of MIDS algorithm.

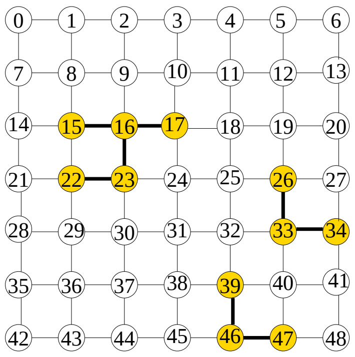


Figure 6.13: Residual graph after executing step 1(b)iv of MIDS algorithm.

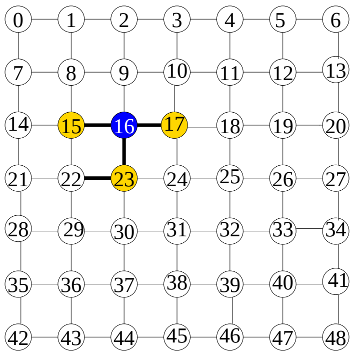


Figure 6.14: Result of applying $\epsilon^{\bullet}_{B}(X^{\times})$ with SE in Fig.6.3(d) and executing steps 1(b)ii–1(b)iii of MIDS algorithm.

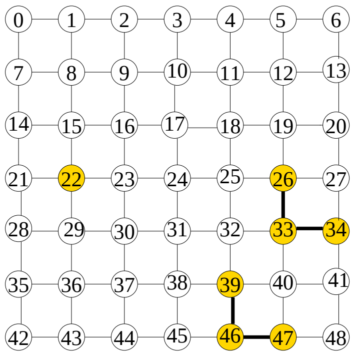


Figure 6.15: Residual graph after executing step 1(b)iv of MIDS algorithm.

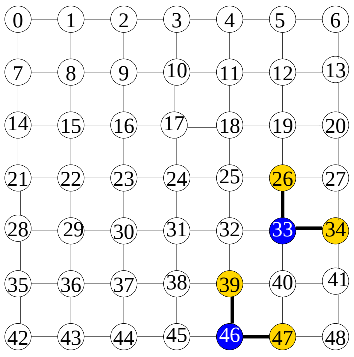


Figure 6.16: Result of applying $\epsilon^{\bullet}_{B}(X^{\times})$ with SE in Fig.6.3(f) and result of executing steps 1(b)ii–1(b)iii of MIDS algorithm.

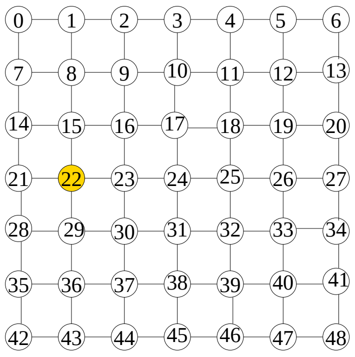


Figure 6.17: Residual graph after executing step 1(b)iv of MIDS algorithm which has only an isolated node. Executing step 2 will add this to MIDS.

6.3.2 MDS Algorithm

The Minimum Dominating Set(MDS), $MDS(Y^\bullet, X)$ is very similar to MIDS. Note, however, that we are not iterating for the construction of MDS unlike in MIDS. We apply the operators using each structuring element on the whole graph.

1. For each structuring element $B \in \mathcal{B}$ do the following:
   (a) Use the erosion operator $\epsilon_B^\bullet$ with the structuring element B on $X^\times$. Let us say that this results in a set of nodes $E^\bullet$.
   (b) If $E^\bullet = \phi$ go to the next structuring element B.
   (c) Otherwise, $Y^\bullet = Y^\bullet \cup E^\bullet$.
   (d) We apply the operator $\psi_B(X) = (X \setminus \beta_B(X) = (X^\bullet \setminus \delta^\bullet \circ \alpha_B(X), X^\times \setminus \Delta \circ \alpha_B(X))$ to remove the dominating and dominated nodes and the edges incident on them from the given graph X. Note that this removes many dominating nodes and their neighborhood in one iteration unlike in MIDS.
2. Add isolated nodes, if any, to the dominating set to get the final dominating set: $Y^\bullet = Y^\bullet \cup X^\bullet$.
3. We, now, apply a post-processing step to remove all redundant nodes from $Y^\bullet$. We call a dominating node redundant, if the node and all its neighbors are covered by other dominating nodes. $\Pi(A)$ is the set of dominated nodes of set A. Redundant node removal is expressed as $Y^\bullet = Y^\bullet \setminus \{x \in Y^\bullet \mid x \in \Pi(Y^\bullet \setminus x) \; and \; \forall e_{x,y} \in X^\times, y \in \Pi(Y^\bullet \setminus x)\}$.

This is given in algorithmic format in Algorithm 17. The algorithm is illustrated in Figs. 6.18–6.23. When step 1a is executed on the graph in Fig. 6.18 with the SE of Fig. 6.3(a), the resulting nodes are 23 and 33. The subgraphs induced by these nodes along with their dominated nodes and the edges incident on all these nodes is given in Fig. 6.19. The residual graph on removing this induced graph from the original graph is given in Fig. 1d. On applying $\epsilon_B^\bullet$ using the SE of Fig. 6.3(b) to the residual graph, the nodes 44, 45 and 46 are selected. The subgraph induced by these nodes along with their dominated nodes and the edges incident on them is given in Fig. 6.21. Removing this from the graph of Fig. 1d gives the graph in Fig. 6.22. Since all of these are isolated

Algorithm 17: $MDS(X = (X^\bullet, X^\times), Y^\bullet)$

```
    $Y^\bullet := \phi$
    **for** $B \in \mathcal{B}$ **do**
        $E^\bullet := \epsilon^\bullet_B(X^\times)$
        **if** $E^\bullet = \phi$ **then**
            **continue**
        **end if**
        $Y^\bullet := Y^\bullet \cup E^\bullet$
        $\alpha_{B_v}(X^\times) = \delta^\times_{B_v} \circ \epsilon^\bullet_{B_v}$
        $\beta_{B_v}(X) = (\delta^\bullet \circ \alpha_{B_v}(X), \Delta \circ \alpha_{B_v}(X))$
        $X = \psi_{B_v}(X) = (X \setminus \beta_{B_v}(X) = (X^\bullet \setminus \delta^\bullet \circ \alpha_{B_v}(X), X^\times \setminus \Delta \circ \alpha_{B_v}(X))$
    **end for**
    $Y^\bullet := Y^\bullet \cup X^\bullet$
    $Y^\bullet = Y^\bullet \setminus \{x \in Y^\bullet \mid x \in \Pi(Y^\bullet \setminus x) \; and \; \forall e_{x,y} \in X^\times, y \in \Pi(Y^\bullet \setminus x)\}$
    **return**
```

nodes, they will be added to the dominating set in step 2. Now, the post-processing step 3 is applied to this set. The result is removal of the node 45 from the dominating set as 45 and its neighbors, 44, 46 are dominated by 44 and 46, which are themselves members of the dominating set. Neighbor 38 of 45 is dominated by dominating set member 31. The final dominating set is shown in Fig. 6.23 with the dominating nodes in *blue*, dominated nodes in *yellow* and the redundant dominating node in *red*.

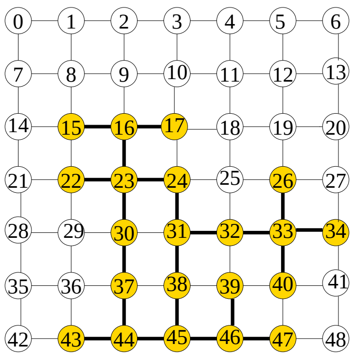


Figure 6.18: Example Topology to illustrate the MDS algorithm.

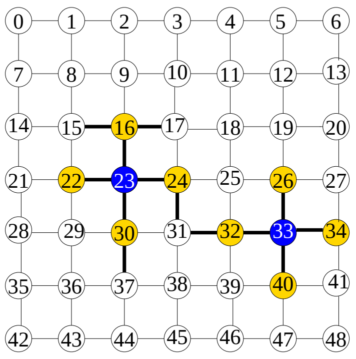


Figure 6.19: Subgraph induced by the dominating and dominated nodes after executing step 1a with SE in Fig.6.3(a).

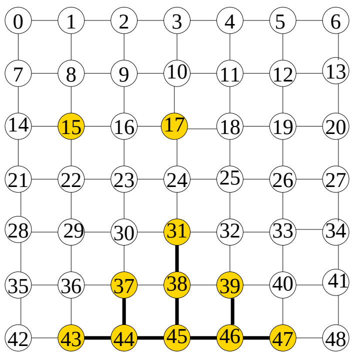


Figure 6.20: Residual graph after removing graph of Fig.6.19 from that in Fig. 6.18 (step 1d).

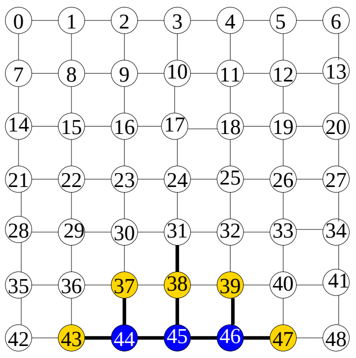


Figure 6.21: Result of executing step 1a with SE in Fig.6.3(b).

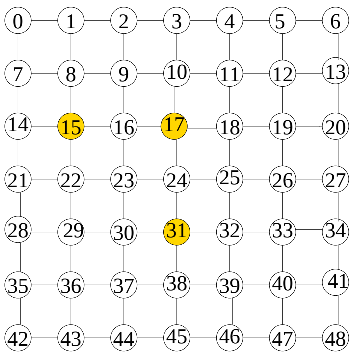


Figure 6.22: Residual graph after removing graph of Fig. 6.21 from that in Fig. 6.20.

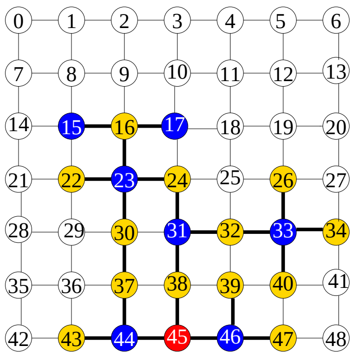


Figure 6.23: Dominating set found after the post-processing step with redundant nodes (marked in red) removed.

Limitations of the MDS Algorithm: One of the limitations is the question of whether we should remove the neighborhood of the nodes returned by the $\epsilon_B^\bullet$ operator. We look at two different topologies to understand the effect of this decision on the cardinality of the dominating set. In one, the cardinality increases if we remove neighborhood whereas it reduces in the other.

Consider the graph in Fig. 6.24. Here, when we apply the 4-edge structuring element erosion operator, the result is node 24. If we remove the neighborhood of this node from the graph, we have two isolated nodes as shown in Fig. 6.25, both of which need to be added to the dominating set. If we had not removed the neighborhood, we could add node 25 as part of the 2-edge structuring element erosion. This node covers both 18 and 32 and so the resultant cardinality of the dominating set is two, which is optimal.

When we consider the graph in Fig. 6.26, the result of the first erosion with the 2-edge structuring element of Fig. 6.3(g), is that nodes 23 and 32 are added to the dominating set. After removing their neighborhood, another 2-edge structuring element (Fig. 6.3(i)) matches at node 30 of the resultant graph and this becomes part of the dominating set. When its neighborhood is removed, a single edge structuring element erosion leads to 17 being part of the dominating set. Thus, the final dominating set is 23, 32, 30, 17 for a cardinality of four.

If we remove only the dominating nodes but not the neighborhood in this example graph, we end up with 23, 32 being added in the first round as earlier. However, erosion with other 2-edge structuring elements results in 24, 29 and 30 being part of the eroded set. Thus, the dominating set will be 23, 32, 24, 29, 30. This is one more than the cardinality obtained if we remove the neighborhood of the dominating nodes. Thus, it can be seen that removing the neighborhood of a node can increase cardinality in some instances and reduce in others.

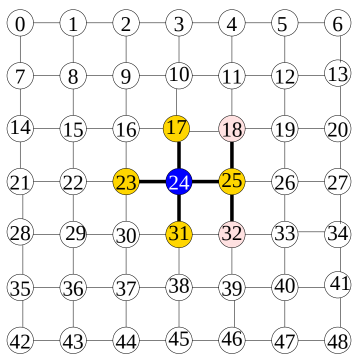


Figure 6.24: Example topology to illustrate removal of neighborhood of a dominating node results in increase in cardinality. Node 24 is the dominating node after step 1a.

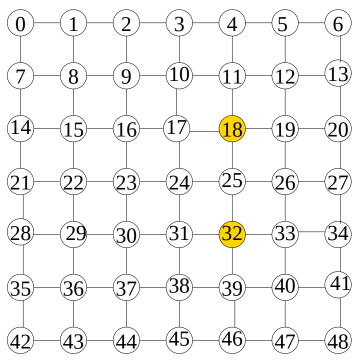


Figure 6.25: Removal of the neighborhood of the dominating node 24 results in isolated nodes 18 and 32 being added to the dominating set.

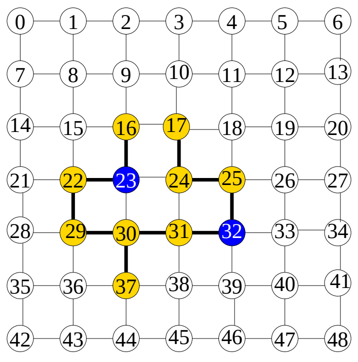


Figure 6.26: Example Topology to illustrate the advantage of removing the neighborhood of dominating nodes. Nodes 23 and 32 are the result of applying $\epsilon^{\bullet}_{B}$ using SE of 6.3(g).

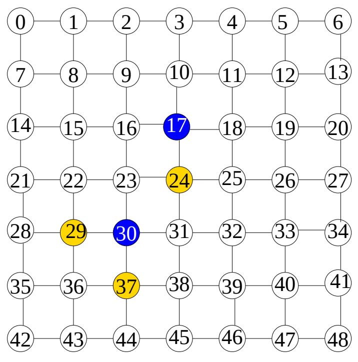


Figure 6.27: Erosion with the 2-edge structuring element Fig.6.3(i) and 1-edge structuring element of Fig.6.3(o) adds nodes 30 and 17 to the dominating set.

6.4 Experimentation and Results

Table 6.1: Cardinality (i) of MIDS returned by the Maximum Degree Heuristic, 3HD Heuristic and Morphology Algorithms for Incomplete Grid Graph Instances

Edges	**Max.Deg. Heuristic**		**3HD Heuristic**		**Morphology**	
	i	t **(ms)**	i	t **(ms)**	i	t **(ms)**
50	15.35	0.138	15.4	0.10025	15.3	352.042
100	26.5	0.2688	27.4	0.16305	26.8	700.504
250	53.9	0.59785	53.1	0.3078	54.05	1729.8
500	100.3	1.1482	98.05	0.5583	100.05	4331.52
750	140.6	1.66625	134.3	0.8351	140	6701.06
1000	181.1	2.3031	175.8	1.1548	181.35	10687.9

Table 6.2: Cardinality (γ) of MDS returned by the Optimal Approximation Algorithm and Morphology Algorithms for Incomplete Grid Graph Instances

Edges	**Heuristic**		**Morphology**	
	γ	t **(ms)**	γ	t **(ms)**
50	14.25	0.7296	14.95	168.047
100	24.45	2.17165	25.85	247.804
250	50.2	10.9223	56	399.16
500	92.85	33.0356	98.65	644.136
750	127.85	74.8657	143.4	787.486
1000	168.9	123.271	182.25	989.726

As part of the experimentation, we used the incomplete grid graph generator software from Section 3.5.2. We computed the average of the dominating set cardinality using our morphological algorithms and compared them to the results of the standard heuristics. These results are presented in Tables 6.1 and 6.2. We find that our algorithm for MIDS performs as well as the maximum degree and 3HD heuristics, which were found to be the best heuristics for grid graphs as seen in Section 3.5.3. The worst case cardinality is for 1000 edges which is 3% higher than the 3HD heuristic. It can be seen that the morphological algorithm matches the maximum degree heuristic for all instances. It is worse than 3HD because the morphological algorithm also suffers from the same issue of isolated nodes as the maximum degree heuristic discussed in Sections 3.5.3 and 2.3.1.1.

We also find that our MDS algorithm performs on the average about 8% worse than the optimal approximation algorithm [102]. This can be explained by the fact that, before applying the next structuring element, we are removing the neighborhood of nodes selected by the erosion operator. Sometimes, this can lead to an increase in cardinality. On the other hand, not removing the neighborhood leads to an increase in cardinality in other cases.

The time taken to compute MDS and MIDS using morphological operators is much more than the heuristics as we have not optimized the implementation for performance. Efficient implementations, both sequential and parallel, have been proposed by Vincent [100] and by Géraud et al. [42] which suggest that these running times may be reduced considerably making them comparable to the other approaches.

6.5 Distance Transform Computation using the Extended Graph Morphology Operators

We applied our extended graph morphological operators to compute the distance transform of an image and show that they are exactly identical in operation to standard morphological operators on sets of points. The extended operators defined by us can be directly used on all images modeled as graphs. As yet, our operators are defined for binary images and so we illustrate the distance transform on binary images only.

The algorithm for computation of distance transform works as follows: we initially mark the distance of all nodes in the graph as 1 to indicate they are at a minimum distance of 1 from the background pixels. We apply the $\epsilon_B^\bullet$ operator with the structuring element of Fig. 6.3(a) on the edge set of the given graph. The nodes returned by this operator, S, are marked to be distance 2 from the background pixels. We, now, apply the same operator on the subgraph induced on S. Any nodes returned are marked to be at a distance 3 from the background pixels. This operation is repeated until $S = \phi$. This is nothing but ultimate erosion with the structuring element given. We give an example image in Fig. 6.28 and its corresponding distance transform computed by our algorithm in Fig. 6.29.

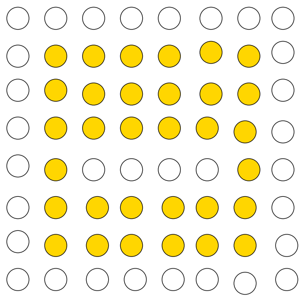


Figure 6.28: Image to illustrate distance transform using ultimate erosion with structuring elements.

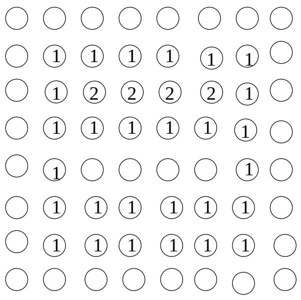


Figure 6.29: Distance transform of the image, with nodes in the given graph labeled with the distance computed by the ultimate erosion using $\epsilon_B^\bullet$ with SE of Fig.6.3(a).

6.6 Applications of Dominating Sets to Image Analysis

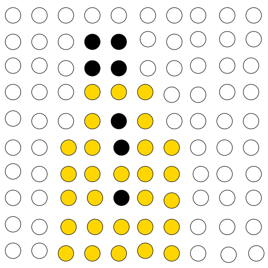


Figure 6.30: Image from Gonzalez and Woods [46], with the nodes in black representing the skeleton obtained using a 3×3 structuring element.

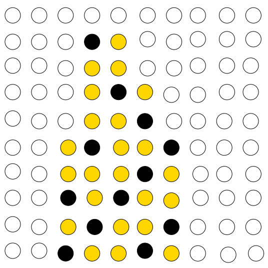


Figure 6.31: Skeleton of the same image with the MIDS nodes (in black) representing the skeleton.

In this section, we apply dominating sets, for the first time, to the problem of image analysis. Specifically, we illustrate two uses of dominating sets in image analysis, i.e., the dominating set as the skeleton of the image and in clustering of the image. Clustering is a very natural application of dominating sets and has been used for this purpose in wireless networks and document clustering. We propose a novel hierarchical clustering scheme using an overlay of dominating sets for image clustering.

6.6.1 MIDS Skeleton

In this section, we demonstrate the use of dominating sets, and in particular, MIDS, to represent the skeleton of an image [20]. We define the skeleton as the nodes returned by the MIDS algorithm. In Fig. 6.30, we reproduce the image used by Gonzalez and Woods [46] to illustrate the computation of a skeleton using ultimate erosion with structuring elements. The skeleton of the image using a standard 3×3 structuring element of pixels is represented by the black nodes in the figure. The skeleton represented by the MIDS of the image is shown in Fig. 6.31. By also storing the structuring element that caused a node to be selected as the dominating node, we can reconstruct the image by applying dilation operations. A second example is taken from [38] and is shown in Fig. 6.32. We reproduce the skeleton (from the same source) obtained using the standard 3×3 structuring elements. The skeleton is represented by the black nodes in the figure. The skeleton represented by the MIDS of the image is given in Figure 6.33, with the MIDS nodes shown in black.

The MIDS Skeleton is an interesting alternative to the skeletons based on distance transforms, Medial Axis transform and others. The other skeletons generally tend to capture the centrality of the object while the MIDS Skeleton is more uniformly distributed. The potential advantage of the MIDS Skeleton is that it may be possible to reconstruct the original graph with relatively fewer operations as the independence property ensures that any node is reconstructed from only one dominating node (i.e., from the skeleton). Further, from the examples shown, it appears that the size of the MIDS Skeleton is comparable to the other skeletons.

6.6.2 Hierarchical Clustering using Dominating Sets

We present a hierarchical clustering scheme which uses an overlay of dominating sets and illustrate the scheme with a couple of examples. The algorithm is given in algorithmic format in Algorithm 18 and is as follows:

1. Repeat the following on the given graph X until MIDS constructed has only one node or there are only disconnected components.

   (a) Using the 4-node structuring element in Fig.6.36(b) at each node y, compute $\epsilon^{\bullet}_{B_y}(X^{\times})$. Add the node y, if it satisfies the erosion property, to the

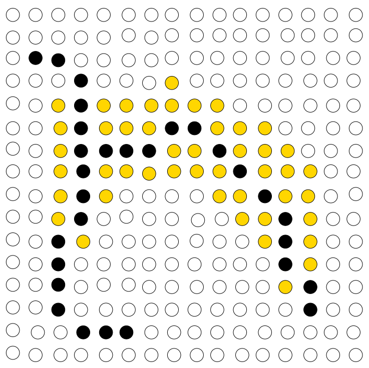


Figure 6.32: Example Image [38] used to illustrate construction of a skeleton using ultimate erosion with structuring elements. The skeleton nodes are in black.

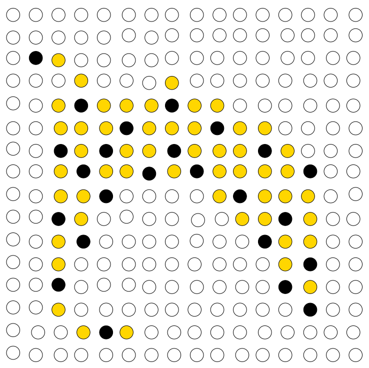


Figure 6.33: Skeleton of the image with the MIDS nodes (in black) representing the skeleton of the image.

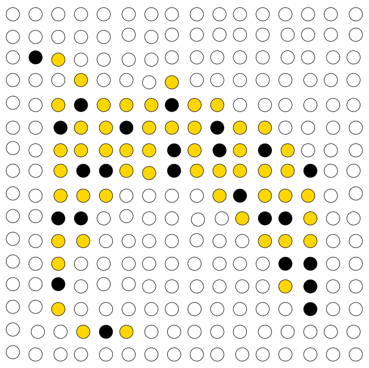


Figure 6.34: Skeleton of the image with the optimal approximation MDS algorithm. Nodes in black are the MDS nodes representing the skeleton of the image.

dominating set $Y^{\bullet}$. Calculate the residual graph $\psi_{B_y}(X)$. Repeat this step until there are no nodes that are part of the eroded set using this structuring element.

(b) Repeat Step 1a using the structuring elements of Fig. 6.36(c) and 6.36(d) on the residual graph obtained at the end of step 1a. Note that these two structuring elements are applied in parallel to the residual graph of step 1a and not in sequence. Any isolated nodes left at the end of applying each structuring element are also added to the dominating set.

(c) We, now, construct the overlay graph, X, of the dominating nodes. For this, we add an edge $e_{u,v}$ between dominating nodes u and v, if they or their dominated nodes are adjacent to each other along the grid.

Algorithm 18: *Hierarchical_Clustering*$(X = X^{\bullet}, X^{\times})$

```
$D := \phi$
**while** $(|D| \neq 1)$ *and* $(|X^{\times}| \neq 0)$ **do**
  $Y := X$
  **while** $(z := \epsilon^{\bullet}_{B_y}(Y^{\times})) \neq \phi$ **do**
    $D := D \cup z$ where $B$ is the 4-node structuring element in Fig.6.36(b).
    $Y := \psi_{B_y}(Y)$
  **end while**
  $Y' := Y$
  **while** $(z := \epsilon^{\bullet}_{B_y}(Y'^{\times})) \neq \phi$ **do**
    $D := D \cup z$ where $B_y$ is the 2-node structuring element in Fig.6.36(c)
    $Y' := \psi_{B_y}(Y')$
  **end while**
  $D := D \cup Y'^{\bullet}$
  $Y' := Y$
  **while** $(z := \epsilon^{\bullet}_{B_y}(Y'^{\times})) \neq \phi$ **do**
    $D := D \cup z$ where $B_x$ is the 2-node structuring element in Fig.6.36(d)
    $Y' := \psi_{B_y}(Y')$
  **end while**
  $D := D \cup Y'^{\bullet}$
  $X :=$ *Construct_Overlay_Graph*$(X, D)$
**end while**
**return** $D$
```

An example image is given in Fig. 6.35(a). The dominating nodes of the graph representing this image, after the first iteration, are shown in the darker shade. The overlay graph induced by the dominating nodes is shown in Fig. 6.35(b). The dominating set of this graph is shown in a lighter shade. The overlay graph of the dominating nodes of Fig.6.35(b) is shown in Fig.6.35(c). Finally, only the top node of the graph in Fig.6.35(c) is the dominating node and the clustering algorithm halts. As the MIDS at

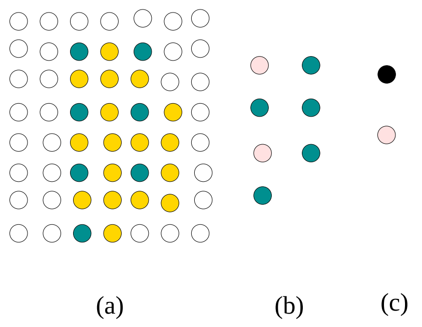


Figure 6.35: An example of an image which results in a single cluster after *three* iterations of clustering on overlays of dominating sets. Iteration 1 is shown in (a), iteration 2 in (b) and iteration 3 in (c).

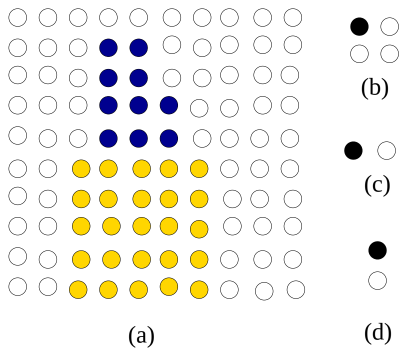


Figure 6.36: Hierarchical clustering of the image shown in Fig. 6.30. We get *two* clusters shown in different shades in (a). The 4-node SE is shown in (b), the 2-node SEs are shown in (c) and (d).

the last overlay has only one node, it implies that the entire graph is a single cluster. Another example is shown in Fig. 6.36 which shows the clusters formed for the image in Fig.6.30 using this algorithm. In this case, there are two disjoint dominating nodes after *four* iterations leading to two clusters in the given image. These are represented by different shades for the two clusters.

It may be seen that the MIDS clustering algorithm requires neither the number of clusters nor seed points as input. It starts by operating on 1-hop nodes and then proceeds to more and more distant nodes as it travels up the overlay graphs. In this sense, it is analogous to the algorithms that use a distance threshold for clustering.

6.7 Summary

We extended the dilation and erosion operators defined on graphs in [22] to use structuring elements. Using the ordered set of structuring elements, we have designed algorithms to compute Minimum Dominating Set (MDS) and Minimum Independent Dominating Set (MIDS). We find that the cardinality returned by our algorithms is similar to the best heuristics in the literature. Our distance transform algorithm using graph morphological operators returns similar results to those defined using morphological operators on point sets. We also proposed the application of dominating set as the skeleton of images and show that they provide a sufficiently interesting alternative to the methods proposed in literature. We propose a novel hierarchical clustering scheme using an overlay graph of dominating sets for clustering of images. While dominating sets have been used for clustering in many domains, this is the first time, to our knowledge, it has been applied to clustering of images. In future, we plan to explore the use of dominating sets in other aspects of image analysis. We also wish to define granulometries for more intuitive approaches to hierarchical clustering.

Chapter 7

Conclusions and Future Work

"Tut, tut, child!" said the Duchess, "Everything's got a moral, if only you can find it."
Alice's Adventures in Wonderland, Lewis Carroll

Dominating sets find many uses in diverse domains such as wireless networks, information retrieval, optical networks, query selection in databases, facility location and so on. Minimizing the dominating sets leads directly to savings in terms of energy dissipation in wireless networks, cost savings in optical networks, search efficiency in information retrieval etc.. Algorithms that result in better minimization of dominating sets can be used for such applications with great effect. Our empirical study of exact algorithms, PTAS and heuristics for MIDS showed that while PTAS is of theoretical interest, it is not practical for large graphs or high density graphs. Further, we found that most of the approximation algorithms proposed are limited to bounded independence graphs such as UDGs or have other limitations.

Metaheuristic algorithms have been proven to be a good alternative for practical solutions to many $\mathcal{NP}$-hard problems. In this thesis, we proposed metaheuristic algorithms for the dominating set problems and showed that these algorithms result in superior solutions compared to the heuristics. For MDS particularly, the metaheuristics give better solutions than the optimal approximation algorithm, making them the algorithm of choice when minimization of cardinality is crucial.

We also provide completely novel algorithms for dominating sets using graph morphological operators. This is the first time that these operators have been applied to standard graph theoretic problems such as the computation of MIDS and MDS. We also proposed a new application of dominating sets by using them for image analysis.

We have shown that dominating sets present a sufficiently interesting alternative to represent the skeleton of the image. We present a novel hierarchical clustering scheme using an overlay graph of dominating nodes and illustrate its use in clustering of images.

The rest of the sections of this chapter elaborate on these conclusions. The final section indicates the future directions of research derived from our work.

7.1 Empirical Study of MIDS Algorithms

We performed an empirical study of two exact algorithms for MIDS, viz., the algorithm due to Liu and Song [64] and an intelligent enumeration algorithm. We find that while the former has better asymptotic complexity, the latter takes less time for small graphs. PTAS algorithms for various dominating sets use exact algorithms on local neighborhood of nodes. In such cases, we conclude that it is better to use intelligent enumeration.

We compared the cardinality of the only PTAS for MIDS [56] with that returned by the heuristics for UDGs and Grid Graphs. We find that the PTAS returns, at best, the cardinality returned by the maximum degree and 3HD heuristics. This comes at the cost of high computational time for small graphs or low density graphs. For high density and large graphs, it is not practical, as it uses an exact algorithm for local optimal MIDS computation.

The 3HD heuristic is the best for Grid Graphs as choosing the maximum degree node leaves many isolated nodes in grid graphs. The maximum degree heuristic is the best for all other types of graphs studied.

7.2 Metaheuristic Algorithms for Dominating Sets

The conclusions we draw from our metaheuristic algorithms for the Minimum Dominating Set, Minimum Weight Dominating Set and Minimum Capacitated Dominating Set problems are as follows:

1. The Hybrid Genetic and Hybrid Ant-Colony Optimization Algorithms that we proposed perform similarly for the problems we studied. Either of them can be used for computing dominating sets. The ACO algorithms, however, take less time to arrive at the same solution as the hybrid genetic algorithm in many cases.

2. The local-search component, i.e., the heuristic designed for the minimization of cardinality of the solution constructed by basic metaheuristic algorithms, is crucial for the performance of these metaheuristic algorithms.

3. For the ACO algorithm, using a combination of pheromone and heuristic components to determine the probability of selecting the next node adds to the computational cost but does not help in reducing cardinality. Thus, we conclude that for these problems, a simple state-transition rule that uses only the pheromone deposit on the nodes, in combination with the local search, is sufficient for reducing cardinality.

4. Use of a pre-processing step in the ACO algorithms does not improve cardinality in MWDS whereas, in CAPMDS, this step is essential to reduce cardinality for some instances. We conclude from our experimentation, that a pre-processing step is only useful when, because of constraints, metaheuristics are not able to reach some parts of search space which, the greedy heuristic, used in preprocessing, is able to reach.

5. The heuristics we worked with are all superior to the metaheuristic algorithms proposed earlier in the literature. Our proposed metaheuristic algorithms are superior to the heuristics without exception. However, for some instances in CAPMDS, we believe that an improvement of the coverage algorithm will reduce cardinality even further.

7.3 Morphological Algorithms for Dominating Sets

We extended the graph morphological operators to use structuring elements. We used these operators to construct algorithms for MIDS and MDS. We applied the extended operators and the dominating sets to image analysis problems such as distance transform and skeleton. We have proposed a novel hierarchical clustering scheme that uses an overlay graph of dominating nodes.

We can conclude the following from our algorithms:

1. The cardinality returned by the MIDS algorithm is as good as the maximum degree heuristic, and slightly worse than the 3HD heuristic, which is the best for

grid graphs. The cardinality returned by the MDS algorithm is worse by about 8% than the optimal approximation algorithm for MDS.

2. The time taken by the morphological algorithms is worse than the heuristics as it involves more operations.

3. The distance transform computed using ultimate erosion with the $\epsilon_B^\bullet$ operator, where B is the 4-edge structuring element of Fig. 6.3(a), gives equivalent results as that using point sets.

4. We use the dominating set to represent the skeleton of the image. By associating the structuring element that resulted in the node being part of the dominating set, we will be able to reconstruct the original image. This has the advantage of reconstruction with fewer operations as each pixel is reconstructed exactly once in MIDS.

5. The hierarchical clustering scheme gives intuitive results with the sample images tested.

7.4 Future Work

The following constitutes the future work of this thesis:

Improvement of Coverage Algorithm in CAPMDS: As seen in Sections 5.4.3 and 5.4.4, the improvement achieved by metaheuristics over the heuristics for CAPMDS drop down to 3-5% from 15-20% for high degree of connectivity. We believe that, in these cases, the choice of which neighbors are dominated by a dominating node has an effect on the cardinality of CAPMDS. We propose that pheromone can be added on the edges also, i.e., a multi-pheromone solution to the problem. Whenever a node is selected, we update the pheromone on it as well as the pheromone on the edges that connect it to the neighbors which are dominated by it. If a certain subset of neighbors leads to a better minimization of cardinality, the edges to these neighbors will have a higher pheromone and so these neighbors will be chosen more often in future. Further, we can improve the pre-processing step by generating more solutions and choosing those which have the best cardinality. The pre-processing will also update the edge pheromone that will lead to certain neighbors being preferred from the beginning of the actual ACO algorithm. We can combine the edge pheromone concept with the hybrid genetic algorithm also.

Improvement of Morphological MIDS and MDS Algorithms: In Section 6.3.2, we identified the limitations of the morphological algorithm for MDS. It is possible that this algorithm can be improved using conditional dilation. We will need to define conditional dilation with respect to graphs and then apply it to the MDS algorithm. The running times of these algorithms can also be improved upon by an efficient implementation. Efficient implementation of morphological algorithms is a thrust area of research in morphology. Morphological algorithms are highly amenable to parallel implementation and this will also be explored in the future.

Reconstruction of Original Image using MIDS Skeleton: As identified in Section 6.6.1, the MIDS can be used to represent the skeleton of the image. We identified a potential advantage of the MIDS skeleton as the reconstruction of the original image in fewer operations. This requires storing additional state information with each dominating node such as the SE, the erosion with which, added the node to the dominating set. Then, the reconstruction algorithm has to be developed to use this state information. This needs to be compared against other reconstruction algorithms in terms of fidelity achieved and the time takne on different image data sets. If found to be better in space and time complexity, this can be used for image compression also.

Hierarchical Clustering Algorithm for General Graphs: The hierarchical clustering algorithm proposed for images can be extended to general graphs. In grid graphs, we had the restriction that the dominating and/or dominated nodes had to be neighbors along the grid. Hence, we needed to use specialized structuring elements for the clustering.

The hierarchical clustering algorithms for general graphs is as follows.

1. Given the graph $G = (V, E)$, we compute the MDS, D, of the graph.
2. Construct overlay graph $G^{'}$ as follows:
   (a) All the dominating nodes are vertices in the overlay graph.

   (b) Add an edge between two dominating nodes u, v if:
      - the dominating nodes u and v are neighbors
      - if a dominating node is a neighbor of a dominated node of the other
      - if the dominated nodes of u and v are neighbors of each other

3. Repeat step 1 with G' as G until fixed number of iterations or $\mid D \mid= 1$ or G' is a disconnected graph.

The MDS can be computed using either the optimal approximation algorithm or our proposed metaheuristics. For example, if the given original graph is large, running the optimal approximation algorithm will be fast. Once the size of the nodes in the overlay graph has become manageable, we can apply the metaheuristics to reduce the cardinality of the MDS obtained.

We plan to apply this algorithm to Document Clustering and Chemoinformatics data. It can also be used in consensus clustering as one of the clustering schemes.

References

[1] A.A.Mousa, Waiel F. Abd El-Wahed, and R.M.Rizk-Allah. **A hybrid ant colony optimization approach based local search scheme for multiobjective design optimizations**. *Electric Power Systems Research*, **81**:1014–1023, 2011. (22, 26)

[2] Scott T. Acton. **Fast Algorithms for Area Morphology**. *Digital Signal Processing*, **11**(3):187–203, 2001. (5)

[3] Paola Alimonti and Tiziana Calamoneri. **Improved approximations of independent dominating set in bounded degree graphs**. In *Proceedings 22nd International Workshop on Graph-Theoretic Concepts in Computer Science, WG'96*, pages 2–16, 1996. (32, 33)

[4] Bassam Aoun, Raouf Boutaba, Youssef Iraqi, and Gary Kenward. **Gateway Placement Optimization in Wireless Mesh Networks With QoS Constraints**. *IEEE Journal on Selected Areas in Communications, Vol. 24, No. 11*, pages 2127–2136, November 2006. (2)

[5] Judit Bar-Ilan, Guy Kortsarz, and David Peleg. **How to allocate network centers**. *Journal of Algorithms*, **15**:385–415, November 1993. (2, 32, 36, 96, 99, 103)

[6] Stefano Basagni. **Distributed Clustering for Ad hoc Networks**. In *Proceedings ISPAN'99 International Symposium on Parallel Architectures, Algorithms and Networks*, pages 310–315, 1999. (42)

[7] J. E. Beasley and P. C. Chu. **A genetic algorithm for the set covering problem**. *European Journal of Operational Research Volume 94, Issue 2*, pages 392–404, October 1996. (64)

[8] Isabelle Bloch. **Lattices of fuzzy sets and bipolar fuzzy sets and mathematical morphology**. *Information Sciences*, **181**:2002–2015, 2011. (5)

[9] Isabelle Bloch and Alain Bretto. **Mathematical Morphology on Hypergraphs: Preliminary Definitions and Results**. In *Proceedings of Discrete Geometry for Computer Imagery, DGCI-2011*, LNCS 6607, pages 429–440, 2011. (5)

[10] Christian Blum. **Ant Colony Optimization for the edge-weighted k-cardinality tree problem**. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO-2002, pages 27–34, 2002. (22)

[11] Adrian Bondy and U.S.R. Murty. *Graph Theory*. Springer, Berlin, 2008. (5)

[12] Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. **Fast algorithms for min independent dominating set**. *CoRR abs/0905.1993*, 2009. (32, 42)

[13] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. **A new Rank based Version of Ant System: A Computational Study**. *Central European Journal of Operations Research and Economics*, **7**:25–38, 1999. (24)

[14] Yidong Chen and Edward R. Dougherty. **Texture Classification by Gray-Scale Morphological Granulometries**. In Petros Maragos, editor, *Visual Communications and Image Processing '92*, pages 931–942. SPIE 1818, 1992. (5)

[15] Yuanzhu Peter Chen, Arthur L. Liestman, and Jiangchuan Liu. **Clustering Algorithms for Ad Hoc Wireless Networks**. *Ad Hoc and Sensor Networks*, pages 145–164, 2006. (1, 9, 42)

[16] V Chvatal. **A greedy heuristic for the Set Covering Problem**. *Mathematics of Operations Research, Vol. 4, No. 3*, pages 233–235, 1979. (60)

[17] Brent N. Clark and Charles J. Colbourn. **Unit disk graphs**. *Discrete Mathematics, Vol. 86, Issues 1-3*, pages 165–177, December, 1990. (1, 15, 16, 42)

[18] Mary L. Comer and Edward J. Delp. **Morphological operations for color image processing**. *J. Electronic Imaging*, **8**(3):279–289, 1999. (5)

[19] Thoman H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1990. (17)

[20] Michel Couprie and Hugues Talbot. **Distance, Granulometry and Skeleton**. In Laurent Najman and Hughes Talbot, editors, *Mathematical Morphology*, pages 265–289. John Wiley and Sons Inc., 2010. (151)

[21] Jean Cousty, Gilles Bertrand, Michel Couprie, and Laurent Najman. **Collapses and Watersheds in Pseudomanifolds**. In Petra Wiederhold and Reneta Barneva, editors, *Combinatorial Image Analysis*, **5852** of *Lecture Notes in Computer Science*, pages 397–410, 2009. (5)

[22] Jean Cousty, Laurent Najman, and Jean Serra. **Some Morphological Operators in Graph Spaces**. In *Proceedings of International Symposium on Mathematical Morphology, ISMM-2009*, pages 149–160, 2009. (6, 7, 28, 133, 134, 136, 139, 154)

[23] Marek Cygan, Marcin Pilipczuk, and Jakub Onufry Wojtaszczyk. **Capacitated Domination Faster Than $O(2^n)$**. In *SWAT*, pages 74–80, 2010. (32)

[24] Decheng Dai and Changyuan Yu. **A $5 + \epsilon$-approximation algorithm for minimum weighted dominating set in unit disk graph**. *Theoretical Computer Science 410*, pages 756–765, 2009. (32)

[25] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. (19, 62)

[26] J.L. Deneurborg, S. Aron, S. Goss, and J.M. Pasteels. **The self-organizing exploratory pattern of the Argentine ant**. *Journal of Insect Behavior*, **3**:159–168, 1990. (22)

[27] Fábio Dias, Jean Cousty, and Laurent Najman. **Some morphological operators on simplicial complex spaces**. In *Proceedings of the 16th IAPR international conference on Discrete geometry for computer imagery*, DGCI'11, pages 441–452, 2011. (5)

[28] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. **Positive Feedback as a Search Strategy**. Technical report, 1991. (22, 24)

[29] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. **Ant System: Optimization by a colony of cooperating agents**. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, **26**(1):29–41, 1996. (22, 23)

[30] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004. (22, 23)

[31] Kathryn A Dowsland and Jonathan M Thompson. **An improved ant colony optimisation heuristic for graph colouring**. *Discrete Applied Mathematics*, **156**:313–324, 2008. (26, 27)

[32] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003. (20, 21)

[33] Kayhan Erciyes, Orhan Dagdeviren, Deniz Cokuslu, and Deniz Ozsoyeller. **Graph Theoretic Clustering Algorithms in Mobile Ad Hoc Networks and Wireless Sensor Networks**. *Appl. Comput. Math., no. 2*, pages 162–180, 2007. (1, 9, 42)

[34] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. **Efficient Graph-Based Image Segmentation**. *Int. J. Comput. Vision*, **59**(2):167–181, September 2004. (5)

[35] Pedro F. Felzenszwalb and Ramin Zabih. **Dynamic Programming and Graph Algorithms in Computer Vision**. *IEEE Trans. Pattern Anal. Mach. Intell.*, **33**(4):721–740, 2011. (5)

[36] Serge Fenet and Christine Solnon. **Searching for Maximum Cliques with Ant Colony Optimization**. In *Proceedings of the 3rd European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP-2003*, LNCS 2611, pages 236–245, 2003. (22)

[37] F.G.Nocetti, J.S.Gonzalez, and I.Stojmenovic. **Connectivity-based k-hop clustering in wireless networks**. *Telecommunication Systems 22(1-4)*, pages 205–220, 2003. (42, 43)

[38] Robert Fisher. **Image Processing Learning Resources**. http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm. (xv, 151, 152)

[39] Robert Fisher. **Image Processing Learning Resources**. http://homepages.inf.ed.ac.uk/rbf/HIPR2/index.htm. (30)

[40] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Ioan Todinca. **Exponential time algorithms for the minimum dominating set problem on some graph classes**. *ACM Trans. Algorithms*, **6**(1):9:1–9:21, December 2009. (32)

[41] Serge Gaspers and Mathieu Liedloff. **A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set in Graphs**. In *Proceedings 32nd International Workshop on Graph-Theoretic Concepts in Computer Science, WG-2006*, pages 78–89, 2006. (32, 42)

[42] Thierry Géraud, Hughes Talbot, and Marc Van Droogenbroeck. **Algorithms for Mathematical Morphology**. In Laurent Najman and Hughes Talbot, editors, *Mathematical Morphology*, pages 323–353. John Wiley and Sons Inc., 2010. (149)

[43] Beat Gfeller and Elias Vicari. **A faster distributed approximation scheme for the connected dominating set problem for growth-bounded graphs**. In *Proceedings of the 6th international conference on Ad-hoc, mobile and wireless networks*, ADHOC-NOW'07, pages 59–73, 2007. (3, 33, 34, 40)

[44] Beat Gfeller and Elias Vicari. **A randomized distributed algorithm for the maximal independent set problem in growth-bounded graphs**. In

Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, PODC '07, pages 53–60, 2007. (3)

[45] David E. Goldberg and Kalyanmoy Deb. **A comparative analysis of selection schemes used in genetic algorithms**. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann, 1991. (20)

[46] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison Wesley, New York, 1993. (xv, 31, 150, 151)

[47] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. **Self-organized shortcuts in the Argentine ant**. *Naturwissenschaften*, **76**(12):579–581, 1989. (22)

[48] Magns M. Halldrsson. **Approximating the minimum maximal independence number**. *Information Processing Letters, Vol. 46, Issue 4*, pages 169–172, June, 1993. (42)

[49] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Domination in Graphs: Advanced Topics*. Marcel Dekker Inc., New York, USA, 1998. (1, 9)

[50] Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker Inc., New York, USA, 1998. (1, 9)

[51] Abdel-Rahman Hedar and Rashad Ismail. **Hybrid Genetic Algorithm for Minimum Dominating Set Problem**. In *Proceedings ICCSA 2010, Part IV, LNCS 6019*, pages 457–467, 2010. (55, 56, 57, 69)

[52] Henk Heijmans and Luc Vincent. **Graph Morphology in Image Analysis**. In E.Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 171–203. Marcel-Dekker, New York, 1992. (5, 6)

[53] Chin Kuan Ho, Yashwant Prasad, and Hong Tat Ewe. **An enhanced Ant Colony Optimization Metaheuristic for Minimum Dominating Set Problem**. *Applied Artificial Intelligence*, **20**:881–903, 2006. (22, 26, 27, 55, 56)

[54] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992. (18)

[55] Mounire El Houmaidi and Mostafa A. Bassiouni. **k-Weighted Minimum Dominating Sets for Sparse Wavelength Converters Placement under Non-uniform Traffic**. In *Proceedings International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS'03)*, pages 56–61, 2003. (2)

[56] Johann L. Hurink and Tim Nieberg. **Approximating Minimum Independent Dominating Sets in Wireless Networks**. *Information Processing Letters*, pages 155–160, 2008. (3, 32, 42, 44, 157)

[57] Raka Jovanovic, Milan Tuba, and Dana Simian. **Ant Colony Optimization Applied to Minimum Weight Dominating Set Problem**. In *Proceedings of the 12th WSEAS international conference on Automatic control, modelling and simulation (ACMOS'10)*, pages 322–326, 2010. (55, 56, 61, 69, 75, 76, 113)

[58] Mong-Jen Kao and Han-Lin Chen. **Approximation Algorithms for the Capacitated Domination Problem**. In *Proceedings of the 4th International Conference on Frontiers in Algorithmics, FAW'10*, pages 185–196, 2010. (32)

[59] Hideki Katagiri, Tomohiro Hayashida, Ichiro Nishizaki, and Qingqiang Guo. **A hybrid algorithm based on tabu search and ant colony optimization for k-minimum spanning tree problems**. *Expert Systems with Applications*, **39**:5681–5686, 2012. (26, 28)

[60] Fabian Kuhn and Thomas Moscibroda. **Distributed Approximation of Capacitated Dominating Sets**. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures SPAA'07*, pages 161–170, 2007. (3, 32, 34, 38, 96, 100, 120, 131)

[61] Fabian Kuhn, Thomas Moscibroda, Tim Nieberg, and Roger Wattenhofer. **Fast deterministic distributed maximal independent set computation on growth-bounded graphs**. In *Proceedings of the 19th international conference on Distributed Computing*, DISC'05, pages 273–287, 2005. (3, 32, 35)

[62] G Leguizamon and Z Michalewicz. **A New version of the ant system for subset problem**. In *Proceedings of the Congress on Evolutionary Computation*, pages 1459–1464, 1999. (22)

[63] Chunhung Richard Lin and Mario Gerla. **Adaptive Clustering for Mobile Wireless Networks**. *IEEE Journal on Selected Areas in Communications, Vol. 15, No. 7*, pages 1265–1275, 1997. (42, 43)

[64] Chunmei Liu and Yinglei Song. **Exact Algorithms for Finding the Minimum Independent Dominating Set in Graphs**. In *Proceedings International Symposium on Algorithms and Computation, ISAAC-2006*, pages 439–448, 2006. (6, 32, 42, 43, 44, 46, 47, 48, 54, 157)

[65] J. Loughry, J.I. van Hemert, and L. Schoofs. **Efficiently Enumerating the Subsets of a Set**, 2000. http://applied-math.org/subset.pdf. (46, 47)

[66] Petros Maragos. **Pattern Spectrum and Multiscale Shape Representation**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**:701–715, 1989. (5)

[67] M.V. Marathe, H. Breu, H. B. Hunt Iii, S. S. Ravi, and D. J. Rosenkrantz. **Simple Heuristics for Unit Disk Graphs**. *Networks*, **25**:59–68, 1995. (42)

[68] Michele Mastrogiovanni. **The Clustering Simulation Framework: A Simple Manual**, 2007. http://www.michele-mastrogiovanni.net/software/download/README.pdf. (49, 61, 69, 74, 113)

[69] Brian McLaughlan and Kemal Akkaya. **Coverage-based Clustering of Wireless Sensor and Actor Networks**. In *Proceedings International Conference on Pervasive Services*, pages 45–54, 2007. (42, 44, 48, 55)

[70] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. **BRITE User Manual**. `http://www.cs.bu.edu/brite/user_manual/node42.html`. (69)

[71] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. **BRITE: An Approach to Universal Topology Generation**. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, MASCOTS-2001*, 2001. (49, 69)

[72] Fernand Meyer and Jean Stawiaski. **Morphology on Graphs and Minimum Spanning Trees**. In *Proceedings of International Symposium on Mathematical Morphology, ISMM-2009*, LNCS 5720, pages 161–170, 2009. (5)

[73] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Prentice-Hall of India, New Delhi, 1998. (20)

[74] Sara Morin, Caroline Gagné, and Marc Gravel. **Ant colony optimization with a specialized pheromone trail for the car-sequencing problem**. *European Journal of Operational Research*, **197**:1185–1191, 2009. (27)

[75] M.R.Garey and D.S.Johnson. *Computers and Tractability, A guide to the theory of NP-Completeness*. Freeman and Company, New York, 1979. (1, 16, 42)

[76] Laurent Najman and Hugues Talbot. *Mathematical Morphology*. John Wiley and Sons, London, UK, 2010. (28)

[77] Tim Nieberg and Johann Hurink. **A PTAS for the Minimum Dominating Set Problem in Unit Disk Graphs**. In *Proceedings of the Third international conference on Approximation and Online Algorithms*, WAOA'05, pages 296–306, Berlin, Heidelberg, 2005. Springer-Verlag. (3, 15, 32, 39)

[78] A. T. Popev. **Morphological operations on fuzzy sets**. In *Proceedings of Fifth International Conference on Image Processing and its Applications*, pages 837–840, 1995. (5)

[79] Marc Reimann and Marco Laumanns. **Savings based Ant Colony Optimization for the Capacitated Minimum Spanning Tree Problem**. *Computers and Operations Research*, **33**:1794–1822, 2006. (22, 27)

[80] Andrea C. Santos, Fatiha Bendali, Jean Mailfert, Christophe Duhamel, and Kean-Mean Hou. **Heuristic for Designing Energy-efficient**

Wireless Sensor Network Topologies. *Journal of Networks, Vol. 4, No. 6*, pages 436–444, 2009. (42)

[81] Luís Santos, Jo ao Coutinho-Rodrigues, and John R. Current. **An improved ant colony optimization based algorithm for the capacitated arc routing problem**. *Transportation Research Part B*, **44**:246–266, 2010. (22)

[82] Lloyd J. Sartor and Arthur R. Weeks. **Morphological operations on color images**. *J. Electron. Imaging*, **10**:548–559, 2001. (5)

[83] Johannes Schneider and Roger Wattenhofer. **A log-star distributed maximal independent set algorithm for growth-bounded graphs**. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 35–44, 2008. (15)

[84] Johannes Schneider and Roger Wattenhofer. **An optimal maximal independent set algorithm for bounded-independence graphs**. *Distributed Computing*, **22**(5-6):349–361, 2010. (3, 32)

[85] Jean Serra. *Image Analysis and Mathematical Morphology.* Academic Press, New York, 1982. (5, 30, 31)

[86] Jean Serra. **Introduction to Mathematical Morphology**. *Computer Vision, Graphics and Image Processing, 35*, pages 283–305, 1986. (28)

[87] Chao Shen and Tao Li. **Multi-Document Summarization via the Minimum Dominating Set**. In *Proceedings 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 984–992, August 2010. (2)

[88] Jianbo Shi and Jitendra Malik. **Normalized Cuts and Image Segmentation**. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 731–, 1997. (5)

[89] Shyong Jian Shyu, Peng-Yeng Yin, and Bertrand M.T. Lin. **An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem**. *Annals of Operation Research, Vol. 131, Numbers 1-4*, pages 283–304, 2004. (4, 22, 61, 67)

[90] Alok Singh and A.K.Gupta. **A Hybrid Heuristic for the Minimum Weight Vertex Cover Problem**. *Asia-Pacific Journal of Operational Research, Vol. 23, No. 2*, pages 273–285, June 2006. (4, 62)

[91] Alok Singh and Anurag Singh Baghel. **New Metaheuristic Approaches for the Leaf-Constrained Minimum Spanning Tree problem**. *Asia-Pacific Journal of Operational Research Volume 25, No. 4*, pages 575–589, 2008. (4, 67)

[92] Christine Solnon. **Boosting ACO with a Preprocessing Step**. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, pages 163–172, 2002. (26, 27, 69, 111)

[93] Christine Solnon and Derek Bridge. **An Ant Colony Optimization Meta-Heuristic for Subset Selection Problems**. *Systems Engineering using Particle Swarm Optimization*, pages 3–25, 2007. (4, 22)

[94] Christine Solnon and Serge Fenet. **A study of ACO capabilities for solving the maximum clique problem**. *Journal of Heuristics, Vol. 12*, pages 155–180, 2006. (4, 22, 26, 67)

[95] Stanley R Sternberg. **Grayscale morphology**. *Comput. Vision Graph. Image Process.*, **35**(3):333–355, Sep 1986. (5)

[96] Thomas Stützle and Holger H. Hoos. $\mathcal{MAX}-\mathcal{MIN}$ **Ant System**. *Future Generation Compute Systems*, **16**:889–914, 2000. (26, 67)

[97] Dhanant Subhadrabandhu, Saswati Sarkar, and Farooq Anjum. **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*, pages 97–107, 2004. (2)

[98] Shyam Sundar, Alok Singh, and André Rossi. **New heuristics for two bounded-degree spanning tree problems**. *Inf. Sci.*, **195**:226–240, 2012. (26)

[99] Johan M. M. van Rooij and Hans L. Bodlaender. **Exact algorithms for dominating set**. *Discrete Appl. Math.*, **159**(17):2147–2164, October 2011. (32)

[100] Luc Vincent. **Morphological Algorithms**. In E.Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 255–288. Marcel-Dekker, New York, 1992. (149)

[101] Yu Wang, Weizhao Wang, and Xiang-Zang Li. **Efficient Distributed Low-Cost Backbone Formation for Wireless Networks**. *IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 7*, pages 681–693, July 2006. (3, 32)

[102] Roger Wattenhoffer. **Distributed Dominating Set Approximation**. `http://www.disco.ethz.ch/lectures/ss04/distcomp/lecture/chapter12.pdf`. (32, 57, 149)

[103] B Waxman. **Routing of Multipoint Connections**. *IEEE Journal of Selected Areas in Communication Vol. 6, No. 9*, pages 1617–1622, 1988. (69)

[104] Ping Wu, Ji-Rong Wen, Huan Liu, and Wei-Ying Ma. **Query Selection Techniques for Efficient Crawling of Structured Web Sources**. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 47–, 2006. (2)

[105] Xu Zhu, Wei Wang, Shan Shan, Zhong Wang, and Weili Wu. **A PTAS for the minimum weighted dominating set problem with smooth weights on unit disk graphs**. *Journal of Combinatorial Optimization*, **23**(4):443–450, May 2012. (3, 33, 40)

[106] Feng Zou, Yuexuan Wang, Xiao-Hua Xu, Xianyue Li, Hongwei Du, Pengjun Wan, and Weili Wu. **New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs**. *Theoretical Computer Science*, 2009. doi:10.1016/j.tcs.2009.06.022. (32)

List of Publications

[1] ANUPAMA POTLURI, ATUL NEGI. **Some Observations on Algorithms for Computing Minimum Independent Dominating Set**. Srinivas Aluru, Sanghamitra Bandyopadhyay, Ümit V. Çatalyürek, Devdatt P. Dubhashi, Phillip H. Jones, Manish Parashar, Bertil Schmidt (Eds.): *Proceedings of International Conference on Contemporary Computing, IC3-2011*, pages 57-68, CCIS 168, Springer 2011, ISBN 978-3-642-22605-2.

[2] ANUPAMA POTLURI, ALOK SINGH. **Two Hybrid Meta-heuristic Approaches for Minimum Dominating Set Problem**. Bijaya K. Panigrahi, Ponnuthurai Nagaratnam Suganthan, Swagatam Das, Suresh Chandra Satapathy (Eds.): *Proceedings International Conference on Swarm, Evolutionary and Memetic Computing*, SEMCCO (2) 2011: 97-104, LNCS 7077, Springer 2011, ISBN 978-3-642-27241-7.

[3] ANUPAMA POTLURI, ALOK SINGH. **A Greedy Heuristic and its Variants for Minimum Capacitated Dominating Set**. Manish Parashar and Dinesh Kaushik and Omer F. Rana and Ravi Samtaney and Yuanyuan Yang and Albert Y. Zomaya (Eds.): *Proceedings of International Conference on Contemporary Computing, IC3-2012*, pages 28-39, CCIS 306, Springer 2012, ISBN 978-3-642-32128-3.

[4] ANUPAMA POTLURI, ALOK SINGH. **Hybrid Metaheuristic Algorithms for Minimum Weight Dominating Set**. Available online 22 July 2012, in the journal *Applied Soft Computing.* http://dx.doi.org/10.1016/j.asoc.2012.07.009. Published by Elsevier.

[5] ANUPAMA POTLURI, ALOK SINGH. **Metaheuristic Algorithms for Computing Capacitated Dominating Set with Uniform and Variable Capacities**.

Communicated to *Swarm and Evolutionary Computation* in June 2012. Published by Elsevier.

[6] Anupama Potluri, Chakravarthy Bhagvati. **Novel Morphological Algorithms for Dominating Sets on Graphs with Applications to Image Analysis**. Accepted for publication in *International Workshop on Combinatorial Image Analysis – IWCIA 2012* to be held in November 2012 in Austin, TX, USA. Published by Springer.