Independent Domination in Odd Graphs

BY

Ahmed Ibrahim Al-Herz

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

February 2012

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **AHMED I. AL-HERZ** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE**.

Thesis Committee

[illegible]

Dr. Mohammed Al-Suwaiyel
(Thesis Advisor)

[illegible]

Dr. Nasir Al-Darwish
(Member)

[illegible]

Dr. Salahadin Mohammed
(Member)

[illegible]

Dr. Adel Ahmed
Department Chairman

[illegible]

Dr. Salam A. Zummo
Dean of Graduate Studies

14/3/12

Date

DEDICATION

I dedicate this work to all my family members, especially my parents, my wife and my son.

ACKNOWLEDGMENT

All praise and thanks are due to **ALLAH** Almighty for his countless and continuous blessings.

Acknowledgment is due to King Fahd University of Petroleum & Minerals for supporting this research.

I wish to express my deep appreciation to my advisor Dr. Mohammed Alsuwaiyel for all the invaluable help and support he gave me throughout the course of this work. His invaluable suggestions made this work interesting and learning for me. I also wish to thank the other members of my thesis committee Dr. Nasir Al-Darwish and Dr. Salahadin Mohammed for their interest, suggestions and support.

Further, I would like to thank Dr. Adel Ahmed, chairman of ICS Department for his encouragement and availing the use of the facilities in the department.

Finally, I wish to express my gratefulness to my parents, my wife and other family members, their prayers and support are always with me.

TABLE OF CONTENTS

LIST OF TABLES

Page No.

LIST OF FIGURES

ABSTRACT

NAME	:	AHMED I. AL-HERZ
TITLE	:	INDEPENDENT DOMINATION IN ODD GRAPHS
MAJOR FIELD	:	COMPUTER SCIENCE
DATE OF DEGREE	:	February 2012

Domination in graph theory is a natural model for many location problems in computer science and operations research. Finding a minimum independent dominating set in general graphs is NP-hard, and it was studied extensively. In this thesis, the first approximation algorithms for independent dominating sets in odd graphs are introduced. Our approach is based on partitioning the graph to different sets in order to simplify the complexity of the graph, then finding an independent dominating set or an independent set in each part, and merging the sets while resolving any violation in the independence or domination properties. Also, we present experimental results and comparisons between the proposed algorithms and greedy and randomized algorithms. The results show that the proposed algorithms give the best approximation quality.

ملخص الرسالة

الإســــــــــم	:	أحمد ابراهيم الحرز
عنوان الدراسة	:	الهيمنة المستقلة في الرسوم البيانية الغربية
التخصـــــــص	:	علوم الحاسب الآلي
تاريخ التخــرج	:	فبراير 2012

الهيمنة في نظرية الرسم البياني تعتبر نموذج طبيعي لكثير من المشاكل المتعلقة بالمواقع في علوم الحاسوب وبحوث العمليات. العثور على الحد الأدنى لمجموعة مستقلة و مهيمنة في الرسوم البيانية العامة يعتبر من المشاكل الحدودية الغير محددة، و هذه المشكلة درست من قبل على نطاق واسع. في هذه الأطروحة، يتم عرض خوارزميات تقريبية للمرة الأولى لمجموعة مهيمنة و مستقلة في الرسم البياني الغريب. ويستند نهجنا على تقسيم الرسم البياني لمجموعات مختلفة من أجل تبسيط التعقيد في الرسم البياني والعثور على مجموعة مستقلة تهيمن على الأجزاء المقسمة من الرسم البياني، ثم دمج النتائج في حين حل أي اشكال في خصائص الاستقلال أو الهيمنة. وبالاضافة الى ذلك، نقدم نتائج تجريبية ومقارنة بين الخوارزميات التقريبية المقترحة و الخوارزميات الجشعة و العشوائية. نتائج التجارب تظهر أن الخوارزميات التقريبية المقترحة تعطي نتائج أفضل بالنسبة لحجم المجموعة وخصوصا على الرسوم البيانية الغربية ذات الحجم الكبير.

CHAPTER 1

INTRODUCTION

Domination in graph theory is a natural model for many location problems in computer science and operations research. Domination has many applications in the real world [Hayn97]. Examples of such applications are dominating queens, sets of representatives, school bus routing, computer communication networks, radio stations, social network theory, computer vision [Booi07], pattern recognition [Prie01], scheduling [Bala06], VLSI design [Kuo88], molecular biology [Hayn06], etc.

The minimum independent dominating set (MIDS) is one variant of domination problems which is a well known combinatorial optimization problem. The problem can be defined informally as follows: given a graph, a minimum independent dominating set is a set of vertices of minimum cardinality with the requirement that the dominating vertices are independent, that is none of the vertices are adjacent and every other vertex not included in the set is adjacent to at least one of the vertices in the set. An example of a minimum independent dominating set (the set of black vertices) in a graph can be seen in Figure 1. Before stating the problems formally we will give some definitions in the next section.

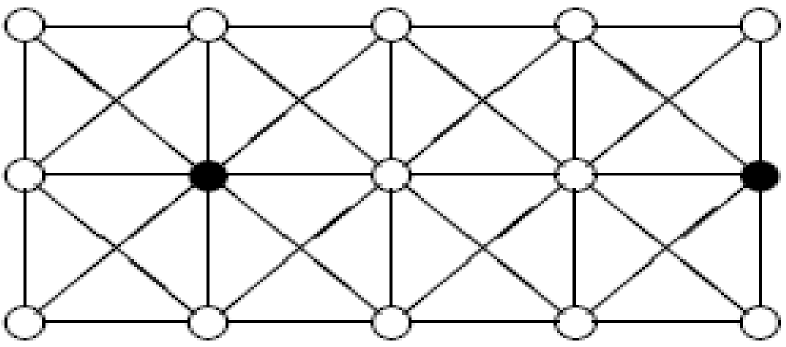


Figure 1: A minimum independent dominating set on a graph [Chan98a].

1.1 DEFINITIONS AND NOTATION

Throughout this thesis all graphs are finite, undirected and simple (i.e. loop–free and without multiple edges). Given a graph $G(V,E)$ where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, and two vertices u, $v \in V$ have an edge between them, or are said to be *adjacent*, if and only if $(u, v) \in E$. If $(u, v) \notin E$, we say that (u, v) is a *non-edge*. Let a vertex $v \in V$, the neighborhood $N(v)$ of v is the set of vertices that are adjacent to v, and $N[v] = N(v) \cup \{v\}$ will be called the closed neighborhood of v. For the degree of v, we use the notation $deg(v) = |N(v)|$, where $| \, . \, |$ is the cardinality of a set which is the number of elements in a set. For any subset $H \subset V$, we denote by $G[H]$ the subgraph of G induced by H. For $v \in H$, for some subset H, we denote by $deg'\ H(v)$ the degree of v in $G[H]$ or, if it is clear by the context, we denote it by $deg'(v)$. For convenience, we set $N[H] = \{N[v] : \forall\ v \in H\}$. For simplicity, we may set $n = |V|$ and $m = |E|$.

Definition 1.1 A *dominating set D* in a graph $G(V, E)$ is a subset of V in which each vertex $v \in (V - D)$ is adjacent to at least one vertex $u \in D$, i.e., $(v, u) \in E$. An independent dominating set is a dominating set where all vertices in D are independent, i.e., $(u, v) \notin E$, for all $u, v \in D$. The optimization version of the independent domination problem is finding the independent dominating set D such that the cardinality of D is minimum.

Definition 1.2 A maximal independent set M is an independent set of a graph $G(V, E)$ that is not a subset of any other independent set. That is, it is a set such that every edge $(v, u) \in E$ has at least one endpoint not in M and every vertex not in M has at least one neighbor in M. A maximal independent set is also a dominating set in the graph, and every dominating set that is independent must be maximal independent set, so maximal independent sets are also independent dominating sets. A graph may have many maximal independent sets of varying sizes; a largest maximal independent set is called a maximum independent set.

Definition 1.3 The decision version of the independent dominating set problem can be stated as follow:

Instance: $G = (V, E)$, positive integer $K \leq |V|$

Question: Is there a dominating set of size K or less for G, i.e., a subset V' $\subseteq V$ with $|V'| \leq K$ such that for all $u \in V\text{-}V'$ there is a $v \in V'$ for which $(u, v) \in E$?

Definition 1.4 The optimization version of the independent dominating set problem can be stated as follow:

Instance: $G = (V, E)$

Question: Is there a dominating set for G, i.e., a subset $V' \subseteq V$ with $|V'| = K$ such that for all $u \in V\text{-}V'$ there is a $v \in V'$ for which $(u, v) \in E$ and K is minimum?

Theorem 1.1 The decision version of independent dominating set problem is $\mathbb{NP}$-complete. The proof can be found in [Gare79].

Theorem 1.2 The minimum independent dominating set problem is $\mathbb{NP}$-hard. The proof can be found in [Gare79]. Knowing that an $\mathbb{NP}$ problem is $\mathbb{NP}$-hard, we also know that we cannot compute an optimal solution in polynomial time, unless $\mathbb{P} = \mathbb{NP}$.

1.2 ODD GRAPHS

Because the independent dominating set problem for general graphs is hard, researchers turned their attention to solving the problem on restricted families of graphs. Each family of graphs may have special properties or unique structures, which can be used to come up with polynomial time or approximation algorithms. In this thesis, we will consider odd graph family that has a unique structure. The family of odd graphs was introduced by [Bigg79] in the context of graph theory. In this section, we will introduce the odd graphs, more details on odd graphs and their properties will be given in chapter 3.

Definition 1.5 For a positive integer d, let $\Omega = \{1, 2, \ldots, 2d - 1\}$ and $V = \{\{x_1, x_2, \ldots, x_{d-1}\} \mid x_i \in \Omega\}$, that is, the set of all (d-1)-subsets of Ω. The odd graph $O_d = (V, E)$ is defined as the graph with V as its vertex set and two vertices are connected if and only if their corresponding subsets are disjoint.

O_d is a d-regular graph ($deg(v) = d \;\forall\, v \in V$) with $n = \binom{2d-1}{d-1}$ vertices and $m = \frac{d}{2}\binom{2d-1}{d-1}$ edges. We will refer to d as the dimension of O_d. In particular, the 3-dimensional odd graph is the well-known Peterson graph. Figure 2 shows typical drawings of O_d, d = 2, 3, 4. The odd graph of dimension 1 consists of one vertex and no edges.

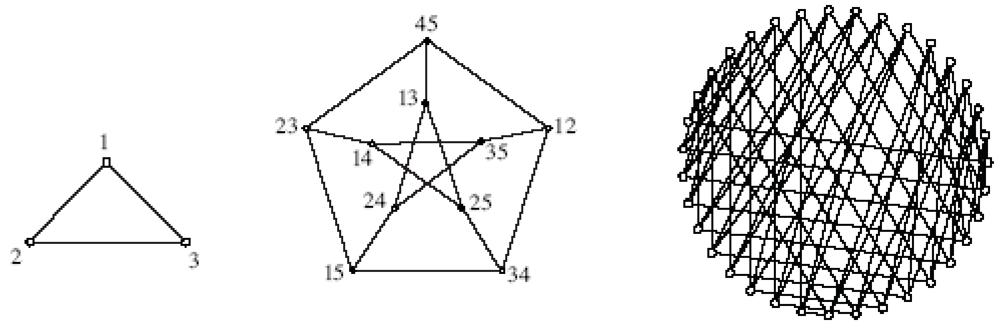


Figure 2: Drawings of O_d, d = 2, 3, 4.

1.3 APPLICATIONS OF MIDS IN COMMUNICATION NETWORKS

From an application point of view, independent and dominating set in a communication networks are important structures, and many optimization approaches rely on these structures.

In clustering schemes, independent sets result in clusterheads that have local control of their cluster without interference. Additionally, a dominating independent set based clustering scheme ensures that the entire network is covered. For example, especially in energy-efficient computing, clustering allows for some nodes to perform fewer tasks by delegating them to their respective clusterhead. On the other hand, the tasks of these clusterheads then result in additional energy consumption. Here, using as few clusterheads as possible, i.e. choosing them according to minimum independent dominating set, results in energy savings for the network.

A standard approach for reducing energy consumption is to carefully schedule node activity. As has been observed in [Chen02], whenever there are sufficiently many nodes in a region, only a small fraction of nodes need be active for forwarding messages, etc. The rest of the nodes can enter a sleep mode, thereby conserving energy. The problem of maximizing the number of nodes which are asleep at any given time while maintaining sufficient activity in the network is usually modeled as the problem of finding a small dominating set in the network. Once a small dominating set is found, the nodes in the dominating set collectively act as “coordinators” for the network and the rest of the nodes go to sleep.

In a communication network, broadcasting schemes are required. Each individual node is neither able to store the entire topology information, nor to keep updated information about the changes in the network. The broadcasting schemes have relied on flooding the network. Basic, network-wide flooding causes the broadcast storm problem [Ni99], resulting in excessive contention and collisions, i.e. a large communication protocol overhead. Using a dominating set of small size to propagate flooding messages overcomes this problem, and greatly reduces the number of messages needed, and thus the protocol overhead as well. So, nodes in an independent set do not interfere each other during simultaneous transmissions, and nodes in a dominating set can be used to efficiently reach the entire network by broadcasts from only these nodes, these two properties can be achieved by minimum independent dominating set.

1.4 OBJECTIVE OF THE RESEARCH

Although considerable amount of works for the independent domination problem have emerged in the past, the first algorithmic result on this topic was given by Bayer, Proskurowski, Hedetniemi and Mitchell in 1977 [Byer77]. They gave a linear-time algorithm for the independent domination problem on trees. On the other hand, at about the same time Garey and Johnson [Gare79] constructed the first proof that the domination problem is $\mathbb{NP}$-complete for general graphs. Since then, many algorithmic results are studied for variants of the domination problem in different classes of graphs.

One of the graph classes, which have not been investigated in term of independent domination, is the odd graphs class. [Ghaf91] pointed out their potential as fault-tolerant multiprocessor networks. Their efficiency was analyzed in terms of routing, combinatorial structure, maximal fault tolerance [Ghaf91], symmetry [Bigg79], fault diameter [Ghaf91], [Kim08a]. Odd networks are competitive with mesh and hypercube variants. For the same number of nodes, odd networks are superior to comparable mesh and hypercube variants when the network cost (degree×diameter) is used as a measure.

The minimum independent dominating set is very important problem in communication networks; this is most obvious in parallel computing systems. Also, finding a solution to the minimum independent dominating set on networks with very large number of vertices, such as a high dimensional odd graph, can be time consuming; in this situation an approximate solution can be much more efficient. So, a natural question arises of whether an approximate solution to the minimum independent dominating set problem on odd graph network within an acceptable time is feasible or not. Thus, the primary objective of this thesis can be stated as ”designing an efficient approximation algorithm for the minimum independent dominating set problem by exploiting the unique structure of the class of odd graphs”. Another objective of this thesis is comparisons of our proposed approximation algorithm with generic approximation algorithms namely, simple greedy and randomized heuristics.

CHAPTER 2

LITERATURE REVIEW

Many approaches were used to find the minimum or an approximate independent dominating set on a graph. These approaches range from enumeration of all sets of the vertices to solving the problem for special graph classes. Next, we will review the literature regarding the approaches that have been used to solve this problem.

2.1 BRUTE FORCE

The minimum independent dominating set problem can be trivially solved in $O(2^n)$ by simply enumerating all the subsets of V, and check whether the set is dominating and independent with minimum cardinality. Clearly this approach is exponential and not practical.

2.2 EXACT ALGORITHMS

Many attempts have been done to design efficient but yet exponential algorithms that give optimal solution for $\mathbb{NP}$-complete problem. The first work that gives an exact exponential time algorithm for minimum independent dominating set has been done by Randerath and Schiermeyer [Rand04]. They used the result due to Moon and Moser [Moon65] who showed in 1965 that the number of maximal independent sets of a graph is upper bounded by $3^{n/3}$. They used an algorithm enumerating all the maximal independent sets to obtain an $O(1.4423^n)$ time algorithm for the minimum independent dominating set. Gaspers and Liedloff [Gasp06] presented an $O(1.3569^n)$ time algorithm for solving the minimum independent dominating set using the Measure & Conquer approach to analyze its running time. A simple $O*(\sqrt[3]{3}^n)$ time algorithm based on a maximal matching was developed by Liu and Song [Liu06] to solve this problem on general graphs. Here, $O*(.)$ implies the existence of an additional polynomial factor in the corresponding time complexity result. For sparse graphs, e.g. graphs with degree bounded by 3 and 4, they showed that a few new branching techniques can be applied to these graphs and the resulting algorithms have time complexities $O*(1.3803^n)$ and $O*(1.5368^n)$. Bourgeois, Escoffier and Paschos [Bour10] devised a branching algorithm that can find a minimum independent dominating set on any graph with running time $O*(1.3416^n)$ and polynomial space.

2.4 APPROXIMATION ALGORITHMS

It was shown that the minimum dominating set can be approximated with a constant factor if we apply the algorithm on restricted types of graphs. An algorithm which gives a constant performance ratio independent of the size of the instance is referred to as constant-factor approximation. Hurink and Nieberg [Huri08] presented the first polynomial-time approximation scheme ($\mathbb{PTAS}$) for the minimum independent dominating set problem in graphs of polynomially bounded growth. Graphs of bounded growth are used to characterize wireless communication networks. The algorithm accepts any undirected graph of bounded growth as input, and returns a ($1+\varepsilon$)-approximate minimum dominating set, where ε is a real number greater than 0. Duckworth and Wormald [Duck02] presented a heuristic, which is a random greedy algorithm, for finding a small independent dominating set of cubic graphs. They proved that D, the minimum independent dominating set, asymptotically almost surely satisfies $0.2641n \leq |\mathrm{D}| \leq 0.2794n$. A deterministic version of the randomized algorithm was analyzed in [Duck10] using linear programming. It was shown that, given an n-vertex cubic graph, the deterministic algorithm returns an independent dominating set of size at most $29n/70 + O(1)$. Bourgeois, Escoffier and Paschos [Bour10] showed that, for every $r > 3$, it is possible to compute an

$r-((r-1)/r)$ $\log_2 r$-approximate solution (If an algorithm guarantees to return solutions with a performance guarantee of at most r, then the algorithm has an r-approximate solution) for the minimum independent dominating set within time $O^*(2^{n\log_2 r/r})$.

2.5 SPECIAL GRAPH CLASSES

One of the sites for research on ℕℙ -complete graph problems is to consider the algorithmic complexity when they are restricted to special graph classes. The motivation was to find graph classes with nice structural properties, that enable the design of polynomial time algorithms for ℕℙ-complete graph problems when the input graphs are restricted to the special graph class. Originally small classes such as interval graphs and permutation graphs were considered. This led researches to look for larger graph classes, for which polynomial time domination algorithms can still be design. Recent examples are the classes of AT-free graphs, dually chordal graphs and homogeneously orderable graphs.

This section reviews the literature regarding the special graphs classes and whether the minimum independent dominating set can be found in a linear time or it cannot be solved linearly that is it belongs to ℕℙ-complete class.

2.5.1 SPECIAL GRAPH CLASSES (MIDS PROBLEM IS IN $\mathbb{P}$)

In 1977 T. Byer et al. proved that minimum independent dominating set in trees can be computed in linear time [Byer77]. M. Faber discovered in 1982 that minimum independent dominating set can be obtained in linear time in chordal graphs [Fabe82]. He presented a linear algorithm to locate a minimum weight independent dominating set in a chordal graph with 0-I vertex weights. The problem was put into the framework of linear programming. In particular, they exhibited a linear program with 0-1 solutions which correspond to independent dominating sets in the given graph. The algorithm utilizes perfect elimination ordering of choral graphs. Using the same methodology they solved the problem in strongly chordal graphs given a strong elimination ordering [Fabe84]. Moreover, minimum independent dominating set for doubly chordal graph, split graph and undirected path graph were proved to be solvable linearly since these special graph classes are subset of chordal graph. The minimum independent dominating set for series parallel graph can be found linearly which was discovered by J. Pfaff, R. Laskar and S.T. Hedetniemi in 1984 [Pfaf84]. M. Atallah and S. Kosaraju proved in 1989 that permutation graph's independent dominating set is linearly solvable [Atal88]. They reduced the problem of finding the minimum independent dominating set to the problem of computing a shortest maximal increasing subsequence in linear time, the shortest maximal increasing subsequence problem is solvable in linear time, and thus the problem of minimum independent dominating set is also solvable in linear time. M. Faber presented an algorithm in 1989 to solve the minimum independent dominating set in linear time for $2K_2$-free graphs [Fabe89] . In 1990 E. Elmallah and L. Stewart discovered that k-polygon graph's independent dominating set can be solved in linear time [Elma90]. The independent dominating set for partial k-tree for bounded k is also in P and was proved by S. Arnborg, J. Lagergren and D. Sees in 1991 [Arnb91]. They transformed the graph of bounded tree width formulated as second order logic sentences to binary tree in linear time, then the decision if the graph has an independent dominating set of certain cardinality can be determined if the satisfiablity of monadic second order problem on a binary tree can be decided which can be done in linear time. Minimum independent dominating set can be solved in cocomparability graphs in linear time by a dynamic programming approach using a linear scan through the labeling of the given graph, this approach was presented by Kratsch and Stewart in 1993 [Krat93]. AT-free graph is one of the special graphs that its independent domination set can be obtained in linear time and was discovered by H. Broersma, T. Kloks, D. Kratsch and H. Muller in 1997 [Bro97]. M. Chang proposed algorithms to solve the minimum independent dominating set in linear time on interval and circular-arc graphs [Chan98b]. In 1999 J. Bang-Jensen, J Huang, G. Macgillivary and A. Yeo presented an algorithm that solve the minimum independent dominating set for convex bipartite graph linearly [Bang99]. Furthermore, the convex–round graphs' minimum independent dominating set is also solved in linear time [Bang99]. Claw-free AT-free graph's minimum independent dominating set is in P, which was proved by H.Hempel and D. Kratsch in 1999 [Hemp99]. They used lexicographic breadth first search procedure to label the vertices then they used 2-lexicographic breadth first scheme which is a vertex ordering and levels of the labeled graph. The algorithm exploits the information obtained from the scheme to find the set in linear time.

On the other hand, some special graph classes are proved to be in NP class, which means it cannot be solved in linear time, so either approximation algorithms are used to find sub optimal set or exact algorithms are used to find the optimal set. Many graphs have been discovered that can be categorized nonsolvable linearly, next we will review these special graph classes.

2.5.2 SPECIAL GRAPH CLASSES (MIDS PROBLEM IS IN ℕℙ-COMPLETE)

The first special graph class, that its minimum independent dominating set was discovered to be in ℕℙ-complete class, is line graph which was proved by M. Yannakakis and F. Gavril in 1980 [Yann80]. They proved that the edge dominating set problem for bipartite graphs and planar with maximum degree 3 is ℕℙ-complete using reduction from the SAT-3-restricted problem and the node cover problem on planar cubic graphs respectively. The proof is true for the independent dominate edges, since the independent set can be obtained from the dominating set in linear time. The edge version of domination can be thought of as the vertex version of the problem applied to line graphs. Bipartite graph minimum independent dominating set is not solvable linearly which was discovered by D. Corneil and Y. Perl in 1984 [Corn84]. Also, minimum independent dominating set for comparability graphs and triangle-free graphs was discovered to be in the same class [Corn84]. The reduction they used is from the h-dominating set problem for general graphs which is NP-Complete. In 1990, P. Damaschke, H. Muller and D. Kratsch proved that chordal bipartite minimum independent dominating set problem is in ℕℙ-complete class by reduction from the 3SAT problem [Dama90]. Planar graph and planar bipartite graph minimum independent dominating set is also in ℕℙ-complete class, this was discovered in 1995 by I. Zverovich and V. Zverovich [Zver95]. They used a linear reduction from dominating set problem for 3-regular planar graph. In 1998 A. Brandstast, V. Chepoli and F. Dragan proved that dually chordal graph minimum independent dominating set is in ℕℙ-complete [Bran98]. Moreover, minimum independent dominating set for homogeneously orderable graph is not solvable linearly [Bran98]. The reduction they used is from the independent dominating set problem for general graphs which is ℕℙ-complete problem.

CHAPTER 3

ODD GRAPHS

3.1 INTRODUCTION

Suppose d is an integer not less than 2 and Ω is a set of odd cardinality $2d - 1$, e.g. $\Omega = \{1, 2, \ldots, 2d - 1\}$. Odd graph O_d can be defined as follows: the vertex set V of O_d, is the set of subsets v of Ω which have cardinality $|v| = d - 1$, and two vertices are adjacent when the subsets are disjoint [Bigg79]. The graphs O_2 (= $K3$), O_3 (= Petersen's graph) and O_4 are depicted in Figure 2 in section 1.2.

We will refer to the elements of Ω as labels. A pair of adjacent vertices in O_d, corresponds to a pair of disjoint (d - 1)-subsets of the ($2d$ - 1)-set Ω so there is just one label "the odd label" not occurring in either of the subsets. This "odd" label will be assigned to the edge joining the two vertices. Thus the edge set E of O_d, is partitioned into $2d$ - 1 disjoint sets,

$$E_\omega = \{(u,v) \in E \mid u \cup v = \Omega - \omega\}$$

Since a given vertex v contains d - 1 labels and $|\,\Omega\,|$ = 2d - 1, there are d labels available for the edges incident with v. This shows that O_d is a regular graph with degree d. By simple counting arguments we have,

$|V| = \binom{2d-1}{d-1}$

$|E| = \frac{1}{2}\, d\, \binom{2d-1}{d-1} = \frac{1}{2}\,(2d-1)\binom{2d-2}{d-1}$

$|E_\omega| = \frac{1}{2}\binom{2d-2}{d-1} = \binom{2d-3}{d-2}$

3.2 INDEPENDENT SETS AND CHROMATIC NUMBERS

For each label ω in Ω define the subset V_ω of V to be the set of vertices which contain ω**.** Since any two vertices in V_ω intersect, they are not adjacent and V_ω is an independent set in O_d. The cardinality of V_ω is $\binom{2d-2}{d-2}$ [Bigg79]. The set-theoretical result of Erdos et al. [Erdo61][Hilt67] has the following consequence:

Theorem 3.1: Let I be any independent set of vertices in O_d. Then $|\,I\,| \leq \binom{2d-2}{d-2}$ and if $|\,I\,| = \binom{2d-2}{d-2}$ we must have $I = V_\omega$ for some ω in Ω.

An independent set is maximal if the addition of any new vertex destroys its independence. Theorem 3.1 characterizes the maximal independent sets in O_d which also have maximum cardinality. Now let us consider the maximal independent sets of minimum cardinality. If M is any maximal independent set, then every vertex not in M must be adjacent to at least one vertex in M. Hence the sets,

$$D(m) = \{v \in V \mid v = m \text{ or } (v, m) \in E\}, \ (m \in M)$$

must cover V. In a d-regular graph (such as O_d), $|D(m)| = d + 1$, *so* $(d + 1)\ |M| \geq |V|$ [Bigg79].

The bounds on the cardinality of a maximal independent set M in O_d which were obtained by Biggs are as follows:

$$\binom{2d-2}{d-2} \geq |\ M\ | \geq \frac{1}{d+1} \binom{2d-1}{d-1}$$

The upper bound is attained for every value of $d \geq 2$, but, the lower bound is rarely attained.

The set $V_{\overline{\omega}}$ of vertices, not containing the label ω, has cardinality $\binom{2d-2}{d-1}$. The members of $V_{\overline{\omega}}$ are paired by the rule that (u, v) is a pair when u and v are complementary subsets of Ω - ω. The paired vertices are joined by an edge whose label is ω (these are the only edges in the vertex subgraph $O_d[V_{\overline{\omega}}]$). The previous observations can be combined to obtain a useful "standard representation" of O_d, as in Figure 3 [Bigg79]. The diagram indicates that each vertex in V_{ω} is joined to d vertices in $V_{\overline{\omega}}$, while each vertex in $V_{\overline{\omega}}$ is joined to d - 1 vertices in V_{ω}. The edges in $O_d[V_{\overline{\omega}}]$ are just those in the set E_{ω}.

It is easy to check that O_d contains odd circuits, and the standard representation indicates at once that there is a proper 3-coloring of the vertices.

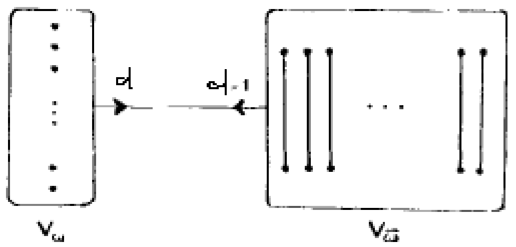


Figure 3: Representation of O_d [Bigg79].

Theorem 3.2 [Zeli85]: The chromatic number of every odd graph is equal to 3.

Proof:

Consider an odd graph O_d. Let U_1 be the set of all sets belonging to V and containing the label 1, let U_2 be the set of all sets belonging to V - U_1 and containing the label 2, let $U_3 = V - (U_1 \cup U_2)$. Any two elements of U_1 are non-adjacent (as vertices of O_d), because their intersection contains the label 1 and therefore it is non-empty. Hence U_1 is an independent set in O_d and analogously so is U_2. Now let $X \in U_3$, $Y \in U_3$. Then the sets X, Y are subsets of the set Ω - {1, 2}. This set has the cardinality $2d - 3$, while each of the sets X, Y has the cardinality $d - 1$. If X, Y were disjoint, their Union $X \cup$

Y would have the cardinality $2(d - 1)$ which is greater than the cardinality of $\Omega - \{1,2\}$; this is impossible. Therefore $X \cap Y \neq \emptyset$ for any two elements X, Y of U_3 and U_3 is an independent set in O_d too. The vertices of O_d can be coloured by three colours 1, 2, 3 in such a way that by the colour i ($i = 1, 2, 3$) the vertices belonging to U_i are coloured. This colouring is admissible; no two vertices of the same colour are adjacent. We have proved that $\chi(O_d) \leqq 3$, where $\chi(O_d)$ is the chromatic number of O_d.

Now we shall construct the sets $X_1, ..., X_d$ and $Y_1, ..., Y_d$ as follows. We put $X_1 = \{1, ..., d - 1\}$. If X_i is constructed for some i, then we put $Y_i = \Omega - (X_i \cup \{2d - i\})$. If Y_i is constructed for some i, then we put $X_{i+1} = \Omega - (Y_i \cup \{i\})$. The reader himself may verify that then $Y_d = X_1$. Further $X_i \cap Y_i = \emptyset$ for $i = 1, ..., d$ and $X_{i+1} \cap Y_i = \emptyset$ for $i = 1, ..., d-1$. Therefore $X_1, Y_1, X_2, Y_2, ..., X_d, Y_d = X_1$ are vertices of a circuit in O_d having the length $2d - 1$ which is an odd number. Hence O_d is not bipartite and $\chi(O_d) = 3$. Together with the previous inequality this yields $\chi(O_d) = 3$.□

3.3 SHORTEST DISTANCE AND DIAMETER

Theorem 3.3 [Bigg79]**:** In the graph O_d the possible values of $\partial(u, v)$ are in one-to-one correspondence with the possible values $(0, 1, . . . , d - 1)$ of $| u \cap v |$; explicitly,

$$\partial(u, v) = 2r \Leftrightarrow | u \cap v | = (d - 1) - r$$

$$\partial(u, v) = 2r + 1 \Leftrightarrow |u \cap v| = r$$

Proof:

Let ∂ denote the usual distance function and $\mathfrak{D}_i(u)$ denote the set of vertices v such that $\partial(u, v) = i$. Clearly, $\mathfrak{D}_0(u) = \{u\}$, and $\mathfrak{D}_1(u)$ consists of the d vertices adjacent to u. If $\partial(u, v) = 2$, then there is a vertex x adjacent to (that is, disjoint from) both u and v. If the edges (u, x) and (x, v) carry the labels σ and τ, respectively, we see that the subset v is obtained from u by removing the label τ and substituting σ. Thus, $|u \cap v| = d - 2$.
Conversely, any pair of $(d - 1)$ subsets which overlap in all except one element must be separated by two steps in O_d. Continuing in this way, it can be seen that if $\partial(u, v) = 2r$, then v can be obtained from u by removing r labels and substituting r different ones, so that $|u \cap v| = (d - 1) - r$.
Similarly, if $\partial(u, v) = 2r + 1$, then $|u \cap v| = r$.□

Theorem 3.4 [Zeli85]: Let u, v be two vertices of the graph O_d, let $|u \cap v| = r$. Then the distance of the vertices u, v in O_d is $\Delta(r) = \min(2r + 1, 2d - 2r - 2)$.

Proof:

If for two pairs u_1, v_1 and u_2, v_2 of vertices of O_d we have $|u_1 \cap v_1| = |u_2 \cap v_2|$, then evidently there exists a permutation of the set Ω which maps u_1 onto u_2 and v_1 onto v_2 as we will see in section 3.4; this permutation induces an automorphism of O_d which again maps u_1 onto u_2 and v_1 onto v_2. This implies that the distance of two vertices of O_d is a function of the cardinality of their intersection and we may denote it by $\Delta(r)$, where r is this cardinality. Now let us have two vertices u, v of O_d, let $r = | u \cap v |$. If $r = 0$, then $u \cap v = \emptyset$ and the vertices u, v are adjacent; their distance is 1, therefore $\Delta(0) = 1$, which fulfills the assertion. If $r = d - 1$, then $u = v$, because $| u | = | v | = d - 1$. The distance of u and v is 0, therefore $\Delta(d\text{-}1) = 0$, which again fulfils the assertion. Now let r be an arbitrary integer such that $2 \leq r \leq d - 2$. We have $| u - v | = | v - u | = d - 1 - r$, $| \Omega - (u \cup v) | = r + 1$. Let P be the shortest path in O_d connecting u and v. Let u_0 (or v_0) be the vertex of P adjacent to u (or v respectively). Evidently $\partial(u, v) = \partial(u_0, v_0) + 2$, where ∂ denotes the distance of two vertices. We have $u \cap u_0 = v \cap v_0 = \emptyset$, therefore the intersection $u_0 \cap v_0 \subseteq \Omega - (u \cup v)$ and $| u_0 \cap v_0 | \leq r + 1$. On the other hand, the set u_0 can have at most $d - 1 - r$ elements in common with v and the other vertices of u_0 belong to $\Omega - (u \cup v)$, hence $| u_0 \cap (\Omega - (u \cup v)) | \geq r$ and analogously $| v_0 \cap (\Omega - (u \cup v)) | \geq r$. This implies $| u_0 \cap v_0 | \geq r - 1$. Thus there are three possibilities for the cardinality of $u_0 \cap v_0$, namely $r - 1$ or $r + 1$. As P is the shortest path connecting u and v, the sets u_0, v_0 must be chosen so that their distance might be the least possible, i.e, $\partial(u_0, v_0) = \min(\Delta(r - 1), \Delta(r), \Delta(r+ 1))$. As $\Delta(r) = \partial(u, v) = \partial(u_0, v_0) + 2$, the equalities $\partial(u_0, v_0) = \Delta(r)$ and $| u_0 \cap v_0 | = \mathrm{r}$ are impossible. There can be only either $\partial(u_0, v_0) = r - 1$ and,

$\Delta(r) = \Delta(r - 1) + 2$, or $\partial(u_0, v_0) = r + 1$ and $\Delta(r) = \Delta(r + 1) + 2$. Suppose that $\Delta(r) = \Delta(r - 1) + 2$ holds, hence $\partial(u_0, v_0) = \Delta(r - 1)$ and $| u_0 \cap v_0 | = r - 1$. If $r = 1$, then u_0, v_0 are adjacent and $\partial(u, v) = \Delta(l) = 3$ (evidently it cannot be less) which fulfills the assertion. If $r \geq 2$, consider the interrelation between $\Delta(r - 1)$ and $\Delta(r - 2)$. Analogously there is $\Delta(r - 1) = \Delta(r - 2) + 2$ or $\Delta(r - 1) = \Delta(r) + 2$. But, as we have supposed $\Delta(r) = \Delta(r - 1) + 2$, we must have $\Delta(r - 1) = \Delta(r - 2) + 2$. Inductively we can prove that if $\Delta(r) = \Delta(r - 1) + 2$ for some m, then $\Delta(p) = \Delta(p - 1) + 2$ for each integer p such that $2 \leq p \leq r$. Analogously if $\Delta(r) = \Delta(r + 1) + 2$ for some r, then $\Delta(q) = \Delta(q +1) + 2$ for each integer q such that $r \leq q \leq d - 2$. As it has been proved $\Delta(0) =1$, $\Delta(d - 1) = 0$, the function $\Delta(r)$ is uniquely determined as $\Delta(r)= \min(2r + 1, 2d – 2r - 2)$. □

Corollary 3.1 [Zeli85]: The diameter and the radius of the graph O_d are both equal to $d - 1$.

The number $d - 1$ is evidently the maximum of $\Delta(r)$; it is attained in $r = \frac{1}{2}(d - l)$ for d odd and in $r = \frac{1}{2} d - l$ for d even. As O_d is vertex-transitive, its radius is equal to its diameter.

Theorem 3.5 [Zeli85]: The graph O_d for every integer $d \geq 2$ is geodetic. A graph is geodetic if for every pair for vertices the shortest path between them is unique.

Proof:

In the proof of Theorem 3.4 it was shown that for given vertices *u*, *v* the vertices u_0, v_0 (the vertices adjacent to u and v respectively in the shortest path connecting *u* and *v*) are determined uniquely. Thus by induction we can prove that whole the shortest path between *u* and *v* is uniquely determined. □

The graph O_d is an example of a geodetic graph of the diameter $d - 1$ which is simultaneously regular of the degree *d*.

3.4 SYMMETRY AND THE SPECTRUM

Any permutation π of the set Ω induces an automorphism of O_d since the subsets $\pi(u)$ and $\pi(v)$ are disjoint if *u* and *v* are. Thus the symmetric group S_{2d-1} is a subgroup of the automorphism group Aut(O_d).

Theorem 3.6 [Bigg79]**:** The automorphism group of O_d is the symmetric group S_{2d-1}, acting in the obvious way on the $(d - 1)$-subsets of the $(2d - 1)$ set Ω**.**

Proof:

To show that Aut(O_d) = S_{2d-1} the deep result of Theorem 3.1 can be used.

Any automorphism θ of O_d must take an independent set of vertices to an independent set with the same cardinality; hence, by Theorem 3.1, θ $(V_\sigma) = V_\tau$ for some τ in Ω.

Let $\bar{\theta}$ be the corresponding induced permutation of Ω, defined by θ $(\sigma) = \tau$ if and only if θ $(V_\sigma) = V_\tau$. The mapping $\theta \mapsto \bar{\theta}$ is a homomorphism of Aut(O_d) -into S_{2d-1} and it is onto by the remarks at the beginning of this paragraph. Finally, it is one-to-one, since if $\bar{\theta}$ is the identity, then θ $(V_\omega) = V_\omega$ for each ω in Ω; thus if the vertex x contains label ω, so does θ (x), and consequently θ $(x) = x$. □

Suppose u, v, x, y are vertices of O_d, and $\partial(u, v) = \partial(x, y)$. Then Theorem 3.3 tells us that $| u \cap v | = | x \cap y |$, and so a permutation of Ω may be constructed which takes u to x and v to y. This means that the graph O_d is distance-transitive, and a battery of algebraic results may be applied to it [Bigg74]. The intersection array is a rectangular array in which the ith column has three entries c_i, a_i, and b_i, defined as follows. Let u and v be any pair of vertices such that $\partial(u, \text{v}) = i$ (all such pairs are equivalent in O_d , by the distance-transitive property); set,

$$\left.\begin{matrix} ci \\ bi \\ ai \end{matrix}\right\} = \text{The number of vertices x which are adjacent to v and satisfy } \partial(u, \text{x}) = \left\{\begin{matrix} i-1 \\ i \\ i+1 \end{matrix}\right.$$

[Bigg79] Figure 4 may clarify the definitions.

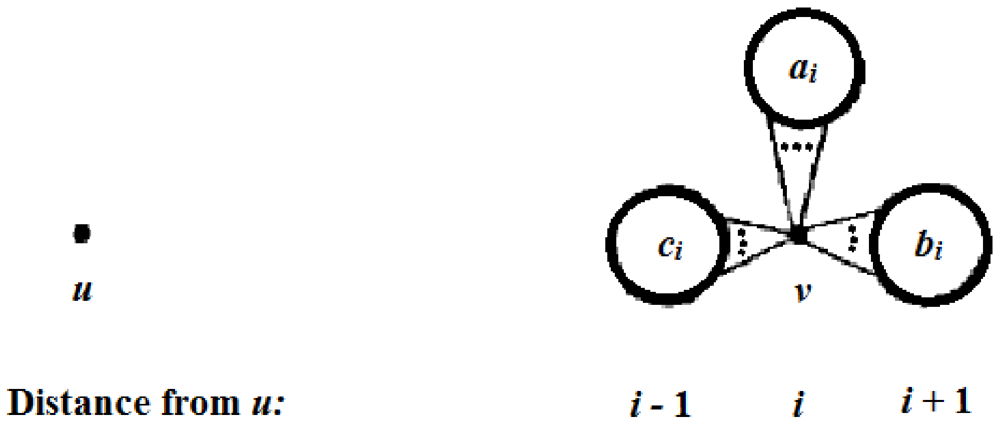


Figure 4: Intersection numbers [Bigg79].

Since the degree of v is d, we have $c_i + a_i + b_i = d$, and, since the diameter is d - 1, there are d columns (i = 0, 1, . . . , d - 1), the numbers c_0 and b_{d-1} being undefined. Simple counting arguments lead to the explicit array for O_d, which has a remarkable pattern. When d is even, [Bigg79] obtain,

$$\begin{array}{ccccccccc}
 & 1 & 1 & 2 & 2 & \ldots\ldots\ldots \frac{1}{2}d - 1 & \frac{1}{2}d - 1 & \frac{1}{2}d \\
0 & 0 & 0 & 0 & 0 & \ldots\ldots\ldots\ldots\; 0 & 0 & \frac{1}{2}d \\
d & d - 1 & d - 1 & d - 2 & d - 2 & \ldots\ldots\ldots\ldots\; \frac{1}{2}d + 1 & \frac{1}{2}d + 1 &
\end{array}$$

and when d is odd, [Bigg79] has,

$$\begin{array}{cccccccc}
 & 1 & 1 & 2 & 2 & \ldots\ldots\ldots \frac{1}{2}(d - 1) & \frac{1}{2}(d - 1) \\
0 & 0 & 0 & 0 & 0 & \ldots\ldots\ldots\ldots\; 0 & \frac{1}{2}(d + 1) \\
d & d - 1 & d - 1 & d - 2 & d - 2 & \ldots\ldots\ldots\ldots\; \frac{1}{2}(d + 1) &
\end{array}$$

The general theory of distance-transitive graphs tells us that the spectrum of O_d is completely determined by the intersection array. In other words, all the eigenvalues of the adjacency matrix, and their multiplicities, may be calculated.

Theorem 3.7 [Bigg79]**:** The eigenvalues of O_d are the integers $\lambda_i = (-1)^i(d - i)$ $(0 \leq i \leq d - 1)$, and the multiplicity of λ_i is

$$m(\lambda_i) = \binom{2d-1}{i} - \binom{2d-1}{i-1}$$

The strong distance-transitivity property implies, in particular, that the automorphism group of O_d is transitive on vertices and on pairs of adjacent vertices.

In the terminology of Biggs [Bigg74], the graph is symmetric**.** For such graphs, Biggs studied the action of the automorphism group on the arcs, as defined below.

An s-arc is a sequence $x_1, x_2, \ldots, x_s$ of vertices such that x_i and x_{i+1}, are adjacent $(0 \leq i \leq s - 1)$ but x_i and x_{i+2} are not identical $(0 \leq i \leq s - 2)$. Since we have a simple representation of the automorphisms of O_d as the permutations of Ω it is easy to verify the following:

Theorem 3.8 [Bigg79]**:** The automorphism group of O_d acts transitively on the set of all 3-arcs, but not on the 4-arcs $(d \geq 3)$.

An s-arc is said to be consistent if there is an automorphism θ such that $\theta(x_0, x_1, \ldots, x_{s-1}) = (x_1, x_2, \ldots, x_s)$ [Bigg79]. It follows from Theorem 3.8 that in O_d $(d \geq 3)$ all arcs of length not exceeding 4 are consistent. If we repeatedly extend a consistent arc, retaining at each stage the consistency property, we must eventually return to the initial vertex.

The resulting circuit is itself symmetric in the sense that there is a graph automorphism which rotates it through one step. A theorem due to Conway [Conw77] states that a symmetric graph of degree d has just d - 1 classes of symmetric circuits. The proof of the theorem provides a recursive method for the construction of the symmetric circuits in general [Bigg79]; here just the result for the graph O_d is described.

Biggs began by remarking that an arc or circuit in O_d is uniquely determined by its initial vertex and the sequence of edge labels. The construction of symmetric circuits proceeds as follows. Let Λ be any subset of Ω having odd cardinality not less than 3, and suppose the members of Λ are ordered so that,

$$\Lambda = \{\lambda_0, \lambda_1, ,\ldots, \lambda_{2r}\}, 1 \leq r \leq d - 1 \text{ [Bigg79]}$$

Let $\{X, Y\}$ be an equipartition of the set Ω - Λ, so that $|X| = |Y| = d - r - 1$. For values of r in the range 1, 2, ..., d - 2 we obtain a symmetric circuit by starting from the initial vertex $v = X \cup \{\lambda_1, \lambda_3, ,\ldots, \lambda_{2r-1}\}$ and proceeding along the edges labeled $\lambda_0, \lambda_1, ,\ldots, \lambda_{2r}, \lambda_0, \lambda_1, ,\ldots, \lambda_{2r}$. This gives a circuit of even length $4r + 2$ (Figure 5). When $r = d - 1$, starting from v and proceeding along the edges labeled $\lambda_0, \lambda_1, ,..., \lambda_{2d-2}$ gives a circuit of odd length $2d - 1$ (in this case the sets X and Y are both empty). The construction provides $d - 1$ classes of symmetric circuits, and, by the theorem quoted above, these are the only symmetric circuits in O_d. The required "rotation" automorphisms are induced by composing the cyclic permutation $\lambda_0, \lambda_1, ,..., \lambda_{2r}$ of Ω with any permutation that takes X to Y.

Theorem 3.9 [Bigg79]**:** The graph O_d ($d \geq 3$) has symmetric circuits of length $2d - 1$ and $6, 10, \ldots, 4d - 6$. The girth of the graph is 5 if $d = 3$, and 6 for all $d \geq 4$.

Biggs remarked that the graphs do contain even circuits of lengths 8, 12, . . . , but these do not have the symmetry property.

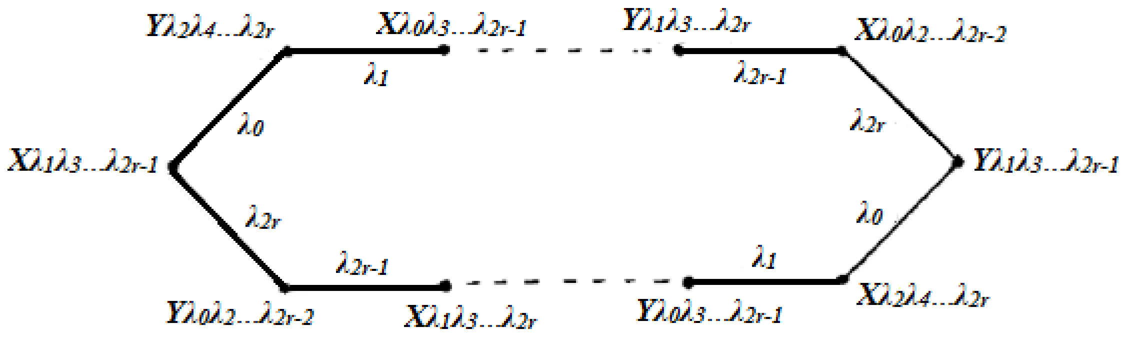

$Y\lambda_2\lambda_4...\lambda_{2r}$
$X\lambda_0\lambda_3...\lambda_{2r-1}$
$Y\lambda_1\lambda_3...\lambda_{2r}$
$X\lambda_0\lambda_2...\lambda_{2r-2}$
λ_1
λ_{2r-1}
λ_0
λ_{2r}
$X\lambda_1\lambda_3...\lambda_{2r-1}$
$Y\lambda_1\lambda_3...\lambda_{2r-1}$
λ_{2r}
λ_0
λ_{2r-1}
λ_1
$Y\lambda_0\lambda_2...\lambda_{2r-2}$
$X\lambda_1\lambda_3...\lambda_{2r}$
$Y\lambda_0\lambda_3...\lambda_{2r-1}$
$X\lambda_2\lambda_4...\lambda_{2r}$

Figure 5: A symmetric circuit of length $4r + 2$ in O_d.

3.5 EDGE PARTITIONS, COLORING AND DOMINATIONS

It was mentioned in Section 3.1 that the edge set of O_d is partitioned into $2d$ - 1 sets E_ω, ($\omega \in \Omega$), where the edges in E_ω are those joining two vertices whose union does not contain ω. This fact, together with the representation (Figure 4), is relevant to the study of the factors and edge colorings of O_d [Bigg79].

If F is a 1-factor of O_d, then it must contain exactly one edge incident with each of the vertices in V_ω. The number of such edges is thus $|V_\omega| = \binom{2d-2}{d-2}$.

The edges of F not incident with V_ω must each carry the label ω, and since $|F| = \frac{1}{2}\binom{2d-1}{d-1}$, the number of them is,

$$\frac{1}{2}\binom{2d-1}{d-1} - \binom{2d-2}{d-2} = \frac{1}{d}\,|E_\omega| \text{ [Bigg79]}$$

In other words, the number of edges carrying the label ω is constant, independent of ω. The same is true for any r-factor, $1 \leq r \leq d$.

Theorem 3.10 [Bigg79]**:** In any r-factor of O_d ($1 \leq r \leq d$), the number of edges carrying a given label ω is independent of ω.

By Vizing's theorem [Vizi64], the number of colors needed to color the edges of the odd graph O_d is either d or $d + 1$, and in the case of the Petersen graph O_3 it is $d + 1$. When d is a power of two, the number of vertices in the graph is odd, from which it again follows that the number of edge colors is $d + 1$. However, O_5, O_6, and O_7 can each be edge-colored with d colors.

An edge-dominating set in a graph G is a subset D_E of the edge set $E(G)$ of G with the property that to each edge $e \in E(G) - D_E$ there exists an edge $f \in D_E$ such that the edges e, f have a common end vertex. The minimal number of vertices of an edge-dominating set in G is called the edge-domination number of G.

Analogously to the domatic number of a graph [Cock77] we may define the edge domatic number of a graph G.

An edge-domatic partition of a graph G is a partition of the edge set $E(G)$ of G, all of whose classes are edge-dominating sets in G. The maximal number of classes of an edge-domatic partition of G is called the edge-domatic number of G.

Theorem 3.11 [Zeli85]: The edge-domination number of the graph O_d is equal to $\frac{1}{2}\binom{2d-2}{d-1}$ and its edge-domatic number is equal to $2d - 1$.

Proof:

Let $\omega \in \Omega$ and let E_ω be the set of all edges e of O_d labeled with ω. Let f be an edge of O_d not belonging to E_ω, and labeled with τ. Then, $\tau \neq \omega$. Let u, v be the end vertices of f. Exactly one of the sets u, v contains the label ω; without loss of generality let it be u. Let $w = \Omega - (v \cup \{ \omega \})$; then v and w are joined by an edge belonging to E_ω. As f was chosen arbitrarily, it has been proved that E_ω is an edge-dominating set (for an arbitrary ω).

Now let us look for the cardinality of E_ω. If x is an arbitrary vertex of $\Omega - \{ \omega \}$ of the cardinality $d - 1$ and $y = \Omega - (x \cup \{ \omega \})$, then the vertices x, y are joined by an edge belonging to E_ω and vice versa. The number of subsets of $\Omega – \{ \omega \}$ of the cardinality d-1 is equal to$\binom{2d-2}{d-1}$. Having in mind that for a subset x of $\Omega – \{ \omega \}$ of the cardinality $d - 1$ the set $y = \Omega - (x \cup \{\omega\})$ is also a subset of $\Omega – \{ \omega \}$ of the cardinality $d - 1$, we find that the number of unordered pairs $\{x, y\}$ of described sets is $\frac{1}{2}\binom{2d-2}{d-1}$ and this is also the cardinality of E_ω. This number does not depend on ω, thus all the sets E_ω for ω = 1, ..., $2d – 1$ have equal cardinalities. The edge-domination number of O_d is thus at most $\frac{1}{2}\binom{2d-2}{d-1}$ and its edge-domatic number is at least $2d$ - 1.

The edge-domatic number of a graph is evidently equal to the domatic number [Cock77] of its line-graph. The degree of each vertex of the line-graph of O_d is $2d$ - 2 and this implies [Cock77] that its domatic number (and thus the edge-domatic number of O_d) is at most $2d - 1$. It has been proved that the edge-domatic number of O_d is $2d - 1$.

Now suppose that there exists an edge-dominating set D_E of a cardinality $c < \frac{1}{2}\binom{2d-2}{d-1}$. For each edge $e \in D_E$ the set consisting of e and all edges having a common end vertex with e has the cardinality $2d - 1$. As each edge of O_d either is in D_E, or has an end vertex in common with an edge of D_E, the number of edges of O_d is at most $c(2d - 1) < \frac{1}{2}(2d - 1)\binom{2d-2}{d-1}$ $= \frac{1}{2}\, d\binom{2d-1}{d-1}$. But the number at the right-hand side of this inequality is the number of edges of O_d. (The number of vertices is $\binom{2d-1}{d-1}$ and the graph is regular of the degree d.) As $c(2d - 1)$ is less, we have a contradiction. Thus each E_ω is an edge dominating set of the least cardinality and the edge-domination number of O_d is $\frac{1}{2}\binom{2d-2}{d-1}$. □

3.6 GRAPH DECOMPOSITIONS

Theorem 3.12 [Zeli85]: Let T_d be a tree with the vertex set $\{a, b, c_1, \dots, c_{d-1}, k_1, \dots, k_{d-1}\}$ and with the edges ab, ac_i, bk_i for $i = 1, \dots, d-1$. Then the graph O_d can be decomposed into $\frac{1}{2}\binom{2d-2}{d-1}$ pairwise edge-disjoint subgraphs which are all isomorphic to T_d. Moreover, each of these subgraphs contains exactly one edge from each set E_ω for $\omega = 1, \dots, 2d - 1$.

Proof:

Let $\omega \in \Omega$, let E_ω have the same meaning as in the proof of Theorem 3.11. Let e_1, e_2 be two elements of E_ω. Suppose that these edges have a common end vertex u. Let v_1 (or v_2) be the end vertex of e_1 (or e_2 respectively) distinct from u. Then $\Omega - (u \cup v_1) = \Omega - (u \cup v_2) = \{ \omega \}$ and $u \cap v_1 = u \cap v_2 = \emptyset$. This implies $v_1 = v_2$ and also $e_1 = e_2$, because O_d is a graph without multiple edges. We have proved that there exist no two distinct edges of E_ω which would have an end vertex in common. Now suppose that to the edges e_1, e_2 of E_ω there exists an edge f which has common end vertices with both e_1, e_2.

Let u_1 (or u_2) be the common end vertex of e_1 (or e_2 respectively) and f. Let v_1 (or v_2) be the end vertex of e_1 (or e_2 respectively) distinct from u_1 and u_2. Then $\Omega - (u_1 \cup v_1) = \Omega - (u_2 \cup v_2) = \{ \omega \}$, $u_1 \cap \mathrm{v}_1 = u_2 \cap v_2 = u_1 \cap u_2 = \emptyset$. This implies that none of the vertices u_1, u_2, v_1, v_2 contains ω. As $u_1 \cap u_2 = \emptyset$, we have $\Omega - (u_1 \cup u_2) = \{ \omega \}$ and $f \in E_\omega$. According to the above proved this is possible only if $e_1 = e_2 = f$. Therefore if the labels of e_1 and e_2 are equal and $e_1 \neq e_2$, then the distance between an arbitrary end vertex of e_1 and an arbitrary vertex of e_2 is at least 2.

Now let e be an edge of O_d. Let $G[e]$ be the subgraph of O_d consisting of the edge e, all edges having a common end vertex with e and of end vertices of all of these edges. This is a tree isomorphic to T_d. If e_1, e_2 are two distinct edges of $G[e]$, then either they have a common end vertex, or there exists an edge of $G[e]$ which has common end vertices with both of them. According to the above proved the labellings of edges of $G[e]$ are pairwise different.

Let $T(\omega)$ be the set of subtrees $G[e]$ for all edges $e \in E_\omega$. Any two distinct trees from $T(\omega)$ are edge-disjoint; otherwise there would exist two distinct edges of E_ω with a common end vertex or with the property that there exists an edge having common vertices with both of them. The cardinality of $T(\omega)$ is equal to that of E_ω, namely $\frac{1}{2}\binom{2d-2}{d-1}$. Each tree from $T(\omega)$ has $2d$ - 1 edges. Hence the union of all trees from $T(\omega)$ has $\frac{1}{2}\binom{2d-2}{d-1}(2d-1) = \frac{1}{2}\, d\binom{2d-1}{d-1}$ edges and this is the number of edges of O_d. It has been proved that $T(\omega)$ is the required decomposition. □

To contract an edge of a graph means to delete this edge and to identify its end vertices.

Theorem 3.13 [Zeli85]: The graph $O_d'(\omega)$ obtained from O_d by contracting every edge e labeled with ω, where ω is an integer between 1 and $2d$ - 1, is a bipartite graph.

Proof:

By the described contractions each tree from $T(\omega)$ is transformed into a star. Hence $O_d'(\omega)$ is a graph which is the union of edge-disjoint stars with the property that each of them contains all edges incident with its centre in $O_d'(\omega)$. Every graph with this property is bipartite. □

Let $P(n)$ be the set of all linear orderings of the set $\{1, ..., n\}$. Let π_1, π_2 be elements of $P(n)$. We say that π_1, π_2 are dihedrally equivalent, if either $\pi_1 = \pi_2$, or π_2 can be obtained from π_1 by acyclic permutation, by reversing or by a super-position of a cyclic permutation and a reversing. The relation thus defined is evidently an equivalence on the set $P(n)$.

Let C be a circuit of the length n whose edges are labelled by pairwise different numbers from the set $\{1, ...,n\}$. If we run around C and write the labels of the traversed edges, we may obtain different linear orderings of the set $\{1, ...,n\}$ according to in which vertex we have started and in which sense we have gone. These orderings form one class of the dihedral equivalence. We may say that to C a class of the dihedral equivalence on $P(n)$ corresponds.

The number of classes of the dihedral equivalence on $P(n)$ is evidently equal to $\frac{1}{2}(n-1)!$.□

Theorem 3.14 [Zeli85]: The graph O_d with the labelling λ is the union of $\frac{1}{2}(2d-2)!$ circuits of the length $2d-1$ which correspond to pairwise different classes of the dihedral equivalence on $P(2d-1)$. Each edge of O_d belongs to $(d-1)!^2$ and each vertex to $\frac{1}{2}d!(d-1)!$ such circuits.

Proof:

Let C be a class of the dihedral equivalence on $P(2d - 1)$. Let $\pi \in C$ and $[a_1, ..., a_{2d-1}] = \pi$. Let $U_1 = \{a_i \mid i \text{ even}, 2 \leq i \leq 2d - 2\}$. We construct the sets $U_2, ..., U_{2d-1}$ recursively. If U_i is constructed for some i, then $U_{i+1} = \Omega - (U_i \cup \{i\})$. Any two vertices U_i, U_{i+1} are adjacent in O_d. Further it may be easily proved that $\Omega - (U_{2d-1} \cup \{2d-1\}) = U_1$ and the vertices U_{2d-1}, U_1 are adjacent, too. We have obtained a circuit in O_d; evidently this circuit corresponds to C. We may construct such a circuit for each class of the dihedral equivalence on $P(2d - 1)$. From the construction it is evident that circuits corresponding to the same class are identical and that each edge of O_d is contained in some of these circuits. The family of the mentioned circuits will be denoted by $\mathfrak{C}$.

The graph O_d is evidently vertex-transitive and edge-transitive. (A graph is vertex-transitive, if to any two of its vertices there exists its automorphism which maps one vertex onto the other. Analogously the edge-transitivity is defined.) This implies that for any two vertices V_1, V_2 of O_d the number of circuits of $\mathfrak{C}$ containing V_1 is equal to the number of those containing V_2 and an analogous assertion holds for edges, too. Thus the number of circuits from $\mathfrak{C}$ containing any vertex is obtained by dividing the sum of lengths of all circuits of $\mathfrak{C}$, namely $\frac{1}{2}(2d-2)!(2d - 1)$, by the number of vertices of O_d, namely $\binom{2d-1}{d-1}$; the result is $\frac{1}{2}d!(d - 1)!$. If we divide the number $\frac{1}{2}$ (2d-2)!(2d - 1) by the number of edges of O_d, namely $\frac{1}{2}\ d\ \binom{2d-1}{d-1}$, we obtain the number of circuits of $\mathfrak{C}$ containing any edge, namely (d - 1)!^{2}. □

3.7 HAMILTONIAN CIRCUITS AND PATHS

It is well known that O_3 is not Hamiltonian and that it does not have an edge 3-coloring (three disjoint 1-factors). At one time [Bigg72] it was thought that such anomalies might persist throughout the whole family but that is now known to be false:

Theorem 3.15 [Bigg79]**:** When d = 4, 5, 6, 7, O_d contains [$d/2$] edge-disjoint Hamiltonian circuits [Mere72][Mere73].

It is tempting to conjecture that Theorem 3.15 is true for all $d \geq 4$**.** However, in general, the construction of even a single Hamiltonian circuit in O_d seems to be rather difficult, one advance on Theorem 3.15 is the construction of one Hamiltonian circuit in O_8 [Math76]. In addition, Shields and Savage [Shie04] used a carefully designed heuristic to find Hamiltonian circuits in O_d for $4 \leq d \leq 14$.

Lov´asz [Lov´a70] conjectured that every connected vertex-transitive graph has a Hamiltonian path. An attempt to provide more evidence to support Lov´asz conjecture is to compute the Hamiltonian paths for $d \geq 2$.

However, a direct computation of Hamiltonian paths in O_d is not feasible for large values of d. The graph O_2 (a triangle) and O_3 (the Petersen graph), both of which have Hamiltonian paths (Figure 6). Balaban [Bala72] showed that O_4 and O_5 have Hamiltonian paths. Meredith and Lloyd [Mere72] showed that O_6 and O_7 have Hamiltonian paths. Mather [Math76] showed that O_8 has a Hamiltonian path. Shields and Savage [Shie99] used a carefully designed heuristic to find Hamiltonian paths in O_d for $d \leq 14$. Bueno and Faria showed that O_d has a Hamiltonian path for $15 \leq d \leq 18$ [Buen09]. Instead of directly running any heuristics, they used existing results on the middle levels problem [Shie99][Shie09].

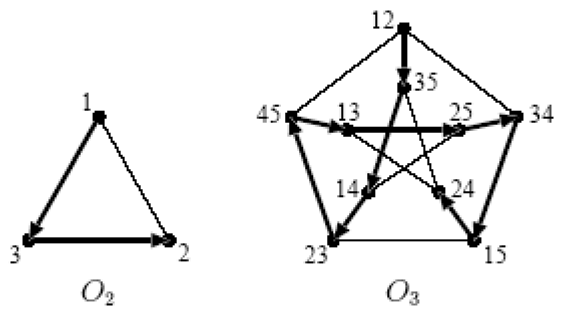


Figure 6: The O_2 and O_3, with highlighted Hamiltonian paths [Buen09].

3.8 CODE

In Section 3.2 we noted that a maximal independent set M in any d-regular graph must satisfy the inequality $(d + 1) \mid M \mid \geq \mid V \mid$. Equality holds if the collection of "disks" $D(m)$ $(m \in M)$ covers the vertex-set V exactly. In order to connect with later terminology, [Bigg79] used the term perfect 1-code to denote a maximal independent set with the minimum cardinality $\frac{1}{d+1} \mid V \mid$.

When is there a perfect 1-code in O_d? The obvious necessary condition that $d + 1$ should be a divisor of $\binom{2d-1}{d-1}$ is by no means sufficient. If x and y are distinct vertices of a perfect 1-code M in O_d, then $D(x)$ and $D(y)$ do not overlap, and we have $\partial(x, y) \geq 3$. It follows from Theorem 3.3 that $\mid x \cap y \mid < d - 2$; hence any $(d - 2)$ subset of Ω is contained in at most one vertex belonging to M. But each vertex contains $d - 1$ such subsets, and the total number occurring is,

$$(d-1)|M| = \frac{d-1}{d+1}\binom{2d-1}{d-1} = \binom{2d-1}{d-2} \text{ [Bigg79]}$$

Thus, every $(d - 2)$-subset of Ω occurs exactly once as subset of a vertex belonging to M. It has been shown that M must be a Steiner system $S(d - 2, d - 1, 2d - 1)$; that is, a collection of $(d - 1)$ subsets usually called blocks of a $(2d - 1)$ set with the property that each $(d – 2)$ subset occurs just once in a block.

If a perfect 1-code in O_d, or $S = S(d-2, d-1, 2d-1)$, does exist, then it induces a unique extended system $S^+ = S(d-1, d, 2d)$. The extension is constructed by adding one new label ∞ to Ω and taking the new blocks to be of two kinds: (i) the blocks of S with ∞ added, and (ii) the complements in Ω of the blocks of S.

Conversely, if a system with the parameters of S^+ is given, then S may be obtained by deleting one label and taking complements of the blocks not containing it. Assmus and Hermoso [Assm74] have shown that if S^+ has a flag-transitive group of automorphisms, then $d = 4$ or $d = 6$. Hence, if a perfect 1-code in O_d exists when $d \neq 4, 6$, its construction is certain to be very complicated.

Theorem 3.16 [Bigg79]: If there is a perfect 1-code in O_d with flag-transitive extension, then $d = 4$ or 6.

The systems do exist in the cases $d = 4$ and $d = 6$; a representation of the perfect 1-code in O_4 is shown in Figure 7.

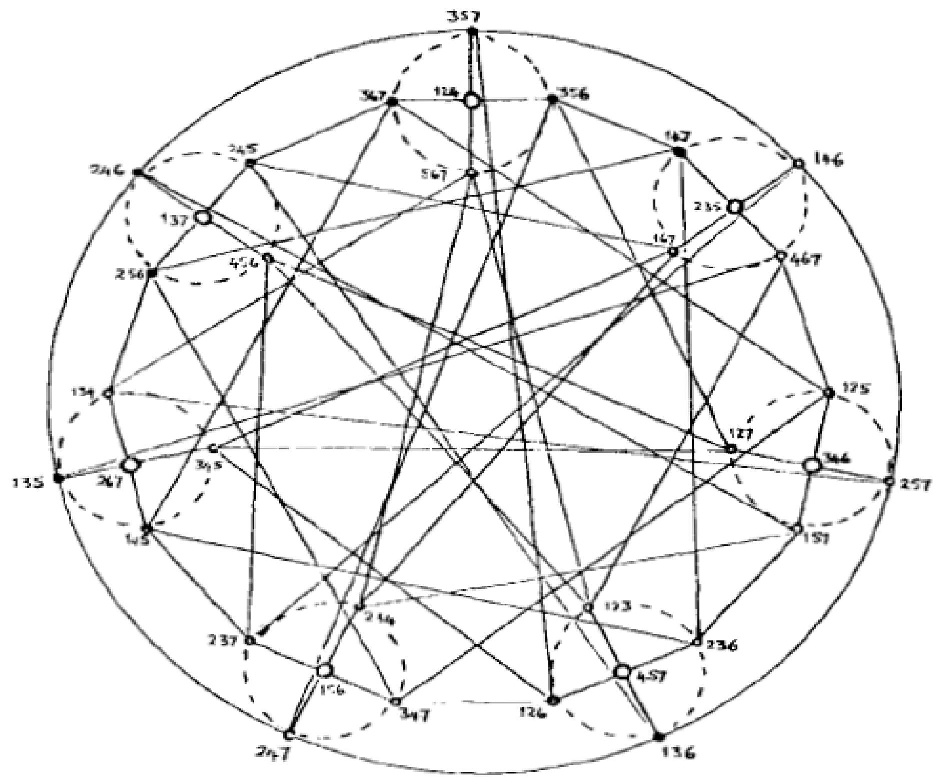


Figure 7: A perfect 1-code in O_4 [Bigg79].

Since it seems that the simple lower bound for the size of a maximal independent set in O_d is rarely attained, the difficult question of finding the actual minimum arises [Bigg79].

CHAPTER 4

APPROXIMATION ALGORITHMS FOR INDEPENDENT DOMINATION IN ODD GRAPHS

4.1 INTRODUCTION

In sections 3.2 and 3.8 we saw that the lower bound of the minimum maximal independent set, which is $|V|/(d+1)$, is rarely attained and the actual cardinality of it is an open problem. In this chapter we present approximation algorithms that find an approximate minimum independent dominating set by partitioning the vertices of the odd graphs to simplify the complex structure of the graph. In section 4.2, a partitioning scheme will be presented with some observations. In section 4.3, the approximation algorithms are described and the correctness of the algorithm along with analysis is given. In addition, an example of finding the MIDS in O_4 is given to illustrate the algorithm. Generic greedy and randomized algorithms are given in section 4.4. Finally, in section 4.5 we will compare the performance of our algorithms with the generic algorithms empirically.

4.2 VERTICES PARTITION

Let $\Omega = \{1, 2, \ldots, 2d - 1\}$, $p = 2d - 2$ and $q = 2d - 1$, so $\Omega_d = \Omega_{d-1} \cup \{p, q\}$. If $s \subseteq \Omega_{d-1}$, then $\bar{s} = \Omega_{d-1} \setminus s$, that is, the complement of s with respect to $\Omega_{d-1} = \{1, 2, \ldots, 2d - 3\}$.

The set of vertices of O_d is divided into four categories according to whether their labels contain either p or q, both p and q or neither. So, V is partitioned into four sets: A, B, C and D such that A consists of all subsets with both p and q, B consists of all subsets with neither p nor q, C consists of all subsets with p but not q, and D consists of all subsets with q but not p. Thus, for $d \geq 3$, $A = \{\{x_1, x_2, \ldots, x_{d-3}, p, q\} \mid x_i \in \Omega_{d-1}\}$, $B = \{\{x_1, x_2, \ldots, x_{d-1}\} \mid x_i \in \Omega_{d-1}\}$, $C = \{\{x_1, x_2, \ldots, x_{d-2}, p\} \mid x_i \in \Omega_{d-1}\}$, and $D = \{\{x_1, x_2, \ldots, x_{d-2}, q\} \mid x_i \in \Omega_{d-1}\}$. Since V_{d-1} is the set of all $(d - 2)$-subsets of Ω_{d-1}, the sets B, C and D can be rewritten as

$$B = \{\bar{s} \mid s \in V_{d-1}\}, C = \{s \cup p \mid s \in V_{d-1}\}, D = \{s \cup q \mid s \in V_{d-1}\}.$$

The cardinalities of these sets are given by: $|A| = \binom{2d-3}{d-3}$, $|B| = |C| = |D| = \binom{2d-3}{d-2} = |V_{d-1}|$.

The partitioning of the set of vertices of O_d induces a partitioning of its edges as defined by

$E_{ab} = \{(a, b) \mid a \in A, b \in B \text{ and } a \cap b = \emptyset\}$,

$E_{bc} = \{(b, c) \mid b \in B, c \in C \text{ and } b \cap c = \emptyset\} = \{(s \cup p, \bar{s}) \mid s \in V_{d-1}\}$,

$E_{bd} = \{(b, d) \mid b \in B, d \in D \text{ and } b \cap d = \emptyset\} = \{(s \cup q, \bar{s}) \mid s \in V_{d-1}\}$,

$E_{cd} = \{(c, d) \mid c \in C, d \in D \text{ and } c \cap d = \emptyset\} = \bigcup_{(s,t) \in O_{d-1}} \{(sp, tq), (tp, sq)\}$.

The last equality follows from the fact that an edge (s, t) in O_{d-1} gives rise to two edges linking two vertices in C with two vertices in D, namely (sp, tq) and (tp, sq).

Figure 8 shows the new drawings of the odd graph O_d, d = 2, 3, 4. In Figure 8 (b) for the Peterson graph, $A = \{\{4, 5\}\}$, $B = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$, $C = \{\{1, 4\}, \{2, 4\}, \{3, 4\}\}$ and $D = \{\{1, 5\}, \{2, 5\}, \{3, 5\}\}$ (here, $p = 4$; $q = 5$).

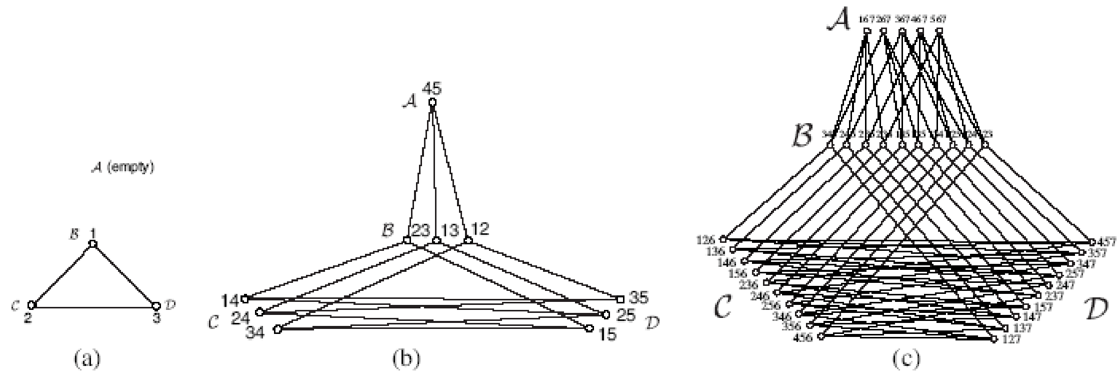


Figure 8: New drawings of the odd graph O_d, d = 2, 3, 4.

Proposition 4.1: Given the partition $\{A, B, C, D\}$ of the vertices of the odd graph O_d, $d \geq 3$, we have

(i) Each vertex in A is connected to d vertices in B.

(ii) Each vertex in B is connected to d - 2 vertices in A, one vertex in C and one vertex in D.

(iii) Each vertex in C is connected to 1 vertex in B and d - 1 vertices in D.

(iv) Each vertex in D is connected to 1 vertex in B and d - 1 vertices in C.

Proof:

(i) Observe that all subsets in $C \cup D$ contain p or q while subsets in B contain neither.

Hence, all subsets disjoint from those in A are contained in B. So, let $a = \{x_1, x_2, \ldots, x_{d-3}, p, q\} \in A$ and $B_a = \{\{y_1, y_2, \ldots, y_{d-1}\} \mid y_i \in \overline{\{x_1, x_2, \ldots, x_{d-3}\}}\}$ be the subsets in B disjoint from a.

Then, $|B_a|$ is the number of ways to choose d-1 numbers from $\Omega_{d-1} \backslash \{x_1, x_2, \ldots, x_{d-3}\}$. Hence, the number of subsets in B disjoint from a is $|B_a| = \binom{2d-3-(d-3)}{d-1} = \binom{d}{d-1} = d$.

(ii) Let $b \in B$. Since all subsets in A contain both p and q, and those in B contain neither, the number of subsets in A disjoint from b is equal to the number of ways to choose d - 3 numbers from $\Omega_{d-1} \setminus b$. Hence, b is connected to exactly $\binom{2d-3-(d-1)}{d-3} = \binom{d-2}{d-3} = d - 2$ subsets in A of the form $\{x_1, x_2, \ldots, x_{d-3}, p, q\}$, where $x_i \in \overline{b}$, $1 \leq i \leq d - 3$. Since all subsets in C contain p but not q, and those in B contain neither, the number of subsets in C disjoint from b is equal to the number of ways to choose d - 2 numbers from $\Omega_{d-1} \setminus b$. Hence, the number of subsets in C disjoint from b is $\binom{2d-3-(d-1)}{d-2} = \binom{d-2}{d-2} = 1$. That is, b is disjoint from exactly one subset in C, namely $\overline{b} \cup p$. Similarly, b is disjoint from exactly one subset in D, namely $\overline{b} \cup q$.

(iii) Let $c \in C$ and $c' = c \setminus \{p\}$. Then, the number of subsets in B disjoint from c is equal to the number of ways to choose d - 1 numbers from $\Omega_{d-1} \setminus c'$. Hence, the number of subsets in B disjoint from c is $\binom{2d-3-(d-2)}{d-1} = \binom{d-1}{d-1} = 1$. That is, c is disjoint from exactly one subset in B, namely $\overline{c'}$. By definition, C consists of all (d - 2)-subsets of Ω_{d-1} suffixed by p, and D consists of all (d - 2)-subsets of Ω_{d-1} suffixed by q. Note that c' is a (label of a) vertex in V_{d-1}. Hence, by definition of O_{d-1}, c' is disjoint from d - 1 vertices $x_1, x_2, \ldots, x_{d-1}$ in O_{d-1}. Then, for $i \in \{1, 2, \ldots, d - 1\}$, c is disjoint from $x_i \cup q$ in O_d. Consequently, c is disjoint from exactly d - 1 subsets in D.

(iv) Similar to (iii). □

By Proposition 4.1, the following bipartite graphs are present in O_d: $O_d[E_{ab}]$ is a bipartite graph in which $\forall\, a \in A$ deg(a) = d and $\forall\, b \in B$ deg(b) = d - 2, $O_d[E_{cd}]$ is a (d - 1)-regular bipartite graph, and $O_d[E_{bc}]$ and $O_d[E_{bd}]$ are 1-regular bipartite graphs. Moreover, $\{O_d[E_{ab}]$, $O_d[E_{cd}]$, $O_d[E_{bc}]$, $O_d[E_{bd}]$ $\}$ is a decomposition of the odd graph O_d into four bipartite graphs.

Lemma 4.1: The odd graph O_d, $d \geq 2$, contains $|V_{d\text{-}1}|$ vertex-disjoint paths of length 2.

Proof:

See Figure 9. Let $x \in V_{d\text{-}1}$. Then $c = x \cup p \in C$, $b = \bar{x} \in B$, and $d = x \cup q \in D$. Since (c, b) and (b, d) are edges in O_d, $\pi = c, b, d$ is a path of length 2 in O_d. Obviously, if $\pi' = c', b', d'$ with $c' \neq c$, then π and π' are vertex-disjoint. It follows that the number of such paths is $|C| = |V_{d\text{-}1}|$.□

As an illustration of Lemma 4.1, the following three paths are present in the Peterson graph shown in Figure 8 (b): π_1 = 14, 23, 15, π_2 = 24, 13, 25 and π_3 = 34, 12, 35.

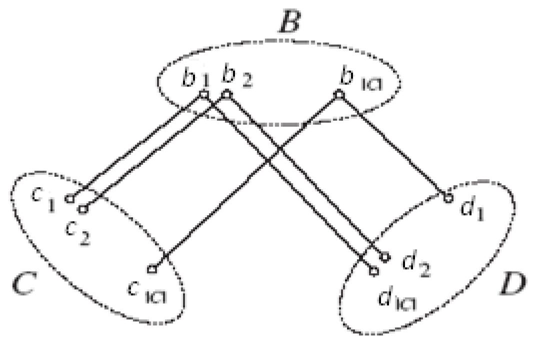


Figure 9: O_d contains $|V_{d-1}|$ vertex-disjoint paths.

Theorem 4.2: $B \cup C$ is a vertex cover for O_d of size $2|V_{d-1}|$, which is minimum.

Proof:

Recall that $O_d[E_{cd}]$ is the $(d - 1)$-regular bipartite subgraph induced by the vertex set $C \cup D$. By Hall and Konig classical arguments, $O_d[E_{cd}]$ has a perfect matching whose cardinality is equal to a minimum vertex cover C for $O_d[E_{cd}]$. Thus, $|C| = |V_{d-1}|$. By Lemma 5.1, O_d, $d \geq 2$, contains $|V_{d-1}|$ vertex-disjoint paths $\pi_i = u_i, v_i, w_i$, where $u_i \in C$, $v_i \in B$, $w_i \in D$, $1 \leq i \leq |V_{d-1}|$. So, C forms a subset of the end-vertices of these paths. Since these paths are vertex-disjoint, they contain exactly $|V_{d-1}|$ edges that are covered by C. Consequently, $|V_{d-1}|$ additional vertices are required to cover the remaining $|V_{d-1}|$ edges, and hence all paths. It follows that the cardinality of any vertex cover for O_d is at least $2|V_{d-1}|$.

On the other hand, since any edge in O_d has one of its ends in either B or C, it immediately follows that $B \cup C$ is a vertex cover for O_d of cardinality $|B \cup C| = 2|C| = 2|V_{d\text{-}1}|$.□

Corollary 4.1: $A \cup D$ is a maximum independent set for O_d of size V_d - $2|V_{d\text{-}1}|$.

4.3 APPROXIMATION ALGORITHMS

It is clear from section 4.2 that the set of vertices in set B is a maximal independent set. The approximation algorithms that we propose reduce the set of the maximal independent set while maintaining the independence and the domination properties. Next, a detailed description of the algorithms is given.

4.3.1 ALGORITHMS DESCRIPTION

The algorithms can be divided into 4 stages. In stage 1, the algorithms perform vertices partitioning and initialize set IDS_B with set B. In stage 2, they find an independent set in $\mathcal{E}_{CD}$ ($O_d[E_{cd}]$ which is a (d - 1)-regular bipartite graph). In stage 3, they remove vertices from set IDS_B and find the fixed vertices from set IDS_B and set A. In stage 4, the algorithms find an independent dominating set in $\mathcal{E}_{BA}$ ($O_d[E_{ab}]$ which is a bipartite graph). Next, we will discuss each stage in detail.

Stage 1:

First, the algorithm partition O_d to the sets A, B, C and D as explained in section 5.2. At the initialization stage the algorithm initializes set IDS$_B$ with set B.

Stage 2:

Next, the algorithm finds the independent set of vertices (IS$_{CD}$) in the induced bipartite graph $\mathcal{E}_{CD}$. We design two algorithms for obtaining IS$_{CD}$, The first one finds IS$_{CD}$ with the following restriction, all shortest distance between any two vertices in set C or D are greater than two, and on the other hand, the second method relaxes the restriction and allows two vertices with shortest distance of length 2 in IS$_{CD}$.

The first algorithm for finding IS$_{CD}$: The algorithm finds a set of vertices (IS$_C$) from set C with the condition that the shortest distance between any pair of vertices is greater than two. The algorithm starts by choosing a vertex from set C, let it be c_1 then add it to IS$_C$. Then, the algorithm finds a vertex from set C, say c_n such that the shortest distance between c_n and any vertex in IS$_C$ is equal to the diameter, then it adds c_n to IS$_C$, repeat this step until there is no more vertices satisfy the condition. At this stage, the distance between all vertices in IS$_C$ is equal to the diameter. The algorithm then finds every vertex from set C such that the distance between a vertex and all vertices in IS$_C$ is greater than two, add the vertices to IS_C. The algorithm repeats the previous steps for each vertex in C as the starting vertex, and lastly, it selects IS_C with maximum cardinality. The algorithm deletes all adjacent vertices to the vertices in IS_C. The algorithm repeats all previous steps to find IS_D, and finally takes the union of IS_C and IS_D as IS_{CD}. Empirically, it was found that this method works for $d \leq 7$.

The second algorithm for finding IS_{CD}: The algorithm chooses the first vertex from C, for example the order of vertices in O_3 is {14,24,34} and the first vertex is {14}, if the vertex is adjacent to d-1 vertices then adds it to IS_{CD} and delete it with the adjacent vertices. Then, it chooses the first vertex from D, if it is adjacent to d-1 vertices then adds it to IS_{CD} and delete it with adjacent vertices. The algorithm repeats in order for all vertices in $\mathcal{E}_{CD}$, until there is no vertex that is adjacent to d-1 vertices. The algorithm repeats the previous steps and finds vertices that have d-2 neighbors, and so on until the cardinality of the adjacent vertices of all vertices is equal to 2.

We will refer to Algorithm 1 as Approx. 1 if it uses Algorithm 2 to find IS_{CD} and Approx. 2 if it uses Algorithm 3.

Stage 3:

After the algorithm finds the IS_{CD} from the induced bipartite $\mathcal{E}_{CD}$, it removes any vertex from set IDS_B if it is adjacent to vertices or a vertex in

IS$_{CD}$. Let fixed$B$ be a set of vertices in IDS$_B$, such that a vertex $v \in$ fixed$B$ is adjacent to $w \in C$ and $u \in D$, where w is not dominated by any vertex from set D and vertex u is not dominated by any vertex from set C. Vertices from fixed$B$ cannot be removed from the IDS$_B$, since removing such vertices will violate the domination property because the adjacent vertices will be undominated. Consequently, any vertex from set A that is adjacent to a vertex from fixed$B$ will be added to a set fixed$A$.

Stage 4:

Let set $vA = A \setminus$ fixed$A$. The algorithm chooses a vertex from set vA if the vertex is adjacent to d vertices from set IDS$_B$ and removing the d vertices from the IDS$_B$ does not violate the domination property, we add the chosen vertex to the IDS$_A$ and delete the adjacent vertices from the IDS$_B$. If removing the d adjacent vertices causes a violation in the domination property, the algorithm select another vertex from vA and repeats the previous steps. The algorithm repeats the previous steps for all vertices in vA. The algorithm repeats all previous steps for checking cardinalities of adjacent vertices from d-1 to 2. The algorithm will exit the loop if all vertices in vA cause a domination violation. At the end of this stage the algorithm sets the IDS$_{BA}$ to the union of IDS$_B$ and IDS$_A$. Finally, the algorithm finds the final IDS (the independent dominating set for O_d) by taking the union of IS$_{CD}$ and IDS$_{BA}$.

Algorithm 1 Approximation Algorithm for Independent Dominating Set

```
Input: $O_d$
Output: Independent Dominating Set (IDS)
1: partition $O_d$ into sets $A$, $B$, $C$, and $D$ as defined above, $IDS_B \leftarrow B$
2: $IS_{CD} \leftarrow$ FindIS$_{CD}$ ($C$, $D$)
3: $IDS_B \leftarrow IDS_B \backslash \forall v$ such that $N(v) \in IS_{CD}$
4: Find fixed vertices from set $IDS_B$ and $A$
5: $IDS_A \leftarrow \emptyset$, $vA \leftarrow A \setminus$ fixed$A$
6: for i $\leftarrow d$ to 2 do Step 7, 8 and 9
7: for $\forall v \in vA$ do Steps 8 and 9
8: choose $v$ such that $|N(v)| = i$ and removing $N(v)$ does not cause a violation
9: $IDS_B \leftarrow IDS_B \setminus N(v)$, $IDS_A \leftarrow IDS_A \cup \{v\}$
10: $IDS_{BA} \leftarrow IDS_B \cup IDS_A$
11: $IDS \leftarrow IDS_{BA} \cup IS_{CD}$
```

Figure 10: Algorithm 1 for Independent Dominating Set.

Algorithm 2 Algorithm for IS$_{CD}$ such that $\partial(u, v) > 2$ for $\forall$ $u, v \in$ IS$_{CD}$

```
Input: $C$, $D$
Output: $IS_{CD}$
1: $IS_{CD} \leftarrow \emptyset$, $IS_C \leftarrow \emptyset$, $IS_D \leftarrow \emptyset$
2: for each $v \in C$ do Steps 3, 4, 5
3: add $v$ to $IS_C$
4: starting from $v$ add every vertex from $C$ to $IS_C$ such that the distance between
any pair of vertices in $IS_C$ is $d$-1.
5: find all vertices in $C$ with dis. > 2 to all vertices in $IS_C$ and add them to $IS_C$
6: select $IS_C$ with maximum cardinality
7: delete all adjacent vertices to $IS_C$ from $D$
8: repeat Steps 2, 3, 4, 5 and 6 for all vertices in $D$
9: $IS_{CD} \leftarrow IS_C \cup IS_D$
```

Figure 11: Algorithm 2 the first algorithm for finding IS$_{CD}$.

Algorithm 3 Algorithm for IS_{CD} such that $\exists$ *u, v* $\in IS_{CD}$ with $\partial(u, v) = 2$

```
Input: $C, D$
Output: $IS_{CD}$
1: $IS_{CD} \leftarrow \emptyset$
2: for i ← $d$-1 to 2 do Step 3, 4, and 5
3: for each vertex $v \in C$ starting from the first vertex and $u \in D$ starting from the
first vertex do Steps 4 and 5
4: if $|N(v)| = i$ then add it to $IS_{CD}$ and delete $N[v]$
5: if $|N(u)| = i$ then add it to $IS_{CD}$ and delete $N[u]$
```

Figure 12: Algorithm 3 the second algorithm for finding IS_{CD}.

Example: We will demonstrate the described algorithms for finding the minimum independent dominating set in O_4.

First stage:

Referring to step 1 in Algorithm 1, the algorithm partition the graph to the following sets,

A={167, 267, 367, 467, 567}, B={123, 124, 125, 134, 135, 145, 234, 235, 245, 345}, C={126, 136, 146, 156, 236, 246, 256, 346, 356, 456}, D={127, 137, 147, 157, 237, 247, 257, 347, 357, 457}.

IDS_B is initialized with B, and the result is the following set,

IDS_B = {123, 124, 125, 134, 135, 145, 234, 235, 245, 345}. The result graph can be seen in Figure 13 (the dominating vertices are colored with black).

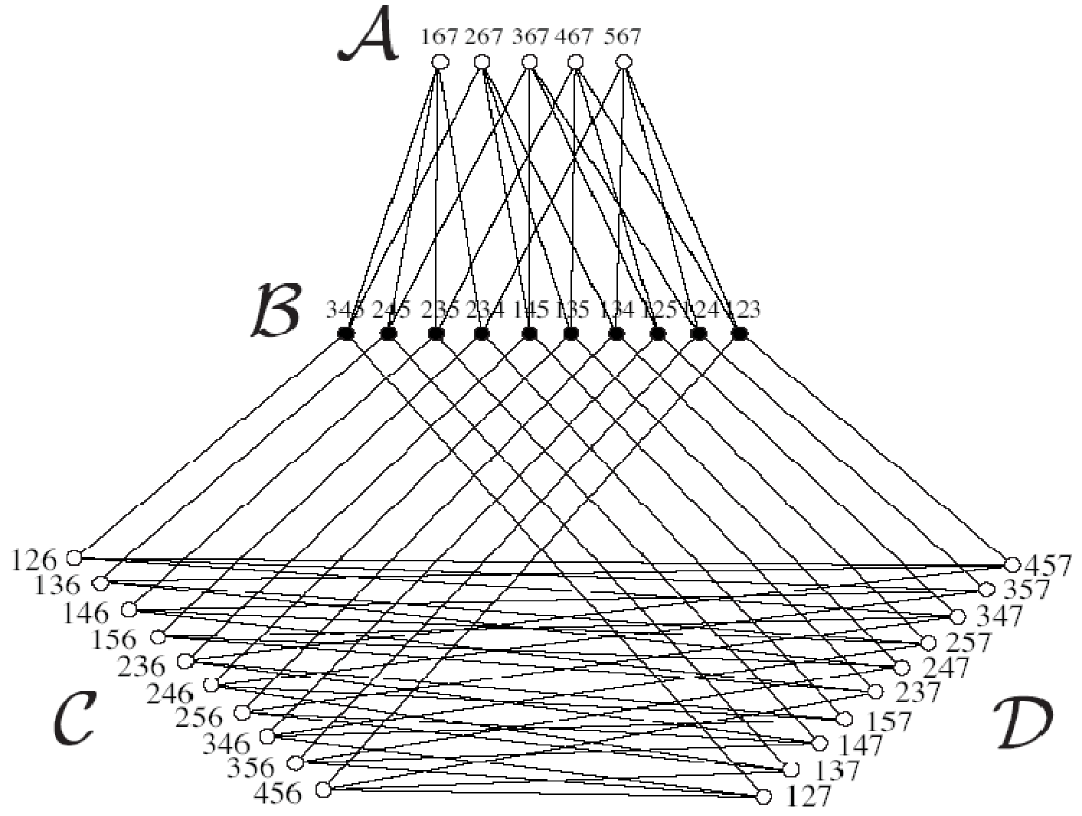


Figure 13: O_4 after the first stage.

Second stage:

Finding IS_{CD} using Algorithm 2:

First we select a vertex from *C*. let this vertex be the first one and add it to IS_C, so IS_C = {126}.

Referring to step 4 in Algorithm 2, starting from vertex {126}, we find a set of vertices such that the distance between any pair of vertices is equal to *d*-1 = 3, then add them to IS_C. The algorithm finds only one vertex which is {236} so, IS_C = {126, 346}.

Since there is no a vertex such that the distance between a vertex and both {126, 346} is greater than 2, step 5 will not find any vertex.

The algorithm repeats the previous steps for all vertices in *C*. The result is the following sets,

IS_C = {136, 246}, IS_C = {146, 236}, IS_C = {156, 236}, IS_C = {236, 146}, IS_C = {246, 136}, IS_C = {256, 136}, IS_C = {346, 126}, IS_C = {356, 126}, IS_C = {456, 126}.

Since all IS_C have the same cardinality we will select any set, let IS_C = {126, 346}.

Step 7 deletes the adjacent vertices which are {457, 357, 347} and {127, 157, 257} from set *D*.

Step 8 repeats the previous steps for *D* and the result will be IS_D= {137, 247}.

Finally, IS_{CD} = $IS_C \cup IS_D$= {126,137,247,346}.

Finding IS_{CD} using Algorithm 3:

Step 4 in Algorithm 3 selects the first vertex from set *C* which is {126} , the algorithm checks if it is adjacent to *d*-1 vertices which is true so, we add it to IS_{CD} and delete it with the adjacent vertices which are {457, 357, 347}.

Step 5 selects the first vertex from *D* that is adjacent to 3 vertices which is {127} add it to IDS$_{CD}$ and delete it with the adjacent vertices which are {346, 356, 456}.

At this stage there are no more vertices that are adjacent to 3 vertices, so we check for vertices that are adjacent to two vertices. Step 4 and 5 select vertex {136} from *C* and vertex {137} from *D* which are adjacent to two vertices and delete them with the adjacent vertices which are {257, 247} and {256, 246}. Again, step 4 and 5 select vertices from *C* and *D* that are adjacent to 2 vertices, the algorithm selects {236} and {237} adds them to IS$_{CD}$ and deletes them with the adjacent vertices which are {157,147} and {156, 146}.

At this stage there are no more vertices that are adjacent to 2 vertices, so we stop. IS$_{CD}$ = {126, 127, 136, 137, 236, 237}.

Third stage:

We will choose the result found by Algorithm 2, so IDS$_{CD}$ = {126, 137, 346, 247}. The result graph is shown in Figure 14.

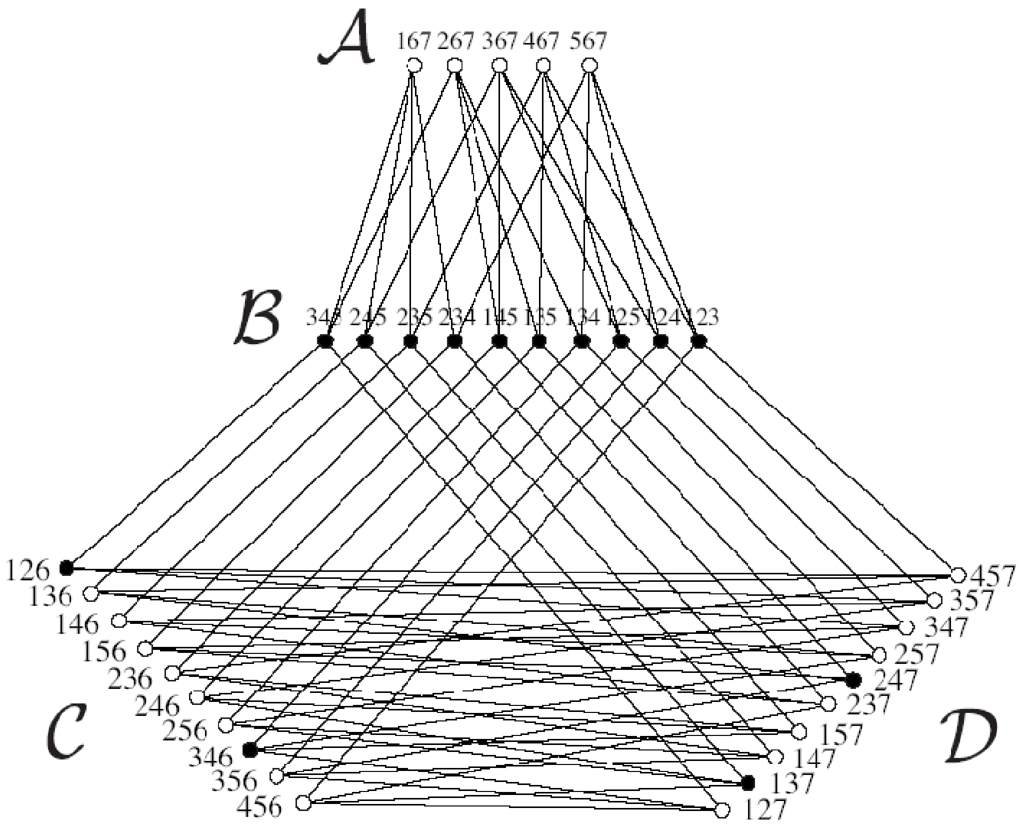


Figure 14: O_4 after the second stage.

Step 3 in Algorithm 1 deletes vertices from set IDS_B that are adjacent to any vertex in IS_{CD}. Those vertices are {345, 125, 135, 245}. So, IDS_B= {123, 125, 135, 145, 234, 235}. The result graph is shown in Figure 15.

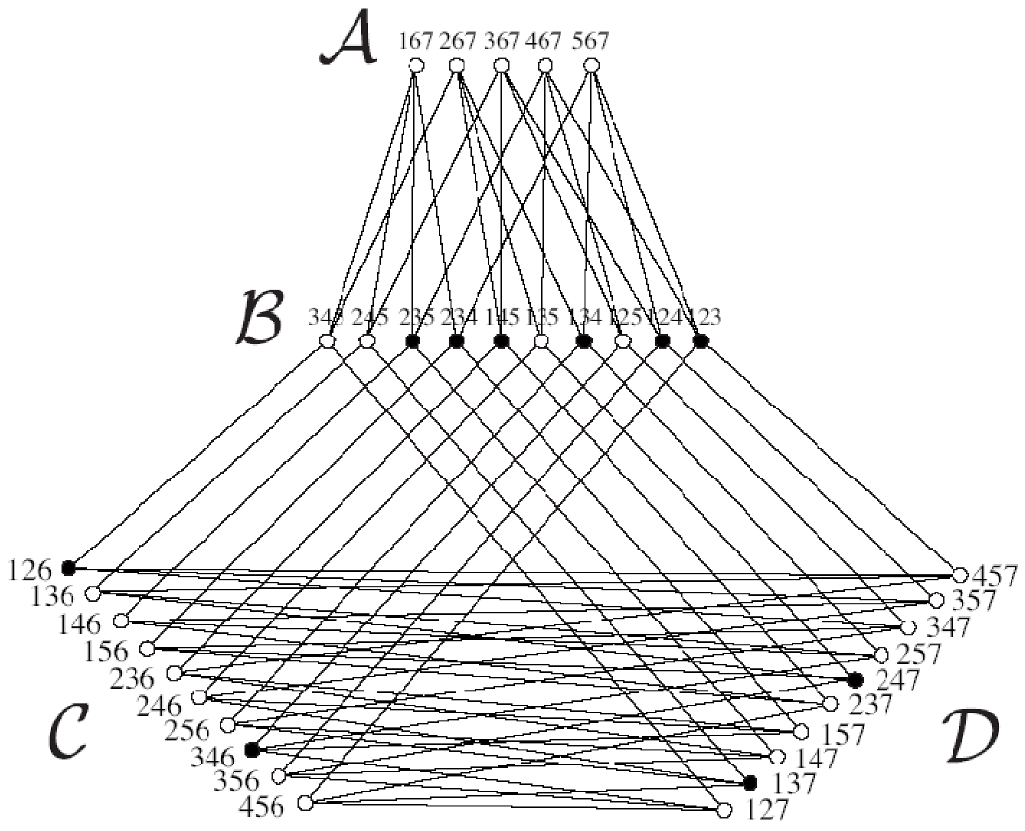


Figure 15: O_4 after removing deletes vertices from set IDS_B.

Step 4 finds the fixed vertices in IDS_B and set *A*. Notice the vertices {237, 147} from set *D* and {146, 236} from set *C*, they are not dominated by any vertices from *D* and *C*, those vertices are connected to vertices {145, 235} from IDS_B which cannot be removed from IDS_B. The adjacent vertices to {145, 235} from set *A* will be the fixed vertices in set *A* which are {267, 367, 167, 467}. The graph after finding the fixed vertices is shown in Figure 16 (The fixed vertices are circled).

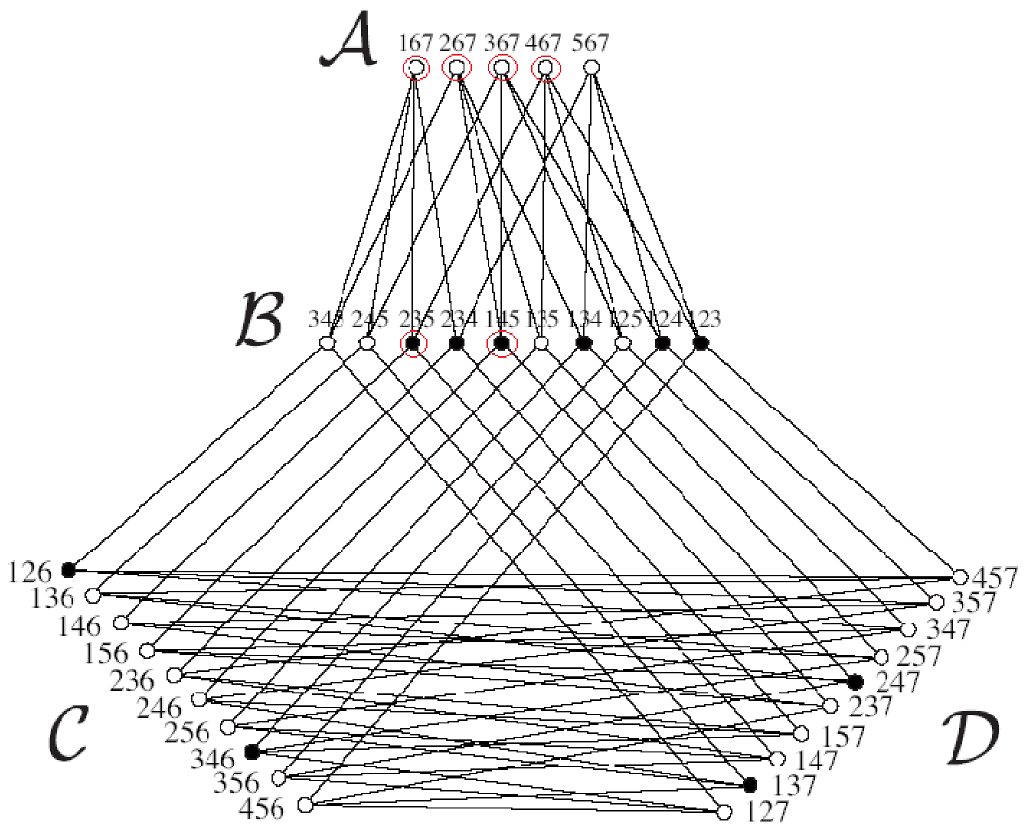


Figure 16: O_4 after the third stage.

Fourth stage:

Step 5: vA = $A\backslash$ fixed$A$ = {567}

Step 8 selects {567} from vA and checks the number of adjacent vertices which are {123, 124,134, 234} from set IDS_B. Since {567} is adjacent to d = 4 vertices and removing the adjacent vertices will not violate the domination property, we add it to IDS_A, delete it from vA and delete the adjacent vertices from IDS_B which are {145, 235}. Since there is no more vertices in vA we stop. IDS_{BA} = $\text{IDS}_B \cup \text{IDS}_A$ = {145, 235, 567}.

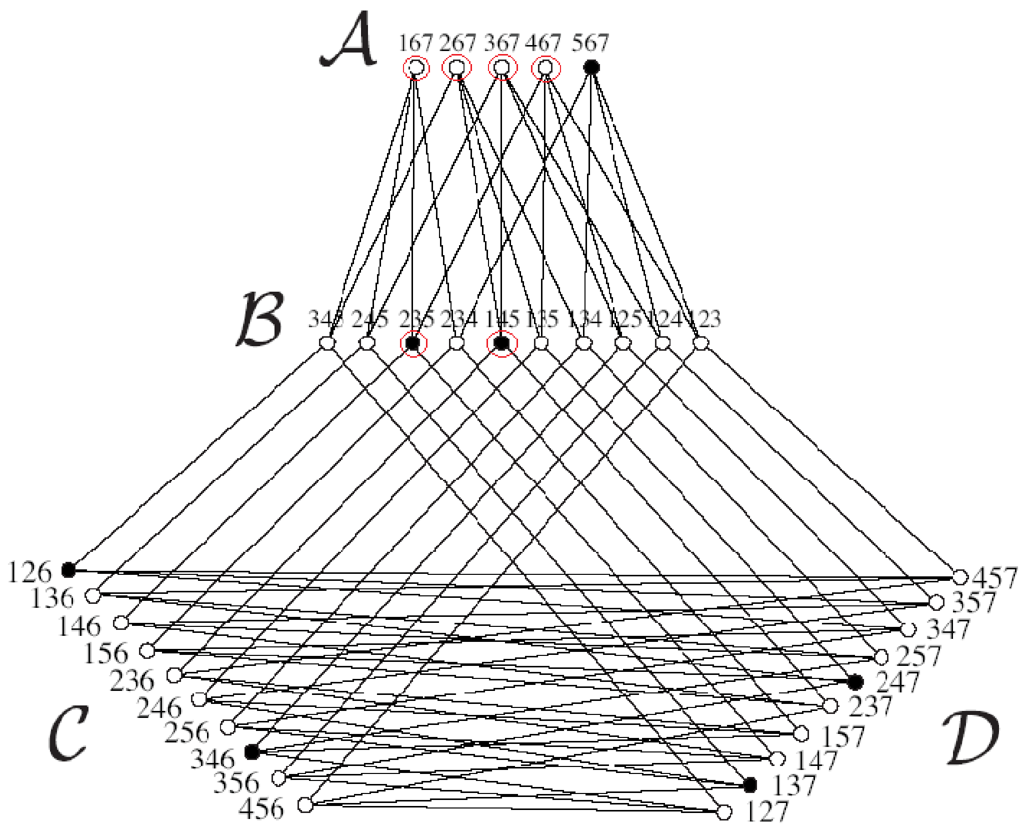


Figure 17: O_4 after the fourth stage.

Step 11: IDS = $IS_{CD} \cup IDS_{BA}$ = {126, 247, 346, 137, 145, 235, 567}. The final IDS can be seen in Figure 17.

4.3.2 CORRECTNESS

In this section, we will show that Algorithm 1 always find a correct independent dominating set, in particular we will prove that the algorithm maintains the independence and the domination properties throughout the algorithm execution.

Proposition 4.2: Approx. 1 for $d \leq 7$ and Approx. 2 for any d find a correct independent dominating set.

Proof:

It is clear that the set of vertices in set B is a maximal independent set, so the independence and the domination properties are maintained. After the algorithm finds IS_{CD} from set C and D, removing the adjacent vertices from IDS_B in step 3 in Algorithm 1 must lead to correct IDS which must preserve the domination and the independence properties. Now let's consider removing adjacent vertices from IDS_B, we have three cases where a vertex from set IDS_B must be removed to maintain the independence property. Let the dominating vertex, which is adjacent to a vertex from set IDS_B; be from set D, the argument is true for a dominating vertex from set C by symmetry. The first case (Figure 18): a vertex from IDS_B is connected to dominating vertices from set D and set C. The algorithm can remove vertex b_1 without violating the domination and the independence properties. Vertices c_n and d_n are not connected since there is no a cycle of length 3 in O_d so, vertices c_n and d_n are dominating and independent.

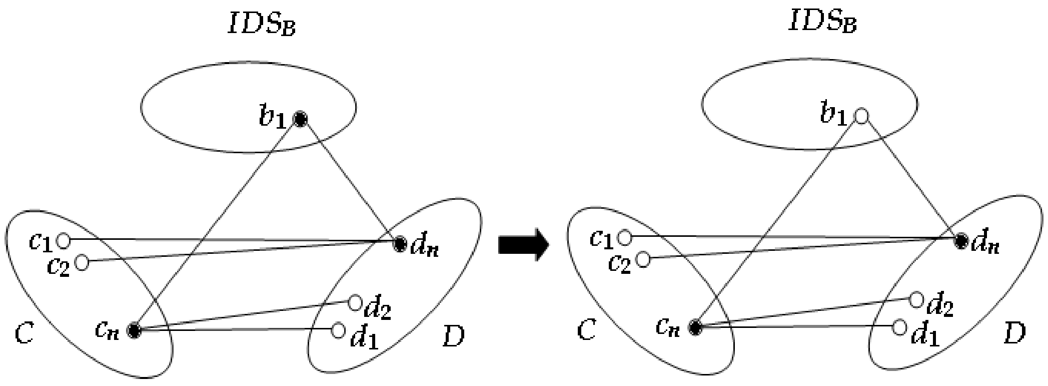


Figure 18: The first case of removing vertices from IDS$_B$.

Case two (Figure 19): a vertex from set IDS$_B$ is connected to a dominating vertex from *D* and an undominating vertex from set *C*. In addition, vertex c_n is also dominated by different vertex from *D*. We can remove vertex b_1 without violating the domination and independence properties. Since vertex c_n is dominated and the two vertices from *D* are independent so, we preserve the domination and the independent properties.

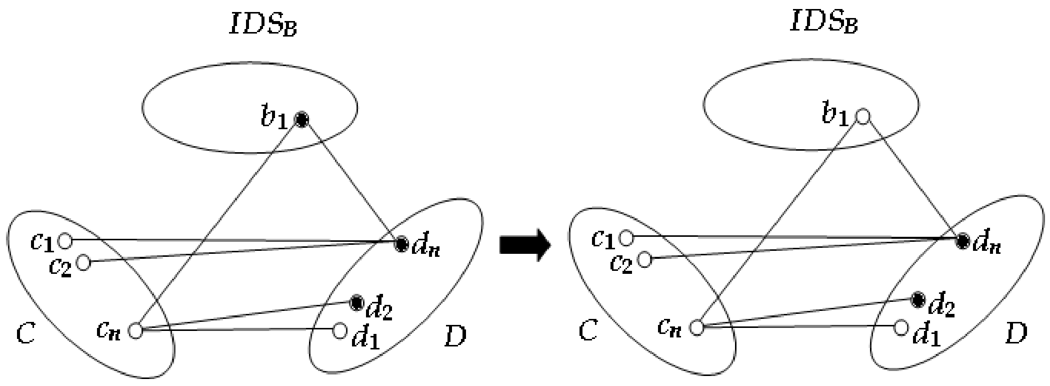


Figure 19: The second case of removing vertices from IDS$_B$.

The third case (Figure 20): a vertex from IDS$_B$ is connected to a dominating vertex from *D* and an undominating vertex from *C*. In addition, c_n vertex is also not dominated by any vertex from *D*. When we remove vertex b_1, the domination property will be violated since vertex c_n will be not dominated by any vertex. The presented approximation algorithms do not allow case 3, let's consider the two methods to find set IS$_{CD}$.

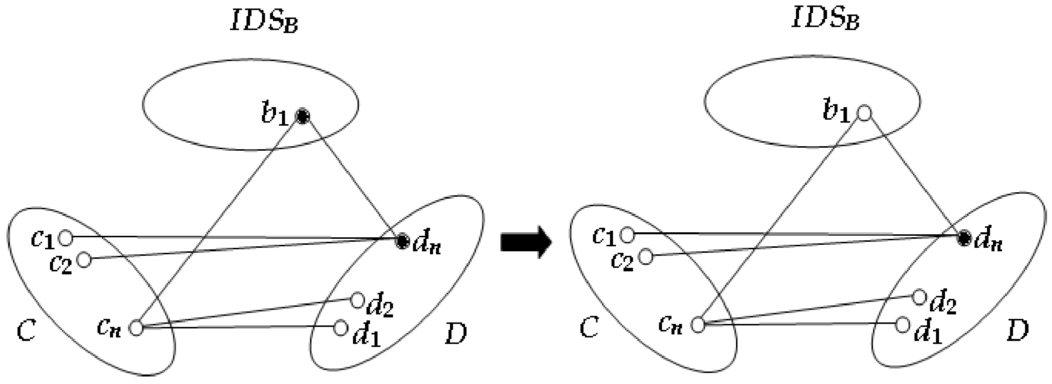


Figure 20: The third case of removing vertices from IDS$_B$.

Method 1: the algorithm finds the maximum independent set such that the distances between all vertices are greater than two, notice that the distance between the dominating vertex d_n and the undominating vertices from *D* is at least 3 which means the independent dominating set is not maximum, since one of these vertices must be dominating to have maximum independent dominating set, so this case is prevented. Method

1 does not find the maximum independent set for $d \geq 8$, so this case exists when $d \geq 8$. Method 2: If vertex c_n is connected to d-1 undominating vertices from set D (the algorithm has not selected any vertex from the d-1 vertices), then vertex c_n must have been chosen as a dominating vertex so we cannot have such a case.

In addition to the previous cases, we have one case where removing a vertex from IDS_B is caused by selecting a vertex from set A to be added to the dominating set. This case is shown in Figure 21. This case occurs when vertex b_1 is connected to vertices from set C and D such that they are not dominated by any other vertices either from set D and C respectively.

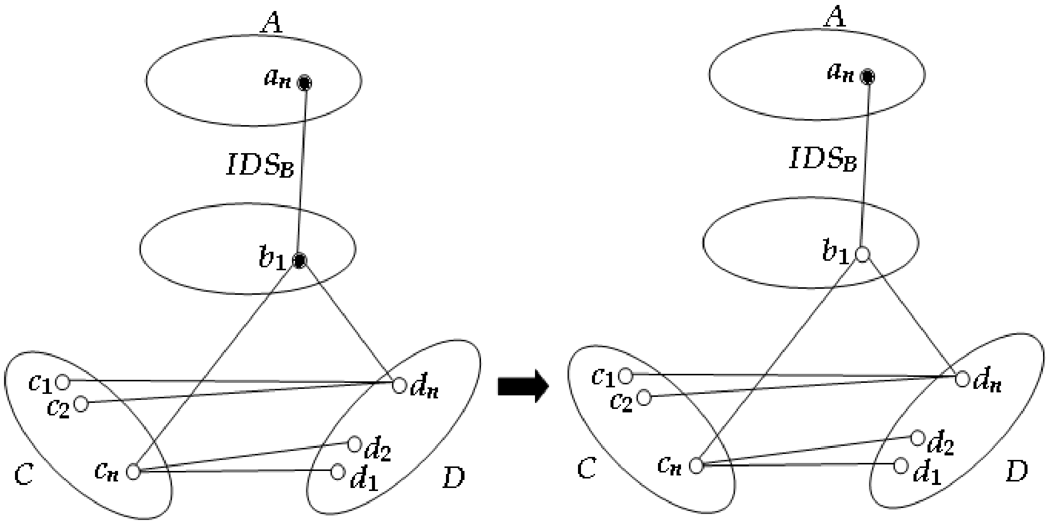


Figure 21: A case of removing vertices from IDS_B which violates the domination property.

If vertex b_1 is removed, vertices C and D will be undominated which violate the domination property. The algorithm does not allow removing vertex b_1 by making it a fixed vertex, consequently all d-2 vertices from set A, which are connected to that fixed vertex, cannot be chosen to be added to the dominating set, which means vertex b_1 cannot be removed from the independent dominating set, hence the algorithm preserves the domination prosperity.

When the algorithm chooses a vertex $a \in A$ to be added to the independent dominating set, a is either connected to dominating or dominated vertex $b \in B$. If b is dominated then adding a and removing b will not violated the domination and the independence properties, if b is dominating then removing the vertex b from the dominating set will preserve the domination property. We have one situation where domination property is violated, that is when a vertex a is connected to a dominating b, and the dominating b is connected to another undominating vertex from set A which is not connected to any other dominating vertex from B except b this situation is depicted in Figure 22. Clearly the algorithm prevents this situation by checking if a vertex from B is the only dominating vertex connected to a.□

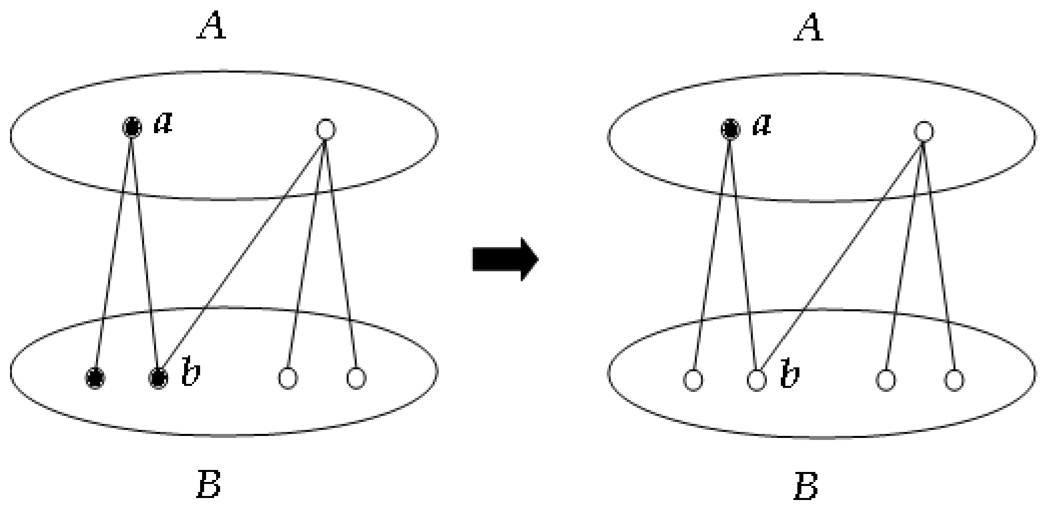


Figure 22: A situation where domination property is violated.

4.3.3 ANALYSIS

Partitioning the graph requires O($|V|$), Finding the vertices from set C and D costs O($2|V_{d-1}|^2$) using Algorithm 2, and O($2d^2|V_{d-1}|$) using Algorithm 3. Finding fixed vertices from set B requires O($|V_{d-1}|$). The process of adding vertices from set A to the independent dominating set costs O($d^4\ |A|$). So the time complexity is O($|V|+2|V_{d-1}|^2+\ |V_{d-1}|+\ d^4\ |A|$) = O($|V_{d-1}|^2$) using Approx.1 and O($|V|+2d^2|V_{d-1}|\ +|V_{d-1}|+d^4\ |A|$) = O($d^4\ |A|$) using Approx. 2.

4.4 GREEDY AND RANDOM ALGORITHMS

Two well-known algorithms for independent dominating set in a graph are greedy and random algorithms, which are listed in Figures 23 and 24. The algorithms are similar, difference is that in the random, the vertex selected in step 4 is selected at random; whereas in the greedy it is a maximum degree vertex (ties are broken randomly).

4.4.1 ALGORITHMS DESCRIPTION

The greedy algorithm selects a vertex of maximum degree, while the random algorithm selects a vertex at random, then both algorithms deletes that vertex and all of its neighbors from the graph, and repeats this process until the graph becomes empty.

Algorithm 4 Greedy Independent Dominating Set

```
Input: $O_d$
Output: Independent Dominating Set (IDS)
1: IDS ← ∅
2: while $V \neq \emptyset$ do
3: choose $v \in V$ such that the degree of $v$ is maximum
4: IDS ← IDS ∪ {$v$}
5: $V \leftarrow V \setminus N[v]$
6: end while
```

Figure 23: Algorithm 4 Greedy Independent Dominating Set.

4.4.2 CORRECTNESS

During the execution of the algorithm, the set of not yet considered vertices gives the set of all vertices that could be added to IDS without violating the independence property of IDS. Algorithm 4 and 5 constructs a maximal independent set, since we always remove all conflicting vertices.

4.4.3 ANALYSIS

It is clear from the algorithms that they require linear time in the number of vertices and edges, in addition to the time required for searching the maximum degree vertex in the greedy algorithm. However, the greedy algorithm can be implemented in time linear in the number of edges and vertices, independent of the degree.

Algorithm 5 Random Independent Dominating Set

```
Input: $O_d$
Output: Independent Dominating Set (IDS)
1: IDS $\leftarrow \emptyset$
2: while $V \neq \emptyset$ do
3: choose $v \in V$ at random
4: IDS $\leftarrow$ IDS $\cup \{v\}$
5: $V \leftarrow V \setminus N[v]$
6: end while
```

Figure 24: Algorithm 5 Random Independent Dominating Set.

4.5 EXPERIMENTAL RESULTS

4.5.1 EXPERIMENTAL SETUP

This section presents experimental results and comparisons of the approximation algorithms discussed above: the new approximation algorithms, the greedy and the randomized algorithms. All algorithms were performed on odd graphs of dimension 3 to dimension 13 except the first algorithm which was performed on odd graphs up to dimension 7.

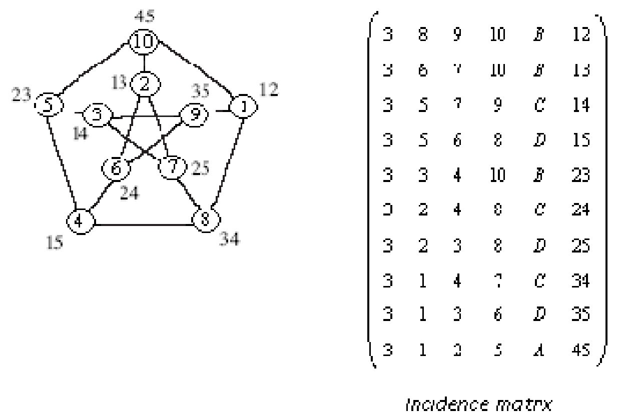
Our main measure of performance is the cardinality of the independent dominating set which is machine independent.

All algorithms were implemented using C sharp. We ran the experiments on Sun virtual machine running on 64-bit Windows 7 operating system, the virtual box install 64-bit Windows 7 with Intel Xeon @ 2.93 GHz CPU and 8 GB RAM running Windows 7.

I have used the Incidence Matrix structure (suggested by Dr. Al-Darwish who also gave BuildOddGraph() and GreedyMinIndDomSet() procedures) to represent odd graphs. For our purpose we defined the incidence matrix as the matrix IM[1..n, 0..(d+2)] as follows (see Figure 25):

- IM[i,0] is set to the degree of vertex i, and
- The i-th row IM[i,1..d] lists the vertices that are adjacent to i (i.e., IM[i,j]=x if and only if (i,x) is an edge).
- In addition, two additional columns can be used to store the set name that the vertex belongs to and the label of the vertex.

This representation is space efficient for graphs where the degree of any vertex is equal to *d*, such as odd graphs.



$$\begin{pmatrix} 3 & 8 & 9 & 10 & B & 12 \\ 3 & 6 & 7 & 10 & B & 13 \\ 3 & 5 & 7 & 9 & C & 14 \\ 3 & 5 & 6 & 8 & D & 15 \\ 3 & 3 & 4 & 10 & B & 23 \\ 3 & 2 & 4 & 8 & C & 24 \\ 3 & 2 & 3 & 8 & D & 25 \\ 3 & 1 & 4 & 7 & C & 34 \\ 3 & 1 & 3 & 6 & D & 35 \\ 3 & 1 & 2 & 5 & A & 45 \end{pmatrix}$$

Incidence matrx

Figure 25: O_3 and its incidence matrix representation.

4.5.2 EXPERIMENTAL RESULTS

For each algorithm, we consider its approximation quality. Table 1 shows the results of these experiments. Abbreviations in the table are as follows:

—Approx. 1: The approximation algorithm using (Algorithm 2) to find IS_{CD} such that $\partial(u, v) > 2$ for $\forall\ u, v \in IS_{CD}$.

—Approx. 2: The approximation algorithm using (Algorithm 3) to find IS_{CD} such that $\exists\ u, v \in IS_{CD}$ with $\partial(u, v) = 2$.

In odd graph of dimension three, the four algorithms' performances are similar providing the same approximation quality except the random algorithm which found a larger IDS. When an odd graph of dimension four is the input the Approx. 1 and the greedy algorithms provide the same performance ratio, while Approx. 2 and the randomized approach found worse results. In odd graph of dimensions five and higher, the algorithms start giving different performances. As shown in Table 1, the approximation quality of the Approx. 1 and 2 algorithms turns out to be higher than the greedy and the randomized algorithms. Moreover, Approx. 1 algorithm dominates Approx. 2 algorithm. This can be explained by the fact that Approx. 1 algorithm finds the maximum independent set in the induced bipartite $\mathcal{E}_{CD}$ with minimum distance of 3 which maximize the number of non overlapping neighbors of the dominating vertices which in turn minimizes the independent dominating set, whereas Approx. 2 algorithm allows finding independent set in the induced bipartite $\mathcal{E}_{CD}$ with distance of two. Furthermore, it was observed that the greedy approach give worse results as the number of vertices increases, since it selects a vertex with maximum degree among many vertices with the same degree without considering the degrees of the neighbors and the further neighbors. Also, the randomized approach selects a vertex at random which raises the possibility of selecting a vertex with lower degree which means a larger set of independent dominating vertices. It is worth noting that as the dimension of odd graphs increases the difference in the performance quality between the new approximation algorithms and the greedy and the randomized algorithm also increases.

TABLE 1 APPROXIMATION QUALITIES

d	$\|V\|$	$\|E\|$	Lower Bound*	Approx 1. |IDS|	Approx 2. |IDS|	Greedy |IDS|	Random |IDS|
3	10	15	3	3	3	3	4
4	35	70	7	7	10	7	11
5	126	315	21	26	26	39	41
6	462	1386	66	66	93	118	139
7	1716	6006	215	259	316	386	452
8	6435	25740	715	_	1097	1310	1519
9	24310	109395	2431	_	3842	4676	5503
10	92378	461890	8398	_	14217	15389	19726
11	352716	1939938	29393	_	48106	54696	71522
12	1352078	8112468	104006	_	175052	197582	261002
13	5200300	33801950	371450	_	637949	731096	955580

*Lower Bound = $\lceil |V|/(d+1) \rceil$

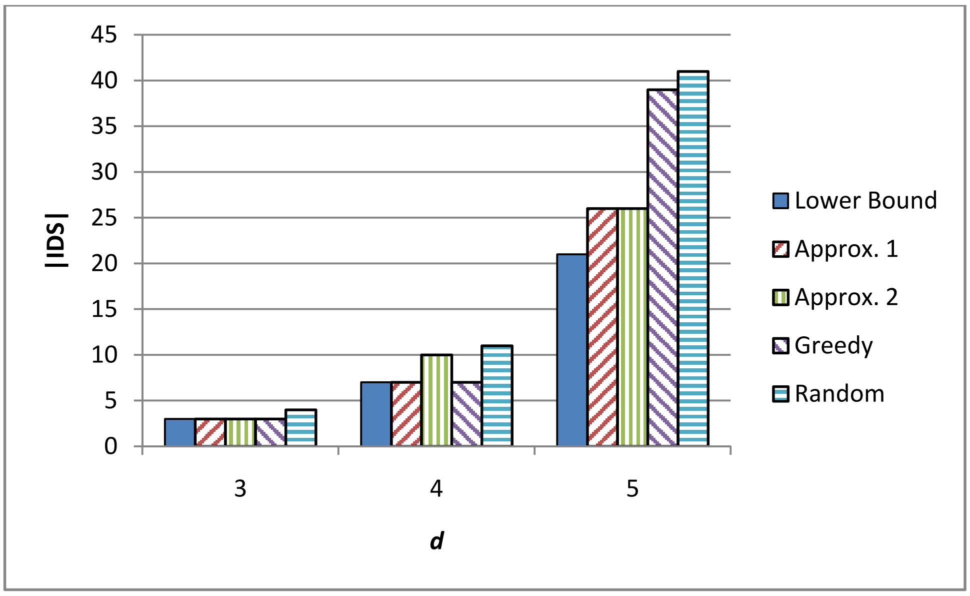


Figure 26: |IDS| in odd graphs of dimensions 3-5.

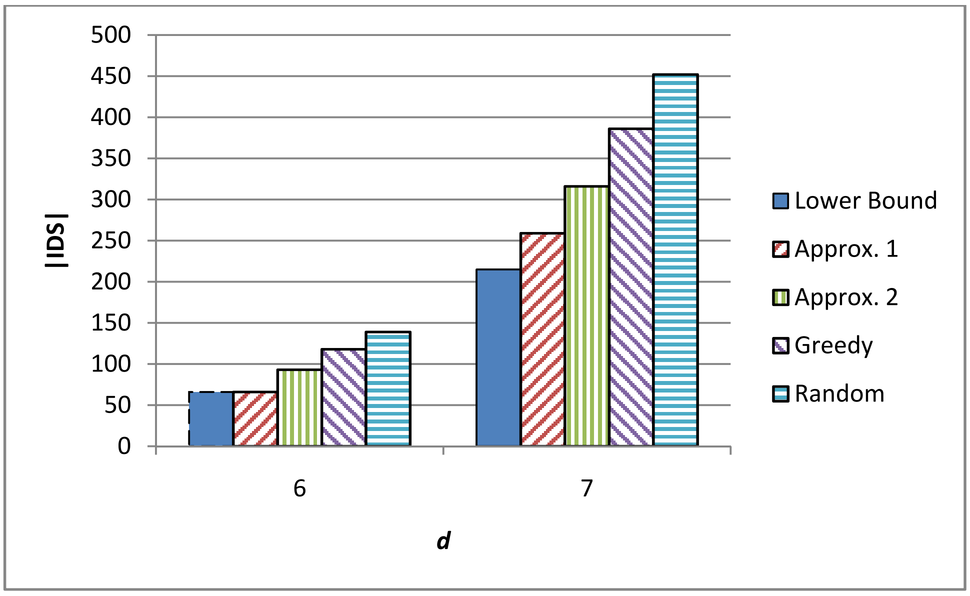


Figure 27: |IDS| in odd graphs of dimensions 6-7.

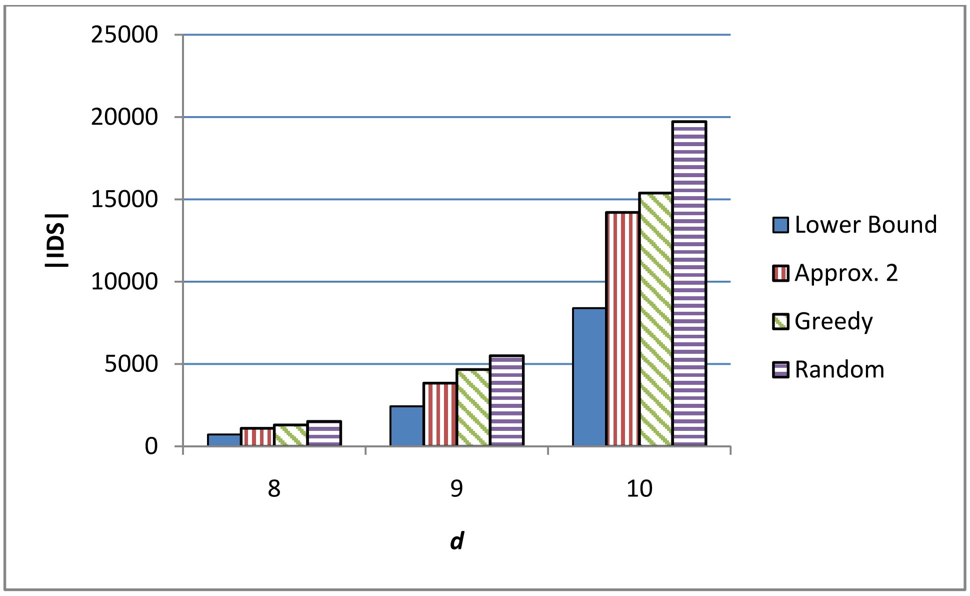


Figure 28: |IDS| in odd graphs of dimensions 8-10.

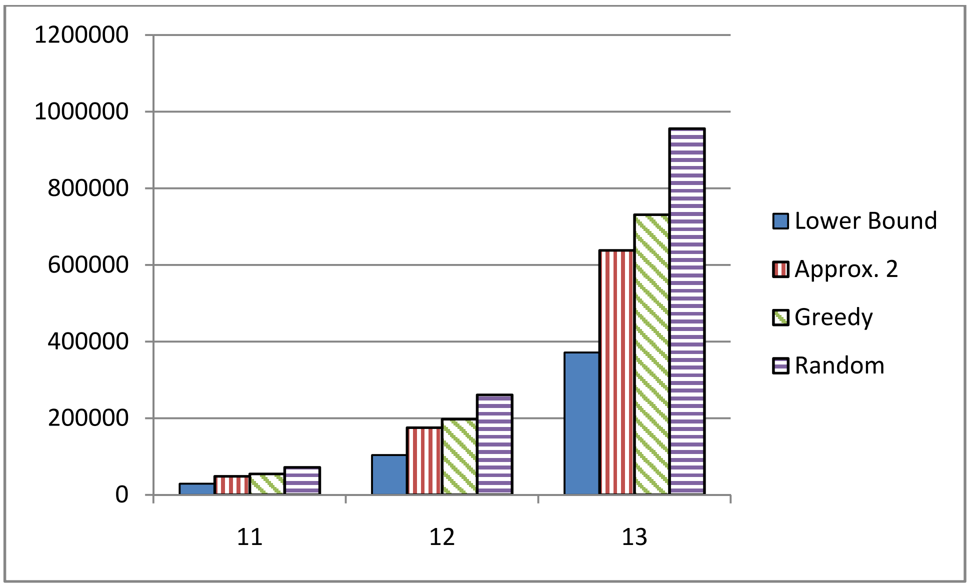


Figure 29: |IDS| in odd graphs of dimensions 11-13.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

As mentioned in the literature, independent and dominating sets in communication network are important structures, and many optimization approaches rely on these. Many exact and approximation algorithms were proposed in the past to solve the problem either on general or special family of graphs. One of the graph classes, which have not been investigated in term of independent domination, is the odd graphs class.

In this thesis, the first approximation algorithms for independent dominating set in odd graph are introduced. Our approach is based on partitioning the graph to different sets in order to simplify the complexity of the graph, then finding the independent dominating sets or the independent sets on the partitioned parts of the graph and merging the results while resolving any conflicts in the independence or domination properties. In this thesis, we designed two approximation algorithms, namely Approx.1 and Approx. 2. Approx. 1 produces the best results, however it gives correct results in odd graphs up to dimensions seven, for higher dimension the algorithm does not produce a valid independent dominating set since the solution to maximum independent set with distance greater than two between any vertices in the induced bipartite $\mathcal{E}_{CD}$ cannot be attained. Approx. 2 algorithm gives comparable excellent results and it produces a valid set for dimensions that are higher than seven since we relaxed the distance restriction to allow a distance of two between some vertices. In addition, we proved the correctness of the two approximation algorithms and analyzed them. Also, we presented experimental results and comparison between the two approximation algorithms and the greedy and the randomized algorithms. The results of the experiments show that Approx. 1 and Approx. 2 give the best approximation quality especially in high dimensional odd graphs.

In short, the following have been achieved in the thesis:

- The first approximation algorithms for MIDS in odd graphs are introduced.
- Analyses and correctness of the proposed approximation algorithms are presented.
- Experiments are presented, which show that the approximation algorithms find significantly smaller sets than those found by the greedy and the random algorithms.

There are several open problems that can be investigated in future works. The following summarizes some of the interested problems:

- Designing an approximation algorithm for independent dominating set in odd graph with weighted vertices or edges.
- Finding the upper bound of the proposed approximation algorithms.
- Proving or disproving the following conjecture: Approx. 1 algorithm finds the optimal set.
- Proving or disproving the following conjecture: the minimum independent dominating problem in odd graph is in $\mathbb{P}$ if and only if the problem of maximum independent set on the induced bipartite $\mathcal{E}_{CD}$ with minimum distance of three is in $\mathbb{P}$.

APPENDIX A

ALGORITHMS IMPLEMENTATION

```
using System;
using System.IO;
using System.Collections.Generic;

//Authors: Ahmed Al-Herz and Dr. Nasir Al-Darwish

namespace IndepDomSet
{
       class ApproximationAlg
       {
        static int[,] IM ;//incidence matrix for a graph -- column 0 records count
                         //of adjacent vertices
        //cw is used in BuildOddGraph();
       // vertices in ODD graph are numbered 1 to n where  cw[i] is the
       //corresponding set (as bit vector)
        static int[] cw;
        static int size_B; //size of set B, C or D in odd graphs
        static int size_A; //size of set A in odd graph


        static void Main(string[] args)
        {
            int n ;
            int[] S ;
            for (int d = 5; d < 10; d++)
            {
                n = BuildOddGraph(d);
                int k = RandomMinIndDomSet(out S, n, d);
  Console.WriteLine("for odd d= " + d +" "+ValidIndpDomSet(S, n) + " " + k +"\n");

            }
            return;
        }

       // This procedure tests if the vertices where S[i] = 1 form a covering
       //IS
        static bool ValidIndpDomSet(int[] S, int n )
        {
            for (int v = 1; v <= n; v++)
                if (S[v] == 1) // check Independence
                {  for (int i = 1; i <= IM[v, 0]; i++)
                    if (S[IM[v, i]] == 1) {return false; }
                }
                else if (!IsCovered(S, v)) {return false; }
            return true;
        }

        static bool IsCovered(int[] s, int v)
        {
       // v is covered if one of its neighbors is a vertex in S and where S[i] = 1
            for (int i = 1; i <= IM[v, 0]; i++)
                if (s[IM[v, i]] == 1) return true;

            return false;
        }

         // Greedy algorithm for for Min Indpendent Dominating set
```

```
static int GreedyMinIndDomSet(out int[] s, int n, int dimension)
{
    int maxdeg, maxv;
    // s[v] = 0 unchecked , 1 in IDS,  -1 covered
    int[] deg = new int[n + 1];
    s = new int[n + 1];

    for (int v = 1; v <= n; v++) { s[v] = 0; deg[v] = IM[v, 0]; }

    // find vertex  of max degree
    // Note: degree is updated to discount covered vertices

    int vcount = 0;

    while (true)
    {   // find vertex in S with maximum degree (maxdeg)
        maxdeg = int.MinValue; maxv = 0;
        for (int v = 1; v <= n; v++)
            if ((s[v] == 0) && (deg[v] > maxdeg))
            {
                maxdeg = deg[v];
                maxv = v;
            }

        if (maxv == 0) break;

        if (s[maxv] == -1) Console.WriteLine(" vertex already covered");

        // add the verex maxv to IDS
        s[maxv] = 1;
        vcount++;

   // Now update degree to discount covered vertices (i.e. neighbours of
  //maxv)
        for (int i = 1; i <= dimension; i++)
        {
            int v = IM[maxv, i];
            if (s[v] == -1) continue;

            s[v] = -1;
            for (int j = 1; j <= dimension; j++)
               if (deg[IM[v, j]]> 0)
                  deg[IM[v, j]]--;
        }
    }

    return vcount;

}

// Random algorithm for for Min Indpendent Dominating set
static int RandomMinIndDomSet(out int[] s, int n, int dimension)
{

    // s[v] = 0 unchecked , 1 in IS,  -1 covered
    int[] deg = new int[n + 1];
    s = new int[n + 1];

    for (int v = 1; v <= n; v++) { s[v] = 0; deg[v] = IM[v, 0]; }

    int vcount = 0;

    Random r = new Random(); // random generator

    List<int> vertexset = new List<int>();

    //populate a list with all vertices
    for(int i =1; i<=n; i++)vertexset.Add(i);

    while (true)
    {
        if (vertexset.Count == 0) break;
```

```
                int randomIndex =r.Next(vertexset.Count);
                int targetV = vertexset[randomIndex]; // select a random vertex
                vertexset.Remove(targetV); // remove the vertex from the list
                s[targetV] = 1; // add the verex maxv to IS
                vcount++;
// Now update degree to discount covered vertices (i.e. neighbors of the selected
//vertex)
                for (int i = 1; i <= dimension; i++)
                {
                    int v = IM[targetV, i];
                    if (s[v] == -1) continue;
                    vertexset.Remove(v); // remove the neighbors from the list
                    s[v] = -1;
                    for (int j = 1; j <= dimension; j++)
                       if (deg[IM[v, j]]> 0)
                              deg[IM[v, j]]--;
                }
            }

            return vcount;
        }

        // Approx2 algorithm for for Min Indpendent Dominating set
        static int Approx2(out int[] s, int n, int dimension)
        {

            int[] setA = new int[size_A + 1];
            int[] setB = new int[size_B + 1];
            int[] setC = new int[size_B + 1];
            int[] setD = new int[size_B + 1];
            int[] setApos = new int[size_A + 1];
            int[] setBpos = new int[size_B + 1];
            int[] setCpos = new int[size_B + 1];
            int[] setDpos = new int[size_B + 1];
            int[] ISd = new int[size_B + 1];
            int[] ISdpos = new int[size_B + 1];
            int[] ISc = new int[size_B + 1];
            int[] IScpos = new int[size_B + 1];
            int[] stemp = new int[size_B + 1];
            int[] IDSa = new int[size_A + 1];
            int[] IDSapos = new int[size_A + 1];
            int idsblength = size_B;

            int a = 1, b = 1, c = 1, d = 1;

            // s[v] = 0 unchecked , 1 in IS,  -1 covered
            int[] deg = new int[n + 1];
            s = new int[n + 1];

            for (int v = 1; v <= n; v++) { s[v] = 0; deg[v] = IM[v, 0]; }

            //partitioning the graph to four sets
            for (int i = 1; i <= n; i++)
            {
                if (IM[i, dimension + 1] == 1)
                      { setA[a] = IM[i, dimension + 2]; setApos[a] = i; a++; }
                if (IM[i, dimension + 1] == 2)
                      { setB[b] = IM[i, dimension + 2]; setBpos[b] = i; b++; }
                if (IM[i, dimension + 1] == 3)
                      { setC[c] = IM[i, dimension + 2]; setCpos[c] = i; c++; }
                if (IM[i, dimension + 1] == 4)
                      { setD[d] = IM[i, dimension + 2]; setDpos[d] = i; d++; }
            }

            int kc = 1;
            int kd = 1;

            //Find independent set in the bipartite CD
            for (int degree = dimension; degree >= 3; degree--)
            {
                for (int i = size_B; i >=1 ; i--)
                {
```

```
                if (deg[setDpos[i]] == degree && s[setDpos[i]] == 0)
                {
                    ISd[kd] = setD[i];
                    ISdpos[kd] = setDpos[i];
                    kd++;
                    s[setDpos[i]] = 1;
                    for (int j = 1; j <= dimension; j++)
                    {
                        int v = IM[setDpos[i], j];
                        if (s[v] == -1 || IM[v, dimension + 1] == 2) continue;

                        s[v] = -1;

                        for (int k = 1; k <= dimension; k++)
                    { if (IM[IM[v, k], dimension + 1] == 4) deg[IM[v, k]]--; }
                    }
                }

                if (deg[setCpos[i]] == degree && s[setCpos[i]] == 0)
                {
                    ISc[kc] = setC[i];
                    IScpos[kc] = setCpos[i];
                    kc++;
                    s[setCpos[i]] = 1;
                    for (int j = 1; j <= dimension; j++)
                    {
                        int v = IM[setCpos[i], j];
                        if (s[v] == -1 || IM[v, dimension + 1] == 2) continue;

                        s[v] = -1;

                        for (int k = 1; k <= dimension; k++)
                    { if (IM[IM[v, k], dimension + 1] == 3) deg[IM[v, k]]--; }
                    }
                }
            }
        }

        int isdlength = kd-1;

        for (int i = 1; i <= kd - 1; i++)
        {
            for (int j = 1; j <= dimension; j++)
            {
                int v = IM[ISdpos[i], j];
                if (s[v] == -1) continue;

                s[v] = -1;

                if (IM[v, dimension + 1] == 2)
                {
                    idsblength--;
                    for (int k = 1; k <= dimension; k++)
                    { if (IM[IM[v, k], dimension + 1] == 1) deg[IM[v, k]]--; }
                }

            }
        }

        int isclength = kc - 1;

        for (int i = 1; i <= kc - 1; i++)
        {
            for (int j = 1; j <= dimension; j++)
            {
                int v = IM[IScpos[i], j];
                if (s[v] == -1) continue;

                s[v] = -1;

                if (IM[v, dimension + 1] == 2)
                {
```

```
                idsblength--;
                for (int k = 1; k <= dimension; k++)
                { if (IM[IM[v, k], dimension + 1] == 1) deg[IM[v, k]]--; }
            }

        }
    }

    HashSet<int> fixedB = new HashSet<int>();
    HashSet<int> fixedA = new HashSet<int>();

    ////Find fixed vertices in set B and A
    for (int i = size_B; i >= 1; i--)
    {
        if (s[setDpos[i]] == 0)
        {
            for (int j = 1; j <= dimension; j++)
            {
                int v = IM[setDpos[i], j];
                if (IM[v, dimension + 1] == 2)
                {
                    s[v] = 1;
                    for (int k = 1; k <= dimension; k++) s[IM[v, k]] = -1;
                    fixedB.Add(IM[v, dimension + 2]);
                }
            }
        }

        if (s[setCpos[i]] == 0)
        {
            for (int j = 1; j <= dimension; j++)
            {
                int v = IM[setCpos[i], j];
                if (IM[v, dimension + 1] == 2)
                {
                    s[v] = 1;
                    for (int k = 1; k <= dimension; k++) s[IM[v, k]] = -1;
                    fixedB.Add(IM[v, dimension + 2]);
                }
            }
        }
    }

    for (int i = size_A; i >= 1; i--)
    {
        for (int j = 1; j <= dimension; j++)
        {
            int v = IM[setApos[i], j];
           if (fixedB.Contains(IM[v, dimension + 2]))
              { fixedA.Add(IM[setApos[i], dimension + 2]); }
        }
    }


    bool violate = false;
    int ka = 1;
    int numofvio = 0;
    int limofvio = size_A - fixedA.Count;
    int degrees = dimension;

    //Find independent dominating set in the bipartite BA
    while (limofvio > 0 && limofvio > numofvio && degrees >= 2)
    {
        numofvio = 0;

        for (int j = size_A; j >= 1; j--)
        {
            violate = false;
            if (!fixedA.Contains(setA[j]) && s[setApos[j]] == 0)
            {
                if (deg[setApos[j]] == degrees)
                {
                    for (int k = 1; k <= dimension; k++)
                    {
                        int v = IM[setApos[j], k];
                        if (s[v] == 0)
                        {
                            for (int l = 1; l <= dimension; l++)
                            {
               if (IM[IM[v, l], dimension + 1] == 1 && deg[IM[v, l]] == 1)
                                {
                                    violate = true; break;
                                }
                            }
                        }
                        if (violate)
                            break;
                    }

                    if (!violate)
                    {
                        IDSa[ka] = setA[j];
                        IDSapos[ka] = setApos[j];
                        ka++;
                        limofvio--;
                        s[setApos[j]] = 1;
                        for (int k = 1; k <= dimension; k++)
                        {
                            int v = IM[setApos[j], k];
                            if (s[v] == -1) continue;

                            s[v] = -1;
                            idsblength--;

                            for (int l = 1; l <= dimension; l++)
                { if (IM[IM[v, l], dimension + 1] == 1) deg[IM[v, l]]--; }
                        }
                    }
                    else
                    {
                        numofvio++;
                    }
                }
            }
        }

        degrees--;
    }

    //covering vertices from A that caused violation and that of degree 1
    for (int i = 1; i <= size_B; i++)
    {
        if (s[setBpos[i]] == 0)
        {
            s[setBpos[i]] = 1;
            for (int j = 1; j <= dimension; j++)
            {
                s[IM[setBpos[i], j]] = -1;
            }
        }
    }


    return (ka - 1 + isclength + isdlength + idsblength);

}


// Approx1 algorithm for for Min Indpendent Dominating set
static int Approx1(out int[] s, int n, int dimension)
{

    int[] setA = new int[size_A + 1];
    int[] setB = new int[size_B + 1];
```

```
int[] setC = new int[size_B + 1];
int[] setD = new int[size_B + 1];
int[] setApos = new int[size_A + 1];
int[] setBpos = new int[size_B + 1];
int[] setCpos = new int[size_B + 1];
int[] setDpos = new int[size_B + 1];
int[] temp = new int[size_B + 1];
int[] temppos = new int[size_B + 1];
int[] ISd = new int[size_B + 1];
int[] ISdpos = new int[size_B + 1];
int[] ISc = new int[size_B + 1];
int[] IScpos = new int[size_B + 1];
int[] stemp = new int[size_B + 1];
int[] IDSa = new int[size_A + 1];
int[] IDSapos = new int[size_A + 1];
int idsblength=size_B;
int count,count2;
int a=1,b=1,c=1,d=1,z;

// s[v] = 0 unchecked , 1 in IS,  -1 covered
int[] deg = new int[n + 1];
s = new int[n + 1];

for (int v = 1; v <= n; v++) { s[v] = 0; deg[v] = IM[v, 0]; }

for (int i = 1; i <= n; i++)
{
    if (IM[i, dimension + 1] == 1)
          { setA[a] = IM[i, dimension + 2]; setApos[a] = i; a++; }
    if (IM[i, dimension + 1] == 2)
          { setB[b] = IM[i, dimension + 2]; setBpos[b] = i; b++; }
    if (IM[i, dimension + 1] == 3)
          { setC[c] = IM[i, dimension + 2]; setCpos[c] = i; c++; }
    if (IM[i, dimension + 1] == 4)
          { setD[d] = IM[i, dimension + 2]; setDpos[d] = i; d++; }
}

int kd = 1;
int maxkd = 1;

int not2 = dimension - 2; //intersection size if the distance equals 2
int diam;
//intersection size if the distance equals diameter
if (dimension % 2 == 0)
    diam = (dimension - 2) / 2;
else
    diam=(dimension-1)-((dimension-1)/2);

//Find independent set in the bipartite CD
for (int i = size_B; i >= 1; i--)
{
    kd = 1;
    for (int j = 1; j <= size_B; j++) { stemp[j] = 0; }
    temp[kd] = setD[i];
    stemp[i] = 1;
    temppos[kd] = setDpos[i];
    for (int j = size_B; j >=1 ; j--)
    {
        count2 = 0;
        if (stemp[j] == 1) continue;
        for (int k = 1; k <= kd; k++)
        {
            count = 0;
            z = setD[j] & temp[k];
            while (z != 0)
            { count = count + (z % 2); z = z / 2; }
            if (count == diam)
            { count2++; }
        }

        if (count2 == kd)
        { kd++; temp[kd] = setD[j]; temppos[kd] = setDpos[j];
```

```
          stemp[j] = 1; }
    }

    for (int j = size_B; j >=1 ; j--)
    {
        count2 = 0;
        if (stemp[j] == 1) continue;

        for (int k = 1; k <= kd; k++)
        {
            count = 0;
            z = setD[j] & temp[k];
            while (z != 0)
            { count = count + (z % 2); z = z / 2; }
            if (count != not2)
            { count2++; }
        }

        if (count2 == kd)
        { kd++; temp[kd] = setD[j]; temppos[kd] = setDpos[j];
          stemp[j] = 1; }
    }

    if (kd > maxkd )
    {
        maxkd = kd;
        Array.Copy(temp, ISd, size_B + 1);
        Array.Copy(temppos, ISdpos, size_B + 1);
    }
}
int isdlength=maxkd;

for (int i = 1; i <= isdlength; i++) s[ISdpos[i]] = 1;

for (int i = 1; i <= isdlength; i++)
{
    for (int j = 1; j <= dimension; j++)
    {
        int v = IM[ISdpos[i], j];
        if (s[v] == -1) continue;

        s[v] = -1;

    if (IM[v, dimension + 1] == 2)
    {
        idsblength--;
        for (int k = 1; k <= dimension; k++)
        { if (IM[IM[v, k], dimension + 1] == 1) deg[IM[v, k]]--; }
    }

    }
}

int kc = 1;
int maxkc = 1;
for (int i = size_B; i >=1 ; i--)
{
    kc = 1;
    for (int j = 1; j <= size_B; j++) { stemp[j] = 0; }
    if (s[setCpos[i]] == -1) continue;

    temp[kc] = setC[i];
    temppos[kc] = setCpos[i];
    stemp[i] = 1;
    for (int j = size_B; j >=1 ; j--)
    {
        count2 = 0;
        if (s[setCpos[j]] == -1) continue;
        if (stemp[j] == 1) continue;
        for (int k = 1; k <= kc; k++)
        {
            count = 0;
```

```
                z = setC[j] & temp[k];
                while (z != 0)
                { count = count + (z % 2); z = z / 2; }
                if (count == diam)
                { count2++; }

            }

            if (count2 == kc)
            { kc++; temp[kc] = setC[j]; temppos[kc] = setCpos[j];
              stemp[j] = 1; }
        }

        for (int j = size_B; j >=1 ; j--)
        {
            count2 = 0;
            if (s[setCpos[j]] == -1) continue;
            if (stemp[j] == 1) continue;
            for (int k = 1; k <= kc; k++)
            {
                count = 0;
                z = setC[j] & temp[k];
                while (z != 0)
                { count = count + (z % 2); z = z / 2; }
                if (count != not2)
                { count2++; }
            }

            if (count2 == kc)
            { kc++; temp[kc] = setC[j]; temppos[kc] = setCpos[j];
              stemp[j] = 1; }
        }
        if (kc > maxkc)
        {
            Array.Copy(temp, ISc, size_B + 1);
            Array.Copy(temppos, IScpos, size_B + 1);
            maxkc = kc;
        }
    }

    int isclength=maxkc;

    for (int i = 1; i <= isclength; i++) s[IScpos[i]] = 1;

    for (int i = 1; i <= isclength; i++)
    {
        for (int j = 1; j <= dimension; j++)
        {
            int v = IM[IScpos[i], j];
            if (s[v] == -1) continue;

            s[v] = -1;

            if (IM[v, dimension + 1] == 2)
            {
                idsblength--;
                for (int k = 1; k <= dimension; k++)
                { if (IM[IM[v, k], dimension + 1] == 1) deg[IM[v, k]]--; }
            }

        }
    }

    HashSet<int> fixedB = new HashSet<int>();
    HashSet<int> fixedA = new HashSet<int>();

    ////Find fixed vertices in set B and A
    for (int i = size_B; i >=1 ; i--)
    {
        if (s[setDpos[i]] == 0)
        {
            for (int j = 1; j <= dimension; j++)
        {
            int v = IM[setDpos[i], j];
            if (IM[v, dimension + 1] == 2)
            {
                s[v] = 1;
                for (int k = 1; k <= dimension; k++)
                    s[IM[v, k]] = -1;
                fixedB.Add(IM[v, dimension + 2]);
            }
        }
    }

    if (s[setCpos[i]] == 0)
    {
        for (int j = 1; j <= dimension; j++)
        {
            int v = IM[setCpos[i], j];
            if (IM[v, dimension + 1] == 2)
            {
                s[v] = 1;
                for (int k = 1; k <= dimension; k++)
                    s[IM[v, k]] = -1;
                fixedB.Add(IM[v, dimension + 2]);
            }
        }
    }
}

for (int i = size_A; i >=1 ; i--)
{
        for (int j = 1; j <= dimension; j++)
        {
            int v = IM[setApos[i], j];
            if (fixedB.Contains(IM[v, dimension + 2]))
                  { fixedA.Add(IM[setApos[i], dimension+2]); }
        }

}



bool violate = false;
int ka = 1;
int numofvio = 0;
int limofvio = size_A - fixedA.Count;
int degree = dimension;

//Find independent dominating set in the bipartite BA
while (limofvio > 0 && limofvio > numofvio && degree >= 2)
    {
        numofvio = 0;

        for (int j = size_A; j >=1 ; j--)
        {

            violate = false;
            if (!fixedA.Contains(setA[j]) && s[setApos[j]] == 0)
            {
                if (deg[setApos[j]] == degree)
                {
                    for (int k = 1; k <= dimension; k++)
                    {
                        int v = IM[setApos[j], k];
                        if (s[v] == 0)
                        {
                            for (int l = 1; l <= dimension; l++)
                            {
           if (IM[IM[v, l], dimension + 1] == 1 && deg[IM[v, l]] == 1)
                                {
                                    violate = true; break;
                                }
                            }
```

```
                                    }
                                    if (violate)
                                     break;
                                }
                                if (!violate)
                                {
                                    IDSa[ka] = setA[j];
                                    IDSapos[ka] = setApos[j];
                                    ka++;
                                    limofvio--;
                                    s[setApos[j]] = 1;
                                    for (int k = 1; k <= dimension; k++)
                                    {
                                        int v = IM[setApos[j], k];
                                        if (s[v] == -1) continue;

                                        s[v] = -1;
                                        idsblength--;

                                        for (int l = 1; l <= dimension; l++)
                        { if (IM[IM[v, l], dimension + 1] == 1) deg[IM[v, l]]--; }
                                    }
                                }
                                else
                                {
                                    numofvio++;
                                }
                            }
                        }
                    }
                    degree--;
                }

            //covering vertices from a that caused violation ans that of degree 1
            for (int i = 1; i <= size_B; i++)
            {
                if (s[setBpos[i]] == 0)
                {
                    s[setBpos[i]] = 1;
                    for (int j = 1; j <= dimension; j++)
                        s[IM[setBpos[i], j]] = -1;
                }
            }

            return ka - 1 + isclength + isdlength + idsblength;

        }


        public static int BuildOddGraph(int d)
        {   //Building Odd graph
            int i, j, k, z, count;

            int w = d - 1; // # of 1s in a bits vectors

            // n = 2 to power 2d-1

            int n =  1 << (2 * d - 1);
            int n2 = n;

            //  vertex ids originally going from 1 to n

            cw = new int[n + 1];

            for (i = 1; i <= n; i++) cw[i] = i;

            k = 0;
            for (i = 1; i <= n; i++)
            {   count = 0;
                z = cw[i];
                while (z != 0)
                { count = count + (z % 2); z = z / 2; }
                if (count == w)
                { k++; cw[k] = cw[i];  }
            }

            n = k;

            Console.WriteLine("vertex count:" + n);
            int maxdeg = d;

            IM = new int[n + 1, maxdeg + 3];

            for (i = 1; i < n; i++)
            {
                for (j = i + 1; j <= n; j++)
                { // find intersection of cw[i] and cw[j]
                    z = cw[i] & cw[j];
                    if (z == 0)
                    {
                        IM[i, 0]++; IM[i, IM[i, 0]] = j;
                        IM[j, 0]++; IM[j, IM[j, 0]] = i;
                    }
                }
            }
            size_B = 0;
            size_A = 0;
            for (i = 1; i <= n; i++)
            {
                if ((cw[i] & 2) == 2 && (cw[i] & 1) != 1)
                { size_B++; IM[i, maxdeg + 1] = 3; IM[i, maxdeg + 2] = cw[i]; }
                if ((cw[i] & 1) == 1 && (cw[i] & 2) != 2)
                { IM[i, maxdeg + 1] = 4; IM[i, maxdeg + 2] = cw[i]; }
                if ((cw[i] & 3) == 3)
                { size_A++; IM[i, maxdeg + 1] = 1; IM[i, maxdeg + 2] = cw[i]; }
                if ((cw[i] & (n2 - 4)) == cw[i])
                {IM[i, maxdeg + 1] = 2; IM[i, maxdeg + 2] = cw[i]; }
            }

            return n;
        }

    }

    }
```

REFERENCES

[Arnb91] S. Arnborg, J. Lagergren, D. Seese. “Easy problems for tree-decomposable graphs”. J. Algorithms 12, pp. 308–340, 1991.

[Assm74] E. F. Assmus, M. T. Hermw. “Non-existence of Steiner systems of type S(d - 1, 2d)”. Math. Z. 138: 171-172, 1974.

[Atal88] M. Atallah, G. Manacher, J. Urrutia. “Finding a minimum independent dominating set in a permutation graph”. Discrete Applied Mathematics, v.21 n.3, pp.177-183, 1988.

[Bala72] A. T. Balaban. “Chemical graphs”. part XIII, combinatorial patterns. Rev. Roumain Math. Pures Appl., 17: 3-16, 1972.

[Bala06] B. Balasundaram, S. Butenko. "Graph domination, coloring and cliques in telecommunications". Handbook of Optimization in Telecommunications, pp. 865-890, 2006.

[Bang99] J. Bang–Jensen, J. Huang, G. Macgillivray, A. Yeo. “Domination in convex bipartite and convex–round graphs”. Manuscript, 1999.

[Bigg72] N. L. Biggs. “An edge-colouring problem”. Am. Math. Mon. 79: 1018-1020, 1972.

[Bigg74] N. L Biggs. “Algebraic Graph Theory”. Cambridge University Press, London. 1974.

[Bigg79] N. L. Biggs. “Some Odd Graph Theory”. Annals of New York Academy of Sciences, vol. 319, pp. 71-81, 1979.

[Booi07] O. Booij, Z. Zivkovic, B. Krose. "Image based navigation using a topological map". Proceedings of the 13th Annual Conference of the Advanced School for Computing and Imaging, Netherlands, pp. 1-8, June 2007.

[Bour10] M. Bourgeois, B. Escoffier, V. Th. Paschos. “Fast Algorithms for min independent dominating set”. Proceedings of SIROCCO 2010, LNCS 6058, pp. 247–261, 2010.

[Bran98] A. Brandsta, V. Chepoi, F. Dragan. “The algorithmic use of hyper tree structure and maximum neighborhood orderings” Discr. Appl. Math. 82(1-3), pp. 43-77, 1998.

[Bro97] H. Broersma, T. Kloks, D. Kratsch, H. Muller. “Independent sets in asteroidal triple–free graphs”. Proceedings of ICALP’97, Springer, LNCS 1256, pp. 760–770, 1997.

[Buen09] L.R. Bueno, L. Faria, C.M.H. Figueiredo, G.D. Fonseca. “Hamiltonian paths in odd graphs”. Appl. Anal. Discrete Math., 3, pp. 386–394, 2009.

[Byer77] T. Bayer, A. Proskurowski, S. Hedetniemi, S. Mitchell. "Independent domination in trees". Prec. 8th Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, WinnipeB, pp. 321-328, 1977.

[Chan98a] G.J. Chang. "Algorithmic aspects of domination in graphs". in D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial optimization 3, pp. 339-405, 1998.

[Chan98b] M.S. Chang. “Efficient algorithms for the domination problems on interval and circular–arc graphs”. SIAM J. Comp. 27, pp. 1671-1694, 1998.

[Chen02] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris. “Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks”. ACM Wireless Networks Journal, 8(5), 2002.

[Cock77] E. J. Cockayne, S. T. Hedetniemi. “Towards a theory of domination in graphs”. Networks 7, pp. 247-261, 1977.

[Conw77] J. H. Conway, N. L. Biggs. Symmetric circuits. Preprint, 1977.

[Corn84] D.G. Corneil, Y. Perl. “Clustering and domination in perfect graphs”. Discr. Appl. Math. 9, pp. 27-39, 1984.

[Dama90] P. Damaschke, H. Muller, D. Kratsch. “Domination in convex and chordal bipartite graphs”. Inform. Proces. Lett. 36, pp. 231-236, 1990.

[Duck02] W. Duckworth, N.C. Wormald. “Minimum independent dominating sets of random cubic graphs”. Random Structures and Algorithms 21, pp. 147-161, 2002.

[Duck10] W. Duckworth, N. Wormald. “Linear programming and the worst-case analysis of greedy algorithms on cubic graphs”. Electron J Comb 17:#R177, 28 pp, 2010.

[Elma90] E.S. Elmallah, L.K. Stewart. “Domination in polygon graphs”. Congr. Numer. 77, pp. 63-76, 1990.

[Erdo61] P. Erdos, Ko. Chao, R. Rado. “Intersection theorems for systems of finite sets”. Q. J. Math. (Oxford) 12: 313-320, 1961.

[Fabe82] M. Farber. “Independent domination in chordal graphs”. Oper. Res. Lett. 1, pp. 134-138, 1982.

[Fabe84] M. Farber. “Domination, independent domination, and duality in strongly chordal graphs”. Discr. Appl. Math. 7, pp. 115-130, 1984.

[Fabe89] M. Farber. “On diameters and radii of bridged graphs”. Discrete Math. 73, pp. 249-260, 1989.

[Gare79] M. R. Garey, D. S. Johnson. “Computers and intractability. A guide to the theory of NP-completeness”. W. H. Freeman, San Francisco, 1979.

[Gasp06] S. Gaspers, M. Liedloff. “A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs”. In F. V. Fomin, editor, Proc. International Workshop on Graph Theoretical Concepts in Computer Science, WG’06, volume 4271 of LNCS, pp. 78-89. Springer-Verlag, 2006.

[Ghaf91] A. Ghafoor and T.R. Bashkow. “A Study of Odd Graphs as Fault-Tolerant Interconnection Networks”. IEEE Trans. Computers, vol. 40, no. 2, pp. 225-232, 1991.

[Hayn06] T.W. Haynes, D. Knisley, E. Seier, Y. Zou. "A quantitative analysis of secondary RNA structure using domination based parameters on trees". BMC Bioinformatics, 7 (1), pp. 108, 2006.

[Hayn97] T. W. Haynes, S. T. Hedetniemi, P. J. Slater. “Fundamentals of Domination in Graphs”. Marcel Dekker, Inc., New York, 1997.

[Hemp99] H. Hempel, D. Kratsch. “On claw–free asteroidal triple–free graphs”. Manuscript, 1999.

[Hilt67] A. J. W. Hilton, E. C. Milner. “Some intersection theorems for systems of finite sets”. Q. J. Math. (Oxford) 18: 369-384, 1967.

[Huri08] J.L. Hurink, T. Nieberg. “Approximating Minimum Independent Dominating Sets in Wireless Networks”. Inform. Process. Lett. 109, pp. 155-160, 2008.

[Krat93] D. Kratsch, L. Stewart. “Domination on cocomparability graphs”. SIAM J. Discrete Math. 6(3), pp. 400-417, 1993.

[Kuo88] S.Y. Kuo, W.K. Fuchs. "Spare allocation and reconfiguration in large area VLSI". Proc. 25th ACM/IEEE Design Automation Conference, pp. 609-612, 1988.

[Lov´a70] L. Lov´asz. Problem 11. In Combinatorial Structures and their Applications. Gordon and Breach, 1970.

[Liu06] C. Liu, Y. Song. “Exact Algorithms for Finding the Minimum Independent Dominating Set in Graphs”. LNCS, Volume 4288, pp. 439-448, 2006.

[Math76] M. Mather. “The Rugby footballers of Croam”. J. Comb. Theory (B) 20: 62-63, 1976.

[Mere72] G. H. J. Meredith, E. K. Lloyd. “The Hamiltonian graphs O4 to O7”. In Combinatorics (Proc. Conf. Combinatorial Math., Math. Inst., Oxford, 1972), pp. 229-236. Inst. Math. Appl., Southend, 1972.

[Mere73] G. H. J. Meredith, E. K. Lloyd. “The footballers of Croam”. J. Comb. Theory (B) 15: 161-166, 1973.

[Moon65] J. W. Moon, L. Moser. “On cliques in graphs”. Israel J. Math., **3**, pp. 23-28, 1965.

[Ni99] S.Y. Ni, Y.C. Tseng, Y.S. Chen, J.P. Sheu. “The broadcast storm problem in a mobile, ad-hoc network”. In Proc. MOBICOM, pp. 151-162, 1999.

[Pfaf84] J. Pfaff, R. Laskar, S.T. Hedetniemi. “Linear algorithms for independent domination and total domination in series–parallel graphs”. Technical Report 441, Clemson University, 1984.

[Prie01] C.E. Priebe, J.G. DeVinney, and D.J. Marchette. "On the distribution of the domination number for random class cover catch digraphs". Statistics and Probability Letters, 55: 239-246, 2001.

[Rand04] B. Randerath, I. Schiermeyer. “Exact algorithms for Minimum Dominating Set”. Technical Report zaik-469, Zentrum fur Angewandte Informatik, Koln, Germany, April 2004.

[Shie04] I. Shields, C. D. Savage. “A note on Hamilton cycles in Kneser graphs”. Bull. Inst. Combin. Appl., 40: 13-22, 2004.

[Shie99] I. Shields, C. D. Savage. “A Hamilton path heuristic with applications to the middle two levels problem”. In Proceedings of the Thirtieth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Boca Raton, FL, 1999). Congr. Numer., volume 140, pp. 161-178, 1999.

[Shie09] I. Shields, B. J. Shields, C. D. Savage. “An update on the middle levels problem". Discrete Math., 309, pp. 5271-5277, 2009.

[Vizi64] V. G. Vizing. "On an estimate of the chromatic class of a p-graph". Diskret. Analiz. 3: 25-30, 1964.

[Yann80] M. Yannakakis, F. Gavril. “Edge dominating sets in graphs”. SIAM J. Appl. Math.38(3), pp. 364-372, 1980.

[Zeli85] B. Zelinka. “Odd graphs”. Arch. Math., Brno 21, pp. 181-187, 1985.

[Zver95] I. Zverovich, V. Zverovich. “An induced subgraph characterization of domination perfect graphs”. Journal of Graph Theory 20 (3), pp. 375-395, 1995.

VITA

Name:	Ahmed Ibrahim Al-Herz
Nationality:	Saudi
Address:	P.O. Box: 766, KFUPM, Dhahran 31261, Saudi Arabia
Email Address:	a.alherz@yahoo.com
Phone:	+96638602287

Ahmed Al-Herz was born on April 24, 1982 in USA. He earned his Bachelor of Science degree in Computer Engineering in June 2005 from King Fahd University of Petroleum & Minerals (KFUPM). Al-Herz completed his Master of Science degree in Computer Science in February 2012 from KFUPM.

Ahmed Al-Herz is currently a graduate assistant in Information and Computer Science Department at KFUPM.