FINDING MAXIMUM INDEPENDENT SET IN A GRAPH USING CELLULAR AUTOMATA WITH HEURISTIC ALGORITHMS

By
Naif A. Al-Sukhni

Supervisor
Dr. Ahmed A. Sharieh

This Thesis was submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Science

Faculty of Graduate Studies
The University of Jordan
May, 2008

This Thesis (Finding Maximum Independent Set in a Graph using Cellular Automata with Heuristic Algorithms) was Successfully Defended and Approved on 15/5/2008.

Examination Committee	**Signature**
Dr. Ahmad Sharieh (Supervisor) Assoc. Prof. in Parallel Processing	----------------------------
Dr. Mohammed Qatawneh (Member) Assoc. Prof. in Parallel Systems and Networks	----------------------------
Dr. Abdel Latif Abu Dalhoum (Member) Assoc. Prof. in genetic algorithms and Complex Systems	----------------------------
Prof. Saleh Oqeili (Member) Prof. in Computer Engineering (Hashemite University -Vice President)	----------------------------

COMMITTEE DECISION

This Thesis (Finding Maximum Independent Set in a Graph using Cellular Automata with Heuristic Algorithms) was Successfully Defended and Approved on 15/5/2008.

Examination Committee	**Signature**
Dr. Ahmad Sharieh (Supervisor) Assoc. Prof. in Parallel Processing	------------------------------
Dr. Mohammed Qatawneh (Member) Assoc. Prof. in Parallel Systems and Networks	------------------------------
Dr. Abdel Latif Abu Dalhoum (Member) Assoc. Prof. in genetic algorithms and Complex Systems	------------------------------
Prof. Saleh Oqeili (Member) Prof. in Computer Engineering (Hashemite University -Vice President)	------------------------------

DEDICATION

First of all, Thanks GOD for accomplishing this Thesis.

I am grateful to my parents, my sister, my brothers and my wife for their moral support and encouragement.

ACKNOWLEDGEMENT

My deepest gratitude goes to my supervisor Dr. Ahmad Shariah for his effort, support, advice, patience and help to accomplish this thesis.

I would like also to thank all my instructors at my collage for their great effort, support, advice, and help.

List of contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Description
IS	Independent Set
MIS	Maximum Independent Set
CA	Cellular Automaton
HCA	Heuristic Cellular Automata
CPU	Central processing Unit
SNMD	Select Node with Minimum Degree
SNMDN	Select the Node with Minimum Degree among its Neighbors
SNXD	Select Node with Maximum Degree
SNXDN	Select the Node with Maximum Degree among its Neighbors
SNMXD	Select Node with Minimum or Maximum Degree
SISD	Single Instruction, Single Data
MISD	Multiple Instruction stream, Single Data
SIMD	Single Instruction, Multiple Data
MIMD	Multiple Instruction stream, Multiple Data stream
RAM	Random Access Memory
M. Wilf's	Modified Wilf
CA-MIS	Cellular Automaton - Maximum Independent Set
D	Graph Density
N	Graph Size
E	Graph Edges
M	Number of Threads
S	Speedup
S_p	Parallel Speedup
E	Efficiency
E_P	Parallel Efficiency
T_1	Sequential running time
T_p	Parallel running time

T_{Wilf}	Wilf Running time
$T_{M.Wilf}$	Modified Wilf Running time
T_{CA}	Cellular Automata Running time
R_t	The ratio of optimization on the running time
A	Accuracy
UN	UnKnown

List of Appendices

FINDING MAXIMUM INDEPENDENT SET IN A GRAPH USING CELLULAR AUTOMATA WITH HEURISTIC ALGORITHMS

By
Naif A. Al-Sukhni

Supervisor
Dr. Ahmed A. Sharieh

ABSTRACT

This Thesis introduces four versions of a heuristic algorithm based on cellular automata (HCA) to find a Maximum Independent Set (MIS) in a given graph. Finding a MIS in a given graph is one of the fundamental problems in the combinatorial optimization. An Independent Set (IS) in a graph is a set of vertices that are mutually non-adjacent. The IS problem is finding a maximum size independent set in a given graph. The exact algorithms to find a MIS are limited to handle only graphs of small sizes. The proposed algorithms (HCA) were implemented on machine with single processor using single thread and multithreads and handle graphs of large sizes.

The HCA algorithms use cellular automaton based on weighted factors. They choose a candidate node to be in a MIS in several methods. The two versions (single and multithreads) of the algorithms make the selection based picking: a anode randomly, a node with maximum degree, or a node with minimum degree. The HCA algorithms were analyzed, implemented, and tested. The performance of parallel implementation, using multithreads, was measured on PC under Windows XP operating system. The effects of

number of nodes, number of edges, and the degree of nodes in a graph on the efficiency of the HCA algorithms were investigated.

The Theoretical results indicates that the complexity of the algorithm is $O(n^{3.8})$ and it's not going be worst than $O(n^4)$.

The results indicate that the heuristic algorithms produce MISs with sizes close to the MISs produced by exact algorithms such as Modified Wilf algorithm. For example, about 75% of the sizes of the MIS produced by the Minimum Degree algorithm have the same size as those produced by the exact algorithm. The Minimum degree factor algorithm is the most accurate algorithm (97.35%) in general, followed by the Random Degree algorithm in the second level (95.25%), and finally the Maximum degree factor algorithm (89.44 %). The HCA algorithms have less CPU run time than Wilf's and modified Wilf's. When we changed a graph (with N nodes) and with fixed density (D), we found that the size of MIS (i(G)) is increased as N increased. The sizes of MISs increase as the density of the graph decrease. When density increases, the sizes of MISs become the same regardless the value of N. For example, when D = 0.9, the i(G) of a graph with N = 100 nodes is almost equal to the i(G) of a graph with N=1000. The experimental results indicated that for a fixed density the sizes of the generated MISs are very close. For example, the sizes of the generated MISs for density = 0.9 remain five nodes as the sizes change from 400 through 1000. The parallel implementation of the HCA algorithm using Multithreading does not appear to be great performance over the sequential one for the suggested samples, and it could be more suitable for graphs with large size (ex. 10,000) and on multiprocessors or multi-computers environment. In conclusion, the contributions of this research are introduction of four heuristic algorithms, measurement of their performances, and opening new ideas for parallel implementations based on cellular automata to solve the problem of finding a MIS.

Chapter 1
Introduction

This Thesis introduces a new Heuristic Cellular Automata (HCA) algorithms based on Weighted Factors for finding a MIS in a Graph. The algorithms will use cellular automata and heuristic approaches by choosing the candidate nodes in MIS: randomly, node with maximum degree, and node with minimum degree. In order to speedup the process of finding MIS and to find all possible MIS's from different starting points, parallel implementations of the sequential algorithms using Multithreads was investigated.

A cellular automaton (CA) is a collection of cells arranged in a grid, such that each cell changes state as a function of time according to a defined set of rules that includes the states of neighboring cells. In other words, all the cells go from their current state to some next state according to the “local rule” (Niesche H, 2006).

An independent set (IS) is a set of vertices in a graph no two of which are adjacent. A maximum independent set (MIS) is a largest independent set for a given graph. A graph G = (V, E) is a collection of vertices V and edges E. A subset S of V is an IS of G if no two vertices of S are adjacent. A MIS of a graph G is an IS of maximum size and is denoted by $I(G)$; its cardinality (denoted by $i(G)$) is greater than or equal to the cardinality of any other IS (en.wikipedia.org ,18).

1.1.1 MIS Applications

The MIS problem has applications in many fields such as: information retrieval, signal transmission analysis, classification theory, economics, scheduling, biomedical engineering (butenko, 2003), graph coloring, job scheduling, optimization problems, parallel processing and pattern recognition (Al-Jaber and Sharieh, 2000). Following are three of some of the MIS applications.

1.1.2 Time Tabling and Scheduling

If we have two classes at a university taught by the same instructor or two courses required by the same group of students, we can't schedule these courses for the same time slot .The problem of finding the minimum number of time slots needed with these restrictions can be modeled by the graph coloring problem and solved by using MIS (Gebremedhin, 1999).

1.1.3 Wireless networking

Gaining an efficient routing and solving fail over and mobility problems of the wireless ad hoc and sensor networks could be achieved using clustering which is one of the most important network organization techniques .The clustering induced by a MIS has been shown to exhibit particularly desirable properties (Kuhn, et al., 2005).

1.1.4 Frequency Assignment

In the process of assigning frequencies to the mobile radios, companies must take into consideration assigning different frequency to the customers that are sufficiently close while those that are distant can share frequencies. A MIS can solve the problem of minimizing the number of frequencies (Gebremedhin, 1999).

.1.2 Problem Definition

If G is a graph and $S \subseteq V(G)$, then S is an IS of vertices of G if no two of the vertices in S are adjacent in G. An independent set S is maximal if it is not a proper subset of another independent set of vertices of G. The independent set problem is that of finding a maximum size independent set in a given graph (en.wikipedia.org ,18 & 19).

This problem is NP-Complete (Dharwadker, 2006; Kako, 2004). Thus, it is strongly predicted that no polynomial time algorithm can find optimal solution. So, approximation or heuristic algorithms and parallel implementation are important to have polynomial run time that can find solutions closed to the optimal one (Kako, 2004; Back and Khuri, 1994).

1.2 Methodology

The following is the methodology that was used in this study

1. MIS Problem and some of the algorithms of this problem (exact, approximation, sequential and parallel) were studied.
2. The process of developing an approximation sequential cellular algorithm based on weighted factors for finding a MIS in a graph was introduced.
3. Choosing the next node during the process (randomly, node with minimum degree, or node with maximum degree) and how this will affect the accuracy of the algorithm and its effects on the performance of the algorithm were investigated.
4. A Java program to implement the proposed algorithms (sequentially) was written.
5. The proposed algorithm was tested and its performance was measured.
6. A Java program to implement the proposed algorithm (parallel) using multithreading environment was written.
7. The results of the proposed algorithms and the modified Wilf's algorithm that produce the exact MIS's were compared.
8. Testing and comparing the serial and parallel algorithms and reporting their performance were performed. These were tested on graphs with different densities and sizes generated randomly, using different number of threads

9. The effect of number of nodes, number of edges, and the degree of nodes in a graph on the efficiency were measured and reported.

10. 1.4 Related Studies

This Section reviews the most recent previous studies to solve the problem of finding a MIS in a graph.

In (Czumaj, Diks and Przytycka, 1998), parallel maximum independent set algorithm for special class of graphs called convex bipartite graphs was proposed. A bipartite graphs G = (V, W, E), where V and W are sets of vertices, is called convex if the vertices in W can be ordered in such a way that the elements of W adjacent to any vertex v ∈ V from an interval (Czumaj, Diks and Przytycka, 1998). The problem is reduced to finding all the vertices in G which are reachable from unmatched vertices in V. The algorithm runs in O (log n) time with (n / log n) processor on a Concurrent Read and Concurrent Write Parallel Random Access Machine (CRCW PRAM), where n is the size of the input graph.

In (Al-Jaber and Sharieh, 2000), five approximation sequential algorithms, based on selecting a vertex with a specific degree or the degree of its neighbors (weights of nodes), are introduced and compared with the Wilf's and modified Wilf's algorithm that produce the exact MIS's. The five approximation sequential algorithms are:

1. Select Node with Minimum Degree (SNMD) Algorithm.
2. Choose a Node and Select the Node with Minimum Degree among its Neighbors (SNMDN) Algorithm.
3. Select Node with Maximum Degree (SNXD) Algorithm.
4. Choose a Node and Select the Node with Maximum Degree among its Neighbors (SNXDN) Algorithm.

5. Select Node with Minimum or Maximum Degree (SNMXD) Algorithm.

In (Al-Ghwari , 2006) a parallel implementation of the five heuristic algorithms that introduced in (Al-Jaber and Sharieh, 2000), was introduced .The parallel implementation gives better results than the sequential implementation for these algorithms.

In (Grama, et al., 2003), the main elements of a Parallel Algorithm were discussed and could be summarized as the following:

a) Parts of the program or problem that can run concurrently (tasks).
b) Processes vs. Processors where you map the tasks onto multiple processes or processors according to the platform that has been used (LAN, Threads,).
c) Distribution data across the different processes or processors.
d) Management of shared data and Synchronization of the processors

In (Kako, 2004), the importance of finding approximate solution for the weighted IS problem (each vertex has a weight), was discussed. Since no polynomial time algorithm can find optimal solution. Approximation algorithms are evaluated on approximation ratio (the ratio between the weight of an optimal solution and the weight of an approximate solution). A good algorithm is the one with approximation ratio close to 1.

In (Niesche, 2006), routing using cellular automata was introduced. The idea is about finding an optimal path between two points on a regular grid (see Figure 1.1) and the following rules were suggested:

1. Using Von Neumann Neighborhood
2. Target is state "1", immutable.
3. All other cells are mutable where new state is : New state = min (neighbors) +1

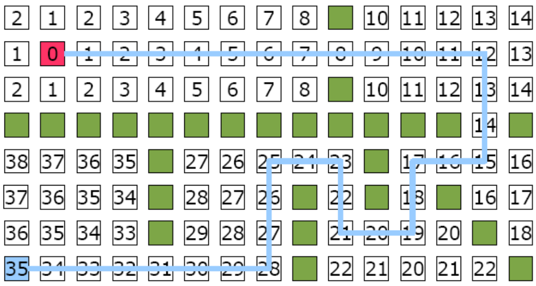


Figure 1.1: Routing using cellular automata (Niesche , 2006)

In (Akhter and Roberts , 2007), the most important step on the process of converting the sequential algorithm to a parallel algorithm (parallelization or Parallel Formulation) is identifying those activities that can be executed in parallel according to the dependencies between tasks.. This may be achieved in several ways by task, by data, or by data flow. Table 1 below summarizes these forms of decomposition (Akhter and Roberts, 2007).

Table 1.1: forms of decomposition (Akhter and Roberts, 2007).

Decomposition	Design	Comments
Task	Different activates assigned to different threads	Common in GUI applications
Data	Multiple Threads performing the same operation but on different blocks of data	Common in audio processing, imaging and in scientific programming
Data Flow	One Thread's output is the input to a second thread	Special case is needed to eliminate startup and shutdown latencies

In (Gfeller and Vicari , 2007) , a randomized distributed algorithm for the maximal independent set problem for special class of graphs called growth-bounded graphs was proposed that computes a MIS in O(log log n log* n) (Gfeller and Vicari , 2007).

This thesis introduces a new technique to find MIS by using CA, which may be a starting point to use CA to solve other related problems or using other related techniques to find MIS.

1.5 Thesis Structure

This thesis includes the following chapters:

Chapter 1: Introduction.

It introduces the Study Problem, Study Methodology and Related Studies.

Chapter 2: Literature Review.

It includes

- The Theoretic Definitions, Notations, and possible approaches for solving the MIS problem.
- Examples of the Exact, Approximation and Heuristic Approaches for Solving MIS Problem.
- An overview about Multithreading Environment and Multiprocessing Environment.

Chapter 3: The Proposed Heuristic Cellular Automata Algorithms based on Weighted Factors

It introduces the Heuristic approach for solving the MIS problem using Cellular Automata Algorithms based on Weighted Factors (Maximum Degree Node, Minimum Degree Node and Random Node)

Chapter 4: Experimental Results and Discussion

It introduces the experiments results of the proposed algorithms and compares the results with the exact algorithms (Modified Wilf).

Chapter 5: Conclusions and Future Work.

Appendices: Segments source codes of the proposed algorithms

Chapter 2
Literature Review

The Objective of this chapter is to define, introduce and details the main concepts that will be used in this thesis. In addition it reviews some of the exact algorithm to find the MIS, multithreading environment, and heuristic approaches to find MIS in a graph.

2.1 Theoretic Definitions and Notations

2.1.1 Graph:
A graph G = (V, E) is a collection of vertices V and edges E (en.wikipedia.org ,15), such that:

- V is a set, whose elements are called vertices or nodes.
- E is a set of pairs (unordered, for the undirected graph) of distinct vertices, called edges or lines.
- The order of a graph is | V | (the number of vertices). A graph's size is | E |, the number of edges. The degree of a vertex is the number of other vertices it is connected to by edges.
- The vertices belonging to an edge are called the ends, endpoints, or end vertices of the edge.
- Two edges of a graph are called adjacent if they share a common vertex.
- In a weighted graph or digraph, each edge is associated with some value, variously called its cost, weight, length or other term depending on the application. Such graphs arise in many contexts. For example in optimal routing problems such as the traveling salesman problem.

2.1.2 **An independent set (IS):** is a set of vertices in a graph where no two of which are adjacent. A maximum independent set (MIS) is a largest independent set for a given graph.

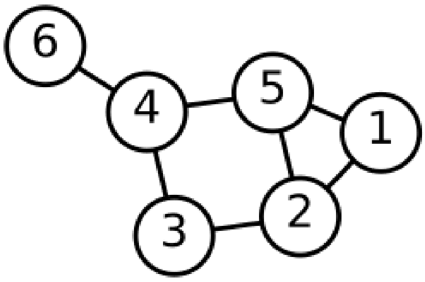


Figure 2.1: A graph with 6 nodes and 7 edges

For example, Figure 2.1 shows a graph with V = {1, 2, 3, 4, 5, 6} and

E = {{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}.

The sets: {4, 1} and {1, 3, 6}, for example, are ISs. The graph contains more than one MIS, such as {1, 3, 6} and {6, 5, 3}.

2.1.3 **Algorithms**

Informally, an algorithm is any well-defined computational procedure that takes input values and produces output values. An algorithm is thus a sequence of computational steps that transform the input into the output. (Cormen, Leiserson, Rivest and Stein, 2001). Another Definition of an algorithm is a tool for solving a well-specified computational problem.

2.1.4 **A Cellular Automaton (CA)** is a collection of cells arranged in a grid, such that each cell changes state as a function of time according to a defined set of rules that includes the states of neighboring cells. In other words, all the cells go from their current state to some next state according to the "local rule" (Niesche H, 2006).A CA is a collection of "colored" cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules based on the states of neighboring cells (Weisstein, 2008). CA can be in a variety of shapes like grid (one dimensional line, two dimensions, square, triangular, and hexagonal grids) as in Figure 2.2

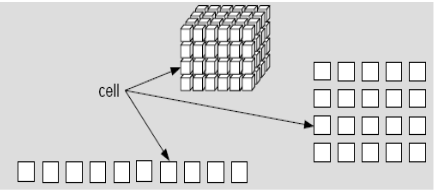


Figure 2.2: CA- Shape Types (Niesche , 2006)

The simplest type of cellular automaton is a binary, nearest-neighbor, one-dimensional automaton which called "elementary cellular automata. There are 256 such automata, each of which can be indexed by a unique binary number whose decimal representation is known as the "rule" for the particular automaton. Figure 2.3 is an illustration of rule 30..

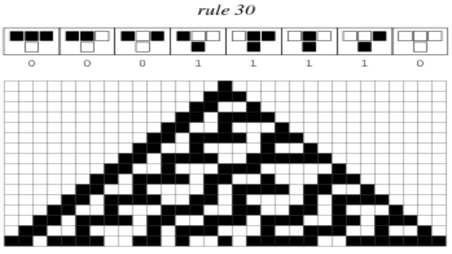


Figure 2.3: CA- Rule 30

In two dimensions, the possible neighborhood could be von Neumann neighborhood (nodes in horizontal and vertical) or Moore neighborhood (nodes in horizontal, vertical and diagonal), see Figure 2.4.

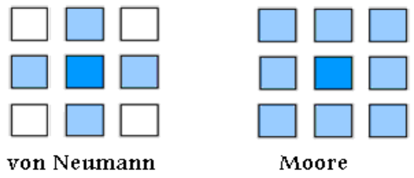


Figure 2.4: CA- Neighborhood Types (Niesche , 2006)

The theory of CA is immensely rich, with simple rules and structures being capable of producing a great variety of unexpected behaviors. A large number of problems can be solved if they are described as cellular algorithms. Typical such problems are physical field's lattice gas models diffusion, games, artificial worlds, image processing, pattern recognition simulation of digital circuits and graph algorithms (Hochberger & Hoffmann , 1996). One of these applications was the problem of finding a MIS for any graph.

2.2 **NP-Complete Problems**

A problem can be posed in a decision problems or optimization problems.

- Optimization problems: In these problems, we are looking to find the feasible solution with the best value. For example, shortest path in network (en.wikipedia.org ,8).
- Decision problems: in which the answer is simply "yes" or "no"? The subset-sum problem (Given a set of integers, does some nonempty subset of them sum to 0) is an example of a problem which is easy to verify, but whose answer is *believed* (but not proven) to be difficult to compute (en.wikipedia.org ,8).

The relation between the complexity classes P and NP is studied in computational complexity theory, the part of the theory of computation dealing with the resources required during computation to solve a given problem.

Two of the most important resources are time (how many steps it takes to solve a problem), and Space (how much memory it takes to solve a problem) (en.wikipedia.org, 8).

The class P consists of all the decision problems that can be solved in polynomial time, ($O(n^k)$, where n is the size of the input to the problem and k constant), on a deterministic sequential machine. The class NP consists of all the decision problems is a positive solutions can be verified in polynomial time, or whose solution can be found in polynomial time on a non-deterministic machine.

Definition: a problem R is NP complete if

- R is NP
- Every NP problem P reduces to R

An equivalent but casual definition: A problem R is NP-complete if R is the most difficult of all NP problems (www.seas.gwu.edu)

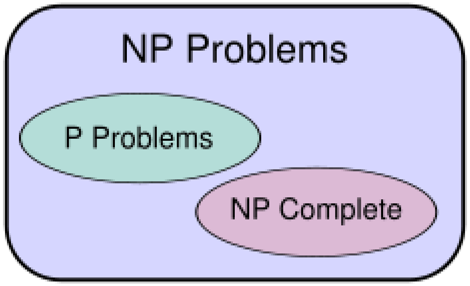


Figure 2.5: NP Problems (en.wikipedia.org,29).

A reduction is a transformation of one problem into another problem. If problem A is reducible to problem B, a solution to B gives a solution to A (en.wikipedia.org, 30).

In 1972, Karp introduced a list of twenty-one NP-complete problems. One of these problems was the problem of finding a maximum independent set in a graph. For example, try to find a MIS with five vertices in a graph (Dharwadker, 2006).Since the decision variant is NP-complete; the optimization variant is NP-hard.

2.3 Approaches for solving MIS problem

2.3.1 Exact Approaches

Using the exact approaches to solve decision problems leads to obtain the optimal solution, but it becomes impractical and slow when the problem size increased. For example, when we have to deal with very large graphs, the exact approaches cannot be applied. In this case heuristics provide the only available option (butenko, 2003).

2.3.2 Approximation Algorithm

Approximation algorithms are algorithms used to find approximate solutions to the optimization problems. Normally when we speak about approximation algorithms, we speak about NP-hard/ NP-Complete problems since they are often associated. It is unlikely that there can ever be efficient polynomial time exact algorithms solving NP-hard problems (wikipedia.org, 4).

2.3.3 Heuristic Approaches

One approach for solving NP-hard problems is the use of heuristic (Gebremedhin, 1999). In computer science, a heuristic (usually a polynomial time algorithm) is a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem (wikipedia.org,16).

Another definition of a heuristic in optimization is any method that finds an “acceptable” feasible solution. The term heuristic has two well-defined technical meanings. Heuristic algorithms, whose general purpose is not to find an optimal solution, but an approximate solution where the time or resources are limited (en.wikipedia.org, 16).

Heuristics are intended to gain computational performance or conceptual simplicity, potentially at the cost of accuracy or precision. The goal of heuristic is to find a good solution rather than finding an optimal one (Gebremedhin, 1999).

2.3.3.1 Categories of Heuristic

There are two Categories of Heuristic: greedy and local improvement

- **Greedy Heuristic**

  It makes the choice that seems to be the best at the moment and proceeds until it find the local optimum (Gebremedhin, 1999). Thus it is fast and simple

- **Local improvement Heuristic**

  These heuristics are iteratively tried to improve the quality of a current solution for problem (Gebremedhin, 1999).

2.4 Examples of the approaches for solving the MIS Problem

Several approaches have been introduced to study the MIS, some of them are exact approach like Wilf's and Modified Wilf's, and others are approximation and heuristics.

2.4.1 Exact approaches for solving the MIS problem

In 1986 Wilf suggested an exact approach for finding MIS using a recursive technique (Al-Jaber and Sharieh, 2000). In the graph in Figure 2.1 the set {4, 1} is an IS and so {1, 3, 6}. The graph contains more than one MIS, such as {1, 3, 6} and {6, 5, 3}. The problem of finding the size of the largest independent set in a given graph is computationally very difficult (Wilf, 1994).

In this approach, all possible IS's generated and the one with maximums size is selected to be the MIS (Al-Jaber and Sharieh, 2000). Suppose we have a graph G and we are looking for the size of the largest independent set of vertices of G (denoted as maxset(G)). If an independent set S of vertices contains vertex v*, then the remaining vertices of S are an independent set in a smaller graph, namely the graph that is obtained from G by deleting vertex v* as well as all vertices that are connected to vertex v* by an edge called the neighborhood of vertex v* (Nbhd(v*)) (Wilf, 1994).

The set S consists of vertex v* together with an independent set of vertices from the graph G − {v*} − Nbhd(v*). If we name an independent set S that doesn't contain vertex v*, then the set S is simply an independent set in the smaller graph G − {v*} (Wilf, 1994).

Suppose there are two numbers maxset(G − {v*}) and maxset(G − v*}− Nbhd(v*)) , then we have the following relation (Wilf, 1994).

$$\text{maxset}(G) = \max\{\text{maxset}(G-\{v^*\}), 1+ \text{maxset}(G-\{v^*\}-\text{Nbhd}(v^*))\}$$

Based on that, Wilf proposed the algorithm in Figure 2.6

```
function maxset1(G);
{return the size of the largest Independent set of vertices of G}
If G has no edges
   Then maxset1:=|V(G)|
Else
  Choose some nonisolated vertex v* of G;
  n1:=   maxset1(G-{v*});
  n2:=   maxset1(G-{v*}-Nbhd(v*))}  ;
  maxset1:=max (n1,1+n2)
end. {maxset1}
```

Figure 2.6: Recursive Algorithm for finding MIS (Wilf, 1994).

In the worst case, the complexity will be $\Theta(n^2)$. If G, actually, has some edges, the additional labor needed to process G consists of two recursive calls on smaller graphs and one computation of the larger of two numbers. If F(G) denotes the total amount of computational to find maxset1(G), then relation (1) is true (Wilf, 1994).

$$F(G) <= cn^2 + F(G-\{v^*\}) + F(G-\{v^*\}-Nbhd(v^*)). \qquad (1)$$

If $f(n) = \max_{|V(G)|=n} F(G)$ and the maximum of (1) is taken over all graphs G of n vertices, then the result is that as in the relation (2) (Wilf, 1994).

$$f(n) <= cn^2+f(n-1)+f(n-2) \qquad (2)$$

Because the graph G− {v*} − Nbhd(v*) might have as many as n − 2 vertices, and would have that many if v* had exactly one neighbor. If we solve the recurrent inequality (2), then relation (3) can be derived (Wilf, 1994).

$$f(n) = O((1.619^n) \qquad (3)$$

Another algorithm is modified Wilf's. It takes into consideration the maximum degree of the node as weight in order to select the node to be added to MIS. As the maximum degree of the selected node increases, the number of generated sub graph decreases and the running time of the algorithm decreases (Al-Jaber and Sharieh, 2000).

TABARIS (Tabu and Bound Applied Repeatedly for IS) is another exact approach for finding MIS by using Tabu search (Al-Jaber and Sharieh, 2000),

By using the exact approaches to solve MIS problem, an optimal solution can be obtained, but it become impractical and slow when the number of vertices increased, even on graphs with several hundreds of vertices. Therefore, when it comes to very large graphs, the exact approaches cannot be applied, and heuristics provide the only available option (butenko, 2003).

2.4.2 Heuristic approaches for solving the MIS problem

2.4.2.1Greedy IS (Gebremedhin, 1999).

In this algorithm:

- *U* denotes the set of vertices under consideration
- G' denotes the graph induced by *U*.
- The heuristic returns the IS denoted by *I*.

Initially U=V, I={} and G' =G, arbitrarily a vertex v is chosen from U and added to *I* ,then vertex v and its neighbors *N(v)* are removed from *U*. The process repeated while U is not empty.

```
Greedy Independent Set ( G )
  begin
      I  = {
}                                                   U  =
V
     G' =  G
      While ( G' ≠  ¢  ) do
            Choose a vertex v arbitrarily from
U                                 I = I ∪
{v}                                                  X =
{v} ∪ N ( v)
                  U = U – X
                  G' = graph induced by U
      end – while
      return I
end
```

Figure 2.7: Greedy IS (Gebremedhin, 1999).

2.4.2.2 Minimum Degree IS

In the previous algorithm instead of picking an arbitrary vertex, a vertex of low degree in the graph G'. Chosen the minimum degree vertex leads to a minimum size of vertices to be removed (*N(v)*) , and the process of finding IS continue for longer time resulting in a large size IS.

2.4.3 Approximate approaches for solving the MIS problem

In (Al-Jaber and Sharieh, 2000), five approximation sequential algorithms, based on selecting a vertex with a specific degree or the degree of its neighbors (weights of nodes), are introduced. The five approximation sequential algorithms are:

1. Select Node with Minimum Degree (SNMD) Algorithm.
2. Choose a Node and Select the Node with Minimum Degree among its Neighbors (SNMDN) Algorithm.
3. Select Node with Maximum Degree (SNXD) Algorithm.
4. Choose a Node and Select the Node with Maximum Degree among its Neighbors (SNXDN) Algorithm.
5. Select Node with Minimum or Maximum Degree (SNMXD) Algorithm.

According to computational results in (Al-Jaber and Sharieh, 2000), the heuristic algorithms produce closer sizes of MIS for smaller density of graphs and produce a good approximation in short time, the SNMDN algorithm has better results in term of approximating the IS’s than the others and in reference to speedup over Wilf's and modified Wilf's, the SNMD is the fastest and the SNXDN is the worst.

2.5 Multithreading and Parallel Processing

Finding the MIS using sequential algorithms, especially, exact algorithms are costly in computation resources (time and cost).As we described before the main Objectives of this thesis are:

1. Introduce a new heuristic approach for solving MIS problem
2. Explore how we can solve MIS problem using multithreading environment.

In this chapter we are going to discuss the main concepts in parallel computing (parallel processing) and multithreading environment.

2.5.1 What and Why parallel processing

Parallel computing is a form of computing in which many instructions are carried out simultaneously. Large problems will be divided into smaller ones that can carry out concurrently (en.wikipedia.org, 27).

The main benefit of parallel processing is increasing computational speed (throughput) by using multiple processors to execute different parts of the same program simultaneously and reduce Wall Clock Time (mpc.uci.edu).The two major techniques for throughput computing are multiprocessing and multithreading (en.wikipedia.org, 23)

There are two basic ways to partition computational work among parallel tasks (mpc.uci.edu):

1. Data parallelism: each task performs the same calculations to different data.
2. Functional parallelism: each task performs different calculations on the same data or different data.

Flynn classified programs and computers by whether they were operating using a single or multiple sets of instructions, whether or not those instructions were using a single or multiple sets of data as in table 2.1 :

Table 2.1 Flynn's taxonomy (en.wikipedia.org, 22).

	Single Instruction	**Multiple Instruction**
Single Data	**SISD**	**MISD**
Multiple Data	**SIMD**	**MIMD**

Single instruction refers to the fact that there is only one instruction stream being acted on by the CPU during any one clock tick .An example of SISD is Personal Computer (mpc.uci.edu).

A traditional computer consists of a processor executing the programs stored in the main memory (Wilkinson B, Allen M.,1998).The architecture of the traditional computer is shown in the Figure 2.8.

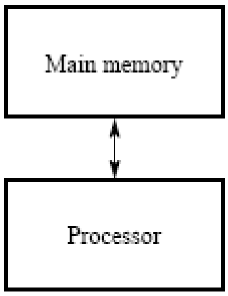


Figure 2.8: The architecture of the traditional computer (Wilkinson B, Allen M., 1998)

The MISD is a type of parallel computing architecture where many functional units perform different operations on the same data. Pipeline architectures belong to this type (en.wikipedia.org, 31).Figure 2.8 shows the structure of MISD.

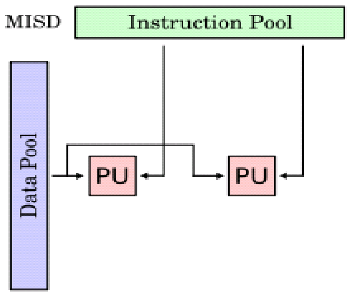


Figure 2.9: MISD (en.wikipedia.org, 31).

In SIMD, different processors may be executing same instruction on multiple streams of data simultaneously.

Machine using MIMD has a number of processors that function asynchronously and independently. Different processors may execute different instructions on different pieces of data. Application areas of MIMD are like computer-aided design/computer-aided manufacturing, simulation, modeling, and as communication switches (en.wikipedia.org, 22). Figure 2.9 shows the structure of MIMD.

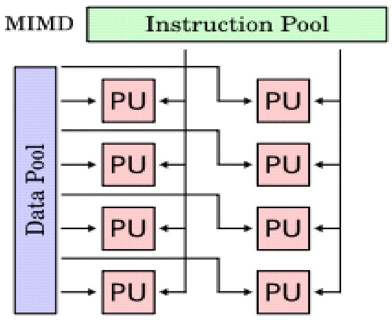
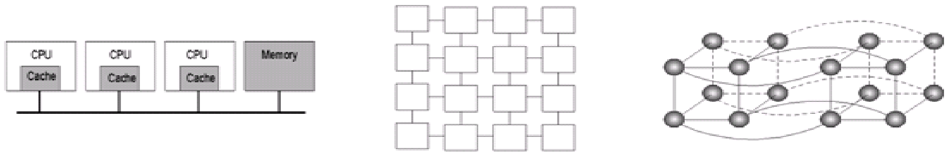


Figure 2.10: MIMD (en.wikipedia.org, 31).

Based on how processors access memory MIMD machines could be classified into (en.wikipedia.org, 22):

- Shared memory machines like (bus-based, extended, or hierarchical type) see figure 2.11(a).

- Distributed memory machines like (hypercube or mesh interconnection) see figure 2.11(b,c).



(a) **(b)** **(c)**

Figure 2.11: (a) MIMD bus-based,**(b)** mesh interconnection and **(c)** hypercube (Tanenbaum ,1992)

2.5.2 Multithreading Environment.

A thread is a low (light) weight process. Using thread is one way for a program to divide itself into two or more simultaneously running tasks. A thread is contained inside a process and different threads of the same process share some resources, while different processes do not (en.wikipedia.org, 33).

Multithreading refers to two or more tasks executing concurrently within a single program. Multiple threads can be executed in parallel on many computer systems or in the same system. This multithreading generally occurs by time slicing, wherein, a single processor switches between different threads (en.wikipedia.org, 33).

While threading enhances performance, it adds complexity to the program. This complexity arises primarily from: threads communication and interaction and waiting time (waiting other threads to finish or to unlock objects) (www.devx.com).

Updating the shared variable between threads causes interleaving between threads and data inconsistency. A lock is a programming language construct that allows one thread to take control of a variable by preventing other threads from reading or writing it while it is locked by another thread (en.wikipedia.org,27).

There are several locking options. The most common of these is the use of the synchronization. With synchronized method (function), only one thread can execute that method at any given time.

As we described before, the benefit of converting sequential algorithm to parallel one is increasing computational speed (throughput). The most important step on the process of converting the sequential algorithm to a parallel algorithm (parallelization or Parallel Formulation) is identifying those activities that can be executed in parallel according to the dependencies between tasks. This may be achieved in several ways : by task, by data, or by data flow. (Akhter and Roberts, 2007).

The required steps in the parallelization process are (Grama, et al., 2003):

First step is finding Concurrent Pieces of work or decomposition. The most common decomposition methods are (Grama, et al., 2003):

1. Data decomposition which is performed by partitions the data and assigns each part of data to specific processes or processors
2. Task Decomposition

- Recursive Decomposition :Suitable for problems that can be solved using the divide-and-conquer paradigm (ex. Quick sort algorithm)
- Exploratory Decomposition
- Speculative Decomposition

3. Hybrid Decomposition

Second step is mapping the tasks : proper mapping needs to take into account the task-dependency that represented using a task-dependency graph (see Figure 2.12) and interaction graphs that captures the pattern of interaction between tasks (see Figure 2.13) (Grama, Gupta, Karypis and Kumar, 2003).

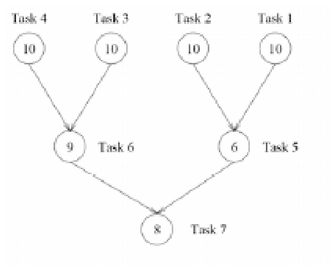


Figure 2.12: task-dependency graph (Grama, Gupta, Karypis and Kumar, 2003)

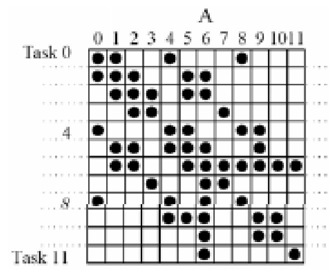


Figure 2.13: task- interaction graph (Grama, Gupta, Karypis and Kumar, 2003).

A data dependency is one of the foundations of knowing how to implement parallel algorithms. No program can run more quickly than the longest chain of dependent calculations (critical path) (en.wikipedia.org, 27).

2.5.3 Parallel Processing Performance

In sequential codes, the performance indicator is the running time, that measured by CPU time as a function of input size. With parallel computing we focus not just on running time, but also on how the additional resources (typically processors) affect this running time (Bader, Moret and Sanders, 2002).

The question that should be answered is “does using twice as many processors cut the running time in half?”. To answer this question, two main concepts must be considered which are (Bader, Moret and Sanders, 2002):

- **Speed Up**(S_P): The absolute speedup is the ratio of the running time of the fastest known sequential implementation (T_1) to that of the parallel running time (T_P) (Gebremedhin, 1999) It is expressed as in equation (1).

$$S_p=T_1 / T_P \quad \text{……………………} \quad (1)$$

- **Efficiency (E):** one of the major overhead sources is the inter processor communication, the performance measure that shows this issue is Efficiency. Efficiency is defined as the ratio of speed up (S_P) to number of processors (P) as in Equation (2) (Gebremedhin, 1999).

$$E_P=S_P/p \quad \text{……………………} \quad (2)$$

The performance factors that affecting the results of parallel processing are:

1. The hardware of the environment that will be used for running the algorithm, such as process (CPU) speed , main Memory and Cache
2. Graph parameters (Size (**N**) and Density(**D**))

   Where, increasing the graph size means more computation time which will effects the speed up and efficiency of a given algorithm. In the other hand, increasing the density value leads to minimize the MIS size and reduce running time. A density of a graph can be computed as in Equation (3).

D = ∑ E for each node / N (N-1) …………………… (3)

Chapter 3
The Proposed Heuristic Cellular Automata Algorithms based on Weighted Factors

As we described before, this Thesis introduces a new HCA Algorithms based on Weighted Factors for finding a MIS in a Graph. The algorithms will use cellular automata and heuristic approaches by choosing the candidate nodes in MIS: randomly, node with maximum degree, and node with minimum degree. To speedup the process of finding MIS and in order to find all possible MIS's from different starting points, parallel implementations of the sequential algorithms using Multithreads was investigated.

3.1 Sequential Algorithm.

The proposed algorithm was mapped into the cellular model according to the following assumptions and rules:

1. Mapping the graph with two dimensional CA by making the cell that has a vertex as mutable and the cell that hasn't vertex as immutable.
2. Using Moore neighborhood for fully connected graphs and semi connected graphs, and von Neumann neighborhood for grid graphs.
3. Each node should assign one of two values either odd (ex. one) or even (ex. two) depends on its neighbor's values. Any node connected to odd node must be even; otherwise it must be odd.
4. Proceed in the process of finding MIS node by node by choosing the node with specific weight (randomly, node with maximum degree, and node with minimum degree)

5. Don’t address the node that already has been addressed.
6. Cells without vertices are state "None" (immutable).
7. Add any cell with degree zero to MIS.

In the basic approach of HCA there is another assumption where we have to address level by level (don’t go to level x+1 until you finished the level x), for example in Figure 3.1(c) nodes (b & d) both have value 2, so we choose one node of them(d) as in Figure 3.1(d), then we proceed with the other node (b) as in Figure 3.1(e), after that we can go to next level .This approach has been modified to increase the accuracy of the MIS size ,so we can go to any level , in order to choose the node with specific weight (minimum degree ,maximum degree or randomly) regardless of it's position in the graph.

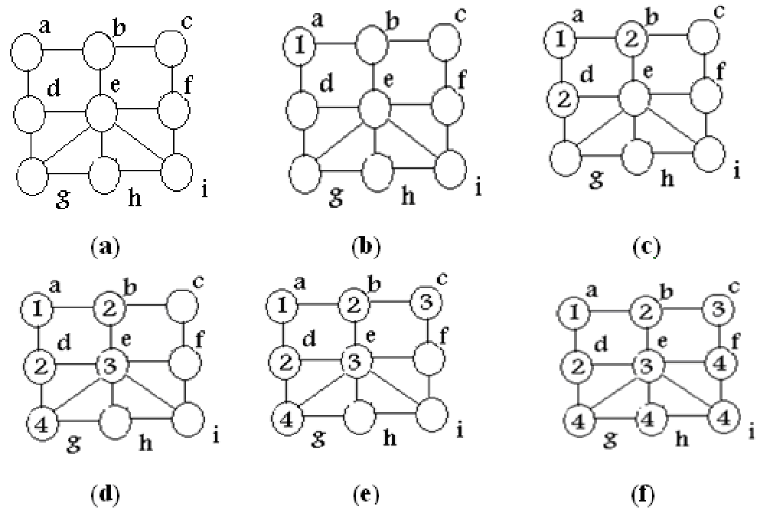


Figure 3.1: HCA Basic Approach

3.1.1 Algorithm Description

The input to the cellular automata MIS (CA-MIS) algorithms is an undirected graph G = (V, E) and the output is a maximum independent set I(G) $\in$ G, where:

- **V_0** denotes to start node.
- **MIS** denotes to the nodes that consist MIS and it’s the same as **$V_{odds.}$**

- **V** $_{\textbf{addressed}}$ denotes to Addressed Nodes
- **V** $_{\textbf{even}}$ denotes to Even Node and **V** $_{\textbf{odd}}$ denotes to Odd Node
- **N(V)** denotes to Node's Neighbors
- **N(V)** $_{\textbf{NA}}$ denotes to Node's Neighbors that are not addressed yet.
- **D(V)** denotes to the degree of node V
- **Q0** denotes to Queue Number 0 that store all the nodes of the graph.
- **Max** $_{\textbf{D(V)}}$ denotes to Node with maximum degree.
- **Min** $_{\textbf{D(V)}}$ denotes to Node with minimum degree.
- **V** $_{\textbf{Random}}$ denotes to the Node that chosen randomly

The following steps describe the process of the Sequential algorithm

1. The algorithm starts by initiating the function's variables, and storing all of the graph's (G) nodes in Q0.
2. Go in do-while loop to check if it visits all the nodes in G or not. This is done by checking if there are still existing nodes in Q0.
3. Remove the selected node from Q0.
4. Get node V information in order to get node's neighbors N(V) .
5. Set V as odd Node (**V** $_{\textbf{odd}}$) ,
6. Check if all the nodes in N(V) are addressed or not in order to get **N(V)** $_{\textbf{NA}}$ **.**
7. If **N(V)** $_{\textbf{NA}}$ not zero **do.**

    7.1 Set $N(V)_{NA}$ as even Nodes (V_{even})

    7.2 Add $N(V)_{NA}$ to the Addressed Nodes

    7.3 Get V_1 from $N(V)_{NA}$ ($V_1 \in$ ($Max_{D(V)}$, $Min_{D(V)}$ or V_{Random}))

    7.4 Remove $N(V)_{NA}$ from Q0

    7.5 Get node V_1 information in order to get node's neighbors $N(V_1)$.

7.6 Checks if all the nodes in $N(V_1)$ are addressed or not in order to get $N(V_1)_{NA}$

8. End **do**

9. If $\mathbf{N(V_1)_{NA}}$ **is** not empty get node V_2 from $\mathbf{N(V_1)_{NA}}$ other wise get node from Q0 .

10. At the end I(G) =MIS and i(G)= length of MIS.

Instruction

```
MIS (G)
    Initiate  MIS variables
    Choose V as starting node
    do {
        Remove the selected node(V)  from Q0
        Get V Information
        Set V as  V odd
        Get N(V)NA
        If  N(V)NA  length != 0 {
            Set  N(V) as Veven
            Add  N(V) to the Addressed Nodes
            Get V1 from N(V) NA (V1 ∈( Max D(V), Min D(V) or VRandom))
            Remove N(V)NA  from Q0
            Get V1 Information
            Get N(V1)NA
        }
      If  Q0 length != 0{
          If  N(V1)NA  != 0
            Get V2 from N(V1)NA ( V2 ∈( Max D(V), Min D(V) or VRandom))
          Else
            Get V2 from Q0( V2 ∈( Max D(V), Min D(V) or VRandom))
      }
    } while Q0 length != 0
      I(G)=MIS
End;
```

Figure 3.2: The sequential CA-MIS algorithm Pseudo code

3.1.2 Examples to explain CA-MIS

Example 1:

The following example illustrates the process of finding the MIS based on weight factor (Minimum Degree) for the graph in Figure 3.1.

Table 3.1: Steps for Finding MIS – Example 1

Steps	Figure
Mapping the graph with two dimensional CA by making the cell that has a vertex as mutable and the cell that hasn't vertex as immutable.	3.1(a)
Before numbering any odd node make sure it is not connected to any other odd node, if it is not connected continue in the normal way, else new state is equal to neighbor(odd) +1	
Don't address the one that already addressed	
Starting with node (a) and assign value 1 to it	3.1(b)
Current state value of (a) is odd then the new state of its neighbors = minimum(neighbor)+1.So(b, d)=2	3.1(c)
Choose Node with maximum degree from (b,d) .Since both nodes have the same degree ,choose any node(ex. node d).	
Get node (d) neighbors (nodes e and g) and get the maximum degree node from these nodes and assign odd value to it (node e =3)	3.1(d)
Since nodes e and g are connected and node (e) odd then assign node g =4	3.1(d)
Before moving to next level , finish processing the current level by assigning value 3 to node (c)	3.1(e)
Choose node with maximum degree (e) and process it's neighbors (nodes f, h and i).	
Assign value 4 to nodes (f, h and i).	3.1(f)

Example 2:

The following example illustrates the process of finding the MIS based on weight factor (Minimum Degree) for the graph in Figure 3.3. The number inside the node is the node number and the number outside the node is the degree of the node. In Table 3.2 the first column represents the step description in the Pseudo code and the second column represents the step number in the Pseudo code.

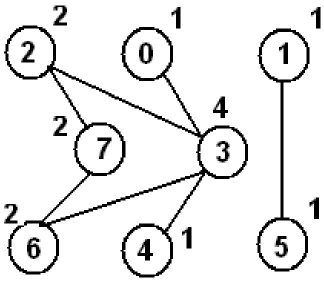
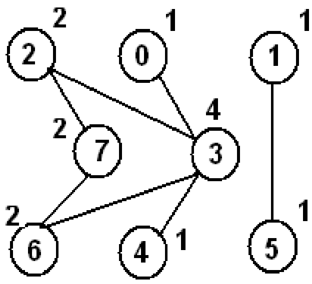
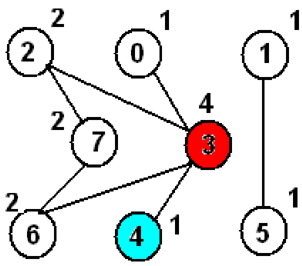
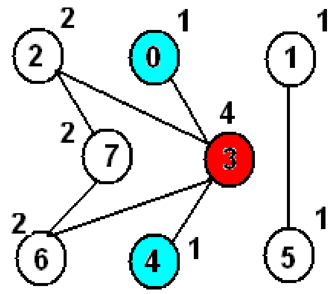
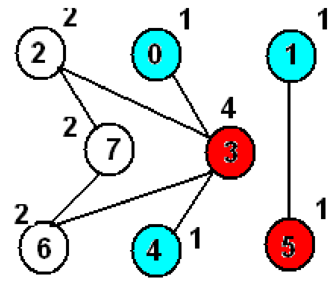
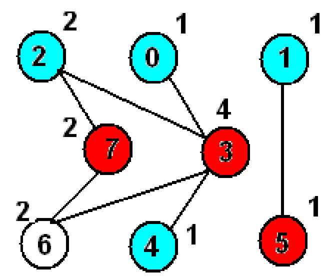
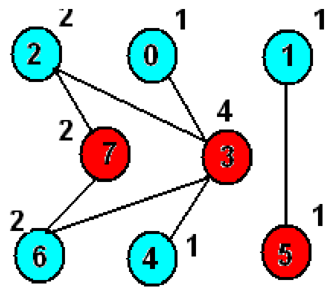


Figure 3.3: a graph with 8 Nodes and 7 Edges

Table 3.2: Steps for Finding MIS – Example 2

Steps	No		
▪ Initiate MIS variables ▪ Choose V_0 from G (ex. Node(4))	1 2	Graph G(8,7) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | MIS= V odd | | | | | | | | |	
▪ Remove (4) from G ▪ Get V Information ▪ Set V as V odd ▪ Get N (4), which is Node(3). ▪ Set N (4) as V_{even} ▪ Get Min. Degree Node from N(4) which is node(3) ▪ Remove N (4) from G.	4 5 6 7 9 11 12	Graph G(8,7) | 0 | 1 | 2 | ~~3~~ | ~~4~~ | 5 | 6 | 7 | MIS= V odd | 4 | | | | | | | |	

▪ Choose Node with Min Degree from N(3) which is (0) ▪ Remove node(0) from G ▪ Get V Information ▪ Set V as V $_{odd}$ ▪ Get $N(0)_{NA}$ ▪ Since $N(0)_{NA}$ is null	18 4 5 6 7 17	Graph G(8,7) ~~0~~ 1 2 ~~3~~ ~~4~~ 5 6 7 MIS= V $_{odd}$ 4 0	
▪ Choose Node with Min Degree from G which is (1) ▪ Remove node (1) from G. ▪ Get V Information ▪ Set V as V $_{odd}$ ▪ Get $N(1)_{NA}$ ▪ Set $N(1)_{NA}$ as V_{even} ▪ Get Min. Degree Node from $N(1)_{NA}$ ▪ Remove N(1) from G. ▪ Get V Information ▪ Get $N(5)_{NA}$ ▪ Since $N(5)_{NA}$ is null	20 4 5 6 7 9 11 12 13 14 17	Graph G(8,7) ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ 5 6 7 MIS 4 0 1	
▪ Choose Node with Min Degree from G which is (2) ▪ Remove node(2) from G ▪ Get V Information ▪ Set Node(2) as V $_{odd}$ ▪ Get N(2) ▪ Set N (2) as V_{even} ▪ Get Min. Degree Node from N(2) which is (7) ▪ Remove N(2) from G.	20 4 5 6 7 9 11 12	Graph G(8,7) ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 MIS 4 0 1 2	

▪ Choose Node with Min Degree Node from N(7) which is node(6) ▪ Remove node (6) from G. ▪ Get V information ▪ Add Node(6) to V_{odd} ▪ Get N(6) ▪ Since N(6) is null go to end the loop. ▪ The output is	18 4 5 6 7 22	Graph G(8,7) ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ MIS = V_{odd} 4 0 1 2 6 MIS={0,1,2,4,6} And size of MIS is 5	

3.3 Analyzing of the algorithms

Algorithms are devised to solve the same problem often differ dramatically in their efficiency. These differences can be much more significant than differences due to hardware and software. Efficiency means needs fewer resources including CPU time and Computer Memory (space) (Cormen, Leiserson, Rivest and Stein, 2001).The running time of an algorithm on a particular input is the number of primitive operations or "steps" executed. It is convenient to define the notion of step so that it is as machine independent as possible (Cormen, Leiserson, Rivest and Stein, 2001).

Two algorithms may perform the same task, but one is more "efficient" than the other. The time taken by the MIS procedure depends on:

- The size(N) of a graph (G), finding MIS in graph with size of 1000 nodes takes longer than finding MIS with size of 10 Nodes.

- The density (D) of a graph and the nodes degree, an algorithm to find a MIS can take different amounts of time for two different input sequences of the same size. As the maximum degree of the selected node increases, the number of generated sub graph decreases and the running time of the algorithm decreases.

In general, the time taken by an algorithm grows with the size of the input, so it is traditional to describe the running time of a program as a function of the size of its input.

The complexity (Big O of running time) of the suggested algorithms is derived from the pseudo code as shown in tables (3.3, 3.6 and 3.7):

Table 3.3 MIS Using Maximum Degree Factor proposed algorithm Pseudo code

No	Instruction	Cost	Times
	MIS (G)		
1	Initiate MIS variables	C1	1
2	Choose V as starting node	C2	1
3	do {	C3	$n^{0.8}$
4	Remove the selected node(V) from Q0	n^2	$n^{0.8}$
5	Get V Information	C4	$n^{0.8}$
6	Set V as V_{odd}	C5	$n^{0.8}$
7	Get $N(V)_{NA}$	n^2	$n^{0.8}$
8	If $N(V)_{NA}$ length != 0 {	C6	$n^{0.8}$
9	Set N(V) as V_{even}	C7	$n^{0.8}$
10	Add N(V) to the Addressed Nodes	C8	$n^{0.8}$
11	Get V1 from $N(V)_{NA}$ (V1 = $Max_{D(V)}$)	n	$n^{0.8}$
12	Remove $N(V)_{NA}$ from Q0	n^2	$n^{0.8}$
13	Get V_1 Information	C9	$n^{0.8}$
14	Get $N(V_1)_{NA}$	n^2	$n^{0.8}$
15	}	C10	$n^{0.8}$
16	If Q0 length != 0{	C11	$n^{0.8}$
17	If $N(V_1)_{NA}$!= 0	C12	$n^{0.8}$
18	Get V_2 from $N(V_2)_{NA}$ (V_2 = $Max_{D(V)}$)	n	$n^{0.8}$
19	Else	C13	$n^{0.8}$

20	Get V_2 from Q0(V_2 = Max $_{D(V)}$)	n	$n^{0.8}$
21	}	C14	$n^{0.8}$
22	} while Q0 length != 0	C15	$n^{0.8}$
23	I(G)=MIS	C16	1
	End;		

The complexity of this algorithm is based on the following factors

1. Do - While loop (steps from 3 to 22 in the MIS(G) pseudo code)

   The Do-While loop value (lv) is equal to the size of the MIS (*i(G)*) .It is less than the graph size n (lv < n) and it could be calculated from the equations

   $$lv = i(G) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots.. (1)$$

   $$lv = n^L \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots.. (2)$$

   $$i(G)= n^L \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots.. (3)$$

   To calculate the complexity of the algorithm we need to compute L and we must take the worst case while finding this value. So we have to choose the right values of N and D where the ratio of dividing the size of the MIS on the number of nodes (R_{MIStoN}= i(G) / n) is the biggest.

   According to the results obtained from the exact algorithms (Wilf and Modified Wilf) this value (R_{MIStoN}) is the biggest value when (n=10 and d=0.1) and as shown in table (3.4). So by solving the Equation (3) we have the following result

   $7=10^L \rightarrow L \approx 0.8 \rightarrow vl = n^{0.8}$ (Where 7 is the size of the MIS when n=10)

Table 3.3 shows a complexity comparison between Wilf and the proposed algorithm.

Table 3.4 Computing the complexity of the algorithm

n	i(G)	R_{MIStoN}	L	$O(n^{3.8})$	$O(1.619^n)$
10	7	**0.7**	**0.8**	6,310	124
20	11	0.55	0.8	87,885	15,309
30	14	0.47	0.77	410,262	1,894,111
40	17	0.43	0.77	1,224,131	234,354,852
50	19	0.38	0.77	2,858,157	28,996,290,205
60	22	0.37	0.75	5,714,454	3,587,657,081,106
70	22	0.31	0.73	10,265,321	443,894,140,965,424
80	25	0.31	0.73	17,050,690	54,922,196,834,566,200
90	26	0.29	0.72	26,676,051	6.8E+18
100	28	0.28	0.72	39,810,717	8.41E+20
200	35	0.18	0.67	554,515,875	7.07E+41
300	41	0.14	0.65	2,588,575,092	5.94E+62
400	44	0.11	0.63	7,723,745,711	5E+83
500	48	0.1	0.63	18,033,748,824	4.2E+104
600	50	0.08	0.61	36,055,768,070	3.5E+125
700	50	0.07	0.6	64,769,799,409	3E+146
800	53	0.07	0.6	107,582,578,868	2.5E+167
900	53	0.06	0.6	168,314,501,772	2.1E+188
1000	53	**0.05**	**0.6**	251,188,643,151	1.8E**+209**

In this thesis we are interesting in finding the MIS for the graphs with large size, where Wilf and Modified Wilf can't computed any more for small values of d .So the value of L become for example 0.75 when n=60 and 0.6 when n=700,800,900 or 1000.

2. The called functions in the pseudo code.

   All of these functions or methods search certain string to retrieve or remove sub string from it. And the complexity of each method can be computed as the following

Table 3.5 Methods complexity

Step No.	Step	T(n)
4	Remove the selected node(V) from Q0	$(n-1)*(n-1) \approx O(n^2)$
7	Get $N(V)_{NA}$	$(n-1)*(n-1) \approx O(n^2)$
11	Get V_1 from $N(V)_{NA}$	$(n-1) \approx O(n)$
12	Remove $N(V)_{NA}$ from Q0	$(n-1)*(n-1) \approx O(n^2)$
14	Get $N(V_1)_{NA}$	$(n-1)*(n-1) \approx O(n^2)$
18	Get V_2 from $N(V_1)_{NA}$	$(n-1) \approx O(n)$
20	Get V_2 from Q0	$(n-1) \approx O(n)$

3. Number of nodes in the graph (n).

The Required time for finding IS of each node is

$T(n)= (C1+C2+C16) * 1 + (C3+C4+C5+C6+C7+C8+C9+ C10+C11+C12$

$+C13+C14+C15)*(n^{0.8}) + 4*(n^2)*(n^{0.8}) +3*(n)*(n^{0.8})$

$T(n) \approx C1*(n^2)(n^{0.8}) + C2 *(n)*(n^{0.8}) + C3(n^{0.8})+ C4$

$$T(n) \approx O(n^{2.8}) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

We obtain the MIS size by finding the IS of each node in the graph and take the biggest value as the MIS value, thus we have the complexity as in Equation (5)

So the big O is equal to

$$T(n)= O(n)*O(n^{2.8}) \approx O(n^{3.8}) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

It's not going be worst than $O(n^4)$.

We can summarize the benefits of using local rules of CA and heuristic approaches in the proposed algorithms in the following points:

1. Minimizing the search space.
2. Making the process of finding the MIS straight forward.
3. Decreasing the running time and Increasing the performance of the proposed algorithms.

Table 3.6 MIS Using Minimum Degree Factor proposed algorithm Pseudo code

No	Instruction	Cost	Times
	MIS (G)		
1	Initiate MIS variables	C1	1
2	Choose V as starting node	C2	1
3	do {	C3	$n^{0.8}$
4	Remove the selected node(V) from Q0	n^2	$n^{0.8}$
5	Get V Information	C4	$n^{0.8}$
6	Set V as V_{odd}	C5	$n^{0.8}$
7	Get $N(V)_{NA}$	n^2	$n^{0.8}$
8	If $N(V)_{NA}$ length != 0 {	C6	$n^{0.8}$
9	Set N(V) as V_{even}	C7	$n^{0.8}$
10	Add N(V) to the Addressed Nodes	C8	$n^{0.8}$
11	Get V1 from $N(V)_{NA}$ (V1 = Min $_{D(V)}$)	N	$n^{0.8}$
12	Remove $N(V)_{NA}$ from Q0	n^2	$n^{0.8}$
13	Get V_1 Information	C9	$n^{0.8}$
14	Get $N(V_1)_{NA}$	n^2	$n^{0.8}$
15	}	C10	$n^{0.8}$
16	If Q0 length != 0{	C11	$n^{0.8}$
17	If $N(V_1)_{NA}$!= 0	C12	$n^{0.8}$
18	Get V_2 from $N(V_2)_{NA}$ (V_2 = Min $_{D(V)}$)	N	$n^{0.8}$
19	Else	C13	$n^{0.8}$
20	Get V_2 from Q0(V_2 = Min $_{D(V)}$)	N	$n^{0.8}$
21	}	C14	$n^{0.8}$
22	} while Q0 length != 0	C15	$n^{0.8}$
23	I(G)=MIS	C16	1
	End;		

$T(n) \approx O(n^{3.8})$

Table 3.7 MIS Using Random Degree Factor proposed algorithm Pseudo code

Instruction	Cost	Times
MIS (G)		
Initiate MIS variables	C1	1
Choose V as starting node	C2	1
do {	C3	$n^{0.8}$
Remove the selected node(V) from Q0	n^2	$n^{0.8}$
Get V Information	C4	$n^{0.8}$
Set V as V_{odd}	C5	$n^{0.8}$
Get $N(V)_{NA}$	n^2	$n^{0.8}$
If $N(V)_{NA}$ length != 0 {	C6	$n^{0.8}$
Set N(V) as V_{even}	C7	$n^{0.8}$
Add N(V) to the Addressed Nodes	C8	$n^{0.8}$
Get V1 from $N(V)_{NA}$ (V1 = V_{Random}))	n	$n^{0.8}$
Remove $N(V)_{NA}$ from Q0	n^2	$n^{0.8}$
Get V_1 Information	C9	$n^{0.8}$
Get $N(V_1)_{NA}$	n^2	$n^{0.8}$
}	C10	$n^{0.8}$
If Q0 length != 0{	C11	$n^{0.8}$
If $N(V_1)_{NA}$!= 0	C12	$n^{0.8}$
Get V_2 from notAddressedNodes ($V_2 = V_{Random}$)	n	$n^{0.8}$
Else	C13	$n^{0.8}$
Get V_2 from Q0($V_2 = V_{Random}$)	n	$n^{0.8}$
}	C14	$n^{0.8}$
} while Q0 length != 0	C15	$n^{0.8}$
I(G)=MIS	C16	1
End;		

T(n) $\approx$ ***O(*** $n^{3.8}$**)**

The Accuracy (A) was reported experimentally .The sequential running time (T_{CA}) for the proposed approaches and the Wilf's approach running time (T_{Wilf}) will be measured. The ratio of optimization on the running time (R_t) is obtained by divided the Wilf's running time (T_{Wilf}) into the proposed algorithms running time (T_{CA}) as in Equation (6)

$$R_t = T_{Wilf} \; / \; T_{CA} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(6)$$

The Accuracy (A) is the absolute value of the ratio of *i(G) of CA-MIS* to Wilf's' *i(G)* as in Equation(7) A= | i(G) of CA-MIS / i(G) of Wilf's' | ………………….(7)

3.4 The Parallel Implementation of the algorithm

To speedup the process of finding MIS and in order to find all possible MIS's from different starting points, parallel implementations of the sequential algorithms using Multithreads were investigated.

In order to reach to the best model in this study, we assume the following points:

1. For a given graph G of size (s) and density (d), the cellular algorithm and the multithreading was used to find the size of MIS and MIS.
2. The graph was represented in data structures such as matrix and string.
3. We used data decomposition.
4. We studied and tested many graphs (produced randomly) with different densities and sizes.
5. The Multi-Threading environment consists of one main thread that will act as central thread for data management and processors synchronization (server) and many threads that will act as processor for each task or data part (clients). There will be **M** threads (T_1, T_2, … , T_M); where M is less than or equals to the number of nodes in the graph. Each thread runs to find a MIS starting with a specific node in the graph. Then, the largest MIS is selected to be as the MIS in the graph (Figure 3.4).

6. The algorithm works as following :
   - The program starts by creating M threads (T1, ... ,T_M) as shown in Figure 3.4
   - Start all the threads. Each thread contains code that loop until there are no nodes need processing.
   - Each thread takes one node and starts finding MIS for this node then it add the result to array and starts processing another node.
   - The process continues until finding all MIS for all Nodes. MIS is the MIS with maximum length.

The complexity (Big O of running time) of the suggested algorithms was derived from the pseudo in table 3.8:

Table 3.8(a) Parallel MIS Using Minimum Degree Factor proposed algorithm Pseudo code - Client

Instruction	Cost	Times
MIS (G){		
InitProgVar();	**C1**	**1**
while (G.length() != 0) {	**C2**	**n**
String t=GetNodeFromQ0();	**C3**	**n**
if(t !="@"){	**C4**	**n**
call MISProgramm(G);	**$n^{2.8}$**	**n**
MISArr[inx]=MIS;	**C5**	**n**
inx++;	**C6**	**n**
}	**C7**	**0**
}	**C8**	**0**
}	**C9**	**0**

$T(n) \approx O(n^{3.8})$

Table 3.8(b) Parallel MIS Using Minimum Degree Factor proposed algorithm Pseudo code - Server

Instruction	Cost	Times
Define Array of Threads	**C1**	**1**
Loop for Number of threads {	**C2**	**M**
Create thread Instance	**C3**	**M**
Start the Instance	**C4**	**M**
}	**C5**	**0**

O(n) =C

The speedup (Sp) and the efficacy (E) were reported experimentally and the sequential running time (Ts) and the multithreading running time (Tp) will be measured. The speed Up is the ratio of the running time of the fastest known sequential implementation (Ts) to that of the parallel running time (Tp) see Equation (8) .

$$Sp=Ts / Tp \ldots\ldots\ldots\ldots(8)$$

And the Efficiency (E) =the ratio of speed up (S) to number of processors (P) see Equation (9).

$$E= Sp / p \ldots\ldots\ldots\ldots (9)$$

7. Number of threads that give us the best speedup was figured out to achieve the required optimality.

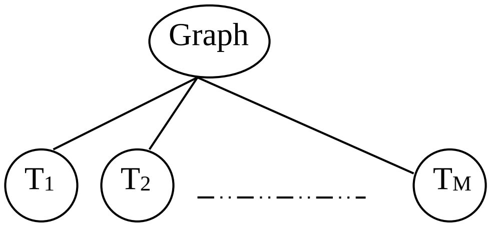


Figure 3.4: The Multi-Threading environment

Chapter 4
Experimental Results and Discussion

In addition to Wilf and Modified Wilf, we implemented the proposed algorithms using Java programming language

The Computational experiments have been run using Pentium 4 PC with 3.00 GHZ CPU and 512MB RAM and Windows XP SP3. A program for generating a random graphs using different size N and density D was implemented in Java Language. Different graphs vary in their size (from 10 nodes to 1000 nodes) were generated, for each size the density varies from 0.1 to 0.9.

Table 4.1 shows the computational results for the generated graphs, each with different sizes (from 10 to 100 increases by 10 and from 200 to 1000 increases by 100). The table (4.1) shows the sizes of the MIS and CPU running time of Wilf's, the modified Wilf's and the heuristic proposed algorithms. In some cases (like N=60 and D=0.1) we can't measure the CPU running time (ms) of Wilf's and the modified Wilf's algorithms even after three hours of running , so we add the value (10800000) that represents the three hours as estimated time and Unknown (UN) for MIS size. The table (4.1) compares the exact results (obtained by the exact algorithms Wilf's and modified Wilf's) and the non exact results (obtained by the heuristic algorithms).

Table 4.2 shows accuracy of each algorithm compared to the modified Wilf's. The accuracy is obtained by dividing the size of MIS of the proposed algorithm into the size of MIS of the modified algorithm.

The results indicate that the heuristic algorithms produce closer sizes for any density of graphs. For example, the Minimum Degree algorithm there are about 75% of the sizes of the MIS produced by proposed algorithms have the same size as the exact algorithm , 23% differ in their size by one node and 2% differ in their size by two nodes.

Accuracy Summary Table 4.3 and Figure 4.1 show that the Minimum degree factor algorithm is the most accurate algorithm (97.35%) in general, followed by the Random Degree algorithm in the second level (95.25%) and finally the Maximum degree factor algorithm (89.44 %). These values obtained by taken the average of all of the accuracy values for the sample in table 4.1.

Chosen the minimum degree vertex leads to a minimum size of vertices to be removed (N(v)) , and the process of finding IS continue for longer time resulting in a large size IS.

Table 4.2 also shows the ratio of optimization on the running time (R_t) that obtained by divided the Wilf's running time (T_{Wilf}) into the proposed algorithms running time (T_{CA}) as shown in Equation (1)

$$R_t = T_{Wilf} \;/\; T_{CA} \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

It is very clear that the proposed algorithms have less CPU run time than Wilf's and modified Wilf's in most cases In some cases the proposed algorithms take 0.000004 of the Wilf's and modified Wilf's running time.

The run time of the proposed algorithms shows a great R_t as the density of the graph decreases as in Tables (4.5, 4.6, 4.7, 4.8) (The proposed algorithms achieve great performance for small density). The results indicate that the proposed algorithms have more CPU running time when D=0.9.

The summery table 4.4 and Figure 4.2 show that the fastest algorithm is the maximum Degree Algorithm in general, followed by the minimum Degree Algorithm, and finally the Random Degree Algorithm.

When we change the size of Graph (N) and make the Graph density D fixed and small (Figure 4.3), we found that the size of MIS (i(G)) is increases as N increases. The sizes of MIS's Increase as the density of the graph decreases.

When density increases the size of MIS become fixed regardless the value of N as in Figure 4.4, for example in Table 4.13 when D=0.9 the i(G) of the Graph with N=1000 is almost equal to the i(G) of the Graph with N=100.

The experimental results in Tables 4.13 indicate that for a fixed density the sizes of the generated MIS's are very close. For example, the sizes of the generated MIS's for density = 0.9 remains 5 as the sizes change from 400 through 1000.

Table 4.23 shows the results of the Speedup (S_p) for the parallel implementation which obtained by divided the running time of Parallel CA-MIS over Sequential CA-MIS for N=1000. The result of S_p is equal to1.2. The parallel implementation of the proposed algorithm using Multithreading does not appear to be great performance over the sequential one for the suggested samples, and it could be more suitable for graphs with large size (ex. 10,000) and on multiprocessors or multicomputers environment.

The number of threads is one of the important factors that affect the S_P value. For Example increasing the number of threads from 25 to 50 for a graph with N=1000 increases the running time of the parallel implementation to 110% and decreases the S_P value.

Table 4.1 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches (**N**: Size of the Graph – **D**: Density, **S**: IS Size, **T**: CPU Running Time (ms))

		Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	**D**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
10	**0.1**	7	0	7	16	7	31	7	16	7	0
	0.2	6	0	6	0	6	15	6	0	6	0
	0.3	5	0	5	0	5	16	5	0	5	0
	0.4	5	0	5	0	5	0	5	0	5	0
	0.5	4	0	4	0	4	0	4	0	3	16
	0.6	3	0	3	0	3	16	3	0	3	0
	0.7	3	0	3	0	3	16	3	0	3	0
	0.8	2	0	2	0	2	16	2	0	2	0
	0.9	2	16	2	0	2	16	2	0	2	0
20	**0.1**	11	31	11	47	11	16	11	0	10	0
	0.2	9	0	9	16	9	15	9	16	8	15
	0.3	8	0	8	0	8	15	8	0	7	0
	0.4	6	0	6	0	5	16	6	0	5	0
	0.5	5	0	5	0	5	15	5	0	5	0
	0.6	4	0	4	0	4	16	4	0	4	0
	0.7	4	0	4	0	4	16	4	0	4	0
	0.8	4	0	4	0	4	16	4	16	4	15
	0.9	3	0	3	0	3	16	3	0	3	0
30	**0.1**	14	953	14	984	14	31	14	16	12	0
	0.2	11	156	11	141	11	16	11	15	11	15
	0.3	9	31	9	47	9	16	9	0	9	0
	0.4	7	15	7	0	7	16	7	0	7	16
	0.5	6	0	6	0	6	10	6	0	5	16
	0.6	5	0	5	16	5	16	5	0	4	0
	0.7	5	0	5	0	5	15	5	0	4	0
	0.8	4	0	4	0	4	31	4	0	3	0
	0.9	3	0	3	0	3	0	3	0	3	0
40	**0.1**	17	35391	17	30094	16	15	17	15	15	0
	0.2	13	1016	13	953	12	16	13	15	12	15
	0.3	10	141	10	156	10	16	10	16	9	0
	0.4	8	47	8	31	8	15	8	16	7	16
	0.5	7	15	7	16	7	31	7	0	7	16
	0.6	5	0	5	16	5	32	5	0	5	15
	0.7	5	0	5	16	5	31	5	16	5	0
	0.8	4	15	4	15	4	31	4	15	4	0
	0.9	3	15	3	15	3	15	3	16	3	1
50	**0.1**	19	454953	19	339500	18	31	19	16	17	15

	0.2	14	5953	14	4984	14	31	14	31	12	31
	0.3	11	672	11	578	10	31	11	15	10	31
	0.4	10	141	10	141	10	31	10	15	7	31
	0.5	8	47	8	47	8	31	8	31	7	15
	0.6	6	15	6	16	6	31	6	15	6	15
	0.7	5	16	5	16	5	31	5	15	5	15
	0.8	4	15	4	16	4	31	4	15	4	16
	0.9	4	15	4	15	4	31	4	16	4	15
60	**0.1**	UN	10800000	UN	10800000	21	46	22	31	18	16
	0.2	16	40078	16	31672	15	47	16	32	14	31
	0.3	11	2125	11	1906	10	47	11	31	10	47
	0.4	10	344	10	344	10	47	9	32	8	32
	0.5	8	93	8	78	7	62	8	31	7	31
	0.6	7	31	7	32	6	47	6	31	6	31
	0.7	5	15	5	32	5	62	5	31	5	31
	0.8	4	0	4	16	4	47	4	16	4	15
	0.9	4	16	4	15	4	31	4	31	3	32
70	**0.1**	UN	10800000	UN	10800000	22	46	22	47	20	46
	0.2	17	216188	17	148969	15	62	16	47	14	31
	0.3	12	7531	12	6609	12	63	12	46	11	47
	0.4	11	922	11	813	10	62	11	47	8	47
	0.5	8	203	8	203	7	62	8	47	8	62
	0.6	7	62	7	63	6	62	7	78	6	46
	0.7	6	31	6	55	5	63	6	31	5	47
	0.8	4	16	4	48	4	62	4	47	4	47
	0.9	4	16	4	15	4	62	4	31	3	47
80	**0.1**	UN	10800000	UN	10800000	25	78	25	63	22	62
	0.2	17	765906	17	519922	15	78	16	63	16	78
	0.3	14	22313	14	17313	12	93	13	62	11	78
	0.4	10	2063	10	1750	10	94	10	63	9	78
	0.5	8	359	8	328	8	94	8	79	8	63
	0.6	8	125	8	109	8	78	7	78	6	63
	0.7	6	47	6	70	6	78	6	62	5	63
	0.8	5	15	5	70	5	79	5	57	5	63
	0.9	4	15	4	16	4	63	4	63	4	63
90	**0.1**	UN	10800000	UN	10800000	25	109	26	78	22	94
	0.2	19	3656250	19	2142203	17	110	17	94	16	93
	0.3	14	66938	14	47828	13	109	13	93	12	94
	0.4	11	4828	11	3921	11	109	11	94	9	94
	0.5	9	765	9	703	8	110	9	94	7	94
	0.6	8	203	8	188	8	109	8	78	7	78
	0.7	6	62	6	100	5	110	5	78	5	93
	0.8	5	31	5	90	5	93	5	78	5	79
	0.9	4	15	4	60	4	94	4	63	4	63
100	**0.1**	UN	10800000	UN	10800000	26	125	28	109	25	110

	0.2	19	12000000	19	6542265	17	141	18	109	16	109
	0.3	15	151578	15	105422	13	141	13	125	12	109
	0.4	11	8546	11	6937	11	156	10	125	9	140
	0.5	10	1422	10	1204	9	140	9	125	8	125
	0.6	8	328	8	282	7	140	7	125	7	125
	0.7	6	170	6	150	6	140	6	125	6	109
	0.8	5	161	5	130	5	125	5	110	4	110
	0.9	4	16	4	16	4	125	4	93	3	93
200	**0.1**	UN	10800000	UN	10800000	32	985	35	875	31	859
	0.2	UN	10800000	UN	10800000	21	937	22	953	20	985
	0.3	UN	10800000	UN	10800000	16	985	16	984	15	938
	0.4	13	1508000	13	1110391	12	969	12	985	12	938
	0.5	11	89234	11	71047	10	953	10	937	9	937
	0.6	10	11328	10	9500	8	906	9	907	7	907
	0.7	7	2000	7	1734	7	875	7	844	7	859
	0.8	6	1324	6	1092	6	797	6	781	5	766
	0.9	5	156	5	156	4	672	5	672	4	657
300	**0.1**	UN	10800000	UN	10800000	38	2719	41	2703	36	2671
	0.2	UN	10800000	UN	10800000	23	2968	25	3000	22	2968
	0.3	UN	10800000	UN	10800000	16	3031	17	3062	16	3016
	0.4	UN	10800000	UN	10800000	13	3015	13	3031	12	3000
	0.5	12	1442078	12	1073828	11	2954	11	3000	10	2922
	0.6	9	109844	9	89797	8	2860	9	2859	8	2797
	0.7	10	110579	10	89250	8	2813	9	2859	8	2797
	0.8	6	4355	6	3244	5	2453	6	2516	6	2360
	0.9	5	3667	5	2222	5	2156	5	2187	4	2047
400	**0.1**	UN	10800000	UN	10800000	41	6343	44	6422	37	6172
	0.2	UN	10800000	UN	10800000	24	6844	25	6797	23	6688
	0.3	UN	10800000	UN	10800000	18	6953	18	6969	17	6797
	0.4	UN	10800000	UN	10800000	14	6891	15	6969	12	6844
	0.5	UN	10800000	UN	10800000	11	6703	12	6953	11	6688
	0.6	11	688563	11	534813	9	6469	10	6454	9	6344
	0.7	8	59547	8	49578	7	6047	8	6188	7	5953
	0.8	6	8578	6	7297	6	5579	6	5562	6	5390
	0.9	5	1688	5	1438	5	4860	5	4797	5	4766
500	**0.1**	UN	10800000	UN	10800000	45	12453	48	12625	42	12266
	0.2	UN	10800000	UN	10800000	25	13234	27	13281	24	13062
	0.3	UN	10800000	UN	10800000	18	13515	20	13485	17	13172
	0.4	UN	10800000	UN	10800000	14	13297	15	13516	14	13109
	0.5	UN	10800000	UN	10800000	11	13062	12	13109	10	12781
	0.6	11	2702500	11	2046219	9	12453	10	12672	8	12297
	0.7	9	202297	9	160328	7	11769	8	12000	8	11500
	0.8	7	24078	7	19922	6	10891	6	10937	6	10672
	0.9	5	3547	5	3015	5	9421	5	9344	4	9250
600	**0.1**	UN	10800000	UN	10800000	46	21984	50	22094	42	21453

	0.2	UN	10800000	UN	10800000	26	23156	28	23250	24	22843
	0.3	UN	10800000	UN	10800000	21	23453	19	23531	17	23094
	0.4	UN	10800000	UN	10800000	15	23344	16	23656	14	23047
	0.5	UN	10800000	UN	10800000	11	22703	12	22860	11	22313
	0.6	UN	10800000	UN	10800000	9	21641	10	22031	9	21359
	0.7	9	467485	9	377062	8	20594	8	20578	7	20093
	0.8	8	49422	8	41203	7	18922	6	19062	6	18734
	0.9	5	6218	5	5203	5	16640	5	16594	5	16282
700	**0.1**	UN	10800000	UN	10800000	46	37047	50	37125	45	36969
	0.2	UN	10800000	UN	10800000	27	39250	30	39953	27	39172
	0.3	UN	10800000	UN	10800000	20	39563	20	39984	19	39359
	0.4	UN	10800000	UN	10800000	15	39141	16	39532	14	38984
	0.5	UN	10800000	UN	10800000	12	38031	13	38593	12	38062
	0.6	UN	10800000	UN	10800000	10	36360	11	37281	9	36172
	0.7	9	1151188	9	914703	8	34719	9	35078	8	34140
	0.8	7	105594	7	85344	6	31828	7	32015	6	31531
	0.9	6	12375	6	10297	6	27704	5	28640	6	27422
800	**0.1**	UN	10800000	UN	10800000	49	57172	53	57204	45	57062
	0.2	UN	10800000	UN	10800000	28	60031	29	60656	27	59406
	0.3	UN	10800000	UN	10800000	20	60562	21	61265	19	60421
	0.4	UN	10800000	UN	10800000	15	60187	16	60578	15	59313
	0.5	UN	10800000	UN	10800000	13	58563	13	59094	12	85219
	0.6	UN	10800000	UN	10800000	10	56078	11	56282	10	55969
	0.7	9	2541610	9	1970406	8	53187	9	53657	8	52563
	0.8	7	198484	7	159141	7	48937	7	49172	6	48218
	0.9	5	15667	5	125544	5	43547	5	43625	5	43016
900	**0.1**	UN	10800000	UN	10800000	51	70125	53	69407	47	69672
	0.2	UN	10800000	UN	10800000	29	74484	30	75109	28	73937
	0.3	UN	10800000	UN	10800000	20	75250	21	76032	20	74672
	0.4	UN	10800000	UN	10800000	16	74703	16	74907	15	73547
	0.5	UN	10800000	UN	10800000	13	72360	13	71953	11	71968
	0.6	UN	10800000	UN	10800000	10	69578	10	70781	10	68922
	0.7	UN	10800000	UN	10800000	8	65453	9	66187	8	64485
	0.8	7	317734	7	255234	7	60931	7	61312	6	59906
	0.9	5	254222	5	203445	5	53219	5	53469	5	51875
1000	**0.1**	UN	10800000	UN	10800000	50	102016	53	102187	48	101719
	0.2	UN	10800000	UN	10800000	29	107328	32	107828	28	106750
	0.3	UN	10800000	UN	10800000	21	108641	22	109438	20	107781
	0.4	UN	10800000	UN	10800000	16	107735	17	108672	16	106515
	0.5	UN	10800000	UN	10800000	12	105172	13	106907	13	104265
	0.6	UN	10800000	UN	10800000	10	101375	10	101063	10	100203
	0.7	UN	10800000	UN	10800000	8	96047	8	96343	8	94547
	0.8	8	566063	8	457172	7	89062	7	88969	6	88391
	0.9	6	502970	6	411720	5	137765	6	140454	5	136219

Table 4.2 The Accuracy and the ratio of optimization on the running time (R_t) of the proposed Approaches over M.Wilf

N: Size of the Graph – **D**: Density, **S**: IS Size, **T**: CPU Running Time

		Random CA-MIS Accuracy	Minimum CA-MIS Accuracy	Maximum CA-MIS Accuracy	Random CA-MIS R_t	Minimum CA-MIS R_t	Maximum CA-MIS R_t
N	**D**	**%**	**%**	**%**	**times**	**times**	**times**
10	**0.1**	100	100	100	0.5	1	16
	0.2	100	100	100	1	1	1
	0.3	100	100	100	1	1	1
	0.4	100	100	100	1	1	1
	0.5	100	100	75	1	1	1
	0.6	100	100	100	1	1	1
	0.7	100	100	100	1	1	1
	0.8	100	100	100	1	1	1
	0.9	100	100	100	1	1	1
20	**0.1**	100	100	90.9	2.9	47	47
	0.2	100	100	88.9	1.1	1	1.1
	0.3	100	100	87.5	1	1	1
	0.4	83.3	100	83.3	1	1	1
	0.5	100	100	100	1	1	1
	0.6	100	100	100	1	1	1
	0.7	100	100	100	1	1	1
	0.8	100	100	100	1	1	1
	0.9	100	100	100	1	1	1
30	**0.1**	100	100	85.7	31.7	61.5	984
	0.2	100	100	100	8.8	9.4	9.4
	0.3	100	100	100	2.9	47	47
	0.4	100	100	100	1	1	1
	0.5	100	100	83.3	1	1	1
	0.6	100	100	80	1	16	16
	0.7	100	100	80	1	1	1
	0.8	100	100	75	1	1	1
	0.9	100	100	100	1	1	1
40	**0.1**	94.1	100	88.2	2006.3	2006.3	30094
	0.2	92.3	100	92.3	59.6	63.5	63.5
	0.3	100	100	90	9.8	9.8	156
	0.4	100	100	87.5	2.1	1.9	1.9
	0.5	100	100	100	0.5	16	1

	0.6	100	100	100	0.5	16	1.1
	0.7	100	100	100	0.5	1	16
	0.8	100	100	100	1	1	1
	0.9	100	100	100	1	0.9	3
50	**0.1**	94.7	100	89.5	10951.6	21218.8	22633.3
	0.2	100	100	85.7	160.8	160.8	160.8
	0.3	90.9	100	90.9	18.6	38.5	18.6
	0.4	100	100	70	4.5	9.4	4.5
	0.5	100	100	87.5	1.5	1.5	3.1
	0.6	100	100	100	0.5	1.1	1.1
	0.7	100	100	100	0.5	1.1	1.1
	0.8	100	100	100	0.5	1.1	1
	0.9	100	100	100	0.5	0.9	1
60	**0.1**	UN	UN	UN	234782	348387	675000
	0.2	93.8	100	87.5	673.9	989.8	1021.7
	0.3	90.9	100	90.9	40.6	61.5	40.6
	0.4	100	90	80	7.3	10.8	10.8
	0.5	87.5	100	87.5	1.3	2.5	2.5
	0.6	85.7	85.7	85.7	0.7	1	1
	0.7	100	100	100	0.5	1	1
	0.8	100	100	100	0.3	1	1.1
	0.9	100	100	75	0.5	0.5	0.5
70	**0.1**	UN	UN	UN	234783	229787	234783
	0.2	88.2	94.1	82.4	2402.7	3169.6	4805.5
	0.3	100	100	91.7	104.9	143.7	140.6
	0.4	90.9	100	72.7	13.1	17.3	17.3
	0.5	87.5	100	100	3.3	4.3	3.3
	0.6	85.7	100	85.7	1	0.8	1.4
	0.7	83.3	100	83.3	0.9	1.8	1.2
	0.8	100	100	100	0.8	1	1
	0.9	100	100	75	0.2	0.5	0.3
80	**0.1**	UN	UN	UN	138462	171429	174194
	0.2	88.2	94.1	94.1	6665.7	8252.7	6665.7
	0.3	85.7	92.9	78.6	186.2	279.2	222
	0.4	100	100	90	18.6	27.8	22.4
	0.5	100	100	100	3.5	4.2	5.2
	0.6	100	87.5	75	1.4	1.4	1.7
	0.7	100	100	83.3	0.9	1.1	1.1
	0.8	100	100	100	0.9	1.2	1.1
	0.9	100	100	100	0.3	0.3	0.3

90	**0.1**	UN	UN	UN	99082.6	138462	114894
	0.2	89.5	89.5	84.2	19474.6	22789.4	23034.4
	0.3	92.9	92.9	85.7	438.8	514.3	508.8
	0.4	100	100	81.8	36	41.7	41.7
	0.5	88.9	100	77.8	6.4	7.5	7.5
	0.6	100	100	87.5	1.7	2.4	2.4
	0.7	83.3	83.3	83.3	0.9	1.3	1.1
	0.8	100	100	100	1	1.2	1.1
	0.9	100	100	100	0.6	1	1
100	**0.1**	UN	UN	UN	86400	99082.6	98181.8
	0.2	89.5	94.7	84.2	46399	60020.8	60020.8
	0.3	86.7	86.7	80	747.7	843.4	967.2
	0.4	100	90.9	81.8	44.5	55.5	49.6
	0.5	90	90	80	8.6	9.6	9.6
	0.6	87.5	87.5	87.5	2	2.3	2.3
	0.7	100	100	100	1.1	1.2	1.4
	0.8	100	100	80	1	1.2	1.2
	0.9	100	100	75	0.1	0.2	0.2
200	**0.1**	UN	UN	UN	10964.5	12342.9	12572.8
	0.2	UN	UN	UN	11526.1	11332.6	10964.5
	0.3	UN	UN	UN	10964.5	10975.6	11513.9
	0.4	92.3	92.3	92.3	1145.9	1127.3	1183.8
	0.5	90.9	90.9	81.8	74.6	75.8	75.8
	0.6	80	90	70	10.5	10.5	10.5
	0.7	100	100	100	2	2.1	2
	0.8	100	100	83.3	1.4	1.4	1.4
	0.9	80	100	80	0.2	0.2	0.2
300	**0.1**	UN	UN	UN	3972	3995.6	4043.4
	0.2	UN	UN	UN	3638.8	3600	3638.8
	0.3	UN	UN	UN	3563.2	3527.1	3580.9
	0.4	UN	UN	UN	3582.1	3563.2	3600
	0.5	91.7	91.7	83.3	363.5	357.9	367.5
	0.6	88.9	100	88.9	31.4	31.4	32.1
	0.7	80	90	80	31.7	31.2	31.9
	0.8	83.3	100	100	1.3	1.3	1.4
	0.9	100	100	80	1	1	1.1
400	**0.1**	UN	UN	UN	1702.7	1681.7	1749.8
	0.2	UN	UN	UN	1578	1588.9	1614.8
	0.3	UN	UN	UN	1553.3	1549.7	1588.9
	0.4	UN	UN	UN	1567.3	1549.7	1578
	0.5	UN	UN	UN	1611.2	1553.3	1614.8

	0.6	81.8	90.9	81.8	82.7	82.9	84.3
	0.7	87.5	100	87.5	8.2	8	8.3
	0.8	100	100	100	1.3	1.3	1.4
	0.9	100	100	100	0.3	0.3	0.3
500	**0.1**	UN	UN	UN	867.3	855.4	880.5
	0.2	UN	UN	UN	816.1	813.2	826.8
	0.3	UN	UN	UN	799.1	800.9	819.9
	0.4	UN	UN	UN	812.2	799.1	823.9
	0.5	UN	UN	UN	826.8	823.9	845
	0.6	81.8	90.9	72.7	164.3	161.5	166.4
	0.7	77.8	88.9	88.9	13.6	13.4	13.9
	0.8	85.7	85.7	85.7	1.8	1.8	1.9
	0.9	100	100	80	0.3	0.3	0.3
600	**0.1**	UN	UN	UN	491.3	488.8	503.4
	0.2	UN	UN	UN	466.4	464.5	472.8
	0.3	UN	UN	UN	460.5	459	467.7
	0.4	UN	UN	UN	462.6	456.5	468.6
	0.5	UN	UN	UN	475.7	472.4	484
	0.6	UN	UN	UN	499.1	490.2	505.6
	0.7	88.9	88.9	77.8	18.3	18.3	18.8
	0.8	87.5	75	75	2.2	2.2	2.2
	0.9	100	100	100	0.3	0.3	0.3
700	**0.1**	UN	UN	UN	291.5	290.9	292.1
	0.2	UN	UN	UN	275.2	270.3	275.7
	0.3	UN	UN	UN	273	270.1	274.4
	0.4	UN	UN	UN	275.9	273.2	277
	0.5	UN	UN	UN	284	279.8	283.7
	0.6	UN	UN	UN	297	289.7	298.6
	0.7	88.9	100	88.9	26.3	26.1	26.8
	0.8	85.7	100	85.7	2.7	2.7	2.7
	0.9	100	83.3	100	0.4	0.4	0.4
800	**0.1**	UN	UN	UN	188.9	188.8	189.3
	0.2	UN	UN	UN	179.9	178.1	181.8
	0.3	UN	UN	UN	178.3	176.3	178.7
	0.4	UN	UN	UN	179.4	178.3	182.1
	0.5	UN	UN	UN	184.4	182.8	126.7
	0.6	UN	UN	UN	192.6	191.9	193
	0.7	88.9	100	88.9	37	36.7	37.5
	0.8	100	100	85.7	3.3	3.2	3.3
	0.9	100	100	100	2.9	2.9	2.9
900	**0.1**	UN	UN	UN	154	155.6	155

	0.2	UN	UN	UN	145	143.8	146.1
	0.3	UN	UN	UN	143.5	142	144.6
	0.4	UN	UN	UN	144.6	144.2	146.8
	0.5	UN	UN	UN	149.3	150.1	150.1
	0.6	UN	UN	UN	155.2	152.6	156.7
	0.7	UN	UN	UN	165	163.2	167.5
	0.8	100	100	85.7	4.2	4.2	4.3
	0.9	100	100	100	3.8	3.8	3.9
1000	**0.1**	UN	UN	UN	105.9	105.7	106.2
	0.2	UN	UN	UN	100.6	100.2	101.2
	0.3	UN	UN	UN	99.4	98.7	100.2
	0.4	UN	UN	UN	100.2	99.4	101.4
	0.5	UN	UN	UN	102.7	101	103.6
	0.6	UN	UN	UN	106.5	106.9	107.8
	0.7	UN	UN	UN	112.4	112.1	114.2
	0.8	87.5	87.5	75	5.1	5.1	5.2
	0.9	83.3	100	83.3	3	2.9	3

Table 4.3 Accuracy Summary for the sample (summary of table 6.2)

Wight Factor	Random	Minimum	Maximum
Accuracy	95.25 %	97.345 %	89.44 %

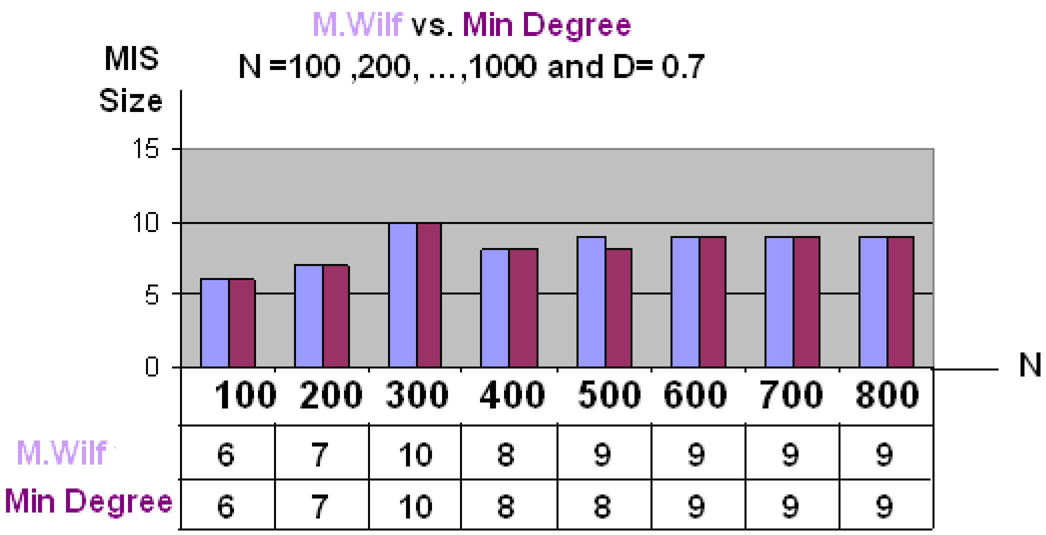


Figure 4.1: IS Size Comparison between M.Wilf and Minimum CA-MIS

Table 4.4 The ratio of optimization on the running time (R_t) Summary for the sample (summary of table 4.2)

Wight Factor	Random	Minimum	Maximum
Sum of Speedup	719,127	830,512	845,765

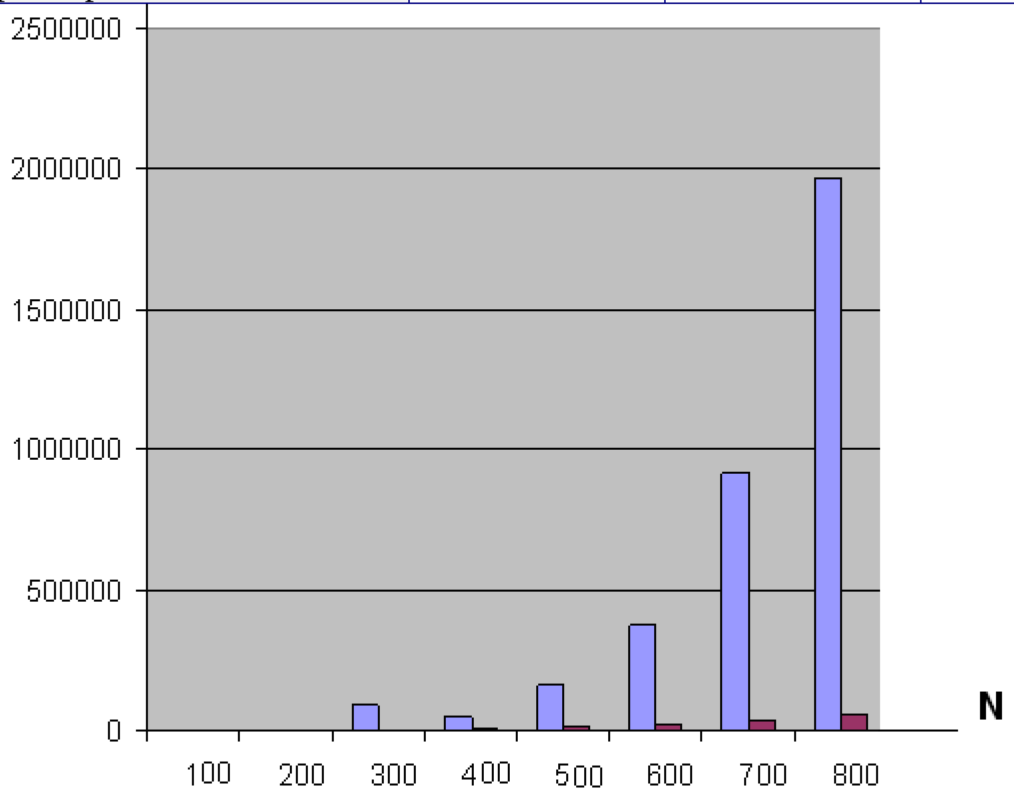


Figure 4.2: Speedup for Minimum CA-MIS over M.Wilf

Table 4.5 shows the results of the IS size and the running time for Wilf, M.Wilf and the proposed Approaches for Fixed density value (D=0.1) while changing the size of graph (100 , 200 , … , 1000) to study the effects of the small density value on the results. The results indicate that we can't obtain any results from Wilf and M.Wilf Algorithms for this density (D=0.1).

Table 4.5 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.1 and N=(100,200,..,1000)

D=0.1	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
100	-	∞	-	∞	26	125	28	109	25	110
200	-	∞	-	∞	32	985	35	875	31	859
300	-	∞	-	∞	38	2719	41	2703	36	2671
400	-	∞	-	∞	41	6343	44	6422	37	6172
500	-	∞	-	∞	45	12453	48	12625	42	12266
600	-	∞	-	∞	46	21984	50	22094	42	21453
700	-	∞	-	∞	46	37047	50	37125	45	36969
800	-	∞	-	∞	49	57172	53	57204	45	57062
900	-	∞	-	∞	51	70125	53	69407	47	69672
1000	-	∞	-	∞	50	102016	53	102187	48	101719

Tables (4.6, 4.7) show the results of the IS size and the running time for Wilf, M.Wilf and the proposed Approaches for Fixed density value (D=0.2,D=0.3) while changing the size of graph (100, 200, … , 1000) to study the effects of the small density value on the results. The results indicate that we can't obtain any results from Wilf and M.Wilf Algorithms for this density (D=0.2, D=0.3) when the size of the graph are 200 or more..

Table 4.6 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.2 and N=(100,200,..,1000)

D=0.2	**Wilf's**		**M. Wilf's**		**Random CA-MIS**		**Minimum CA-MIS**		**Maximum CA-MIS**	
N	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
100	19	12000000	19	6542265	17	141	18	109	16	109
200	-	∞	-	∞	21	937	22	953	20	985
300	-	∞	-	∞	23	2968	25	3000	22	2968
400	-	∞	-	∞	24	6844	25	6797	23	6688
500	-	∞	-	∞	25	13234	27	13281	24	13062
600	-	∞	-	∞	26	23156	28	23250	24	22843
700	-	∞	-	∞	27	39250	30	39953	27	39172
800	-	∞	-	∞	28	60031	29	60656	27	59406
900	-	∞	-	∞	29	74484	30	75109	28	73937
1000	-	∞	-	∞	29	107328	32	107828	28	106750

Tables (4.8 , 4.9 ,4.10) show the results of the IS size and the running time for Wilf, M.Wilf and the proposed Approaches for Fixed density value (D=0.4,0.3,0.5) while changing the size of graph (100, 200, … , 1000) to study the effects of the medium density value on the results. The results indicate that we can’t obtain any results from Wilf and M.Wilf Algorithms for medium density (D=0.4, 0.3, 0.5) when the size of the graph are 600 or more.

As the density of the graph becomes large (Dense), we can obtain the results from Wilf and M.Wilf Algorithms and the running time of these algorithms become equal to or less than the running time of the proposed algorithms.

Table 4.7 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.3 and N=(100,200,..,1000)

D=0.3	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	S	T	S	T	S	T	S	T	S	T
100	15	151578	15	105422	13	141	13	125	12	109
200	-	∞	-	∞	16	985	16	984	15	938
300	-	∞	-	∞	16	3031	17	3062	16	3016
400	-	∞	-	∞	18	6953	18	6969	17	6797
500	-	∞	-	∞	18	13515	20	13485	17	13172
600	-	∞	-	∞	21	23453	19	23531	17	23094
700	-	∞	-	∞	20	39563	20	39984	19	39359
800	-	∞	-	∞	20	60562	21	61265	19	60421
900	-	∞	-	∞	20	75250	21	76032	20	74672
1000	-	∞	-	∞	21	108641	22	109438	20	107781

Table 4.8 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.4 and N=(100,200,..,1000)

D=0.4	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	S	T	S	T	S	T	S	T	S	T
100	11	8546	11	6937	11	156	10	125	9	140
200	13	1508000	13	1110391	12	969	12	985	12	938
300	-	∞	-	∞	13	3015	13	3031	12	3000
400	-	∞	-	∞	14	6891	15	6969	12	6844
500	-	∞	-	∞	14	13297	15	13516	14	13109
600	-	∞	-	∞	15	23344	16	23656	14	23047
700	-	∞	-	∞	15	39141	16	39532	14	38984
800	-	∞	-	∞	15	60187	16	60578	15	59313
900	-	∞	-	∞	16	74703	16	74907	15	73547
1000	-	∞	-	∞	16	107735	17	108672	16	106515

Table 4.9 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.5 and N=(100,200,..,1000)

D=0.5	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	S	T	S	T	S	T	S	T	S	T
100	10	1422	10	1204	9	140	9	125	8	125
200	11	89234	11	71047	10	953	10	937	9	937
300	12	1442078	12	1073828	11	2954	11	3000	10	2922
400	-	∞	-	∞	11	6703	12	6953	11	6688
500	-	∞	-	∞	11	13062	12	13109	10	12781
600	-	∞	-	∞	11	22703	12	22860	11	22313
700	-	∞	-	∞	12	38031	13	38593	12	38062
800	-	∞	-	∞	13	58563	13	59094	12	85219
900	-	∞	-	∞	13	72360	13	71953	11	71968
1000	-	∞	-	∞	12	105172	13	106907	13	104265

Table 4.10 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.6 and N=(100,200,..,1000)

D=0.6	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	S	T	S	T	S	T	S	T	S	T
100	8	328	8	282	7	140	7	125	7	125
200	10	11328	10	9500	8	906	9	907	7	907
300	9	109844	9	89797	8	2860	9	2859	8	2797
400	11	688563	11	534813	9	6469	10	6454	9	6344
500	11	2702500	11	2046219	9	12453	10	12672	8	12297
600	-	∞	-	∞	9	21641	10	22031	9	21359
700	-	∞	-	∞	10	36360	11	37281	9	36172
800	-	∞	-	∞	10	56078	11	56282	10	55969
900	-	∞	-	∞	10	69578	10	70781	10	68922
1000	-	∞	-	∞	10	101375	10	101063	10	100203

Table 4.11 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.7 and N=(100,200,..,1000)

D=0.7	**Wilf's**		**M. Wilf's**		**Random CA-MIS**		**Minimum CA-MIS**		**Maximum CA-MIS**	
N	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
100	6	109	6	94	6	140	6	125	6	109
200	7	2000	7	1734	7	875	7	844	7	859
300	10	110579	10	89250	8	2813	10	2859	8	2797
400	8	59547	8	49578	7	6047	8	6188	7	5953
500	9	202297	9	160328	7	11769	8	12000	8	11500
600	9	467485	9	377062	9	20594	9	20578	7	20093
700	9	1151188	9	914703	8	34719	9	35078	8	34140
800	9	2541610	9	1970406	8	53187	9	53657	8	52563
900	-	∞	-	∞	8	65453	9	66187	8	64485
1000	-	∞	-	∞	8	96047	8	96343	8	94547

Table 4.12 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.8 and N=(100,200,..,1000)

D=0.8	**Wilf's**		**M. Wilf's**		**Random CA-MIS**		**Minimum CA-MIS**		**Maximum CA-MIS**	
N	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
100	5	47	5	47	5	125	5	110	4	110
200	6	1324	6	1092	6	797	6	781	5	766
300	6	2515	6	2234	5	2453	6	2516	6	2360
400	6	8578	6	7297	6	5579	6	5562	6	5390
500	7	24078	7	19922	6	10891	6	10937	6	10672
600	8	49422	8	41203	7	18922	6	19062	6	18734
700	7	105594	7	85344	6	31828	7	32015	6	31531
800	7	198484	7	159141	7	48937	7	49172	6	48218
900	7	317734	7	255234	7	60931	7	61312	6	59906
1000	8	566063	8	457172	7	89062	7	88969	6	88391

Table 4.13 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches

Fixed D=0.9 and N=(100,200,..,1000)

D=0.9	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
N	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**	**S**	**T**
100	4	16	4	16	4	125	4	93	3	93
200	5	156	5	156	4	672	5	672	4	657
300	5	625	5	562	5	2156	5	2187	4	2047
400	5	1688	5	1438	5	4860	5	4797	5	4766
500	5	3547	5	3015	5	9421	5	9344	4	9250
600	5	6218	5	5203	5	16640	5	16594	5	16282
700	6	12375	6	10297	6	27704	5	28640	6	27422
800	5	20625	5	16969	5	43547	5	43625	5	43016
900	5	32360	5	26594	5	53219	5	53469	5	51875
1000	6	50297	6	41172	5	137765	6	140454	5	136219

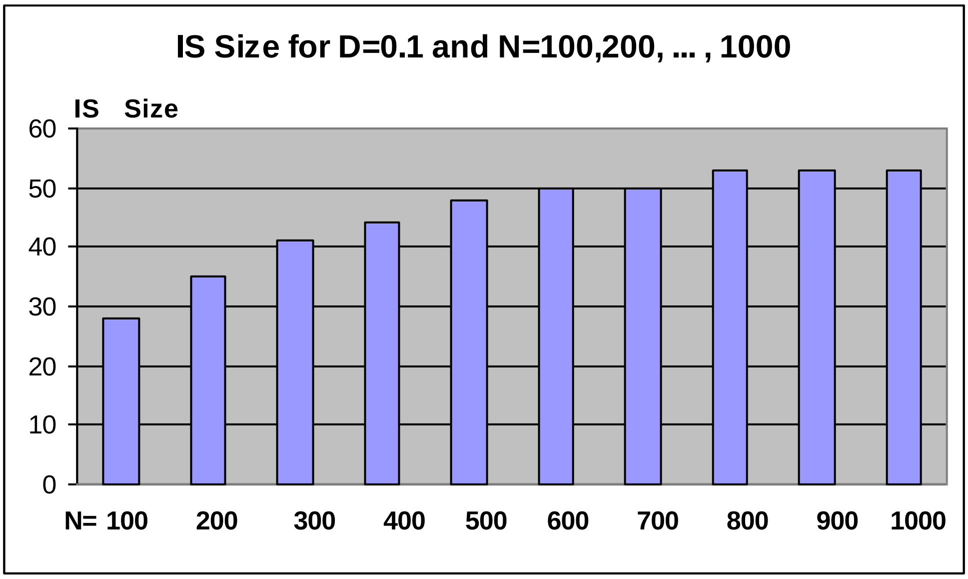


Figure 4.3 : IS Size for small D(0.1)

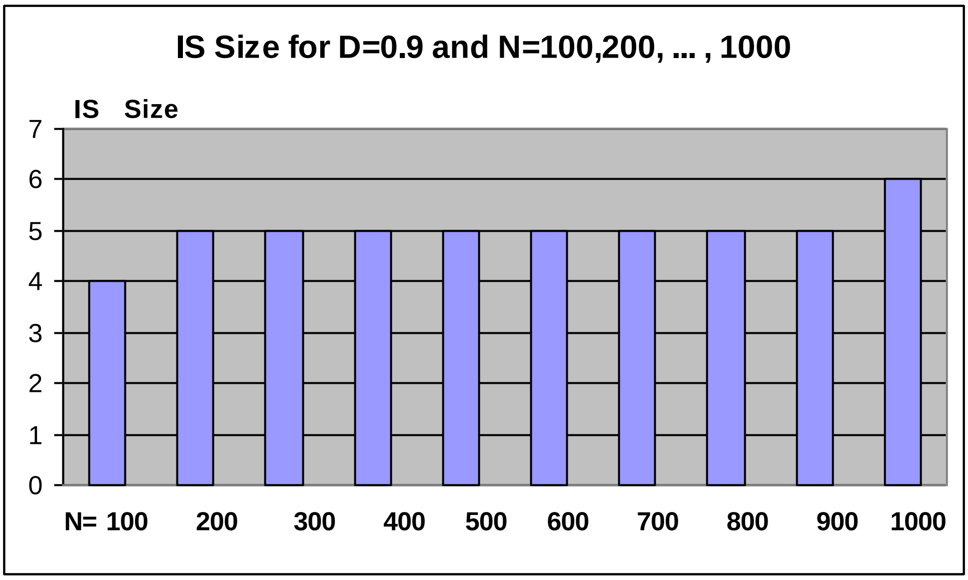


Figure 4.4 : IS Size for large D(0.9)

Tables (4.14 to 4.22) show the Experimental Results for Wilf, M.Wilf and the proposed Approaches while changing the density D from 0.1 to 0.9 and N=100 for three experiments to take the average .

Table 4.14 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.1 and N=100 for three experiments

Exp. No.	D=0.1	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	**100**	-	∞	-	∞	26	125	28	109	25	110
2	**100**	-	∞	-	∞	27	359	28	219	25	219
3	**100**	-	∞	-	∞	26	312	28	218	25	218
Average						**26**	**265**	**28**	**182**	**25**	**182**

Table 4.15 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.2 and N=100 for three experiments

Exp. No.	D=0.2	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	**100**	19	12000000	19	6542265	17	141	18	109	16	109
2	**100**	19	11976864	19	5442288	18	250	18	218	16	218
3	**100**	19	13478364	19	5788265	18	234	18	219	16	218
Average		19	12485076	19	5924273	18	208	18	182	16	182

Table 4.16 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.3 and N=100 for three experiments

Exp. No.	D=0.3	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	**100**	15	151578	15	105422	13	141	13	125	12	109
2	**100**	15	134031	15	96765	12	250	13	234	12	234
3	**100**	15	128281	15	95329	13	250	13	219	12	219
Average		15	137963	15	99172	13	214	13	193	12	187

Table 4.17 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.4 and N=100 for three experiments

Exp. No.	D=0.4	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	11	8546	11	6937	11	156	10	125	9	140
2	100	11	7750	11	6406	11	234	10	219	9	235
3	100	11	7437	11	6312	11	250	10	218	9	234
Average		11	7911	11	6552	11	213	10	187	9	203

Table 4.18 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.5 and N=100 for three experiments

Exp. No	D=0.5	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	10	1422	10	1204	9	140	9	125	8	125
2	100	10	1281	10	1125	9	250	9	250	8	234
3	100	10	1250	10	1109	8	250	9	235	8	219
Average		10	1318	10	1146	9	213	9	203	8	193

Table 4.19 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.6 and N=100 for three experiments

Exp. No	D=0.6	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	8	328	8	282	7	140	7	125	7	125
2	100	8	297	8	266	8	250	7	219	7	219
3	100	8	282	8	266	7	250	7	235	7	219
Average		8	302	8	271	7	213	7	193	7	188

Table 4.20 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.7 and N=100 for three experiments

Exp. No	D=0.7	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	6	146	6	130	6	133	6	121	6	109
2	100	6	141	6	143	6	123	6	111	6	117
3	100	6	155	6	128	6	141	6	117	6	126
Average		6	147	6	134	6	132	6	116	6	117

Table 4.21 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.7 and N=100 for three experiments

Exp. No	D=0.8	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	5	133	5	125	5	88	5	89	4	78
2	100	5	123	5	118	5	111	5	93	4	112
3	100	5	128	5	121	4	90	5	87	4	99
Average		6	128	5	121	5	96	5	90	4	96

Table 4.22 Sequential Experimental Results for Wilf, M.Wilf and the proposed Approaches for Fixed D=0.9 and N=100 for three experiments

Exp. No	D=0.9	Wilf's		M. Wilf's		Random CA-MIS		Minimum CA-MIS		Maximum CA-MIS	
	N	S	T	S	T	S	T	S	T	S	T
1	100	4	94	4	88	4	125	4	93	3	93
2	100	4	99	4	93	4	266	4	172	3	172
3	100	4	89	4	83	4	203	4	188	3	172
Average	Average	6	94	4	88	4	198	4	151	3	146

Table 4.23 Minimum CA-MIS (Parallel vs. Sequential Speedup)

		Minimum CA-MIS	Parallel	Speedup Ratio
N	**D**	**Time**	**Time**	**Sp**
1000	**0.1**	102187	76832	1.33
	0.2	107828	88384	1.22
	0.3	109438	96848	1.13
	0.4	108672	90560	1.2
	0.5	106907	89089	1.2
	0.6	101063	91048	1.11
	0.7	96343	71898	1.34
	0.8	88969	68438	1.3
	0.9	140454	117045	1.2

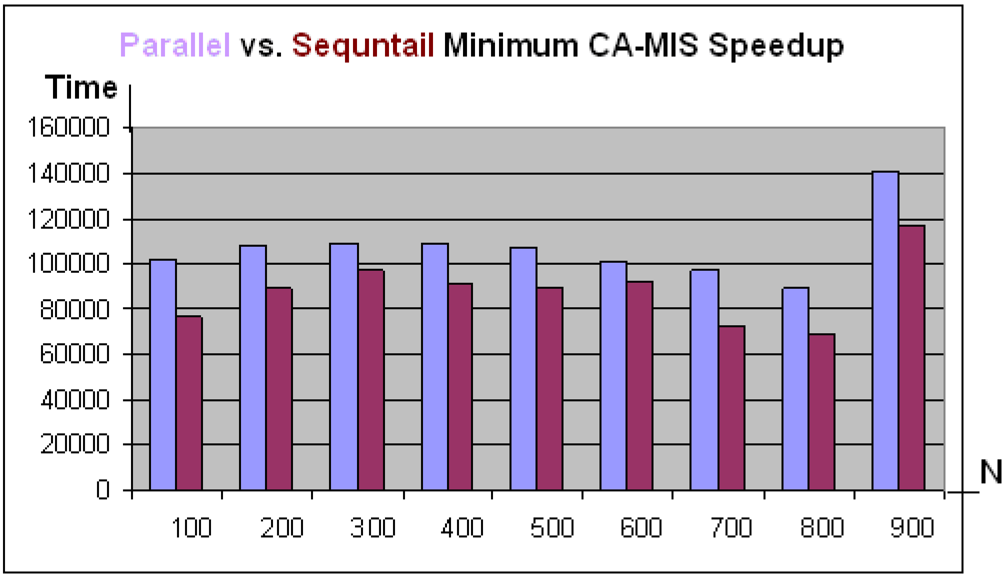


Figure 4.5: Parallel vs. Sequential Minimum CA-MIS Speedup

For N=1000 and D=(0.1,0.2,…,0.9)

Chapter 5
Conclusion and Future Work

5.1 Conclusion

Finding an exact MIS in a graph is NP-hard problem (Dharwadker, 2006; Kako, 2004). It is worst for small density in general. Thus, it is strongly predicted that no polynomial time algorithm can find optimal solution. So, approximation or heuristic algorithms and parallel implementation are important to have polynomial run time that can find solutions closed to the optimal one (Kako, 2004; Back and Khuri, 1994).

This Thesis proposed a new Heuristic Cellular Automata Algorithms based on Weighted Factors for finding a MIS in a Graph. The algorithms use cellular automata and heuristic approaches by choosing the candidate nodes in MIS. The node can be selected in randomly, node with maximum degree, and node with minimum degree. To speedup the process of finding MIS and in order to find all possible MIS's from different starting points, parallel implementations of the sequential algorithms using Multithreads was investigated.

The major contribution of this thesis is the devised of heuristic algorithms that can be used to find exact or approximation of a MIS in a graph with large size and small densities in acceptable running time. Those algorithms were analyzed and tested

This thesis starts by introducing the Study Problem, Study Methodology and Related Studies. After that it introduce the Theoretic Definitions, Notations, and possible approaches for solving the MIS problem. And gives an example of the Exact, Approximation and Heuristic Approaches for Solving MIS Problem.

And an overview about Multithreading Environment and Multiprocessing Environment. After that it introduces The Proposed Heuristic Cellular Automata Algorithms based on Weighted and the Experimental Results of proposed algorithms and compares the results with the exact algorithms (Modified Wilf).

The results indicate that the Minimum degree factor algorithm is the most accurate algorithm (97.35%) in general, followed by the Random Degree algorithm in the second level (95.25%), and finally the Maximum degree factor algorithm (89.44 %). The proposed algorithms have less CPU run time than Wilf's and modified Wilf's. When we change the size of Graph (N) and make the Graph density D fixed, we found that the size of MIS (i(G)) is increases as N increases. The sizes of MIS's increases as the density of the graph decreases. When density increases the size of MIS become fixed regardless the value of N. For example, when D=0.9 the i(G) of the Graph with N=1000 is almost equal to the i(G) of the Graph with N=100. The experimental results indicated that for a fixed density the sizes of the generated MIS's are very close. For example, the size of the generated MIS's for density = 0.9 remains five nodes as the sizes change from 400 through 1000.

The parallel implementation of the proposed algorithm using Multithreading does not appear to be great performance over the sequential one for the suggested samples, and it could be more suitable for graphs with large size (ex. 10,000) and on multiprocessors or multicomputers environment.

5.2 Future Work

1. Implementing, Running and Testing the Parallel Approach on multi computer and multi processing environment.
2. Use this approach to investigate other applications.
3. Taken into consideration the nodes position in addition to the node degree, for example if the node in the middle between two nodes, this node shouldn't be odd because it will remove the other nodes.
4. Using Data Mining and Artificial Intelligence (AI) to add more Intelligence to this technique.

REFERENCES

Akhter S., Roberts J. (2007).**"Apply the Fundamentals of Parallel Programming to Multiprocessor Designs"**: Part 1. Retrieved October 8, 2007 from **http://www.embedded.com/columns/technicalinsights/199700235?_requestid=801258**

Al-Ghwari A. (2006).**"Parallel Implementation of Algorithms Based on Weight Factors for Maximum Independent Set in a Graph"**. Master Thesis, University of Jordan, Amman, Jordan.

Al-Jaber A.,Sharieh A. (2000).**"Algorithms Based on Weighted Factors for Maximum Independent Set. Dirasat**", Pure Sciences, Volume 27 (1),pure science no.1 P74-91.Februry 2000.

Approximation. Retrieved October 8, 2007 from **http://en.wikipedia.org/wiki/Approximation_algorithm**

Back T., Khuri S. (1994). **"An Evolutionary Heuristic for the Maximum Independent Set Problem"**. Retrieved October 10, 2007 from **"http://citeseer.ist.psu.edu/83286.html"**

Bader D., Moret B, Sanders P.(2002).**"Algorithm Engineering for Parallel Computation"**.Retrieved October 14,2007 from **"http://lcbb.epfl.ch/~moret/dagstuhl2.pdf"**

Butenko S. (2003). **"Maximum Independent Set and Related Problems, with Applications"**. Retrieved October 10, 2007 from **"http://etd.fcla.edu/UF/UFE0001011/butenko_s.pdf"**

Complexity_classes_P_and_NP. Retrieved April 28, 2007 from **"http://en.wikipedia.org/wiki/Complexity_classes_P_and_NP"**

Cormen T,Leiserson C,Rivest R and Stein C.(2001).**"Introduction to Algorithms"**, Second Edition. Retrieved October 14, 2007 from **"http://www.amazon.ca/Introduction-Algorithms-Second-Thomas-Cormen/dp/0262032937"**

Czumaj A., Diks K., Przytycka T. (1998). **”Parallel Maximum Independent Set in Convex Bipartite Graphs”**. Retrieved October 15, 2007 from **"http://citeseer.ist.psu.edu/61811.html"**

Dharwadker A. (2006). **“The Independent Set Algorithm”** .Retrieved October 15, 2007 **from "http://www.geocities.com/dharwadker/independent_set"**

Gebremedhin A. (1999). **“Parallel Graph Coloring”**. Thesis, University of Bergen, Norway.

Gfeller B., Vicari E. (2007).**”A Faster Distributed Approximation Scheme for the Connected Dominating Set Problem for Growth-Bounded Graphs”**. Conference on Ad-Hoc, Mobile, and Wireless Networks, 2007, pp 59-73.

Grama A., Gupta A., Karypis G., Kumar V.(2003).**”Introduction to Parallel Computing, Principles of Parallel Algorithm Design”**. 2nd Edition .Retrieved October 20, 2007 from “**http://www-users.cs.umn.edu/~karypis/parbook/Lectures/GK-CS5451/Chapter 3- Principles of Parallel Algorithm Design.pdf”.**

Graph. Retrieved October 8, 2007 from **"http://en.wikipedia.org/wiki/Graph_%28mathematics%29"**

Heuristic. Retrieved October 8, 2007 from **“http://en.wikipedia.org/wiki/Heuristic”**

Hochberger C., Hoffmann R.(1996).**”Solving Routing Problems with Cellular Automata”**. Retrieved October 1, 2007 from **"http://citeseer.ist.psu.edu/hochberger96solving.html"**

Independent Set. Retrieved November 1, 2007 from **“http://en.wikipedia.org/wiki/Independent_set”**

Independent Set Problem. Retrieved November 1, 2007 from **“http://en.wikipedia.org/wiki/Independent_set_problem”**

Kako A. (2004).**”Approximation Algorithms for the Weighted Independent Set Problem”**. Retrieved November 1, 2007 from **“http://www.is.nagoya-u.ac.jp/thesis/M2004/cm/M350301050e.pdf”**

Kuhn F., Moscibroda T., Nieberg T, and Wattenhofer R.,(2005),**"Fast Deterministic Distributed Maximal Independent Set Computation on Growth-Bounded Graphs"** , Retrieved November 1, 2007 from **"http://www.springerlink.com/content/2jfvwkrm6vtuxpxe/fulltext.pdf"**

Multiple Instruction stream, Multiple Data stream. Retrieved February 20, 2008 from **"http://en.wikipedia.org/wiki/MIMD"**

Multithreading. Retrieved February 14, 2008 from **"http://en.wikipedia.org/wiki/Multithreading"**

Multi-Threading in a Java Environment. Retrieved February 15, 2008 from **"http://www.devx.com/go-parallel/Article/28911"**

Niesche H. (2006),**"Introduction to cellular automata"**, Seminar "Organic Computing". Retrieved October 1, 2007 from **"http://kbs.cs.tu-berlin.de/teaching/sose2006/oc/folien/CellularAutomataPaper.pdf"**

NP-Complete Theory .Retrieved April 28, 2007 from **"http://www.seas.gwu.edu/~ayoussef/cs212/ NP-Complete Theory.html"**

Parallel Computing .Retrieved February 17, 2008 from **"http://en.wikipedia.org/wiki/Parallel_computing"**

Parallel Processing Concepts. Retrieved February 17, 2008 from **"http://mpc.uci.edu/wget/www.tc.cornell.edu/Services/Edu/Topics/ParProgCons/Parallel Processing Concepts.htm"**

P=NP problem. Retrieved October 8, 2007 from **"http://en.wikipedia.org/wiki/P_%3D_NP_problem"**

Reduction Complexity .Retrieved February 14, 2008 from **"http://en.wikipedia.org/wiki/Reduction_complexity"**

Single Instruction stream, Multiple Data stream. Retrieved February 20, 2008 from **“http://en.wikipedia.org/wiki/SIMD”**

Tanenbaum A (1992).**” Distributed Modern Operating Systems”**. 2nd Edition .Retrieved October 20, 2007 from **"http://portal.acm.org/citation.cfm?id=6074 "**

Threading. Retrieved February 14, 2008 from **“http://en.wikipedia.org/wiki/Threading”.**

Weisstein, Eric W.(2008).**”Cellular Automaton”**. Retrieved February 17, 2008 from **“http://mathworld.wolfram.com/CellularAutomaton.html “.**

Wilf H. S. (1994) .**”Algorithms and Complexity”**. Retrieved November 8, 2007 from **"http://www/cis.upenn.edu/wilf"**

Wilkinson B, Allen M.(1998). **“Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers”**. Prentice Hall

Appendices

Program Name	Program Description	Page No.
MainCAMIS	**This program is the Main Program that is responsible to run all the implemented algorithms (Wilf, Modified Wilf, Sequential Proposed Algorithms and the Parallel Proposed Algorithm).**	76
RanAtGraph	**This program is the Program that generates all the random graphs with certain size and density.**	79
RanModCAMIS	**This program is the implementation of the proposed Sequential Algorithms (Minimum Degree, Maximum Degree and the Random Factors).**	83
ThreadClient	**This program is the implementation of Proposed Parallel Algorithm.**	96
MWilf	**This program is the implementation of Modified Wilf Algorithm**	101
Wilf	**This program is the implementation of Wilf Algorithm.**	106
MaxTest	**This program is the Program that responsible to return the node with Minimum Degree, Maximum Degree and the Randomly.**	111
WriteToFile	**This program is the Program that responsible to deal with the files on Operating system to save the results .**	115
Node	**This program is used to represents the Node in graph.**	117

Program Name	**MainCAMIS**
Program Description	**This program is the Main Program that is responsible to run all the implemented algorithms (Wilf, Modified Wilf, Sequential Proposed Algorithms and the Parallel Proposed Algorithm).**

```
package CA_MIS;

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;

public class MainCAMIS {

	public static void main(String[] args) throws IllegalArgumentException, IOException,
ClassNotFoundException, InstantiationException, IllegalAccessException,
InvocationTargetException {
		RanModCAMIS rmis=new RanModCAMIS();
		Wilf ya=new Wilf();
		MWilf mw=new MWilf();
		ThreadClient tc=new ThreadClient();
		DataOutputStream out =	new DataOutputStream(new
BufferedOutputStream(new FileOutputStream("C:\\Results\\CA-Results.doc")));
		DataOutputStream wout =  new DataOutputStream(new
BufferedOutputStream(new FileOutputStream("C:\\Results\\Wilf-Results.doc")));
		DataOutputStream mwout =new DataOutputStream(new
BufferedOutputStream(new FileOutputStream("C:\\Results\\MWilf-Results.doc")));
		WriteToFile w=new WriteToFile();


		for(int n=10;n<=100;n=n+10){
			for(int d=1;d<5;d++){
				System.out.println("PP-MIS is starting for N= "+n +" and D="+d
+"\n");
			tc.CAMISMain(n,d,w,wout);
				System.out.println("CA-MIS is starting for N= "+n+" and D=
"+d);
				rmis.CAMISMain(n,d,w,out);
				System.gc();
				System.out.println("M.Wilf is starting for N= "+n+" and D=
"+d);
				mw.CAMISMain( n,d,w,mwout);
				System.gc();
				System.out.println("Wilf is starting for N= "+n+" and D= "+d);
				ya.CAMISMain( n,d,w,wout);
				System.gc();
			}
		}




		for(int n=200;n<=1000;n=n+100){
			for(int d=1;d<5;d++){
```

```
                                System.out.println("PP-MIS is starting for N= "+n +" and D="+d
+"\n");
                           tc.CAMISMain(n,d,w,wout);
                                System.out.println("CA-MIS is starting for N= "+n+" and D=
"+d);
                                rmis.CAMISMain(n,d,w,out);
                                System.gc();
                                System.out.println("M.Wilf is starting for N= "+n+" and D=
"+d);
                                mw.CAMISMain( n,d,w,mwout);
                                System.gc();
                                System.out.println("Wilf is starting for N= "+n+" and D= "+d);
                                ya.CAMISMain( n,d,w,wout);
                                System.gc();
                        }
                }


        w.closeOutStr(out);
        w.closeOutStr(wout);
        w.closeOutStr(mwout);
        }
}
```

Program Name	**RanAtGraph**
Program Description	**This program is the Program that generates all the random graphs with certain size and density.**

```
package CA_MIS;

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;

public class RanAtGraph {

	String S = "";

	String Nodes(int n) {
		return S.substring(0, n);
	}

	char checkIfConnected(String a, String b) {
		char a2 = 'n';
		String t;
		int e;

		if (b.length() != 0) {

			for (int y = 0; y < a.length(); y = y + 3) {

				t = a.substring(y, y + 3);
				for (int s = 0; s < b.length(); s = s + 3) {
					e = b.substring(s, s + 3).compareTo(t);
					if (e == 0) {
						a2 = 'y';
						return a2;
					}
				}

			}
		}
		return a2;

	}

	public static void main(String[] args) throws IOException {
		RandomNo t = new RandomNo();
		String ret;
		int noOfNodes = 80;
		int GD = 5;
		double D = 0.34;
		int cs;
		DataOutputStream out = new DataOutputStream(new BufferedOutputStream(
		new FileOutputStream("D:\\eclipse\\workspace\\CA_MIS\\src\\CA_MIS\\G_"
```

```
                                                        + noOfNodes + "_" + GD +
".java")));
            int sizeOfNodes = noOfNodes * 4;
            String[] arrOfStr = new String[noOfNodes];
            for (int q = 0; q < noOfNodes; q++)
                  arrOfStr[q] = "";
            RanAtGraph rg = new RanAtGraph();
            String h1 = "";
            WriteToFile w = new WriteToFile();
            w.outStr(out, "package CA_MIS;\n");
            w.outStr(out, "class G_" + noOfNodes + "_" + GD + " {\n\n");
            int E = 0;
            for (int g = 0; g < noOfNodes; g++) {
                  if (g < 10)
                        h1 = "00" + g;
                  else if (g > 9 && g < 100)
                        h1 = "0" + g;
                  else
                        h1 = "" + g;
                  rg.S = rg.S + "@" + h1;
            }
            int ref3;
            int si = 0;
            String s;
            w.outStr(out, "public RanModCAMIS initGraph(RanModCAMIS util){\n");
            w.outStr(out, "  util.InitMIS(util);\n");
            w.outStr(out, "  String Nodes =\"" + rg.S + "\";\n");
            w.outStr(out, "  util.node=new Node[" + noOfNodes + "];\n");
            w.outStr(out, "  for(int i=0;i<" + noOfNodes + ";i++){\n");
            w.outStr(out, "     util.node[i]=new Node();\n");
            w.outStr(out, "  }\n\n");
            System.out.println("Nodes are " + rg.S + " with lenght "+ rg.S.length());

            ret = rg.Nodes(sizeOfNodes);
            Random r = new Random(noOfNodes);
            Random r1 = new Random(noOfNodes);

            for (int j = 0; j < ret.length(); j = j + 4) {
                  for (int y = 0; y <= D * noOfNodes; y++) {
                        int ref2 = (int) t.rand(1, noOfNodes - 1);
                        int ff = ref2 * 4 + 4;
                        char x = rg.checkIfConnected(ret.substring((ref2 * 4) + 1,
                                    (ref2 * 4) + 4), arrOfStr[j / 4]);
                        cs = rg.S.substring(j + 1, j + 4).compareTo(ret.substring(ref2 * 4 +
1, ref2 * 4 + 4));
                        if ((cs != 0) && (x != 'y')) {
                              arrOfStr[j / 4] = arrOfStr[j / 4]
                                          + ret.substring(ref2 * 4 + 1, ref2 * 4 + 4);
                              arrOfStr[ref2] = arrOfStr[ref2]
                                          + rg.S.substring(j + 1, j + 4);
```

```
                    }
                }

            }

            for (int g = 0; g < ret.length(); g = g + 4) {
                s = rg.S.substring(g + 1, g + 4);
                si = (Integer.valueOf(s)).intValue();
                w.outStr(out, "  util.node[" + si + "].Degree="+ arrOfStr[g / 4].length() /
3 + ";");
                w.outStr(out, "\n");
                w.outStr(out, "  util.node[" + si + "].Nigh=" + "\""+ arrOfStr[g / 4] +
"\";");
                w.outStr(out, "\n");
                w.outStr(out, "\n");
                E = (E + arrOfStr[g / 4].length() / 3);
            }
            w.outStr(out, "//  String D=\"" + D + " E =" + E + " and N = "+ noOfNodes + "
and  D= " + (float) E
                    / (noOfNodes * (noOfNodes - 1)) + "\";");
            System.out.println("E = " + E + " and N = " + noOfNodes + "  and  D= "+
(float) E / (noOfNodes * (noOfNodes - 1)));
            w.outStr(out, "\nreturn util; \n}");
            w.outStr(out, "\n}");
            w.closeOutStr(out);
        }
}
```

Program Name	**RanModCAMIS**
Program Description	**This program is the implementation of the proposed Sequential Algorithms (Minimum Degree, Maximum Degree and the Random Factors).**

```
package CA_MIS;

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.Calendar;

public class RanModCAMIS {

        MaxTest t=new MaxTest();
            RanModCAMIS util;
            static String j="";
            String Q0 = "";
            String Q1 = "";
            String Q2 = "";
            String Addressed = "";
            String ParentName;
            int x=0;
            String MISRes;
            public int Degree, Value, ParentNo;
            public String Nigh = "";
            String Odd = "";
            String MIS = "";
            //int noOfNodes=60;
            int[] MISArr;
            public Node[] node;

            void addToMIS(String mis) {
                  MIS = MIS + mis;
            }

            char checkIfConnected(String a, String b) {

                  if (b.indexOf(a) == -1)
                        return 'n';
                  else
                        return 'y';
            }

            char checkIfConnectedToOdd(String a, String b) {
                  char a2 = 'n';
                  if (b.length() != 0) {
                        for (int y = 0; y < a.length(); y=y+3) {
                              if (b.indexOf(a.substring(y, y + 3)) != -1) {
                                    a2 = 'y';
```

```
                                        break;
                                } else
                                        a2 = 'n';
                        }
                }
                return a2;
        }

        void getNodeInfo(RanModCAMIS u1,String nodeName) {
                int g=(Integer.valueOf(nodeName)).intValue();
                Degree=u1.node[g].Degree;
                Nigh=u1.node[g].Nigh;
        }

        void setNodeInfo(RanModCAMIS u1,String nodeName, int val, String par, int
parNo) {
                int g=(Integer.valueOf(nodeName)).intValue();
                u1.node[g].value = val;
                u1.node[g].ParentName = par;
                u1.node[g].ParentNo = parNo;
        }


        public void AddNodeToOdd(String Nodes) {
                Odd = Odd +"@"+ Nodes;
        }

        public void AddNodeToAddressed(String Nodes) {
                for(int y=0;y<Nodes.length();y=y+3)
                        Addressed = Addressed + "@"+Nodes.substring(y,y+3);

        }

        public String CheckIfNodeAddressed(String Nodes) {
                String t = "";
                int e;
                String notAddressed = "";
                for (int i = 0; i < Nodes.length(); i=i+3) {
                        t = Nodes.substring(i, i + 3);
                        e = Addressed.indexOf(t);
                        if (e == -1)
                                notAddressed = notAddressed + t;

                }
                return notAddressed;
        }

        public void QueueNodesToQ0(String Nodes) {
                for(int y=0;y<Nodes.length();y=y+3)
                  Q0 = Q0 + "@"+Nodes.substring(y,y+3);
```

```
                //System.out.println("Q0 is "+Q0);
                //Q0=Q0+Nodes;

        }


        String GetNodeFromQ0() {
                String q1c = "";
                if (Q0.length() == 0) {
                        return "@";
                }
                q1c = Q0.substring(1, 4);
                Q0 = Q0.subSequence(4, Q0.length()).toString();
                return q1c;
        }


        public void InitMIS(RanModCAMIS util) {

            util.Q0 = "";
                 util.Q1 = "";
                 util.Q2 = "";
                 util.Addressed = "";
                 util.ParentName="";
                 util.Nigh = "";
                 util.Odd = "";
                 util.MIS = "";
        }

        void setOddNodeInfo(RanModCAMIS u, String retNodeQ2,int temp1,String
retNodeQ1){
                u.setNodeInfo(u,retNodeQ2, temp1 + 1,retNodeQ1, temp1);
                u.addToMIS(retNodeQ2);
                u.AddNodeToOdd(retNodeQ2);
                u.AddNodeToAddressed(retNodeQ2);
        }

        void setEvenNodeInfo(RanModCAMIS u, String retNodeQ2,int temp1,String
retNodeQ1){
                u.setNodeInfo(u,retNodeQ2, temp1 + 2,retNodeQ1, temp1);
        }
        void setEvenNodesInfo(RanModCAMIS u, String retNodeQ2,int temp1,String
retNodeQ1){
                for(int i=0;i<retNodeQ2.length();i=i+3){
                        u.setNodeInfo(u,retNodeQ2.substring(i,i+3), temp1 +
2,retNodeQ1, temp1);
                        }
        }
        String GetNodeFromQ0(int inxc) {
                String q1c = "";
```

```
                    if (Q0.length() == inxc-1) {
                            return "@";
                    }
                    return Q0.substring(inxc+1,inxc+4);
            }


            public String CheckIfNodeQueuedQ0(String Nodes) {
                    String t = "";
                    int e;
                    String notAddressed = "";
                    for (int i = 0; i < Nodes.length(); i=i+3) {
                            t = Nodes.substring(i, i + 3);
                            e = Q0.indexOf(t);
                            if (e == -1)
                                    notAddressed = notAddressed+ t;

                    }
                    return notAddressed;
            }


            int maxMIS(String[] misArr){
                    int A=misArr[0].length();
                    int index=0;

                    for(int m1=1;m1<misArr.length;m1++){
                            if(misArr[m1].length() > A){
                                    index=m1;
                                    A=misArr[m1].length();
                            }

                    }

                    return index;
            }

            void makeGNStr(RanModCAMIS w){

                    int inx;
                    String n="";
                    int g1;
                    for(int i=0;i<w.Nigh.length();i=i+3){
                            inx = Q0.indexOf(w.Nigh.substring(i, i + 3));
                            if (inx != -1){
                                    int g=(Integer.valueOf(w.Nigh.substring(i,i+3))).intValue();
                                    Q0 = Q0.substring(0, inx-1) + Q0.substring(inx + 3,
Q0.length());
                            }
                    }
```

```
                }
                void makeGNStr(String m){
                        int inx;
                        inx = Q0.indexOf(m);
                        if (inx != -1)
                                        Q0 = Q0.substring(0, inx-1) + Q0.substring(inx + 3,
Q0.length());
                        }



        void RanMIS(int g,RanModCAMIS u, String startNode,String xQ0) {
                int inxm=0;
                int z=0;
                String w = "";
                String notAddNodes;
                String retNodeQ0;
                String retNodeQ1;
                String retNodeQ2;
                u.InitMIS(u);
                int cnt = 0;
                int cnt2;
                Q0=xQ0;
                retNodeQ0=startNode;
                int ex=0;
                do {
                        u.makeGNStr(retNodeQ0);
                        notAddNodes = retNodeQ0;
                        u.getNodeInfo(u,notAddNodes);
                        u.setOddNodeInfo( u,  notAddNodes, 1, notAddNodes);
                        notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                        if(notAddNodes.length() != 0){
                                        u.setEvenNodesInfo(u,  notAddNodes, 2, notAddNodes);
                                        u.AddNodeToAddressed(notAddNodes);
                                        z=t.RanDeg1(u,notAddNodes);
                                        u.makeGNStr(u);
                                        u.getNodeInfo(u,notAddNodes.substring(z,z+3));
                                        notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                                }
                        if(Q0.length() != 0){
                                        if(notAddNodes.length() != 0){
                                                z=t.RanDeg1(u,notAddNodes);
                                                int z1=Q0.indexOf(notAddNodes.substring(z,z+3));
                                                Q0=Q0.substring(z1-1,z1+3)+Q0.substring(0,z1-
1)+Q0.substring(z1+3,Q0.length());
                                        }else{
                                                z=t.RanDeg(u,Q0);

Q0=Q0.substring(z,z+4)+Q0.substring(0,z)+Q0.substring(z+4,Q0.length());
```

```
                                   }
                              retNodeQ0=u.GetNodeFromQ0();
                    }
                            else
                                   ex=1;
                    } while (ex != 1);

            if(( u.MIS.length()/3) > u.x){
               u.x=u.MIS.length()/3;
               u.MISRes=u.MIS;
            }
    }

    void MinMIS(int g,RanModCAMIS u, String startNode,String xQ0) {
               int inxm=0;
                    int z=0;
                    String w = "";
                    String notAddNodes;
                    String retNodeQ0;
                    String retNodeQ1;
                    String retNodeQ2;
                    u.InitMIS(u);
                    int cnt = 0;
                    int cnt2;
                    Q0=xQ0;
                    retNodeQ0=startNode;
                    int ex=0;
                    do {
                            u.makeGNStr(retNodeQ0);
                            notAddNodes = retNodeQ0;
                            u.getNodeInfo(u,notAddNodes);
                                   u.setOddNodeInfo( u,  notAddNodes, 1, notAddNodes);
                                   notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                                   if(notAddNodes.length() != 0){
                                           u.setEvenNodesInfo(u,  notAddNodes, 2,
notAddNodes);
                                           u.AddNodeToAddressed(notAddNodes);
                                           z=t.minDeg1(u,notAddNodes);
                                           u.makeGNStr(u);
                                           u.getNodeInfo(u,notAddNodes.substring(z,z+3));
                                           notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                                           }
                                   if(Q0.length() != 0){
                                           if(notAddNodes.length() != 0){
                                                  z=t.minDeg1(u,notAddNodes);
                                                  int
z1=Q0.indexOf(notAddNodes.substring(z,z+3));
                                                  Q0=Q0.substring(z1-
1,z1+3)+Q0.substring(0,z1-1)+Q0.substring(z1+3,Q0.length());
```

```
                                        }else{
                                             z=t.minDeg(u,Q0);

Q0=Q0.substring(z,z+4)+Q0.substring(0,z)+Q0.substring(z+4,Q0.length());

                                        }
                                   retNodeQ0=u.GetNodeFromQ0();
                              }
                               else
                                        ex=1;
                         } while (ex != 1);

                    if(( u.MIS.length()/3) > u.x){
                       u.x=u.MIS.length()/3;
                       u.MISRes=u.MIS;
                    }
          }

     void MinMIS1(int g,RanModCAMIS u, String startNode,String xQ0) {
        int inxm=0;
          int z=0;
          String w = "";
          String notAddNodes;
          String retNodeQ0;
          String retNodeQ1;
          String retNodeQ2;
          u.InitMIS(u);
          int cnt = 0;
          int cnt2;
          Q0=xQ0;
          retNodeQ0=startNode;
          int ex=0;
          do {
               u.makeGNStr(retNodeQ0);
               notAddNodes = retNodeQ0;
                    u.getNodeInfo(u,notAddNodes);
                    u.setOddNodeInfo( u,  notAddNodes, 1, notAddNodes);

                    notAddNodes=u.Nigh;
                    if(notAddNodes.length() != 0){

                         z=t.minDeg1(u,notAddNodes);
                         u.makeGNStr(u);
                         u.getNodeInfo(u,notAddNodes.substring(z,z+3));
                         notAddNodes=u.Nigh;
                         }

                    if(Q0.length() != 0){
                         if(notAddNodes.length() != 0){
                              z=t.minDeg1(u,notAddNodes);
```

```
                                                            int z1=Q0.indexOf(notAddNodes.substring(z,z+3));
                                                            if(z1 != -1)
                                                            Q0=Q0.substring(z1-1,z1+3)+Q0.substring(0,z1-
1)+Q0.substring(z1+3,Q0.length());
                                                  }else{
                                                            z=t.minDeg(u,Q0);

Q0=Q0.substring(z,z+4)+Q0.substring(0,z)+Q0.substring(z+4,Q0.length());

                                                  }
                                            retNodeQ0=u.GetNodeFromQ0();
                                        }
                                         else
                                                  ex=1;
                              } while (ex != 1);

                    if(( u.MIS.length()/3) > u.x){
                       u.x=u.MIS.length()/3;
                       u.MISRes=u.MIS;
                    }
}

           RanModCAMIS   DynmcMISClass(RanModCAMIS r, int nodesNO,int GDeg) throws
ClassNotFoundException, InstantiationException, IllegalArgumentException,
IllegalAccessException, InvocationTargetException{
                                String  name="CA_MIS.G_"+nodesNO+"_"+GDeg;
                                      Class c=Class.forName(name);
                                      Object o=c.newInstance();
                                      Method methods[]=c.getMethods();
                                      RanModCAMIS t=(RanModCAMIS) methods[0].invoke(o, new
Object[]{r});
                                      return t;
            }
           void MaxMIS(int g,RanModCAMIS u, String startNode,String xQ0) {
                    int inxm=0;
                    int z=0;
                    String w = "";
                    String notAddNodes;
                    String retNodeQ0;
                    String retNodeQ1;
                    String retNodeQ2;
                    u.InitMIS(u);
                    int cnt = 0;
                    int cnt2;
                    Q0=xQ0;
                    retNodeQ0=startNode;
                    int ex=0;
                    do {
                              u.makeGNStr(retNodeQ0);
                              notAddNodes = retNodeQ0;
```

```
                                u.getNodeInfo(u,notAddNodes);
                                u.setOddNodeInfo( u,  notAddNodes, 1, notAddNodes);
                                notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                                if(notAddNodes.length() != 0){
                                        u.setEvenNodesInfo(u,  notAddNodes, 2, notAddNodes);
                                        u.AddNodeToAddressed(notAddNodes);
                                   z=t.maxDeg1(u,notAddNodes);
                                   u.makeGNStr(u);
                                   u.getNodeInfo(u,notAddNodes.substring(z,z+3));
                                   notAddNodes=u.CheckIfNodeAddressed(u.Nigh);
                                        }

                                if(Q0.length() != 0){
                                        if(notAddNodes.length() != 0){
                                                z=t.maxDeg1(u,notAddNodes);
                                                int z1=Q0.indexOf(notAddNodes.substring(z,z+3));
                                                Q0=Q0.substring(z1-1,z1+3)+Q0.substring(0,z1-
1)+Q0.substring(z1+3,Q0.length());
                                        }else{
                                                z=t.maxDeg(u,Q0);

Q0=Q0.substring(z,z+4)+Q0.substring(0,z)+Q0.substring(z+4,Q0.length());

                                        }
                                   retNodeQ0=u.GetNodeFromQ0();
                                }
                                 else
                                        ex=1;
                        } while (ex != 1);

                if(( u.MIS.length()/3) > u.x){
                   u.x=u.MIS.length()/3;
                   u.MISRes=u.MIS;
                }
        }

         void CAMISMain(int PnoOfNodes,int PDegree,WriteToFile w,DataOutputStream
out) throws IOException, IllegalArgumentException, ClassNotFoundException,
InstantiationException, IllegalAccessException, InvocationTargetException {
                MISArr=new int[PnoOfNodes];
                RanModCAMIS r=new RanModCAMIS();
                RanModCAMIS util=DynmcMISClass(r,PnoOfNodes,PDegree);
                String
SQ0="@000@001@002@003@004@005@006@007@008@009@010@011@012@013@014@
015@016@017@018@019@020@021@022@023@024@025@026@027@028@029@030@03
1@032@033@034@035@036@037@038@039@040@041@042@043@044@045@046@047@
048@049@050@051@052@053@054@055@056@057@058@059@060@061@062@063@06
4@065@066@067@068@069@070@071@072@073@074@075@076@077@078@079@080@
081@082@083@084@085@086@087@088@089@090@091@092@093@094@095@096@09
7@098@099@100@101@102@103@104@105@106@107@108@109@110@111@112@113@
```

**114@115@116@117@118@119@120@121@122@123@124@125@126@127@128@129@13
0@131@132@133@134@135@136@137@138@139@140@141@142@143@144@145@146@
147@148@149@150@151@152@153@154@155@156@157@158@159@160@161@162@16
3@164@165@166@167@168@169@170@171@172@173@174@175@176@177@178@179@
180@181@182@183@184@185@186@187@188@189@190@191@192@193@194@195@19
6@197@198@199@200@201@202@203@204@205@206@207@208@209@210@211@212@
213@214@215@216@217@218@219@220@221@222@223@224@225@226@227@228@22
9@230@231@232@233@234@235@236@237@238@239@240@241@242@243@244@245@
246@247@248@249@250@251@252@253@254@255@256@257@258@259@260@261@26
2@263@264@265@266@267@268@269@270@271@272@273@274@275@276@277@278@
279@280@281@282@283@284@285@286@287@288@289@290@291@292@293@294@29
5@296@297@298@299@300@301@302@303@304@305@306@307@308@309@310@311@
312@313@314@315@316@317@318@319@320@321@322@323@324@325@326@327@32
8@329@330@331@332@333@334@335@336@337@338@339@340@341@342@343@344@
345@346@347@348@349@350@351@352@353@354@355@356@357@358@359@360@36
1@362@363@364@365@366@367@368@369@370@371@372@373@374@375@376@377@
378@379@380@381@382@383@384@385@386@387@388@389@390@391@392@393@39
4@395@396@397@398@399@400@401@402@403@404@405@406@407@408@409@410@
411@412@413@414@415@416@417@418@419@420@421@422@423@424@425@426@42
7@428@429@430@431@432@433@434@435@436@437@438@439@440@441@442@443@
444@445@446@447@448@449@450@451@452@453@454@455@456@457@458@459@46
0@461@462@463@464@465@466@467@468@469@470@471@472@473@474@475@476@
477@478@479@480@481@482@483@484@485@486@487@488@489@490@491@492@49
3@494@495@496@497@498@499@500@501@502@503@504@505@506@507@508@509@
510@511@512@513@514@515@516@517@518@519@520@521@522@523@524@525@52
6@527@528@529@530@531@532@533@534@535@536@537@538@539@540@541@542@
543@544@545@546@547@548@549@550@551@552@553@554@555@556@557@558@55
9@560@561@562@563@564@565@566@567@568@569@570@571@572@573@574@575@
576@577@578@579@580@581@582@583@584@585@586@587@588@589@590@591@59
2@593@594@595@596@597@598@599@600@601@602@603@604@605@606@607@608@
609@610@611@612@613@614@615@616@617@618@619@620@621@622@623@624@62
5@626@627@628@629@630@631@632@633@634@635@636@637@638@639@640@641@
642@643@644@645@646@647@648@649@650@651@652@653@654@655@656@657@65
8@659@660@661@662@663@664@665@666@667@668@669@670@671@672@673@674@
675@676@677@678@679@680@681@682@683@684@685@686@687@688@689@690@69
1@692@693@694@695@696@697@698@699@700@701@702@703@704@705@706@707@
708@709@710@711@712@713@714@715@716@717@718@719@720@721@722@723@72
4@725@726@727@728@729@730@731@732@733@734@735@736@737@738@739@740@
741@742@743@744@745@746@747@748@749@750@751@752@753@754@755@756@75
7@758@759@760@761@762@763@764@765@766@767@768@769@770@771@772@773@
774@775@776@777@778@779@780@781@782@783@784@785@786@787@788@789@79
0@791@792@793@794@795@796@797@798@799@800@801@802@803@804@805@806@
807@808@809@810@811@812@813@814@815@816@817@818@819@820@821@822@82
3@824@825@826@827@828@829@830@831@832@833@834@835@836@837@838@839@
840@841@842@843@844@845@846@847@848@849@850@851@852@853@854@855@85
6@857@858@859@860@861@862@863@864@865@866@867@868@869@870@871@872@
873@874@875@876@877@878@879@880@881@882@883@884@885@886@887@888@88
9@890@891@892@893@894@895@896@897@898@899@900@901@902@903@904@905@
906@907@908@909@910@911@912@913@914@915@916@917@918@919@920@921@92
2@923@924@925@926@927@928@929@930@931@932@933@934@935@936@937@938@**

```
939@940@941@942@943@944@945@946@947@948@949@950@951@952@953@954@95
5@956@957@958@959@960@961@962@963@964@965@966@967@968@969@970@971@
972@973@974@975@976@977@978@979@980@981@982@983@984@985@986@987@98
8@989@990@991@992@993@994@995@996@997@998@999";
            long time=System.currentTimeMillis();
            System.gc();
            String h1="";
            for(int g=0;g<PnoOfNodes;g++){
                if(g < 10)
                     h1="00"+g;
                else if(g > 9 && g < 100)
                    h1="0"+g;
                else
                    h1=""+g;
                util.RanMIS(g,util,h1,SQ0.substring(0,PnoOfNodes*4));
            }
            w.outStr(out,"Ran G with N="+ PnoOfNodes +" & D="+PDegree + "
T="+(System.currentTimeMillis()-time)+" ms & S="+util.x+" MIS="+util.MISRes+"\n");

            System.gc();
            util.MISRes="";
            util.x=0;
            long time1=System.currentTimeMillis();
            for(int g=0;g<PnoOfNodes;g++){
                if(g < 10)
                     h1="00"+g;
                else if(g > 9 && g < 100)
                    h1="0"+g;
                else
                    h1=""+g;
                util.MinMIS(g,util,h1,SQ0.substring(0,PnoOfNodes*4));
            }
            w.outStr(out,"Min G with N="+ PnoOfNodes +" & D="+PDegree + "
T="+(System.currentTimeMillis()-time1)+" ms & S="+util.x+" MIS="+util.MISRes+"\n");
            System.out.println("Running  Minimum Graph with Nodes "+ PnoOfNodes +"
and Degree "+PDegree + " tooks "+(System.currentTimeMillis()-time1)+" ms and the result is
"+util.x+" MIS= "+util.MISRes);

                System.gc();
            util.MISRes="";
            util.x=0;
            long time2=System.currentTimeMillis();
            for(int g=0;g<PnoOfNodes;g++){
                if(g < 10)
                     h1="00"+g;
                else if(g > 9 && g < 100)
                    h1="0"+g;
                else
                    h1=""+g;
                util.MaxMIS(g,util,h1,SQ0.substring(0,PnoOfNodes*4));
```

```
                }
                w.outStr(out,"Max G with N="+ PnoOfNodes +" & D="+PDegree + "
T="+(System.currentTimeMillis()-time2)+" ms & S="+util.x+" MIS="+util.MISRes+"\n\n");
                System.gc();

        }
}
```

Program Name	**ThreadClient**
Program Description	**This program is the implementation of Proposed Parallel Algorithm.**

```
package CA_MIS;

import java.io.DataOutputStream;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.Calendar;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class ThreadClient {
static long g1;

        private static class WorkerThread extends Thread {

                 Lock lock = new ReentrantLock();

                volatile static    String G = "";
                static volatile String MIS ="";
            int noOfNOdes=0;
            int Degree=0;
            static char ind='t';
                static long g2;
                 WorkerThread(String PG,int PnoOfNodes,int PDegree){
                        this.Degree=PDegree;
                        this.noOfNOdes=PnoOfNodes;
                        this.G=PG.substring(0,(noOfNOdes)*4);
                        this.MIS=PG.substring(0,(noOfNOdes)*4);

                }
                volatile String[] MISArr=new String[noOfNOdes];
                volatile int inx=0;
                ThreadClient tc=new ThreadClient();


                synchronized  String GetNodeFromQ0() {
                        String q1c = "@";
                        lock.lock();
                        synchronized (MIS){
                        if (MIS.length() == 0) {
                                ind='f';
                                return "@";
                        }
                        q1c = MIS.substring(1, 4);
                        MIS = MIS.substring(4, MIS.length());
                        lock.unlock();
                        return q1c;
               }
                }
```

```
                public void run()  {


                        int[] MISArr=new int[noOfNOdes];
                        RanModCAMIS r=new RanModCAMIS();

                        try {

                                RanModCAMIS util=r.DynmcMISClass(r,noOfNOdes,Degree);
                                long time=System.currentTimeMillis();
                                while (G.length() != 0 && ind =='t') {
                                        String t=GetNodeFromQ0();
                                        if(t !="@"){
                                        util.MinMIS(0,util, t,G);

                                        }
                                }
                                System.out.println("Time is "+(System.currentTimeMillis()-
time));

                        } catch (IllegalArgumentException e) {

                                e.printStackTrace();
                        } catch (ClassNotFoundException e) {

                                e.printStackTrace();
                        } catch (InstantiationException e) {

                                e.printStackTrace();
                        } catch (IllegalAccessException e) {

                                e.printStackTrace();
                        } catch (InvocationTargetException e) {

                                e.printStackTrace();
                        }



                }

        }
        void CAMISMain(int PnoOfNodes,int PDegree,WriteToFile wf,DataOutputStream
out) throws IOException, IllegalArgumentException, ClassNotFoundException,
InstantiationException, IllegalAccessException, InvocationTargetException {
                ThreadClient t=new ThreadClient();
                String strMIS =
"@000@001@002@003@004@005@006@007@008@009@010@011@012@013@014@015
@016@017@018@019@020@021@022@023@024@025@026@027@028@029@030@031@0
32@033@034@035@036@037@038@039@040@041@042@043@044@045@046@047@048
```

**@049@050@051@052@053@054@055@056@057@058@059@060@061@062@063@064@0
65@066@067@068@069@070@071@072@073@074@075@076@077@078@079@080@081
@082@083@084@085@086@087@088@089@090@091@092@093@094@095@096@097@0
98@099@100@101@102@103@104@105@106@107@108@109@110@111@112@113@114
@115@116@117@118@119@120@121@122@123@124@125@126@127@128@129@130@1
31@132@133@134@135@136@137@138@139@140@141@142@143@144@145@146@147
@148@149@150@151@152@153@154@155@156@157@158@159@160@161@162@163@1
64@165@166@167@168@169@170@171@172@173@174@175@176@177@178@179@180
@181@182@183@184@185@186@187@188@189@190@191@192@193@194@195@196@1
97@198@199@200@201@202@203@204@205@206@207@208@209@210@211@212@213
@214@215@216@217@218@219@220@221@222@223@224@225@226@227@228@229@2
30@231@232@233@234@235@236@237@238@239@240@241@242@243@244@245@246
@247@248@249@250@251@252@253@254@255@256@257@258@259@260@261@262@2
63@264@265@266@267@268@269@270@271@272@273@274@275@276@277@278@279
@280@281@282@283@284@285@286@287@288@289@290@291@292@293@294@295@2
96@297@298@299@300@301@302@303@304@305@306@307@308@309@310@311@312
@313@314@315@316@317@318@319@320@321@322@323@324@325@326@327@328@3
29@330@331@332@333@334@335@336@337@338@339@340@341@342@343@344@345
@346@347@348@349@350@351@352@353@354@355@356@357@358@359@360@361@3
62@363@364@365@366@367@368@369@370@371@372@373@374@375@376@377@378
@379@380@381@382@383@384@385@386@387@388@389@390@391@392@393@394@3
95@396@397@398@399@400@401@402@403@404@405@406@407@408@409@410@411
@412@413@414@415@416@417@418@419@420@421@422@423@424@425@426@427@4
28@429@430@431@432@433@434@435@436@437@438@439@440@441@442@443@444
@445@446@447@448@449@450@451@452@453@454@455@456@457@458@459@460@4
61@462@463@464@465@466@467@468@469@470@471@472@473@474@475@476@477
@478@479@480@481@482@483@484@485@486@487@488@489@490@491@492@493@4
94@495@496@497@498@499@500@501@502@503@504@505@506@507@508@509@510
@511@512@513@514@515@516@517@518@519@520@521@522@523@524@525@526@5
27@528@529@530@531@532@533@534@535@536@537@538@539@540@541@542@543
@544@545@546@547@548@549@550@551@552@553@554@555@556@557@558@559@5
60@561@562@563@564@565@566@567@568@569@570@571@572@573@574@575@576
@577@578@579@580@581@582@583@584@585@586@587@588@589@590@591@592@5
93@594@595@596@597@598@599@600@601@602@603@604@605@606@607@608@609
@610@611@612@613@614@615@616@617@618@619@620@621@622@623@624@625@6
26@627@628@629@630@631@632@633@634@635@636@637@638@639@640@641@642
@643@644@645@646@647@648@649@650@651@652@653@654@655@656@657@658@6
59@660@661@662@663@664@665@666@667@668@669@670@671@672@673@674@675
@676@677@678@679@680@681@682@683@684@685@686@687@688@689@690@691@6
92@693@694@695@696@697@698@699@700@701@702@703@704@705@706@707@708
@709@710@711@712@713@714@715@716@717@718@719@720@721@722@723@724@7
25@726@727@728@729@730@731@732@733@734@735@736@737@738@739@740@741
@742@743@744@745@746@747@748@749@750@751@752@753@754@755@756@757@7
58@759@760@761@762@763@764@765@766@767@768@769@770@771@772@773@774
@775@776@777@778@779@780@781@782@783@784@785@786@787@788@789@790@7
91@792@793@794@795@796@797@798@799@800@801@802@803@804@805@806@807
@808@809@810@811@812@813@814@815@816@817@818@819@820@821@822@823@8
24@825@826@827@828@829@830@831@832@833@834@835@836@837@838@839@840
@841@842@843@844@845@846@847@848@849@850@851@852@853@854@855@856@8
57@858@859@860@861@862@863@864@865@866@867@868@869@870@871@872@873**

```
@874@875@876@877@878@879@880@881@882@883@884@885@886@887@888@889@8
90@891@892@893@894@895@896@897@898@899@900@901@902@903@904@905@906
@907@908@909@910@911@912@913@914@915@916@917@918@919@920@921@922@9
23@924@925@926@927@928@929@930@931@932@933@934@935@936@937@938@939
@940@941@942@943@944@945@946@947@948@949@950@951@952@953@954@955@9
56@957@958@959@960@961@962@963@964@965@966@967@968@969@970@971@972
@973@974@975@976@977@978@979@980@981@982@983@984@985@986@987@988@9
89@990@991@992@993@994@995@996@997@998@999";
                    WorkerThread[] threads = new WorkerThread[50];
                    for (int i = 0; i < 20; i++) {
                              threads[i] = new WorkerThread(strMIS,PnoOfNodes,PDegree);
                              threads[i].start();
                    }




          }

}
```

Program Name	**MWilf**
Program Description	**This program is the implementation of Modified Wilf Algorithm.**

```
package CA_MIS;

import java.io.DataOutputStream;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.Calendar;

public class MWilf {
	String GN;
	String G="";
	String MIS;
	int[] resArr=new int[1000];
	String makeGNStr(RanModCAMIS w,MWilf y,String M){

		int inx;
		String s=M.substring(1,4);
		w.getNodeInfo(w,s);
		String t1=M.substring(4,M.length());
		String t2="";
		t2=t1;

		for(int i=0;i<w.Nigh.length();i=i+3){
			inx = t2.indexOf(w.Nigh.substring(i, i + 3));
			if (inx != -1)

				t2 = t2.substring(0, inx-1) + t2.substring(inx + 3, t2.length());
		}
		return t2;
	}

	String makeGStr(String M){

		return M.substring(4,M.length());
		}



	int maxSet(RanModCAMIS w,MWilf y,String G,int c,int ind,String res){

		if (G.length()==0){
			return c;
		}
		MaxTest t=new MaxTest();
		int z=t.maxDeg(w,G);
		G=G.substring(z,z+4)+G.substring(0,z)+G.substring(z+4,G.length());
		String g=y.makeGStr(G);
		String gn=y.makeGNStr(w,y,G);
		if(ind==2){
			c=c+1;
			resArr[c]=1;
```

```
                }
                int m1=maxSet(w,y,g,c,1,res);
                int m2=maxSet(w,y,gn,c,2,res);
                return c;

        }


                void CAMISMain(int PnoOfNodes,int PDegree,WriteToFile
wf,DataOutputStream out) throws IOException, IllegalArgumentException,
ClassNotFoundException, InstantiationException, IllegalAccessException,
InvocationTargetException {
                        MWilf y=new MWilf();


                        String
SQ0="@000@001@002@003@004@005@006@007@008@009@010@011@012@013@014@0
15@016@017@018@019@020@021@022@023@024@025@026@027@028@029@030@031@
032@033@034@035@036@037@038@039@040@041@042@043@044@045@046@047@048
@049@050@051@052@053@054@055@056@057@058@059@060@061@062@063@064@06
5@066@067@068@069@070@071@072@073@074@075@076@077@078@079@080@081@0
82@083@084@085@086@087@088@089@090@091@092@093@094@095@096@097@098@
099@100@101@102@103@104@105@106@107@108@109@110@111@112@113@114@115
@116@117@118@119@120@121@122@123@124@125@126@127@128@129@130@131@13
2@133@134@135@136@137@138@139@140@141@142@143@144@145@146@147@148@1
49@150@151@152@153@154@155@156@157@158@159@160@161@162@163@164@165@
166@167@168@169@170@171@172@173@174@175@176@177@178@179@180@181@182
@183@184@185@186@187@188@189@190@191@192@193@194@195@196@197@198@19
9@200@201@202@203@204@205@206@207@208@209@210@211@212@213@214@215@2
16@217@218@219@220@221@222@223@224@225@226@227@228@229@230@231@232@
233@234@235@236@237@238@239@240@241@242@243@244@245@246@247@248@249
@250@251@252@253@254@255@256@257@258@259@260@261@262@263@264@265@26
6@267@268@269@270@271@272@273@274@275@276@277@278@279@280@281@282@2
83@284@285@286@287@288@289@290@291@292@293@294@295@296@297@298@299@
300@301@302@303@304@305@306@307@308@309@310@311@312@313@314@315@316
@317@318@319@320@321@322@323@324@325@326@327@328@329@330@331@332@33
3@334@335@336@337@338@339@340@341@342@343@344@345@346@347@348@349@3
50@351@352@353@354@355@356@357@358@359@360@361@362@363@364@365@366@
367@368@369@370@371@372@373@374@375@376@377@378@379@380@381@382@383
@384@385@386@387@388@389@390@391@392@393@394@395@396@397@398@399@40
0@401@402@403@404@405@406@407@408@409@410@411@412@413@414@415@416@4
17@418@419@420@421@422@423@424@425@426@427@428@429@430@431@432@433@
434@435@436@437@438@439@440@441@442@443@444@445@446@447@448@449@450
@451@452@453@454@455@456@457@458@459@460@461@462@463@464@465@466@46
7@468@469@470@471@472@473@474@475@476@477@478@479@480@481@482@483@4
84@485@486@487@488@489@490@491@492@493@494@495@496@497@498@499@500@
501@502@503@504@505@506@507@508@509@510@511@512@513@514@515@516@517
@518@519@520@521@522@523@524@525@526@527@528@529@530@531@532@533@53
4@535@536@537@538@539@540@541@542@543@544@545@546@547@548@549@550@5
51@552@553@554@555@556@557@558@559@560@561@562@563@564@565@566@567@
```

```
568@569@570@571@572@573@574@575@576@577@578@579@580@581@582@583@584
@585@586@587@588@589@590@591@592@593@594@595@596@597@598@599@600@60
1@602@603@604@605@606@607@608@609@610@611@612@613@614@615@616@617@6
18@619@620@621@622@623@624@625@626@627@628@629@630@631@632@633@634@
635@636@637@638@639@640@641@642@643@644@645@646@647@648@649@650@651
@652@653@654@655@656@657@658@659@660@661@662@663@664@665@666@667@66
8@669@670@671@672@673@674@675@676@677@678@679@680@681@682@683@684@6
85@686@687@688@689@690@691@692@693@694@695@696@697@698@699@700@701@
702@703@704@705@706@707@708@709@710@711@712@713@714@715@716@717@718
@719@720@721@722@723@724@725@726@727@728@729@730@731@732@733@734@73
5@736@737@738@739@740@741@742@743@744@745@746@747@748@749@750@751@7
52@753@754@755@756@757@758@759@760@761@762@763@764@765@766@767@768@
769@770@771@772@773@774@775@776@777@778@779@780@781@782@783@784@785
@786@787@788@789@790@791@792@793@794@795@796@797@798@799@800@801@80
2@803@804@805@806@807@808@809@810@811@812@813@814@815@816@817@818@8
19@820@821@822@823@824@825@826@827@828@829@830@831@832@833@834@835@
836@837@838@839@840@841@842@843@844@845@846@847@848@849@850@851@852
@853@854@855@856@857@858@859@860@861@862@863@864@865@866@867@868@86
9@870@871@872@873@874@875@876@877@878@879@880@881@882@883@884@885@8
86@887@888@889@890@891@892@893@894@895@896@897@898@899@900@901@902@
903@904@905@906@907@908@909@910@911@912@913@914@915@916@917@918@919
@920@921@922@923@924@925@926@927@928@929@930@931@932@933@934@935@93
6@937@938@939@940@941@942@943@944@945@946@947@948@949@950@951@952@9
53@954@955@956@957@958@959@960@961@962@963@964@965@966@967@968@969@
970@971@972@973@974@975@976@977@978@979@980@981@982@983@984@985@986
@987@988@989@990@991@992@993@994@995@996@997@998@999";


        y.G=SQ0.substring(0,PnoOfNodes*4);
                RanModCAMIS w=new RanModCAMIS();
                long time=System.currentTimeMillis();
                int cnt=0;
                String res="";
                w.InitMIS(w);
                RanModCAMIS util =w.DynmcMISClass(w,PnoOfNodes,PDegree);
                int cret=y.maxSet(util,y,y.G,cnt,2,res);



         for(int q1=1;q1 < y.resArr.length;q1++){
                if(y.resArr[q1] ==0){
                        System.out.println("Running  Wilf Graph with Nodes "+
PnoOfNodes +" and Degree "+PDegree + " tooks ="+(System.currentTimeMillis()-time)+" ms
& S="+(q1-1)+"\n");



wf.outStr(out,"M.Wilf G with N="+ PnoOfNodes +" & D="+PDegree + "
T="+(System.currentTimeMillis()-time)+" ms & S="+(q1-1)+"\n");
                        break;
```

```
                        }
            }


          }
}
```

Program Name	**Wilf**
Program Description	**This program is the implementation of Wilf Algorithm.**

```
package CA_MIS;

import java.io.DataOutputStream;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.Calendar;

public class Wilf {
	String GN;
	String G="";
	String MIS;
	int[] resArr=new int[1000];
	String makeGNStr(RanModCAMIS w,Wilf y,String M){

		int inx;
		String s=M.substring(1,4);
		w.getNodeInfo(w,s);
		String t1=M.substring(4,M.length());
		String t2="";
		t2=t1;

		for(int i=0;i<w.Nigh.length();i=i+3){
			inx = t2.indexOf(w.Nigh.substring(i, i + 3));
			if (inx != -1)

				t2 = t2.substring(0, inx-1) + t2.substring(inx + 3, t2.length());
		}

		return t2;
	}

	String makeGStr(String M){
		return M.substring(4,M.length());
		}



	int maxSet(RanModCAMIS w,Wilf y,String G,int c,int ind,String res){

		if (G.length()==0){
			return c;
		}

		String g=y.makeGStr(G);
		String gn=y.makeGNStr(w,y,G);
		if(ind==2){
			c=c+1;
			resArr[c]=1;
```

```
                        res=res+g;
                }

                int m1=maxSet(w,y,g,c,1,res);
                int m2=maxSet(w,y,gn,c,2,res);

                return c;

        }



                void CAMISMain(int PnoOfNodes,int PDegree,WriteToFile
wf,DataOutputStream out) throws IOException, IllegalArgumentException,
ClassNotFoundException, InstantiationException, IllegalAccessException,
InvocationTargetException {
                        Wilf y=new Wilf();



String
SQ0="@000@001@002@003@004@005@006@007@008@009@010@011@012@013@014@0
15@016@017@018@019@020@021@022@023@024@025@026@027@028@029@030@031@
032@033@034@035@036@037@038@039@040@041@042@043@044@045@046@047@048
@049@050@051@052@053@054@055@056@057@058@059@060@061@062@063@064@06
5@066@067@068@069@070@071@072@073@074@075@076@077@078@079@080@081@0
82@083@084@085@086@087@088@089@090@091@092@093@094@095@096@097@098@
099@100@101@102@103@104@105@106@107@108@109@110@111@112@113@114@115
@116@117@118@119@120@121@122@123@124@125@126@127@128@129@130@131@13
2@133@134@135@136@137@138@139@140@141@142@143@144@145@146@147@148@1
49@150@151@152@153@154@155@156@157@158@159@160@161@162@163@164@165@
166@167@168@169@170@171@172@173@174@175@176@177@178@179@180@181@182
@183@184@185@186@187@188@189@190@191@192@193@194@195@196@197@198@19
9@200@201@202@203@204@205@206@207@208@209@210@211@212@213@214@215@2
16@217@218@219@220@221@222@223@224@225@226@227@228@229@230@231@232@
233@234@235@236@237@238@239@240@241@242@243@244@245@246@247@248@249
@250@251@252@253@254@255@256@257@258@259@260@261@262@263@264@265@26
6@267@268@269@270@271@272@273@274@275@276@277@278@279@280@281@282@2
83@284@285@286@287@288@289@290@291@292@293@294@295@296@297@298@299@
300@301@302@303@304@305@306@307@308@309@310@311@312@313@314@315@316
@317@318@319@320@321@322@323@324@325@326@327@328@329@330@331@332@33
3@334@335@336@337@338@339@340@341@342@343@344@345@346@347@348@349@3
50@351@352@353@354@355@356@357@358@359@360@361@362@363@364@365@366@
367@368@369@370@371@372@373@374@375@376@377@378@379@380@381@382@383
@384@385@386@387@388@389@390@391@392@393@394@395@396@397@398@399@40
0@401@402@403@404@405@406@407@408@409@410@411@412@413@414@415@416@4
17@418@419@420@421@422@423@424@425@426@427@428@429@430@431@432@433@
434@435@436@437@438@439@440@441@442@443@444@445@446@447@448@449@450
@451@452@453@454@455@456@457@458@459@460@461@462@463@464@465@466@46
7@468@469@470@471@472@473@474@475@476@477@478@479@480@481@482@483@4
84@485@486@487@488@489@490@491@492@493@494@495@496@497@498@499@500@
```

```
501@502@503@504@505@506@507@508@509@510@511@512@513@514@515@516@517
@518@519@520@521@522@523@524@525@526@527@528@529@530@531@532@533@53
4@535@536@537@538@539@540@541@542@543@544@545@546@547@548@549@550@5
51@552@553@554@555@556@557@558@559@560@561@562@563@564@565@566@567@
568@569@570@571@572@573@574@575@576@577@578@579@580@581@582@583@584
@585@586@587@588@589@590@591@592@593@594@595@596@597@598@599@600@60
1@602@603@604@605@606@607@608@609@610@611@612@613@614@615@616@617@6
18@619@620@621@622@623@624@625@626@627@628@629@630@631@632@633@634@
635@636@637@638@639@640@641@642@643@644@645@646@647@648@649@650@651
@652@653@654@655@656@657@658@659@660@661@662@663@664@665@666@667@66
8@669@670@671@672@673@674@675@676@677@678@679@680@681@682@683@684@6
85@686@687@688@689@690@691@692@693@694@695@696@697@698@699@700@701@
702@703@704@705@706@707@708@709@710@711@712@713@714@715@716@717@718
@719@720@721@722@723@724@725@726@727@728@729@730@731@732@733@734@73
5@736@737@738@739@740@741@742@743@744@745@746@747@748@749@750@751@7
52@753@754@755@756@757@758@759@760@761@762@763@764@765@766@767@768@
769@770@771@772@773@774@775@776@777@778@779@780@781@782@783@784@785
@786@787@788@789@790@791@792@793@794@795@796@797@798@799@800@801@80
2@803@804@805@806@807@808@809@810@811@812@813@814@815@816@817@818@8
19@820@821@822@823@824@825@826@827@828@829@830@831@832@833@834@835@
836@837@838@839@840@841@842@843@844@845@846@847@848@849@850@851@852
@853@854@855@856@857@858@859@860@861@862@863@864@865@866@867@868@86
9@870@871@872@873@874@875@876@877@878@879@880@881@882@883@884@885@8
86@887@888@889@890@891@892@893@894@895@896@897@898@899@900@901@902@
903@904@905@906@907@908@909@910@911@912@913@914@915@916@917@918@919
@920@921@922@923@924@925@926@927@928@929@930@931@932@933@934@935@93
6@937@938@939@940@941@942@943@944@945@946@947@948@949@950@951@952@9
53@954@955@956@957@958@959@960@961@962@963@964@965@966@967@968@969@
970@971@972@973@974@975@976@977@978@979@980@981@982@983@984@985@986
@987@988@989@990@991@992@993@994@995@996@997@998@999";
                        y.G=SQ0.substring(0,PnoOfNodes*4);
                        RanModCAMIS w=new RanModCAMIS();
                        long time=System.currentTimeMillis();
                        int cnt=0;
                        String res="";
                        w.InitMIS(w);
                        RanModCAMIS util =w.DynmcMISClass(w,PnoOfNodes,PDegree);
                        int cret=y.maxSet(util,y,y.G,cnt,2,res);

            for(int q1=1;q1 < y.resArr.length;q1++){
                        if(y.resArr[q1] ==0){


        System.out.println("Running  Wilf Graph with Nodes "+ PnoOfNodes +" and Degree
"+PDegree + " tooks ="+(System.currentTimeMillis()-time)+" ms & S="+(q1-1)+"\n");
                                wf.outStr(out,"Wilf G with N="+ PnoOfNodes +" & D="+PDegree
+ " T="+(System.currentTimeMillis()-time)+" ms & S="+(q1-1)+"\n");
                                break;
                        }
            }
```

```
        }
}
```

Program Name	MaxTest
Program Description	**This program is the Program that responsible to return the node with Minimum Degree, Maximum Degree and the Randomly.**

```
package CA_MIS;

import java.util.Random;



public class MaxTest {
	int degree=0;
	String nigh="";
	String[] graph;


	int minDeg(RanModCAMIS u,String nigh){
		int g = 0;
		g=(Integer.valueOf(nigh.substring(1,4))).intValue();
		int A=u.node[g].Degree;
		int index=0;
		String s;
		int rep=0;
		for(int m1=0;m1<nigh.length();m1=m1+4){
			s=nigh.substring(m1+1,m1+4);
			g=(Integer.valueOf(s)).intValue();
			if(u.node[g].Degree < A){
				index=g;
				rep=m1;
				A=u.node[g].Degree;
			}

		}

		return rep;
	}

	int minDeg1(RanModCAMIS u,String nigh){
		int g = 0;
		g=(Integer.valueOf(nigh.substring(0,3))).intValue();
		int A=u.node[g].Degree;
		int index=0;
		String s;
		int rep=0;
		for(int m1=0;m1<nigh.length();m1=m1+3){
			s=nigh.substring(m1,m1+3);
			g=(Integer.valueOf(s)).intValue();
			if(u.node[g].Degree < A){
				index=g;
				rep=m1;
				A=u.node[g].Degree;
			}
```

```
		}

		return rep;
	}

	int maxDeg(RanModCAMIS u,String nigh){
		int g = 0;
		g=(Integer.valueOf(nigh.substring(1,4))).intValue();
		int A=u.node[g].Degree;
		int index=0;
		String s;
		int rep=0;
		for(int m1=0;m1<nigh.length();m1=m1+4){
			s=nigh.substring(m1+1,m1+4);
			g=(Integer.valueOf(s)).intValue();
			if(u.node[g].Degree > A){
				index=g;
				rep=m1;
				A=u.node[g].Degree;
			}

		}

		return rep;
	}
	int maxDeg1(RanModCAMIS u,String nigh){
		int g = 0;
		g=(Integer.valueOf(nigh.substring(0,3))).intValue();
		int A=u.node[g].Degree;
		int index=0;
		String s;
		int rep=0;
		for(int m1=0;m1<nigh.length();m1=m1+3){
			s=nigh.substring(m1,m1+3);
			g=(Integer.valueOf(s)).intValue();
			if(u.node[g].Degree > A){
				index=g;
				rep=m1;
				A=u.node[g].Degree;
			}

		}

		return rep;
	}

	int RanDeg(RanModCAMIS u,String nigh){
		RandomNo t=new RandomNo();
		int res=(int) t.rand(0,(nigh.length()/4)-1);
```

```
                return res*4;
        }

        int RanDeg1(RanModCAMIS u,String nigh){
                RandomNo t=new RandomNo();
                int res=(int) t.rand(0,(nigh.length()/3)-1);

                return res*3;
        }


        public static void main(String[] args) {
        RanModCAMIS util=new RanModCAMIS();
        util.InitMIS(util);
        MaxTest t=new MaxTest();
        String s="000001002003004005006007008009010011012013014015016017018019";
        t.maxDeg(util,s);




        }
}
```

Program Name	**WriteToFile**
Program Description	**This program is the Program that responsible to deal with the files on Operating system to save the results .**

```
package CA_MIS;

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;


public class WriteToFile {
	DataOutputStream out;
	void outStr(DataOutputStream out,String str) throws IOException{
		out.writeBytes(str );
	}

	void closeOutStr(DataOutputStream out) throws IOException{
		out.close();
	}


	public static void main(String[] args) throws IOException {

	}
}
```

Program Name	Node
Program Description	This program is used to represents the Node in graph.

```
package CA_MIS;


public class Node {
	public int Degree;
	public String Nigh;
	public int value;
	public String ParentName;
	public int ParentNo;

}
```

ايجاد أكبر مجموعة عقد مستقلة في مخطط باستخدام الخلايا الالية وخوارزميات تقريبية

إعداد
نايف أحمد السخني

المشرف
الدكتور أحمد الشرايعة

Arabic Summary

ملخص

تقدم هذه الرسالة أربعة خوارزميات تقريبية مبنيةعلى استخدام الخلايا الالية لايجاد أكبر مجموعة عقد مستقلة MIS في مخطط. تعتبر عملية ايجاد أكبر مجموعة عقد مستقلة من المشاكل الأساسية في التحسين الاندماجي. تعرف مجموعة العقد المستقلة IS في مخطط على انها مجموعة من العقد بحيث لا يكون هناك اتصال مباشر بين اي عقدتين في هذه المجموعة.تعرف مشكلة العقد المستقلة هي كيفية ايجاد الحد الأقصى لحجم مجموعة عقد مستقلة في مخطط معين.

تعتبر الخورزميات الدقيقة (Exact) المستخدمة لايجاد أكبر مجموعة عقد مستقلة في مخطط ، خورزميات محددودة لمعالجة المخططات ذات الحجم الصغير.

هذه الرسالة تهدف لايجاد أكبر مجموعة عقد مستقلة في مخطط باستخدام الخلايا الالية وخوارزميات تقريبية من خلال العوامل المرجحه عن طريق اختيار العقدة المرشحة بواسطة عدة طرق. ومن هذه الطرق الاختيار : عشوائيا ، العقدة ذات اقصى درجة ، والعقده ذات أقل درجة أدنى وتحسين الفاعلية والكفاءة من خلال التنفيذ المتوازي لهذه الخورزميات,

تم تحليل , تنفيذ وفحص هذه الخوارزميات لمخططات متعددة وباحجام مختلفة وكثافات متعددة.

كما تم فحص أداء التنفيذ المتوازي ، باستخدام التنفيذ المتعدد Multi threading في لغة جافا ، وكانت تقاس على جهاز كمبيوتر يعمل تحت نظام التشغيل ويندوز اكس. بي.

كما تم ايضا دراسة تأثير عدد احجام وكثافة الخلايا على كفاءة التنفيذ.

التحليل النظري اظهر ان هذه الخورزميات تحمل درجة تعقيد Complexity $O(n^{3.8})$ ولن يكون اسوء من $O(n^4)$.

وتبين النتائج ان الخوارزميات التقريبية المقترحة انتجت حجم عقد مستقلة مقارب للقيم الفعلية لمخططات بكثافات معينة . على سبيل المثال ، الخورزمية التقريبية المبنية على اساس اختيار العقدة ذات الدرجة الادنى , انتجت 75% من العقد المستقلة والمطابقة للقيم الفعلبة.وأعتبرت هي الادق في هذا الخصوص , ومن ثم الخورزمية التقريبية المبنية على اساس اختيار العقدة عشوائيا وفي النهاية الخورزمية التقريبية المبنية على اساس اختيار العقدة ذات الدرجة الاعلى.وتبين ان زيادة حجم المخطط يؤدي لزيادة حجم العقد المستقلة.

عند تغيير حجم المخطط وتثبيت الكثافة تبين زيادة حجم العقد المستقلة عند زيادة حجم المخطط. تبين زيادة حجم العقد المستقلة عند نقصان كثافة المخطط. عند زيادة كثافة المخطط يصبح حجم العقد المستقلة ثابت بغض النظر عن حجم المخطط. وبينت النتائج انه عند كثافة ثابته تبين ثبات حجم العقد المستقلة عند زيادة حجم المخطط فمثلا عند كثافة d=0.9 وتغيير حجم المخطط N=400,500,600,700,800,900,1000 تبين ثبات حجم العقد المستقلة تقريبا.
من ناحية التحسين في وقت التنفيذ اظهرت الخورزميات المقترحة تحسينا كبيرا في وقت التنفيذ مقارنة مع وقت تنفيذ خورزمية ويلف Wilf المعدلة .ولم يظهر التنفيذ المتوازي تاثير واضح في تحسين اداء الحورزمية من حيث تقليل الوقت اللازم للتنفيذ , ويعتقد ان التنفيذ المتوازي سيكون ملائما اكثر للمخططات ذات الحجم الكبير (مثلا 10000) وللتنفيذ من خلال اجهزة كمبيوتر مشبوكة من خلال شبكة او جهاز كمبيوتر يضم عدة وحد معالجة مركزية.

ميزة هذه الرسالة هي اقتراح , تطبيق وفحص كفاءة أربعة خورازميات تقريبية باستخدام الخلايا الالية وفتح المجال امام افكار جديدة لاستخدام الخلايا الحياة في تطبيقات جديدة والتنفيذ المتوازي المبني على الخلايا الحية لحل مشكلة MIS.