TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

IDK70LT

Aleksei Kulitškov 132555IAPM

INVESTIGATING EFFECTS OF APPLYING DIFFERENT HEURISTIC COLORING TECHNIQUES ON MODERN MAXIMUM CLIQUE ALGORITHMS

Master's thesis

Supervisor: Deniss Kumlander

PhD

Senior Researcher

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

IDK70LT

Aleksei Kulitškov 132555IAPM

HEURISTILISE VÄRVIMISE ERINEVATE TEHNIKATE RAKENDAMISE MÕJU KAASAEGSETELE SUURIMA KLIKI LEIDMISE ALGORITMIDELE

Magistritöö

Juhendaja: Deniss Kumlander

PhD

Vanem teadur

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Aleksei Kulitškov

09.05.2016

Abstract

Developers face many different problems every day. Some of them are not so complex, but the majority of them demand a lot of time in order to find the best solution. Consequently, a lot of researches have been conducted in this area, especially, in the field of graph theory. Why graph theory is so significant? The answer is simple: it can be applied to many different areas, such as technology, mathematics and science.

Well-known graph theory problems are graph coloring and finding the maximum clique in an undirected graph, or shortly - MCP. And these problems are closely related. Vertex coloring is usually considered an initial step before the start of finding maximum clique of a graph. The maximum clique problem is considered to be of NP-hard complexity, which means that there is no algorithm found that could solve this kind of problem in a polynomial time. This problem is of high importance since it could be encountered in a wide range of applications, for example, in computer or social network analysis. Therefore, it is crucial to develop a new or improve the currently known algorithm, which is meant to solve such kind of a problem.

This thesis starts from describing basic concepts of graph theory and its problems to introduce the main topic. After that, 17 coloring algorithms are introduced, described and tested against random and DIMACS instances of graphs and those, which showed the best results, are taken for further research. Then we move to algorithms that solve the problem of maximal clique. Almost all of them depend on the coloring of the vertices, which is made in the process of execution. In this work, we are going to investigate the effects of applying different types of coloring algorithms on modern maximum clique algorithms. The algorithms, which were chosen as the main objects of research, were initially invented in Tallinn University of Technology and are called VColorU and VRecolor-BT-u. At first, we perform an extensive experimental evaluation of these algorithms together with selected variants of coloring algorithms. In order to receive appropriate results, we conducted tests on random and DIMACS graph instances. Furthermore, to see if there was any influence depending on number of vertices of the graph, random tests were run with graphs having different densities. The results of our study show the time used by the algorithms to determine the maximal clique, the number of colors used in the process of vertex coloring and number of branches analyzed by maximum clique algorithm after vertex coloring.

It could be clearly seen from the results that some of the coloring algorithms helped to improve the VColorU and VRecolor-BT-u algorithms on graphs with certain densities. There are also some promising ideas brought up at the end of our work that might become a good start for future researches.

This thesis is written in English and is 126 pages long, including 4 chapters, 80 figures and 29 tables.

Annotatsioon

Heuristilise Värvimise Erinevate Tehnikate Rakendamise Mõju Kaasaegsetele Suurima Kliki Leidmise Algoritmidele

Tarkvaraarendajad puutuvad igapäevaselt kokku paljude probleemidega. Mõned neist pole eriti keerulised, kuid enamuste probleemide parima lahenduse leidmiseks kulub palju aega. Sellest tulenevalt on selles valdkonnas tehtud palju teaduslikke uuringuid, eriti graafiteooria vallas. Miks on graafiteooria nii oluline? Vastus on lihtne: seda saab kasutada paljudes erinevates valdkondades nagu näiteks tehnoloogia, matemaatika ning loodusteadused.

Teada tuntud graafiteooria probleemid on graafi värvimine ja maksimaalse arvu klikkide leidmine mittesuunalikus graafikus (inglise keeles lühidalt: MCP). Ja need probleemid on tugevalt omavahel seotud. Kõrgeima tipu esmast värvimist peetakse esimeseks sammuks enne, kui asutakse leidma graafi maksimaalsete klikkide arvu. Maksimaalsete klikkide probleemi peetakse keerukaks „NP-hard“ tasemel, mis tähendab sel puudub polünoomilise (piiratud) aja jooksul lahendamisega hakkama saav algoritm. See probleem on suure tähtsusega, sest võib ilmneda paljudes olukordades, näiteks arvuti- ja sotsiaalvõrgustike analüüsis. Seepärast on oluline arendada välja uus algoritm või muuta vana paremaks, mis on mõeldud seda tüüpi probleemide lahendamiseks.

See väitekiri algab peateema sissejuhatuseks graafiteooria põhikontseptsioonide ja selle probleemide kirjeldamisega. Peale seda tutvustatakse, kirjeldatakse ja testitakse juhuslike ja DIMACS tüüpi graafidega 17 värvimise algoritmi ja need, mis annavad parimaid tulemusi võetakse täiendava uurimise alla. Siis liigume edasi maksimaalsete klikkide arvu algoritmide juurde. Peaaegu kõik need sõltuvad tippude värvimisest, mis tehakse ära protsessi täitmise käigus. Selles töös uurime graafide erinevat tüüpi värvimisalgoritmide mõju tänapäevastele maksimaalse klikkide arvu algoritmidele.

Algoritmid, mis valiti selle uurimistöö peamisteks objektideks, töötati esmalt välja Tallinna Tehnikaülikoolis ning nende nimed on „VColorU“ ja „VRecolor-BT-u“. Alguses viime läbi nende algoritmide ulatusliku katsehindamise koos valitud värvimise algoritmide variantidega. Selleks, et saada sobiv arv tulemusi, viisime katsetusi läbi juhuslikel ja DIMACS graafidel. Peale selle, et testida, kas eksisteerib graafide tippude arvust sõltuv mõju, viidi läbi juhuslikke katseid erineva tihedusega graafidega. Meie uurimistöö tulemused näitavad maksimaalsete klikkide arvu leidmisele kulunud aega, kõrgeima tipu värvimise protsessis kasutatud värvide arvu ja maksimaalsete klikkide arvu algoritmi poolt analüüsitud harude arvu peale tipu värvimist.

Tulemustest on selgelt näha, et mõned värvimisalgoritmid aitasid parandada „VColorU“ ja „VRecolor-BT-u“ algoritme teatud tihedusega graafide puhul. Meie töö lõpus on samuti toodud ära mõned lubavad ideed, mis võivad osutuda tulevaste uuringute headeks lähtekohtadeks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 126 leheküljel, 4 peatükki, 80 joonist, 29 tabelit.

List of abbreviations and terms

MCP	Maximum Clique Problem
DIMACS	The Center for Discrete Mathematics and Theoretical Computer Science
MCP	Maximal Clique problem
IS	Independent Set
MIS	Maximal Independent Set
GCP	Graph Coloring Problem
LDO	Largest Degree Ordering
IDO	Incidence Degree Ordering
LF	Largest-First
SL	Smallest-Last
JP	Jones and Plassmann
PLF	Parallel Largest-First
PSL	Parallel Smallest-Last

Table of Contents

Table of Figures

List of Tables

1. Introduction

1.1. Background of the Study

The area of this study is **graph theory** – a rapidly developing branch of mathematics, which studies graphs. It could be found in practice in many other fields such as sociology, chemistry and even statistical physics. However, it has a great significance in computer science. It is a perfect tool for analyzing data and could be applied to different scientific problems. Although, we can say that this area is relatively old (it is said that graph theory may take its beginning in the long 1736 when Leonhard Euler published his paper about Seven Bridges of Königsberg problem [1]), it is rapidly developing since the middle of 20th century until now. There are still a lot of different actual problems in this area and, therefore, a lot of space for researches.

But how is it possible to solve different problems with the help of graph theory? What is that in general? Well, if you have a very complicated task and do not know how to find a solution, then maybe it is possible to represent the whole problem as a graph. But firstly it is necessary to introduce the concept of a "**graph**". We consider a graph a set of points and lines, which connect some of the points. In other words, it is a set of objects that are connected in some way. Specifically, it consists of 2 sets: a set of points, which are called **vertices**, and a set of lines, or **edges,** which represent relationships between the connected vertices. It is a mathematical model, which is a simplified version of the real world. So, some real life situations could be converted into graphs. For example, let us take the well-known social networking service "Facebook". Almost everyone uses it and has some people added to the Friends' list. It is possible to simplify this situation and represent it as a graph. For example, suppose that all the people registered on Facebook are the vertices of our new graph. Then, the relationships between them, or should we say edges, represent the fact that the people connected are friends. Very easy. This way by removing all the unnecessary details and making a simple mathematical model, which conveys the main idea of the situation, it is possible to solve many different complex problems, which we face in the real world. Therefore, it is very important to make researches in the graph theory field, to try to improve the currently achieved results or to invent something new, which could help to find solutions to graph problems or be revolutionary and beneficial to the graph theory in common.

1.2. Basic Definitions

In order to proceed, it is necessary to define some terms, which are going to be used further in this work.

Definition 1: **Undirected graph**

Let *G=(V, E)* be an undirected graph. Then, *V* is a finite set of elements called vertices and *E* is a finite set of unordered pairs of vertices, called edges [2]. It is easier to see it on an illustration below (Figure 1).

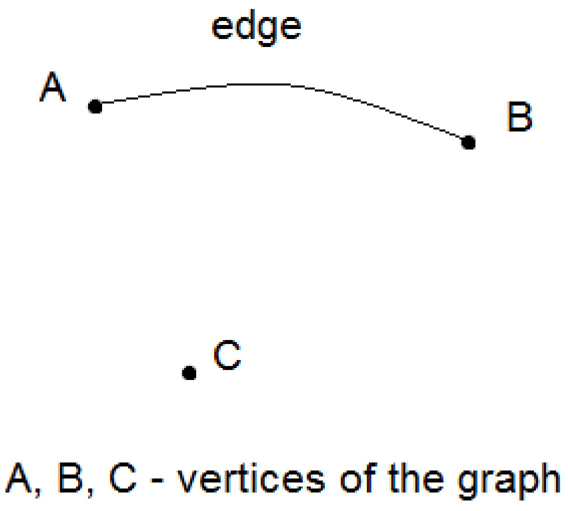

Figure 1. Graph.

Definition 2: **Order**

The cardinality of a set of vertices, or just the number of its elements, is called the order of a graph and is denoted as $n=|V|$.

Definition 3: **Size**

The cardinality of a set of edges, or just the number of its edges, is called the size of a graph and is denoted as $m=|E|$ [2].

Definition 4: **Adjacency**

If v_i and v_j are vertices that belong to one and the same graph and there is a relationship between them, which ends up being an edge, then these vertices are adjacent. In the mathematical equivalence it can written like this: $v_i, v_j \in V$ and $\{v_i, v_j\} \in E$. Furthermore, we can say that if $v_i, v_j \in V$ and $\{v_i, v_j\} \notin E$, then these vertices are nonadjacent (Figure 2).

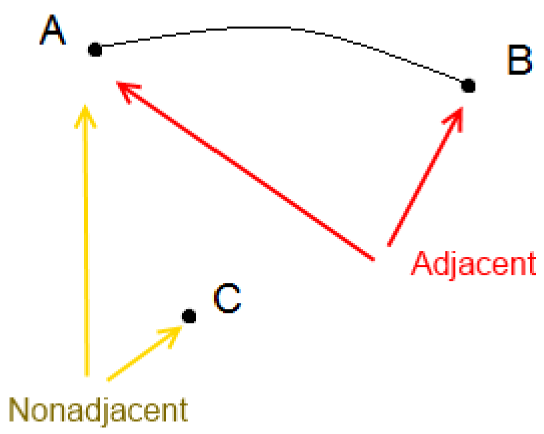

Figure 2. Adjacent and nonadjacent vertices.

Definition 5: **Vertex degree**

The degree of vertex *v* in graph *G* is the number of edges incident to it [2]. Or, in other words, it is the number of this vertex's neighbors, which are connected to it (Figure 3). The maximum degree of a vertex in a graph is the number of edges of a vertex with the maximum neighbors. The minimum degree of a vertex in a graph is the number of edges of a vertex, which has the least neighbors. Usually, the degree of a vertex is denoted as *deg(v).*

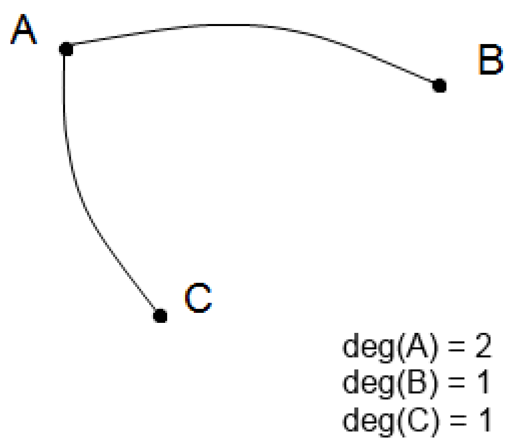

Figure 3. Vertex degree.

Definition 6: **Density**

Density is the ratio of the edges in graph *G* to the number of vertices of the graph. It is defined as *g(G)*. We can define its formula through the previously defined terms (definitions 1.2, 1.3), so that it looks as follows:

$$g(G) = \frac{2 * m}{n * (n - 1)}$$

Definition 7: **Complement graph**

A graph is considered to be complement (Figure 4) if it has the same vertices as graph G and any two vertices in this graph are adjacent only if the same vertices are nonadjacent in the original graph. So, it is possible to say that this is an inversed variant of graph *G*. Or mathematically it would be $V(\bar{G}) = V(G)\ and\ \bar{E} = \{e \in \ \bar{E}\ ,\ e \notin \mathrm{E}\}$.

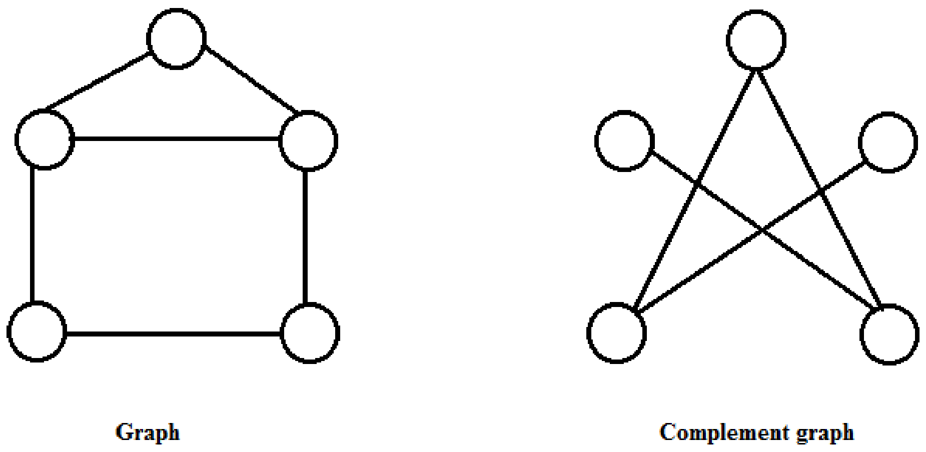

Figure 4. Complement graph.

Definition 8: **Simple graph**

A simple graph is considered to be an undirected graph with finite sets of vertices and edges, which has no loops or multiple edges. We needed to introduce this term because further in this work we are going to use simple graphs for our experiments.

Definition 9: **Subgraph**

A subgraph $G' = (V', E')$ is considered to be a subset of the vertices of graph *G* with the corresponding edges. But not all possible edges may be included (Figure 5). This means that if vertices v_i and v_j are adjacent in graph *G*, then it may happen that on a subgraph of graph *G* they won't have an edge between them. $V' \subseteq V, E' \subseteq E$.

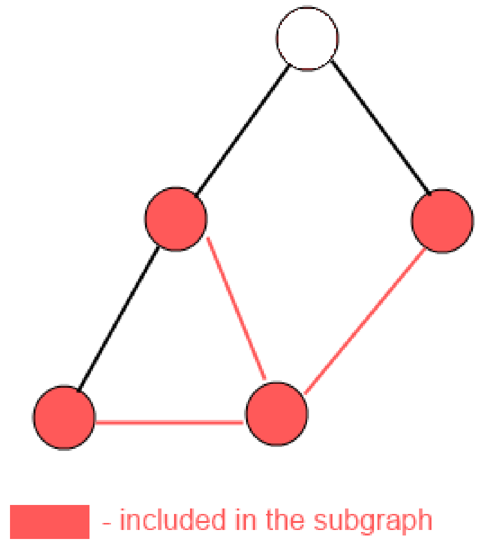

Figure 5. Subgraph.

Definition 10: **Vertex-induced subgraph**

An induced subgraph $G' = (V', E')$ is considered to be a subset of the vertices of graph G with all their corresponding edges (Figure 6). $G[V'] = (V' \subseteq V,\ E' = \{ (v_i, v_j) \mid i \neq j,\ (v_i, v_j) \in E,\ v_i, v_j \in V'\})$.

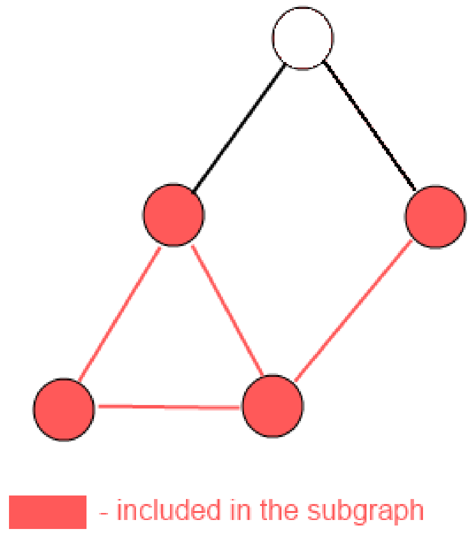

Figure 6. Induced subgraph.

Definition 11: **Complete subgraph**

A complete subgraph $G' = (V', E')$ is considered to be a subset of the vertices of graph G with all their corresponding edges, where each pair of vertices is connected by an edge (Figure 7). We should remember this important term because we are going to need it later on.

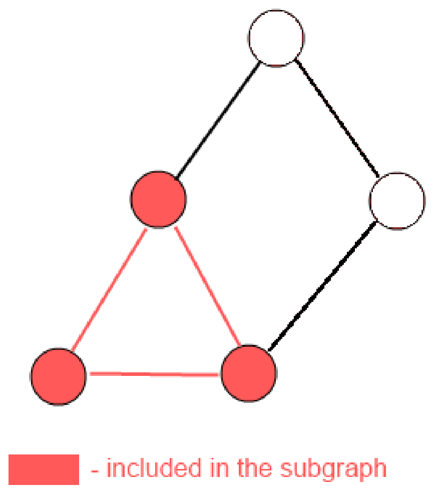

Figure 7. Complete subgraph.

Definition 12: **Clique, maximal clique, clique number**

Clique is a complete subgraph of graph G. The clique V' in graph G is called *maximal* if there does not exist any other V'', such that $V' \subset V''$. The size of the largest maximum clique in graph G is called the *clique number*. [2]

Definition 13: **Independent set**

An independent set (IS) of a graph G is any subset of vertices $V' \subseteq V$, where vertices are not pairwise adjacent. So, it is not hard to conclude that for any clique in graph G, there is an independent set in a complement graph G' and vice versa.

<u>Definition 14:</u> **Vertex coloring**

The assignment of colors to vertices of a graph according to algorithm's construction. If we have an undirected graph $G = (V, E)$, then the process of colors assignment must follow the rules below:

- $(v_i, v_j) \in E,\ i \neq j$
- $c(v_i) \neq c(v_j),\ i \neq j$

Generally, it means that no adjacent vertex must have the same color, the colors of adjacent vertices must be different.

This is the main tool that we are going to use in our work.

<u>Definition 15:</u> **Color class**

A color class is known to be a subset of vertices that belong to a certain color. In other words, all the similarly colored vertices belong to one color class.

<u>Definition 16:</u> **Chromatic number**

A chromatic number of a graph *G* is considered to be the smallest number of colors needed to make a proper coloring of graph *G* [3]. Or we should say that it is the smallest number *k* for which there exists a k-coloring of graph *G* [2]. It is usually denoted by $\chi(G)$.

<u>Definition 17:</u> **Heuristic algorithm**

An algorithm is considered to be heuristic if it finds an approximate solution to the problem in acceptable time. There are many complex problems that need to be solved. But sometimes algorithms take too much time in order to find the best available solution. Why should we wait so long if we could just find a solution, which is acceptable, but may not be the best one there is? That is the main principle of the heuristic algorithm. It gives a considerable solution in a relatively short amount of time.

Definition 18: **Tie**

Situation when vertices have the same saturation degree.

1.3. Graph problems

As it was mentioned before, there is a lot of space for researches in the field of graph theory. Many problems stay acute even nowadays waiting for someone to find a better solution. In this work, we are going to run into several important problems: maximum clique, maximum independent set and coloring problem.

Let's start with the **maximum clique problem**, or shortly **MCP**. The main problem is to find the maximal possible clique, or should we say maximal complete subgraph, of a graph. This means that every two vertices must be pairwise adjacent, joined by an edge (Figure 7). The problem is classified as NP-hard (there are a lot of good books, which have detailed information about complexity, for example "Computer and intractability. A guide to the theory of NP-completeness" by Garey M.R. and Johnson D.S.), the solutions of which is very hard to find by means of conventional methods. Many algorithms and its modifications have been produced to find the maximum clique. The most famous among them are the algorithm that was introduced by Carraghan and Pardalos in 1990 and the one, made by Östergård in 2002 [3]. Due to their popularity and performance, these algorithms became a base for other modifications made specifically to improve the quality and decrease the time spent on finding the maximum clique. The problem of finding the maximum clique has many applications in practice. For example, it could be used in data analyses, for designing error-correction codes and even for computer vision. That is why we can assume this a very important problem and finding a better algorithm would contribute a lot into the current situation.

Another similar problem is the finding of **maximal independent set** (**MIS**). It means that we need to find a subset of vertices that are pairwise nonadjacent or, in other words, none of the vertex in this set must be connected to other vertices of that set (as stated by *definition 13*). The IS is called maximal only if there are no vertices that could be added to the current maximal independent set without ruining its structure. As this problem is closely connected with maximal clique problem, MIS problem is considered to be of NP-hard complexity as well. Also, as stated above, MCP and MIS problems are related in such a way that if we have a clique in a graph, then this clique will be an independent set in the complement graph and vice versa (Figure 8).

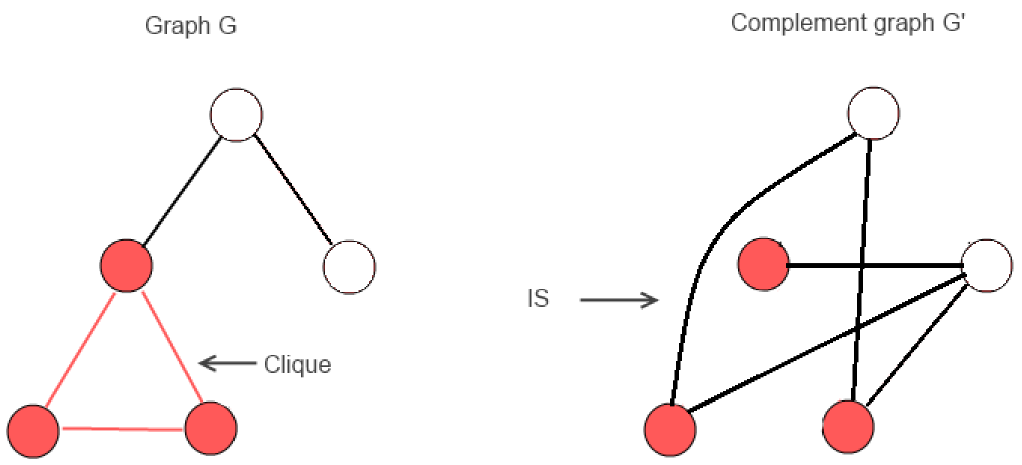

Figure 8. Clique and IS.

And the last problem that will be encountered in our study is **graph coloring problem**, or **GCP**. The main idea is to find the least possible number of colors for coloring a particular graph. It means that any two vertices that have a relationship must be colored differently (*definition 14*). GCP is considered to be a NP-complete problem. It has a lot in common with MIS problem, because all the vertices that share the same color, or are in one color class, can be called an independent set. There are many algorithms made in this field. Also heuristics have been widely used for this problem, for example, a well-known iterative Greedy, made by Welsh and Powell [4], or DSatur algorithm, which was developed by Brelaz. We will talk about them more precisely in *Chapter 2*.

1.4. Research Goals

Now that all the definitions and basic concepts are described, we can move on to revealing the actual goal of our study.

There have been developed quite a lot effective algorithms for the MCP problem. We have already mentioned some of them in the previous subchapters. But the algorithms we are going to take are closely connected to the Tallinn University of Technology. The first one is called VColor-u and was developed in 2005 by Deniss Kumlander. The second algorithm is new, developed only in 2015, and is called VRecolor-BT-u. Its author is Aleksandr Porošin. We are going to describe both of them later in *Chapter 3*, but for now it is necessary to mention that these algorithms work perfectly not on all types of graphs. According to the results, conducted by Deniss Kumlander in his work, VColor-u works effectively on graphs with densities more than 60% [3]. From the results of Aleksandr Porošin's master's thesis, is was found that VRecolor-BT-u algorithm performs well on low to mid densities. Both algorithms use the coloring heuristics under the hood, which is called *Largest-First*. Largest-First coloring algorithm is a sequential heuristic algorithm, which is very fast and provides a decent result in terms of number of colors. However, there are many algorithms that surpass it in the number of colors and sometimes even in time. The question arises immediately: would the change of coloring algorithm help to improve the MCP algorithm itself? That is what we are going to find out in our study. First, we are going to compare different coloring algorithms between themselves according to their results in time and number of colors. Next we will modify the maximum clique algorithms to make them interchangeable in terms of coloring algorithms, so that it would be easy to change the coloring algorithm. And, finally, we are going to conduct our own tests using the DIMACS and random graphs with different densities to see if there are any performance improvements.

As the topic of this thesis is quite extensive, the following goals were determined for our research:

1. Investigate known heuristic coloring algorithms.
2. Define the most efficient coloring algorithms in terms of found color classes.
3. Study the influence of heuristic coloring algorithms on modern maximum clique algorithms (VColor-u, VRecolor-BT-u).
4. Compare performance of maximum clique algorithms with different coloring algorithms and determine if there are any improvements.

1.5. Outline of the Study

Our document is divided into five chapters.

Chapter 1 is mainly a theoretical part of the work. It makes a brief review of the background of our study, describes the main definitions and basic concepts used in this work. As well as that, it reveals the goal of our research and describes briefly the main methods used to achieve it.

Chapter 2 introduces different coloring algorithms, which are going to be used later in the project, their history, different variations and comparison. This allows us to understand how these algorithms should help to improve the maximum clique algorithm.

In order to work with a certain algorithm, it is necessary to know the details about it. So, **Chapter 3** explains the reasons of choosing maximum clique algorithms for our research, briefly describes them, the coloring that they use and their basics. Furthermore, it is told about the improvement of the maximum clique algorithms. All the details about how the algorithms were modified, which coloring algorithms were used and the results of this work are described in this block.

And, finally, in **Chapter 4** it is possible to find the summary of the work and possible future improvements. All the conclusions could be found in this chapter.

2. Coloring Algorithms

2.1. Overview

There are a lot of applications that use a large number of different parameters. Let us say we have a network, where are dozens of nodes. Each node communicates with the nodes that are in the neighborhood and are reachable to it. Imagine that every such communication costs time and money. Sometimes it is necessary to find nodes that are connected to each other not directly, but via an intermediate node. It is possible to model this network as a graph, which has nodes as vertices and its communication links as their edges. Now that we have the graph, we can turn to the help of a coloring algorithm to solve such kind of a problem.

The graph coloring problem is a well-known problem. Its goal is to assign labels to vertices in such a way that no adjacent vertices share the same color. The number of colors used in the process must be as low as possible, thus making the *GCP*'s primary task to minimize this number. These kinds of problem are widely spread and could be found in many computing applications. A striking example of the area, where this problem arises very frequently, is timetabling and scheduling because of the many conflict situations that may occur in the process of allocation of resources. In the first chapter we mentioned that *GCP* problem is of NP-hard complexity, making it very hard to solve. So, resorting to a heuristic approach seems to be very reasonable. It may not have the best performance but can provide one with a solution in a relatively short amount of time.

Many algorithms have been developed to solve the graph coloring problem heuristically. But Greedy remains to be the basic algorithm to assign colors in a graph. It provides a relatively good solution in a small amount of time. The order, in which the algorithm colors the vertices, plays a major role in the process and heavily affects the quality of the coloring. Therefore, there are many algorithms, which employ different ordering heuristics to determine the order before coloring the vertices. These algorithms are mostly based on Greedy but use additional vertex ordering to achieve better performance. As a rule, they surpass Greedy in the number of colors used, producing better results but taking more time to complete. The most popular ordering heuristics are:

a. **First-Fit ordering -** the most primitive ordering existing. Assigns each vertex a lowest possible color. This technique is the fastest amount ordering heuristics.
b. **Degree base ordering** – uses a certain criteria to order the vertices and then chooses the correct one to color. Uses a lot more time compared to First-Fit ordering, but produces much better results in terms of the number of used colors. There are many different degree ordering heuristics, but he most popular among them are:
    - **Random**: colors the vertices of a graph in random order or according to random degree function, i.e. random unique numbers given to every vertex;
    - **Largest-First**: colors the vertices of a graph in order of decreasing degree, i.e. it takes into account the number of neighbors of each vertex;
    - **Smallest-Last**: repeatedly assigns weights to the vertices of a graph with the smallest degree, and removes them from the graph, then colors the vertices according to their weights in decreasing order [5];
    - **Incidence**: sequentially colors the vertices of a graph according to the highest number of colored neighbors;
    - **Saturation**: iteratively colors the vertices of a graph by the largest number of distinctly colored neighbors;
    - **Mixed/Combined**: uses a combination of known ordering heuristics. For example, saturation degree ordering combined with largest first ordering, which is used only to solve situations, when there is a tie, i.e. saturation degree of some vertices is the same.

Sequential algorithms tend to do a lot of tasks that could have been executed simultaneously. That is why many popular algorithms have their parallel versions. In a parallel application, graph coloring is performed in order to partition tasks into subtasks. It means that a certain work that is associated with ordering of vertices or their coloring could be done concurrently. This way it is possible to get a good balance in performance of a coloring algorithm.

Further in this work we are going to describe popular sequential and parallel algorithms, their implementations and results. All of the algorithms have been implemented using C# language.

2.2. Sequential algorithms

2.2.1. Greedy

The original Greedy algorithm was introduced by Welsh and Powell in 1967 [4]. It iterates over the vertices in a graph and assigns each vertex a smallest possible color, which is not assigned to any adjacent vertex, i.e. no neighbor must share the same color. As was mentioned before, it is possible to say that vertices, which are colored by one color, belong to the same color class. If it is impossible to place the vertex into the current color class, then a new color is created. The algorithm can be represented in pseudo-code as follows (Figure 9):

```
1    Let $G = (V, E)$
2    For $i \coloneqq 1$ to $i \coloneqq$ Number of vertices:
3        $c_j = \min C$ , where $C = \{ 1, 2, 3, \dots , m \}$
4        Try to color $v_i$ with $c_j$
5        If no color was found:
6            Create new color class $m \coloneqq m + 1$
7            Color $v_i$ with $c_m$
```

Figure 9. Greedy pseudo-code.

This heuristics is very simple and efficient in terms of time. However, the number of used colors may be not as low as desirable. So, the quality of this algorithm stays relatively low. There could always be found such initial orderings that will dramatically improve the quality. Nevertheless, Greedy remains to be the basic algorithm because of its speed of execution.

2.2.2. Largest-First

Welsh and Powell also suggested an ordering for the greedy algorithm called largest first. It is based on vertices' degrees. The algorithm orders the vertices according to the number of neighbors that each of them has and then starts with the greedy coloring. The pseudo-code can be seen at Figure 10.

```
1    $Let\ G = (V, E)$
2    Order vertices by $\deg(v)$ descending
3    For $i := 1$ to $i :=$ Number of vertices:
4        $c_j = \min C$ , where $C = \{1, 2, 3, \dots, m\}$
5        Try to color $v_i$ with $c_j$
6        If no color was found:
7            Create new color class $m := m + 1$
8            Color $v_i$ with $c_m$
```

Figure 10. Largest-First pseudo-code.

The basic idea of this algorithm is to take care of the vertices with the largest number of neighbors as early as possible because they may contain the highest possible number of conflicts. It works approximately 10-20% better than the Greedy algorithm in terms of the number of colors, whereas the time of completion is almost the same.

2.2.3. Largest-First V2

This is a slightly modified version of Largest-First algorithm. In this algorithm more than one vertex could be colored in each iteration, i.e. after coloring the vertex with the largest number of neighbors, the algorithm also assigns the same color to all the vertices, which follow the rules of coloring - no adjacent vertices must share the same color, and, finally, it removes these vertices from the graph. It is described briefly in the following pseudo-code (Figure 11):

```
1   Let $G = (V, E)$
2   Order vertices by $\deg(v)$ descending
3   $U = V$
4   $C = \{ \}$
5   While $U \neq \emptyset$:
6       Add new color ($c_j$) to $C$
7       Take the first vertex $u$ from $U$
8       Color $u$ with $c_j$
9       Try to color as many vertices as possible with $c_j$
10      Remove the colored vertices from $U$
```

Figure 11. Largest-First V2 pseudo-code.

To show how the algorithm colors the graph, let us go through every step of it. Let us imagine that we have a graph with six vertices as on Figure 12.

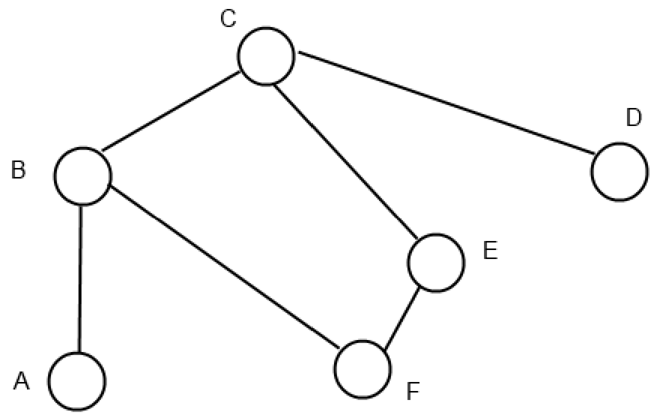

Figure 12. Graph to color.

At first, we have to order the vertices by decreasing degree. The outcome would be as follows: B (3), C (3), E (2), F (2), A (1), D (1). If the algorithm finds a tie, then it just takes the first vertex in the list (different functions can be used to determine the winner, for example, random numbers). After ordering, the algorithms takes the vertex B and colors it, let's say, in red. Then, it tries to color as many vertices as possible in the same color (Figure 13).

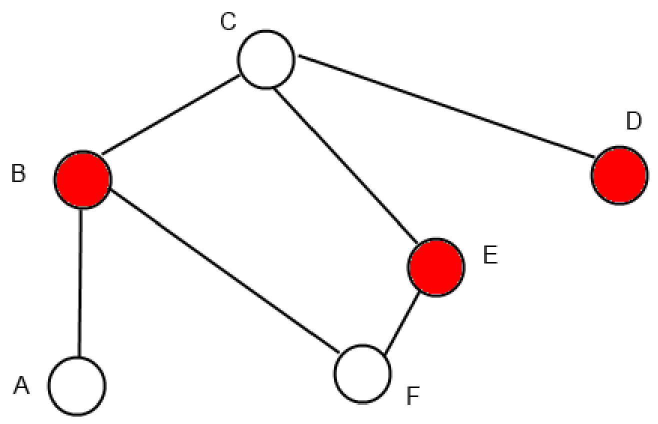

Figure 13. LF: iteration one.

So, after the first iteration we have the following situation: B, E and D are colored red and removed from the list. The remaining ones are C (3), F (2), A (1). When it comes to the second iteration, the LF takes the next color like green and colors the vertex C. After it colors the remaining ones, as they are nonadjacent (Figure 14).

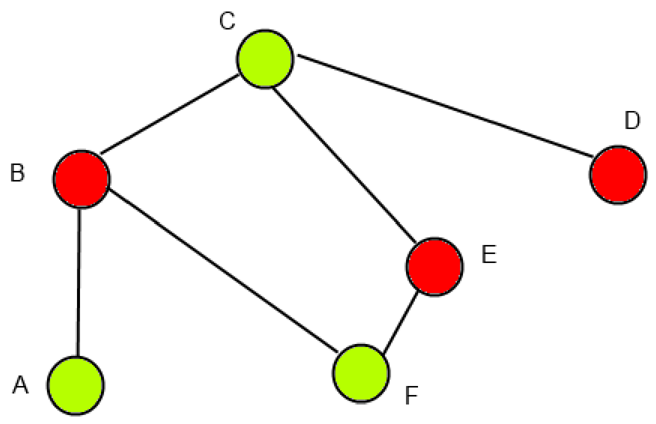

Figure 14. LF: iteration two.

This is the whole process. In this example it is possible to use only 2 colors to color the whole graph.

As can be seen from Figure 11, the structure of the code and the steps needed for the coloring are a little bit different than in the first variation. Although, the algorithm's performance is the same in terms of the number of used colors if compared with the first edition. The time is also almost the same, sometimes even better, however, on the whole the difference is insignificant.

2.2.4. Largest-First V3

Based on the second version we made a third edition of the Largest-First algorithm. The main idea of the algorithm is the same as in V2, however, this time there will be a reordering of vertices in each iteration, meaning that if the vertex is removed from the graph, then its neighbor's degree is decreased. The pseudo-code can be found below (Figure 15).

```
$Let\ G = (V, E)$
$U = V$
$C = \{\ \}$
While $U \neq \emptyset$:
    Order vertices by $deg(u)$ descending
    Add new color ($c_j$) to $C$
    Take the first vertex $u$ from $U$
    Color $u$ with $c_j$
    Try to color as many vertices as possible with $c_j$
    For each neighbor of colored vertices:
        Decrease its degree
    Remove the colored vertices from $U$
```

Figure 15. Largest-First V3 pseudo-code.

Surprisingly, this version of Largest-First gives the best results among them, although, it takes a little bit longer for it to finish the coloring of a graph.

2.2.5. DSatur

This heuristic algorithm was developed by Daniel Brelaz in 1979 [6]. The core idea of it is to order the vertices by their saturation degrees. This means that reordering happens in each iteration. The algorithm orders the vertices by decreasing saturation degree, i.e. the largest number of distinct colors used by neighbors. If a tie occurs, then the vertex is chosen by the largest number of uncolored neighbors. By assigning colors to a vertex with the largest number of distinctly colored neighbors, DSatur minimizes the possibility of setting an incorrect color [3]. Here is the pseudo-code of the algorithm (Figure 16):

```
Let $G = (V, E)$
$U = V$
While $U \neq \emptyset$:
    Order vertices by decreasing saturation degree $deg(u)$
    If a tie, then order by descending number of colored neighbors
    Take the first vertex $u$ from $U$
    Find the minimum color $c_j$ not used in its neighborhood
    Color $u$ with $c_j$
    For each neighbor of $u$:
        Increase its saturation degree if color is not in the neighborhood
        Decrease the number of uncolored neighbors
    Remove the colored vertex from $U$
```

Figure 16. DSatur pseudo-code.

The algorithm works a lot better than the Greedy algorithm, approximately 27-30%. The number of colors used is indeed a lot smaller than if using Greedy, however, it comes at a cost of time, which 3-4 times slower. Anyway, if DSatur is going to be used only as a coloring algorithm in a bigger algorithms than the time it uses is insignificant if compared to the time of the whole algorithm.

2.2.6. DSatur V2

There is another interesting version of DSatur that is worth mentioning [7]. This edition of the algorithms uses a little bit different concept. At first, it finds a largest clique of graph and assigns each a distinct color. Then, it just removes the newly colored vertices from the graph. After this procedure, the algorithm executes as the previous DSatur. It would be easier to see it on Figure 17.

```
1   Let $G = (V,E)$
2   $U = V$
3   Find the largest clique $U^*$ of $U$
4   Assign each vertex from $U^*$ a possibly low distinct color
5   $U = U \setminus U^*$
6   While $U \neq \emptyset$:
7       Order vertices by decreasing saturation degree $deg(u)$
8       If a tie, then order by descending number of colored neighbors
9       Take the first vertex $u$ from $U$
10      Find the minimum color $c_j$ not used in its neighborhood
11      Color $u$ with $c_j$
12      For each neighbor of $u$:
13          Increase its saturation degree if color is not in the neighborhood
14          Decrease the number of uncolored neighbors
15      Remove the colored vertex from $U$
```

Figure 17. DSatur V2 pseudo-code.

In order to find the largest clique and not spend a lot of time, we have chosen to use the Greedy algorithm. But wait, you would not agree with me, stating that this algorithm is for coloring. Yes, that is correct. However, it is possible to find the clique by inverting the graph and finding the color class with the largest number of vertices inside. This is the case.

The greedy algorithm takes a complement graph, finds the largest independent set and colors it with a distinct color. Then, removes these vertices from the graph and starts working as the first version of DSatur. Although, the performance stays the same in terms of number of colors used and time.

2.2.7. Incidence degree ordering (IDO)

This ordering was firstly introduced by Daniel Brelaz [6] and was modified by Coleman and More in their work [8]. In one word, it is a modification of the DSatur algorithm. The main principle of this heuristic is to order vertices by decreasing number of the vertices' colored neighbors. If a tie occurs, it can be decided, which vertex is going to be chosen, by the usage of random numbers. The coloring itself is done by the Greedy algorithm. The pseudo-code is on Figure 18.

```
1    Let $G = (V, E)$
2    $U = V$
3    While $U \neq \emptyset$:
4        Order vertices by decreasing number of colored neighbors
5        If a tie, then order by descending random number
6        Take the first vertex $u$ from $U$
7        Find the minimum color $c_j$ not used in its neighborhood
8        Color $u$ with $c_j$
9        For each neighbor of $u$:
10           Increase its degree
11       Remove the colored vertex from $U$
```

Figure 18. IDO pseudo-code.

The performance of this algorithm is almost the same as with Largest-First, however, the time consumption is about 2 times more compared to the Largest-First. But it is faster than the DSatur algorithm, although, the number of colors is larger.

2.2.8. MinMax

The MinMax algorithm was introduced by Hilal Almara’Beh and Amjad Suleiman in their work in 2012 [9]. The main function of this algorithm is to find the maximum independent set, but it could be used for coloring purposes as well because independent sets are color classes.

The algorithm starts by finding the vertex with minimum degree. If a tie exists, then it takes the vertex with the lowest original number, colors it, places into the independent set list and then removes it and its neighbors from the local graph. The same procedure is repeated until there are no vertices to choose from. Then it removes the newly colored vertices from the main graph and starts over but this time taking the vertex with maximum degree. The algorithm alternates the ordering in each iteration. This can be seen from the pseudo-code on Figure 19.

```
Let $G = (V, E)$
$U = V$
$useMax = false$
$c = 1$
While $U \neq \emptyset$:
    $localGraph = U$
    While $localGraph \neq \emptyset$:
        If $useMax$:
            Find vertex with max degree
        Else:
            Find vertex with min degree
        Color the vertex with $c$
        Remove this vertex and its neighbors from $localGraph$
    $c = c + 1$
    $useMax = !useMax$
    Remove the colored vertices from $U$
```

Figure 19. MinMax pseudo-code.

The MinMax algorithm works almost as the Greedy algorithm in terms of the number of colors used, but works 2.5 times longer. The results are not so good, compared to DSatur or even the largest first degree ordering. Nevertheless, the unique option of this algorithm is to gather as many vertices as possible in one color class and it may help us further.

2.3. Mixed/combined algorithms

2.3.1. IDO-LDO

This algorithm is a combination of incidence degree ordering and largest-first ordering heuristics. As a primary heuristics we use IDO. If a tie occurs, then it will be decided, which vertex is going to be taken, by the largest number of neighbors. So, basically, it is the same IDO algorithm but with the add-in of LDO. This can be clearly seen on the Figure 20.

```
1    Let $G = (V, E)$
2    $U = V$
3    While $U \neq \emptyset$:
4        Order vertices by decreasing number of colored neighbors
5        Then order by decreasing number of neighbors
6        If a tie, then order by descending random number
7        Take the first vertex $u_i$ from U
8        Find the minimum color $c_j$ not used in its neighborhood
9        Color $u_i$ with $c_j$
10       For each neighbor of $u_i$:
11           Increase the number of colored neighbors
12           Decrease the number of neighbors
13       Remove the colored vertex from $U$
```

Figure 20. IDO-LDO pseudo-code.

This algorithm works slightly better than the original Largest-First in terms of the number of colors used, however, it takes longer to execute than the Largest-First.

2.3.2. IDO-LDO-Random

IDO-LDO-Random algorithm is another modified IDO algorithm. This time the random numbers' function was added to decide in a situation of a tie. At first, the algorithm orders the vertices by the largest number of colored neighbors, then by the largest number of neighbors and then, if there are two or more vertices with the exact same details, the one with the largest random number is chosen. There is no need to include the pseudo-code for this particular algorithm because it is absolutely the same as the previous one with the exception that random numbers have been added.

In most cases the performance of this algorithm is better than of Largest-First, IDO or IDO-LDO. But the speed of its execution is slower. Nevertheless, it was worth mentioning this kind of algorithm as well.

2.3.3. LDO-IDO

This modification was introduced by Dr. Hussein Al-Omari and Khair Eddin Sabri in their work in 2006 [10]. The basic heuristic for this algorithm is the Largest-First. If a tie occurs, then the IDO heuristic decides, which vertex to take. On the whole, this is almost the same algorithm as the Largest-First V3 with an IDO function inside, in one word, the first ordering is being done by the largest number of neighbors and then by the largest number of colored neighbors. The pseudo-code could be found on Figure 21.

```
Let $G = (V, E)$
$U = V$
$C = \{ \}$
While $U \neq \emptyset$:
    Order vertices by $deg(u_i)$ descending
    Then if a tie, order by the number of colored neighbors decreasing
    Add new color ($c_j$) to $C$
    Take the first vertex $u$ from $U$
    Color $u$ with $c_j$
    Try to color as many vertices as possible with $c_j$
    For each neighbor of colored vertices:
        Decrease its degree
        Increase the number of colored neighbors
    Remove the colored vertices from $U$
```

Figure 21. LDO-IDO pseudo-code.

This algorithm works more efficiently than the Greedy algorithm in terms of the quality and slightly better than the Largest-First and Largest-First V2. However, it uses much more time to accomplish its goal.

2.3.4. DSatur-LDO

This modification of the DSatur algorithm was also introduced by Dr. Hussein Al-Omari and Khair Eddin Sabri in their work in 2006 [10]. The algorithm works as DSatur but if a tie occurs, then Largest-First algorithm steps into the action to solve the conflict.

According to the results, this heuristic works a little better than the original DSatur within the same amount of time.

2.3.5. DSatur-IDO-LDO

In this algorithm ties are resolved by Incidence Degree Ordering at first, then the remaining ties are resolved by the Largest Degree Ordering [11].

On the whole, this algorithm outperforms the DSatur and DSatur-LDO heuristic regardless the fact that it uses a little bit more time to complete. It is quite an effective modification to use further in our work.

2.4. Parallel algorithms

2.4.1. Jones and Plassmann algorithm

The algorithm was firstly proposed by Jones and Plassmann in their work in 1993 [12]. The algorithm is based on the Lubys parallel algorithm [13]. The core idea was to construct a unique set of weights at the beginning that would be used throughout the algorithm itself. For example, random numbers. Any conflict of the same random numbers is solved by the vertex number. Each iteration the JP algorithm finds the independent set of a graph, i.e. all the vertices, which weight is higher than the weight of the neighboring vertices, and then assigns colors to these vertices using the Greedy algorithm. Every action is done in parallel. The pseudo-code can be seen on Figure 22.

```
1  Let $G = (V, E)$
2  $U = V$
3  While $U \neq \emptyset$:
4      For each $u \in U$ do in parallel:
5          find a set of vertices $I$, such that $\forall v \in neighbors\ of\ U: w(u) > w(v)$
6          For each $i \in I$ do in parallel:
7              color $i$ with minimum color
8      Remove $I$ from $U$
```

Figure 22. JP pseudo-code.

The JP algorithm can be called Parallel Greedy. It gives almost the same results in terms of used colors but does a lot of subtasks simultaneously. The differences can occur only because of the random numbers, because they are assigned randomly to every vertex.

In our work we used random numbers as well, but decided to use the unique ones, so that there would not be two vertices with the same random number.

2.4.2. Jones and Plassmann V2

Another version of JP algorithm was introduced by William Hasenplaugh, Tim Kaler, Tao B. Schardl and Charles E. Leiserson in their work in 2014 [5]. The idea behind the modification was to use recursion. The algorithm orders the vertices in the order of function p, which generates random numbers. It starts by partitioning the neighbors of each vertex into predecessors (the vertices with larger priorities) and successors (the vertices with lower priorities) [5]. If there are no vertices in predecessors, then the algorithm begins coloring. It has a helper function named JpColor, which uses recursion to color the vertices. The color is chosen by collecting all the colors from the predecessors and choosing the smallest possible (this is being done in the GetColor helper function). When the vertex with the empty predecessors list is colored, the algorithm searches for changes in this vertex successors list for vertices with counter equals to zero (it means that all of the predecessors have been colored) and starts coloring them. All this is done in parallel subtasks. It is possible to see all the operations from the pseudo-code on Figure 23.

```
JP($G$):
1   Let $G = (V, E, p)$
2   For each $v \in V$ do in parallel:
3       find a set of predecessors, where $p(u) > p(v)$
4       find a set of successors, where $p(u) < p(v)$
5       $v.counter = |predecessors|$
6   For each $v \in V$ do in parallel:
7       if list of predecessors = $\emptyset$
8           JpColor($v$)

JpColor($v$):
1   $v.color = GetColor(v)$
2   For each $u \in v.successors$ do in parallel:
3       if $u.counter == 0$:
4           JpColor($u$)

GetColor($v$):
1   $C = \{1, 2, 3, \dots, |v.predecessors| + 1\}$
2   for each $u \in v.predecessors$ do in parallel:
3       $C = C - u.color$
4   find the minimum color of $C$ and return it
```

Figure 23. JP V2 pseudo-code.

Unfortunately, no big differences were found in the performance of this algorithm neither by authors nor during our research. The time and quality is the same as with the original JP algorithm.

As with the original JP algorithm, in our work we decided to use the unique random numbers, so that there would not be two vertices with the same random number.

2.4.3. Parallel Largest-First

As a base algorithm for Parallel Largest-First is used JP algorithm, but as the heuristic – Largest-First. The main difference is that instead of weights system, used in JP, here they are replaced by finding the largest degree of each vertex. However, random numbers are not removed. They help to solve situations, when two vertices have the same number of neighbors.

As expected, it works better than the original JP in terms of quality. However, the number of colors in the output is compared to this number of the original Largest-First algorithm.

As with the original JP and JP V2 algorithms, in our work we used the unique random numbers to exclude the chance that two vertices have the same random number.

2.4.4. Parallel Smallest-Last

The Smallest-Last heuristics was firstly introduced by Matula in his work in 1972 [14]. He tried to improve the Largest-First algorithm by a completely different approach. The SL heuristic's system of weights is more sophisticated and complex. The algorithm uses two phases [15]:

1. Weighting phase
2. Coloring phase

The weighting phase begins by finding vertices that correspond to the current smallest degree in the graph. These vertices are assigned the current weight and removed from the graph. The degree of all the neighbors of deleted vertices are decreased. All these steps are repeated until every vertex receives its weight. In our work if a tie occurs, then it is decided which vertex to take by random numbers. They are given to every vertex during the first phase (Figure 24). In the following pseudo-code it is possible to view the weighting phase (Figure 25).

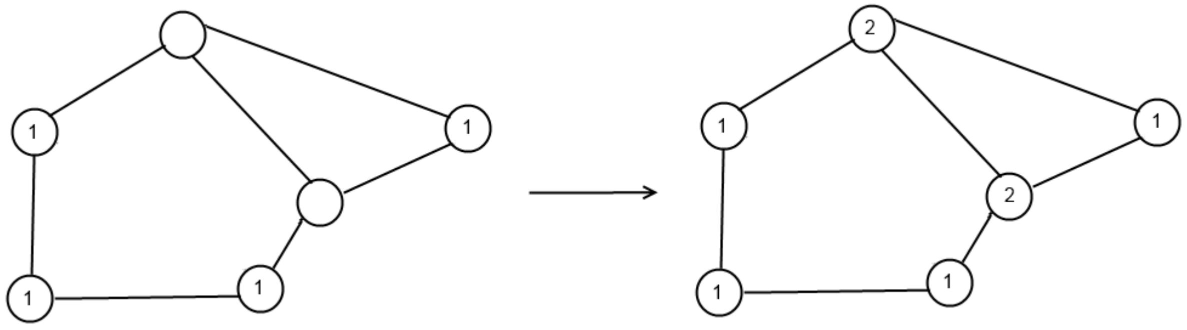

Figure 24. Parallel SL - weighting phase.

```
1   Let $G = (V, E)$
2   $currentWeight = 1$
3   $currentDegree = 1$
4   $U = V$
5   While $\exists\, I = \{\, vertices\ u \in U\ with\ \deg(u) \leq currentDegree \,\}$ do in parallel:
6       For each $i \in I$ do in parallel:
7           $weight(i) = currentWeight$
8           Remove $i$ from $U$
9       $currentWeight = currentWeight + 1$
10      $currentDegree = currentDegree + 1$
```

Figure 25. Parallel SL weighting phase pseudo-code.

After the weighting phase goes the coloring phase. The vertices get colored by their weight in decreasing order. So, the vertices with weight 2 are colored at first, then the vertices with weight 1 (Figure 26).

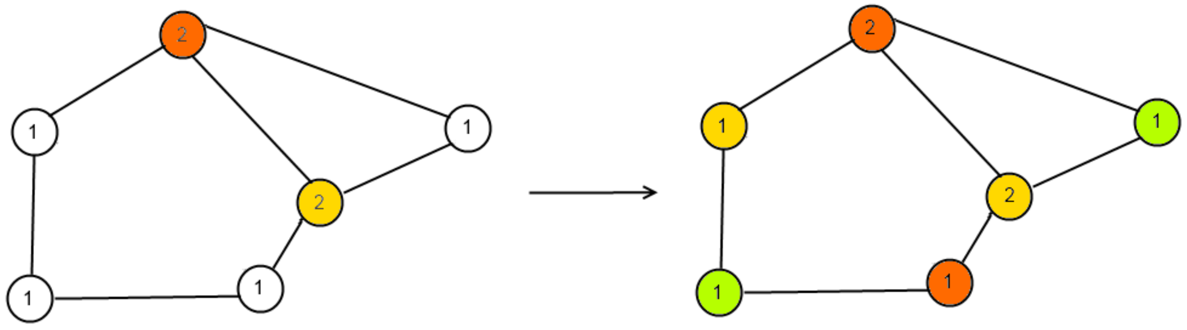

Figure 26. Parallel SL - coloring phase.

The algorithm was supposed to be better than the largest first ordering heuristics and it does in practice. The number of used colors is smaller than if using the parallel largest-first, though the time taken is a little longer.

As with the original JP and JP V2 algorithms, in our work we used the unique random numbers to exclude the chance that two vertices have the same random number.

2.4.5. Non-Parallel Implementations

In theory, parallel implementations should perform better, however, this is the place where our chosen language's peculiarities could bring in unexpected and undesired changes. The main idea is to compare the behavior of parallel and non-parallel implementations of the same algorithm in practice to exclude the possibilities of incorrect results. These algorithms include:

- Greedy From Parallel – non-parallel copy of Jones and Plassmann algorithm;
- Greedy V2 From Parallel – non-parallel copy of Jones and Plassmann V2 algorithm;
- Largest-First From Parallel – non-parallel copy of Parallel Largest-First algorithm;
- Smallest-Last From Parallel – non-parallel copy of Parallel Smallest-Last.

2.5. Tests and results

In this chapter we are going to conduct tests to determine the most acceptable coloring algorithms that will be used further in our research. In order to choose the best algorithms it is necessary to agree on parameters that are going to be compared. In our case they are **number of used colors**, **time** in milliseconds and **density** of the graph. As our main test subject we have chosen the Greedy algorithm. The reason of this choice is pretty obvious: this algorithm is used in most of the modern maximum clique algorithms because its execution time is very low.

Tests are divided into two groups: **random graphs** and **DIMACS graphs** tests. The first part of the chapter describes the former tests results and the second one shows the analysis of the latter tests results.

Tests on randomly generated graphs give a general overview of the algorithms' performance. Test cases are divided by graphs density, starting from 10% and ending with 90%. Results are given in charts and grouped by coloring algorithm's type (sequential, combined or parallel). The computational results of tests are rough and they give us only an initial overview of coloring algorithms' performance.

DIMACS graphs come from a special package of graphs used in the Second DIMACS Implementation Challenge, which provides different types of graphs to make tests on. They all have their own structure according to the specific problem they try to solve. Results are given in numeric values as well as in the view of graphs and grouped by coloring algorithm's type (sequential, combined or parallel).

All algorithms were implemented using C# language in Visual Studio 2013 Professional (.NET Framework 4.5). System characteristics:

- Processor: Intel Core 2 Quad Q9550 2.83GHz
- RAM memory: 8GB
- Operating System: Windows 7 professional 64-bit

2.5.1. Randomly generated graphs

2.5.1.1. Overview

First of all, we are going to research the randomly generated graphs. As stated above, it will give us a picture of the algorithms, their performance and consumed time.

For each density starting from 10% and up to 90% a new random graph was generated and tested with all the algorithms that we had.

Due to quite large number of graphs, which show algorithms' execution time, for readability purpose it was decided to place them in Appendix 1 – Coloring algorithms: Randomly Generated Graphs Test Results.

2.5.1.2. Random graphs' generation function

Before we start analyzing the results of randomly generated graphs, it is necessary to specify the way these graphs were made. The generation function that was used to achieve this goal can be found on Figure 27.

The randomness of graphs had been achieved by using .NET native Random class. The graph itself is an object, which contains the adjacency matrix inside the Values array and number of edges in Edges property. The algorithm generates random vertices x and y in range from 1 to the specified number of vertices and connects them together by simply putting Boolean "true" value into the 2-dimensional array.

The parameters used by this method are:

- "nodes" parameter - the number of vertices of a graph;
- "density" parameter - the specified density of a graph.

```
        public static Graph GenerateGraph(int nodes, double density)
        {
            if (density < 0 || density > 1)
                throw new Exception("0 <= density <= 1");
            int numberOfEdges = Convert.ToInt32(Math.Round(nodes * (nodes - 1) * den-
sity / 2, 0));

            var graph = new Graph
            {
                Values = new bool[nodes, nodes],
                Edges = numberOfEdges
            };

            var random = new Random();
            Thread.Sleep(40);
            var random2 = new Random();

            int x, y;
            for (int i = 0; i < numberOfEdges; i++)
            {
                do
                {
                    x = random.Next(0, nodes);
                    y = random2.Next(0, nodes);
                } while (x == y || graph.Values[x, y]);
                graph.Values[x, y] = true;
                graph.Values[y, x] = true;
            }

            return graph;
           }
```

Figure 27. Random graph's generation function.

2.5.1.3. Sequential algorithms

As can be seen from the charts, every algorithm performs better than the Greedy algorithm in terms of number of used colors almost on every density.

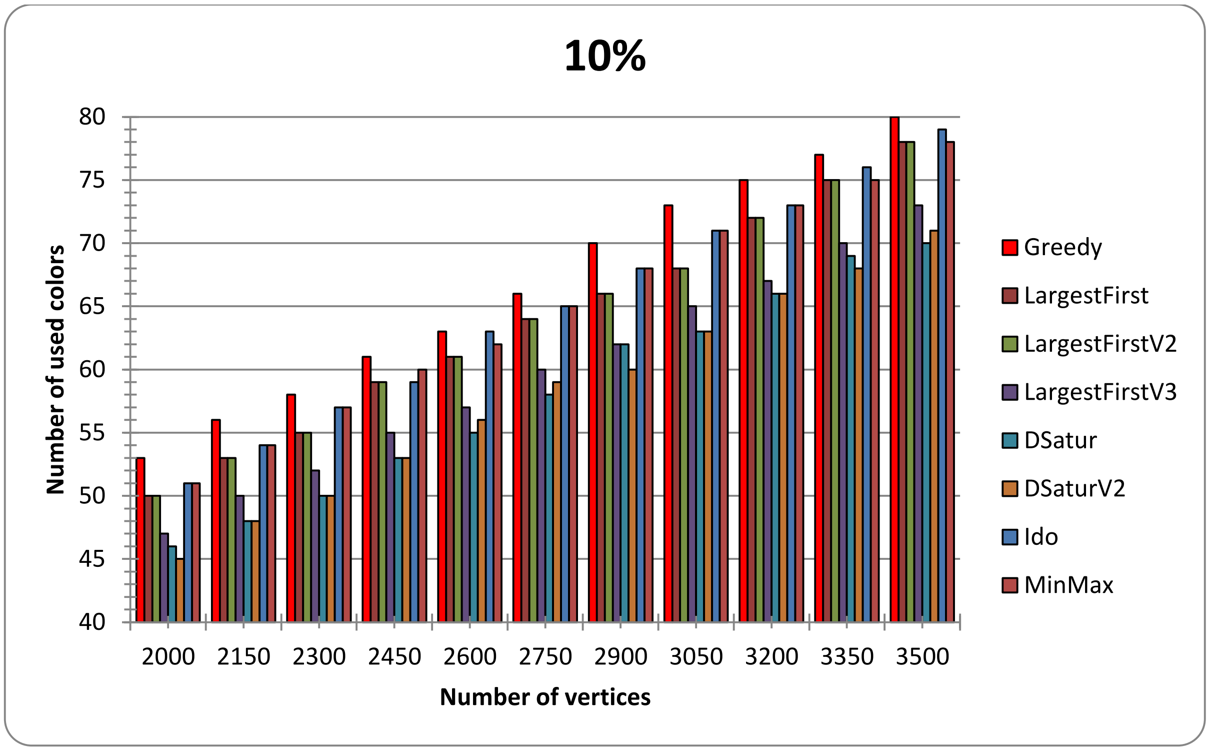

Figure 28. Randomly generated graphs tests' results compared in used colors. Sequential algorithms, density 10%.

However, for example, on 40% density IDO and MinMax algorithms' performance is similar to that of the Greedy algorithm (Figure 29). Also it is worth mentioning that in some particular cases Largest-First and Largest-First V2 algorithms used the same number of colors as the Greedy algorithm, while taking more time to achieve this goal (Figure 29, number of vertices – 560; Figure 30, number of vertices - 380).

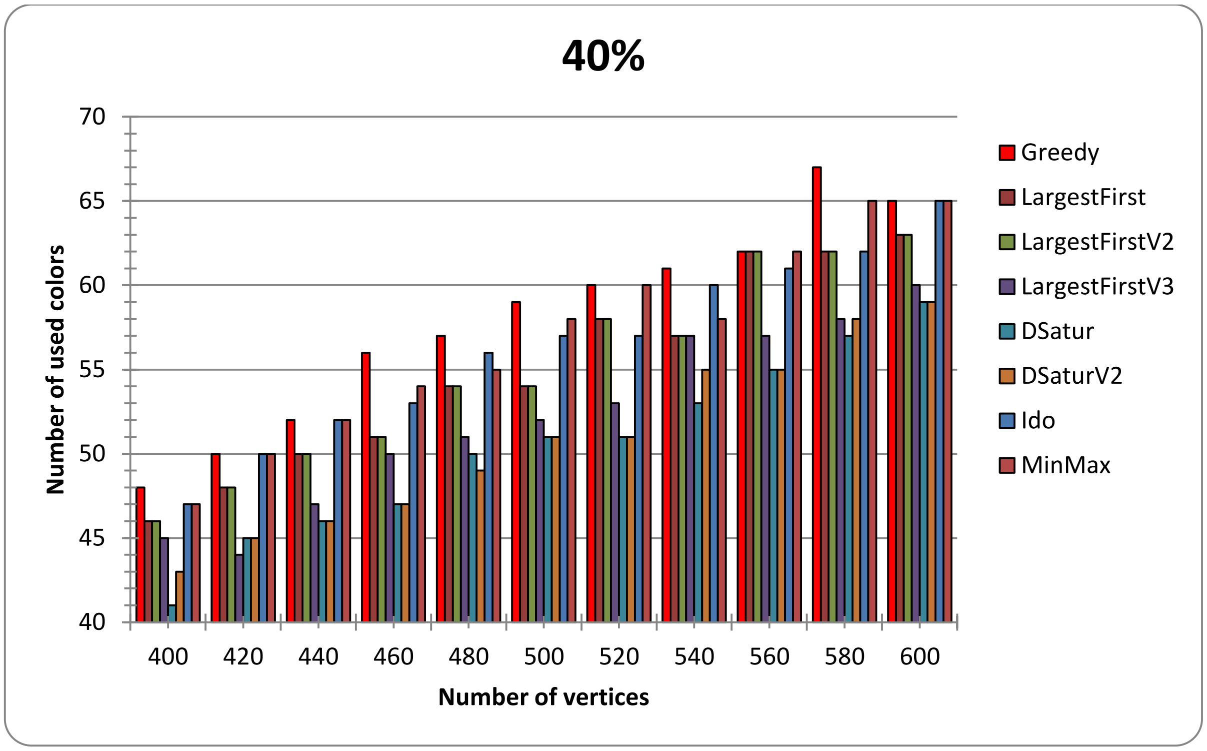

Figure 29. Randomly generated graphs tests' results compared in used colors. Sequential algorithms, density 40%.

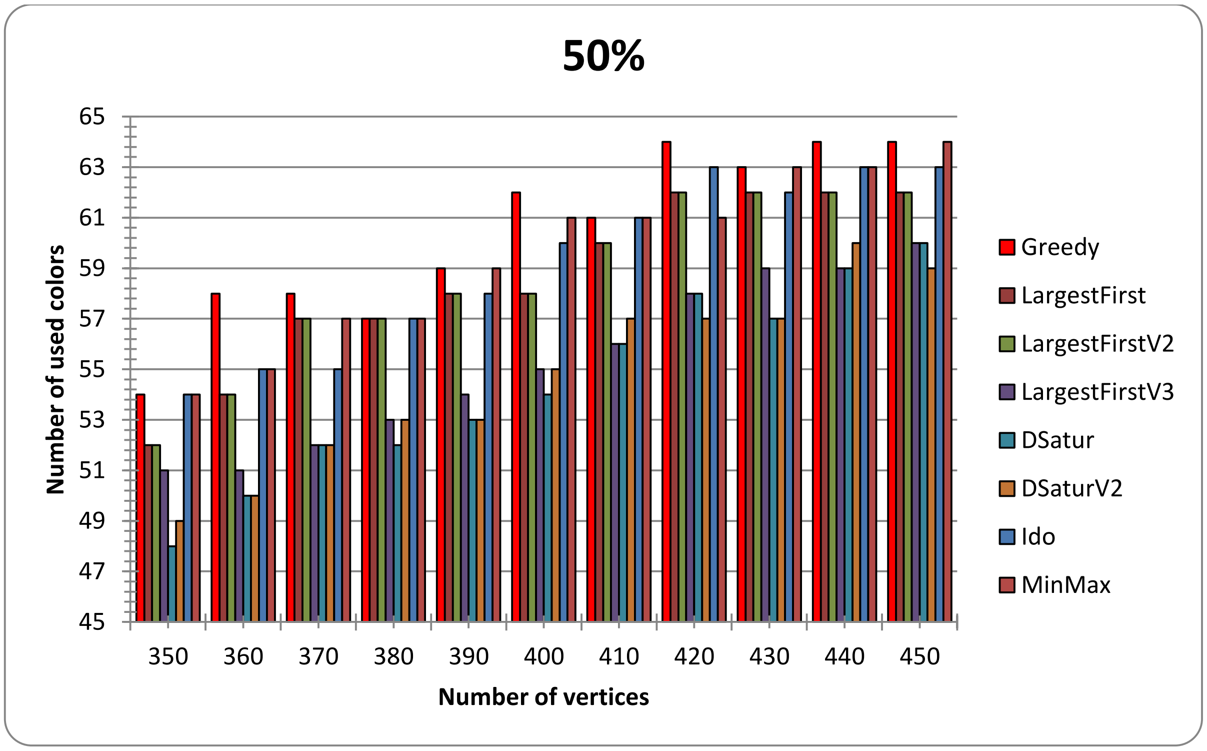

Figure 30. Randomly generated graphs tests' results compared in used colors. Sequential algorithms, density 50%.

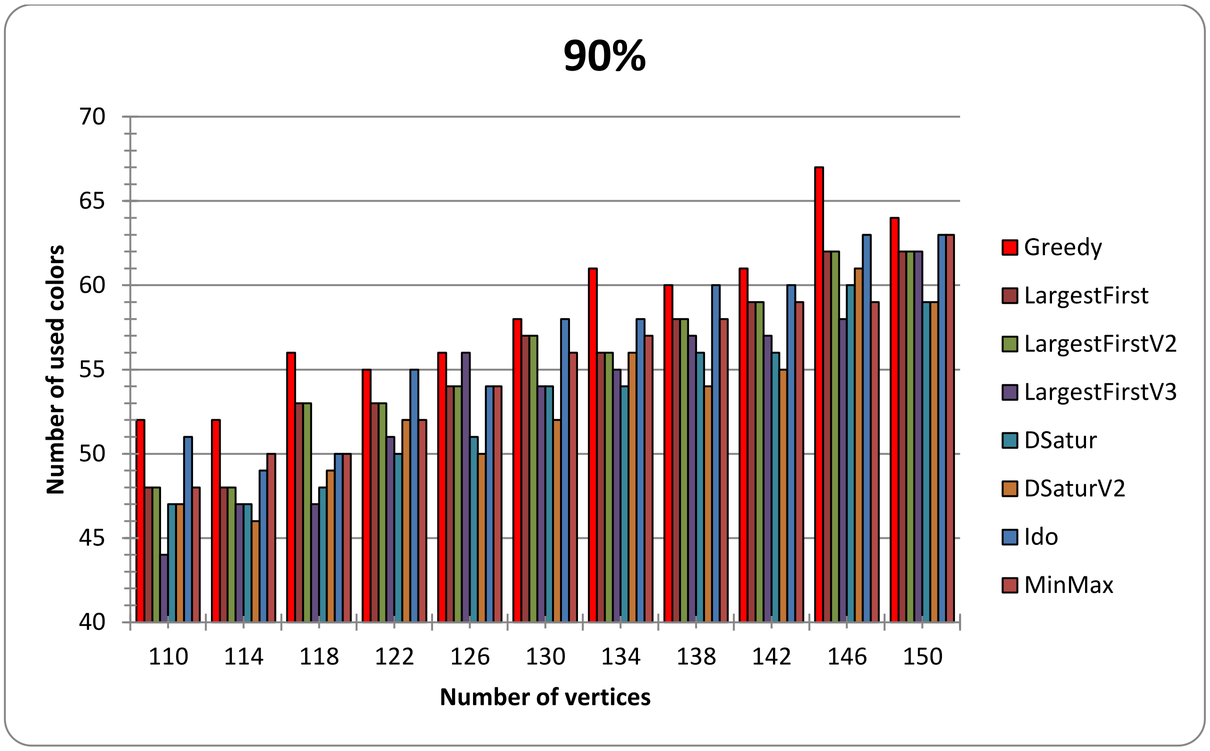

Figure 31. Randomly generated graphs tests' results compared in used colors. Sequential algorithms, density 90%.

The best results in terms of used colors among all the sequential algorithms produced DSatur, DSatur V2 and Largest-First V3. Their performance is much better compared to the Greedy algorithm, however, it comes with a cost of taking more time to complete (results of consumed time can be seen in Appendix 1 – Coloring algorithms: Randomly Generated Graphs Test Results: Sequential algorithms). This behavior can be explained by the fact that graph's vertices undergo a heavy sorting in the process of algorithm's execution unlike the situation with the Greedy algorithm.

It is also possible to see the dependency of DSatur and DSatur V2 execution time from density. The higher the density, the less the time it took for these algorithms to complete: from almost 18 seconds on 10% density to 60 milliseconds at 90% density. Furthermore, in the beginning it takes slightly more time for DSatur V2 to complete its execution compared to DSatur, however, when density reaches 70% the situation changes and DSatur V2 begins to outperform its sibling.

2.5.1.4. Combined algorithms

At first sight, it seems that the results of combined algorithms are very similar to the sequential ones. There are also three leading algorithms, which this time are: DSatur-LDO, DSatur-IDO-LDO and LDO-IDO. Their results in terms of used colors are much better than the Greedy one. However, it is possible to find a dependency. The higher the density is, the more similar performance of algorithms could be found. From 10% to 70% density we can see clear division between these three algorithms and the rest. However, at 80% density and higher (Figure 35, Figure 36) the difference begins to vanish, although the lead in the number of used colors remains.

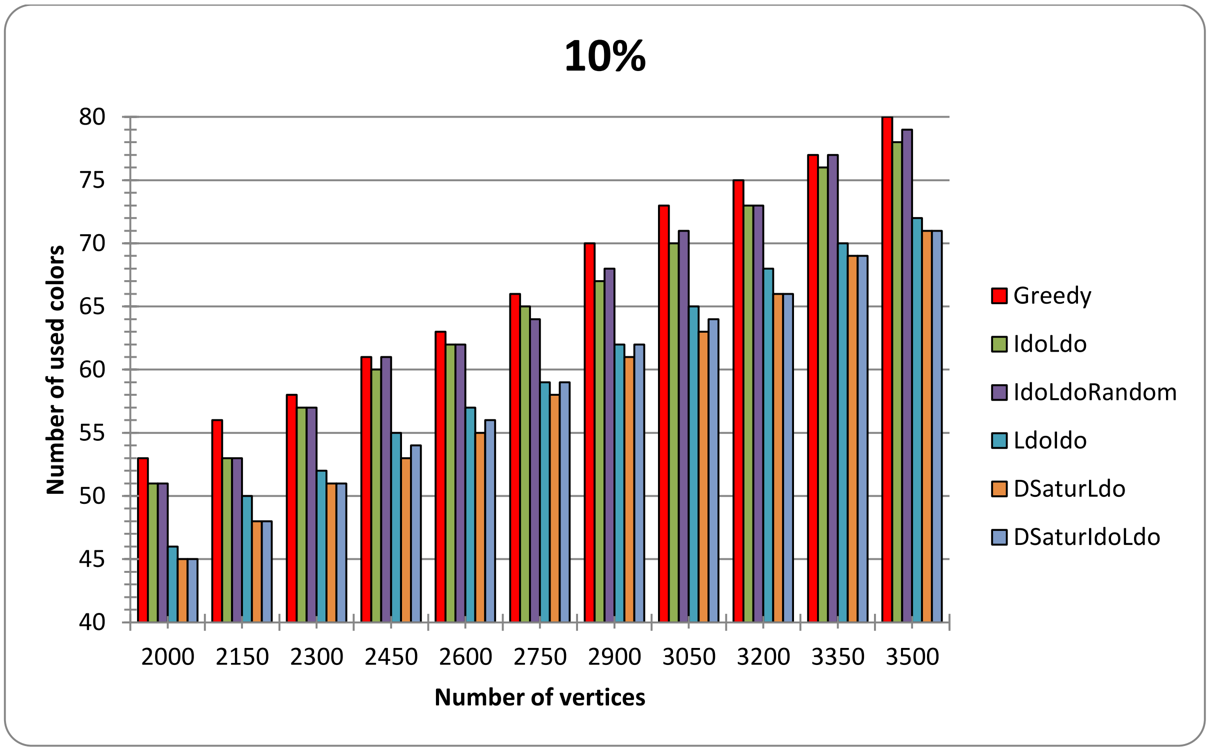

Figure 32. Randomly generated graphs tests' results compared in used colors. Combined algorithms, density 10%.

Furthermore, it is possible to clearly see a very strange behavior of IDO-LDO-Random algorithm at Figure 34 and Figure 35 – it used more colors than the Greedy algorithm in some cases. Does it mean that we have a mistake in our algorithm? This behavior might be caused by the fact that random numbers are used during execution of IDO-LDO-Random algorithm and should be investigated by a separate research.

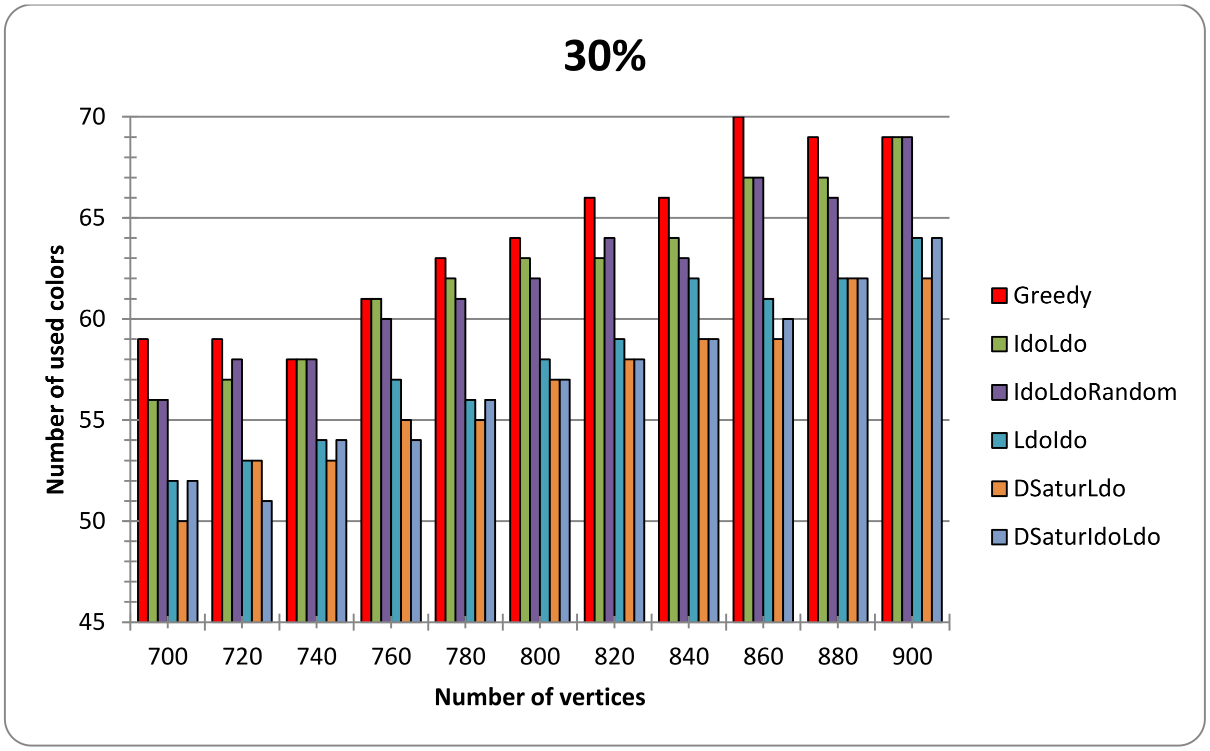

Figure 33. Randomly generated graphs tests' results compared in used colors. Combined algorithms, density 30%.

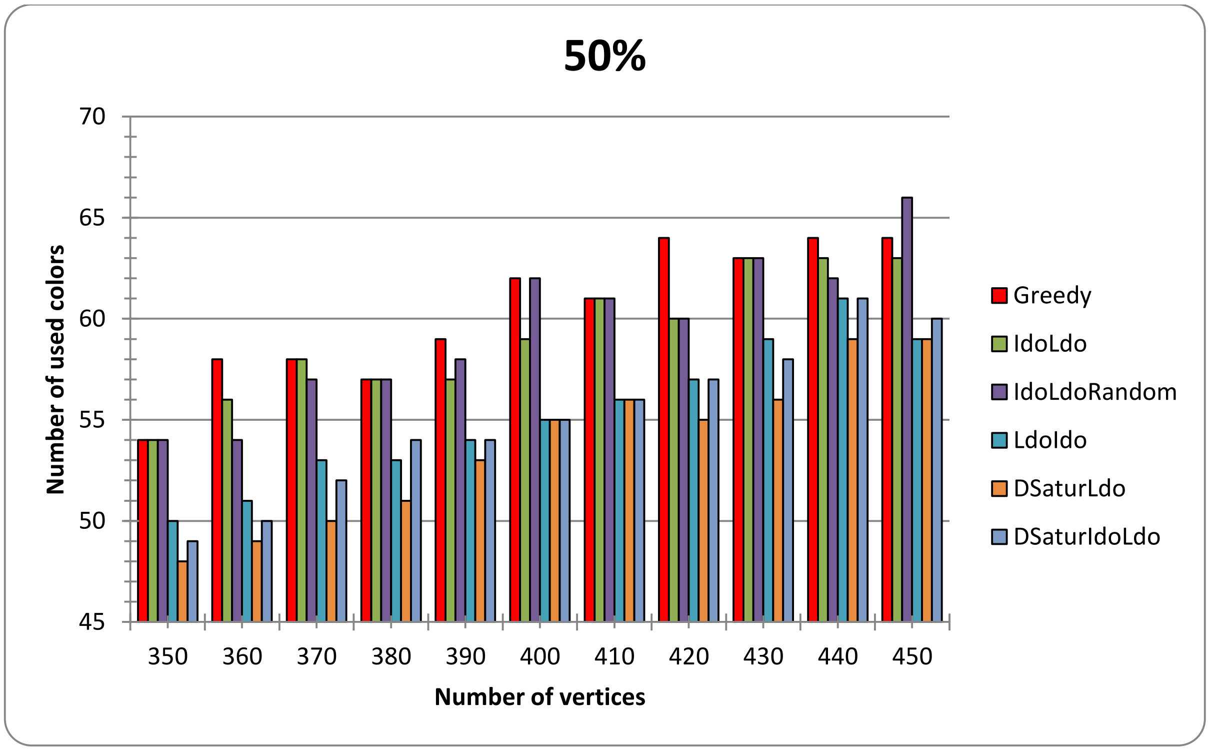

Figure 34. Randomly generated graphs tests' results compared in used colors. Combined algorithms, density 50%.

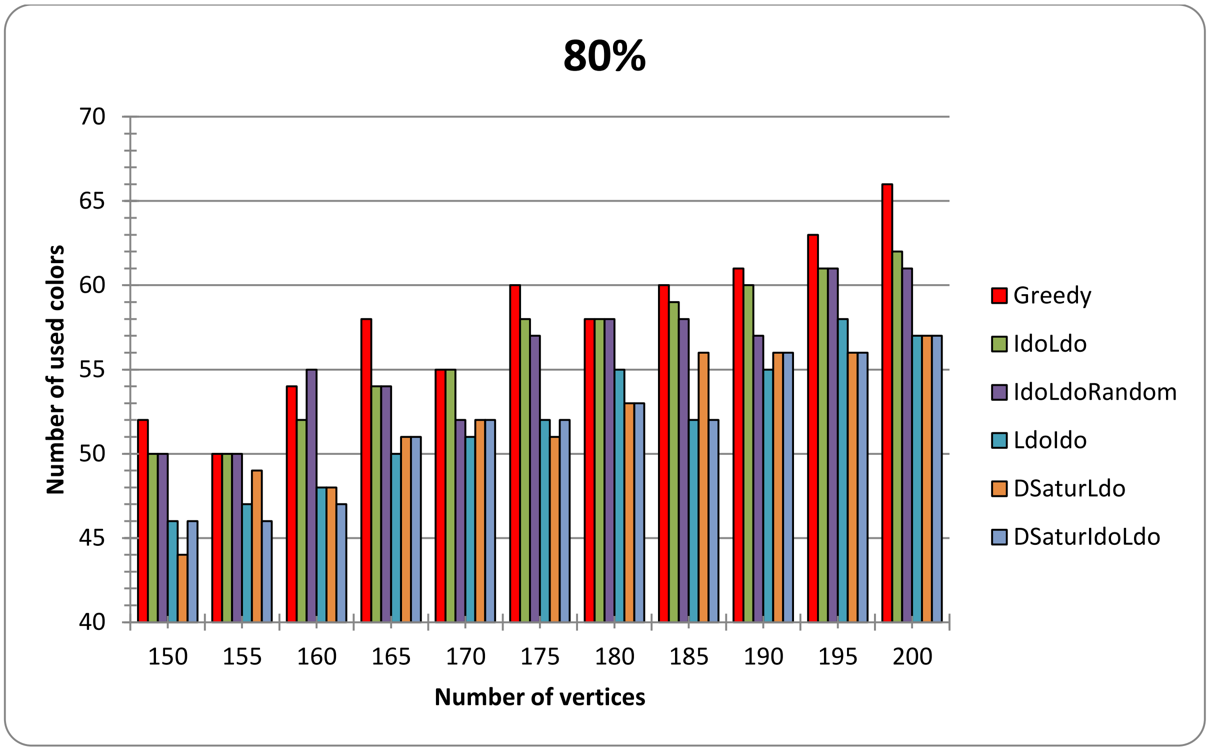

Figure 35. Randomly generated graphs tests' results compared in used colors. Combined algorithms, density 80%.

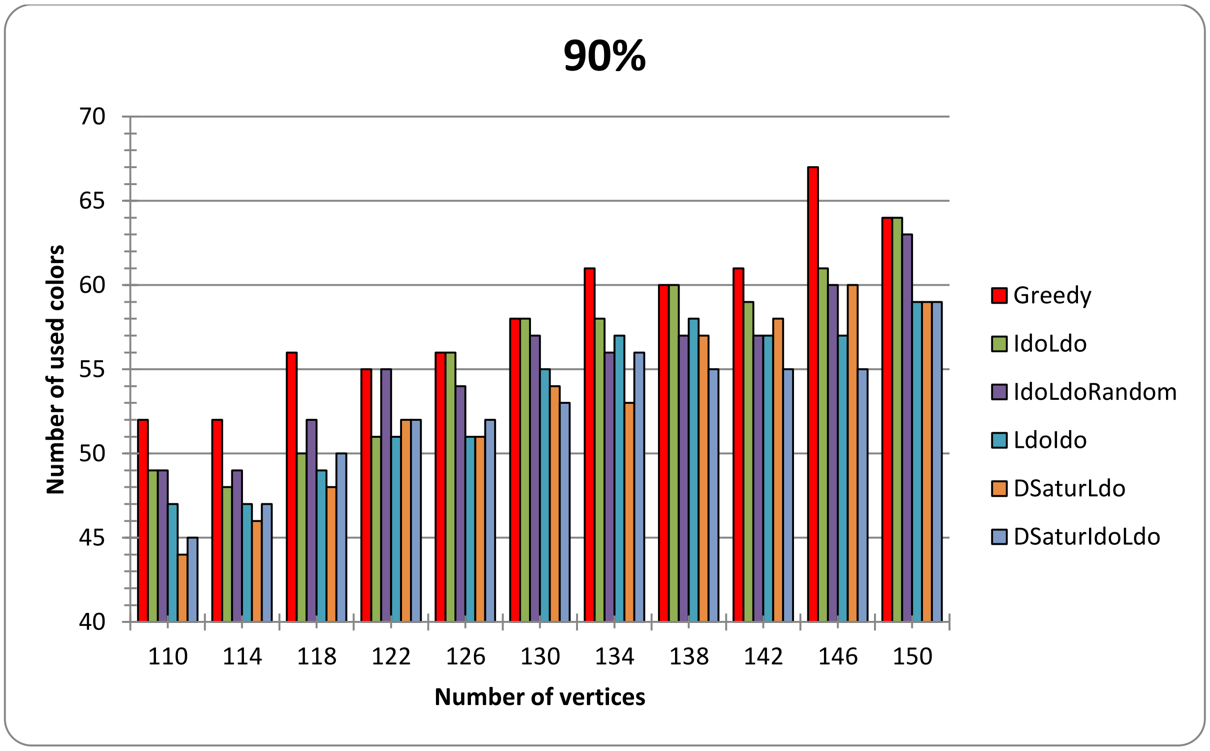

Figure 36. Randomly generated graphs tests' results compared in used colors. Combined algorithms, density 90%.

When it comes to consumed time, then LDO-IDO clearly wins among these three (results of consumed time can be seen in Appendix 1 – Coloring algorithms: Randomly Generated Graphs Test Results: Combined algorithms), although it is bigger than the same of the Greedy one. As mentioned before, this behavior could be explained by the fact that graph’s vertices undergo a heavy sorting in the process of algorithm’s execution unlike the situation with the Greedy algorithm. The tendency towards taking less time as density rises is pointed out for DSatur based algorithms as well. And the fact that DSatur-IDO-LDO takes more time than DSatur-LDO could be explained by extra Incidence Degree Ordering that takes place during algorithm’s execution.

2.5.1.5. Parallel algorithms

It can be seen from the charts that Parallel Largest-First prevails almost in every situation. Along with Parallel Largest-First it is necessary to mention the Parallel Smallest-Last algorithm, however, it shows promising results only at higher densities (80-90%) using almost the same amount of colors and at 90% density even outperforming Parallel Largest-First algorithm. Parallel Jones and Plassmann and its second version perform very similar to the Greedy algorithm, using less or more colors compared to Greedy.

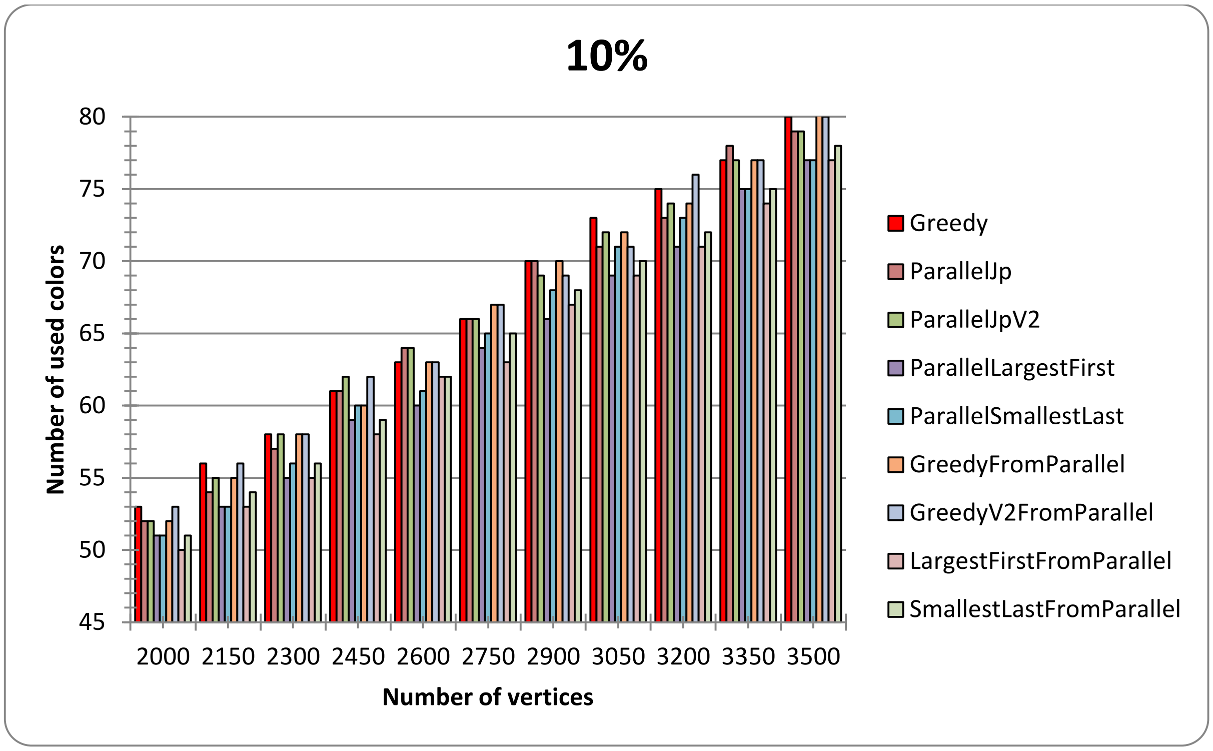

Figure 37. Randomly generated graphs tests' results compared in used colors. Parallel algorithms, density 10%.

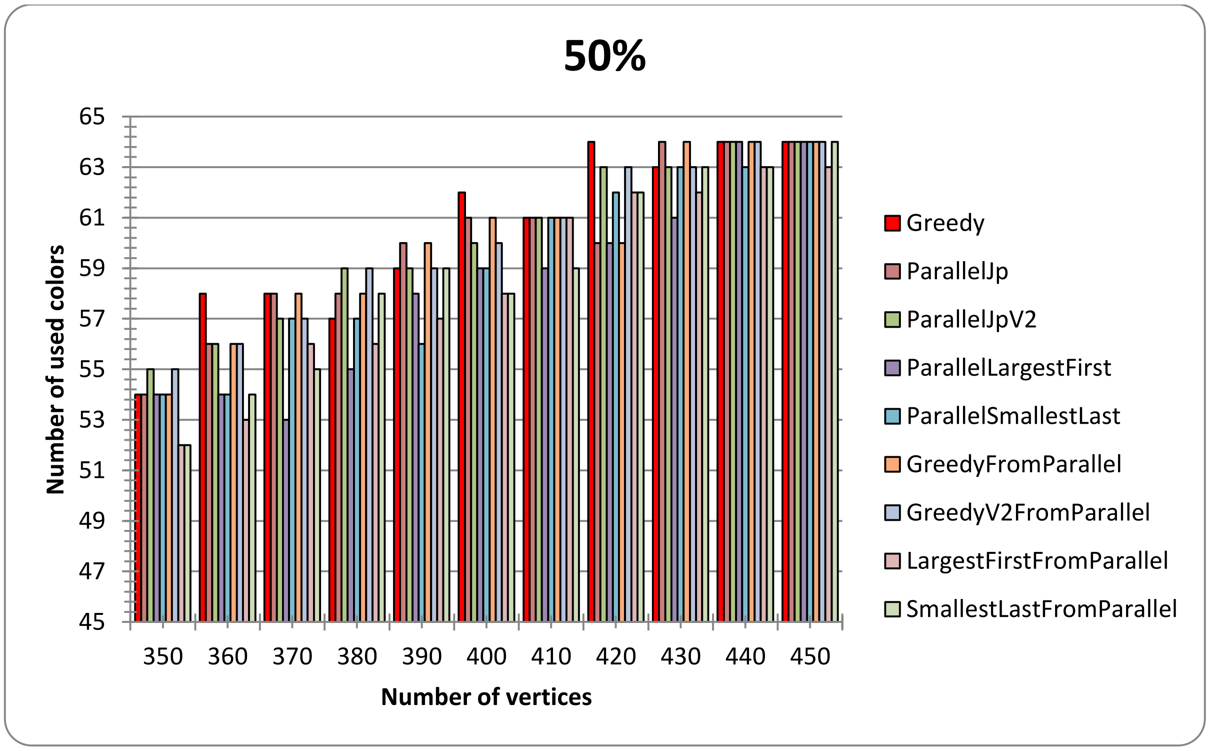

Figure 38. Randomly generated graphs tests' results compared in used colors. Parallel algorithms, density 50%.

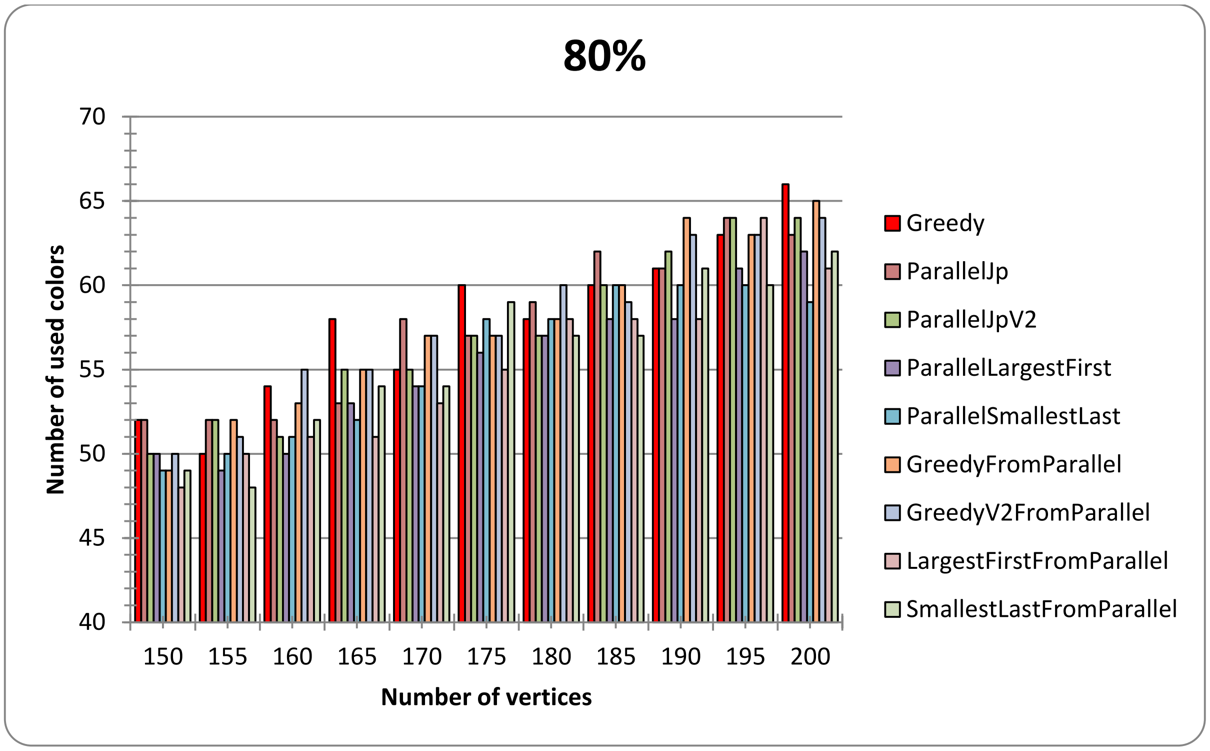

Figure 39. Randomly generated graphs tests' results compared in used colors. Parallel algorithms, density 80%.

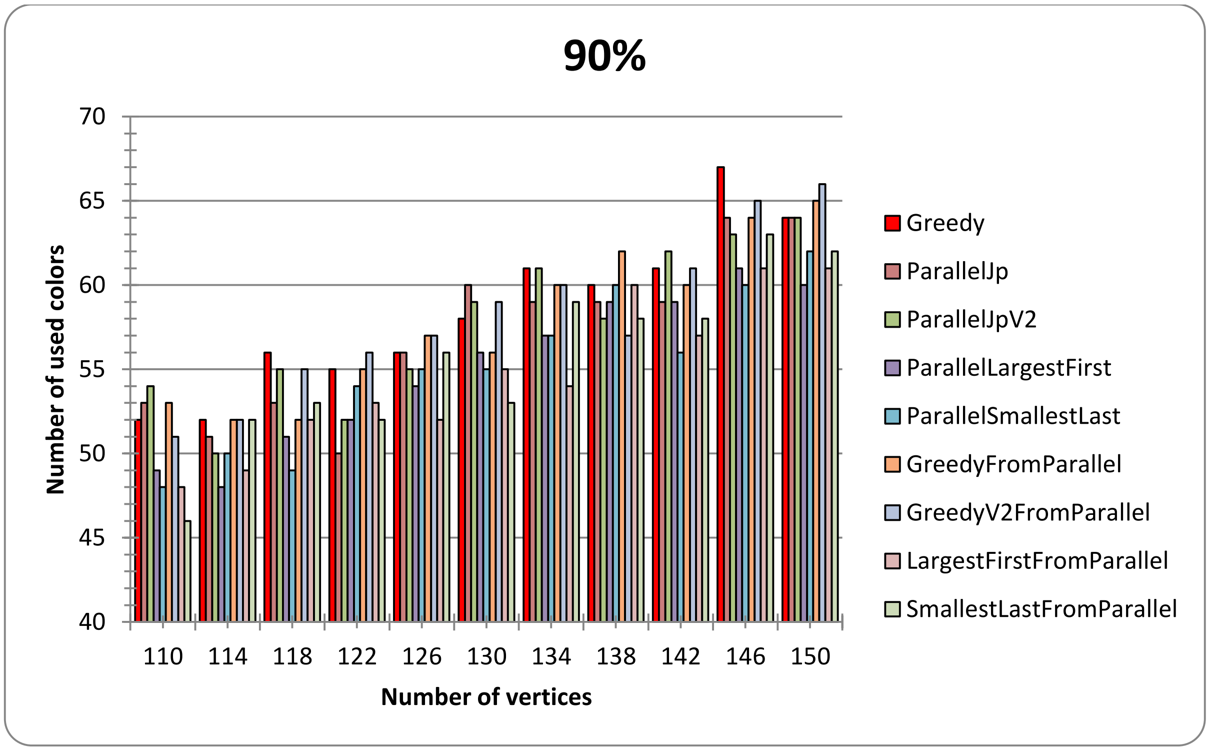

Figure 40. Randomly generated graphs tests' results compared in used colors. Parallel algorithms, density 90%.

In terms of time used to complete the task, Parallel Smallest-Last demonstrates the worst results (results of consumed time can be seen in Appendix 1 – Coloring algorithms: Randomly Generated Graphs Test Results: Parallel algorithms). This behavior is understandable, as Parallel Smallest-Last uses more sophisticated weights system in its process. The performance of Parallel Largest-First is not far away from Parallel Smallest-Last algorithm. The only thing that should be noted is the fact that on 30%, 50% and 80% density Parallel Largest-First algorithm's execution time is very similar to Parallel JP despite the fact that it uses largest first ordering. However, it could be explained by the fact that largest first ordering that is used in Parallel Largest-First algorithm does not consume much more time than the weight system in JP algorithm and might be very effective on those densities.

In theory, parallel algorithms' performance in terms of time should be better, however, in practice our implementations show completely opposite results. It seems that these particular implementations in .NET do not achieve such figures due to the thread management that is happening under the hood of parallel methods. It is possible to draw to such a conclusion after looking at performance of algorithms that are not parallel representations of our algorithms. They take less time than their parallel siblings do.

2.5.1.6. Conclusion

Overall, the majority of algorithms proved to be promising enough to use them in maximum clique algorithms. But we should mention the ones that showed the better results among others in their group in terms of number of used colors. And these algorithms are:

- Among sequential: DSatur, DSatur V2 and Largest-First V3;
- Among combined: DSatur-LDO, DSatur-IDO-LDO and LDO-IDO;
- Among parallel: Parallel Largest-First and Parallel Smallest-Last.

Needless to say that these acquired algorithms have the best chances to be used further in our research, however, the decision will be based on the DIMACS tests' results, as they will show our algorithms' performance with graphs that were specially created to test algorithms for this specific problem.

2.5.2. DIMACS graphs

2.5.2.1. Overview

In this subchapter the same algorithms are analyzed on DIMACS graphs, which are used to test how algorithms are able to solve specified actual problem. As mentioned before, these graphs come from a special package of graphs used in the Second DIMACS Implementation Challenge.

Results are going to be represented in tables and charts. There are going to be three tables and two charts.

The first table shows the number of used colors and consists of five main fields:

- # - id of DIMACS graph;
- Graph name – the name of DIMACS graph;
- Edge density – density of DIMACS graph's edges in percentage;
- Min. color number – number of used colors, provided in DIMACS graph. It is the minimum number of colors that can be used.

The second table demonstrates the number of times that one's algorithm was better than the others or used the minimum number of colors. It uses the results of the previous table (fields painted in gray color).

The third table provides information about time consumption of an algorithm in milliseconds and consists of two fields:

- # - id of DIMACS graph;
- Graph name – the name of DIMACS graph.

According to results, we are going to decrease the number of coloring algorithms that we are going to use further in our research and choose only those, which will show better results.

The charts are going to show the same information as the first and the third tables but graphically. Charts that show us time consumption of coloring algorithms are provided for indicative purposes. On some DIMACS tests it is almost impossible to see time bars because it is almost or equals to zero.

2.5.2.2. DIMACS graphs usage

Before moving on to explaining the received results, it is necessary to show how the DIMACS graphs are being used.

First, the application takes the text files, which contain DIMACS graph's edges in x, y format, as well as graph's size and number of edges. Then, an appropriate 2-dimensional array is created after parsing the size and the number of edges and edges are added to the graph's Values array in a similar way that was used for random graphs.

2.5.2.3. Sequential algorithms

As can be seen from Table 3, every algorithm showed better results than the Greedy algorithm. However, the MinMax algorithm disappointed us with its results, which seem to be only slightly better than those of the Greedy one. But among all the sequential algorithms, there were some, which produced very impressive results and these are: DSatur, DSatur V2 and Largest-First V3. DIMACS tests confirmed the randomly generated graphs tests' results. DSatur V2 was very close to succeed in all DIMACS tests and showed very good performance, although it was the slowest among all in most cases being on par with the first version of DSatur (Table 4 and Table 5). Largest-First V2 demonstrated that it can compete with DSatur algorithm and, although, the number of succeeded attempts is less, it is the absolute leader in terms of consumed time among the three best algorithms.

#	Graph name	Edge density	Vertices	Min. color nr	Greedy	LF	LF2	LF3	DSatur	DSatur2	IDO	MinMax
1	1-FullIns_3.col	23%	30	4	8	4	4	4	4	4	5	4
2	1-Insertions_4.col	11%	67	5	5	5	5	5	5	5	5	5
3	2-FullIns_3.col	15%	52	5	10	5	5	5	5	5	5	6
4	2-Insertions_3.col	11%	37	4	4	4	4	4	4	4	4	4
5	3-Insertions_3.col	7%	56	4	4	4	4	4	4	4	4	4
6	anna.col	10%	138	11	12	11	11	11	11	11	11	12
7	ash331GPIA.col	2%	662	4	10	10	10	6	6	6	8	6
8	david.col	22%	87	11	12	11	11	11	11	11	11	12
9	DSJC125.1.col	10%	125	5	8	7	7	7	6	6	7	8
10	DSJR500.1.col	3%	500	12	15	13	13	13	14	14	14	15
11	fpsol2.i.1.col	10%	496	65	65	65	65	65	65	65	65	65
12	fpsol2.i.2.col	9%	451	30	30	30	30	30	30	30	30	31
13	fpsol2.i.3.col	10%	425	30	30	30	30	30	30	30	30	31
14	games120.col	18%	120	9	9	9	9	9	9	9	9	9
15	homer.col	2%	561	13	15	13	13	13	13	13	13	13
16	huck.col	22%	74	11	11	11	11	11	11	11	11	11
17	inithx.i.1.col	5%	864	54	54	54	54	54	54	54	54	54
18	inithx.i.2.col	7%	645	31	31	31	31	31	31	31	31	31
19	inithx.i.3.col	7%	621	31	31	31	31	31	31	31	31	31
20	jean.col	16%	80	10	10	10	10	10	10	10	10	11
21	le450_25a.col	8%	450	25	28	26	26	25	25	25	26	28
22	le450_25b.col	8%	450	25	27	25	25	25	25	25	25	29
23	miles1000.col	79%	128	42	44	43	43	42	42	42	44	45
24	miles1500.col	100%	128	73	76	73	73	73	73	73	73	74
25	miles250.col	10%	128	8	9	8	8	8	8	8	8	9
26	miles500.col	29%	128	20	22	20	20	20	20	20	20	21
27	miles750.col	52%	128	31	34	32	32	32	31	31	31	33
28	mug88_1.col	4%	88	4	4	4	4	4	4	4	4	4

▨ - best result

Table 1. Results of the Dimacs tests of the sequential algorithms, showing the number of maximum cliques – part 1.

#	Graph name	Edge density	Vertices	Min. color nr	Greedy	LF	LF2	LF3	DSatur	DSatur2	IDO	MinMax
29	mug88_25.col	4%	88	4	4	4	4	4	4	4	4	4
30	mulsol.i.1.col	20%	197	49	49	49	49	49	49	49	49	49
31	mulsol.i.2.col	22%	188	31	31	31	31	31	31	31	31	31
32	mulsol.i.3.col	23%	184	31	31	31	31	31	31	31	31	31
33	mulsol.i.4.col	23%	185	31	31	31	31	31	31	31	31	31
34	mulsol.i.5.col	23%	186	31	31	31	31	31	31	31	31	31
35	myciel3.col	36%	11	4	4	4	4	4	4	4	4	4
36	myciel4.col	28%	23	5	5	5	5	5	5	5	5	5
37	myciel5.col	22%	47	6	6	6	6	6	6	6	6	6
38	queen5_5.col	100%	25	5	8	7	7	7	5	5	6	8
39	queen6_6.col	92%	36	7	11	9	9	8	9	9	9	9
40	queen7_7.col	81%	49	7	10	12	12	11	10	9	13	11
41	queen8_12.col	60%	96	12	15	15	15	14	14	13	16	15
42	queen8_8.col	72%	64	9	13	13	13	12	13	11	12	12
43	queen9_9.col	65%	81	10	16	15	15	13	12	13	14	14
44	r1000.1.col	3%	1000	20	26	23	23	23	20	21	22	25
45	r125.1.col	3%	125	5	5	5	5	5	5	5	5	6
46	r125.1c.col	97%	125	46	51	47	47	47	46	46	49	53
47	r125.5.col	50%	125	36	44	39	39	40	38	37	37	45
48	r250.1.col	3%	250	8	9	8	8	8	8	8	8	9
49	r250.1c.col	97%	250	64	76	68	68	66	65	65	69	70
50	school1.col	26%	385	14	42	32	32	32	20	14	16	36
51	will199GPIA.col	3%	701	7	11	10	10	10	7	7	9	10
52	zeroin.i.1.col	19%	211	49	49	49	49	49	49	49	49	49
53	zeroin.i.2.col	16%	211	30	30	30	30	30	30	30	30	30
54	zeroin.i.3.col	17%	206	30	30	30	30	30	30	30	30	30
55	moonMoser.col	53%	9	3	3	3	3	3	3	3	3	3
56	mycielski.col	18%	11	2	4	4	4	4	4	4	4	4

▨ - best result

Table 2. Results of the Dimacs tests of the sequential algorithms, showing the number of maximum cliques - part 2

Algorithm	# of successes
Greedy	28
Largest-First	39
Largest-First V2	39
Largest-First V3	43
DSatur	49
DSatur V2	52
IDO	39
MinMax	29

Table 3. Sequential algorithms' successes.

#	Graph name	Time (ms)							
		Greedy	LF	LF2	LF3	DSatur	DSatur2	IDO	MinMax
1	1-FullIns_3.col	0	1	9	3	3	2	2	10
2	1-Insertions_4.col	0	0	0	0	0	0	0	0
3	2-FullIns_3.col	0	0	0	0	0	0	0	0
4	2-Insertions_3.col	0	0	0	0	0	0	0	0
5	3-Insertions_3.col	0	0	0	0	0	0	0	0
6	anna.col	0	0	0	0	3	3	3	2
7	ash331GPIA.col	1	9	9	11	90	92	86	45
8	david.col	0	0	0	0	1	1	7	1
9	DSJC125.1.col	0	0	0	0	3	3	2	1
10	DSJR500.1.col	0	5	5	7	52	50	54	27
11	fpsol2.i.1.col	0	5	6	12	60	52	49	53
12	fpsol2.i.2.col	0	3	4	8	46	54	38	36
13	fpsol2.i.3.col	0	3	4	8	42	51	34	31
14	games120.col	0	0	0	0	2	2	2	3
15	homer.col	1	6	6	8	58	58	61	49
16	huck.col	0	0	0	0	1	0	0	0
17	inithx.i.1.col	1	14	16	28	160	154	161	141
18	inithx.i.2.col	1	7	9	15	99	118	82	66
19	inithx.i.3.col	1	7	9	15	96	110	77	62
20	jean.col	0	0	0	0	1	1	1	1
21	le450_25a.col	0	5	7	8	66	44	52	29
22	le450_25b.col	0	5	6	11	71	44	47	40
23	miles1000.col	0	0	1	2	6	8	3	7
24	miles1500.col	0	0	0	4	11	9	4	20
25	miles250.col	0	0	0	2	4	3	3	2
26	miles500.col	0	0	0	1	3	4	4	2
27	miles750.col	0	0	0	2	4	5	3	4
28	mug88_1.col	0	0	0	0	1	1	1	1

Table 4. Results of the Dimacs tests of the sequential algorithms, showing time - part 1.

#	Graph name	Time (ms)							
		Greedy	**LF**	**LF2**	**LF3**	**DSatur**	**DSatur2**	**IDO**	**MinMax**
29	mug88_25.col	0	0	0	0	1	1	1	1
30	mulsol.i.1.col	0	1	2	4	15	11	10	14
31	mulsol.i.2.col	0	1	1	4	13	11	9	10
32	mulsol.i.3.col	0	1	1	4	13	13	6	11
33	mulsol.i.4.col	0	0	1	4	9	8	6	10
34	mulsol.i.5.col	0	1	1	4	13	9	8	10
35	myciel3.col	0	0	0	0	0	0	0	0
36	myciel4.col	0	0	0	0	0	0	0	0
37	myciel5.col	0	0	0	0	0	0	0	0
38	queen5_5.col	0	0	0	0	0	0	0	0
39	queen6_6.col	0	0	0	0	0	0	0	0
40	queen7_7.col	0	0	0	0	1	0	0	0
41	queen8_12.col	0	0	0	1	2	3	1	2
42	queen8_8.col	0	0	0	0	1	1	1	1
43	queen9_9.col	0	0	0	0	2	2	1	2
44	r1000.1.col	0	19	29	40	265	323	248	117
45	r125.1.col	0	0	0	0	2	3	3	2
46	r125.1c.col	0	0	1	4	32	32	5	24
47	r125.5.col	0	0	1	5	8	10	3	11
48	r250.1.col	0	1	2	2	18	25	15	10
49	r250.1c.col	0	2	5	15	231	229	23	101
50	school1.col	0	3	11	14	134	136	38	63
51	will199GPIA.col	2	9	14	18	113	163	104	48
52	zeroin.i.1.col	0	1	1	4	26	9	11	16
53	zeroin.i.2.col	0	1	3	2	15	13	12	22
54	zeroin.i.3.col	0	3	1	3	14	12	10	11
55	moonMoser.col	0	0	0	0	0	0	0	0
56	mycielski.col	0	0	0	0	0	0	0	0

Table 5. Results of the Dimacs tests of the sequential algorithms, showing time - part 2.

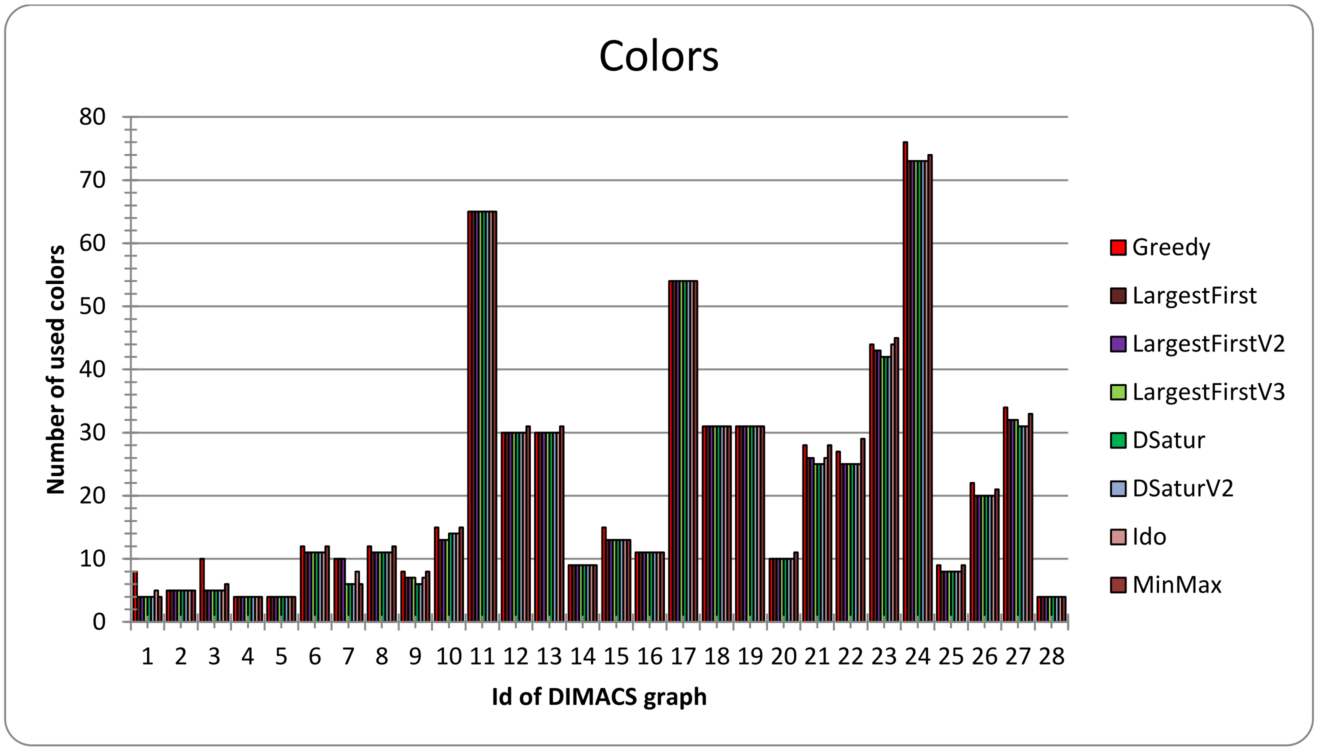

Figure 41. DIMACS graphs tests' results compared in number of used colors. Sequential algorithms - part 1.

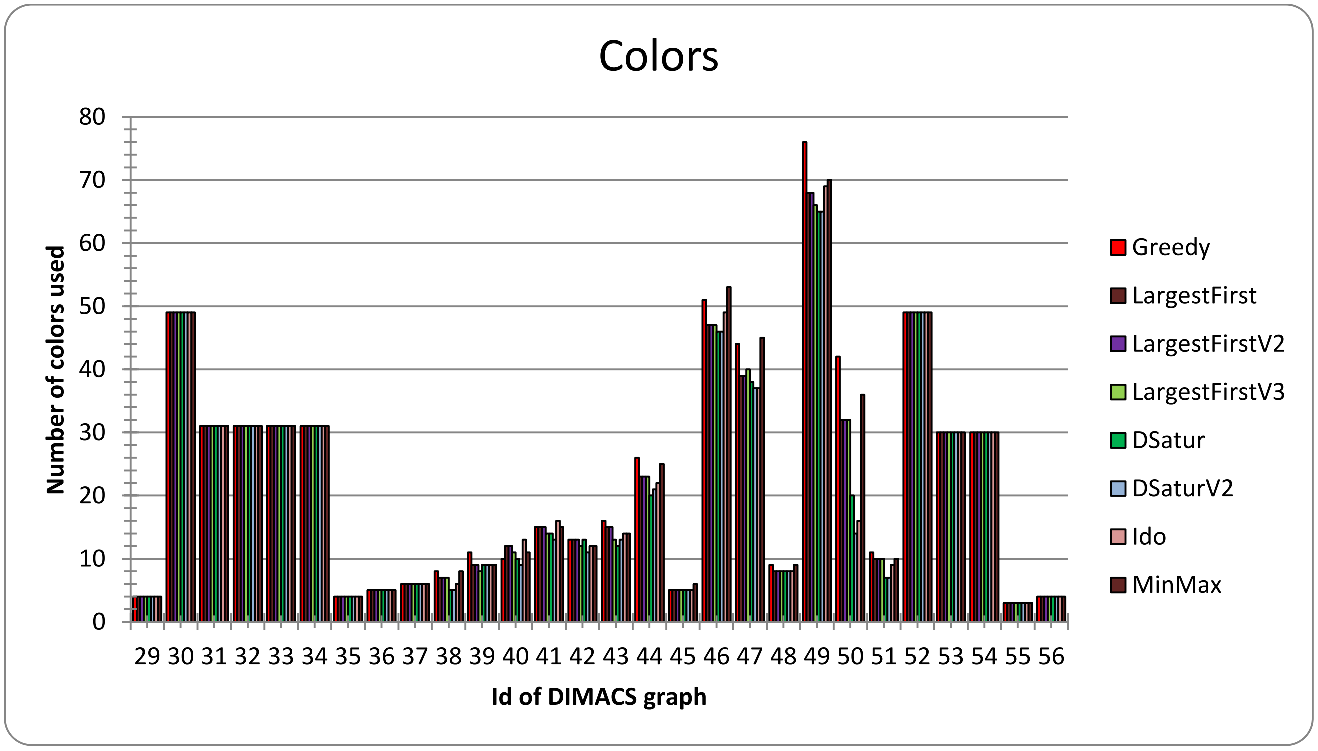

Figure 42. DIMACS graphs tests' results compared in number of used colors. Sequential algorithms - part 2.

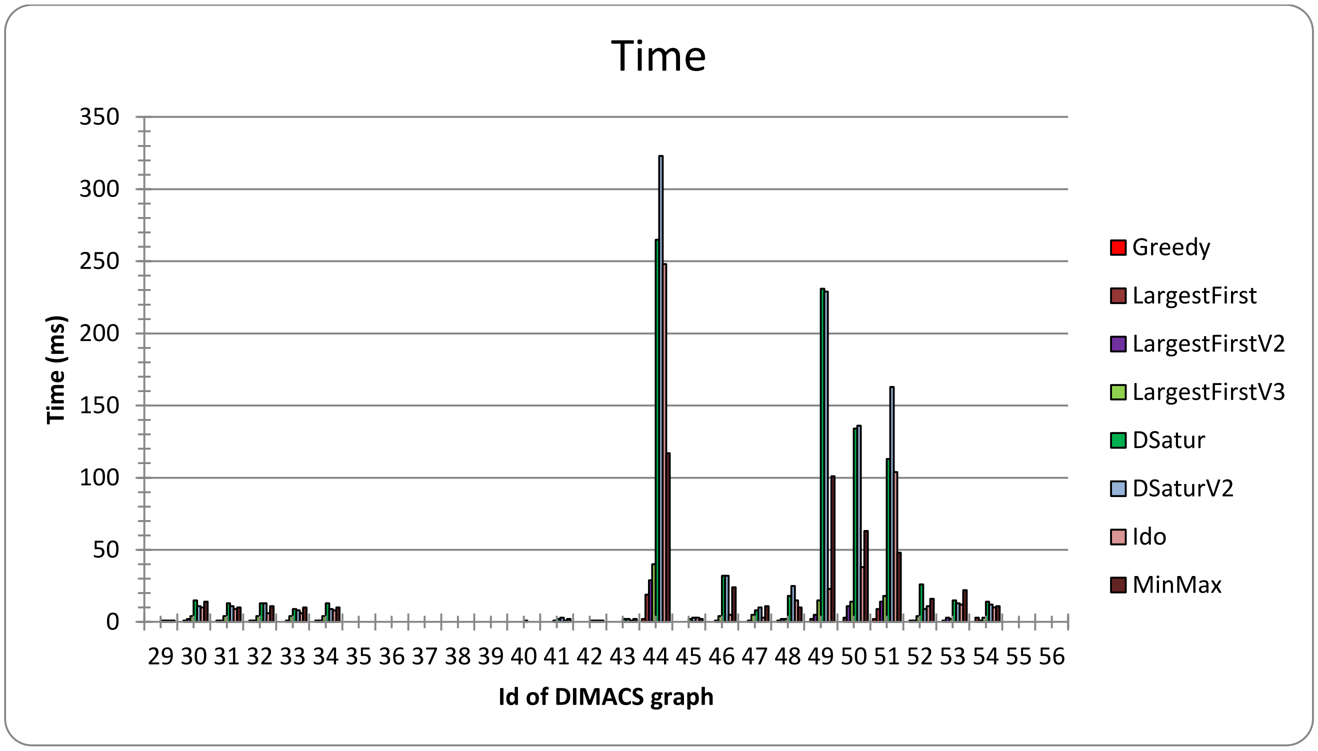

Figure 43. DIMACS graphs tests' results compared in time (ms). Sequential algorithms - part 1.

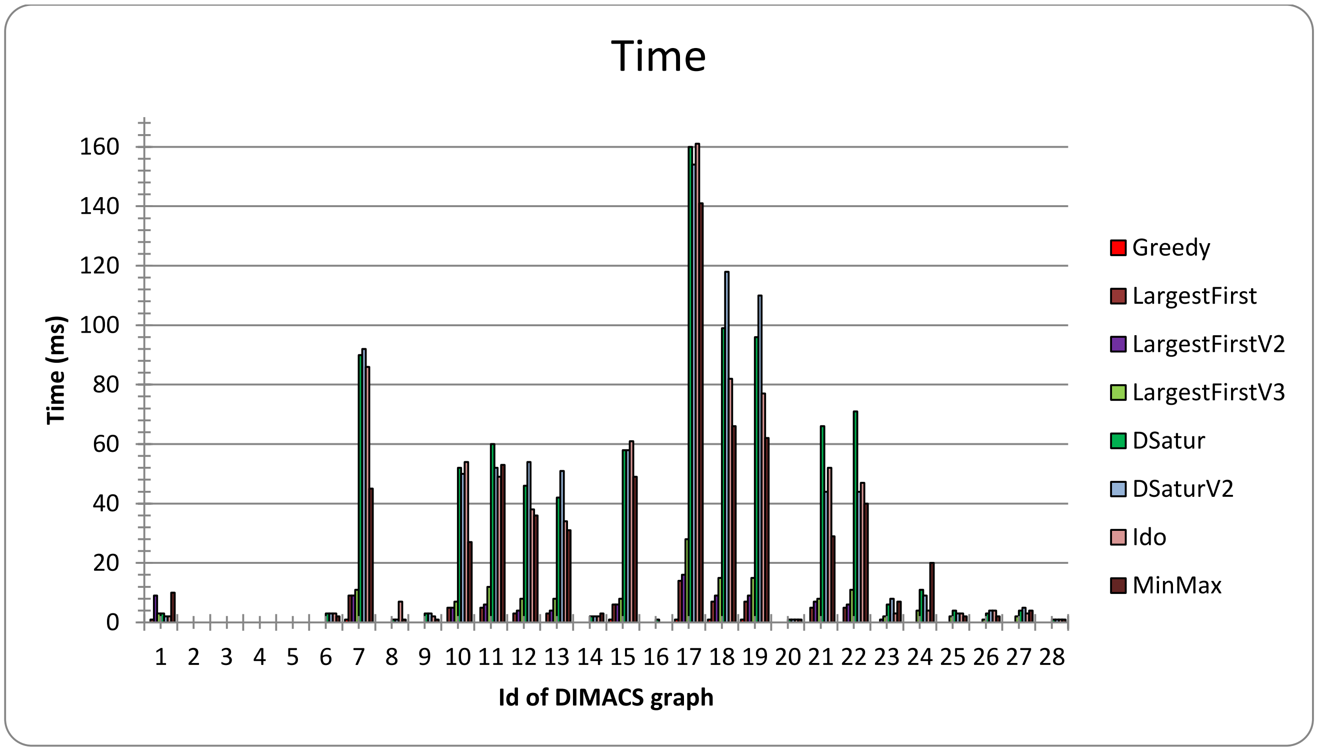

Figure 44. DIMACS graphs tests' results compared in time (ms). Sequential algorithms - part 2.

2.5.2.4. Combined algorithms

As can be seen from Table 8, the story continues and every algorithm showed better results than the Greedy algorithm. And they seem to be very promising. Among all the combined algorithms DSatur based algorithms demonstrated the best performance even out beating the DSatur V2. DIMACS tests confirmed the randomly generated graphs tests' results: DSatur-IDO-LDO, DSatur-LDO and LDO-IDO are the best among combined algorithms. However, if we compare the consumed time, then DSatur-IDO-LDO would be the worst, taking too much time to complete (Table 9 and Table 10). LDO-IDO demonstrated the better time compared to DSatur based algorithms, however, the number of succeeded attempts is considerably smaller.

#	Graph name	Edge density	Vertices	Min. color number	Greedy	LDO-IDO	IDO-LDO	LDO-IDO-Random	DSatur-LDO	DSatur-IDO-LDO
1	1-FullIns_3.col	23%	30	4	8	4	4	4	4	4
2	1-Insertions_4.col	11%	67	5	5	5	5	5	5	5
3	2-FullIns_3.col	15%	52	5	10	5	5	5	5	5
4	2-Insertions_3.col	11%	37	4	4	4	4	4	4	4
5	3-Insertions_3.col	7%	56	4	4	4	4	4	4	4
6	anna.col	10%	138	11	12	11	11	11	11	11
7	ash331GPIA.col	2%	662	4	10	6	8	8	6	6
8	david.col	22%	87	11	12	11	11	11	11	11
9	DSJC125.1.col	10%	125	5	8	7	7	7	6	6
10	DSJR500.1.col	3%	500	12	15	13	13	13	13	12
11	fpsol2.i.1.col	10%	496	65	65	65	65	65	65	65
12	fpsol2.i.2.col	9%	451	30	30	30	30	30	30	30
13	fpsol2.i.3.col	10%	425	30	30	30	30	30	30	30
14	games120.col	18%	120	9	9	9	9	9	9	9
15	homer.col	2%	561	13	15	13	13	13	13	13
16	huck.col	22%	74	11	11	11	11	11	11	11
17	inithx.i.1.col	5%	864	54	54	54	54	54	54	54
18	inithx.i.2.col	7%	645	31	31	31	31	31	31	31
19	inithx.i.3.col	7%	621	31	31	31	31	31	31	31
20	jean.col	16%	80	10	10	10	10	10	10	10
21	le450_25a.col	8%	450	25	28	25	25	25	25	25
22	le450_25b.col	8%	450	25	27	25	25	25	25	25
23	miles1000.col	79%	128	42	44	42	43	43	42	42
24	miles1500.col	100%	128	73	76	73	73	73	73	73
25	miles250.col	10%	128	8	9	8	8	8	8	8
26	miles500.col	29%	128	20	22	20	20	20	20	20
27	miles750.col	52%	128	31	34	32	31	31	31	31
28	mug88_1.col	4%	88	4	4	4	4	4	4	4

▨ - best result

Table 6. Results of the Dimacs tests of the combined algorithms, showing the number of maximum cliques - part 1.

#	Graph name	Edge density	Vertices	Min. color number	Greedy	LDO-IDO	IDO-LDO	LDO-IDO-Random	DSatur-LDO	DSatur-IDO-LDO
29	mug88_25.col	4%	88	4	4	4	4	4	4	4
30	mulsol.i.1.col	20%	197	49	49	49	49	49	49	49
31	mulsol.i.2.col	22%	188	31	31	31	31	31	31	31
32	mulsol.i.3.col	23%	184	31	31	31	31	31	31	31
33	mulsol.i.4.col	23%	185	31	31	31	31	31	31	31
34	mulsol.i.5.col	23%	186	31	31	31	31	31	31	31
35	myciel3.col	36%	11	4	4	4	4	4	4	4
36	myciel4.col	28%	23	5	5	5	5	5	5	5
37	myciel5.col	22%	47	6	6	6	7	6	6	6
38	queen5_5.col	100%	25	5	8	7	7	7	5	5
39	queen6_6.col	92%	36	7	11	8	10	10	9	10
40	queen7_7.col	81%	49	7	10	10	12	14	11	10
41	queen8_12.col	60%	96	12	15	15	15	16	14	14
42	queen8_8.col	72%	64	9	13	12	15	14	12	11
43	queen9_9.col	65%	81	10	16	14	15	16	13	14
44	r1000.1.col	3%	1000	20	26	23	20	20	20	20
45	r125.1.col	3%	125	5	5	5	5	5	5	5
46	r125.1c.col	97%	125	46	51	47	48	47	46	46
47	r125.5.col	50%	125	36	44	38	39	39	38	38
48	r250.1.col	3%	250	8	9	8	8	8	8	8
49	r250.1c.col	97%	250	64	76	66	67	67	65	65
50	school1.col	26%	385	14	42	32	26	26	17	15
51	will199GPIA.col	3%	701	7	11	10	8	8	7	7
52	zeroin.i.1.col	19%	211	49	49	49	49	49	49	49
53	zeroin.i.2.col	16%	211	30	30	30	30	30	30	30
54	zeroin.i.3.col	17%	206	30	30	30	30	30	30	30
55	moonMoser.col	53%	9	3	3	3	3	3	3	3
56	mycielski.col	18%	11	2	4	4	4	4	4	4

▨ - best result

Table 7. Results of the Dimacs tests of the combined algorithms, showing the number of maximum cliques - part 2.

Algorithm	# of successes
Greedy	29
LDO-IDO	44
IDO-LDO	40
IDO-LDO-Random	41
DSatur-LDO	51
DSatur-IDO-LDO	54

Table 8. Combined algorithms' successes.

#	Graph name	Time (ms)					
		Greedy	LDO-IDO	IDO-LDO	LDO-IDO-Random	DSatur-LDO	DSatur-IDO-LDO
1	1-FullIns_3.col	0	8	2	2	2	2
2	1-Insertions_4.col	0	0	0	0	0	1
3	2-FullIns_3.col	0	0	0	0	0	0
4	2-Insertions_3.col	0	0	0	0	0	0
5	3-Insertions_3.col	0	0	0	0	0	0
6	anna.col	0	0	3	6	3	4
7	ash331GPIA.col	1	11	83	108	90	121
8	david.col	0	0	1	1	1	1
9	DSJC125.1.col	0	0	2	3	3	3
10	DSJR500.1.col	0	7	50	62	51	69
11	fpsol2.i.1.col	0	11	45	64	60	73
12	fpsol2.i.2.col	0	8	37	46	44	55
13	fpsol2.i.3.col	0	8	34	40	42	50
14	games120.col	0	0	2	3	2	3
15	homer.col	1	9	58	82	59	77
16	huck.col	0	0	0	1	1	1
17	inithx.i.1.col	1	27	142	198	162	203
18	inithx.i.2.col	1	16	79	100	94	120
19	inithx.i.3.col	1	15	78	96	92	118
20	jean.col	0	0	1	1	1	1
21	le450_25a.col	0	12	53	43	53	78
22	le450_25b.col	0	9	54	61	66	56
23	miles1000.col	0	2	4	5	9	11
24	miles1500.col	0	4	3	7	16	16
25	miles250.col	0	0	5	3	2	5
26	miles500.col	0	1	4	5	5	4
27	miles750.col	0	2	4	5	6	7
28	mug88_1.col	0	0	1	1	1	2

Table 9. Results of the Dimacs tests of the combined algorithms, showing time - part 1.

#	Graph name	Time (ms)					
		Greedy	**LDO-IDO**	**IDO-LDO**	**LDO-IDO-Random**	**DSatur-LDO**	**DSatur-IDO-LDO**
29	mug88_25.col	0	0	1	1	1	2
30	mulsol.i.1.col	0	3	10	11	12	13
31	mulsol.i.2.col	0	4	9	11	14	11
32	mulsol.i.3.col	0	2	9	7	13	11
33	mulsol.i.4.col	0	4	9	12	13	15
34	mulsol.i.5.col	0	4	6	11	14	17
35	myciel3.col	0	0	0	0	0	0
36	myciel4.col	0	0	0	0	0	0
37	myciel5.col	0	0	0	0	0	0
38	queen5_5.col	0	0	0	0	0	0
39	queen6_6.col	0	0	0	0	0	0
40	queen7_7.col	0	0	0	0	1	0
41	queen8_12.col	0	1	2	3	3	4
42	queen8_8.col	0	0	1	1	1	1
43	queen9_9.col	0	0	1	1	1	3
44	r1000.1.col	0	28	239	361	289	304
45	r125.1.col	0	0	3	4	3	3
46	r125.1c.col	0	3	4	7	31	23
47	r125.5.col	0	2	4	5	13	11
48	r250.1.col	0	2	24	22	16	16
49	r250.1c.col	0	17	25	22	199	200
50	school1.col	0	9	41	45	137	117
51	will199GPIA.col	2	17	114	173	122	193
52	zeroin.i.1.col	0	8	7	15	11	19
53	zeroin.i.2.col	0	6	20	9	10	12
54	zeroin.i.3.col	0	2	10	13	14	18
55	moonMoser.col	0	0	0	0	0	0
56	mycielski.col	0	0	0	0	0	0

Table 10. Results of the Dimacs tests of the combined algorithms, showing time - part 2.

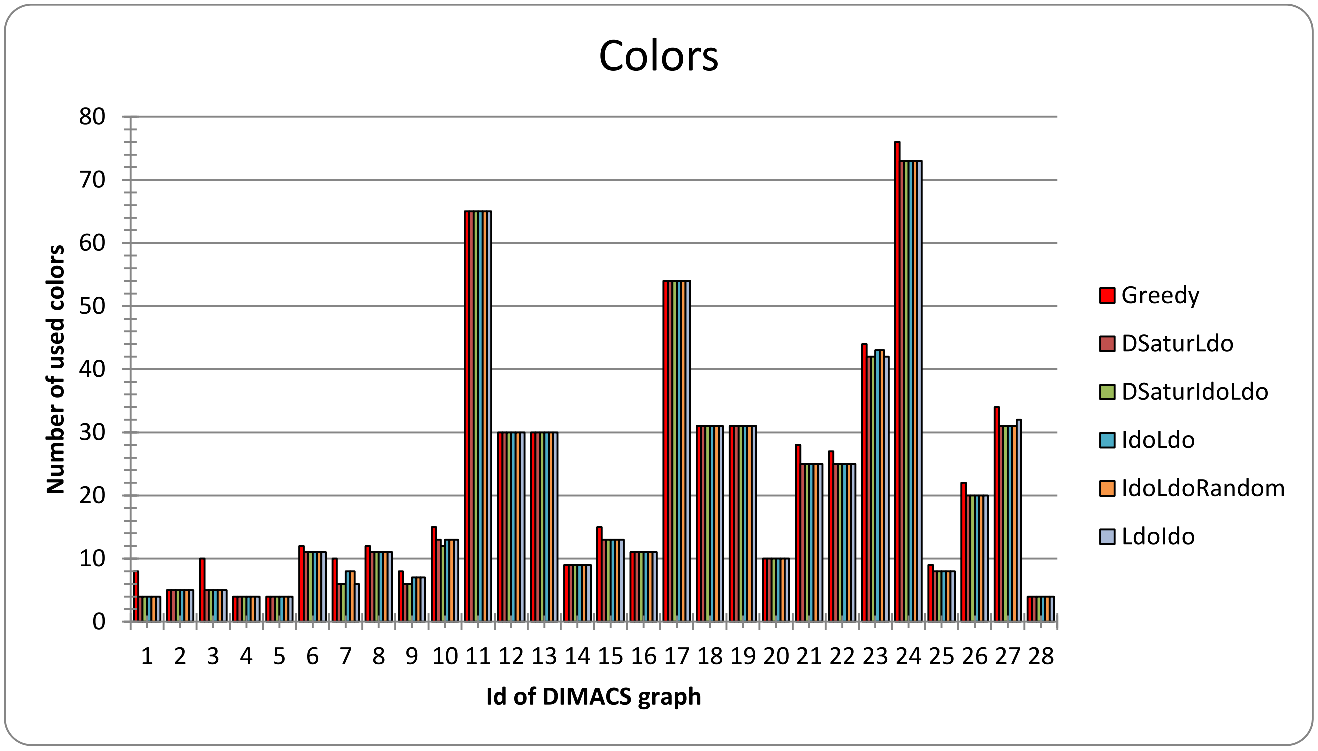

Figure 45. DIMACS graphs tests' results compared in number of used colors. Combined algorithms - part 1.

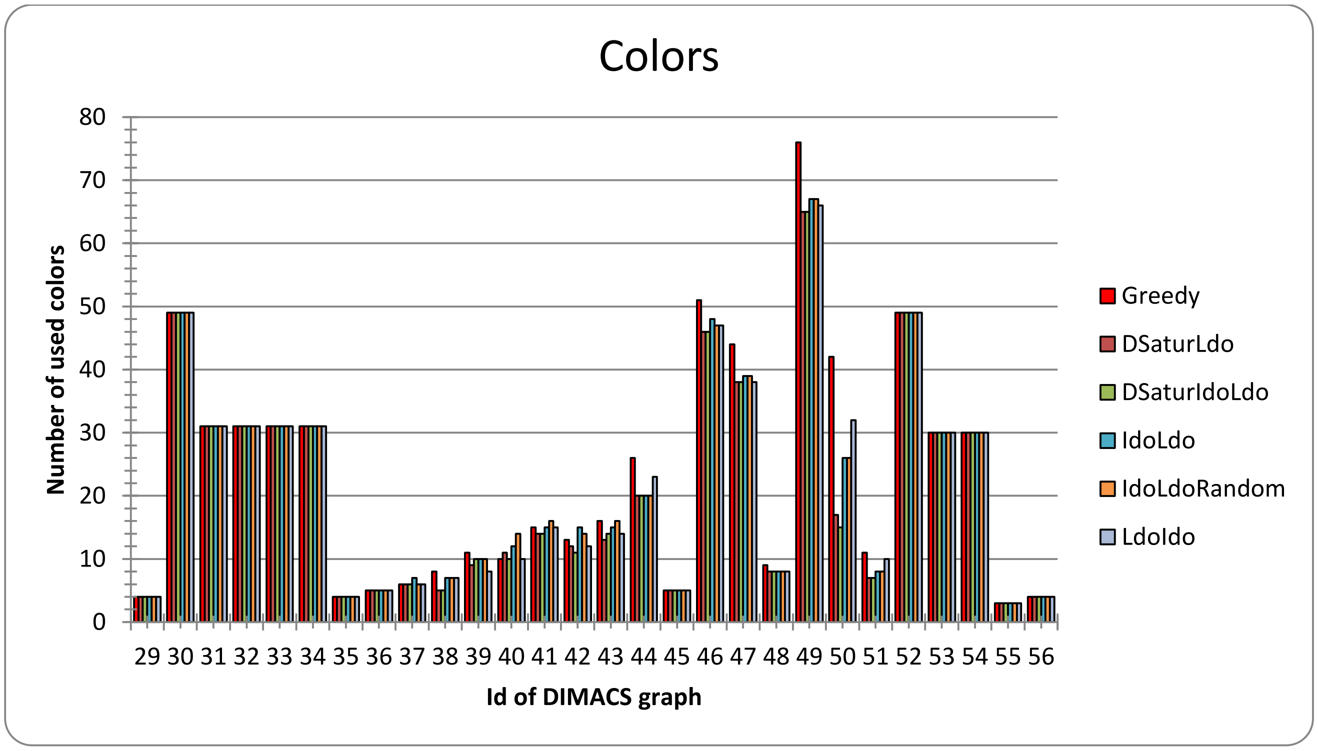

Figure 46. DIMACS graphs tests' results compared in number of used colors. Combined algorithms - part 2.

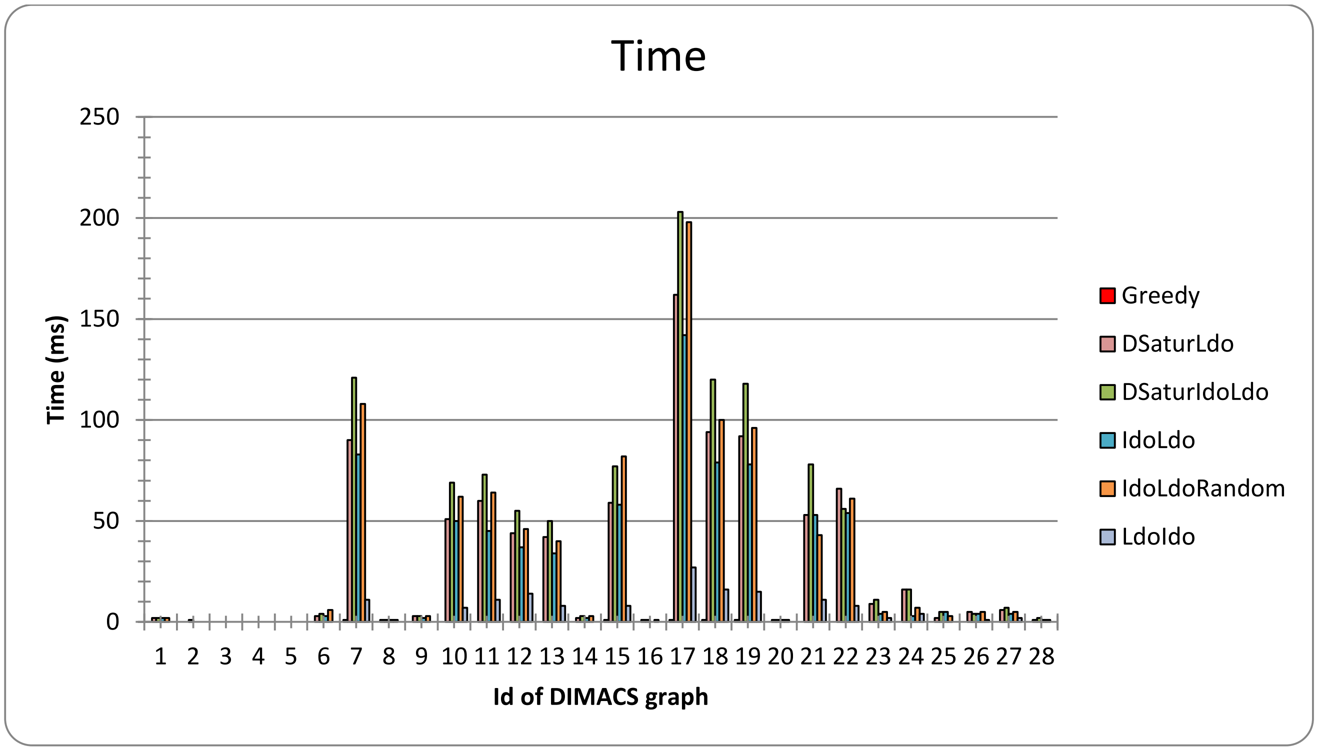

Figure 47. DIMACS graphs tests' results compared in time (ms). Combined algorithms - part 1.

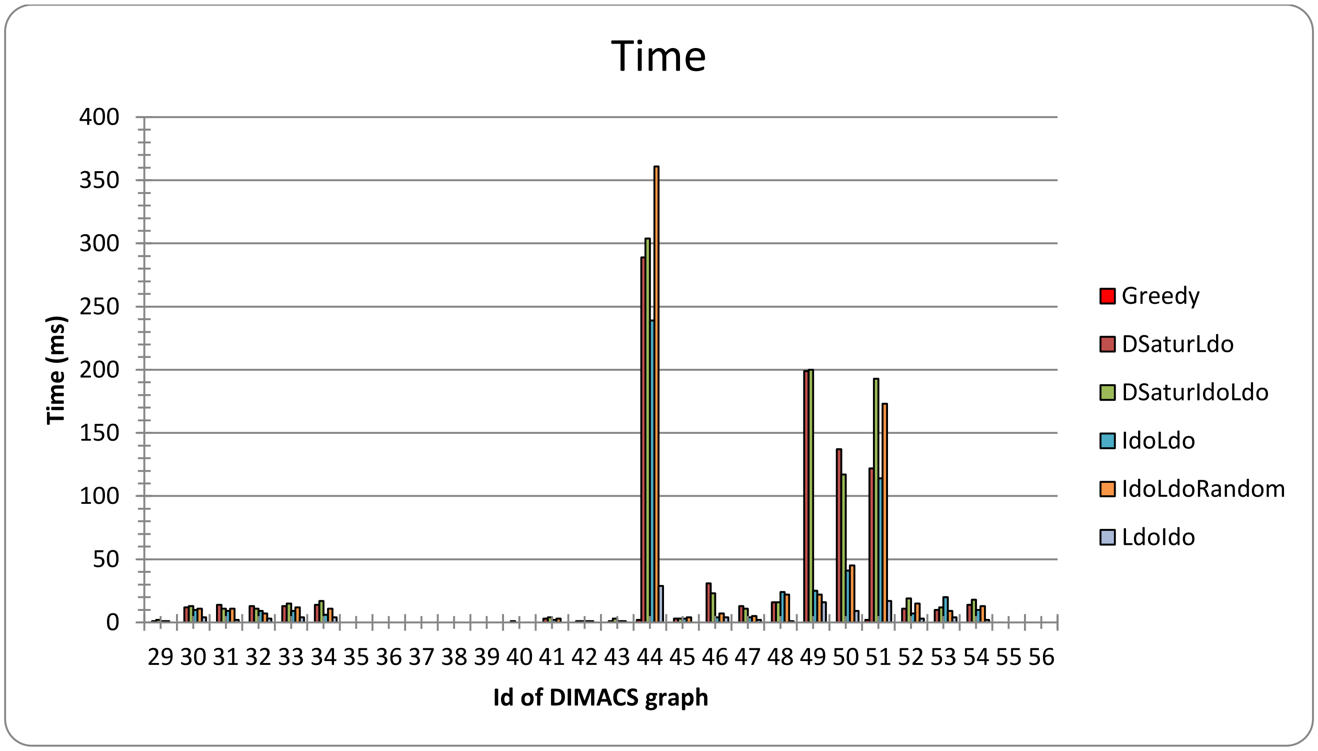

Figure 48. DIMACS graphs tests' results compared in time (ms). Combined algorithms - part 2.

2.5.2.5. Parallel algorithms

As can be seen from Table 13, every algorithm showed better results than the Greedy algorithm, however, this time JP and JP2 demonstrated almost the same results as the Greedy one. Why is that? The answer is simple: JP and JP2 are parallel representatives of Greedy algorithm. On the other hand, Parallel Largest-First and Parallel Smallest-Last proved to be the best among parallel algorithms, showing good results. It is possible to say that DIMACS tests confirmed the randomly generated graphs tests' results. However, if we compare the consumed time, then Parallel Largest-First and Parallel Smallest-Last would be the last among them (Table 14 and Table 15). JP2 seems to take much less time, then other parallel algorithms.

#	Graph name	Edge density	Vertices	Min. color number	Greedy	JP	JP2	PLF	PSL
1	1-FullIns_3.col	23%	30	4	8	4	4	4	4
2	1-Insertions_4.col	11%	67	5	5	6	6	5	5
3	2-FullIns_3.col	15%	52	5	10	5	5	5	5
4	2-Insertions_3.col	11%	37	4	4	4	4	4	4
5	3-Insertions_3.col	7%	56	4	4	4	4	4	4
6	anna.col	10%	138	11	12	11	11	11	11
7	ash331GPIA.col	2%	662	4	10	6	6	8	7
8	david.col	22%	87	11	12	11	12	11	11
9	DSJC125.1.col	10%	125	5	8	8	8	6	7
10	DSJR500.1.col	3%	500	12	15	15	15	13	13
11	fpsol2.i.1.col	10%	496	65	65	65	65	65	65
12	fpsol2.i.2.col	9%	451	30	30	31	31	30	30
13	fpsol2.i.3.col	10%	425	30	30	31	31	30	30
14	games120.col	18%	120	9	9	10	9	9	9
15	homer.col	2%	561	13	15	14	15	13	13
16	huck.col	22%	74	11	11	11	11	11	11
17	inithx.i.1.col	5%	864	54	54	54	54	54	54
18	inithx.i.2.col	7%	645	31	31	31	32	31	31
19	inithx.i.3.col	7%	621	31	31	31	31	31	31
20	jean.col	16%	80	10	10	10	10	10	10
21	le450_25a.col	8%	450	25	28	28	28	25	25
22	le450_25b.col	8%	450	25	27	28	28	25	25
23	miles1000.col	79%	128	42	44	44	44	43	42
24	miles1500.col	100%	128	73	76	74	73	73	73
25	miles250.col	10%	128	8	9	9	10	8	8
26	miles500.col	29%	128	20	22	21	21	20	20
27	miles750.col	52%	128	31	34	33	35	32	31
28	mug88_1.col	4%	88	4	4	4	4	4	4

▨ - best result

Table 11. Results of the Dimacs tests of the parallel algorithms, showing the number of maximum cliques - part 1.

#	Graph name	Edge density	Vertices	Min. color number	Greedy	JP	JP2	PLF	PSL
29	mug88_25.col	4%	88	4	4	4	4	4	4
30	mulsol.i.1.col	20%	197	49	49	49	49	49	49
31	mulsol.i.2.col	22%	188	31	31	31	31	31	31
32	mulsol.i.3.col	23%	184	31	31	31	31	31	31
33	mulsol.i.4.col	23%	185	31	31	31	31	31	31
34	mulsol.i.5.col	23%	186	31	31	31	31	31	31
35	myciel3.col	36%	11	4	4	4	4	4	4
36	myciel4.col	28%	23	5	5	5	5	5	5
37	myciel5.col	22%	47	6	6	7	6	6	6
38	queen5_5.col	100%	25	5	8	7	8	8	6
39	queen6_6.col	92%	36	7	11	9	10	10	11
40	queen7_7.col	81%	49	7	10	11	11	13	11
41	queen8_12.col	60%	96	12	15	14	15	17	17
42	queen8_8.col	72%	64	9	13	12	12	14	13
43	queen9_9.col	65%	81	10	16	14	14	16	15
44	r1000.1.col	3%	1000	20	26	24	24	22	22
45	r125.1.col	3%	125	5	5	6	7	5	5
46	r125.1c.col	97%	125	46	51	53	50	47	49
47	r125.5.col	50%	125	36	44	43	43	39	40
48	r250.1.col	3%	250	8	9	9	9	8	8
49	r250.1c.col	97%	250	64	76	76	73	68	71
50	school1.col	26%	385	14	42	41	41	32	31
51	will199GPIA.col	3%	701	7	11	11	11	10	9
52	zeroin.i.1.col	19%	211	49	49	50	49	49	49
53	zeroin.i.2.col	16%	211	30	30	31	30	30	30
54	zeroin.i.3.col	17%	206	30	30	31	31	30	30
55	moonMoser.col	53%	9	3	3	3	3	3	3
56	mycielski.col	18%	11	2	4	4	4	4	4

▨ - best result

Table 12. Results of the Dimacs tests of the parallel algorithms, showing the number of maximum cliques - part 2.

Algorithm	# of successes
Greedy	27
JP	28
JP2	29
Parallel Largest-First	44
Parallel Smallest-Last	46

Table 13. Parallel algorithms' successes.

#	Graph name	Time (ms)				
		Greedy	JP	JP2	PLF	PSL
1	1-FullIns_3.col	0	32	10	5	14
2	1-Insertions_4.col	0	4	0	4	6
3	2-FullIns_3.col	0	4	3	5	8
4	2-Insertions_3.col	0	2	2	2	5
5	3-Insertions_3.col	0	4	0	3	7
6	anna.col	0	14	1	8	18
7	ash331GPIA.col	1	33	21	45	66
8	david.col	0	6	1	7	14
9	DSJC125.1.col	0	23	2	10	30
10	DSJR500.1.col	0	27	10	34	61
11	fpsol2.i.1.col	0	49	22	45	76
12	fpsol2.i.2.col	0	57	20	49	68
13	fpsol2.i.3.col	0	59	13	32	56
14	games120.col	0	12	1	11	18
15	homer.col	1	30	11	29	74
16	huck.col	0	6	0	5	15
17	inithx.i.1.col	1	85	39	71	120
18	inithx.i.2.col	1	77	29	47	75
19	inithx.i.3.col	1	58	25	44	73
20	jean.col	0	6	1	8	16
21	le450_25a.col	0	64	17	64	91
22	le450_25b.col	0	65	18	69	92
23	miles1000.col	0	22	4	23	42
24	miles1500.col	0	41	6	32	58
25	miles250.col	0	11	1	12	21
26	miles500.col	0	15	3	20	34
27	miles750.col	0	22	4	24	38
28	mug88_1.col	0	5	0	3	7

Table 14. Results of the Dimacs tests of the parallel algorithms, showing time - part 1.

#	Graph name	Time (ms)				
		Greedy	JP	JP2	PLF	PSL
29	mug88_25.col	0	4	0	3	22
30	mulsol.i.1.col	0	34	13	45	58
31	mulsol.i.2.col	0	46	7	23	52
32	mulsol.i.3.col	0	41	10	26	71
33	mulsol.i.4.col	0	37	8	20	58
34	mulsol.i.5.col	0	41	7	20	46
35	myciel3.col	0	2	0	1	5
36	myciel4.col	0	4	0	3	6
37	myciel5.col	0	7	0	5	9
38	queen5_5.col	0	4	0	4	6
39	queen6_6.col	0	5	0	8	8
40	queen7_7.col	0	8	2	8	20
41	queen8_12.col	0	16	2	16	20
42	queen8_8.col	0	11	3	10	13
43	queen9_9.col	0	13	3	14	17
44	r1000.1.col	0	111	46	139	270
45	r125.1.col	0	5	2	19	27
46	r125.1c.col	0	39	25	55	62
47	r125.5.col	0	37	12	46	49
48	r250.1.col	0	19	8	31	40
49	r250.1c.col	0	150	70	133	158
50	school1.col	0	122	32	115	209
51	will199GPIA.col	2	62	32	88	150
52	zeroin.i.1.col	0	39	7	38	60
53	zeroin.i.2.col	0	36	17	42	65
54	zeroin.i.3.col	0	42	6	39	67
55	moonMoser.col	0	1	0	1	3
56	mycielski.col	0	2	0	2	4

Table 15. Results of the Dimacs tests of the parallel algorithms, showing time - part 2.

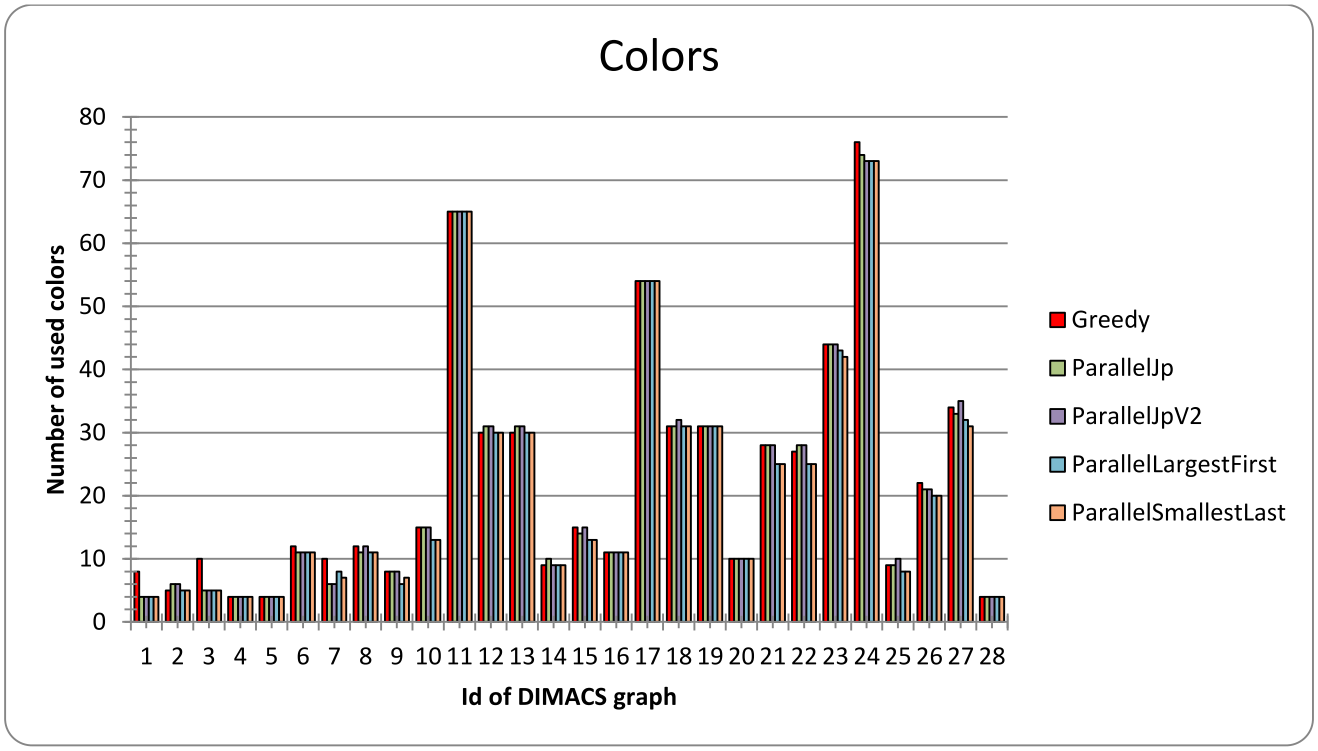

Figure 49. DIMACS graphs tests' results compared in number of used colors. Parallel algorithms - part 1.

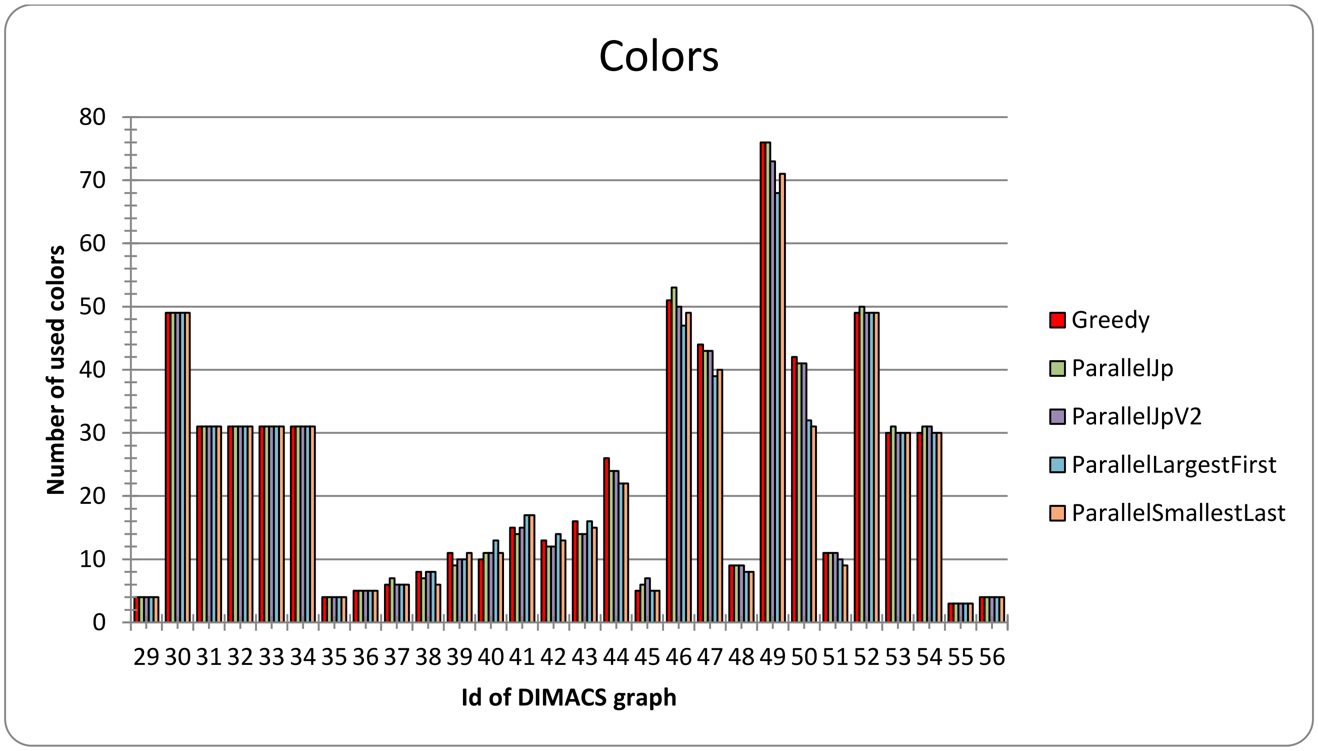

Figure 50. DIMACS graphs tests' results compared in number of used colors. Parallel algorithms - part 2.

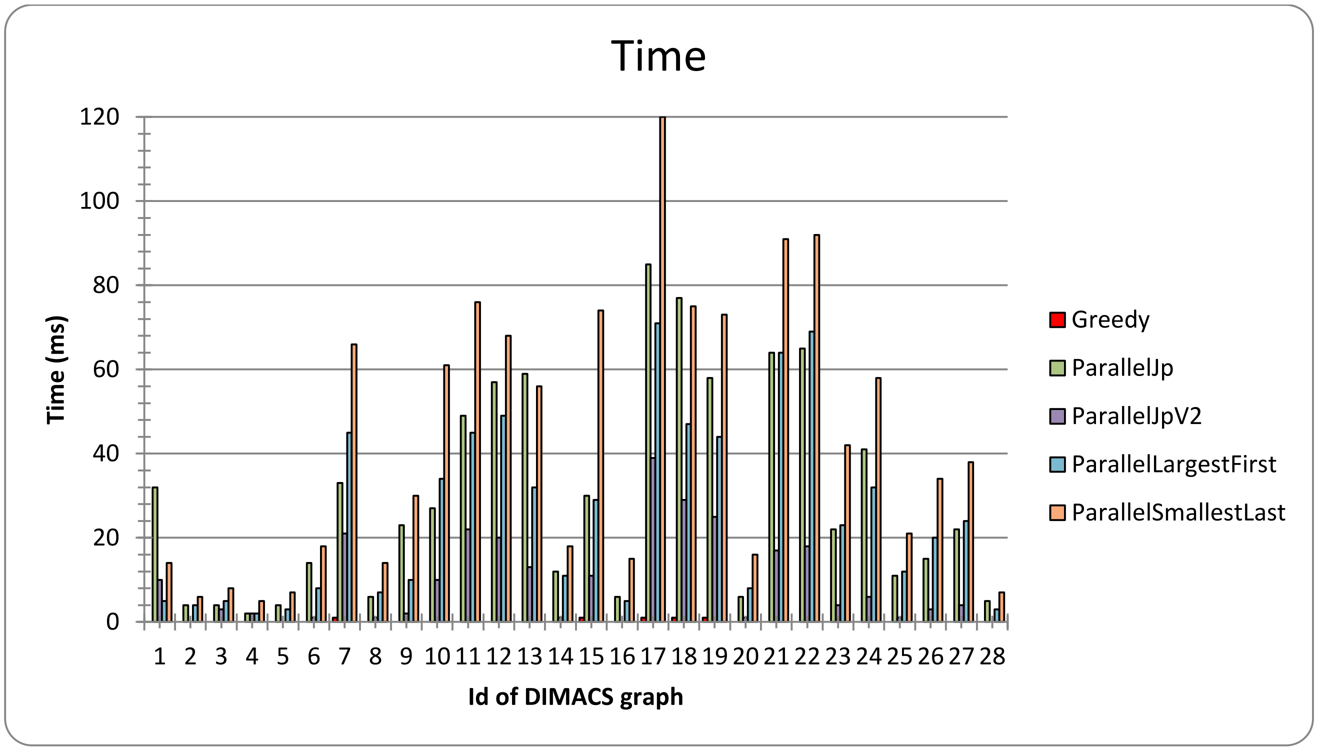

Figure 51. DIMACS graphs tests' results compared in time (ms). Parallel algorithms - part 1.

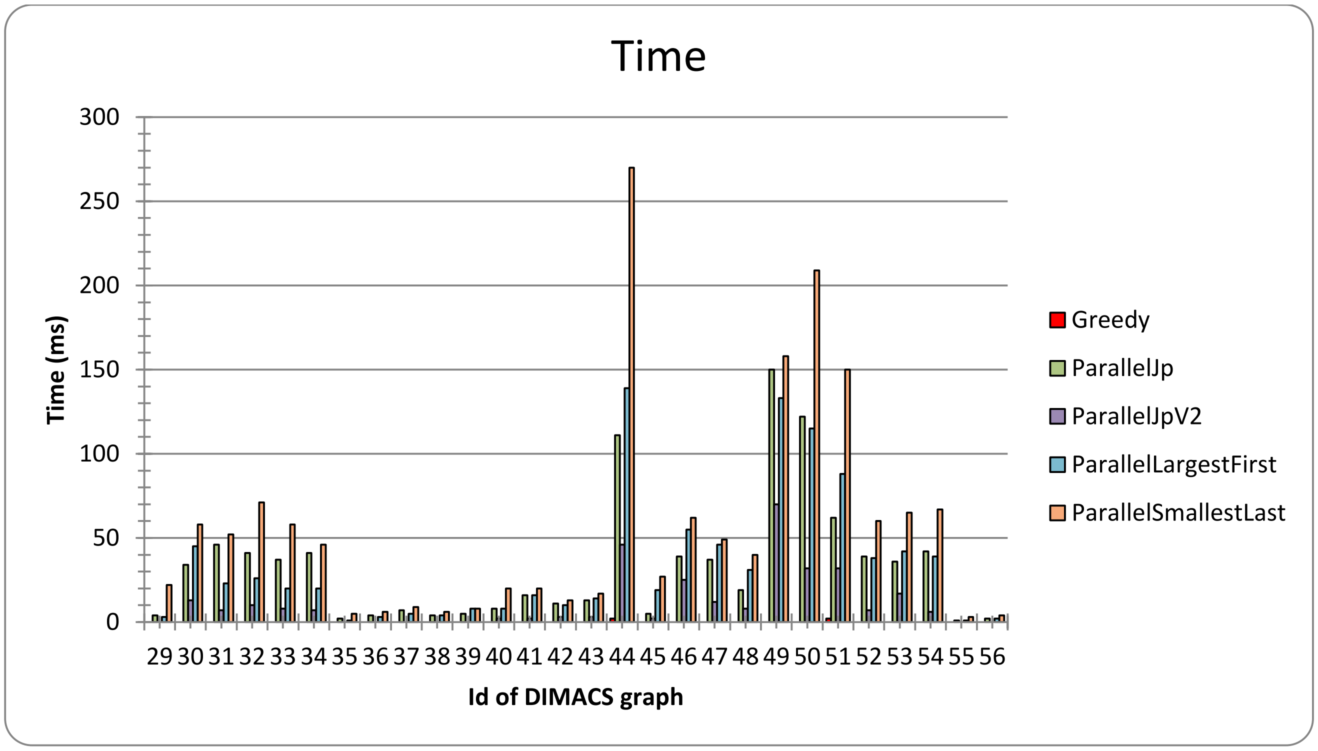

Figure 52. DIMACS graphs tests' results compared in time (ms). Parallel algorithms - part 2.

2.5.2.6. Conclusion

Overall, almost all of the algorithms showed the same results as in the randomly generated graphs tests. DIMACS tests proved that we can use those algorithms further in our research without any doubt.

2.5.3. Overall conclusion

Randomly generated graphs and DIMACS tests showed us the performance of all of the coloring algorithms that we had. According to the results, further in our research we are going to use the following coloring algorithms:

- sequential: DSatur, DSatur V2 and Largest-First V3;
- combined: DSatur-LDO, DSatur-IDO-LDO and LDO-IDO;
- parallel: Parallel Largest-First and Parallel Smallest-Last.

3. Maximum Clique Algorithms

3.1. Overview

The main problem of MCP problem is to find the maximal possible clique of a graph. There are many effective algorithms made specially to solve this problem. We have already mentioned some of them in the previous chapters. Modern algorithms heavily depend on heuristics and, in particular, on vertex coloring. Unfortunately, as coloring problem is classified as NP-complete, it is not possible to use exact algorithms to find color classes. In general, the majority of modern algorithms carry out preliminary work with the help of coloring algorithms. They help to gather and analyze additional information before the main algorithm steps into action.

It was chosen to research relatively new algorithms for our work: VColor-u and VRecolor-BT-u, which both were developed in Tallinn University of Technology. This decision was made mainly because of their novelty, great performance and convenient construction: these algorithms were not researched thoroughly as they are relatively new and they allow us to interchange coloring algorithms without heavily changing their structure.

In this chapter we are going to introduce VColor-u and VRecolor-BT-u algorithms, describe them and conduct tests with different coloring algorithms on random and DIMACS graphs. Our main goal is to research if different coloring algorithms could improve overall maximum clique algorithms' performance.

3.2. VColor-u

VColor-u algorithm was first introduced in 2005 in "Some Practical Algorithms to Solve The Maximum Clique Problem" thesis [3] by Deniss Kumlander. The algorithm's abbreviation can be deciphered as "Vertex Color unweighted". The main idea of this algorithm was to demonstrate efficiency of using independent sets without any additional speeding techniques to solve maximum clique problem.

We are not going to explain the algorithm itself as it is not in the scope of our project (detailed description could be found in Deniss Kumlander's work). However, it is necessary to describe the role of coloring algorithms in VColor-u algorithm. The heuristic that is used in in this case is of Greedy manner – Largest-First. It is used only once in the process of execution, in the beginning, to determine all the color classes one by one until all vertices are colored. And only after the color classes have been received, the main part of the algorithm steps into action.

Therefore, it is necessary to exclude Largest-First algorithm and use other algorithms, which were determined in *Chapter 2*, in its place.

3.3. VRecolor-BT-u

VRecolor-BT-u algorithm was first introduced in 2015 in "Reversed Search Maximum Clique Algorithm Based On Recoloring" thesis [3] by Aleksandr Porošin. The algorithm's abbreviation can be deciphered as "Vertex Recolor Backtracking unweighted". The main idea is the same as in VColor-u algorithm: effective use of independent sets to solve maximum clique problem. But this time it tries to combine reversed search by color classes and in-depth coloring.

Yet again we are not going to explain the algorithm itself as it is not in the scope of our project (detailed description could be found in Aleksandr Porošin's work). What is necessary to do is to explain how vertex coloring happens under the hood. This time it is very complex and, besides initial coloring, implements in-depth coloring i.e. recoloring on each depth. When depth is high, vertex recoloring helps to gain the most accurate data about independent sets on current level. Furthermore, there are two coloring algorithms used in VRecolor-BT-u: one with swaps and one without them. In order to choose, which one should be used in the process of execution, a special constant was introduced. This constant refers to graph's density and equals to 0.35 when referring to initial coloring and

0.55 in case of in-depth coloring. The following diagram briefly describes how the choices of coloring are made (Figure 53).

	density < 0.35	**0.35 ≤ density < 0.55**	**0.55 ≤ density**	**density > 0.55**
initial coloring	🟦	🟧	🟧	🟧
in-depth coloring	🟦	🟦	🟦	🟧

🟦	coloring with swaps
🟧	coloring without swaps

Figure 53. Coloring choice based on density [16].

As can be seen from the diagram, coloring without swaps is used in initial coloring when density is more or equals to 0.35 and in in-depth coloring – when density is more than 0.55. The heuristic being used is of Greedy manner. However, the initial coloring uses largest-first ordering, so the algorithm is Largest-First.

In our work, we consider researching initial coloring without swaps because of the fact that it is used only once before the main algorithm starts to work. Applying time-consuming coloring algorithms to the in-depth coloring will result in significant increase of algorithm's time. Therefore, these considerations led us to limit ourselves with testing only initial coloring of the algorithm as it applies only once in the beginning. The less color classes the initial coloring finds, the less iterations VRecolor-BT-u algorithm will perform in its main routine.

3.4. Tests and results

3.4.1. Randomly generated graphs

3.4.1.1. Overview

First of all, we are going to research the randomly generated graphs. It will give us the picture of algorithms' performance and show us dependencies on one or another coloring algorithm.

For every coloring algorithm that we determined in *Chapter 2* of our research, we created a copy of our selected maximum clique algorithms (VColor-u and VRecolor-BT-u), adjusting the code in a way that we could apply our chosen coloring algorithms.

For algorithms, based on VColor-u, a new random graph was generated for each density starting from 10% and ending with 90%. Concerning VRecolor-BT-u, the starting density was 40%, as it was decided to place our coloring algorithms only into "initial coloring" phase, where the constant, referring to graph's density, is 0.35. For every density ten tests were conducted and average time calculated in milliseconds.

The information is represented in a table. Note that the density parameter is shown at first since the number of vertices, which goes second, heavily depends on the first one. The number of vertices is chosen so, that the time spent on finding the maximum clique for a corresponding density does not exceed the limit of one hour. The table shows algorithm's time consumption. Every column demonstrates the ratio of original maximum clique algorithm's taken time in milliseconds divided by time in milliseconds of modified maximum clique algorithm. Column names tell us the coloring algorithm used in corresponding maximum clique algorithm.

Random graphs' generation function is the same as in *subchapter 2.5.1.2*. System characteristics remain the same as in *subchapter 2.5*.

3.4.1.2. Results of VColor-u Based Algorithms

Table 16 and Table 17 demonstrate VColor-u based algorithms' time consumption ratio on finding maximal clique. Value greater than 1.00 means that VColor-u with corresponding coloring algorithm is faster than original VColor-u (with Largest-First coloring algorithm). If value equals to 1.00, then VColor-u with corresponding coloring algorithm works the same as original VColor-u algorithm in terms of time. And, finally, if value is smaller than 1.00, then original VColor-u algorithm's time consumption is smaller compared to modified VColor-u.

The first interesting thing that must be noted is the fact that when density is 10% using no orderings makes sense as it makes the algorithm faster. VColor-u with Greedy coloring algorithm prevails on density 10%. However, only on that density. The higher the density, the worse the performance of Greedy algorithm becomes.

When using Largest-First V3 coloring algorithm, time consumption goes down from density 10% to 30% but then becomes higher and never outperforms the original Largest-First algorithm. It is possible to say the same about LDO-IDO coloring algorithm.

It should be noted that all algorithms, which used DSatur based coloring algorithms, performed the same way, reaching its peak on density 50%, where time consumption is comparable to the original algorithm with Largest-First, and then starting to increase the overall time of the maximum clique algorithm.

Edge density	Vertices	Greedy	Largest-First V3	DSatur	DSatur V2	DSatur-LDO
0.1	2000	1.05	0.85	0.21	0.20	0.21
0.2	1300	0.95	0.93	0.52	0.52	0.53
0.3	860	0.91	0.93	0.80	0.80	0.79
0.4	600	0.82	0.91	0.88	0.88	0.87
0.5	400	0.81	0.90	0.99	0.98	0.94
0.6	300	0.70	0.89	0.96	0.90	0.91
0.7	200	0.48	0.72	0.87	0.81	0.81
0.8	150	0.31	0.57	0.79	0.69	0.75
0.9	100	0.05	0.34	0.68	0.58	0.65

Table 16. Random graphs test results. VColor-u, time consumption ratio - part 1.

Edge density	Vertices	DSatur-IDO-LDO	LDO-IDO	Parallel Largest-First	Parallel Smallest-Last
0.1	2000	0.20	0.90	0.66	0.53
0.2	1300	0.52	0.93	0.86	0.80
0.3	860	0.80	0.95	0.93	0.86
0.4	600	0.88	0.94	0.94	0.87
0.5	400	0.95	0.93	1.02	0.90
0.6	300	0.91	0.92	1.06	0.82
0.7	200	0.84	0.72	1.07	0.64
0.8	150	0.72	0.60	1.09	0.42
0.9	100	0.64	0.52	1.18	0.13

Table 17. Random graphs test results. VColor-u, time consumption ratio - part 2.

When looking at Table 17 it can be easily seen that VColor-u algorithm, which uses Parallel Largest-First coloring algorithm under the hood, performs better than the original VColor-u algorithm starting from density 50% despite the fact that its implementation in C# is quite poor as was proved in Chapter 2. We could suppose that algorithm with Parallel Smallest-Last must behave the same way, however, the results show us completely opposite figures: starting from density 50% the efficiency of VColor-u with Parallel Smallest-Last as coloring algorithm decreases drastically. This behavior should be investigated in future researches.

3.4.1.3. Results of VRecolor-BT-u Based Algorithms

Table 18 and Table 19 demonstrate VRecolor-BT-u based algorithms' time consumption ratio on finding maximal clique. Value greater than 1.00 means that VRecolor-BT-u with corresponding coloring algorithm is faster than original VRecolor-BT-u (with Largest-First coloring algorithm). If value equals to 1.00, then VRecolor-BT-u with corresponding coloring algorithm works the same as original VRecolor-BT-u algorithm in terms of time. And, finally, if value is smaller than 1.00, then original

VRecolor-BT-u algorithm’s time consumption is smaller compared to modified VRecolor-BT-u.

The first interesting thing that must be noted is the fact that when density is 40%, then using Greedy coloring algorithm is very efficient as it makes the algorithm to spend less time for orderings. However, VRecolor-BT-u with Greedy coloring algorithm performs great only on 40% density. The higher the density, the worse the performance of Greedy algorithm becomes.

It should be noted that all algorithms, which used DSatur based coloring algorithms, performed almost the same way, reaching its peak on density 60% and then starting to increase the overall time of the maximum clique algorithm. However, this time the overall performance is not comparable to the original algorithm with Largest-First ordering and time consumption is considerably higher.

We have seen that the behavior of VRecolor-BT-u algorithm with Greedy and DSatur colorings reminds us of the VColor-u with the corresponding coloring algorithms in use. We can assume that it goes the same for other coloring algorithms but this does not represent the real situation. We can clearly see that VRecolor-BT-u algorithm with Largest-First V3 coloring algorithm spends less time to complete on densities starting from 40% and ending with 70%. This is most likely achieved by reordering of vertices which occurs in every iteration of Largest-First V3 coloring algorithm. The LDO-IDO coloring algorithm’s results also decrease the overall time consumption of VRecolor-BT-u algorithm on densities 40%-60% as can be seen from Table 19.

Edge density	Vertices	Greedy	Largest-First V3	DSatur	DSatur V2	DSatur-LDO
0.4	600	1.05	1.00	0.68	0.65	0.65
0.5	400	0.82	1.03	0.71	0.67	0.67
0.6	300	0.82	1.04	0.74	0.71	0.71
0.7	200	0.73	1.06	0.68	0.63	0.65
0.8	150	0.44	0.94	0.62	0.56	0.59
0.9	100	0.10	0.59	0.36	0.36	0.48

Table 18. Random graphs test results. VRecolor-BT-u, time consumption ratio - part 1.

Edge density	Vertices	DSatur-IDO-LDO	LDO-IDO	Parallel Largest-First	Parallel Smallest-Last
0.4	600	0.71	1.15	0.88	0.87
0.5	400	0.74	1.07	0.98	0.90
0.6	300	0.76	1.04	1.01	0.91
0.7	200	0.72	0.92	1.13	0.88
0.8	150	0.68	0.88	0.88	0.80
0.9	100	0.66	0.77	0.80	0.58

Table 19. Random graphs test results. VRecolor-BT-u, time consumption ratio - part 2.

When looking at parallel coloring algorithms, then it is necessary to point out that Parallel Largest-First coloring algorithm’s usage consumes less time on 60%-70% densities while performance of Parallel Smallest-Last does not come up to expectations.

3.4.2. DIMACS graphs

3.4.2.1. Overview

In this subchapter the same algorithms are analyzed on DIMACS graphs, which are used to test how algorithms are able to solve specified actual problem. As mentioned before, these graphs come from a special package of graphs used in the Second DIMACS Implementation Challenge.

Results are going to be represented in three tables. The first one will provide information about time consumption of an algorithm in milliseconds and besides time has three more fields:

- Graph – the name of DIMACS graph;
- Size – number of edges;
- Density – the density of the algorithm.

The second tables introduces a new important characteristic, which is the number of analyzed branches. This parameter shows how many branches each algorithm analyzes. The less the number of analyzed branches is, the faster the algorithm is. This table looks absolutely the same as the previous one. And, finally, the third table shows distribution of color classes used by coloring algorithms. The representation is the same as in the first table. As number of classes stays the same for VColor-u and VRecolor-BT-u algorithms, we decided to move this table to Appendix 2 – Maximum Clique algorithms: DIMACS Graphs Test Results.

The DIMACS graphs are chosen so, that the time spent on finding the maximal clique does not exceed one hour. When it comes to testing algorithms, based on VColor-u, we use all DIMACS graph instances. However, only those among them, which have density more than 35% are going to be selected for algorithms, based on VRecolor-BT-u because it was decided to place our coloring algorithms only into "initial coloring" phase, where the constant, referring to graph's density, is 0.35.

3.4.2.2. Results of VColor-u Based Algorithms

In this subchapter the same coloring algorithms are used inside VColor-u algorithm and tested on DIMACS graph instances because each of them showed the best results separately.

As can be seen from Table 20 and Table 21, VColor-u algorithm, which uses Greedy algorithm under the hood, demonstrates very impressive results on lower densities. However, it is possible to notice that in some cases it shows better results even on higher densities, for example, on 64%, 76% and 90+% densities. At first, this behavior might seem strange but it can be simply explained – we just have to look at the number of found color classes in Table 28 as well as number of analyzed branches in Table 22. The number of used colors along with number of analyzed branches are the same, so the difference in time could be explained by the fact that Greedy algorithm does not make any orderings, so it works faster in this case.

When comparing original VColor-u algorithm with its siblings that include Largest-First V3, LDO-IDO and DSatur based coloring algorithms, we can see that in some cases they work considerably faster on middle densities. This could be connected to the fact that during the coloring phase they use lower amount of colors. A question immediately arises: DSatur based coloring algorithms produce better results in terms of used colors almost on every graph, so why the time consumption is lower only in some particular cases? The reason of this phenomenon might lie in the order of vertices in color classes as it affects the number of analyzed branches. It can be found from our results that when these coloring algorithms use less color classes but the time is still quite poor, then the number of analyzed branches is drastically higher. Furthermore, these coloring algorithms take more time than the Largest-First coloring algorithm.

Concerning parallel coloring algorithms, using Parallel Largest-First gives advantages on higher densities, from 50% to 90+%. However, we wanted to receive more effective results. Parallel Smallest-Last in this case proved to be non-effective. However, we should bear in mind that parallel coloring algorithms might be hostages of particular language implementation (which in our case is C#).

Graph	Size	Density	Time (ms)				
			Largest-First	Greedy	Largest-First V3	DSatur	DSatur V2
c-fat500-1.clq	500	0,36	9	🟩 3	21	58	60
c-fat500-2.clq	500	0,07	5	🟩 1	11	59	60
c-fat500-5.clq	500	0,19	6	🟩 1	17	98	92
c-fat500-10.clq	500	0,37	7	🟩 2	31	218	202
hamming10-2.clq	1024	0,99	40	🟩 23	335	10181	10419
hamming6-2.clq	64	0,9	0	🟧 0	1	3	2
hamming6-4.clq	64	0,35	0	🟧 0	🟧 0	1	1
hamming8-2.clq	256	0,97	7	🟩 2	22	181	170
hamming8-4.clq	256	0,64	5838	🟩 5616	5845	🟩 537	🟩 539
johnson16-2-4.clq	120	0,76	477	🟩 454	479	985	987
johnson8-2-4.clq	28	0,56	0	🟧 0	🟧 0	🟧 0	🟧 0
johnson8-4-4.clq	70	0,77	5	🟩 2	6	🟩 4	🟩 4
keller4.clq	171	0,65	506	949	1228	575	625
MANN_a27.clq	378	0,99	158062	302396	307324	307343	307322
MANN_a9.clq	45	0,93	0	390	6	7	7
p_hat300-1.clq	300	0,24	26	38	31	66	67
p_hat300-2.clq	300	0,49	1239	61336	🟩 1077	2705	2010
p_hat300-3.clq	300	0,74	317662	327279	🟩 317654	327281	327290
p_hat500-1.clq	500	0,25	286	423	288	464	455
p_hat500-2.clq	500	0,5	191068	317653	197190	308580	308638
p_hat700-1.clq	700	0,25	1312	2155	1347	1891	1943
p_hat1000-1.clq	1000	0,25	6529	10232	6825	8165	8293
san1000.clq	1000	0,5	14900	500027	540008	233091	245001
san200_0.7_1.clq	200	0,7	562509	500010	🟩 6416	🟩 226	🟩 253
san200_0.7_2.clq	200	0,7	28	7555	153	87	75
san200_0.9_1.clq	200	0,9	11716	456531	437523	🟩 7205	🟩 8432
san400_0.5_1.clq	400	0,5	584	866	25754	1149	1805

🟩	- lower time than original
🟧	- time equals to that of the original

Table 20. DIMACS graphs test results. VColor-u, time consumption (ms) - part 1.

Graph	**Size**	**Density**	**Time (ms)**				
			DSatur-LDO	**DSatur-IDO-LDO**	**LDO-IDO**	**Parallel Largest-First**	**Parallel Smallest-Last**
c-fat500-1.clq	500	0,36	58	68	13	62	63
c-fat500-2.clq	500	0,07	58	73	10	61	87
c-fat500-5.clq	500	0,19	98	111	19	452	153
c-fat500-10.clq	500	0,37	212	228	30	154	179
hamming10-2.clq	1024	0,99	10080	10327	357	🟩 32	152
hamming6-2.clq	64	0,9	2	3	1	38	17
hamming6-4.clq	64	0,35	1	1	🟧 0	8	7
hamming8-2.clq	256	0,97	163	248	15	🟩 5	172
hamming8-4.clq	256	0,64	🟩 741	🟩 137	🟩 5710	🟩 791	🟩 1302
johnson16-2-4.clq	120	0,76	🟩 420	🟩 343	🟩 462	🟩 338	735
johnson8-2-4.clq	28	0,56	🟧 0	🟧 0	🟧 0	4	5
johnson8-4-4.clq	70	0,77	🟩 4	🟩 3	🟩 3	14	15
keller4.clq	171	0,65	🟩 430	🟩 410	1235	621	854
MANN_a27.clq	378	0,99	300314	299996	302396	🟩 154543	🟩 145413
MANN_a9.clq	45	0,93	3	3	6	22	14
p_hat300-1.clq	300	0,24	67	72	31	117	127
p_hat300-2.clq	300	0,49	1514	1618	🟩 1150	🟩 1161	3004
p_hat300-3.clq	300	0,74	327279	327322	🟩 317655	🟩 317643	348388
p_hat500-1.clq	500	0,25	456	462	291	410	427
p_hat500-2.clq	500	0,5	290510	308598	🟩 166844	247311	327281
p_hat700-1.clq	700	0,25	1890	1965	1318	1523	1623
p_hat1000-1.clq	1000	0,25	8063	8478	6898	6652	7744
san1000.clq	1000	0,5	110824	165541	450011	17743	15235
san200_0.7_1.clq	200	0,7	🟩 17981	🟩 258	🟩 529422	562511	573120
san200_0.7_2.clq	200	0,7	90	111	76	52	116
san200_0.9_1.clq	200	0,9	🟩 7045	🟩 7550	🟩 1183	44045	600012
san400_0.5_1.clq	400	0,5	34048	1142	25563	1589	784

🟩	- lower time than original
🟧	- time equals to that of the original

Table 21. DIMACS graphs test results. VColor-u, time consumption (ms) - part 2.

Graph	Size	Density	Number of analyzed branches				
			Largest-First	Greedy	Largest-First V3	DSatur	DSatur V2
c-fat500-1.clq	500	0,36	14	14	14	14	14
c-fat500-2.clq	500	0,07	26	26	26	26	26
c-fat500-5.clq	500	0,19	64	64	64	64	64
c-fat500-10.clq	500	0,37	126	126	126	126	126
hamming10-2.clq	1024	0,99	512	512	512	512	512
hamming6-2.clq	64	0,9	32	32	32	32	32
hamming6-4.clq	64	0,35	426	426	426	340	340
hamming8-2.clq	256	0,97	128	128	128	128	128
hamming8-4.clq	256	0,64	2774392	2774392	2774392	165371	165371
johnson16-2-4.clq	120	0,76	489355	489355	489355	887871	887871
johnson8-2-4.clq	28	0,56	59	59	59	38	38
johnson8-4-4.clq	70	0,77	1196	1196	1171	364	364
keller4.clq	171	0,65	380922	745205	963672	382900	435550
MANN_a27.clq	378	0,99	8865948	273084144	43415359	50771824	50550310
MANN_a9.clq	45	0,93	303	478865	4156	5224	5224
p_hat300-1.clq	300	0,24	22355	38343	23068	25493	25170
p_hat300-2.clq	300	0,49	479046	29198380	394533	1147410	761014
p_hat300-3.clq	300	0,74	114123307	183826843	102086891	114729337	121480359
p_hat500-1.clq	500	0,25	252869	430099	261465	286782	268503
p_hat500-2.clq	500	0,5	57408633	156141644	59493075	91438502	105138526
p_hat700-1.clq	700	0,25	936779	1429594	977854	1091677	1039907
p_hat1000-1.clq	1000	0,25	5999391	10550831	65566251	6803864	6961846
san1000.clq	1000	0,5	1083046	300335950	62146686	20660950	21693998
san200_0.7_1.clq	200	0,7	821900625	338978198	1448106	47180	48251
san200_0.7_2.clq	200	0,7	10564	2832008	35546	9445	3502
san200_0.9_1.clq	200	0,9	4272475	290245174	87074934	110768683	1244520
san400_0.5_1.clq	400	0,5	108891	205972	8623043	289927	487406

Table 22. DIMACS graphs test results. VColor-u, number of branches - part 1.

Graph	Size	Density	Number of analyzed branches				
			DSatur-LDO	DSatur-IDO-LDO	LDO-IDO	Parallel Largest-First	Parallel Smallest-Last
c-fat500-1.clq	500	0,36	14	14	14	159	720715
c-fat500-2.clq	500	0,07	26	26	26	385	720715
c-fat500-5.clq	500	0,19	64	64	64	72182	720715
c-fat500-10.clq	500	0,37	126	126	126	126	720715
hamming10-2.clq	1024	0,99	512	512	512	512	512
hamming6-2.clq	64	0,9	32	32	32	389	2768
hamming6-4.clq	64	0,35	307	352	426	352	306
hamming8-2.clq	256	0,97	128	128	128	128	128
hamming8-4.clq	256	0,64	252410	215	5774392	328584	547822
johnson16-2-4.clq	120	0,76	401470	353522	489355	353527	734819
johnson8-2-4.clq	28	0,56	38	37	59	59	38
johnson8-4-4.clq	70	0,77	554	18	1171	1927	556
keller4.clq	171	0,65	278088	277698	961304	436896	575538
MANN_a27.clq	378	0,99	16577654	17043773	35077916	8152230	8262697
MANN_a9.clq	45	0,93	2171	2171	4093	315	317
p_hat300-1.clq	300	0,24	25193	26001	23352	22251	25194
p_hat300-2.clq	300	0,49	553779	630453	411228	376051	1158296
p_hat300-3.clq	300	0,74	113663179	109084862	108658872	105538037	165318529
p_hat500-1.clq	500	0,25	272261	278488	254312	252289	282831
p_hat500-2.clq	500	0,5	87977525	101058410	48253356	77150568	129851667
p_hat700-1.clq	700	0,25	986604	1104598	893465	957306	1037665
p_hat1000-1.clq	1000	0,25	6626485	7107052	6539346	6139185	7099445
san1000.clq	1000	0,5	40504278	53019726	6969681	1324140	1024140
san200_0.7_1.clq	200	0,7	10344610	87844	566446071	926906250	1323242125
san200_0.7_2.clq	200	0,7	9596	18488	23459	3810	1622
san200_0.9_1.clq	200	0,9	11060105	1241313	995610	27560687	360822143
san400_0.5_1.clq	400	0,5	22985290	267929	4783553	720715	177720

Table 23. DIMACS graphs test results. VColor-u, number of branches - part 2.

3.4.2.3. Results of VRecolor-BT-u Based Algorithms

In this subchapter the same coloring algorithms are used inside VRecolor-BT-u algorithm and tested on DIMACS graph instances because each of them showed the best results separately.

Graph	Size	Density	Time (ms)				
			Largest-First	**Greedy**	**Largest-First V3**	**DSatur**	**DSatur V2**
c-fat500-10.clq	500	0,37	222	213	384	428	411
hamming6-2.clq	64	0,9	3	4	4	5	5
hamming6-4.clq	64	0,35	0	0	0	1	1
hamming8-2.clq	256	0,97	471	459	446	611	626
hamming8-4.clq	256	0,64	20	21	30	417	419
hamming10-2.clq	1024	0,99	118039	118853	120835	133855	134101
johnson16-2-4.clq	120	0,76	1043	1065	1043	882	916
johnson8-2-4.clq	28	0,56	0	0	0	0	0
johnson8-4-4.clq	70	0,77	1	1	3	5	5
keller4.clq	171	0,65	118	239	267	224	208
MANN_a27.clq	378	0,99	18719	3436364	1467961	28119	27574
MANN_a9.clq	45	0,93	1	15	5	3	4
p_hat300-2.clq	300	0,49	449	1285	440	646	897
p_hat300-3.clq	300	0,74	40402	1309090	26419	61057	132641
p_hat500-2.clq	500	0,5	12876	143463	19195	31308	27358
san1000.clq	1000	0,5	1436	30581	1104	17172	24113
san200_0.7_1.clq	200	0,7	3091	49	34	123	76
san200_0.7_2.clq	200	0,7	11	847	26	77	137
san200_0.9_1.clq	200	0,9	83	410	129	144	41642
san400_0.5_1.clq	400	0,5	40	518	80	499	524

🟩 - lower time than original

🟨 - time equals to that of the original

Table 24. DIMACS graphs test results. VRecolor-BT-u, time consumption (ms) - part 1.

As can be seen from Table 24, VRecolor-BT-u algorithm, which uses Greedy algorithm under the hood, shows better results on density 40-50%. However, it is possible to notice that in some cases it shows better results on higher densities as well, for example, on 70% and 90+% densities. This behavior can be simply explained by the number of found color classes (Table 28) as well as number of analyzed branches (Table 26). The number of used colors along with number of analyzed branches are the same, so the algorithm just works faster as it does not use any orderings.

When comparing original VRecolor-BT-u algorithm with its siblings that include DSatur based coloring algorithms, we can see that in some cases on densities 50-80% they work considerably faster. As stated above, this could be connected to the fact that during the coloring phase they use lower amount of colors and therefore maximum clique algorithm creates less branches (Table 26 and Table 27).

Graph	**Size**	**Density**	**Time (ms)**				
			DSatur-LDO	**DSatur-IDO-LDO**	**LDO-IDO**	**Parallel Largest-First**	**Parallel Smallest-Last**
c-fat500-10.clq	500	0,37	427	449	381	240	🟩 134
hamming6-2.clq	64	0,9	6	6	4	5	6
hamming6-4.clq	64	0,35	1	1	🟨 0	🟨 0	1
hamming8-2.clq	256	0,97	607	607	478	🟩 447	519
hamming8-4.clq	256	0,64	329	299	24	401	300
hamming10-2.clq	1024	0,99	134294	134033	118279	🟩 79611	422894
johnson16-2-4.clq	120	0,76	🟩 820	🟩 697	🟩 1013	🟩 657	🟩 644
johnson8-2-4.clq	28	0,56	🟨 0	🟨 0	🟨 0	🟨 0	13
johnson8-4-4.clq	70	0,77	6	6	2	2	43
keller4.clq	171	0,65	169	166	284	151	281
MANN_a27.clq	378	0,99	27539	27988	1467961	28790	29072
MANN_a9.clq	45	0,93	2	2	12	8	11
p_hat300-2.clq	300	0,49	675	724	500	548	894
p_hat300-3.clq	300	0,74	59058	42935	🟩 27934	🟩 27859	303207
p_hat500-2.clq	500	0,5	32792	35487	17517	🟩 11173	28584
san1000.clq	1000	0,5	12110	14437	🟩 630	2233	2668
san200_0.7_1.clq	200	0,7	🟩 97	🟩 85	🟩 35	🟩 537	🟩 115
san200_0.7_2.clq	200	0,7	73	76	24	36	68
san200_0.9_1.clq	200	0,9	130	135	170	1107	100
san400_0.5_1.clq	400	0,5	459	467	🟩 35	88	92

🟩	- lower time than original
🟨	- time equals to that of the original

Table 25. DIMACS graphs test results. VRecolor-BT-u, time consumption (ms) - part 2.

It should be noted that VRecolor-BT-u algorithms, using Largest-First V3 and LDO-IDO colorings, demonstrated results similar to those, which we received on random graphs. These algorithms performed better on densities starting from 40% and ending with 76%.

Graph	**Size**	**Density**	**Number of analyzed branches**				
			Largest-First	**Greedy**	**Largest-First V3**	**DSatur**	**DSatur V2**
c-fat500-10.clq	500	0,37	8001	8001	8001	8001	8001
hamming6-2.clq	64	0,9	528	528	528	529	529
hamming6-4.clq	64	0,35	70	70	70	70	70
hamming8-2.clq	256	0,97	8256	8256	8256	8360	8360
hamming8-4.clq	256	0,64	788	788	788	10438	10438
hamming10-2.clq	1024	0,99	131328	131328	131328	132757	132757
johnson16-2-4.clq	120	0,76	323070	323070	323070	243098	243098
johnson8-2-4.clq	28	0,56	44	44	44	32	32
johnson8-4-4.clq	70	0,77	252	252	288	344	344
keller4.clq	171	0,65	11236	20973	27319	14579	12986
MANN_a27.clq	378	0,99	55389	35324182	24192455	61454	61454
MANN_a9.clq	45	0,93	189	1880	1006	510	510
p_hat300-2.clq	300	0,49	24826	97437	26289	32004	57527
p_hat300-3.clq	300	0,74	664515	21599985	683863	1202866	2973543
p_hat500-2.clq	500	0,5	584983	9828984	912033	1260838	1109719
san1000.clq	1000	0,5	13356	462145	11043	78374	182995
san200_0.7_1.clq	200	0,7	547738	3326	1492	9323	1186
san200_0.7_2.clq	200	0,7	398	44161	1105	794	3227
san200_0.9_1.clq	200	0,9	3350	9303	3535	2817	4876475
san400_0.5_1.clq	400	0,5	1241	25773	2519	3069	3924

Table 26. DIMACS graphs test results. VRecolor-BT-u, number of branches - part 1.

Graph	Size	Density	Number of analyzed branches				
			DSatur-LDO	DSatur-IDO-LDO	LDO-IDO	Parallel Largest-First	Parallel Smallest-Last
c-fat500-10.clq	500	0,37	8001	8001	8001	4966	2230
hamming6-2.clq	64	0,9	529	529	528	572	248
hamming6-4.clq	64	0,35	70	70	70	70	70
hamming8-2.clq	256	0,97	8360	8360	8256	7248	8773
hamming8-4.clq	256	0,64	7082	5814	788	15786	13156
hamming10-2.clq	1024	0,99	132757	132757	131328	86553	688566
johnson16-2-4.clq	120	0,76	243690	218422	323070	218422	218422
johnson8-2-4.clq	28	0,56	32	31	44	32	31
johnson8-4-4.clq	70	0,77	335	351	288	262	221
keller4.clq	171	0,65	11281	10739	28920	13646	18629
MANN_a27.clq	378	0,99	61438	61491	24472404	74943	74690
MANN_a9.clq	45	0,93	201	201	1176	189	184
p_hat300-2.clq	300	0,49	34257	35163	29558	24116	55365
p_hat300-3.clq	300	0,74	1058004	868682	746081	658401	11195415
p_hat500-2.clq	500	0,5	1326476	1403129	808703	457809	1233472
san1000.clq	1000	0,5	12124	55128	9001	19332	17620
san200_0.7_1.clq	200	0,7	4532	2458	1596	60262	1846
san200_0.7_2.clq	200	0,7	443	512	1089	391	320
san200_0.9_1.clq	200	0,9	2570	2561	4837	27582	3071
san400_0.5_1.clq	400	0,5	1229	1113	781	1350	1942

Table 27. DIMACS graphs test results. VRecolor-BT-u, number of branches - part 2.

Concerning parallel coloring algorithms, using Parallel Largest-First gives the upper hand on densities from 50% to 90+% and its results are quite good. This cannot be said about Parallel Smallest-Last, which showed similar results to those, which we receive while testing VColor-u based algorithms. However, we should remember about implementation difficulties, connected with language specifics.

4. Conclusion

4.1. Summary

The main topic of our study was to investigate effect of applying coloring algorithms on modern maximum clique algorithms. The problem of coloring a graph with the minimum number of colors is NP-complete task. Therefore, we had to use heuristic algorithms for that purpose. A heuristic algorithm does not guarantee the best result, however, its result is close enough to the best one and the algorithm is faster than the exact one. Vertex coloring is a subroutine, which is included into the maximum clique algorithm, so this step affects the overall performance of the algorithm. The idea behind this was the fact that the closer the number of color classes to the size of the maximum clique, the quicker the maximum clique will be found thanks to more effective pruning. As we know, even a small increase in the size of the maximum clique can result in extra days of work. Therefore, even a small improvement would significantly save working time.

In this resume, we are going to summarize all the work done to reach the goals stated in *subchapter 1.4*. The majority of them are successfully completed in the scope of current work.

Chapter 2 describes graph coloring problem along with introducing different types of coloring algorithms, their history and specifics. Overall, 17 coloring algorithms were described. Some of these coloring algorithms were variations of themselves, however, they proved their chance for existence with tests results. We found out that almost every coloring algorithm, which uses some sort of ordering/reordering would perform better than the Greedy algorithm in terms of number of color classes. However, because of that they would spend much more time to complete. Furthermore, within our research we found out that in theory parallel coloring algorithms should be faster than their sequential variations. In practice, there are too many factors that affect their performance. And the most important one is without doubt the language, chosen for their implementation. In our case, the implementation of parallel functions in C# language spends more time on creating threads and subroutines executed inside are trivial in terms of time consumption.

In *Chapter 3* we introduced maximum clique algorithms that we tried to improve with explanations why such a choice was made. It was decided to take into consideration modern maximum clique algorithms, such as VColor-u and VRecolor-BT-u. Every algorithm was briefly described, modified, compared and tested against random and DIMACS graph instances. Generated tests allow to obtain comparative data that can be represented in a table and demonstrate time consumption of maximum clique algorithms with different coloring algorithms used. On the other hand, DIMACS benchmark instances allow to test the algorithms on problems that are very close to real life as they are constructed based on real tasks. Moreover, in addition to time consumption there are results showing us the number of classes used by one or another coloring algorithm and number of analyzed branches of maximum clique algorithms, which help us to explain that or another behavior of algorithms. We proved that Largest-First V3, DSatur, DSatur V2, DSatur-LDO, DSatur-IDO-LDO, Parallel Largest-First in certain cases could decrease the time of the maximum clique algorithms, as well as the number of created branches. Furthermore, we saw that on lower densities it is possible to use Greedy coloring algorithm to decrease the overall algorithm’s time. Although, we could not find a specific pattern, the results are quite promising and could be used in further researches.

4.2. Future studies

During our research some interesting questions were left behind the scope of our work. In this subchapter we are going to introduce some ideas which could be used in future studies to improve results of our research.

First of all, it is necessary to investigate the influence of vertex ordering in color classes found by coloring algorithms as it heavily affects the main routine of maximum clique algorithms. However, it could be specific for every maximum clique algorithm but worth researching.

Secondly, in our research we have tested parallel algorithms’ approach, using Parallel Largest-First and Smallest-Last algorithms. Further studies require extending this approach by constructing parallel algorithms from other coloring algorithms that showed good results. And there are at least two candidates for this: DSatur and LDO-IDO. This may lead to decreasing the overall time taken to color the graph with minimum number of colors.

Thirdly, it is possible to try to construct a new coloring algorithm, which is similar to Largest-First or Greedy but is faster and simpler than they are. The aim of this algorithm would be to reduce the execution time of the coloring algorithm.

And finally, there were some strange behaviors of algorithms found in the result of our research that could become an inception point for future studies.

Resümee

Kokkuvõte

Meie uurimistöö peamine teema on uurida värvimisalgoritmide rakendamise mõju tänapäevastele maksimaalsete klikkide algoritmidele. Graafide minimaalse värvide arvuga värvimise probleem on „NP-complete“ keerukuse tasemega. Seetõttu tuli kasutada heuristilisi algoritme. Heuristiline algoritm ei taga parimat tulemust, kuid see on parimale tulemusele piisavalt lähedal, ning selle kiirus on suurem kui täpse algoritmi oma. Tipu värvimine on alamrutiin, mis on hõlmatud maksimaalsete klikkide algoritmi, järelikult mõjutab see samm üldist algoritmi jõudlust. Selle taga on mõte, et mida lähemal on värvide klasside arv maksimaalse kliki suurusele, seda kiiremini leitakse tänu tõhusamale kärpimisele maksimaalne klikk. Nagu me teame, isegi väike maksimaalse kliki suuruse kasv võib tähendada mitut lisa töö päeva. Seetõttu hoiab isegi väike edenemine kokku märkimisväärselt töö aega.

Selles resümees võtame kokku kogu tehtud töö, et jõuda alapeatükis 1.4. toodud eesmärkideni. Suurem osa neist õnnestus edukalt selle töö käigus täita.

Peatükk 2 kirjeldab graafi värvimise probleemi koos erinevat tüüpi värvimise algoritmide tutvustamise, nende ajaloo ja üksikasjadega. Kokku kirjeldatakse 17 värvimise algoritmi. Mõned neist algoritmidest on iseenda variatsioonid, kuid nad on enda olemasolu vajalikkust katsete tulemusel tõestanud. Me avastasime, et peaaegu iga värvimise algoritm, mis kasutab mõnda järjestamist või ümberjärjestamist, toimib värvide klasside arvu mõttes paremini kui „Greedy“ algoritm. Kuid selle tõttu kulub nende lõpetamiseks palju rohkem aega. Peale selle avastasime selles uurimistöös, et teoreetiliselt peaksid paralleelse värvimise algoritmid olema kiiremad kui nende järjestikused variatsioonid. Praktikas on aga nende jõudlust mõjutavad tegureid liiga palju. Ja kahtluseta on kõige olulisem neist rakenduse keel. Meie näite puhul kulutas C# keelne paralleelsete funktsioonide rakendus rohkem aega iseseisvate protsesside (inglise keeles: „thread“) tekitamisele ja alamrutiinide sisemine käitamine on ajakulu mõttes tühine.

Peatükis 3 tutvustasime maksimaalsete klikkide algoritme, mida me üritasime parendada, koos nende valiku kasuks osutumise selgitustega. Otsustati võrdlusse võtta tänapäevased maksimaalsete klikkide algoritmid nagu „VColorU“ ja „VRecolor-BT-u“. Iga algoritmi tutvustati põgusalt, muudeti ja võrreldi ja testiti juhuslike ja DIMACS graafidega. Loodud testid lubavad saada võrreldavaid andmeid, mida saab esitada tabelina ja näidata maksimaalsete klikkide algoritmide ajakulu erinevate kasutatud värvimise algoritmidega. Teiselt poolt, DIMACS jõudlustestid lubavad katsetada algoritme probleemidega, mis on väga lähedal reaalse elu probleemidele, sest nad on koostatud tegelike ülesannete baasil. Lisaks ajakulule näitavad tulemused meile ühe või mitme värvimise algoritmi poolt kasutatud klasside arvu ning analüüsitud harude arvu maksimaalsete klikkide algoritmide poolt, mis aitavad meil seletada seda või teist algoritmi käitumist. Me tõestasime, et „Largest-First V3”, “DSatur, DSatur V2”, “DSatur-LDO”, “DSatur-IDO-LDO” ja mõnes olukorras “Parallel Largest-First” võivad vähendada maksimaalsete klikkide algoritmide ajakulu ning loodud harude arvu. Lisaks sellele nägime, et madalama tiheduse puhul on võimalik kasutada kogu algoritmi aja vähendamiseks „Greedy“ värvimise algoritmi. Kuigi me ei suutnud tuvastada täpset mustrit, olid tulemused päris lootustandvad ja neid saab kasutada tulevastes uurimistöödes.

Edasised uurimistööd

Mõned kerkinud huvitavad küsimused jäid väljapoole meie uurimistöö raame. Selles alapeatükis tutvustame mõnda nendest ideedest, mida võib kasutada edasistes uurimistöödes, et meie uurimistöö tulemusi parandada.

Kõigepealt on vaja uurida tippude järjestamise mõju värvimise algoritmide poolt leitud värvimisklassidele, sest need mõjutavad tugevalt maksimaalsete klikkide algoritmi pearutiini. Kuid see võib olla iga maksimaalsete klikkide algoritmi puhul erinev, aga see on väärt edasist uurimist.

Teiseks me katsetasime oma uurimistöös paralleelsete algoritmide lähenemist, kasutades “Parallel Largest-First” ja “Smallest-Last” algoritme. Edasised uuringud nõuavad selle lähenemise laiendamist koostades paralleelseid algoritme teistest värvimise algoritmidest, mis näitasid häid tulemusi. Ja selleks on vähemalt kaks kandidaati: „DSatur” ja “LDO-IDO”. See võib viia kogu graafi minimaalse arvu värvidega värvimiseks kuluva aja vähenemisele.

Kolmandaks on võimalik üritada luua uut värvimise algoritmi, mis on sarnane „Largest-First” või “Greedy” algoritmidele, kuid on kiirem ja lihtsam kui need. Selle algoritmi eesmärk oleks vähendada värvimise algoritmi käitamise aega.

Ja lõpuks leiti meie uurimistöö tulemusel mõned algoritmide imelikud käitumised, mis võivad saada uute uurimistööde lähtekohaks.

References

[1] Norman L. Biggs, E. Keith Lloyd, Robin J. WIlson, Graph Theory 1736-1936, New York: Oxford University Press, 1976.

[2] M. Kubale, Graph Colorings, 2004.

[3] D. Kumlander, Some Practical Algorithms to Solve The Maximum Clique Problem, Tallinn, 2005.

[4] D. J. A. Welsh and M. B. Powell, «An upper bound for the chromatic number of a graph and its application to timetabling problems,» *The Computer Journal,* 1967.

[5] William Hasenplaugh, Tim Kaler, Tao B. Schardl, Charles E. Leiserson, «Ordering Heuristics for Parallel Graph Coloring,» 2014.

[6] D. Brelaz, «New Methods to Color the Vertices of a Graph,» *Communications of the ACM,* 1979.

[7] Paul S. Andrews, Jon Timmis, Nick D.L.Owens, Uwe Aickelin, Emma Hart, Andrew Hone, Andy M. Tyrrell, Artificial Immune Systems, York, UK, 2009.

[8] T. F. Coleman and J. J. More, «Estimation of sparse Jacobian matrices and graph coloring problems,» *SIAM Journal on Numerical Analysis,* 1983.

[9] Hilal Almara'Beh and Amjad Suleiman , «Heuristic Algorithm for Graph Coloring Based On Maximum Independent Set,» *Journal of Applied Computer Science & Mathematics,* т. 6, № 13, 2012.

[10] Hussein Al-Omari, Khair Eddin Sabri, «New Graph Coloring Algorithms,» *American Journal of Mathematics and Statistics,* 2006.

[11] Soma Saha, Gyan Baboo, Rajeev Kumar , «An Efficient EA with Multipoint Guided Crossover for Bi-objective Graph Coloring Problem,» в *Contemporary Computing*, 2011.

[12] M. T. Jones and P. E. Plassmann, «A parallel graph coloring heuristic.,» *SIAM Journal on Scientific Computing,* 1993.

[13] L. M., «A simple parallel algorithm for the maximal independent set problem,» *SIAM Journal*

on Computing, т. 4, № 97, pp. 1053-1063, 1986.

[14] D.W.Matula, G.Marble and J.D.Isaacson, Graph Coloring Algorithms, New York: Academic Press, 1972.

[15] J. R. Allwright , R. Bordawekar , P. D. Coddington , K. Dincer , C. L. Martin, «A Comparison of Parallel Graph Coloring Algorithms,» *Technical Report Tech. Rep. SCCS-666,* 1995.

[16] A. Porošin, «Reversed search maximum clique algorithm based on recoloring» 2015.

Appendix 1 – Coloring algorithms: Randomly Generated Graphs Test Results

Sequential algorithms

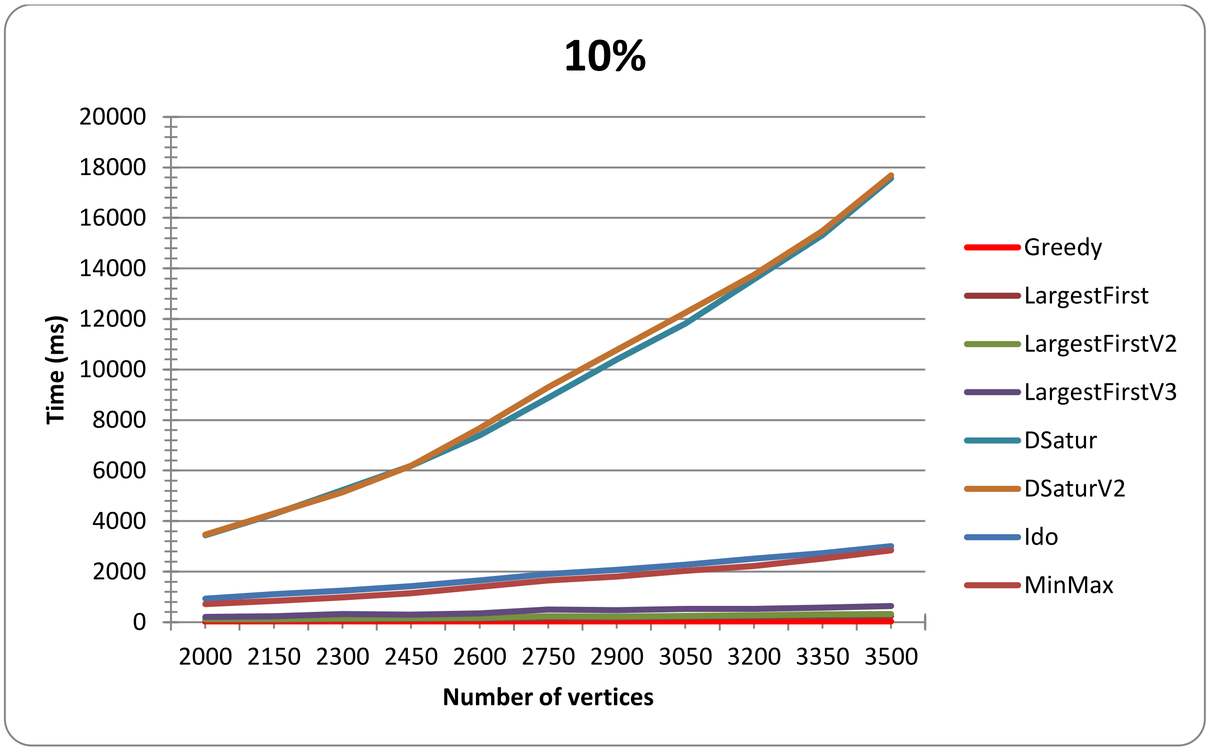

Figure 54. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 10%.

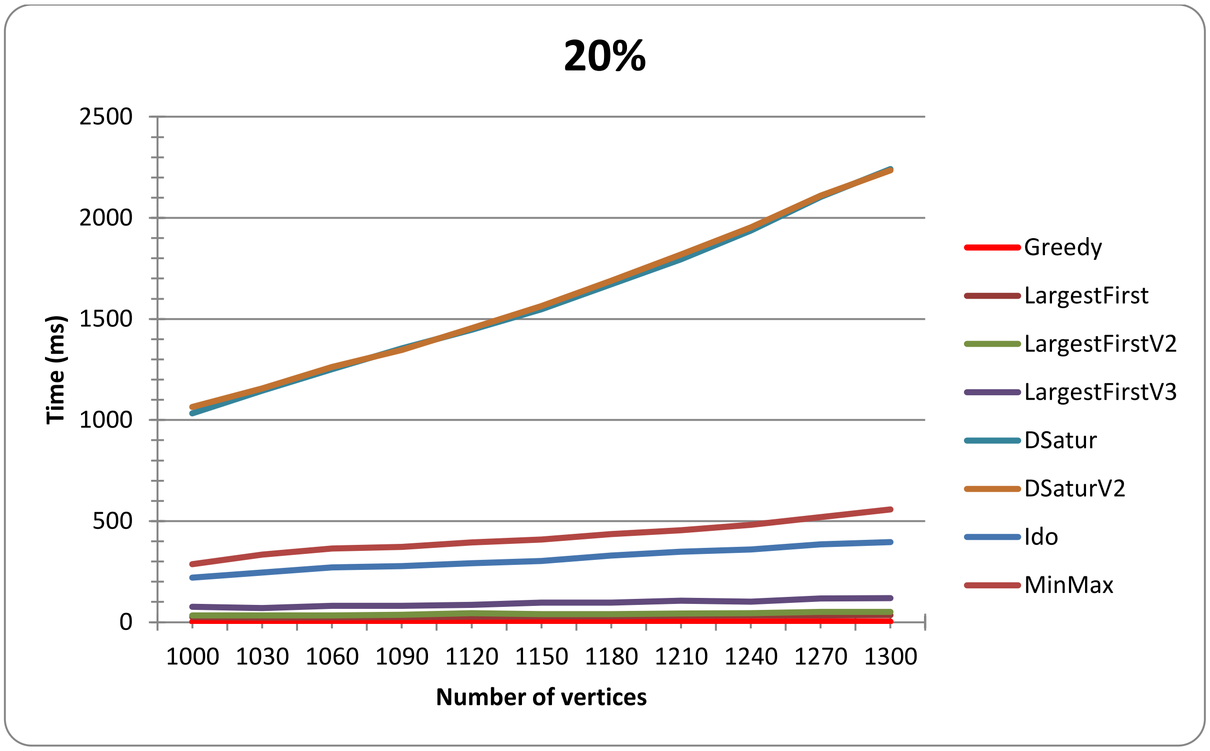

Figure 55. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 20%.

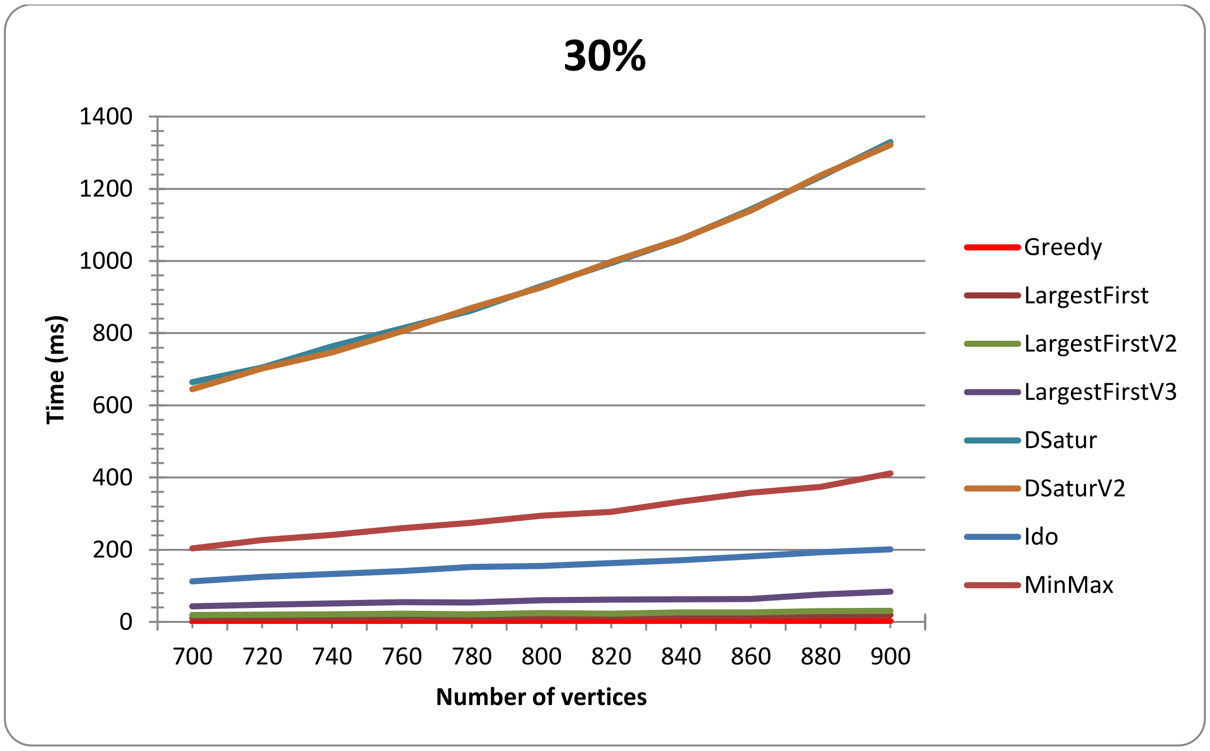

Figure 56. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 30%.

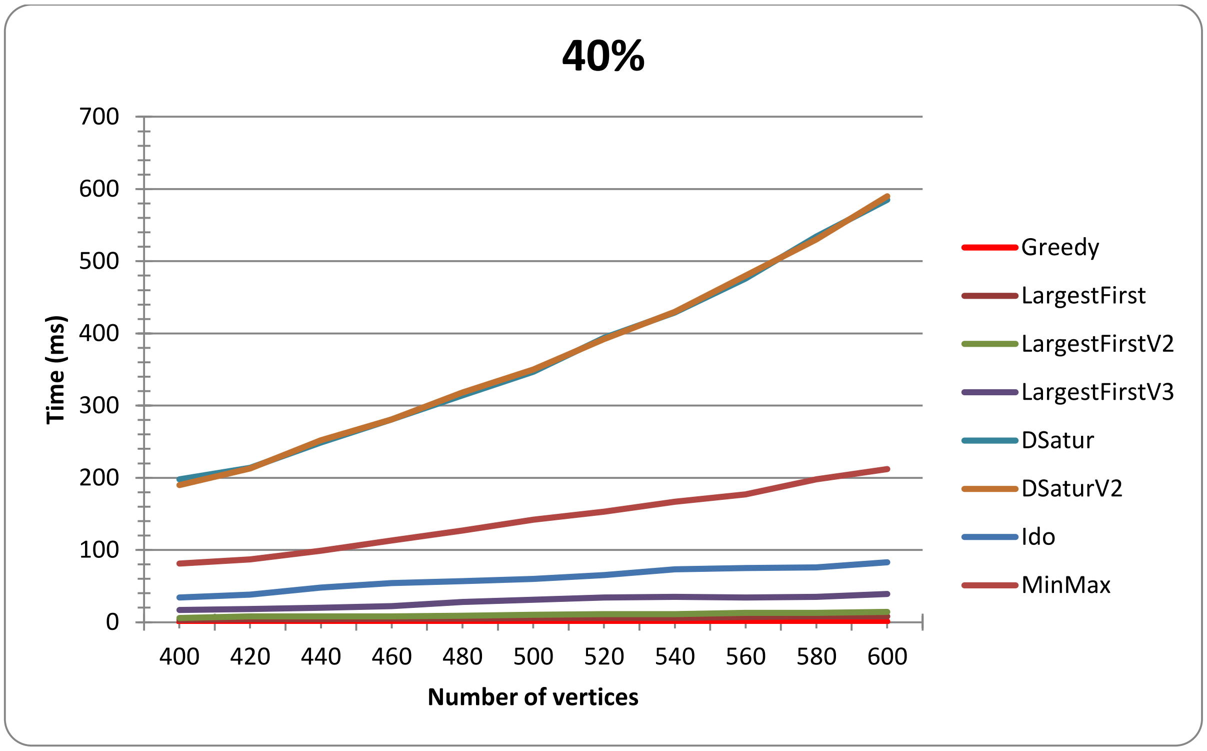

Figure 57. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 40%.

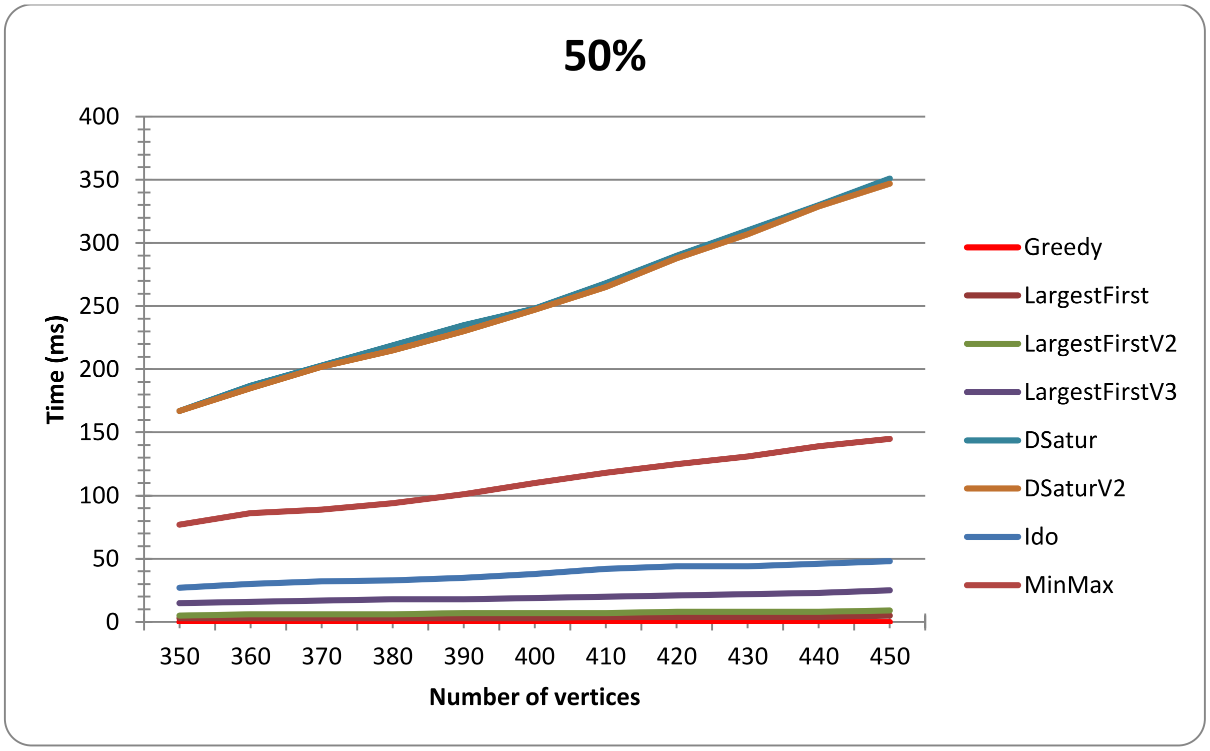

Figure 58. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 50%.

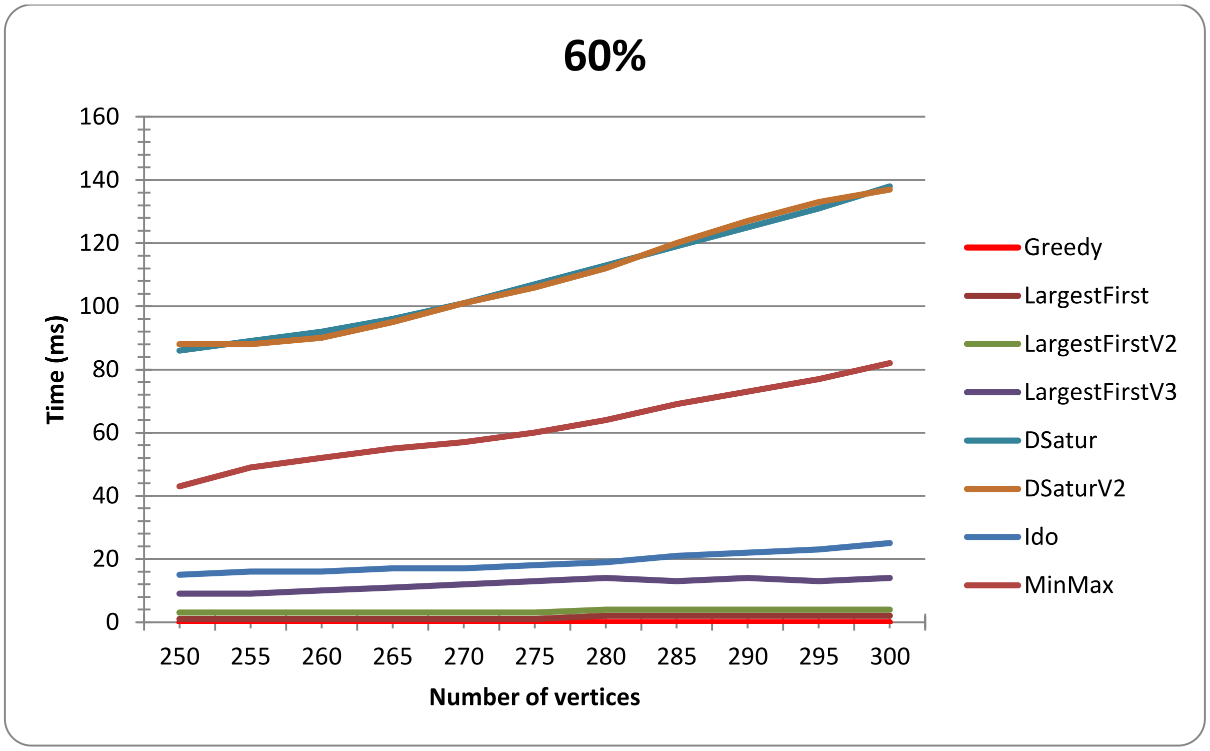

Figure 59. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 60%.

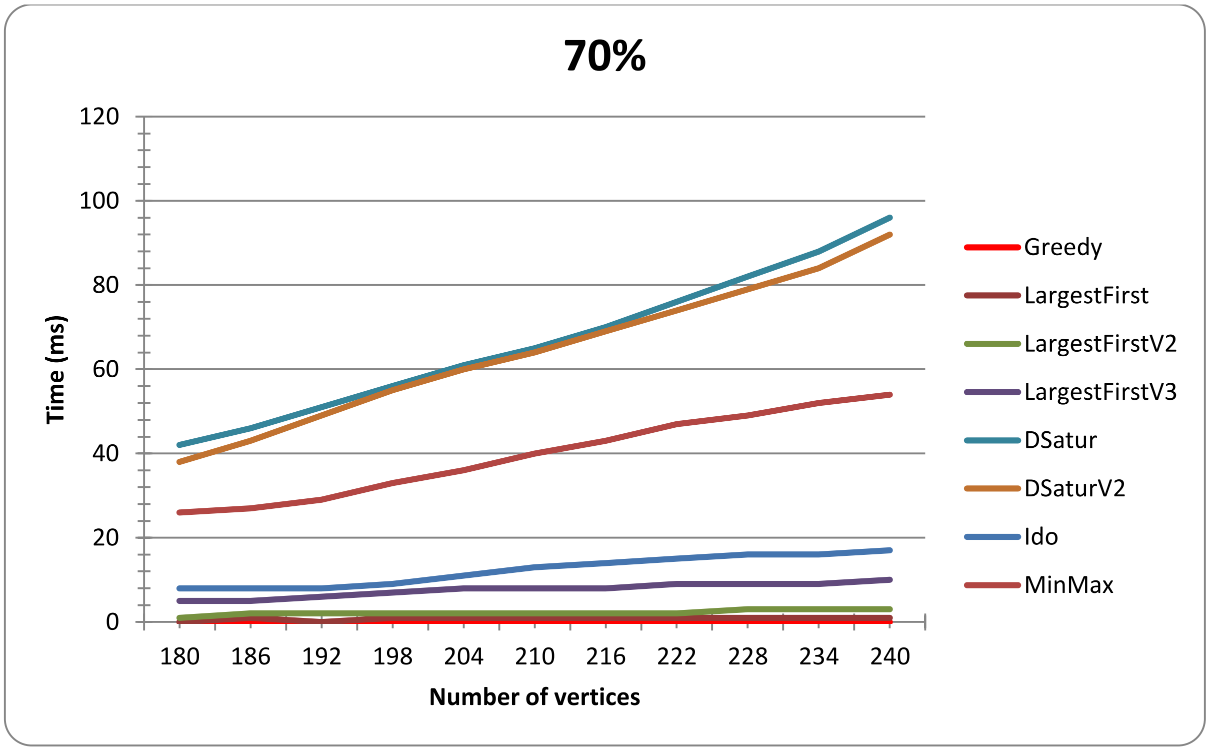

Figure 60. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 70%.

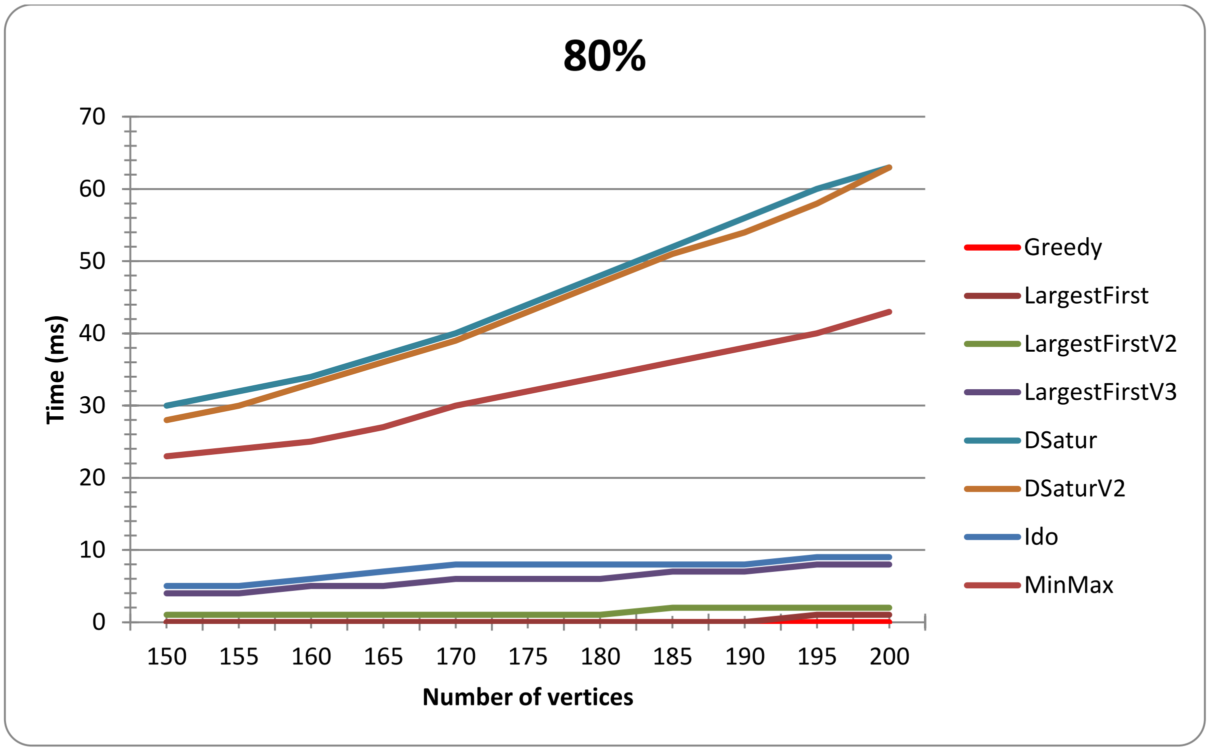

Figure 61. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 80%.

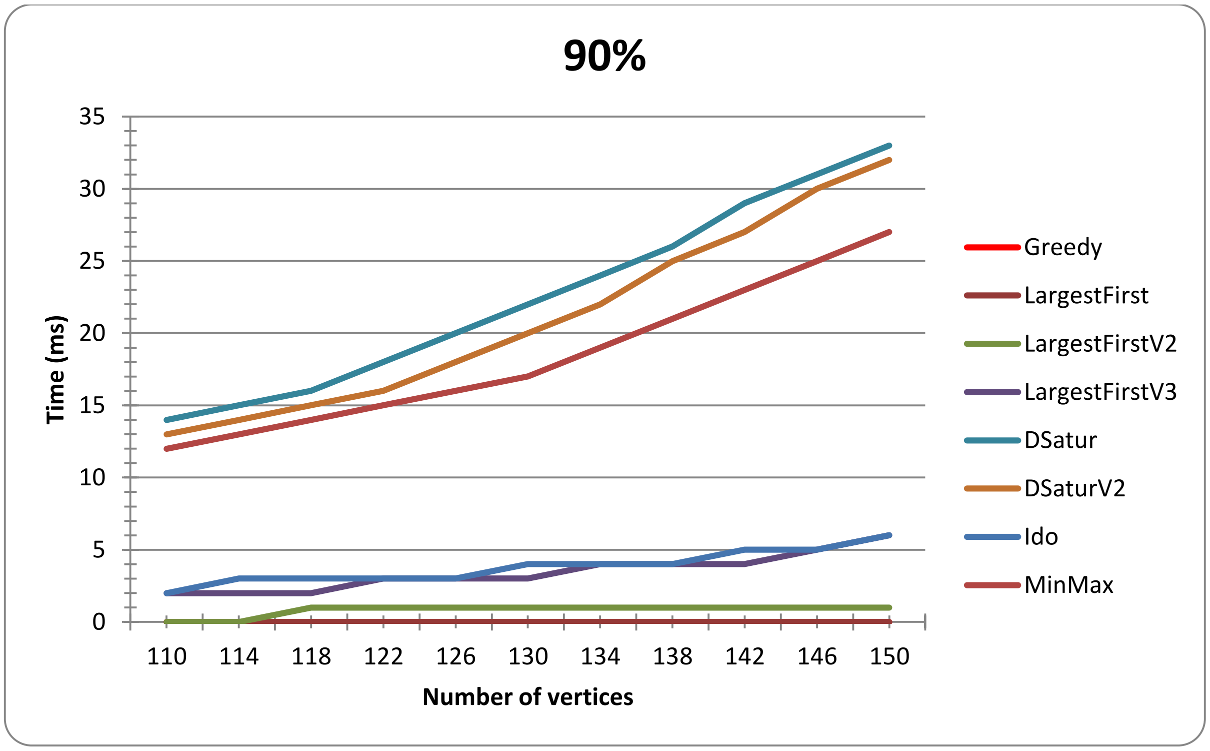

Figure 62. Randomly generated graphs tests' results compared in time (ms). Sequential algorithms, density 90%.

Combined algorithms

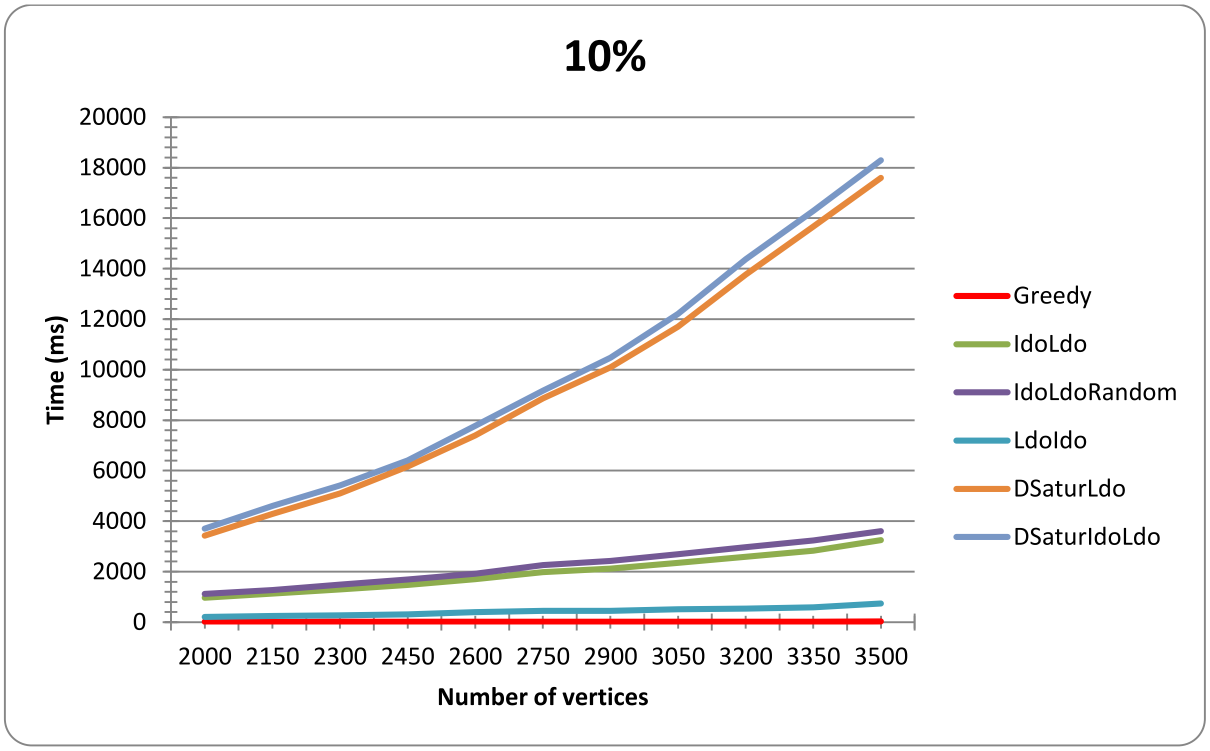

Figure 63. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 10%.

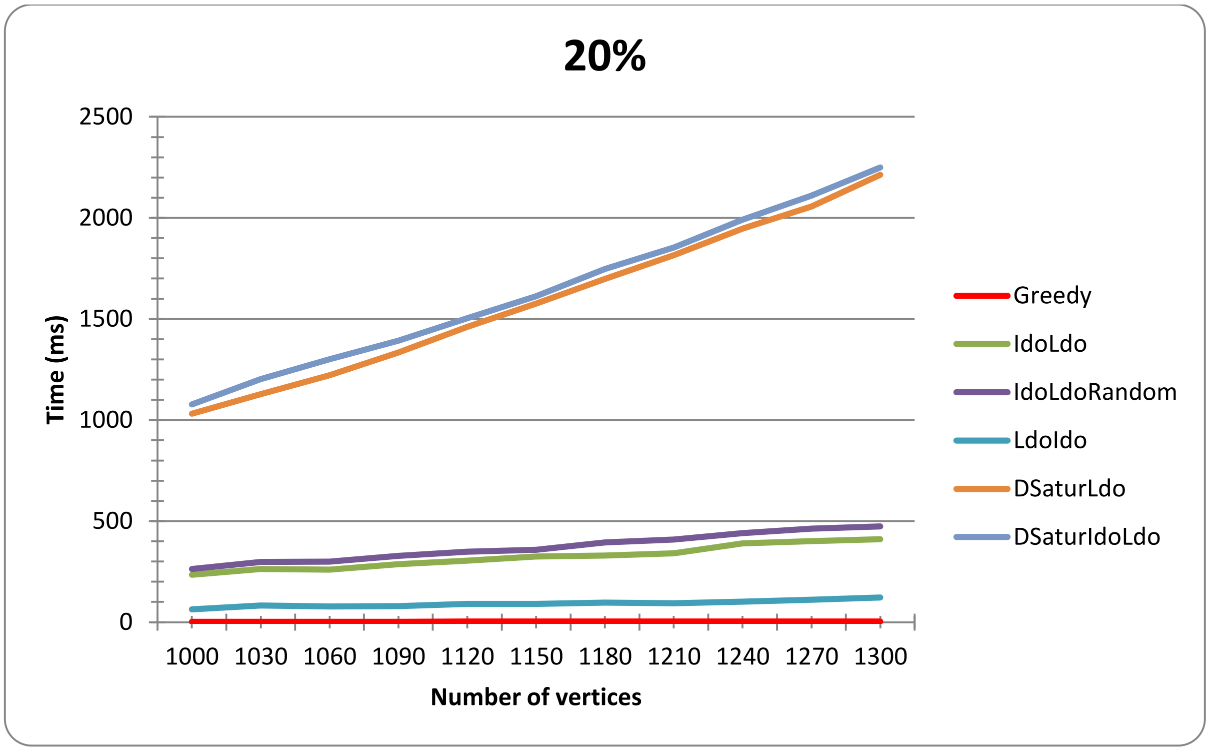

Figure 64. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 20%.

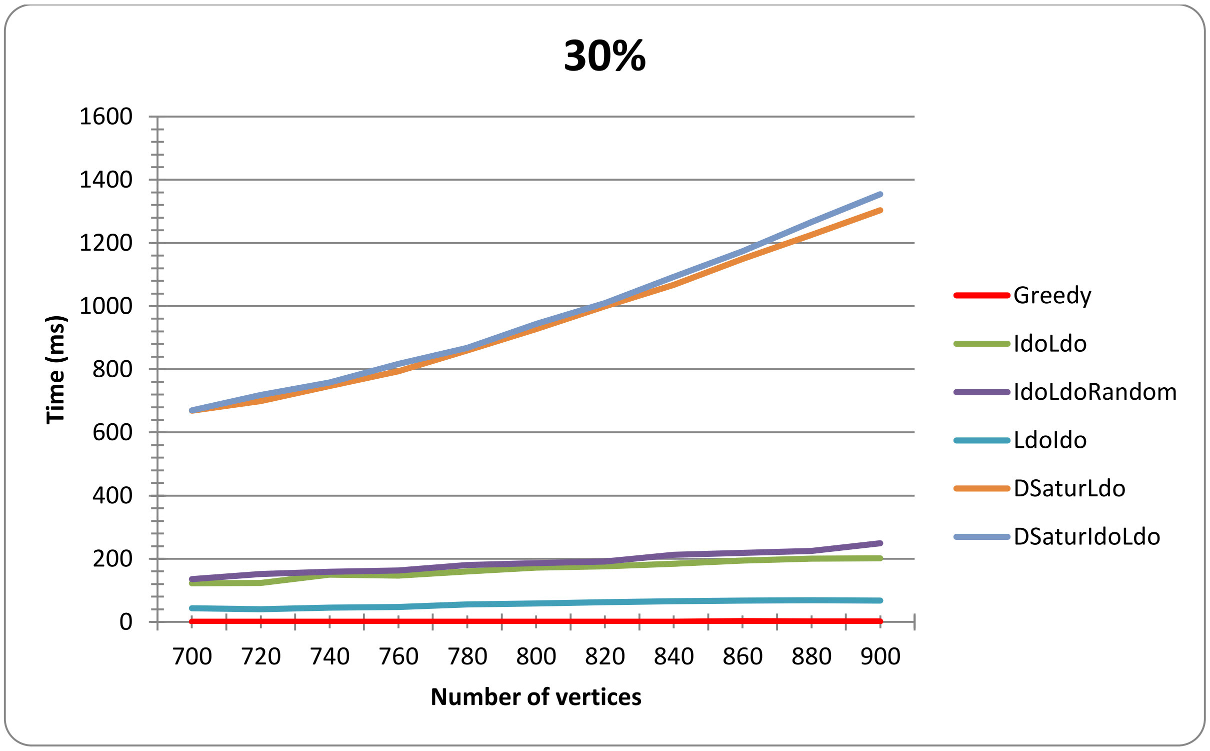

Figure 65. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 30%.

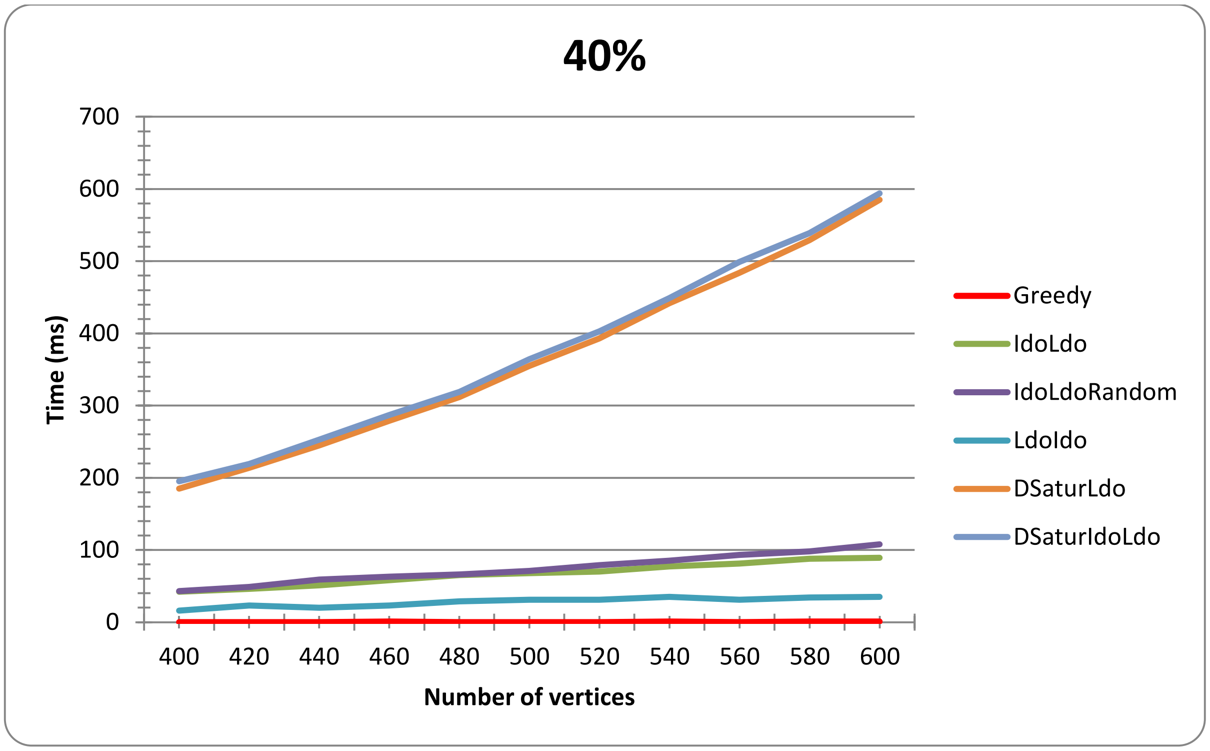

Figure 66. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 40%.

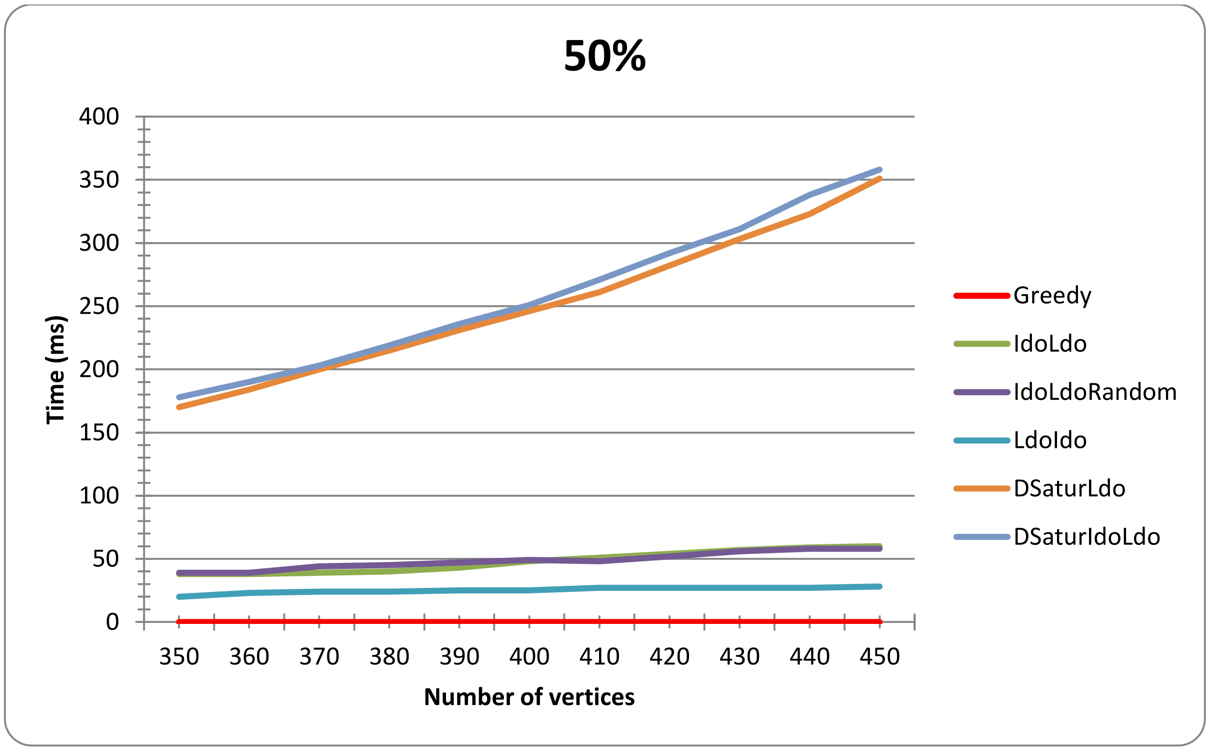

Figure 67. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 50%.

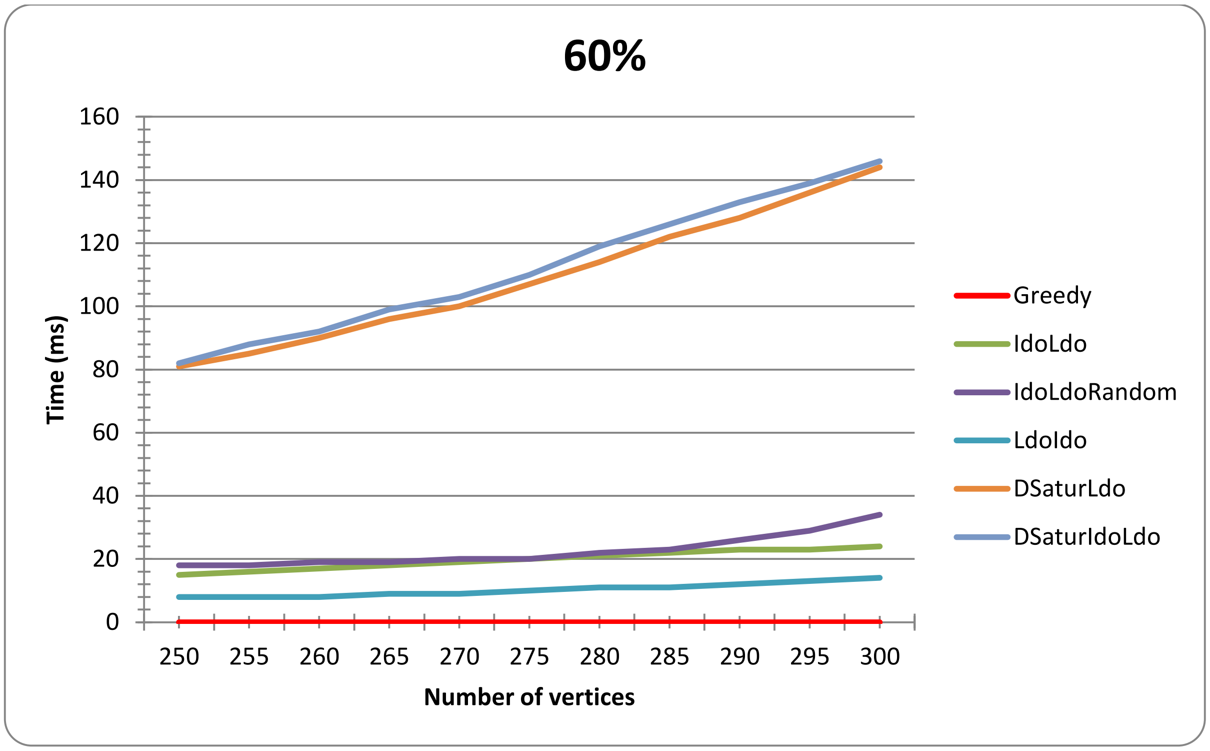

Figure 68. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 60%.

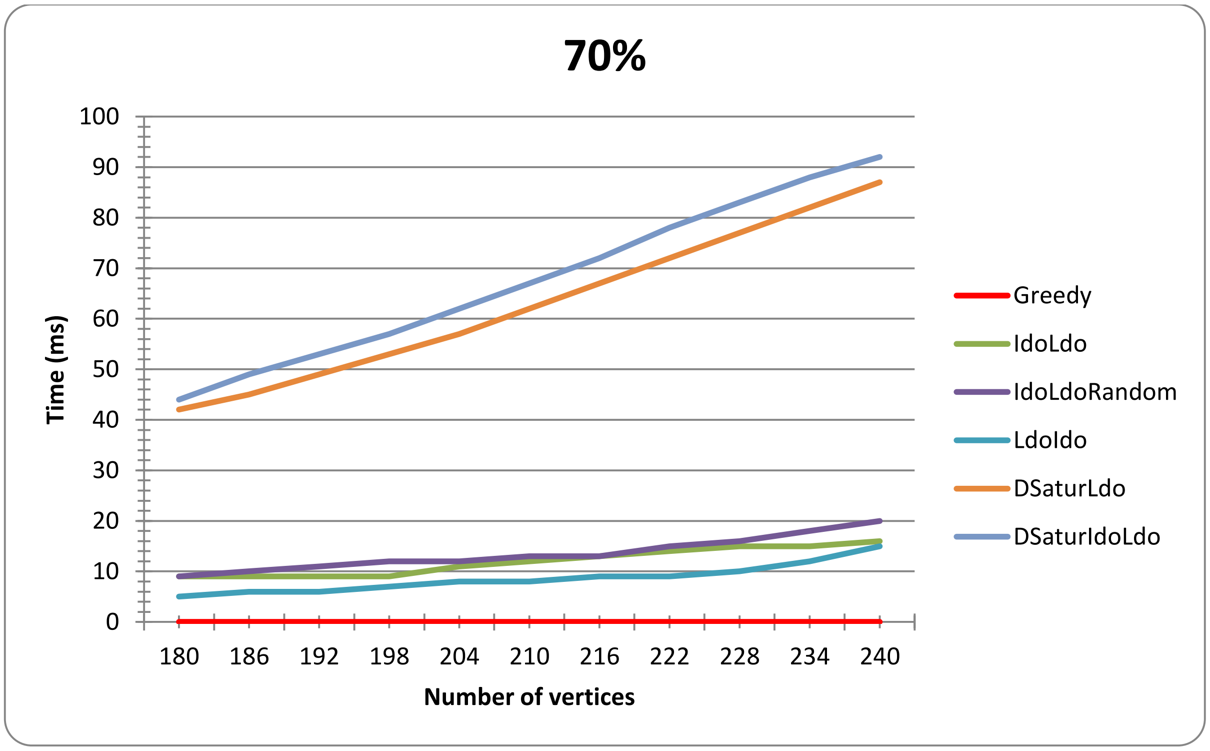

Figure 69. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 70%.

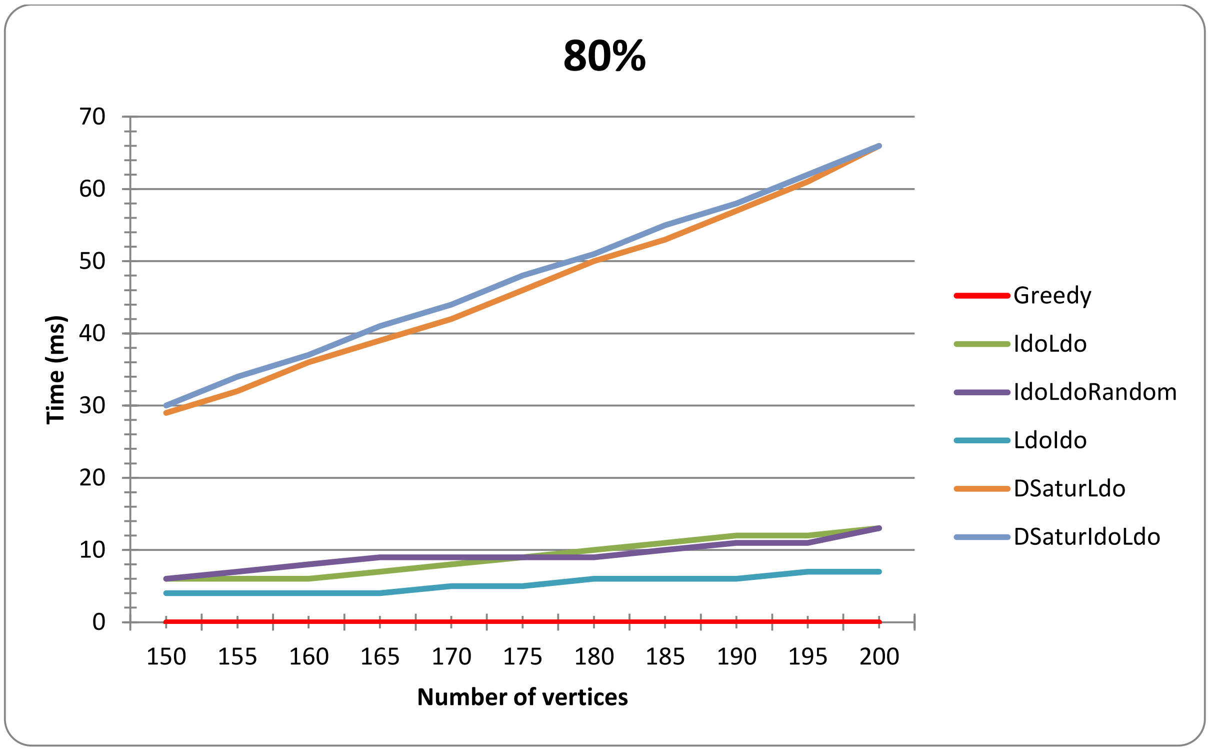

Figure 70. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 80%.

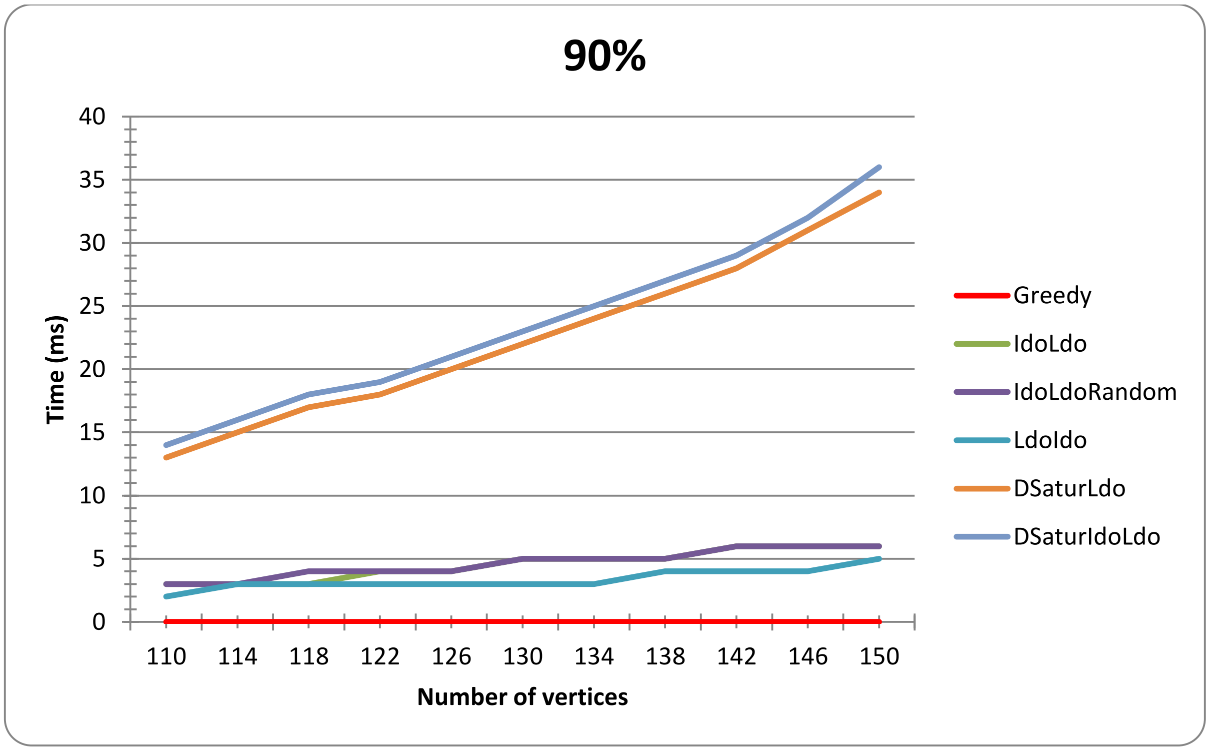

Figure 71. Randomly generated graphs tests' results compared in time (ms). Combined algorithms, density 90%.

Parallel algorithms

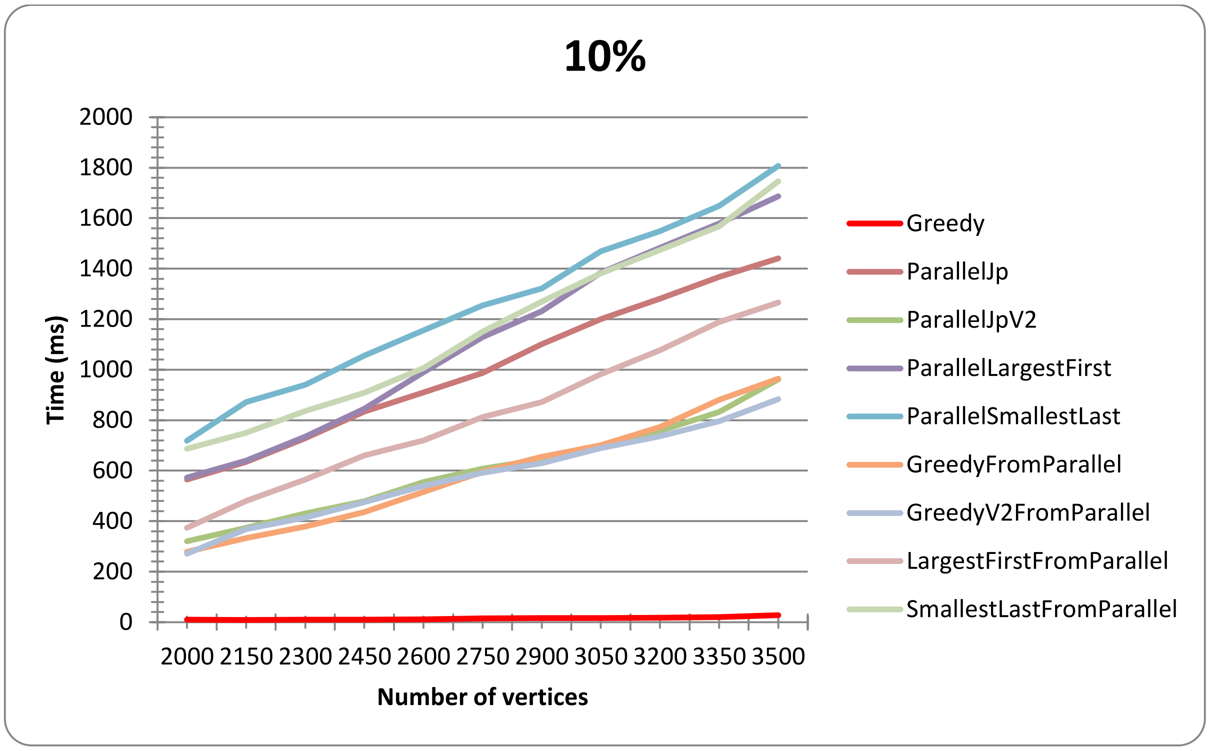

Figure 72. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 10%.

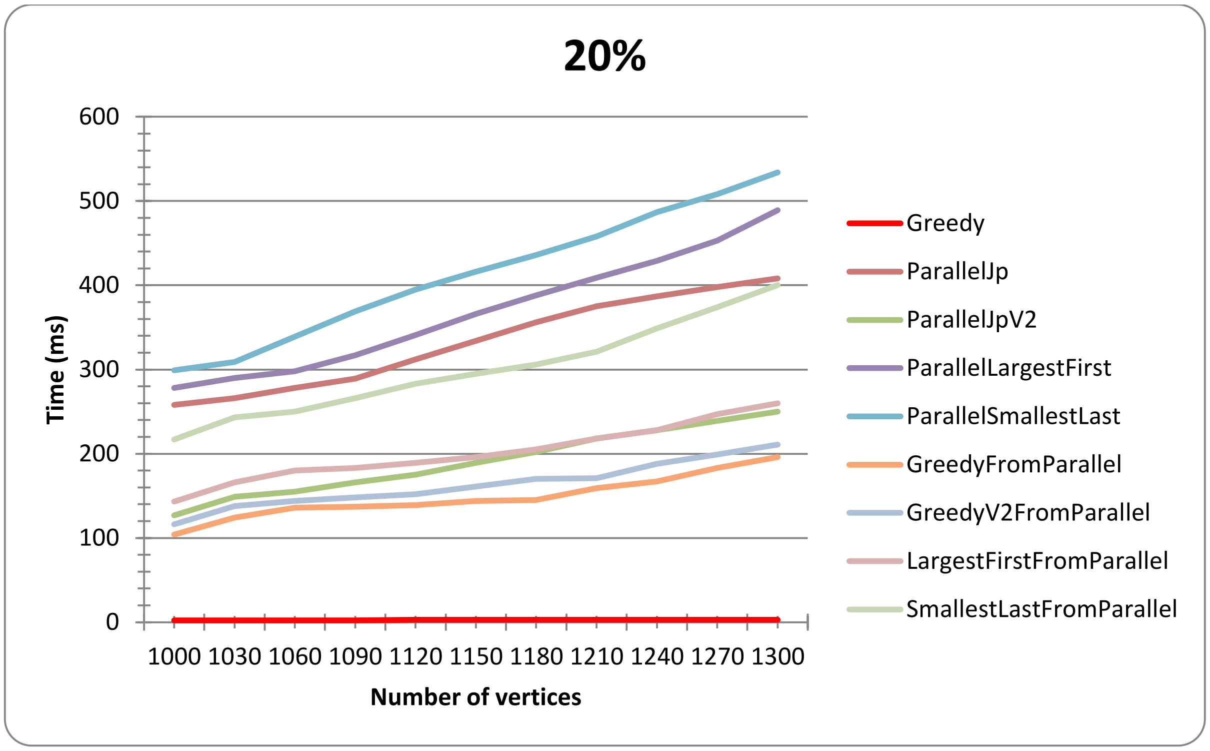

Figure 73. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 20%.

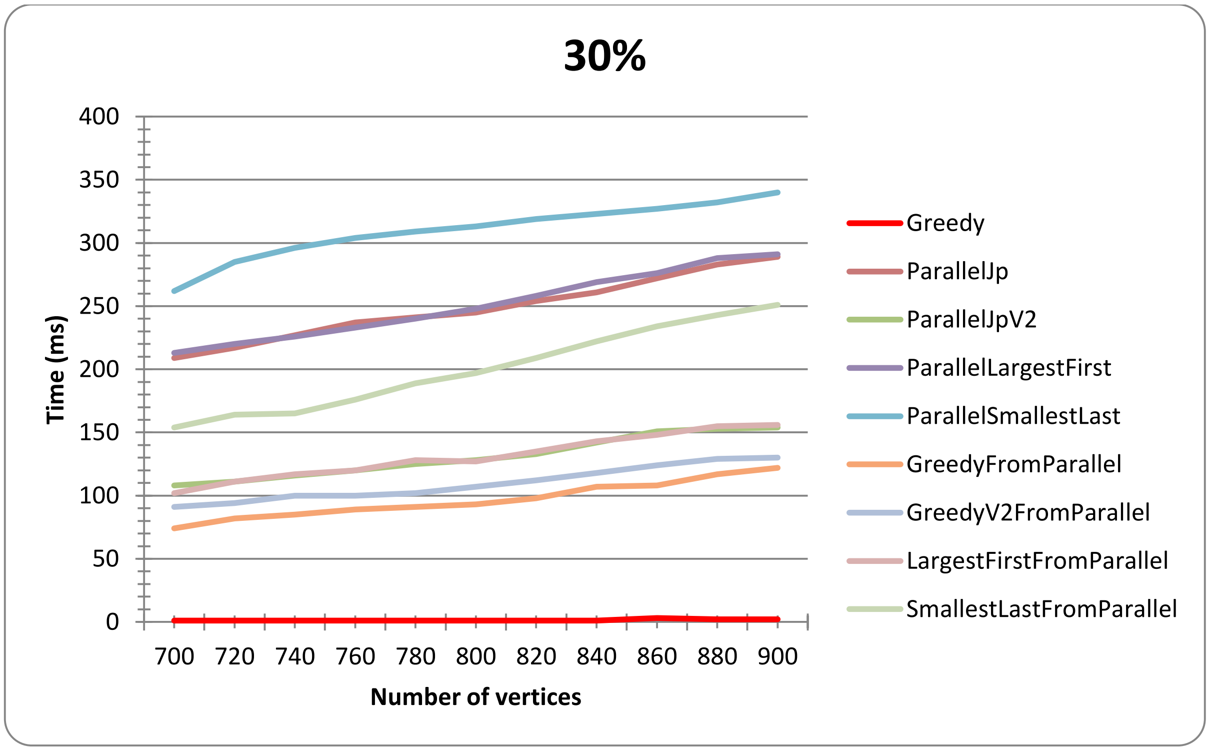

Figure 74. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 30%.

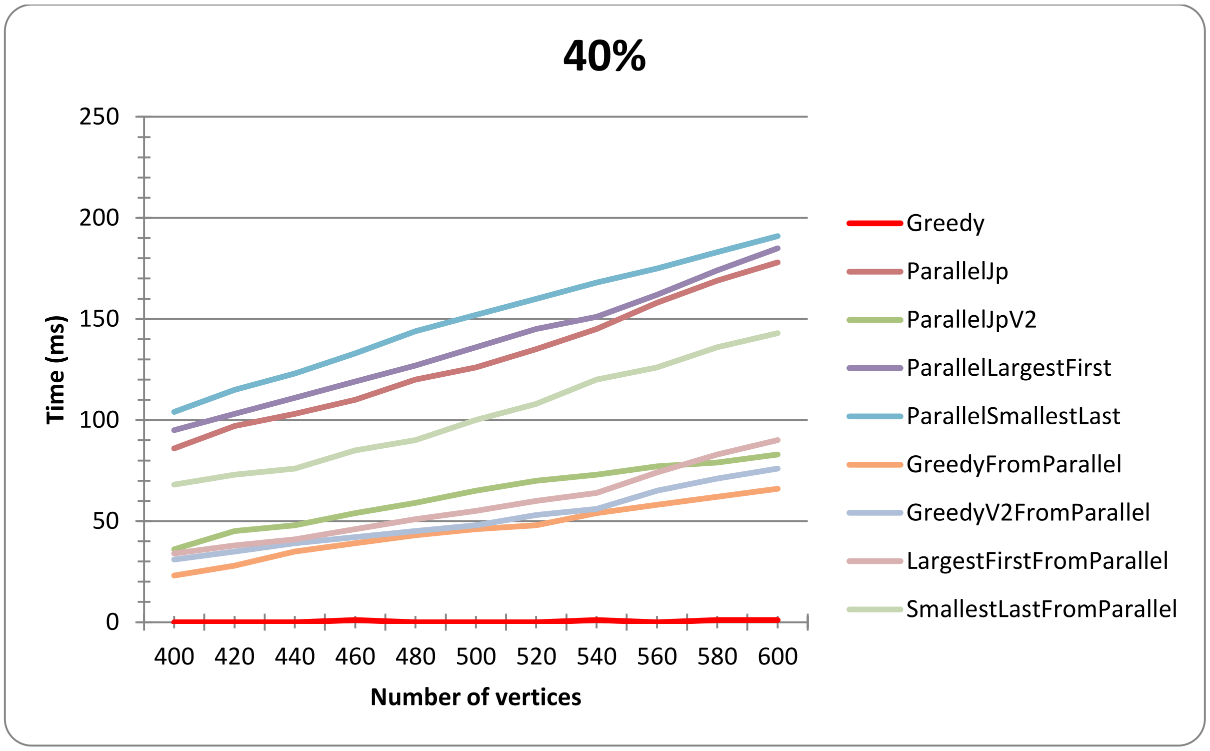

Figure 75. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 40%.

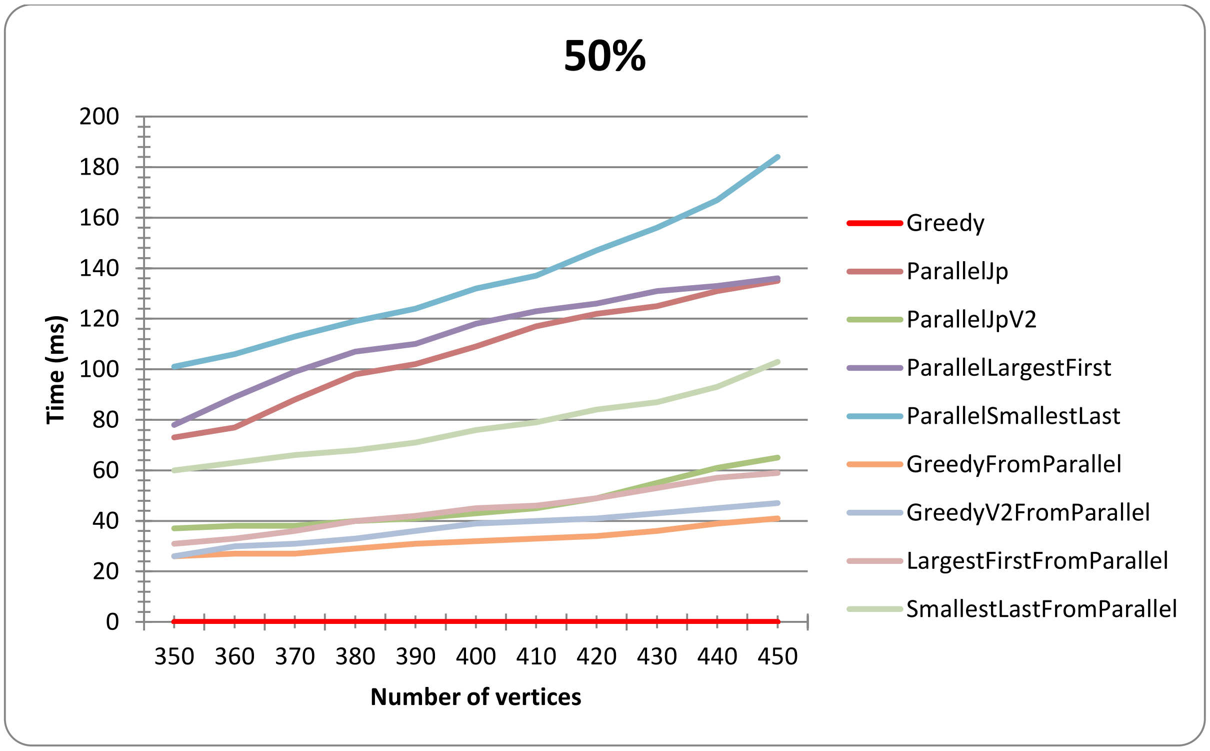

Figure 76. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 50%.

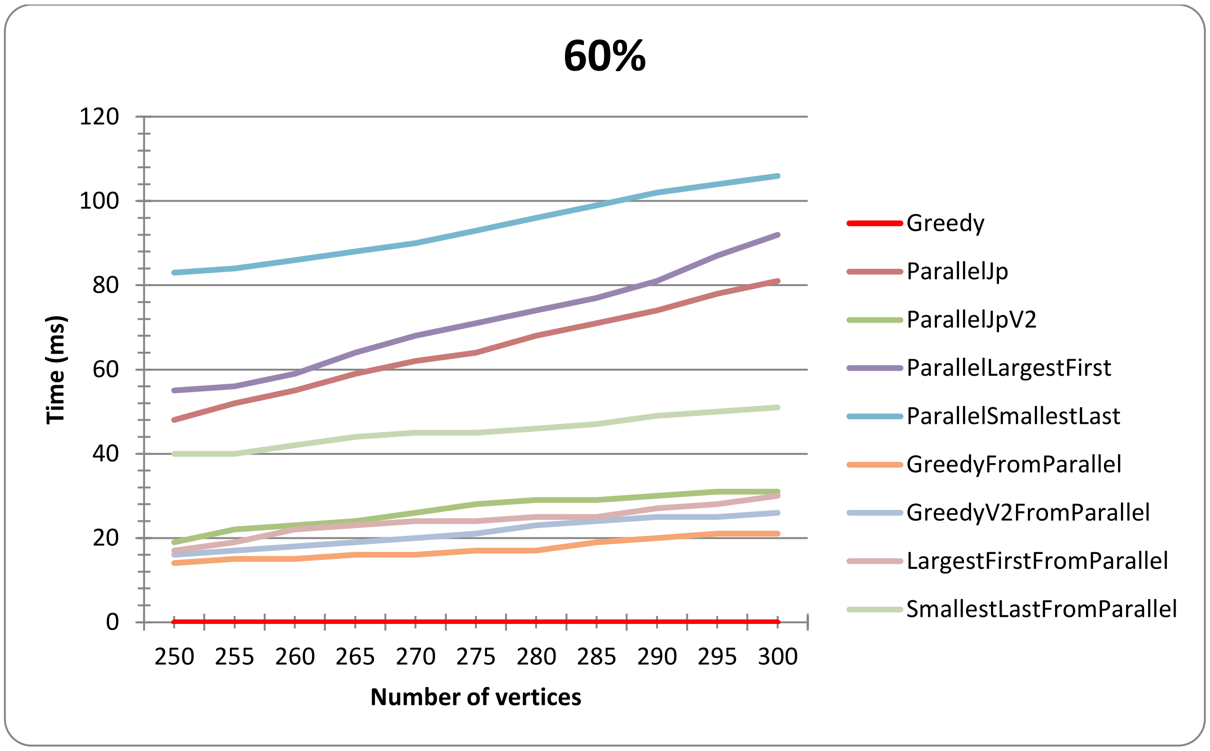

Figure 77. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 60%.

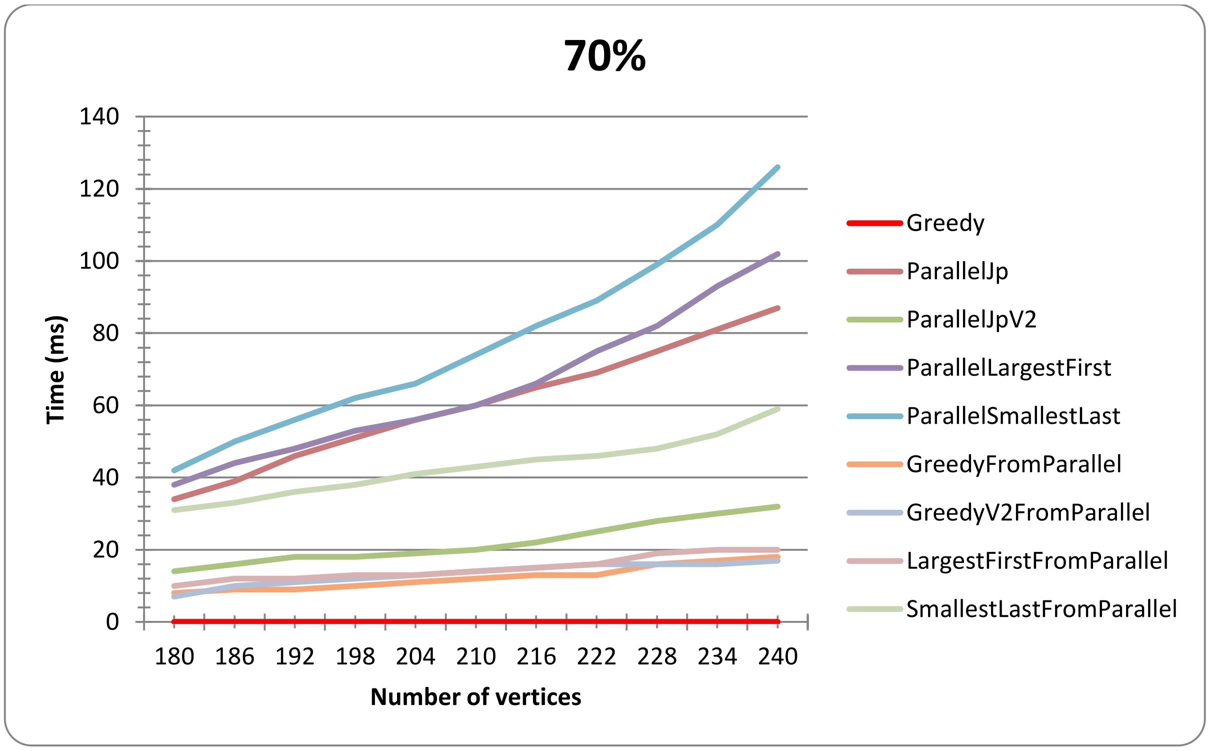

Figure 78. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 70%.

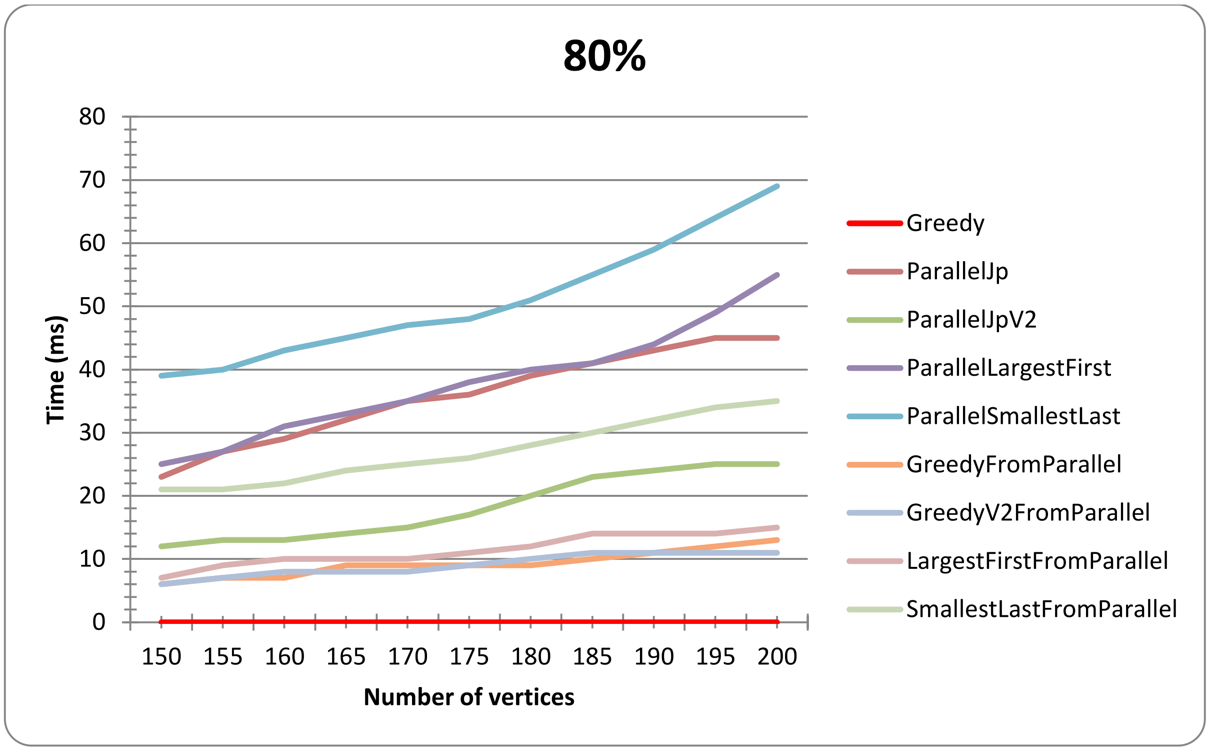

Figure 79. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 80%.

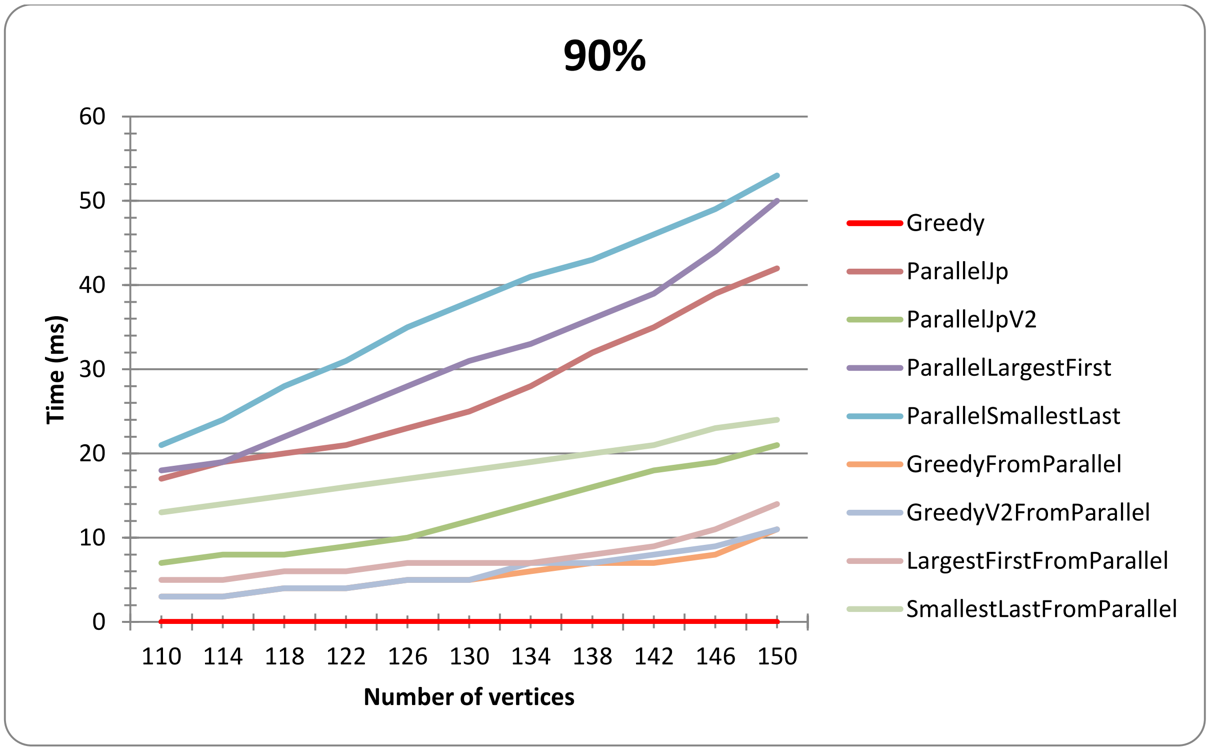

Figure 80. Randomly generated graphs tests' results compared in time (ms). Parallel algorithms, density 90%.

Appendix 2 – Maximum Clique algorithms: DIMACS Graphs Test Results

Graph	Size	Density	Number of color classes				
			Largest-First	Greedy	Largest-First V3	DSatur	DSatur V2
c-fat500-1.clq	500	0,36	14	14	14	14	14
c-fat500-2.clq	500	0,07	26	26	26	26	26
c-fat500-5.clq	500	0,19	64	64	64	64	64
c-fat500-10.clq	500	0,37	126	126	126	126	126
hamming10-2.clq	1024	0,99	512	512	512	512	512
hamming6-2.clq	64	0,9	32	32	32	32	32
hamming6-4.clq	64	0,35	8	8	8	8	8
hamming8-2.clq	256	0,97	128	128	128	128	128
hamming8-4.clq	256	0,64	32	32	32	24	24
johnson16-2-4.clq	120	0,76	14	14	14	17	17
johnson8-2-4.clq	28	0,56	6	6	6	6	6
johnson8-4-4.clq	70	0,77	20	20	20	17	17
keller4.clq	171	0,65	37	38	39	25	23
MANN_a27.clq	378	0,99	144	135	141	136	136
MANN_a9.clq	45	0,93	21	18	19	19	19
p_hat300-1.clq	300	0,24	22	29	23	21	21
p_hat300-2.clq	300	0,49	46	56	43	41	43
p_hat300-3.clq	300	0,74	73	85	71	69	70
p_hat500-1.clq	500	0,25	36	45	33	32	30
p_hat500-2.clq	500	0,5	68	87	68	65	65
p_hat700-1.clq	700	0,25	43	53	41	40	40
p_hat1000-1.clq	1000	0,25	56	69	55	53	53
san1000.clq	1000	0,5	15	47	29	25	26
san200_0.7_1.clq	200	0,7	44	49	43	38	38
san200_0.7_2.clq	200	0,7	18	35	24	23	20
san200_0.9_1.clq	200	0,9	75	92	78	75	76
san400_0.5_1.clq	400	0,5	13	29	23	19	19

Table 28. DIMACS graphs test results. Number of color classes - part 1.

Graph	Size	Density	Number of color classes				
			DSatur-LDO	**DSatur-IDO-LDO**	**LDO-IDO**	**Parallel Largest-First**	**Parallel Smallest-Last**
c-fat500-1.clq	500	0,36	14	14	14	18	17
c-fat500-2.clq	500	0,07	26	26	26	31	31
c-fat500-5.clq	500	0,19	64	64	64	74	74
c-fat500-10.clq	500	0,37	126	126	126	126	126
hamming10-2.clq	1024	0,99	512	512	512	541	547
hamming6-2.clq	64	0,9	32	32	32	34	35
hamming6-4.clq	64	0,35	7	7	8	9	11
hamming8-2.clq	256	0,97	128	128	128	139	134
hamming8-4.clq	256	0,64	24	16	32	31	29
johnson16-2-4.clq	120	0,76	14	14	14	15	18
johnson8-2-4.clq	28	0,56	6	6	6	6	6
johnson8-4-4.clq	70	0,77	17	14	20	22	22
keller4.clq	171	0,65	24	25	39	26	22
MANN_a27.clq	378	0,99	140	140	142	144	144
MANN_a9.clq	45	0,93	19	19	20	21	21
p_hat300-1.clq	300	0,24	22	22	22	24	24
p_hat300-2.clq	300	0,49	42	42	44	44	45
p_hat300-3.clq	300	0,74	69	70	69	73	77
p_hat500-1.clq	500	0,25	32	32	34	34	35
p_hat500-2.clq	500	0,5	66	66	68	69	70
p_hat700-1.clq	700	0,25	40	41	41	44	45
p_hat1000-1.clq	1000	0,25	52	52	54	57	58
san1000.clq	1000	0,5	24	26	28	15	15
san200_0.7_1.clq	200	0,7	42	35	43	38	48
san200_0.7_2.clq	200	0,7	23	23	22	18	18
san200_0.9_1.clq	200	0,9	73	70	77	77	86
san400_0.5_1.clq	400	0,5	21	20	22	13	13

Table 29. DIMACS graphs test results. Number of color classes - part 2.