THE SELECTIVE GRAPH COLORING PROBLEM

by

Ayberk Çalık

B.S., Industrial Engineering, Yıldız Technical University, 2008

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2013

ACKNOWLEDGEMENTS

I wish to thank, first and foremost, my thesis supervisor Assist. Prof. Tınaz Ekim Aşıcı for her valuable assistance in the preparation and completion of this study. Her endless support and assistance help me to hurdle all the obstacles in the completion of this research work.

I am indebted to Prof. Necati ARAS and Assoc. Prof. Fatih ALAGÖZ for their willingness to take part on my thesis committee and their valuable suggestions.

I am thankful to my friends Onur Alsaran, Mehmet Hakan Akyüz, Özgür Emre Sivrikaya, Burak Kocuk, Gökalp Erbeyoğlu and my cousin Ozan Atila for their endless support and help.

I owe my deepest gratitude to my parents for for their never ending love and support.

Last but not least, I would like to thank Ahu Akdemir for her endless support and patience. All of this would not be possible without her.

ABSTRACT

THE SELECTIVE GRAPH COLORING PROBLEM

In the classical graph coloring problem, the vertices of a given graph are colored such that no two adjacent vertices take the same color with the objective of coloring the whole graph with the minimum number of colors. The minimum number of colors that is needed to color a graph is called the chromatic number of the graph. It is known that the decision version of the problem is NP-complete and the optimization version of the problem is NP-hard. The selective graph coloring problem is a generalization of the classical graph coloring problem, where a graph and a partition of its vertices into different clusters are given as an input and the objective is assigning one color to one vertex in each cluster so that two adjacent vertices get different colors and the total number of colors is minimized. The selective graph coloring problem helps modeling and solving several real life problems. Rebuilding or repairing a cellular phone network after a disaster and allocating new frequencies to these transmitters and scheduling problems with selection among a fixed set of time windows for each task are examples of real life problems for which the selective graph coloring problem will be helpful. Such various applications and the intractable nature of the selective graph coloring problems made the usage of heuristic methods inevitable. The main subject of this thesis is developing new heuristic approaches and exact methods for the solution of the selective graph coloring problem. For this purpose, several heuristic approaches and exact methods have been tried on some graph classes and their efficiency is compared through experiments.

ÖZET

SEÇMELİ BOYAMA PROBLEMİ İÇİN SEZGİSEL YÖNTEMLER

Klasik çizge boyama problemi, verilen bir çizgenin köşelerinin iki komşu köşe aynı rengi almayacak şekilde en az renk kullanılarak boyanması problemidir. Bir çizgenin boyanabileceği en az sayıdaki renge, o çizgenin boyama sayısı denir. Klasik çizge boyama probleminin karar versiyonu NP-tam, optimizasyon versiyonu NP-zor'dur. Seçmeli boyama problemi, klasik boyama probleminin bir genellemesidir. Bu problemde, çizge ve bu çizgenin köşelerinin parçalanmış hali girdi olarak verilmektedir. Problemin amacı, iki komşu köşe aynı rengi almayacak şekilde her parçadan bir köşe seçilerek bu şekilde oluşan çizgeyi en az sayıda renkle boyamaktır. Seçmeli boyama problemi çeşitli uygulamaların çözümüne yardımcı olabilecektir. Doğal bir felaketten sonra yeni vericilere frekans atama probleminin çözülebilmesi için GSM ağının tamir edilmesi ya da yeniden kurulması ve her iş paketinin yapılabileceği belli zaman aralıklarının olduğu bir çizelgeleme probleminin çözülmesi bu uygulamalara örnektir. Sezgisel yöntemler, NP-zor problemler için hızlı çalışan ve en iyi sonuca yakın sonuçlar vermeyi amaçlayan yöntemlerdir. Bu nedenle seçmeli boyama probleminin çözümü için sezgisel yöntemlerin geliştirilmesi kaçınılmazdır. Tezin ana konusu seçmeli çizge boyama probleminin çözümü için yeni sezgisel ve tam çözüm yöntemleri geliştirilmesidir. Bu amaçla çeşitli sezgisel ve tam çözüm yöntemleri denenmiş ve deneylerle sonuçları incelenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C_n	A cycle on n vertices
$E(G)$	Edge set of graph G
G'	Subgraph of graph G
$\overline{G}$	Complement of graph G
H	Induced subgraph of graph G
k	Number of colors
$N(v)$	Neighborhood of vertex v
P_n	A path on n vertices
$V(G)$	Vertex set of graph G
V_i	Clusters i of the vertex set V of graph G
$\alpha(G)$	The maximum stable set number of graph G
$\omega(G)$	The maximum clique number of graph G
$\omega_{SEL}(G)$	The maximum selective clique number of graph G
$\theta(G)$	The clique cover number of graph G
$\chi(G)$	The chromatic number of graph G
$\chi_{SEL}(G)$	The selective chromatic number of graph G

LIST OF ACRONYMS/ABBREVIATIONS

DSATUR	Degree of saturation
DSEL-COL	Decision version of the Selective Graph Coloring Problem
onestepCD	One Step Color Degree Algorithm
SEL-COL	Optimization version of the Selective Graph Coloring Problem
SelImpCol	Selective Improper Coloring Algorithm
SelParColDeg	Cluster Degree Based Selective Partial Coloring Algorithm
SelParColNum	Cluster Number Based Selective Partial Coloring Algorithm
TS-PCP	Tabu Search for Partition Coloring Problem

1. INTRODUCTION

Graph coloring problem can be defined as assigning colors to vertices such that each vertex is colored by only one color and two adjacent vertices, which are connected by an edge, have different colors. Some problems such as scheduling classrooms or airplanes, machines or assigning personnel to jobs, or designing circuits can be modeled as graphs and solved by graph coloring algorithms [1].

Graph coloring is a computationally hard problem. If we want to check if a given graph admits k-coloring for a given k such that $k \geq 3$, it is NP-complete. The chromatic number of a graph G, denoted by $\chi(G)$, is the minimum number of colors required to color the vertices of G in such a way that adjacent vertices receive different colors, or equivalently, the minimum number of stable sets needed to cover the vertices of G. It is NP-hard to compute $\chi(G)$ of a given graph G. The 3-coloring problem remains NP-complete even on planar graphs of degree 4 [2].

Assume that a department of a university tries to use the minimum number of classrooms to make five exams that have certain start and end times in a day. One can use classical graph coloring to solve this problem. This example is given in Figure 1.1.

In Figure 1.1, start and end times of five exams are given and the problem is shown as a graph. Vertices of the graph represent time intervals of exams and two vertices are connected with an edge if and only if two time intervals have an intersection. The objective of the problem is to find the minimum number of colors to color the graph. In other words, one tries to find the minimum number of classrooms to make all five exams without any conflict, the solution of the problem is also given in Figure 1.1.

We consider a generalization of the usual graph coloring problem, called selective graph coloring. In this problem, we are given a graph G and a partitioning of its vertex set V into p clusters V_1, V_2, ..., V_p and the problem consists of selecting one vertex from each cluster and associating colors to these selected vertices so that two

Exam	Start Time	End Time
1	10:00	12:00
2	11:00	13:00
3	14:00	16:00
4	12:00	14:00
5	11:00	13:00

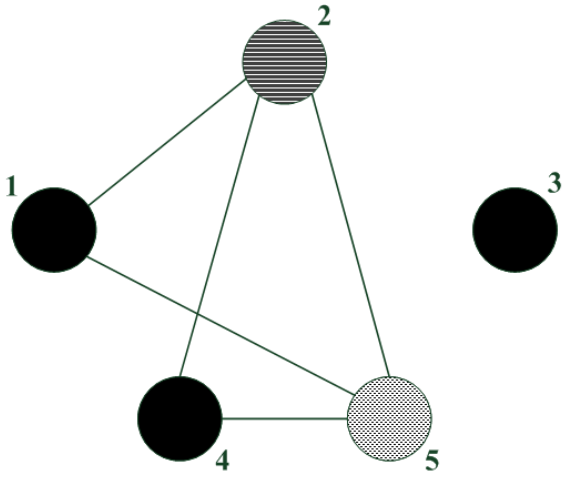

Figure 1.1. A coloring example of a graph.

adjacent vertices get different colors and the total number of colors used is minimized. In other words, firstly one has to select one vertex from each cluster and secondly find a minimum coloring of the resulting sub-instance. A selection V' is defined as $|V' \cap V_i| = 1$ for all $i \in \{1, ..., p\}$ and the coloring of V' is a selective coloring of a given graph and the minimum number of colors required to selectively color the graph is named the selective chromatic number which is denoted by $\chi_{SEL}(G)$.

In Figure 1.2, there is a selective coloring example of a clustered graph. One vertex is selected randomly from each cluster, and these vertices are colored. In selective coloring problem, vertices should be selected to ensure the minimum chromatic number among all possible selections. Therefore, selective coloring problem is much more complex than the usual graph coloring problem. The selective coloring in Figure 1.2 is not optimal because if we select vertices b, c, f, g, they can be colored by one color.

The aim of this study is to generalize classical graph coloring problem to the selective framework in order to satisfy the needs of the real life problems. Then, several methods will be developed to solve the selective general coloring problem in an efficient way.

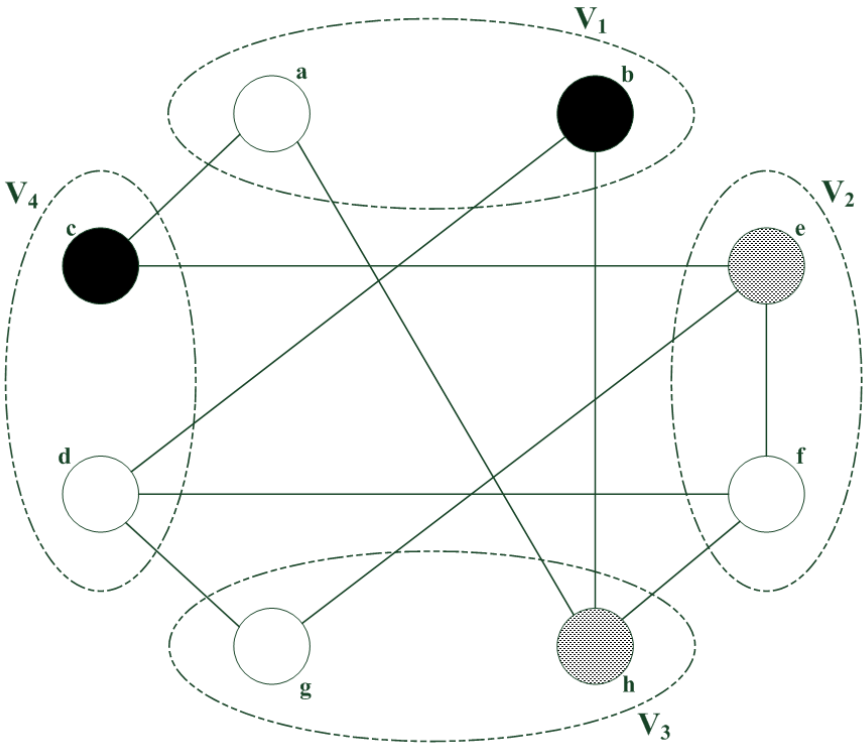

Figure 1.2. A selective coloring example(V_i $\forall i$ indicates partitions).

Two main applications are the motivation of the study of the selective graph coloring problem. First one is in disk graphs, which deals with rebuilding or repairing a cellular phone network (i.e. locating transmitters) after a disaster so that the frequency allocation problem to the new transmitters can be made using as few frequencies as possible. The second one is in interval graphs and it deals with a scheduling problem with a selection among a fixed set of time windows for each task.

The second application is as follows. Assume that a mobile company decided to design a new mobile network to a settlement which the company will start to give service. They have k transmitters that have equal specifications and for each transmitter there are n possible locations available for establishment. The objective of the company is to use the minimum number of frequencies to construct a more cost friendly network. In this problem, there will be k clusters one for each transmitter and each cluster has n vertices for each possible location for that transmitter. An edge is formed between vertices if they have intersecting coverage areas when we establish transmitters on these vertices (locations). The objective of the problem is to choose a location

for each transmitter so that the number of frequencies that are required is minimized.

Assume that a manager is in charge of a company and one of the management positions is open. There are k candidates and every candidate has his own spare time. Since the vacant position is a high level management position, most of the candidates are working for other companies. Therefore, they have limited suitable time intervals for interview. The manager wants to make the job interview himself, thus two candidates cannot make a job interview with the manager at the same time. The objective is to determine whether all candidates can be interviewed or not.

In this example, classical graph coloring is not sufficient and selective graph coloring is needed. Each candidate forms a cluster and each free time of the candidate forms vertices of this cluster. An edge is formed between intersecting intervals. It is possible to make an interview with all candidates if and only if there is an induced subgraph of the clustered graph, which contains exactly one time interval (vertex) for each candidate(cluster) so that none of these intervals (vertices) have a conflict(edge) between them.

The selective graph coloring problem consists of two main phases: selection and coloring. The main difference between selective graph coloring and graph coloring is the selection phase which makes it far more difficult as we will observe in Section 2.2.

The wavelength routing and assignment problem is another application of the selective graph coloring problem. In this problem, an optical network is considered. The network is composed of links that are optical fibers capable of carrying a specified number of wavelengths. For this network, we have a set of desired source-destination connections. The objective of the problem is to find a route and assign a wavelength to each connection such that if two connections share a link, a different wavelength must be assigned to corresponding connections. If the routes are given as input to the problem, the wavelength assignment problem can be easily transformed into the standard vertex coloring problem by representing connections with vertices and put an edge between them if they have a common link. In this problem, the task will be

the assignment of colors to vertices such that no adjacent vertices take the same color which is equivalent to the graph coloring problem [3].

If the routes are given as an input to the problem but each source-destination pair has more than one route, this problem cannot be represented as a standard vertex coloring problem. Selective graph coloring is important for these kinds of problems. One should choose exactly one route from all possible routes for each source-destination pair and then assign a wavelength to that route with the objective of using the minimum number of wavelengths.

The wavelength routing and assignment problem is defined as follows. We are given an undirected network N, $C = \{(s_1, d_1), ..., (s_k, d_k) : s_i, d_i \in N\}$ is a collection of source-destination pairs and $\Lambda = \{\lambda_1, ..., \lambda_m\}$ is a collection of wavelengths. Each vertex in the network represents a node and each edge a link. Every link is capable of carrying any subset of the wavelengths simultaneously and each network node is capable of directing a wavelength from any incoming link to any outgoing link by keeping the same wavelength. For each source-destination pair (s, d), let $\mathcal{P}_{s,d}$ denote the set of all paths between s and d in N. We are interested in the case in which a fixed set of paths $P_{s,d} \subset \mathcal{P}_{s,d}$ is given as input to the problem. Assignment consists of allocating one wavelength and one path to each source-destination pair. We seek an assignment such that the cardinality of Λ is as small as possible [3].

Selective coloring have beeen studied in the literature from several point of views. Selective coloring is named partition coloring in the literature. Li and Simha [3] proposed two groups of approximation algorithms for partition coloring problem. These groups are one step algorithms and two step algorithms and each group has 3 different algorithms. They compared 6 algorithms and found that onestepCD (One Step Color Degree) algorithm is the best one.

Noronha and Ribeiro [4] proposed a tabu search algorithm to deal with partition coloring problem. They used onestepCD algorithm to obtain an initial solution and then their tabu algorithm tries to find a better upper bound for the problem. For the

largest graphs with 900 nodes, the number of colors used by their tabu search heuristic is approximately 80% of the number of colors that used by the onestepCD algorithm.

Ribeiro *et al.* [5] proposed a branch and cut algorithm for partition coloring. They used tabu search heuristic [4] to obtain an initial solution, it gives primal bounds. They proposed an integer programming formulation and applied branch and cut algorithm. Moreover, they gave their results about known graph coloring instances in their study.

Hoshino *et al.* [6] proposed a branch and price approach for the partition coloring problem. They created a new model to formulate the problem and used linear relaxation. They made experiments with random graphs and instances originating from routing and wavelength assignment problems. They solved an instance with 706 vertices and 101.600 edges and this instance could not be handled by Ribeiro *et al.* [5].

There are also some other combinatorial optimization problems which are extended by the selective framework of the graph coloring problem, for instance the traveling salesman problem (TSP). This problem is known as Group-TSP or One-of-a-set TSP. In this problem, again we have a salesman that needs to visit n customers $c_1, ..., c_n$. Each customer c_i, $i \in \{1, ..., n\}$, specifies location $l_1(i), ..., l_{n_i}(i)$ in which he/she is willing to meet the salesman. The objective of the problem is to find a tour of minimum length such that the salesman will meet each one of the customers once and will meet them in one of their specified locations. The classical TSP problem will be obtained if each customer specifies only one location [7].

All algorithms were implemented in C#. CPLEX 11.2 was used as an optimization software package for exact algorithms. All experiments were performed on an Intel Core i7 machine with a 2.20-GHz clock speed and eight Gbyte of RAM memory.

In Chapter 2, preliminaries on graph coloring problem is given. It starts with basic graph theoretical definitions which will be used in the thesis. Then, we continue with the complexity of the problem and 2-SAT reduction for a special case. The last subject of the chapter is partitioned graph generators which will be used for our computational

studies. In Chapter 3, exact algorithms for the selective graph coloring problem are explained in detail. Firstly, different IP formulations of the problem are given and then exact algorithms for interval and disk graphs are studied. We developed a very fast exact solution method for selective coloring in interval graphs which model several scheduling problems and an exact selective clique algorithm for small instances in disk graphs. New heuristic approaches to the selective graph coloring problem are given in Chapter 4. Construction heuristics and different tabu algorithms for the selective graph coloring problem are the main subjects of this chapter. We developed tabu algorithms for the selective graph coloring problem that find better results than those found in the literature to the best of our knowledge. Finally, Chapter 5 concludes the thesis.

2. PRELIMINARIES ON GRAPH COLORING PROBLEM

2.1. Graph Theoretical Definitions

A graph G is a finite nonempty set of vertices V and edges E. In Figure 2.1, each vertex is represented by a circle and each edge is represented by a line segment. Edges are named by their start and end points. For instance, an edge between vertices u and v is named uv. We say u is adjacent to v if there is an edge (uv) between vertices u and v.

In a given graph $G = (V, E)$, the number of vertices in V is called the order of G and the number of edges in E is called the size of G. The neighborhood of a vertex v is the set of all adjacent vertices to v and denoted by $N(v)$. The number of edges incident with a vertex v is called the *degree* of v and is denoted by *deg* v.

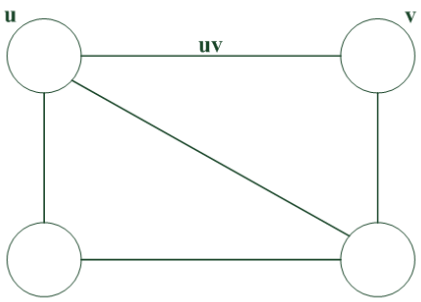

Figure 2.1. An example of a graph representation.

A stable set or an independent set is a set of vertices in a graph, no two of which are adjacent. A maximal stable set is a stable set such that adding any other vertex to the set forces the set to contain an edge.

A clique in an undirected graph $G = (V, E)$ is a subset of the vertex set $C \subseteq V$, such that for every two vertices in C, there exists an edge connecting the two. This is equivalent to saying that the subgraph induced by C is complete. A maximal clique is a clique that cannot be extended by including one more vertex, that is, a clique which is not included in the vertex set of a larger clique. The clique number

of graph G, denoted $\omega(G)$, is the size of the largest complete subgraph of G.

The chromatic number of graph G, denoted $\chi(G)$, is the fewest number of colors needed to color the vertices of G in such a way that adjacent vertices receive different colors, or equivalently, the fewest number of stable sets needed to cover the vertices of G.

The stability number of graph G, denoted $\alpha(G)$, is the size of the largest stable set of G.

The clique cover number of G, denoted $\theta(G)$, is the fewest number of complete subgraphs needed to cover the vertices of G.

G' is a subgraph of G, if vertices and edges of G' form subsets of the vertices and edges of G. H is an induced subgraph of G if it has exactly the edges that appear in G over the same vertex set.

Let $G(V, E)$ be an undirected graph and $G[A]$ is an induced subgraph of $G(V, E)$. If a graph fulfills the following conditions, then the graph is called perfect.

$$\omega(G[A]) = \chi(G[A])(\forall\, A \subseteq V) \tag{2.1}$$

$$\alpha(G[A]) = \theta(G[A])(\forall\, A \subseteq V) \tag{2.2}$$

It is clear by duality that a graph G satisfies Equation 2.1 if and only if its complement $\bar{G}$ satisfies Equation 2.2.

In other words, a graph is perfect if for all of its induced subgraphs, the chromatic

number($\chi(G)$) and the clique number($\omega(G)$) are equal.

A famous result of graph theory is The Perfect Graph Theorem, which says that; a graph is perfect if and only if its complement is perfect [8] [9]. If we remove all edges of G that are in E and add edges that are not in E we obtain the complement of graph G. In Figure 2.2 and Figure 2.3, there is an example of a graph and its complement. Both of them are perfect since they do not contain any odd cycle of length larger than 4 [10]. We could just as easily say, if a graph is perfect then its complement is perfect, since the complement of the complement of G is also G.

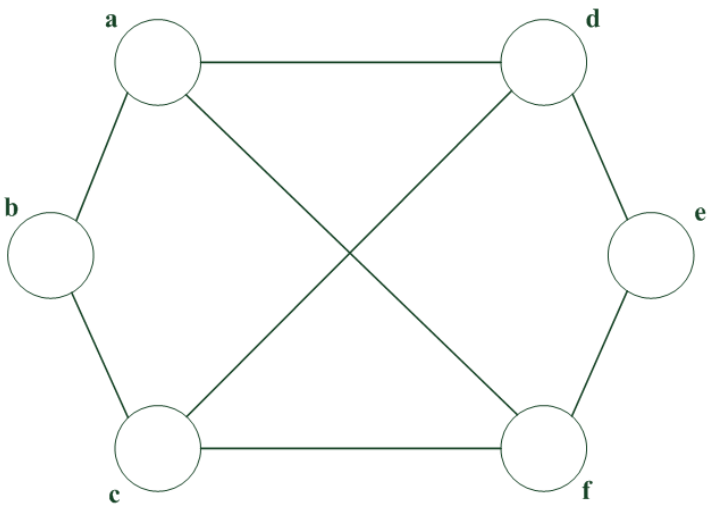

Figure 2.2. A perfect graph.

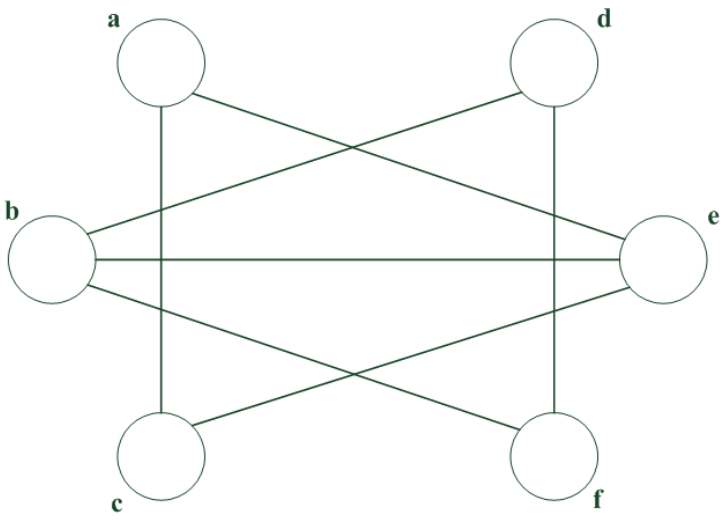

Figure 2.3. The complement of the previous graph.

An undirected graph G is called an interval graph if its vertices can be put into one-to-one correspondence with a set of intervals f of a linearly ordered set such that two vertices are connected by an edge of G if and only if their corresponding intervals have nonempty intersection. We call f an interval representation for G. An interval graph and its interval representation are given in Figure 2.4.

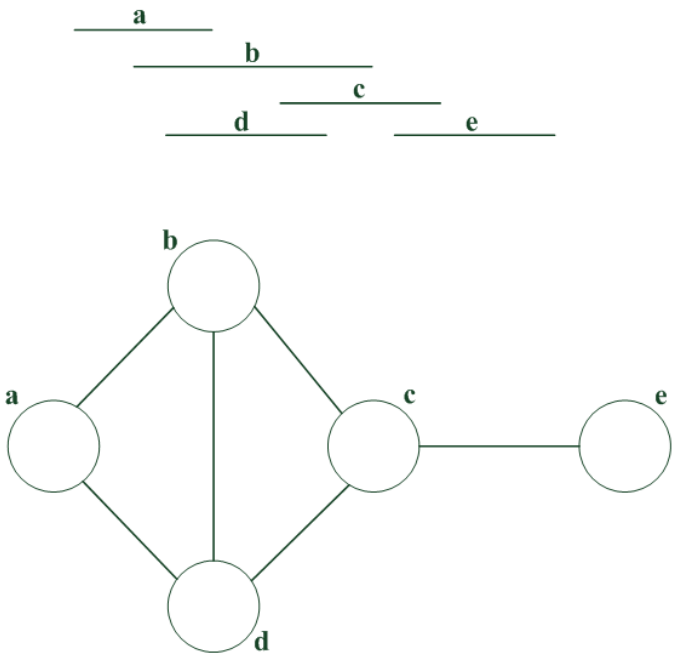

Figure 2.4. An interval graph and its interval representation.

Unit interval graphs are interval graphs that have an interval representation in which each interval has unit length. Studying unit interval graphs is simpler than the interval graphs because all intervals have the same length.

A disk graph is the intersection graph of unit circles in Euclidean plane. Each circle is represented by a vertex and if two circles intersect in Euclidean plane, two representing vertices have an edge between them. A disk graph and its representation are given in Figure 2.5. Unit disk graphs are a special class of disk graphs in which all circles in the graph has equal radius length.

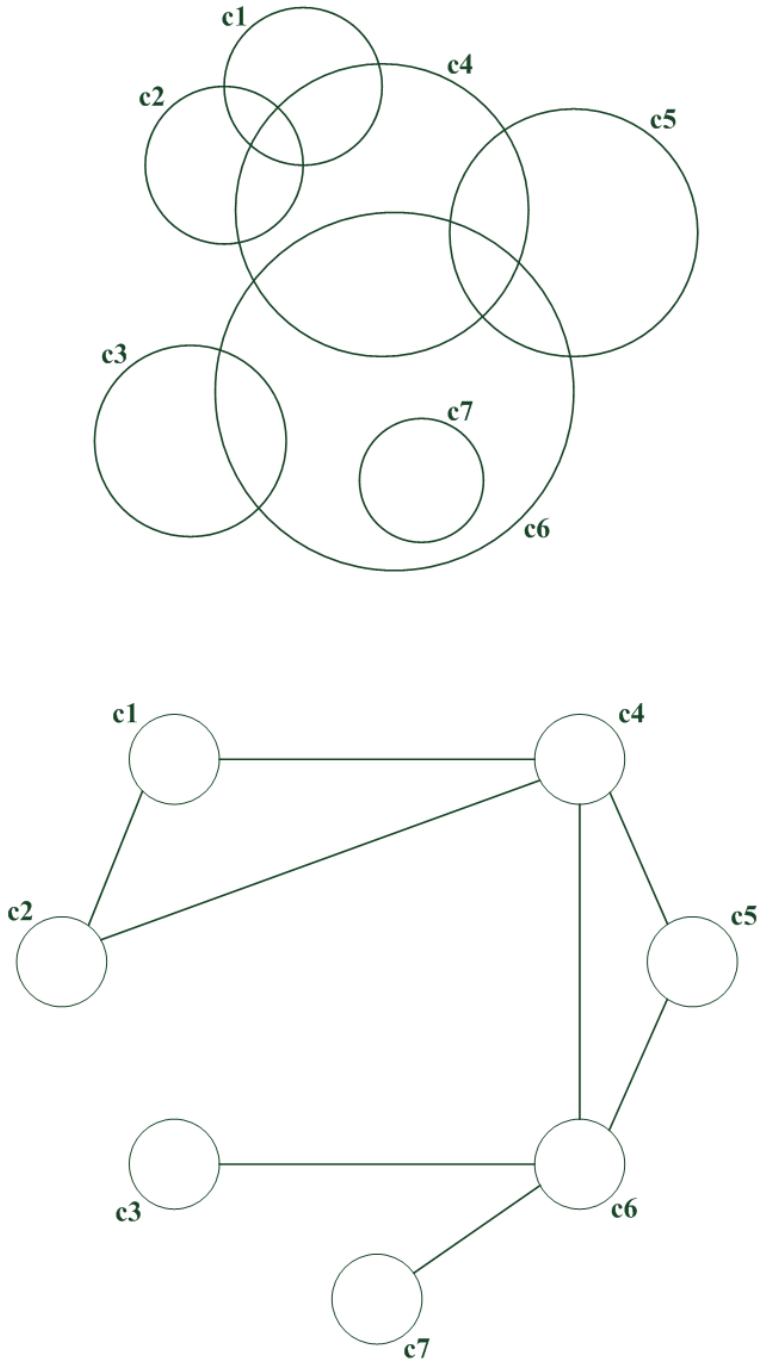

Figure 2.5. A disk graph and its representation.

2.2. Complexity of the Selective Graph Coloring Problem

Consider an undirected graph $G = (V, E)$ and a partition $V_1, ..., V_p$ of its vertex set V. For some integer $k \geq 1$, the selective graph coloring problem consists of finding a subset $V^* \subseteq V$ such that $|V^* \cap V_i| = 1$ for all $i \in \{1, ..., p\}$ and the graph induced by V^* is k-colorable [7].

Here we expect that the selective graph coloring problem is hard for most classes of graphs; discovering some easy cases is an important part of such a complexity study: only a detailed description of easy and hard cases allows to have a good and relevant understanding of the real difficulty of the problem and, as a consequence, of the real scientific challenges associated with it.

Since the classical graph coloring problem is a special case of selective coloring problem when $|V_i| = 1$ for all $i \in \{1, ..., p\}$, selective coloring problem is at least as hard as classical graph coloring problem. It is known that classical graph coloring problem is NP-hard. Decision version of the problem is NP-complete and it can be proved by reduction from 3-SAT [2]. In other words, there is no polynomial time algorithm to solve both the problems unless $P = NP$. Therefore, we can say that the selective graph coloring problem is NP-hard in general. Selective coloring problem has two equivalent versions as every combinatorial problem; optimization and decision. The decision version of the selective graph coloring problem is denoted by DSEL-COL and the optimization version of the selective graph coloring is denoted by SEL-COL. Determining the selective chromatic number of a clustered graph is an optimization problem and deciding whether a clustered graph can be colored by k colors is a decision problem.

The selective graph coloring problem is proved to be NP-complete [3].

Proposition 2.1. *The selective graph coloring problem is NP-Complete.*

Proof. Suppose that we are dealing with a standard graph coloring instance and we

know that the chromatic number of that instance is K. If we seperate this graph to clusters (each cluster has one vertex), add $c-1$ more nodes to each cluster and connect every newly added vertex to all other vertices, this new instance can be colored by K colors (optimum value) if and only if standard graph vertices are selected. □

Theorem 2.1 and Theorem 2.2 show that decision version of selective coloring problem is NP-complete for even easily constructed graphs.

Theorem 2.1. *[7] DSEL-COL is NP-complete for the disjoint union of C_4's even if the partition $V_1, ..., V_p$ satisfies $|V_i| = 3$ for all $i \in \{1, ..., p\}$.*

Theorem 2.2. *[7] DSEL-COL is NP-complete for the disjoint union of P_3's even if the partition $V_1, ..., V_p$ satisfies $2 \leq |V_i| \leq 3$ for all $i \in \{1, ..., p\}$.*

As a consequence of even easy cases are NP-complete for the selective graph coloring problem, we tried to find exact solutions for only graph classes that has special properties. These studies are given in Chapter 3.

2.3. 2-SAT Reduction for a Special Case

To determine the complexity of the problem, some special cases for the selective graph coloring problem should be studied. For that purpose [7] fixed $|V_i| = 2$ for all $i \in \{1, ..., p\}$ and showed that this special case can be reduced to 2-SAT problem. 2-SAT is a special case of the classical satisfiability problem where every clause has exactly 2 variables. It admits an exact solution in polynomial time.

When all clusters in a graph have two vertices, the problem of 1-selective colorability is not NP-complete. To this purpose, having a graph G which has a partition into p clusters $(V_1...V_p)$ and $\forall\, i = 1, ..., p\ |V_i| = 2$ we want to know whether there is any sub-graph $V^* \subseteq V$ such that $|V^* \cap V_i| = 1$ for all $i \in \{1, ..., p\}$ and $\chi(G[V^*]) = 1$.

Proposition 2.2. *1-DSEL-COL with $|V_i| = 2\ \forall i = 1, ..., p$ is polynomial time solvable.*

Proof. Construct a 2SAT instance as follows:

Case 1: All vertices in the same cluster (We create clauses to ensure that we select one or more vertices from each cluster):

$$V_1 = \{v_1, v_2\} \longrightarrow (v_1 \bigvee v_2)$$

. . .

$$V_k = \{v_{2k-1}, v_{2k}\} \longrightarrow (v_{2k-1} \bigvee v_{2k})$$

Case 2: All negations of vertices in the same cluster (We create clauses to ensure that we select one vertex or none from each cluster):

$$V_1 = \{v_1, v_2\} \longrightarrow (\bar{v}_1 \bigvee \bar{v}_2)$$

. . .

$$V_k = \{v_{2k-1}, v_{2k}\} \longrightarrow (\bar{v}_{2k-1} \bigvee \bar{v}_{2k})$$

Case 3: All edges between vertices in different partitions (We create clauses to ensure that we do not select two connected vertices from different clusters):

$$\{v_2, v_3\} \longrightarrow (\bar{v}_2 \bigvee \bar{v}_3)$$

. . .

$$\{v_a, v_b\} \longrightarrow (\bar{v}_a \bigvee \bar{v}_b)$$

Now we claim that there is a truth assignment for 2-SAT instance if and only if $\chi_s(G) = 1$.

If there is a truth assignment, we can say that the vertices whose related variables are assigned true form a selection V' such that $|V' \cap V_i|=1$ $\forall i$ and moreover, they can be colored by one color. In other words, selective chromatic number of the graph is 1.

Now, let V^* be the vertices that give the selective chromatic number of the problem.

$\forall v_i \in V^*$ set in 2-SAT instance: $v_i = 1$ and 0 otherwise. This is clearly a truth assignment.

□

2.4. Partitioned Graph Generators

We need to use computer programs in order to generate different partitioned graph instances. Thus, we can test new heuristic approaches on these generated graphs and compare them with each other. For this purpose, we implemented three different partitioned graph generators according to graph classes.

2.4.1. Partitioned General Graph Generator

Partitioned general graph generator algorithm takes *number of vertices*, *number of clusters*, *density*, *equal size of clusters* as an input. The algorithm creates vertices, clusters and edges between vertices with a given probability. In other words, if desired graph density is 0.5, then the probability of an edge to exist in a graph is 0.5. Therefore, we can obtain a partitioned graph with a desired density. User can customize the partitioned graph with changing the values of *equal size of clusters*. If *equal size of clusters* variable takes "True" value, clusters of the generated partitioned graph will have equal *number of vertices*. Otherwise, clusters will have different number of vertices.

2.4.2. Partitioned Interval Graph Generator

Interval graphs model one of the main applications that we concentrate on. Therefore, we studied on an algorithm and a code that generates partitioned interval graph instances for the partitioned graph coloring problem too. In partitioned general graph generator, we can obtain a graph with a desired density easily, but in partitioned interval graph generator it is harder as we will observe in what follows.

Partitioned interval graph generator does not directly create edges between vertices with a fixed probability. The algorithm creates an interval for each vertex and edges between these vertices are created depending on their intersections. If interval of vertex v and interval of vertex u has an intersecting part, the algorithm adds an edge between them.

Partitioned interval graph generator algorithm takes *number of vertices*, *number of clusters*, *interval line length*, *line length*, *minimum interval length* and *maximum interval length* as an input. We should determine *line length* to create all intervals within that length. *minimum interval length* and *maximum interval length* are lower and upper bounds of each interval, and they should be determined to obtain desired density.

Moreover, there are decision variables that can only take true or false values in the algorithm that helps to customize the partitioned interval graph which will be generated. These decision variables are *equal interval length* and *allow conflicts* (i.e. edges) in clusters. When we give true value to equal interval length variable, the code will fix the length of all intervals to 1. When we give false value to *allow conflicts* variable, clusters will not have any edges among them. We want to investigate different cases, for example every cluster has the same number of vertices. Therefore, interval generator is designed to provide customization on the input.

Partitioned interval graph generation example is shown in Figure 2.6.

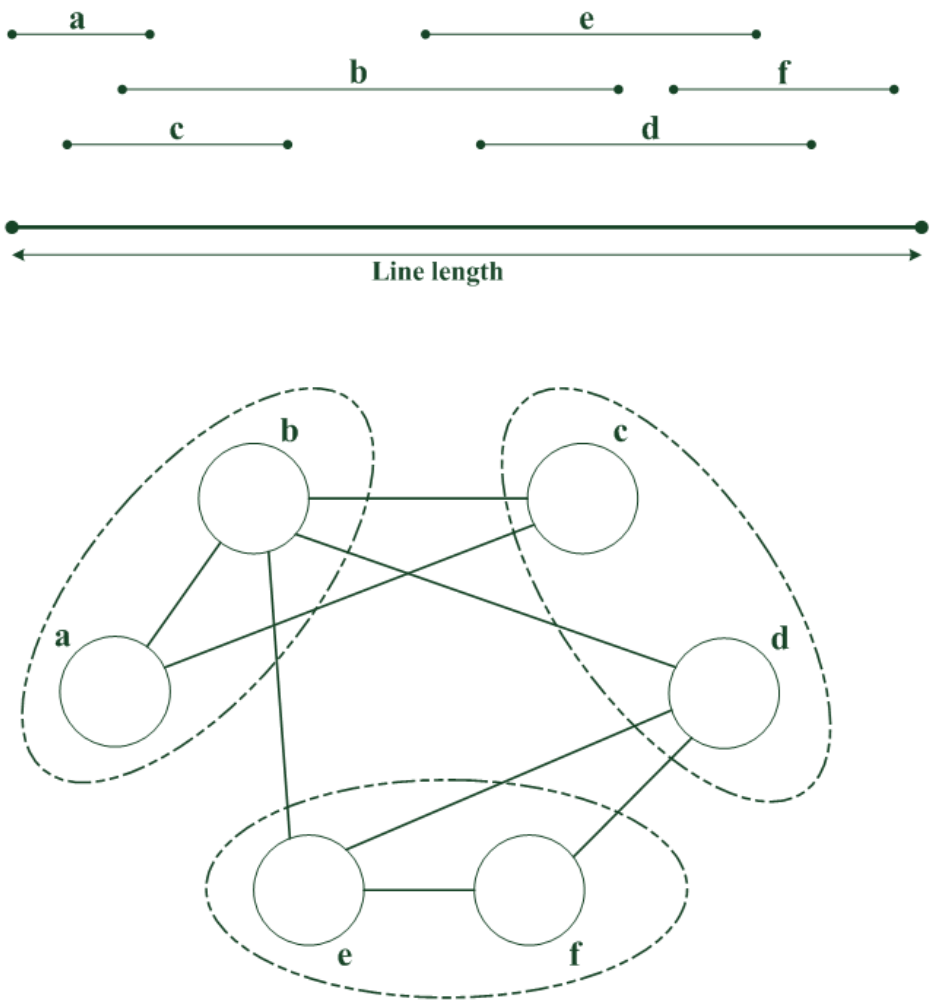

Figure 2.6. Partitioned Interval Graph Generation.

2.4.3. Partitioned Disk Graph Generator

Disk graphs are very important graphs for various applications. Therefore, we decided to create disk graph instances and study them separately. Partitioned disk graph generator algorithm works like partitioned interval graph generator. We cannot give desired density to the graph as a probability of edge existence. Instead, we randomly create disks on euclidean plane. If two disks intersect, the algorithm adds an edge between these two vertices (disks).

Partitioned disk graph generator algorithm takes *number of vertices*, *number of clusters*, *line length-X*, *line length-Y*, *minimum radius*, *maximum radius* and *radius length* as an input. We should determine *line length-X*, *line length-Y*, *minimum radius* and *maximum radius* to obtain a partitioned disk graph with a desired density.

Partitioned disk graph generation example is shown in Figure 2.7.

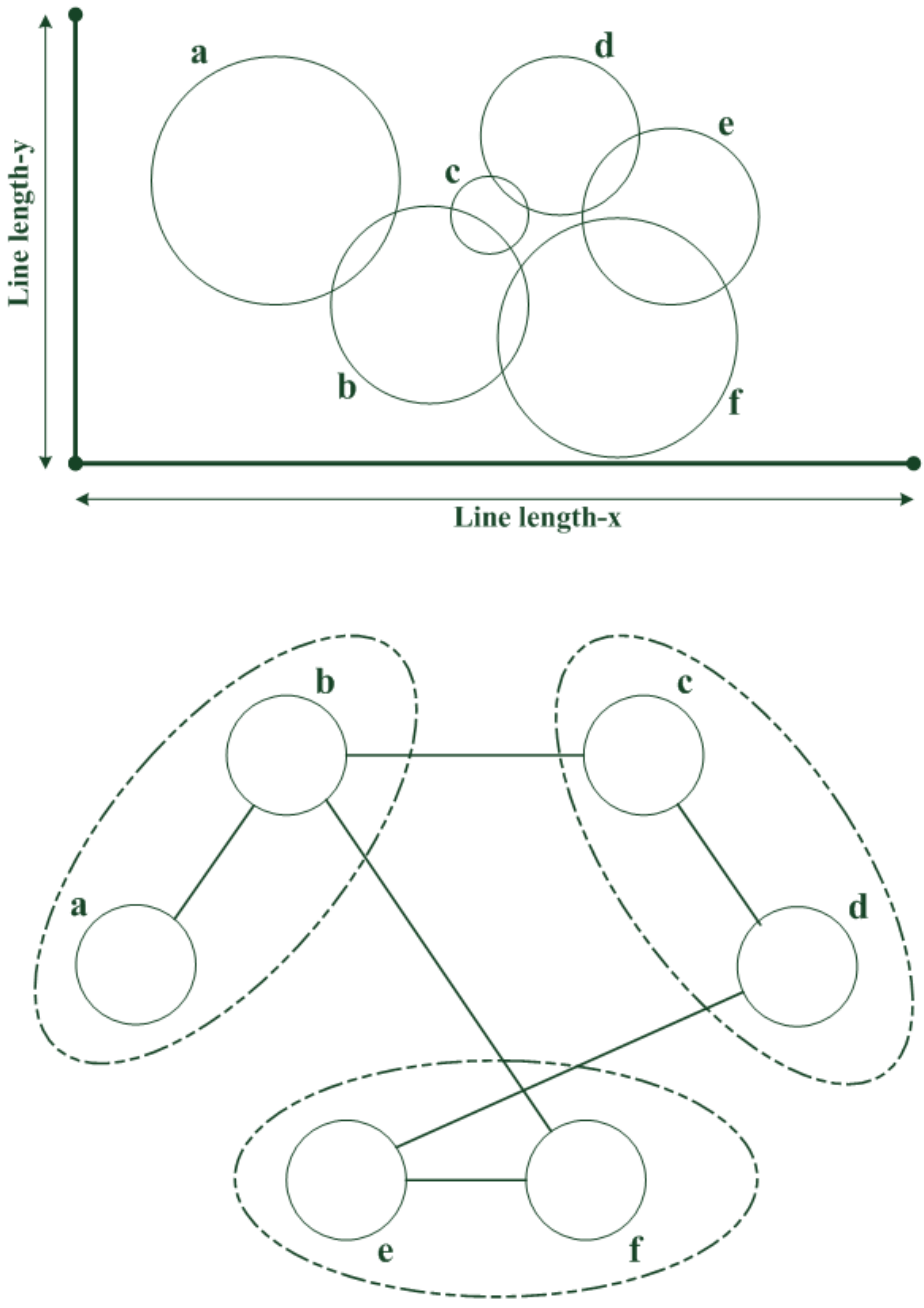

Figure 2.7. Partitioned Disk Graph Generation.

2.4.4. Selective Graph Generator from a Known Graph Instance

This algorithm creates partitioned graph instances from known graph instances by adding new vertices to the graph with a desired density. It takes a known graph instance as an input and divide each vertex to a different cluster. Then the algorithm adds certain number of vertices to each cluster and creates a new partitioned graph instance. New edges are added to the graph depending on the desired probability of edge existence. "DSJC500.5" instance is used in our experimental studies [11].

3. EXACT ALGORITHMS FOR THE SELECTIVE GRAPH COLORING PROBLEM

In this chapter, we studied on new exact methods to solve the selective graph coloring problem for some special graph classes in polynomial time.

3.1. IP Formulation of The Selective Graph Coloring Problem

We can use an integer programming (IP) formulation to solve small selective coloring problem instances. Two different IP formulations for the selective graph coloring problem are given below.

$$\min \sum_{s \in S} x_s \tag{3.1}$$

$$\sum_{i \in V_k} \sum_{i \in S} x_s = 1 \qquad \forall\, V_k \tag{3.2}$$

$$x_s \in \{0, 1\} \quad \forall\, s \in S \tag{3.3}$$

Selective stable set is a stable set that does not contain more than 1 vertex from any cluster. S represents all selective stable sets in a partitioned graph and s is one of them. All elements of S contains maximum 1 vertex from each cluster. x_s is a binary variable, if a stable set(s) is selected it takes 1, otherwise 0.

Objective function is given in Equation 3.1. Our objective is to find the minimum number of stable sets that covers all clusters. In this way, we will find minimum number of colors because each stable set will represent a color class.

Equation 3.2 indicates, for each cluster V_k in any feasible solution, there should

be exactly 1 stable set containing a vertex of V_k. This constraint ensures that there is exactly 1 vertex selected from each cluster.

We can find too many stable sets in a partitioned graph but we should add a constraint to restrict the model to select exactly one vertex from each cluster to obtain a feasible selective coloring. If the model selects more than 1 vertex from a cluster, it will ruin feasibility. Similarly, if the model does not select any vertices from each cluster, it will not be a complete selective coloring. To guarantee that selection, we added Equation 3.2 to the formulation as a constraint set.

In Figure 3.1, there is an example of finding S set for a given instance. In this example, $S = \{(a), (b), (c), (d), (e), (f), (a, f), (b, c), (b, f), (b, c, f), (c, f), (d, e)\}$. IP model will return 1 by selecting only (b, c, f) stable set, it means that the example graph can be selectively colored by only 1 color if we select vertices b,c and f.

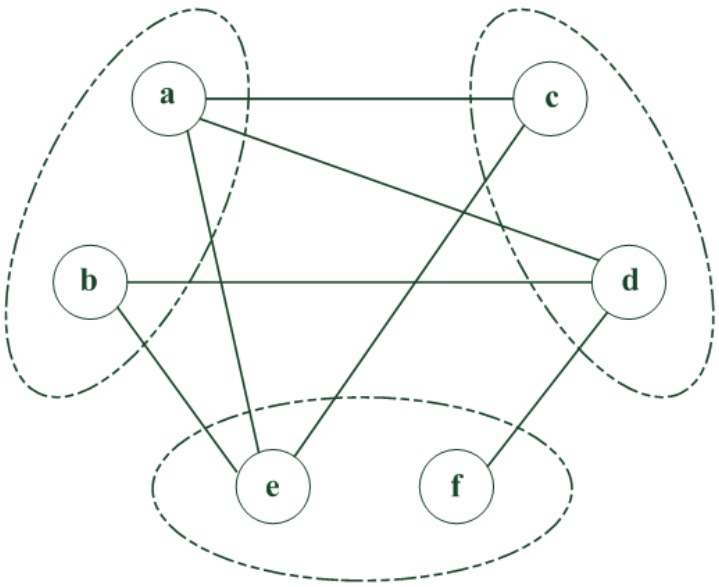

Figure 3.1. An example of finding exact solution with IP Model-1.

In Figure 3.2, there is another example of finding S set for a given instance. In this example, $S = \{(a), (b), (c), (d), (e), (f), (a, f), (b, c), (b, f), (d, e)\}$. IP model will return 2 by selecting one of $([a, f][c]), ([a, f][d]), ([b, c][e]), ([b, c][f]), ([b, f][d]), ([d, e][a]), ([d, e][b])$ stable set couple, it means that the example graph can be colored by 2 colors.

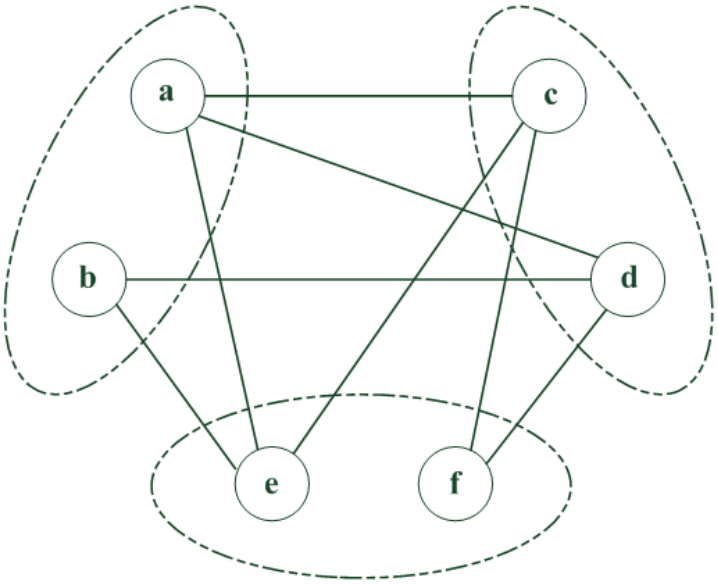

Figure 3.2. An example of finding exact solution with IP Model-2.

This formulation contains an exponential number of variables. Therefore, column generation can be useful. Column generation was used for classical graph coloring problem in [12]. In that article, each variable stands for an independent set in a graph. Our model is a generalization of this model to solve the selective graph coloring problem.

Another way to formulate the selective coloring problem as an integer programming is the following.

$$\min \sum_j z_j \tag{3.4}$$

$$x_{ij} \leq z_j \quad \forall\, j \tag{3.5}$$

$$x_{ij} + x_{kj} \leq 1 \quad \forall\, ik \in E \tag{3.6}$$

$$\sum_{i \in V_k} \sum_j x_{ij} = 1 \quad \forall\, V_k \tag{3.7}$$

$$z_j \in \{0,1\} \quad \forall\, j \tag{3.8}$$

$$x_{ij} \in \{0,1\} \quad \forall\, i,j \tag{3.9}$$

z_j is a binary variable, if color j is used then z_j is equal to 1, 0 otherwise. x_{ij} is also a binary variable. If vertex i is colored with color j, x_{ij} is equal to 1, 0 otherwise. The objective function, given in Equation 3.4, is to use minimum number of colors for a selective coloring problem. Equation 3.5 forces z_j to be 1 if some vertex i get color j, that is if color j is used.

Vertices in a color class cannot have any edges among them. Equation 3.6 provides this condition. The rule that exactly 1 vertex should be colored from each cluster is secured by Equation 3.7.

3.2. Novel Formulations to Solve The Selective Graph Coloring Problem

In this section, let us propose an exact solution method based on a novel formulation

$$\min \quad \chi(G[V']) \tag{3.10}$$

$$\sum_{x_i \in V_j} x_i = 1 \qquad \forall\ V_j \tag{3.11}$$

$$x_i \in \{0, 1\} \tag{3.12}$$

x_i is a binary variable. x_i is equal to 1 if vertex i is in the selection, otherwise it equals to 0. Equation 3.11 is the constraint that guarantees exactly 1 vertex selection from each cluster. We try to find the best selection that gives minimum coloring. However, objective function is not linear and we can define a selection as $G[\hat{x}]$. $\chi(G[\hat{x}])$ is the chromatic number of the selection.

An equivalent formulation with linear objective function is as follows.

$$\min \quad t \tag{3.13}$$

$$\sum_{x_i \in V_j} x_i = 1 \qquad \forall\ V_j \tag{3.14}$$

$$t \geq \chi(G[\hat{x}]) \tag{3.15}$$

$$x_i \in \{0, 1\} \tag{3.16}$$

This model is not well defined because of Equation 3.15. In order to solve it using IP tools, we should be able to express it as a set of linear inequalities. Then, there are two main difficulties in this model. First one is that finding $\chi(G[\hat{x}])$ is NP-complete and the second one is that the model has an exponential number of constraints. For this reason, we separated the model into the master problem and the subproblem. We developed a decomposition method which adds constraints in a cutting-plane fashion. Master problem and subproblem are given below.

$$Master\ Problem: \quad \min \quad t \tag{3.17}$$

$$\sum_{x_i \in V_j} x_i = 1 \qquad \forall\ V_j \tag{3.18}$$

$$x_i \in \{0, 1\} \tag{3.19}$$

Subproblem: If at least 1 vertex does not change in $\hat{x}$, then t cannot be lower than $\chi(G[\hat{x}])$.

In this model, if we can find polynomial time solvable cases for $\chi(G[V'])$ and add $t \geq \chi(G[V'])$ constraint to the master problem in a sequence of some logic, we can find a result in polynomial time. If in some iteration, the same selection with one of the past selections comes to the subproblem from the master problem, it means that there

are no better solutions and this is the stopping criterion of the model.

3.2.1. Exact Selective Coloring Algorithm for Interval Graphs

We concentrated on perfect graphs because their chromatic number is equal to their maximum clique size ($\chi(G[V]) = \omega(G[V])$). Since this method only applies to perfect graphs, if we can find selective maximum clique number of a graph in polynomial time, it will be equal to the selective chromatic number of the graph. We know that there is a Greedy Coloring Algorithm for interval graphs that gives the chromatic number of a graph for classical graph coloring problem. In other words, Greedy Coloring Algorithm on interval graphs gives us optimal coloring in polynomial time. Start points of intervals are sorted in ascending order, and then sorted intervals are colored with the smallest available color [1].

In Figure 3.3, there is an example of Greedy Coloring Algorithm. Firstly, intervals are sorted in ascending order then sorted intervals are colored with the smallest available color. In the example, the chromatic number of the interval graph is 3. Vertices c and h are colored by $c1$, b, d and g are colored by $c2$ and a, e and f are colored by $c3$.

Greedy Coloring Algorithm is a very useful tool. It can give the optimal coloring of a selection in a partitioned graph, but it cannot guarantee the optimal selective coloring of the partitioned graph instance because the selected graph is most probably not the best selection.

We can find all cliques in a graph in polynomial time ($\mathrm{O}(n)$). All intervals sorted by their start points and then we draw a line vertically from every start point and if other intervals touches that line, we add this interval to current clique. An example is given in Figure 3.3. In the selective graph coloring problem, if we find all cliques within the graph G and add all cliques as clique constraints in Equation 3.22 to the selective coloring model we can find selective chromatic number of the graph. Cluster constraints in Equation 3.21 force the model to select exactly 1 vertex from each cluster.

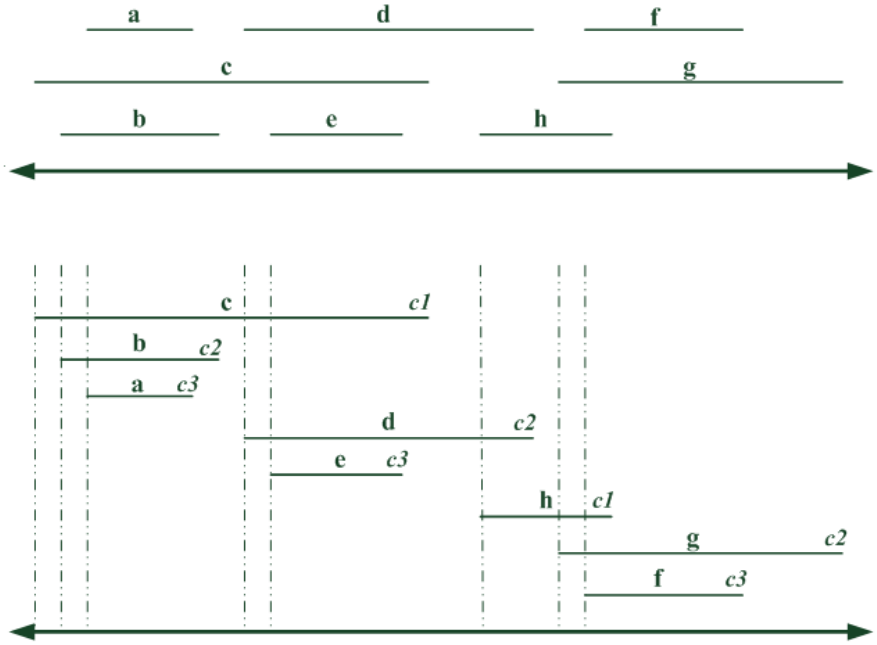

Figure 3.3. An example of Greedy Coloring Algorithm.

The model will find minimum clique among all possible selections and it will be equal to selective chromatic number of the graph. The model is given below.

$$\min \quad t \tag{3.20}$$

$$\sum_{x_i \in V_j} x_i = 1 \qquad \forall \ V_j \tag{3.21}$$

$$t \geq \sum_{i \in K} x_i \qquad \forall \ cliqueK \in G \tag{3.22}$$

$$x_i \in \{0, 1\} \tag{3.23}$$

3.2.2. Computational Results of Exact Selective Coloring Algorithm for Interval Graphs

Computational results of exact selective coloring algorithm for interval graphs are given in this section. We tried the algorithm on different instances. These instances

are created by partitioned interval graph generator. We created instances that have 500, 1000, 2000 and 3000 clusters with 1, 2, 3 or 4 vertices in each cluster and 0.3, 0.5 and 0.7 densities. 3 instances were generated for each different combination. Results are average of (χ_{SEL}) of these 3 instances. Results of 500 and 1000 clustered instances are given in Table 3.1, 2000 and 3000 clustered instances are given in Table 3.2.

Table 3.1. Exact Results for Interval Graphs (500-1000 Clusters).

	χ_{SEL}	**Time (seconds)**
Int500Cl500Ver03Den	106.7	0.0
Int500Cl500Ver05Den	189.0	0.0
Int500Cl500Ver07Den	267.7	0.0
Int500Cl1000Ver03Den	55.7	0.2
Int500Cl1000Ver05Den	91.3	0.2
Int500Cl1000Ver07Den	144.0	0.2
Int500Cl1500Ver03Den	41.0	0.5
Int500Cl1500Ver05Den	64.0	1.6
Int500Cl1500Ver07Den	95.3	3.3
Int500Cl2000Ver03Den	32.3	0.7
Int500Cl2000Ver05Den	50.0	1.9
Int500Cl2000Ver07Den	72.0	4.0
Int1000Cl1000Ver03Den	198.7	0.0
Int1000Cl1000Ver05Den	372.0	0.0
Int1000Cl1000Ver07Den	537.3	0.0
Int1000Cl2000Ver03Den	108.7	0.3
Int1000Cl2000Ver05Den	179.0	0.5
Int1000Cl2000Ver07Den	276.3	0.7
Int1000Cl3000Ver03Den	78.3	1.3
Int1000Cl3000Ver05Den	125.3	3.4
Int1000Cl3000Ver07Den	179.7	5.1
Int1000Cl4000Ver03Den	62.0	1.5
Int1000Cl4000Ver05Den	99.7	4.2
Int1000Cl4000Ver07Den	136.7	7.5

These results show that, although selective coloring is NP-complete in interval graphs, the model for interval graphs is running very fast and it can find the selective chromatic number of very huge graphs within seconds. This is indeed very satisfactory since interval graphs are of particular interest for the selective coloring problem because of the related scheduling problems.

3.2.3. Exact Maximum Selective Clique Algorithm for Unit Disk Graphs

In general, the complexity of finding maximum clique of a graph is NP-hard. Unit disk graphs are exceptional for this problem like interval graphs. Polynomial time algorithm of finding the maximum clique of a unit disk graph is given in [13]. In disk graphs, we know that $(\chi(G[V]) = \omega(G[V]))$ equation does not hold in all instances. But $\omega(G[V])$ gives a lower bound for $\chi(G[V])$. The study on exact maximum selective clique algorithm for unit disk graphs is motivated by that equation.

In this algorithm, at first intersecting disks are found. Then, for every pair of intersecting disks, two new disks are generated centered on the two disks center with radius of the distance between centers of two disks. An example is shown with A and B disks in Figure 3.4.

We will use the intersection area of these new two disks to find a clique within the graph. Every pair of intersecting disks give one clique of the graph with this method. We separate the intersection area into two parts; upper area of intersection and lower area of intersection. An example is shown in Figure 3.5.

By the reason of disks are unit disks, all disks that has centers in upper area and lower area have edges between each other. In other words, there are one complete graph in upper area and one complete graph in lower area. Therefore their complement gives us a bipartite graph. It will help us to find the maximum clique within these vertices by solving maximum flow problem for this bipartite graph. It will give us a clique and we search for all pair of intersecting disks to find these cliques. At last, maximum sized clique of these cliques give us the maximum clique of the graph.

Table 3.2. Exact Results for Interval Graphs (2000-3000 Clusters).

	Exact Results	**Time (seconds)**
Int2000Cl2000Ver03Den	373.3	0.0
Int2000Cl2000Ver05Den	715.3	0.0
Int2000Cl2000Ver07Den	1029.0	0.0
Int2000Cl4000Ver03Den	214.7	0.7
Int2000Cl4000Ver05Den	354.0	1.2
Int2000Cl4000Ver07Den	542.0	1.8
Int2000Cl6000Ver03Den	154.0	3.5
Int2000Cl6000Ver05Den	252.0	5.8
Int2000Cl6000Ver07Den	356.3	11.7
Int2000Cl8000Ver03Den	119.7	7.7
Int2000Cl8000Ver05Den	191.7	8.8
Int2000Cl8000Ver07Den	2650	16.2
Int3000Cl3000Ver03Den	566.3	0.0
Int3000Cl3000Ver05Den	1081.3	0.0
Int3000Cl3000Ver07Den	1529.7	0.1
Int3000Cl6000Ver03Den	3100	1.5
Int3000Cl6000Ver05Den	521.3	1.8
Int3000Cl6000Ver07Den	809.3	2.7
Int3000Cl9000Ver03Den	224.0	5.4
Int3000Cl9000Ver05Den	369.0	10.4
Int3000Cl9000Ver07Den	531.0	18.8
Int3000Cl12000Ver03Den	178.3	6.9
Int3000Cl12000Ver05Den	285.0	15.5
Int3000Cl12000Ver07Den	396.0	25.1

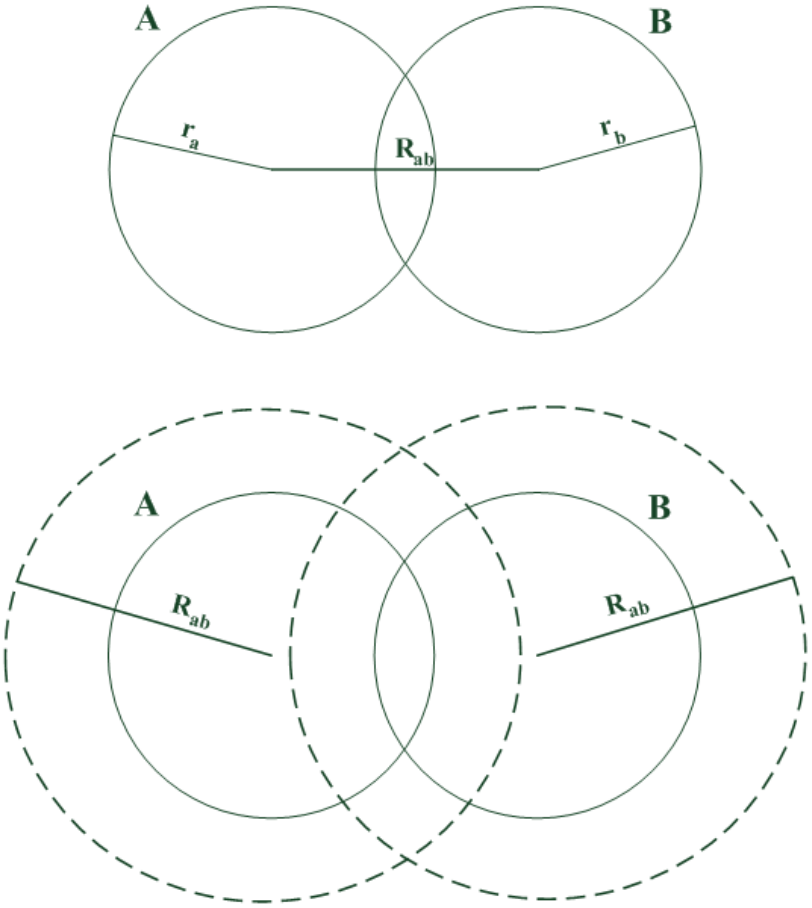

Figure 3.4. An example of generating two new disks.

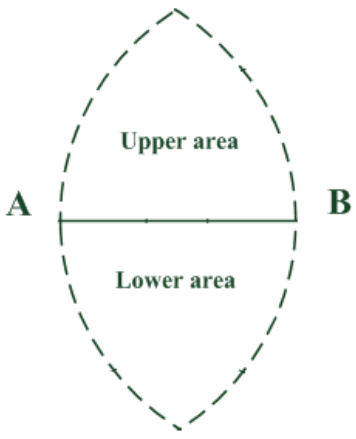

Figure 3.5. An example of separating intersection area.

We used this algorithm to solve exact maximum selective clique algorithm for unit disk graphs to find a lower bound for $\chi_{SEL}(G[V])$ and t will be greater than or equal to $\chi_{SEL}(G[V])$. IP formulation of the problem is given below and subproblem is solved like we gave above. Subproblem of the problem is finding maximum clique of the selected graph with polynomial time algorithm for unit disk graphs. When subproblem finds a maximum clique, it will be added to the master problem as a constraint as $t \geq \sum_{i \in K} x_i$. K is the maximum selective clique in the selection. It will stop when the same vertex selection comes to the subproblem because it means that there are no better different selections that we can try. The maximum clique of this selection will be our result for the problem.

$$Master\ Problem: \quad \min \quad t \tag{3.24}$$

$$\sum_{x_i \in V_j} x_i = 1 \qquad \forall\ V_j \tag{3.25}$$

$$x_i \in \{0, 1\} \tag{3.26}$$

$$Sub\ Problem: \quad t \geq \sum_{i \in K} x_i \tag{3.27}$$

3.2.4. Computational Results of Exact Maximum Selective Clique Algorithm for Unit Disk Graphs

Computational results of exact maximum selective clique algorithm for disk graphs are given in this section. We restricted algorithm time to 1 hour. Due to exponential time increase in CPLEX time, algorithm cannot return an exact solution more than 30 clusters with 2 vertices in each cluster. Moreover, it cannot find a solution for an instance that have 30 clusters and 2 vertices in each cluster with 0.7 density. We extended algorithm time to 6 hours, but even in this case the instance did not return an exact solution.

We tried the algorithm on different instances. These instances are created by partitioned disk graph generator. We created instances that have 10, 20 and 30 clusters with 2 vertices in each cluster, 0.3, 0.5 and 0.7 densities and 1, 2, 3 radius length. 3 instances were generated for each different combination. Results are average of (ω_{SEL}) of these 3 instances. Exact results of these instances are given in Table 3.3.

Table 3.3. Exact Results for Unit Disk Graphs (10-30 Clusters).

	ω_{SEL}	**Total (seconds)**	**CPLEX (seconds)**
Disk10Cl20Ver03Den1Rad	2.7	0.1	0.1
Disk10Cl20Ver03Den2Rad	2.0	0.0	0.0
Disk10Cl20Ver03Den3Rad	2.0	0.0	0.0
Disk10Cl20Ver05Den1Rad	3.7	0.0	0.0
Disk10Cl20Ver05Den2Rad	3.3	0.0	0.0
Disk10Cl20Ver05Den3Rad	2.0	0.0	0.0
Disk10Cl20Ver07Den1Rad	5.7	0.5	0.4
Disk10Cl20Ver07Den2Rad	4.3	0.0	0.0
Disk10Cl20Ver07Den3Rad	3.0	0.0	0.0
Disk20Cl40Ver03Den1Rad	4.3	0.8	0.7
Disk20Cl40Ver03Den2Rad	4.0	0.1	0.1
Disk20Cl40Ver03Den3Rad	4.0	0.1	0.0
Disk20Cl40Ver05Den1Rad	6.7	1.3	1.2
Disk20Cl40Ver05Den2Rad	5.7	2.5	2.3
Disk20Cl40Ver05Den3Rad	5.7	0.9	0.7
Disk20Cl40Ver07Den1Rad	9.3	96.8	95.1
Disk20Cl40Ver07Den2Rad	8.0	7.2	6.6
Disk20Cl40Ver07Den3Rad	7.7	7.1	6.5
Disk30Cl60Ver03Den1Rad	6.3	150.1	148.2
Disk30Cl60Ver03Den2Rad	5.0	0.8	0.7
Disk30Cl60Ver03Den3Rad	5.0	1.4	1.2
Disk30Cl60Ver05Den1Rad	9.3	1054.9	1051.5
Disk30Cl60Ver05Den2Rad	7.7	308.0	302.1
Disk30Cl60Ver05Den3Rad	7.7	1037.7	1030.5

4. NEW HEURISTIC APPROACHES TO THE SELECTIVE GRAPH COLORING PROBLEM

The graph coloring problems are in general among the hardest problems in combinatorial optimization and we already know that the selective graph coloring problem is at least as hard as classical graph coloring problem. Therefore, we cannot find an exact solution for the selective graph coloring problem in a reasonable time. We should use heuristic algorithms to find closer upper bounds to the exact solution. For this purpose, we decided to develop new Tabu Search algorithms to find good upper bounds.

4.1. Construction Heuristics for Partitioned Graphs

In order to reduce the computational times of the heuristic algorithms, construction heuristics are used to start with good starting solutions. Their results are used as an input to the heuristic algorithms. We used three different construction heuristics.

First one is Minimum Outdegree Algorithm. This algorithm calculates for each vertex v the number of neighbors which are not in the same cluster as v. In other words, if a vertex has a neighbor in the same cluster, it is not counted as an outdegree. It is called outdegree of a vertex. The algorithm selects exactly one vertex from each cluster which has minimum outdegree. Then, selected vertices are sorted by their cluster id's in ascending order. If the partitioned graph is a partitioned interval graph, selected vertices are sorted by starting points of their intervals in ascending order. These sorted vertices are colored by the color which is first available; if there are no available colors, the vertex is colored by a new color.

An example of Minimum Outdegree Algorithm is shown in Figure 4.1. In Figure 4.1, vertices that has minimum outdegrees in each cluster are selected. Vertices b, c, e and g are colored by 2 colors with minimum outdegree algorithm.

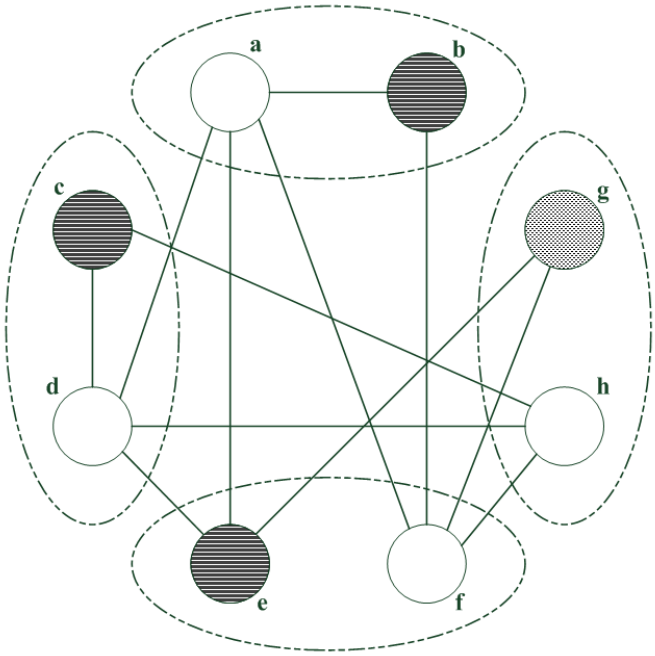

Figure 4.1. An example of Minimum Outdegree Algorithm.

Second construction heuristic is Random Selection Algorithm. This algorithm selects arbitrarily exactly one vertex from each cluster. Then, it colors the selected vertices same as Minimum Outdegree Algorithm.

Third construction heuristic is One Step Color Degree Algorithm [3]. This is the generalization of DSATUR which is widely used and the fastest sequential coloring algorithm [14].

This algorithm calculates and works with color-degrees and uncolored-degrees. Color-degree of a vertex is the number of different colors that its neighbors(adjacent vertices) have. Uncolored-degree of a vertex is the number of uncolored adjacent vertices that the vertex have. Firstly, algorithm removes all edges within the clusters. Then, it calculates color-degrees and uncolored-degrees of all vertices and selects minimum color degreed vertex from each cluster. If there is a tie, the algorithm chooses vertex with the smallest uncolored degree. From these vertices, algorithm selects the vertex with the largest color-degree. If there is a tie, the algorithm chooses vertex with the largest uncolored degree. The smallest available color is given to this vertex and color-degrees and uncolor-degrees are updated. Vertices in the same cluster with the

selected vertex are removed. These processes are repeated until all clusters are colored.

Since the selective graph coloring problem is NP-hard, construction heuristics cannot give us the optimal coloring. However, construction heuristics are very useful to determine an upper bound for our instances. If we know a feasible solution for an instance, this can be used as an initial solution for a heuristic algorithm such as tabu search algorithm. Determining a good upper bound and starting with this solution substantially saves computational time.

4.2. Computational Results of Construction Heuristics

In this section, we compare the results of three different construction heuristics. We generated two different graph instance sets. First set has 500 clusters, 1000 vertices and 9 different densities. For example, "500Cl1000Ver01Den" instance has 500 clusters, 1000 vertices and 0.1 edge density. Second set has a fixed density (0.5) and fixed number of vertices in each cluster which is 2. Their cluster numbers are varied 100 to 1000 and there are 10 different instances of each type.

Table 4.1. Comparison of Construction Heuristics (0.1-0.9 Edge Density).

	onestepCD	**Min.Outneig.**	**Random Sel.**
500Cl1000Ver01Den	14.0	18.1	18.1
500Cl1000Ver02Den	24.5	30.3	30.4
500Cl1000Ver03Den	35.2	41.8	42.6
500Cl1000Ver04Den	46.9	55.3	55.4
500Cl1000Ver05Den	59.8	70.0	69.7
500Cl1000Ver06Den	74.9	86.5	85.7
500Cl1000Ver07Den	92.7	106.0	105.2
500Cl1000Ver08Den	116.8	131.5	130.8
500Cl1000Ver09Den	154.8	172.2	170.7

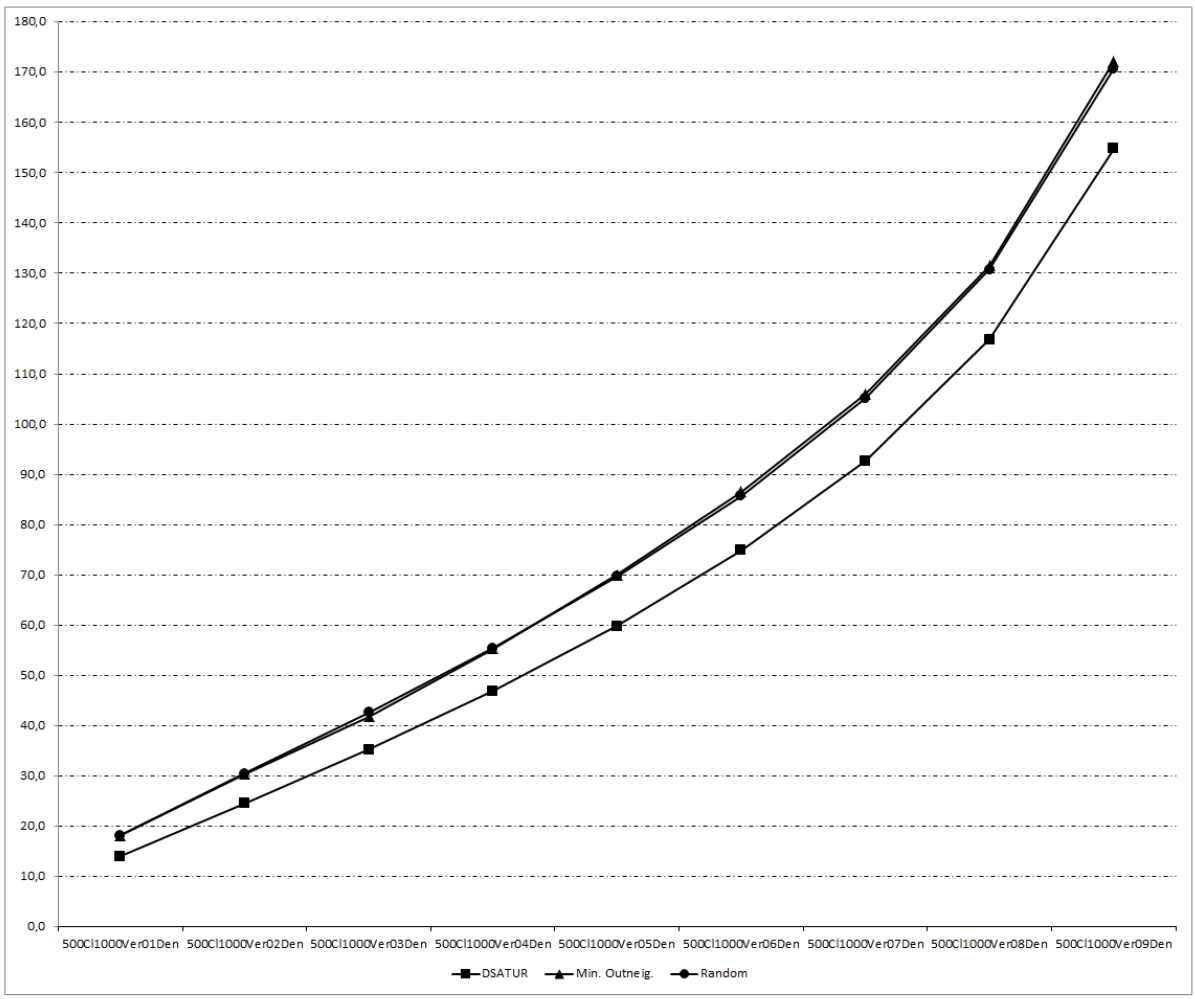

Figure 4.2. Comparison of Construction Heuristics (0.1-0.9 Edge Density).

Table 4.2. Comparison of Construction Heuristics (100-1000 Number of Clusters).

	onestepCD	**Min.Outneig.**	**Random Sel.**
100Cl200Ver05Den	16.4	19.6	19.1
200Cl400Ver05Den	28.2	34.0	33.2
300Cl600Ver05Den	39.4	46.8	46.0
400Cl800Ver05Den	50.2	58.6	57.9
500Cl1000Ver05Den	60.2	70.7	70.0
600Cl1200Ver05Den	70.2	80.4	80.6
700Cl1400Ver05Den	80.2	92.2	91.7
800Cl1600Ver05Den	89.0	101.4	102.3
900Cl1800Ver05Den	98.4	112.4	112.8
1000Cl2000Ver05Den	107.8	122.9	123.1

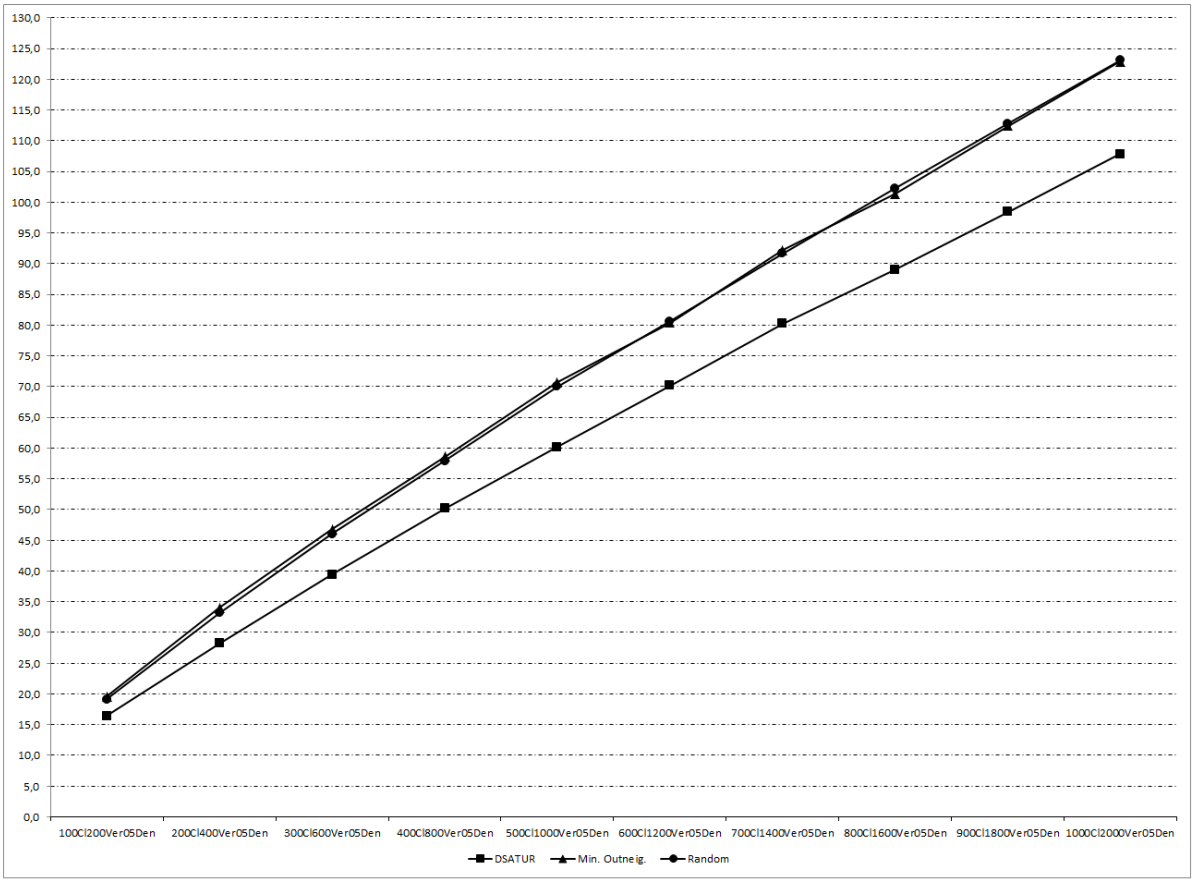

Figure 4.3. Comparison of Construction Heuristics (100-1000 Number of Clusters).

onestepCD algorithm worked better than the other algorithms and all three algorithms give solution to experimental instances in at most 2 seconds. For this reason, we will use onestepCD algorithm as our starting solution for Tabu algorithms.

4.3. New Tabu Search Algorithms for The Selective Graph Coloring Problem

Tabu search is widely used in graph theory problems. Especially, tabu search is very effective in graph coloring problems [15]. Therefore, we chose to use tabu search in the selective graph coloring problem.

Tabu search is a deterministic local search methodology. At each iteration, the best solution in the neighborhood of the current solution is selected as the new current solution, even if it leads to an increase in solution cost. Escaping from a local optimum is what makes the different between a pure local descent. The tabu list, which is a short-term memory, stores recently visited solutions to prevents choosing these solutions in a short-term, thus prevents short-term cycling. Typically, the search stops after a fixed number of iterations or a maximum number of consecutive iterations without any improvement to the best known solution [16].

Let f be an objective function which must be minimized over the solution space S. The basic version of Tabu Search can be described as follows. First, it needs an initial solution $s_0 \in S$ as input. Then, the algorithm generates a sequence of solutions $s_1, s_2, ...$ in the search space S such that $s_{i+1} \in N(s_i)$. When a move is performed from s_i to s_{i+1}, the inverse of that move is stored in a tabu list L. For the following t iterations, where t is the tabu tenure (for historical reasons also called tabu list length), a move stays tabu and cannot be used (with some exceptions) to generate a neighbor solution. The solution s_{i+1} is computed as $s_{i+1} = arg\, min_{s \in N'(s_i)} f(s)$, where $N'(s)$ is a subset of $N(s)$ containing all solutions s' which can be obtained from s either by performing a move that is not in L (i.e. not tabu) or such that $f(s') \leq f(s^*)$, where s^* is the best solution encountered along the search so far. The process is stopped when a fixed number of iterations without improving s^* have been performed or when a given

amount of CPU time has elapsed [17].

Tabus are used to prevent cycling when moving away from local optima through non-improving moves. When we encounter a situation like that, we have to prevent the search from tracing back its steps to where it came from. Certain actions are labeled as tabu to be able to direct the algorithm to head towards. In other words, the search is not allowed to return a recently visited solution in the search space. Tabus are sometimes too powerful. They may prohibit attractive moves, even when there is no danger of cycling, or they may lead to a search process that remains stable. It is thus necessary to use algorithmic devices that will allow one to cancel tabus. These are called aspiration criteria. The simplest and most commonly used aspiration criterion, found in almost all tabu search implementations, allows a tabu move when it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited) [18].

If the length of the tabu list is too small, restricting tabu effect might not be achieved. Conversely, a too long tabu list size creates too many restrictions and it has been observed that the mean value of the visited solutions grows with the increase of the tabu list size. Usually, an order of magnitude of this size may be easily determined. However, it is difficult, even impossible, to fine a tabu list size value that both prevents cycling and is not very restricting at the same time for all the instances of a given optimization problem. This problem can be effectively handled by using a variable tabu list size. Each element of the list belongs to it for a number of iterations that is bounded by given maximal and minimal values [19].

Static tabu tenure is a tenure which does not change during the search process. The tabu tenure is chosen once and for all in the original version of tabu search. For all instances this tabu tenure is kept fixed. This behavior leads to results that compare poorly to other methods of choosing the tabu tenure. One other way is to let the user supply the tabu tenure but this is also not desirable since the user should not be part of a heuristic. By choosing the tabu tenure as a function of the instance size can improve the solutions but choosing the tabu tenure as a function of the mean size of

the neighborhood of a solution is even a better criterion. The idea behind this is that the larger the neighborhood of a solution s is, the more possibilities there are to reach this solution. Therefore, the tabu tenure should be larger in order to avoid revisiting s. Dynamic tabu tenure depends on the current solution and on the move which has been executed to obtain the current solution. No information about previous visited solutions is required. If the tenure is determined on the basis of the entire search history, this is reactive tabu tenure. The tabu tenure is increased if a cycle is detected because we must assume that it was too small to prevent the detected cycle. The tabu tenure is decreased if no cycle is detected during a given number of iterations. In addition, if the search appears to be repeating an excessive number of solutions too often, then the search is diversified by making a number of random moves, proportional to the average of the cycle length, in order to escape a local attractor [17].

Tabu search is a famous local search technique that can be applied to a broad range of practical problems and achieve good results. Local search techniques are iterative procedures that aim to find a solution s, minimizing an objective function f, over a set S of feasible solutions. The iterative process starts from an initial solution in S, and given any solution s, the next solution is chosen in the neighborhood $N(s) \subseteq S$. Typically, a neighbor $s' \in N(s)$ is obtained from s by performing a local change on it [1].

There are two different tabu approaches that are used frequently for graph coloring problems. We developed our tabu algorithms on these common approaches.

The first approach consists of fixing k and starting with an improper k-coloring. An improper coloring may contain conflicts. If two adjacent vertices x and y have the same color, we say that there is a conflict between vertices x and y, respectively, vertices x and y are conflicting vertices, and the edge x, y is a conflicting edge. One may try then to reduce the number of conflicts until, in the best case, a k-coloring is found. If a k-coloring is found, the method is restarted in order to search for a $(k-1)$-coloring, and so on. If no k-coloring is found, we restart the method to search for a $(k+1)$-coloring, and so on. The process is stopped as soon as a fixed k is considered

twice by the method. The output solution is a k-coloring with the smallest k [17].

The second approach uses partial k-colorings with a fixed k. A partial k-coloring consists of k mutually disjoint stable sets $S_1, ..., S_k$ and a set $\mathcal{O}$ of non-colored vertices such that $\mathcal{O} = V \backslash (S_1 \cup ... \cup S_k)$. Note that a partial k-coloring has no conflict. The goal of this approach is to increase the size of the partial solution, which is equivalent to reducing the size of $\mathcal{O}$ [17].

We used both of these approaches for our new tabu algorithms for the selective graph coloring problem. We used construction heuristics to find upper bounds(ub) for partitioned graph instances. After this step, all inputs are ready for the tabu search algorithms.

[4] used tabu search for partition coloring problem (TS-PCP). They used onestepCD algorithm as an initial solution as we did for our tabu algorithms. They defined a coloring conflict as a pair of adjacent nodes in different clusters of the graph which are colored with the same color. Objective of their algorithm is to remove all conflicts within all colors. Set Q is defined as a set that involves vertices has coloring conflicts in colors. In each iteration, algorithm builds Q set again and for each vertex of Q, it tries to find a better solution set. If the algorithm can remove all conflicts after an iteration, algorithm starts again to try to find a least colored solution. Otherwise, it updates tabu list and current solution as best solution. In other words, if after an iteration there are still coloring conflicts, the algorithm retrieve best solution and discard last solution. The algorithm does not permit worse solutions.

Comparisons of results between TS-PCP and our tabu algorithms are given in Section 4.3.7.

4.3.1. Selective Improper Coloring(SelImpCol) Algorithm

We start the algorithm with $ub-1$ colors. We search whether the clustered graph admits a selective coloring using $(ub - 1)$ colors or not. When we find a solution we

decrease the number of colors by one. The procedure repeats until no feasible selective coloring is found using the given number of colors.

In SelImpCol algorithm, if we have a feasible selective $(k+1)$ selective coloring of a partitioned graph, we first need to partition the selected vertices into k different color classes. To this purpose, we remove the color class which has minimum number of vertices in it and assign these vertices to the color classes which has minimum conflict number with these vertices. It is our starting solution for SelImpCol algorithm. Another way of creating a starting solution can be selecting a random vertex from each cluster and assigning them to a random color class. But our way is more efficient in terms of computational efficiency because it starts with fewer number of conflicts and if there is a simple solution, it is found faster. Visualization of SelImpCol Algorithm is shown in Figure 4.4.

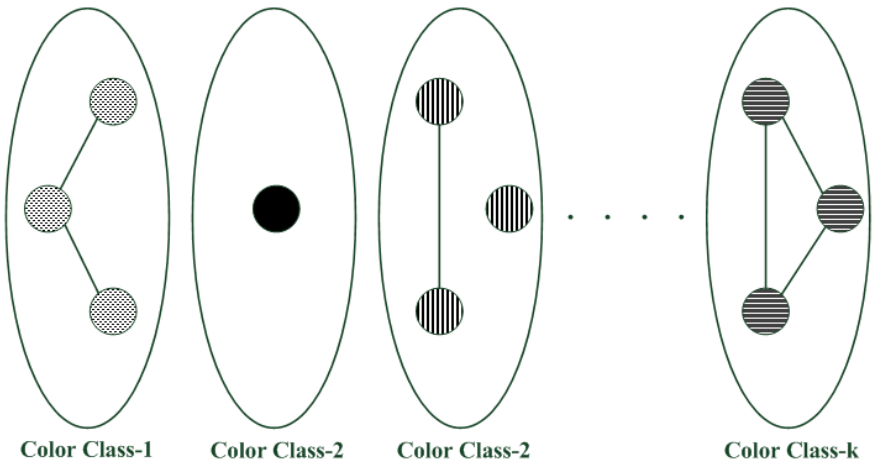

Figure 4.4. SelImpCol Algorithm Visualization.

Starting solution is most probably an infeasible solution because there are conflicts in color classes and our objective is to make this solution feasible by removing these conflicts. In other words, we start with an infeasible situation and try to make it feasible. If total number of conflict equals to 0 after a number of iterations, it means that the partitioned graph admits a selective coloring by k colors.

Objective function of the algorithm is the total number of conflicts(edges in color classes) and there is a variable which stores best objective function value($bestFS$) and

current objective function value(*currFS*). In SelImpCol algorithm, firstly we create a vertex set and add all conflicting vertices in all color classes to that set. It is named *confSet*. Then, the algorithm selects an arbitrary vertex from *confSet*. We know that the selected vertex has 1 or more conflicts in its color class, so we try to remove these conflicts by moving to a neighbor solution. For this purpose, the algorithm tries to color the selected vertex with another color(assign it to another color class) or change the selected vertex with another vertex in the same cluster and assign it to one of the color classes. Objective function value of each option((*newFS*)) is calculated in sequence and the algorithm decides what to do depending on these (*newFS*) values. If (*newFS*) is better than (*bestFS*) then a move is effectuated, else if (*newFS*) is only better than (*currFS*) then the algorithm checks for the move if it is in the tabu list or not. If the move is not in the tabu list then it is executed. If *newFS* is not better than *currFS* and not tabu, the algorithm does not perform the move and stores the solution as a candidate move and looks for other solutions for M times. If the move is a tabu move, the move is not performed and the solution is not stored. If in M candidate moves, the algorithm cannot find any better *newFS* than *currFS*, it makes a move to the non-tabu neighbor with best *newFS* among all neighbors. An example move is shown in Figure 4.5 and Figure 4.6.

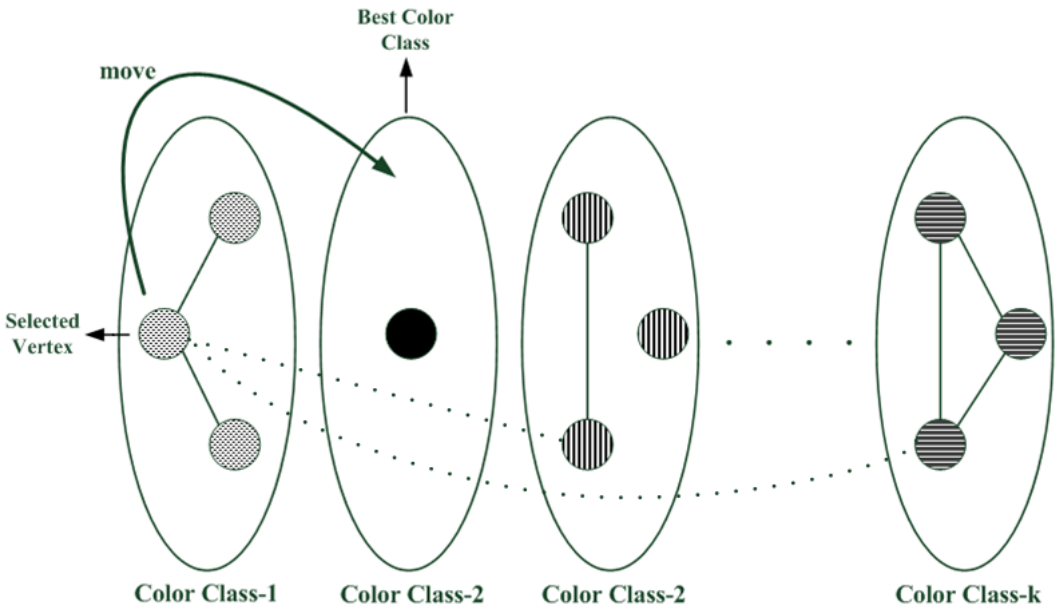

Figure 4.5. Before a move.

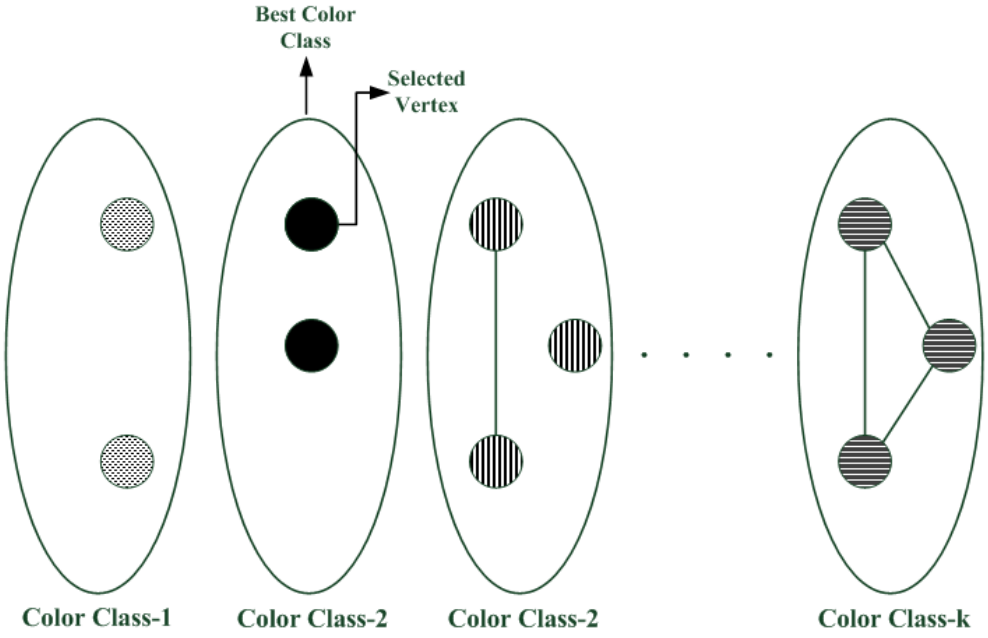

Figure 4.6. After a move.

After each move, tabu list stores moving vertex and its old color class. Tabu list inhibits this vertex to return to the previous color class for t iterations.

Tabu list has a reactive length denoted by t. It is calculated again in every b iterations. Δ is the difference between the maximum and the minimum of objective function during the last b iterations. If $\Delta \leq thresholdValue$ then $t = t + B$, and otherwise $t = t - 1$. B and b are fixed parameters [15]. It is shown in Figure 4.7.

SelImpCol algorithm for the selective graph coloring problem operates as we mentioned above. The algorithm is summarized in Algorithm 4.8.

Input: b, B, t, *thresholdValue*, *iteration*, max $|\mathcal{O}|$ in last b iterations,
min $|\mathcal{O}|$ in last b iterations;
Output: t;
if *iteration* = 0(mod b) **then**
Δ := max $|\mathcal{O}|$ - min $|\mathcal{O}|$;
if $\Delta <$ *thresholdValue* **then**
$t := t + B$;
else
$t := t$ - 1;
end if
end if

Figure 4.7. Reactive Tabu Tenure Algorithm.

4.3.2. Computational Results of SelImpCol Algorithm

First of all, we should find a suitable parameter set for reactive tabu scheme. There are 4 different parameters(b, B, C, *thresholdvalue*) that we can fix to find a good t. We fixed $b = 1000$ and $C = 1$ and searched good parameter values for B and *thresholdvalue*.

We used a known instance("DSJC500.5") to set our parameter values. This instance is a well-known and widely used instance for graph coloring algorithms. We took that instance and assigned each vertex to a different cluster. This instance is called "DSJC500.5-1". Then we added 1 vertex to each cluster and add edges to other vertices with probability 0.5. This instance is called "DSJC500.5-2". We generated "DSJC500.5-3" and "DSJC500.5-4" in the same manner. The same as in [4].

We tried a wide range of parameter values but most of them do not give good results. We showed significant results of parameter tuning experiment for SelImpCol algorithm in Tables 4.3, 4.4, 4.5 and 4.6. SelImpCol algorithm is runned 10 times for all

Input: Clustered Graph Instance, Construction Solution with k colors, b, B, t, *thresholdValue*;

Output: Tabu Solution;

Build a new starting solution as represented above and create *confSet*;

Free the tabu list, Set *iteration* := 0 and *innerIteration* := 0 and $k := k - 1$;;

Set *currFS* := # of edges within color classes and *bestFS* := # of edges within color classes;

repeat

repeat

Select a vertex v from *confSet* arbitrarily and calculate *newFS*; ;

if *newFS* < *bestFS* **then**

bestFS := *newFS* *currFS* := *newFS* *innerIteration* := M Make the move;

else if *newFS* < *currFS* and The move is not in the tabu list **then**

currFS := *newFS* *innerIteration* := M Make the move;

else if The move is not in the tabu list **then**

Store the move;

end if

if The move is not made **then**

Make the best stored move;

end if

until *innerIteration* < M

Update tabu tenure with reactive tabu tenure algorithm

until *iteration* < *maxIteration* and *bestFS* $\neq 0$

Figure 4.8. SelImpCol Algorithm.

different parameter sets for each graph. Maximum iteration number of these instances is $1,000,000$.

Table 4.3. SelImpCol Parameter Tuning (DSJC500.5-1).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-1	65	52	52.1	53	3	1
DSJC500.5-1	65	51	51.3	52	3	3
DSJC500.5-1	65	50	51	52	3	5
DSJC500.5-1	65	50	51.3	52	5	1
DSJC500.5-1	65	**50**	**51.2**	**52**	5	3
DSJC500.5-1	65	51	51.2	52	5	5
DSJC500.5-1	65	51	51.2	52	7	1
DSJC500.5-1	65	51	51.6	52	7	3
DSJC500.5-1	65	51	51.8	52	7	5

Table 4.4. SelImpCol Parameter Tuning (DSJC500.5-2).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-2	59	46	46.0	46	3	1
DSJC500.5-2	59	45	45.9	46	3	3
DSJC500.5-2	59	45	45.7	46	3	5
DSJC500.5-2	59	45	45.3	46	5	1
DSJC500.5-2	59	**45.0**	**45,3**	**46**	5	3
DSJC500.5-2	59	45	45.4	46	5	5
DSJC500.5-2	59	45	45.4	46	7	1
DSJC500.5-2	59	45	45.7	46	7	3
DSJC500.5-2	59	45	45.9	46	7	5

Table 4.5. SelImpCol Parameter Tuning (DSJC500.5-3).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-3	57	43	43.3	44	3	1
DSJC500.5-3	57	42	42.9	43	3	3
DSJC500.5-3	57	43	43.1	44	3	5
DSJC500.5-3	57	43	43.0	43	5	1
DSJC500.5-3	57	**42**	**43.0**	**44**	5	3
DSJC500.5-3	57	42	42.9	43	5	5
DSJC500.5-3	57	43	43.1	44	7	1
DSJC500.5-3	57	43	43.3	44	7	3
DSJC500.5-3	57	43	43.6	44	7	5

Table 4.6. SelImpCol Parameter Tuning (DSJC500.5-4).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-4	55	41	41.9	42	3	1
DSJC500.5-4	55	41	41.3	42	3	3
DSJC500.5-4	55	41	41.5	42	3	5
DSJC500.5-4	55	41	41.2	42	5	1
DSJC500.5-4	55	**41**	**41.5**	**42**	5	3
DSJC500.5-4	55	41	41.6	42	5	5
DSJC500.5-4	55	41	41.4	42	7	1
DSJC500.5-4	55	41	41.4	42	7	3
DSJC500.5-4	55	41	41.7	42	7	5

After these experiments, we decided to use $B = 3$ and $thresholdvalue = 5$ for our further experiments because in average that parameter set gives better results.

We used same instance sets that are generated in Section 4.2. First set has 500 clusters, 1000 vertices and 9 different densities. For example, "500Cl1000Ver01Den" instance has 500 clusters, 1000 vertices and 0.1 edge density. Second set has a fixed density (0.5) and fixed number of vertices in each cluster(2). Their cluster numbers are varied 100 to 1000 and there are 10 different instances.

Each instance type has 10 different instances in it and SelImpCol runs 5 times for each instance. Eventually, average tabu results stand for 50 runs.

Average construction algorithm results, average tabu results and improvement percentage between them are given in Tables 4.7 and 4.9. Their plots are given in Figures 4.9 and 4.11.

Table 4.7. Construction vs SelImpCol Algorithm (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
500Cl1000Ver01Den	14.0	11.0	21.7%
500Cl1000Ver02Den	24.5	18.9	22.9%
500Cl1000Ver03Den	35.2	27.0	23.4%
500Cl1000Ver04Den	46.9	35.6	24.1%
500Cl1000Ver05Den	59.8	45.1	24.6%
500Cl1000Ver06Den	74.9	55.9	25.4%
500Cl1000Ver07Den	92.7	68.9	25.7%
500Cl1000Ver08Den	116.8	85.5	26.8%
500Cl1000Ver09Den	154.8	109.8	29.1%

In Table 4.7, improvement percentage for dense graphs is higher. There are two possibilities for this outcome. First one, SelImpCol works better in high dense graphs. Second one, onestepCD algorithm is more successful in sparse graphs.

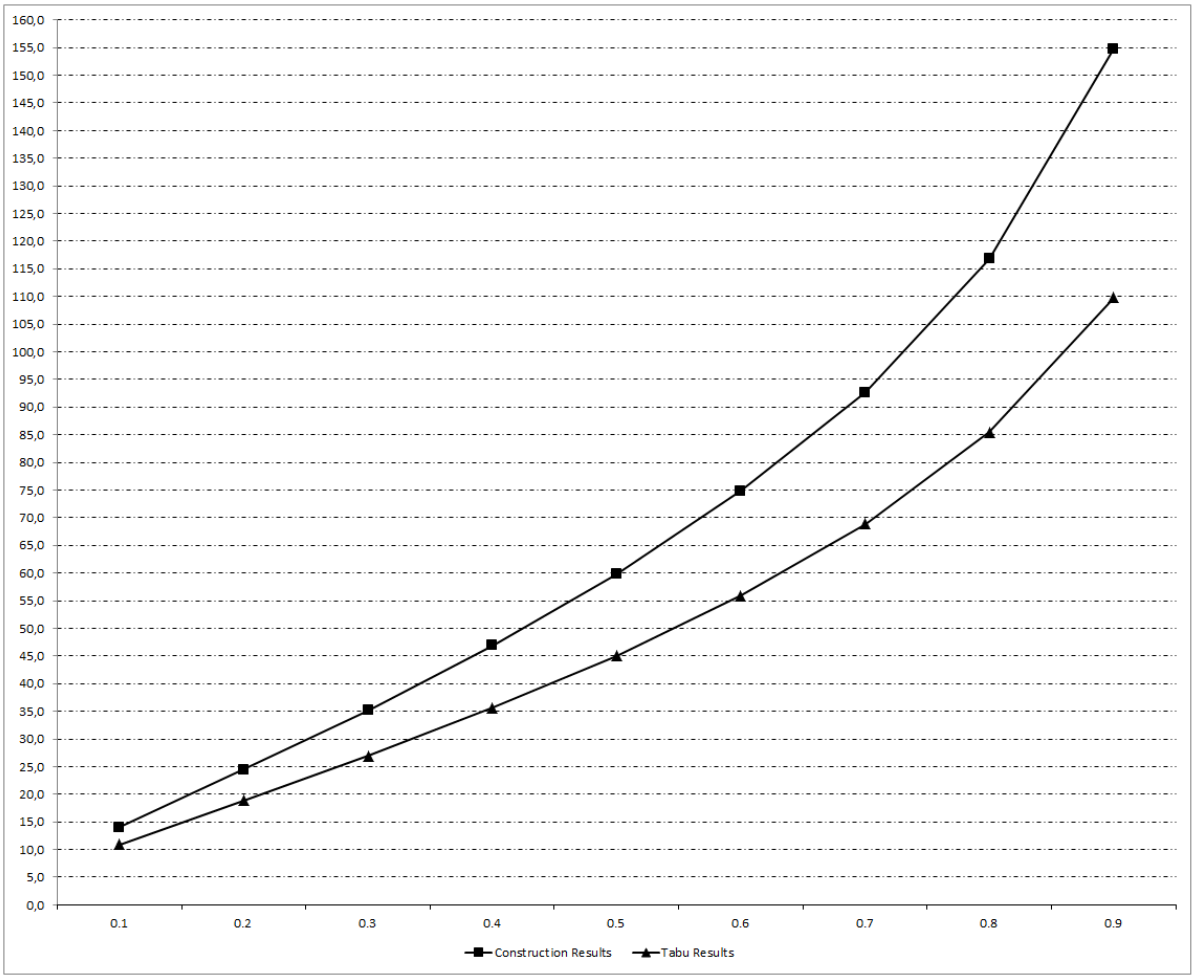

Figure 4.9. Construction vs SelImpCol Algorithm (0.1-0.9 Edge Density).

Table 4.8. Time results of SelImpCol Algorithm (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
500Cl1000Ver01Den	5.4	172.8
500Cl1000Ver02Den	16.9	165.9
500Cl1000Ver03Den	22.6	184.0
500Cl1000Ver04Den	53.9	230.1
500Cl1000Ver05Den	86.8	277.4
500Cl1000Ver06Den	109.7	302.0
500Cl1000Ver07Den	146.9	353.9
500Cl1000Ver08Den	184.7	369.8
500Cl1000Ver09Den	177.4	365.7

In Table 4.8, result and total times are increasing with density. For instance, tabu decreased onestepCD result by 3 colors for "500Cl1000Ver01Den" but it decreased onestepCD result by 45 colors. This is the reason of time differences.

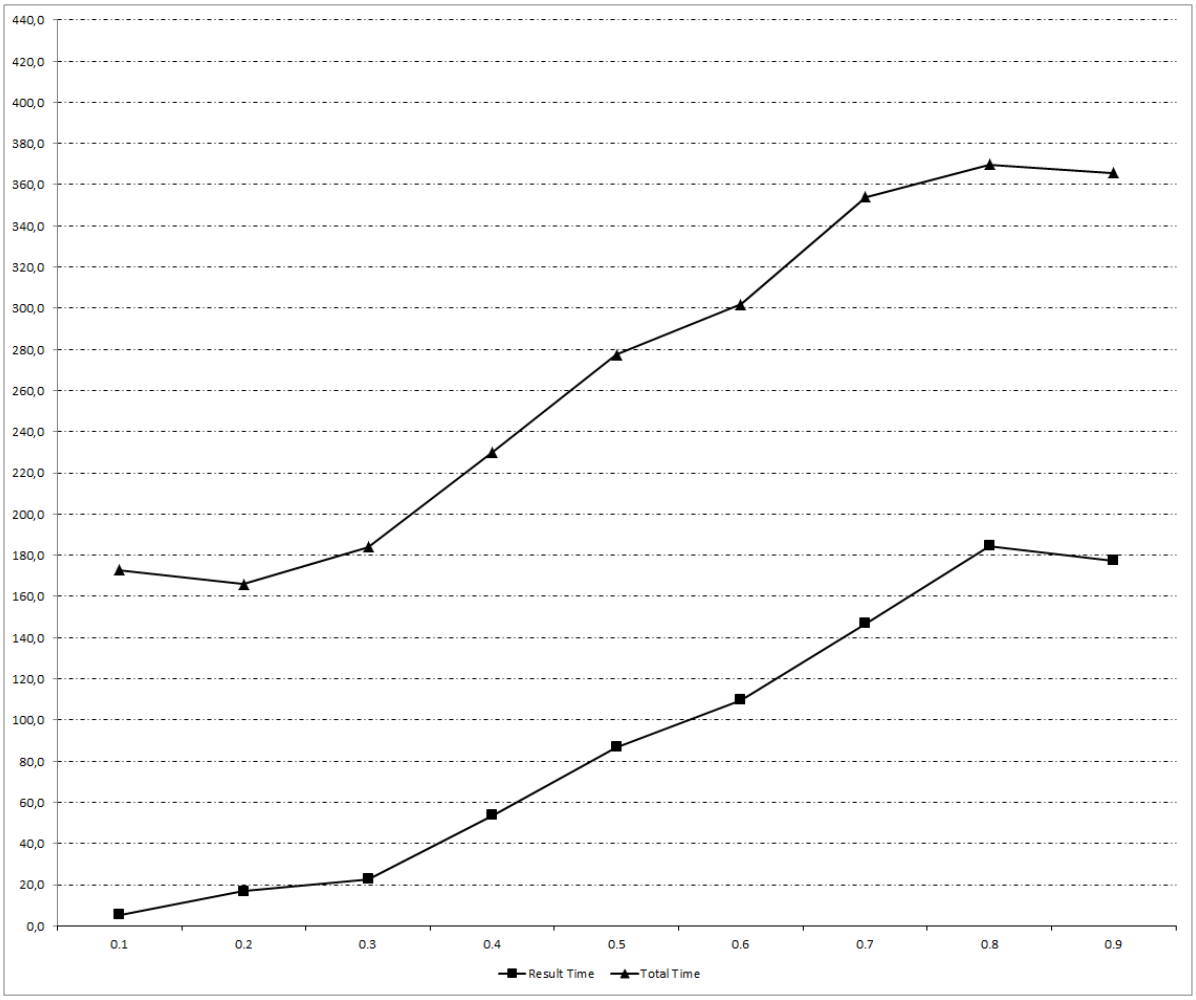

Figure 4.10. Time results of SelImpCol Algorithm (0.1-0.9 Edge Density).

Table 4.9. Construction vs SelImpCol Algorithm (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
100Cl200Ver05Den	16.4	12.0	26.8%
200Cl400Ver05Den	28.2	20.7	26.5%
300Cl600Ver05Den	39.4	29.1	26.2%
400Cl800Ver05Den	50.2	37.2	25.9%
500Cl1000Ver05Den	60.2	45.1	25.1%
600Cl1200Ver05Den	70.2	52.9	24.6%
700Cl1400Ver05Den	80.2	60.5	24.6%
800Cl1600Ver05Den	89.0	68.0	23.6%
900Cl1800Ver05Den	98.4	75.5	23.3%
1000Cl2000Ver05Den	107.8	82.8	23.2%

In Table 4.9, improvement percentage for small graphs is higher. Our stopping condition is total number of iterations and if we increase that value, we will expect to find better results for big graphs.

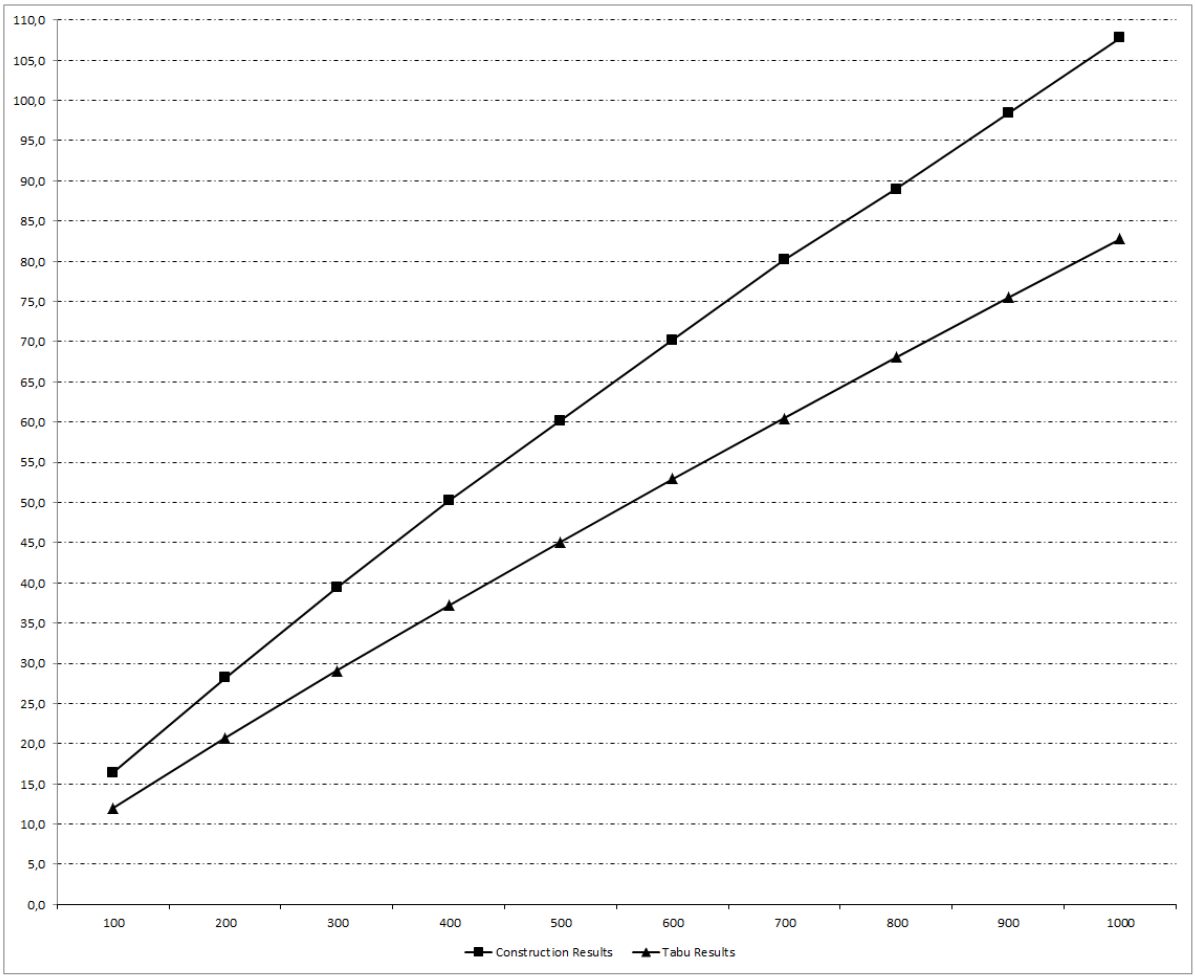

Figure 4.11. Construction vs SelImpCol Algorithm (100-1000 Number of Clusters).

Table 4.10. Time results of SelImpCol Algorithm (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
100Cl200Ver05Den	0.7	41.3
200Cl400Ver05Den	13.1	90.7
300Cl600Ver05Den	37.1	152.2
400Cl800Ver05Den	61.4	206.3
500Cl1000Ver05Den	92.8	281.3
600Cl1200Ver05Den	94.7	312.1
700Cl1400Ver05Den	154.0	414.1
800Cl1600Ver05Den	163.0	450.8
900Cl1800Ver05Den	213.5	556.2
1000Cl2000Ver05Den	282.4	664.3

In Table 4.10, time differences are due to the number of colors that SelImpCol improved.

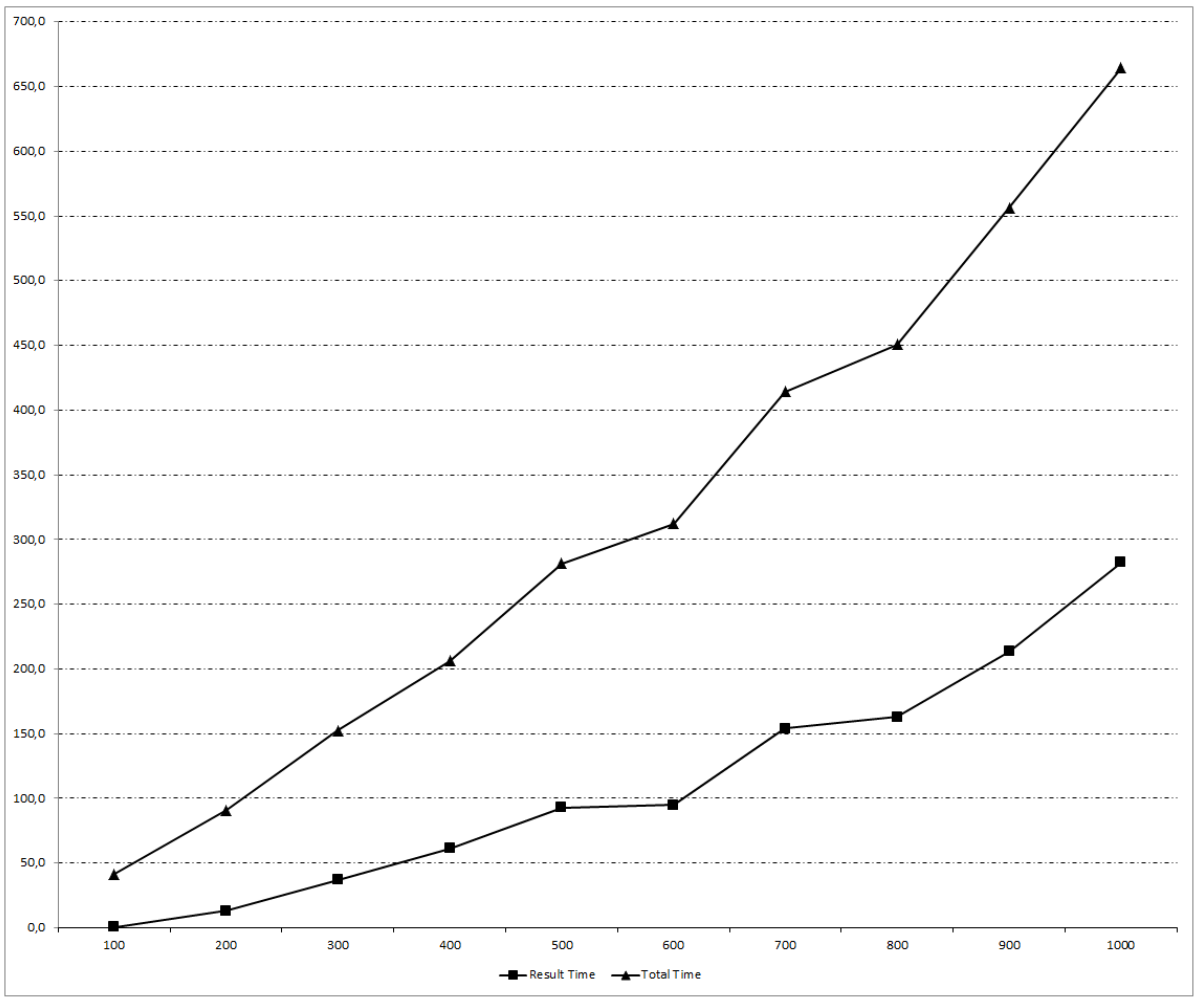

Figure 4.12. Time results of SelImpCol Algorithm (100-1000 Number of Clusters).

We have mentioned about our Exact Selective Coloring Algorithm for Interval Graphs in Section 3.2.1. We want to use this algorithm to find exact results for different interval graph instances and compare them to SelImpCol results for same instances to understand how far our algorithm from the exact result. Although interval graph is a special graph class, it can be helpful to see gaps between exact result, SelImpCol result and onestepCD result.

We generated two different sets of interval graphs. First set has 500 clusters, 1000 vertices and 9 different densities. For example, "Int500Cl1000Ver01Den" instance has 500 clusters, 1000 vertices and 0.1 edge density. Second set has a fixed density of (0.5) and fixed number of vertices in each cluster(2). Their cluster numbers are varied 100 to 1000 and there are 3 different instances. Each instance type has 3 different instances in it and SelImpCol runs 5 times for each instance. Eventually, average tabu results stand for 15 runs.

Average construction algorithm results, average tabu results, exact results, improvement percentage of tabu algorithm and gap between exact result and tabu result are given in Tables 4.11 and 4.13. Their plots are given in Figures 4.13 and 4.15.

In Table 4.11, it is clear that in partitioned interval graphs, onestepCD finds very good upper bounds for dense graphs and by the reason of there is a very small gap SelImpCol cannot decrease result of onestepCD significantly. In more sparse graphs, onestepCD cannot find good upper bounds and SelImpCol decreases the number of colors significantly.

Table 4.11. Construction vs SelImpCol vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improve-ment Percentage**	**Gap with Exact Result**
Int500Cl1000Ver01Den	29.0	22.0	20.7	24.1%	6.1%
Int500Cl1000Ver02Den	52.7	42.6	39.3	19.1%	7.7%
Int500Cl1000Ver03Den	73.3	61.2	56.3	16.5%	8.0%
Int500Cl1000Ver04Den	93.0	80.3	73.7	13.6%	8.3%
Int500Cl1000Ver05Den	104.7	96.5	91.3	7.8%	5.4%
Int500Cl1000Ver06Den	122.7	121.3	116.0	1.1%	4.3%
Int500Cl1000Ver07Den	157.7	155.7	151.3	1.2%	2.8%
Int500Cl1000Ver08Den	202.3	201.9	199.0	0.2%	1.5%
Int500Cl1000Ver09Den	272.0	271.2	270.3	0.3%	0.3%

In Table 4.12, we can see that SelImpCol finds results very fast. In more dense graphs, the gap between exact result and onestepCD is very small, therefore SelImpCol cannot improve results and its time nearly equals to 0.

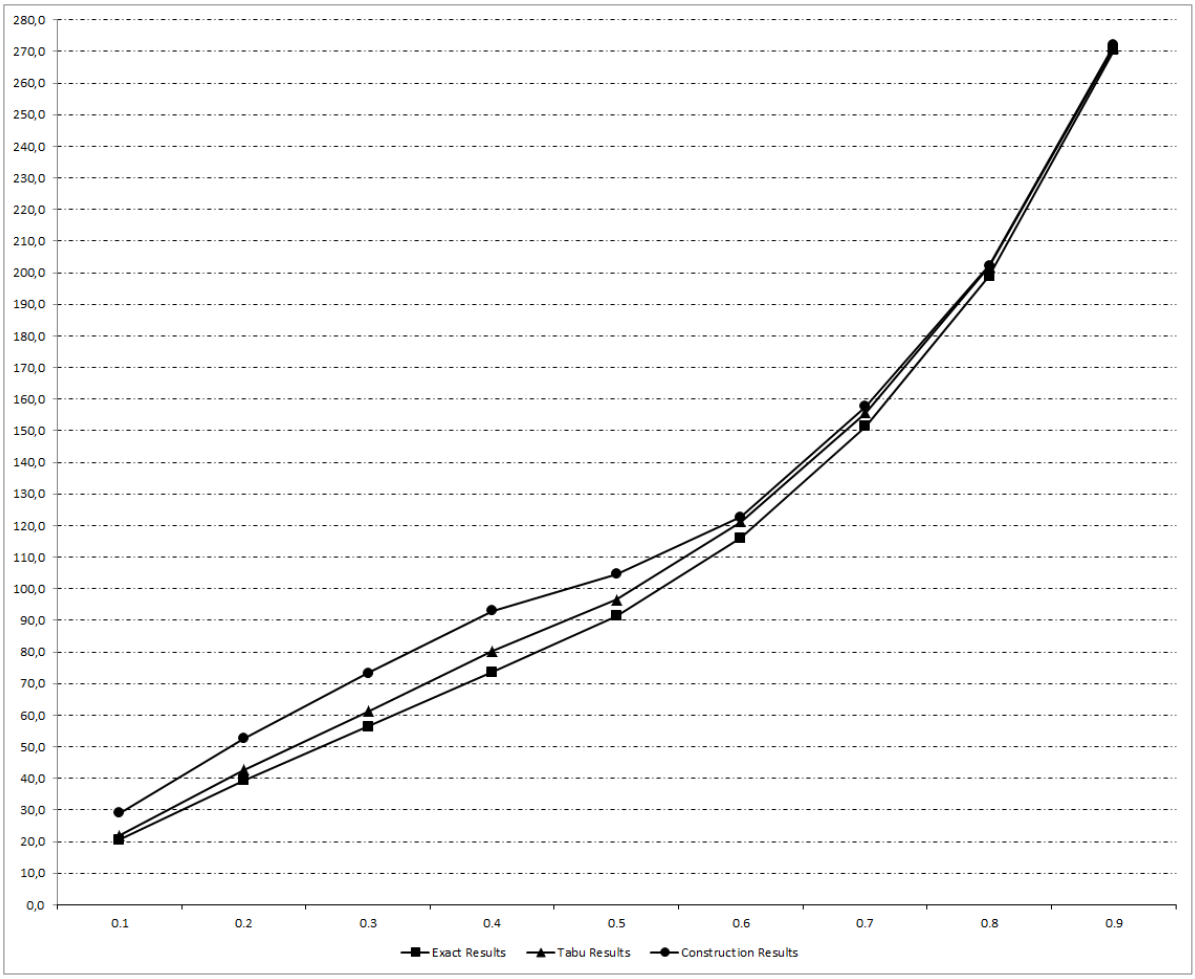

Figure 4.13. Construction vs SelImpCol vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.12. Time results of SelImpCol Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
Int500Cl1000Ver01Den	1.3	166.4
Int500Cl1000Ver02Den	2.9	191.1
Int500Cl1000Ver03Den	6.6	203.6
Int500Cl1000Ver04Den	7.8	217.3
Int500Cl1000Ver05Den	0.4	217.9
Int500Cl1000Ver06Den	0.6	215.3
Int500Cl1000Ver07Den	1.8	204.9
Int500Cl1000Ver08Den	0.0	207.9
Int500Cl1000Ver09Den	0.0	130.2

In Table 4.13, when graph size increases, improvement of SelImpCol decreased and gap with the exact result increased. It is based on the total number of iterations. Bigger graphs require more time to find better solutions.

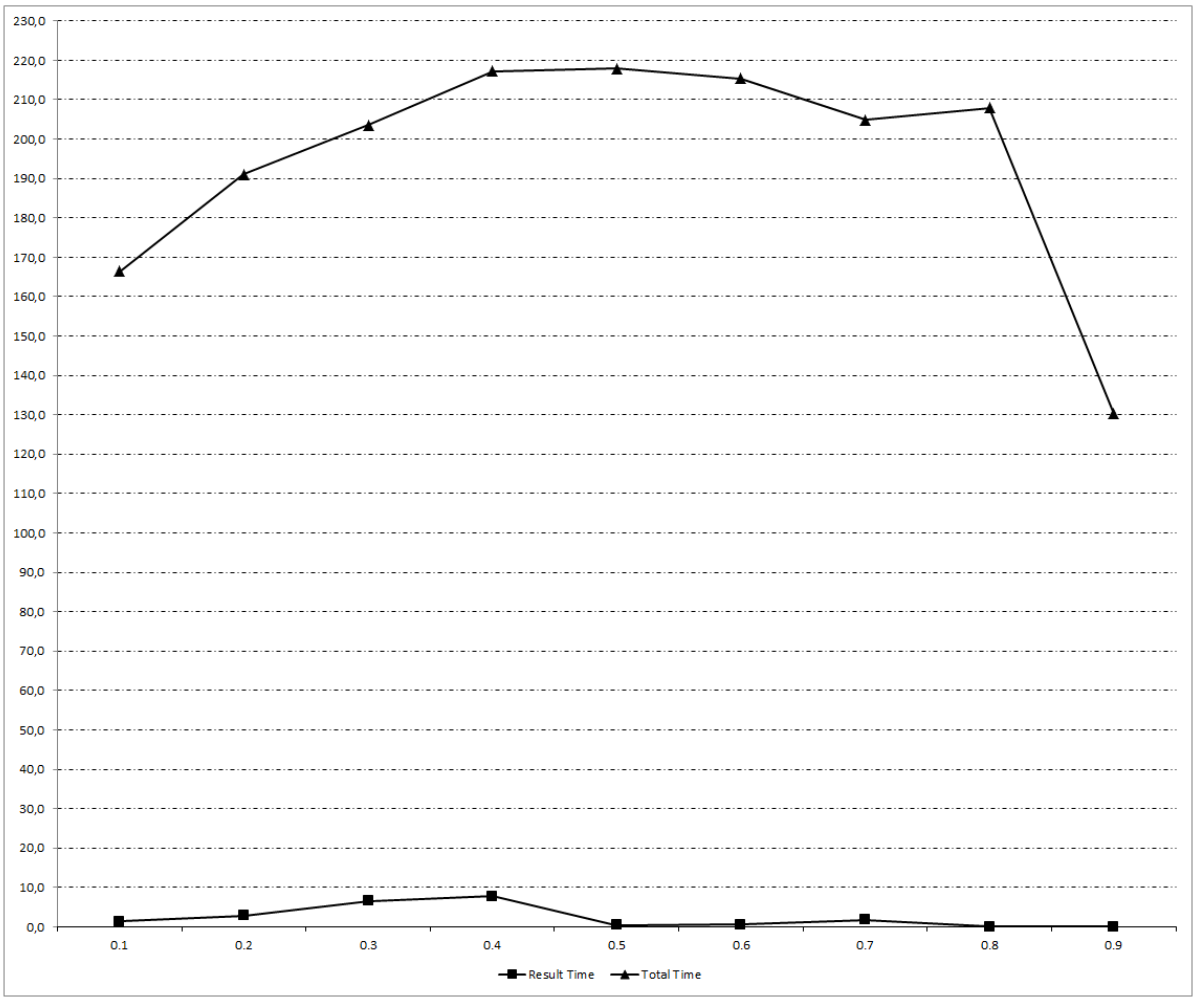

Figure 4.14. Time results of SelImpCol Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.13. Construction vs SelImpCol vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improve-ment Percentage**	**Gap with Exact Result**
Int100Cl200Ver05Den	23.7	21.2	20.7	10.4%	2.5%
Int200Cl400Ver05Den	45.7	41.5	39.7	9.2%	4.3%
Int300Cl600Ver05Den	66.0	61.1	57.3	7.4%	6.2%
Int400Cl800Ver05Den	87.7	80.8	76.3	7.8%	5.5%
Int500Cl1000Ver05Den	106.7	100.0	93.0	6.3%	7.0%
Int600Cl1200Ver05Den	127.7	116.1	108.0	9.1%	7.0%
Int700Cl1400Ver05Den	140.0	133.1	126.7	5.0%	4.8%
Int800Cl1600Ver05Den	167.3	157.3	146.7	6.0%	6.8%
Int900Cl1800Ver05Den	187.0	174.1	161.0	6.9%	7.5%
Int1000Cl2000Ver05Den	213.0	197.5	183.3	7.3%	7.2%

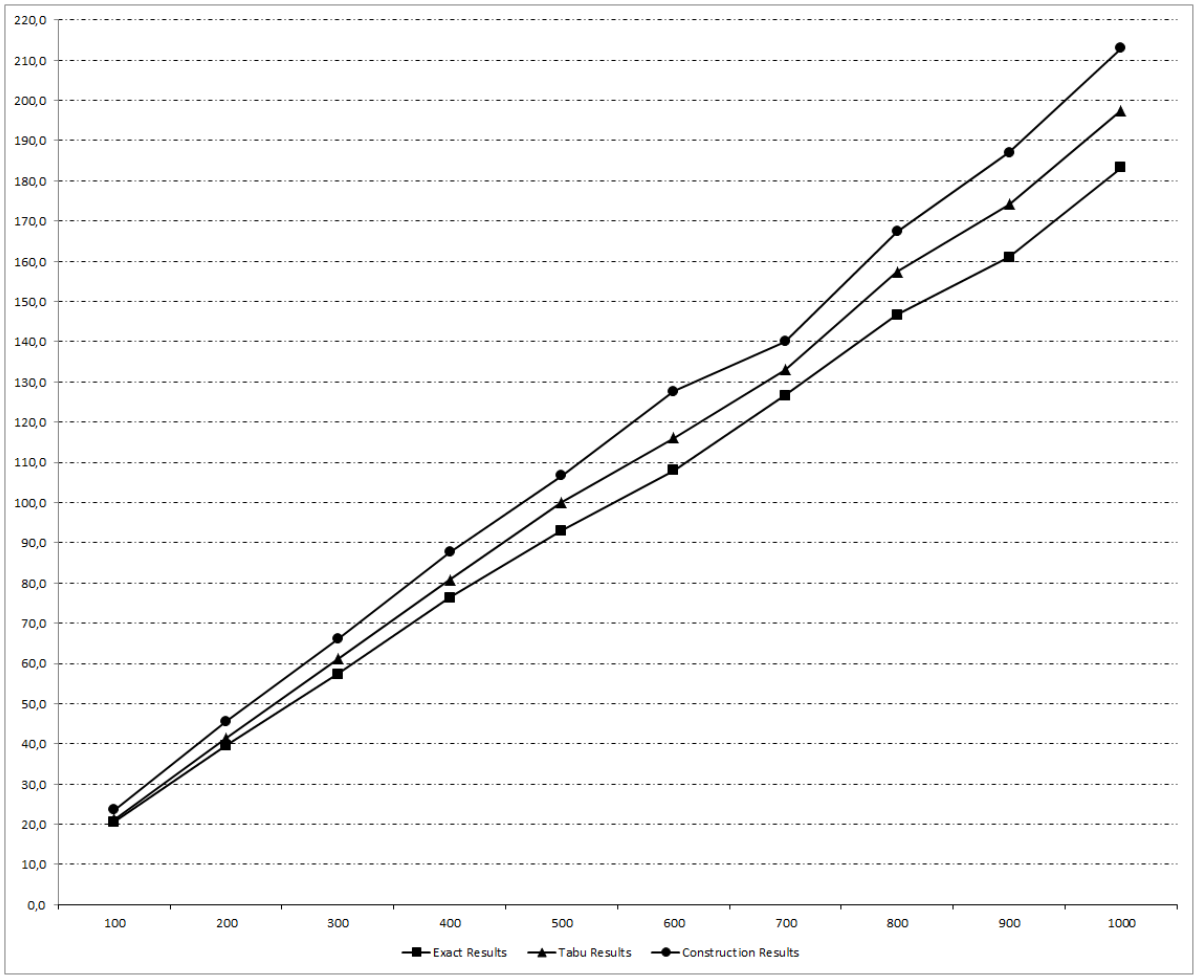

Figure 4.15. Construction vs SelImpCol vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

Table 4.14. Time results of SelImpCol Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
Int100Cl200Ver05Den	0.5	42.2
Int200Cl400Ver05Den	0.5	84.5
Int300Cl600Ver05Den	0.3	124.1
Int400Cl800Ver05Den	0.3	168.1
Int500Cl1000Ver05Den	5.1	215.0
Int600Cl1200Ver05Den	12.8	257.8
Int700Cl1400Ver05Den	0.2	290.3
Int800Cl1600Ver05Den	6.4	340.0
Int900Cl1800Ver05Den	9.2	385.4
Int1000Cl2000Ver05Den	13.9	441.1

In Table 4.14, increasing total time depending on the graph size is shown.

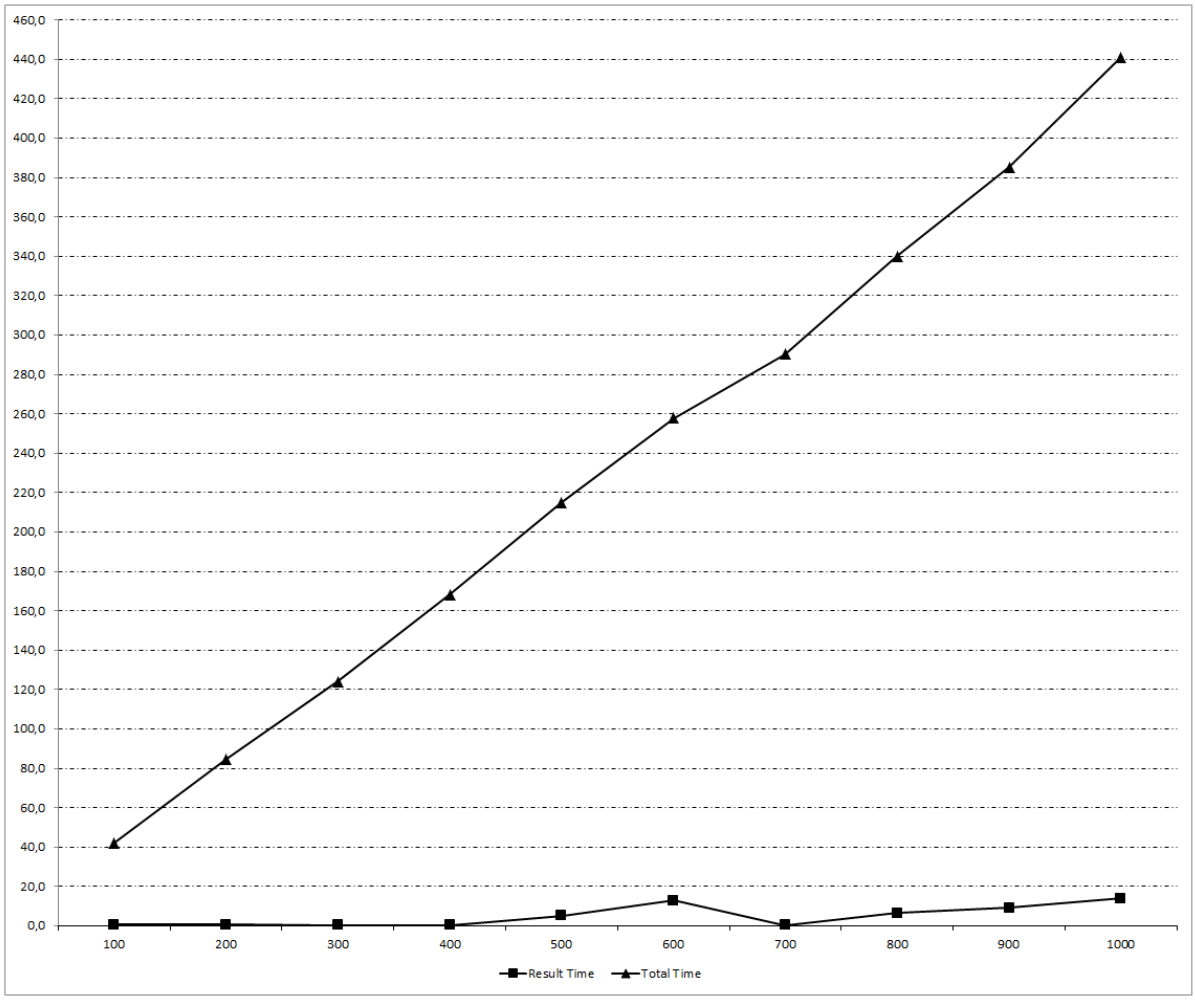

Figure 4.16. Time results of SelImpCol Algorithm in Interval Graphs (100-1000 Number of Clusters).

4.3.3. Selective Partial Coloring Algorithm-Cluster Number Based

We start the selective partial tabu search algorithm(SelParColNum) with $k = ub - 1$(obtained by Construction heuristics). When we find a feasible complete solution we decrease k by one. The procedure repeats until no feasible complete selective coloring is found using the given number of colors and it gives last feasible complete solution as solution.

In SelParColNum algorithm, we search whether the clustered graph admits a selective coloring using k colors or not. To this purpose, for each k we should create k different color groups. These groups are named as color classes. We create a main pool($\mathcal{O}$) to keep uncolored clusters. In other words, if a cluster does not belong to any color classes, it will be in $\mathcal{O}$. Visualization of SelParColNum algorithm is given in Figure 4.17,

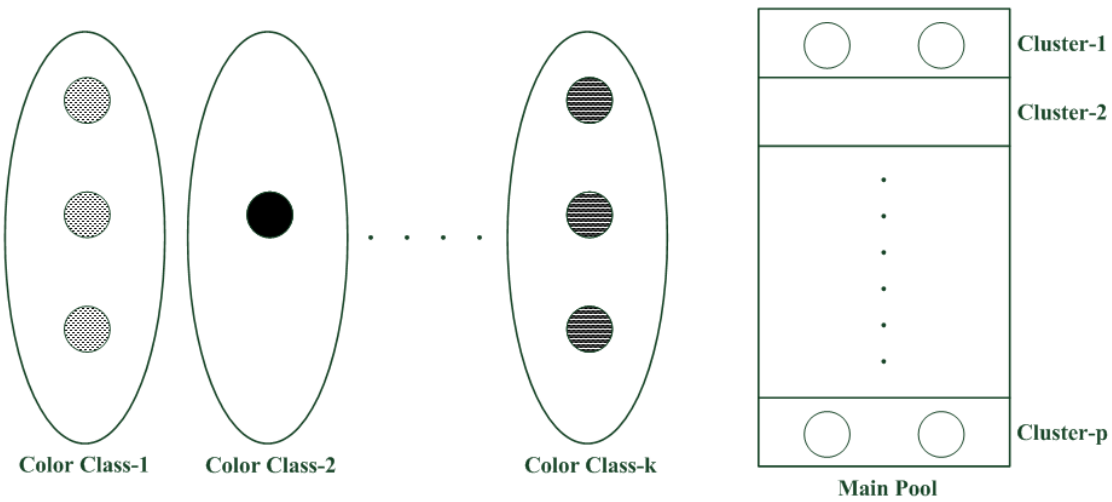

Figure 4.17. SelParCol Algorithm Visualization.

Our objective is to empty out the main pool. For this purpose, we try to color all clusters by selecting exactly 1 cluster in each iteration and color one of its vertex. When a vertex is taken from the main pool and is assigned to one color group, the cluster of this vertex is removed from the main pool. The minimization of the number of clusters in the main pool is our objective function and when it is 0, it means that the clustered graph admits a selective coloring by k colors. When this is the case, we

decrease k by 1 and repeat the procedure. For computational efficiency, after we find k colorings, we delete the color class which has fewest vertices(ties are broken randomly) and return their clusters to the pool and start to search for $k-1$ colorings from these filled color classes.

Any two vertices in a color group cannot have an edge between them because each color group should be a stable set, otherwise it will create a conflict. In each iteration a move is effectuated. A move can be defined in several steps. A candidate cluster is chosen from the main pool. There are five different cluster selection methods that we implemented.

First one is *Big Cluster-Minimum Out-Degree Method*. Out-degree of a vertex is the number of edges with vertices in other clusters. Out-degree of a cluster is the total out-degree of its vertices . This method seeks for the biggest cluster in the pool, if there is a tie, it selects the cluster with the minimum out-degree.

Second one is *Small Cluster-Minimum Out-Degree Method*. This method seeks for the smallest cluster in the pool, if there is a tie, it selects the cluster with the minimum out-degree.

Third one is *Random Cluster Method*. This method selects a cluster from the pool arbitrarily.

Fourth one is *Hybrid of Small Cluster-Minimum Out-Degree and Random Cluster*. This method is a mix of Small Cluster-Minimum Degree Method and Random Cluster Method. It selects the cluster by Small Cluster-Minimum Degree Method with the probability of 0.2 and Random Cluster Method with the probability of 0.8.

Last one is *Hybrid of Big Cluster-Minimum Degree and Random Cluster*. This method is a mix of Big Cluster-Minimum Degree Method and Random Cluster Method. It selects the cluster by Big Cluster-Minimum Degree Method with the probability of p and Random Cluster Method with the probability of $1-p$.

These different selection methods were compared and best one is the *Random Cluster Method*. Other methods did not improved results of test instances. Therefore, *Random Cluster Method* is used as a cluster selection method in our computational studies.

After candidate cluster selection phase, we should determine which vertex from the candidate cluster is assigned to which color class. We implemented a different approach compared to common vertex coloring tabu algorithms. In common vertex coloring tabu algorithm a candidate vertex is determined and the algorithm searches for best color class it will be assigned to. In our approach, we select a candidate cluster and try to find the best vertex-color class pair depending on the candidate cluster.

There is a control of the objective function. Objective function is total number of uncolored clusters. There is a variable which stores best objective function value. If new solution is better than the best solution, then the move is executed and search for vertex-color class pair ends. In this situation, it is not important whether the move is tabu or not. This phenomenon is called *aspiration criterion*. Else, the algorithm checks whether the new solution is better than the current solution or not. If it is better and the move is not in the tabu list, then the move is performed and search for vertex-color class pair ends. By the reason of our current objective value can only be decreased by 1 in an iteration, search ends when a better objective function value is found. Termination of search for vertex-color class pair is important for computational efficiency. If no better objective function value is found than the current objective value in all possibilities, the algorithm selects vertex-color class pair with best objective value among all possibilities. In this case, our objective function value either remains unchanged or increases.

When the move is made, candidate vertex and the vertices in the same cluster are removed from the main pool, because two or more vertices from the same cluster cannot be colored at the same time. Similarly, conflicted vertices with the candidate vertex are removed from the related color class and put back to the main pool to maintain feasibility of the partial solution. Note that the vertices that are in the same

cluster with conflicted vertices are also put back to the main pool.

An example move is shown in Figure 4.18 and Figure 4.19.

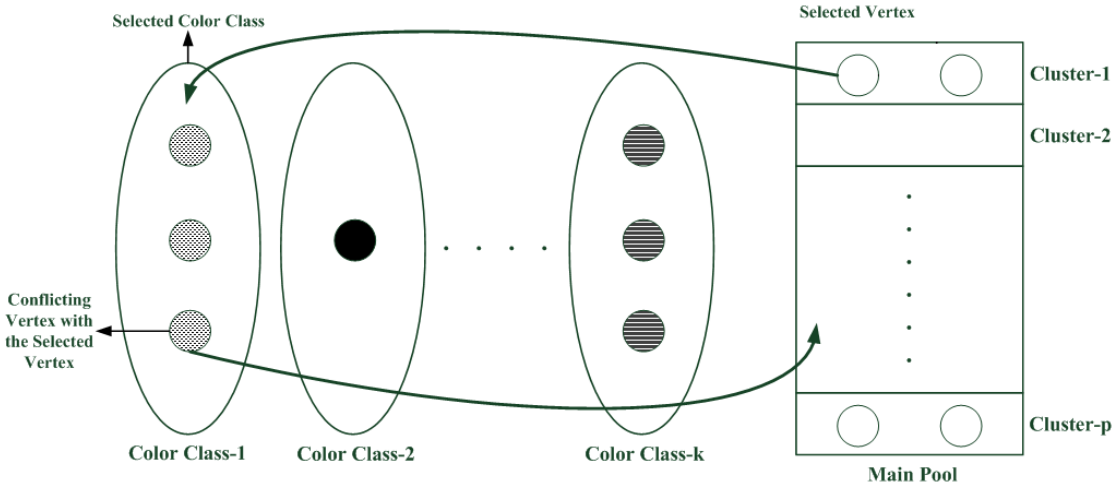

Figure 4.18. Before a move.

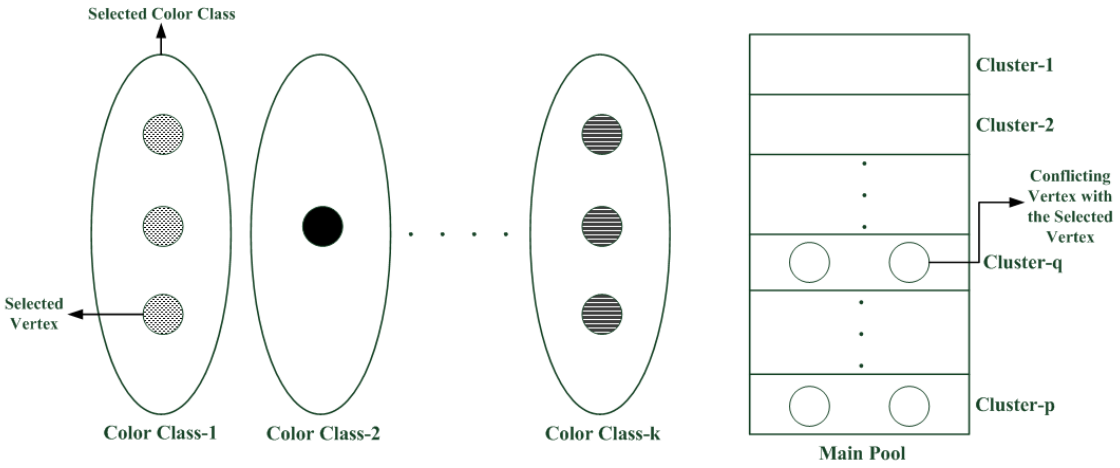

Figure 4.19. After a move.

Tabu list stores the vertices which conflicts with the selected vertex in selected color class. These vertices are put back to the main pool to prevent conflicts. Tabu list inhibits these vertices to turn to the previous color class for t iterations. Tabu list contains the vertices which are put back to the main pool, the color class from which the conflicted vertices are taken and the number of iterations they are inhibited to turn to the previous color class.

Tabu list has a reactive length denoted by t. It is calculated again in every b

iterations. Δ is the difference between the maximum and the minimum of objective function during the last b iterations. If $\Delta \leq$ *thresholdValue* then $t = t + B$, and otherwise $t = t - 1$. B and b are fixed parameters [15].

SelParColNum algorithm for the selective graph coloring problem operates as we mentioned above. The algorithm is summarized in Algorithm 4.20.

Input: Clustered Graph Instance, Construction Solution with k colors, b, B, t, *thresholdValue*;

Output: Tabu Solution;

Build a new starting solution by deleting color class with minimum number of vertices in last solution and add clusters of deleted vertices to $\mathcal{O}$;

Free the tabu list;

Set *iteration* := 0 *currFS* := $|\mathcal{O}|$ *bestFS* := $|\mathcal{O}|$ $k := k - 1$;

repeat

Select a cluster V_i from $\mathcal{O}$ arbitrarily and update $\mathcal{O} := \mathcal{O} \setminus V_i$;

Find best vertex-color class pair and assign best vertex to best color class;

If best vertex has any conflicts with other vertices in best color class, remove them from the color class and add their clusters to $\mathcal{O}$;

Update *currFS*;

if *currFS* < *bestFS* **then**

bestFS := *currFS*

end if

Update tabu tenure with reactive tabu tenure algorithm

until *iteration* < *maxIteration* and *bestFS* $\neq 0$

Figure 4.20. SelParColNum Algorithm.

4.3.4. Computational Results of SelParColNum Algorithm

First of all, we should find a suitable parameter set for reactive tabu scheme. There are 4 different parameters(b, B, C, $thresholdvalue$) that we can fix to find a good t. We fixed $b = 1000$ and $C = 1$ and searched good parameter values for B and $thresholdvalue$.

We used same "DSJC500.5-1", "DSJC500.5-2", "DSJC500.5-3" and "DSJC500.5-4" instances to set our parameter values.

We tried a wide range of parameter values but most of them do not give good results. We showed significant results of parameter tuning experiment for SelParColNum algorithm in Tables 4.15, 4.16, 4.17 and 4.18. SelParColNum algorithm is runned 10 times for all different parameter sets for each graph. Maximum iteration number of these instances is 3.000.000.

Table 4.15. SelParColNum Parameter Tuning (DSJC500.5-1).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-1	65	51	51.7	52	5	1
DSJC500.5-1	65	51	52.5	54	5	3
DSJC500.5-1	65	52	52.4	53	5	5
DSJC500.5-1	65	**50**	**51.2**	**52**	7	1
DSJC500.5-1	65	52	52.2	53	7	3
DSJC500.5-1	65	52	52.0	52	7	5
DSJC500.5-1	65	52	52.0	52	10	1
DSJC500.5-1	65	52	52.4	53	10	3
DSJC500.5-1	65	52	52.8	53	10	5

Table 4.16. SelParColNum Parameter Tuning (DSJC500.5-2).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-2	60	45	45.8	46	5	1
DSJC500.5-2	60	45	46.0	47	5	3
DSJC500.5-2	60	46	46.6	47	5	5
DSJC500.5-2	59	**45**	**45.5**	**46**	7	1
DSJC500.5-2	60	46	46.2	47	7	3
DSJC500.5-2	60	46	46.4	47	7	5
DSJC500.5-2	59	45	45.8	46	10	1
DSJC500.5-2	60	46	46.8	47	10	3
DSJC500.5-2	60	46	46.8	47	10	5

Table 4.17. SelParColNum Parameter Tuning (DSJC500.5-3).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-3	57	43	43.6	44	5	1
DSJC500.5-3	57	43	43.4	44	5	3
DSJC500.5-3	57	44	44.6	45	5	5
DSJC500.5-3	57	**43**	**43.1**	**44**	7	1
DSJC500.5-3	57	44	44.0	44	7	3
DSJC500.5-3	57	44	44.0	44	7	5
DSJC500.5-3	57	43	43.5	44	10	1
DSJC500.5-3	57	44	44.4	45	10	3
DSJC500.5-3	57	44	44.2	45	10	5

Table 4.18. SelParColNum Parameter Tuning (DSJC500.5-4).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-4	55	41	42.1	43	5	1
DSJC500.5-4	55	42	42.5	43	5	3
DSJC500.5-4	55	42	42.4	43	5	5
DSJC500.5-4	55	**41**	**41.7**	**42**	7	1
DSJC500.5-4	55	42	42.2	43	7	3
DSJC500.5-4	55	42	42.6	43	7	5
DSJC500.5-4	55	41	41.8	42	10	1
DSJC500.5-4	55	42	42.6	43	10	3
DSJC500.5-4	55	42	42.4	43	10	5

After these experiments, we decided to use $B = 1$ and $thresholdvalue = 7$ for our further experiments because in average that parameter set gives better results.

We used same instance sets that are used in Section 4.3.2. Each instance type has 10 different instances in it and SelParColNum runs 5 times for each instance. Eventually, average tabu results stand for 50 runs.

Average construction algorithm results, average tabu results and improvement percentage between them are given in Tables 4.19 and 4.21. Their plots are given in Figures 4.21 and 4.23.

Table 4.19. Construction vs SelParColNum Algorithm (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
500Cl1000Ver01Den	14.0	11.0	21.4%
500Cl1000Ver02Den	24.5	19.0	22.4%
500Cl1000Ver03Den	35.2	27.1	23.0%
500Cl1000Ver04Den	46.9	36.2	22.9%
500Cl1000Ver05Den	59.8	46.0	23.1%
500Cl1000Ver06Den	74.9	57.3	23.5%
500Cl1000Ver07Den	92.7	70.3	24.2%
500Cl1000Ver08Den	116.8	87.1	25.3%
500Cl1000Ver09Den	154.8	114.3	26.2%

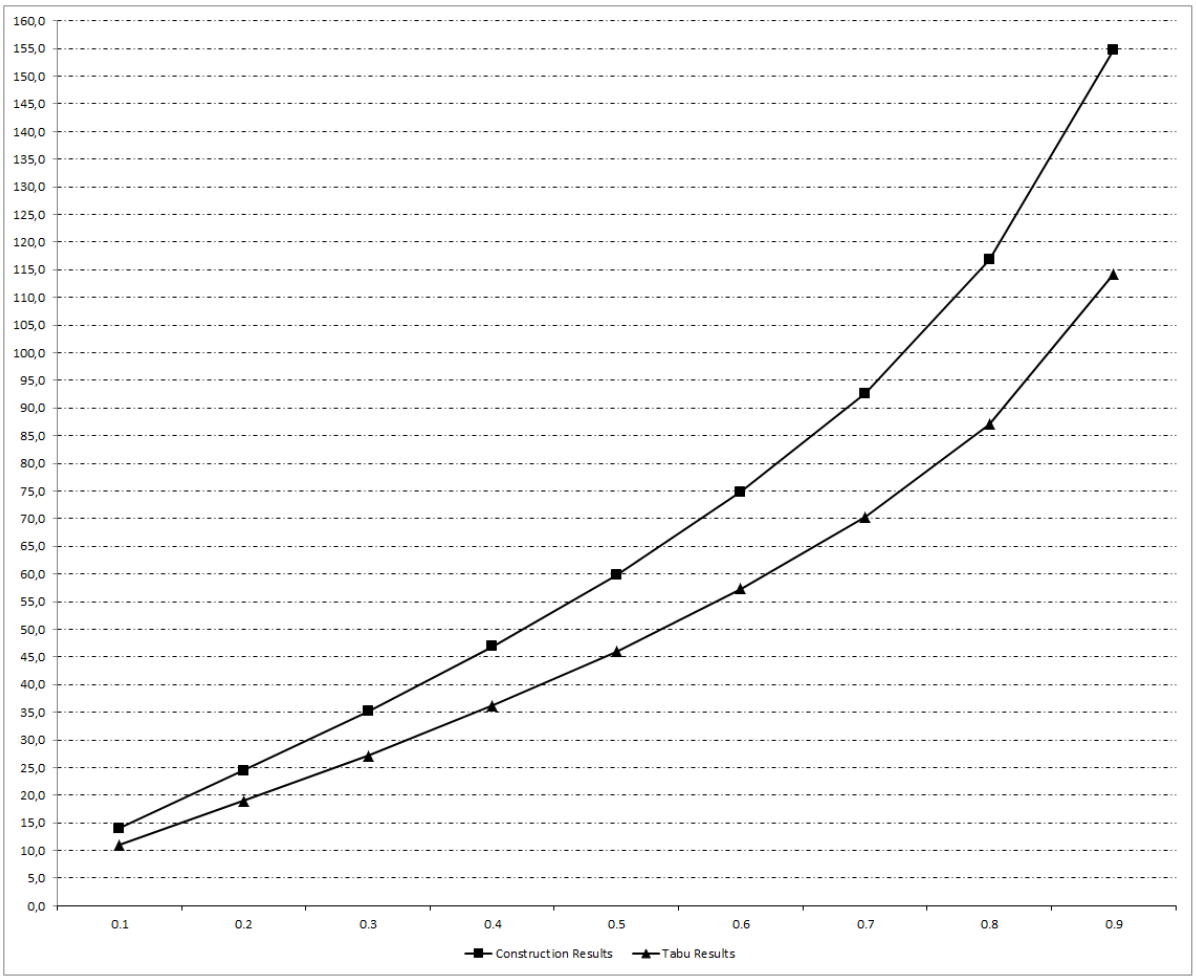

Figure 4.21. Construction vs SelParColNum Algorithm (0.1-0.9 Edge Density).

Table 4.20. Time results of SelParColNum Algorithm (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
500Cl1000Ver01Den	1.0	70.0
500Cl1000Ver02Den	6.1	78.5
500Cl1000Ver03Den	30.7	108.3
500Cl1000Ver04Den	42.9	126.0
500Cl1000Ver05Den	42.7	130.7
500Cl1000Ver06Den	38.8	126.7
500Cl1000Ver07Den	72.5	160.2
500Cl1000Ver08Den	99.3	194.3
500Cl1000Ver09Den	112.2	213.1

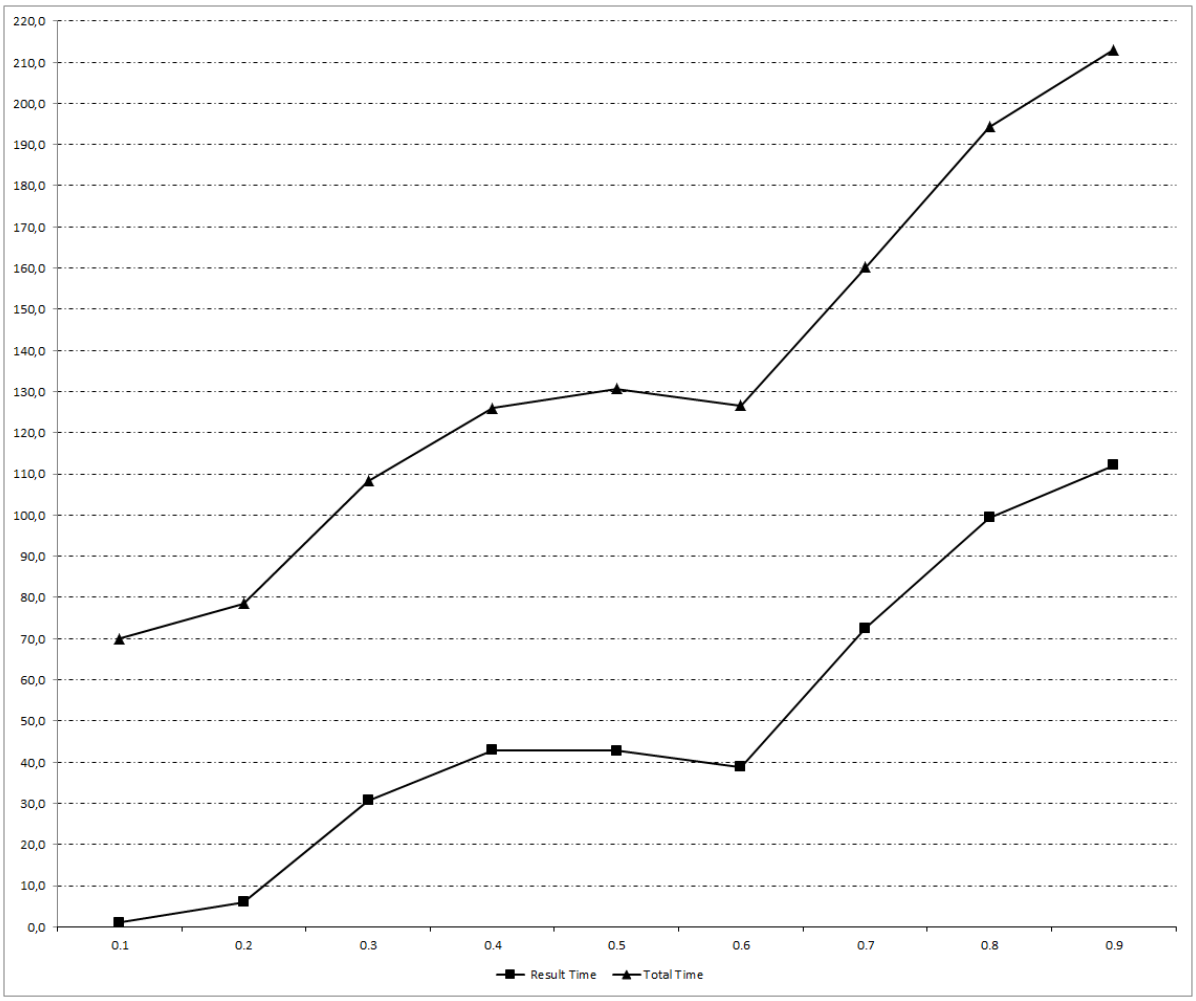

Figure 4.22. Time results of SelParColNum Algorithm (0.1-0.9 Edge Density).

Table 4.21. Construction vs SelParColNum Algorithm (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
100Cl200Ver05Den	16.4	12.0	26.8%
200Cl400Ver05Den	28.2	21.0	25.5%
300Cl600Ver05Den	39.4	29.8	24.4%
400Cl800Ver05Den	50.2	38.0	24.4%
500Cl1000Ver05Den	60.2	46.0	23.6%
600Cl1200Ver05Den	70.2	53.8	23.3%
700Cl1400Ver05Den	80.2	61.7	23.0%
800Cl1600Ver05Den	89.0	69.5	22.0%
900Cl1800Ver05Den	98.4	76.8	21.9%
1000Cl2000Ver05Den	107.8	84.4	21.7%

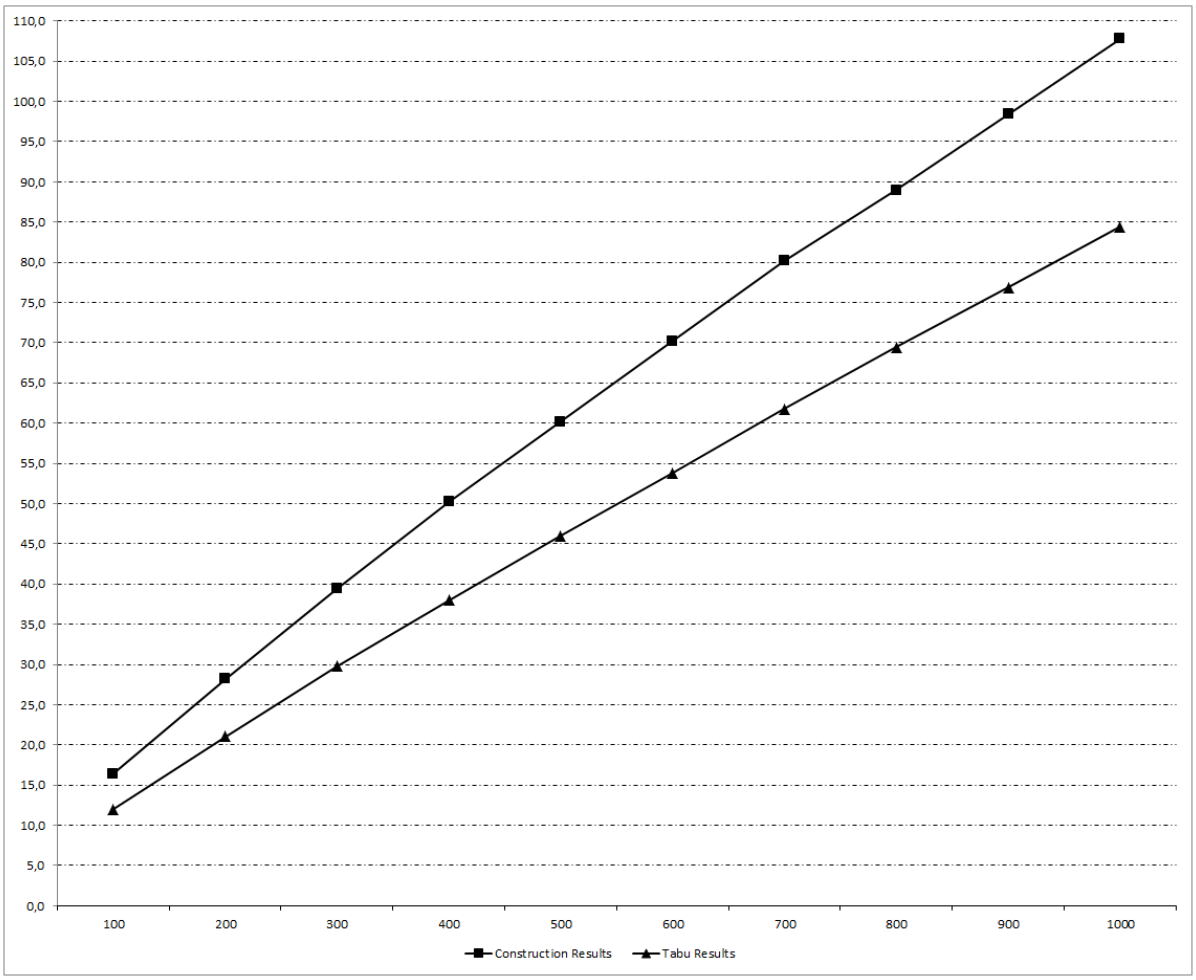

Figure 4.23. Construction vs SelParColNum Algorithm (100-1000 Number of Clusters).

Table 4.22. Time results of SelParColNum Algorithm (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
100Cl200Ver05Den	1.0	19.3
200Cl400Ver05Den	8.2	44.4
300Cl600Ver05Den	12.5	67.2
400Cl800Ver05Den	32.2	103.1
500Cl1000Ver05Den	45.1	135.7
600Cl1200Ver05Den	72.0	188.1
700Cl1400Ver05Den	72.0	199.2
800Cl1600Ver05Den	86.2	231.4
900Cl1800Ver05Den	116.3	282.2
1000Cl2000Ver05Den	151.6	365.9

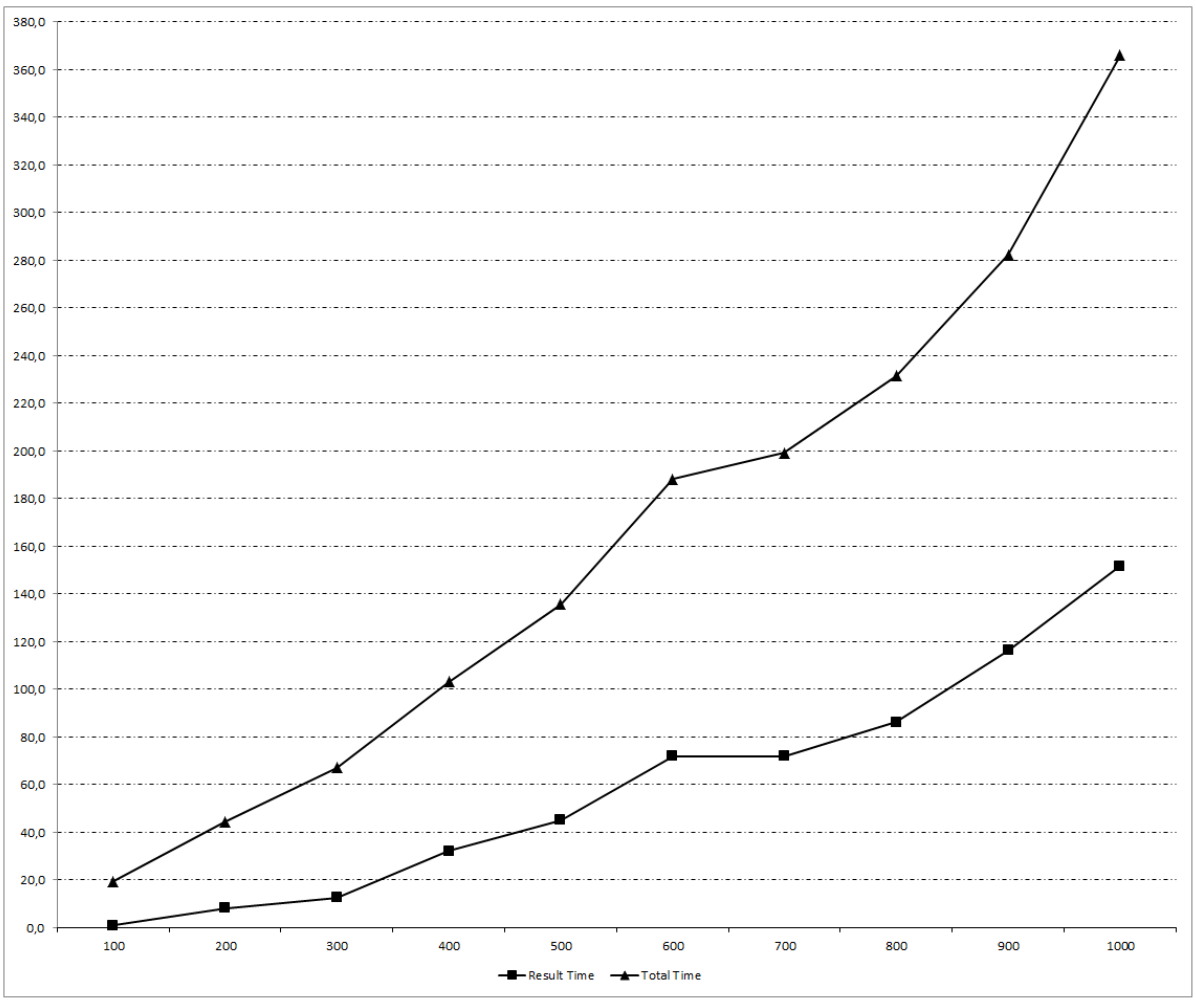

Figure 4.24. Time results of SelParColNum Algorithm (100-1000 Number of Clusters).

We used same interval instance sets in Section 4.3.2. Each instance type has 3 different instances in it and SelParColNum runs 5 times for each instance. Eventually, average tabu results stand for 15 runs.

Average construction algorithm results, average tabu results, exact results, improvement percentage of tabu algorithm and gap between exact result and tabu result are given in Tables 4.23 and 4.25. Their plots are given in Figures 4.25 and 4.27.

Table 4.23. Construction vs SelParColNum vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improvement Percentage**	**Gap with Exact Result**
Int500Cl1000Ver01Den	29.0	22.0	20.7	24.1%	6.1%
Int500Cl1000Ver02Den	52.7	42.4	39.3	19.5%	7.2%
Int500Cl1000Ver03Den	73.3	61.3	56.3	16.5%	8.1%
Int500Cl1000Ver04Den	93.0	80.4	73.7	13.5%	8.4%
Int500Cl1000Ver05Den	104.7	96.3	91.3	8.0%	5.1%
Int500Cl1000Ver06Den	122.7	121.3	116.0	1.1%	4.4%
Int500Cl1000Ver07Den	157.7	155.7	151.3	1.3%	2.8%
Int500Cl1000Ver08Den	202.3	201.8	199.0	0.3%	1.4%
Int500Cl1000Ver09Den	272.0	271.1	270.3	0.3%	0.3%

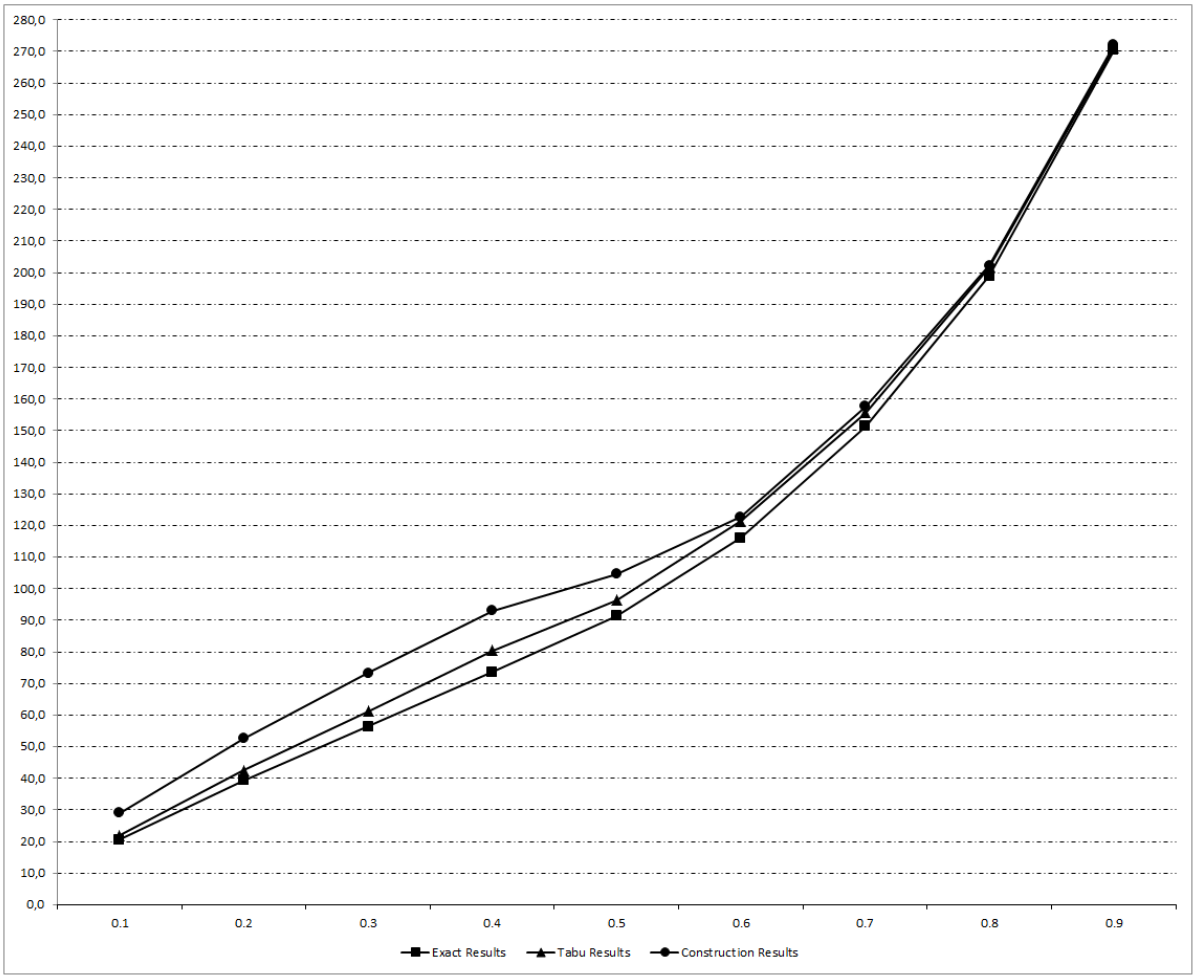

Figure 4.25. Construction vs SelParColNum vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.24. Time results of SelParColNum Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
Int500Cl1000Ver01Den	1.0	71.6
Int500Cl1000Ver02Den	1.5	80.8
Int500Cl1000Ver03Den	3.0	94.3
Int500Cl1000Ver04Den	6.3	104.4
Int500Cl1000Ver05Den	1.6	103.4
Int500Cl1000Ver06Den	2.0	108.6
Int500Cl1000Ver07Den	5.3	117.9
Int500Cl1000Ver08Den	0.2	120.1
Int500Cl1000Ver09Den	0.1	134.5

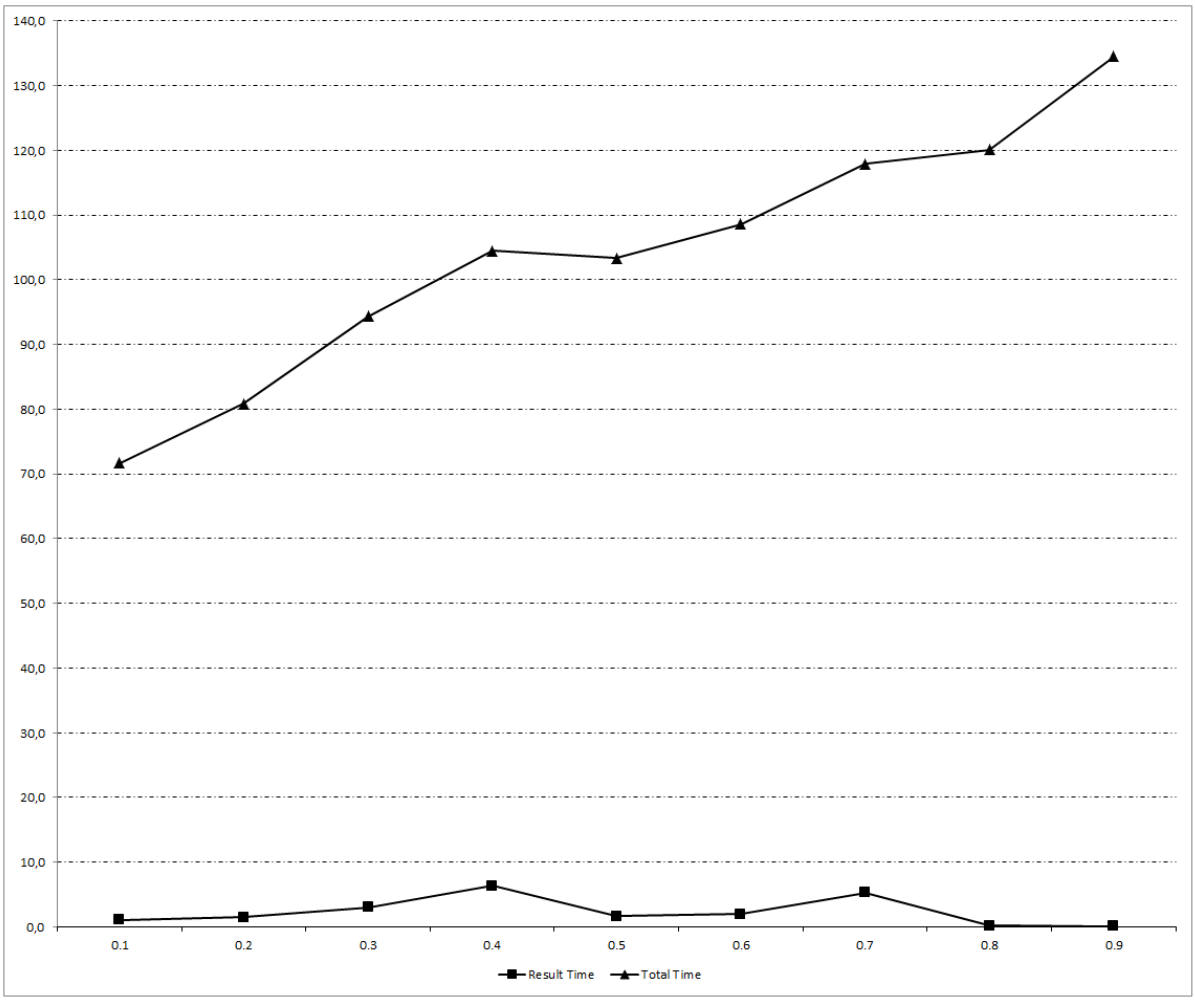

Figure 4.26. Time results of SelParColNum Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.25. Construction vs SelParColNum vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improve-ment Percentage**	**Gap with Exact Result**
Int100Cl200Ver05Den	23.7	21.3	20.7	9.9%	3.1%
Int200Cl400Ver05Den	45.7	41.3	39.7	9.6%	3.9%
Int300Cl600Ver05Den	66.0	60.7	57.3	8.0%	5.6%
Int400Cl800Ver05Den	87.7	80.5	76.3	8.2%	5.1%
Int500Cl1000Ver05Den	106.7	99.6	93.0	6.6%	6.6%
Int600Cl1200Ver05Den	127.7	115.8	108.0	9.3%	6.7%
Int700Cl1400Ver05Den	140.0	132.8	126.7	5.1%	4.6%
Int800Cl1600Ver05Den	167.3	157.0	146.7	6.2%	6.6%
Int900Cl1800Ver05Den	187.0	174.3	161.0	6.8%	7.6%
Int1000Cl2000Ver05Den	213.0	197.6	183.3	7.2%	7.2%

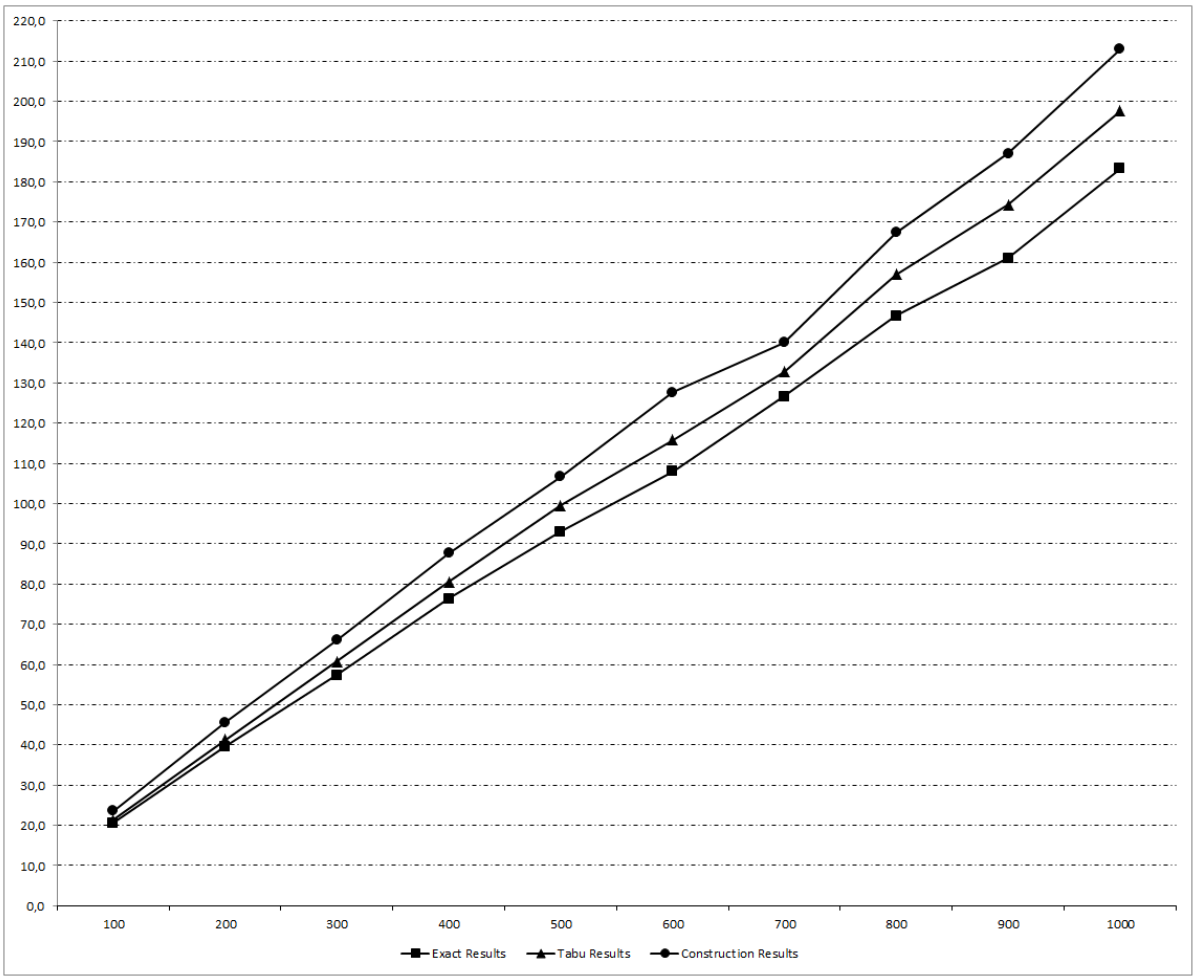

Figure 4.27. Construction vs SelParColNum vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

Table 4.26. Time results of SelParColNum Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
Int100Cl200Ver05Den	1.2	22.8
Int200Cl400Ver05Den	0.2	42.9
Int300Cl600Ver05Den	1.2	61.6
Int400Cl800Ver05Den	0.4	87.8
Int500Cl1000Ver05Den	2.6	115.3
Int600Cl1200Ver05Den	20.8	147.2
Int700Cl1400Ver05Den	0.5	155.3
Int800Cl1600Ver05Den	6.4	180.0
Int900Cl1800Ver05Den	3.8	200.2
Int1000Cl2000Ver05Den	9.0	232.1

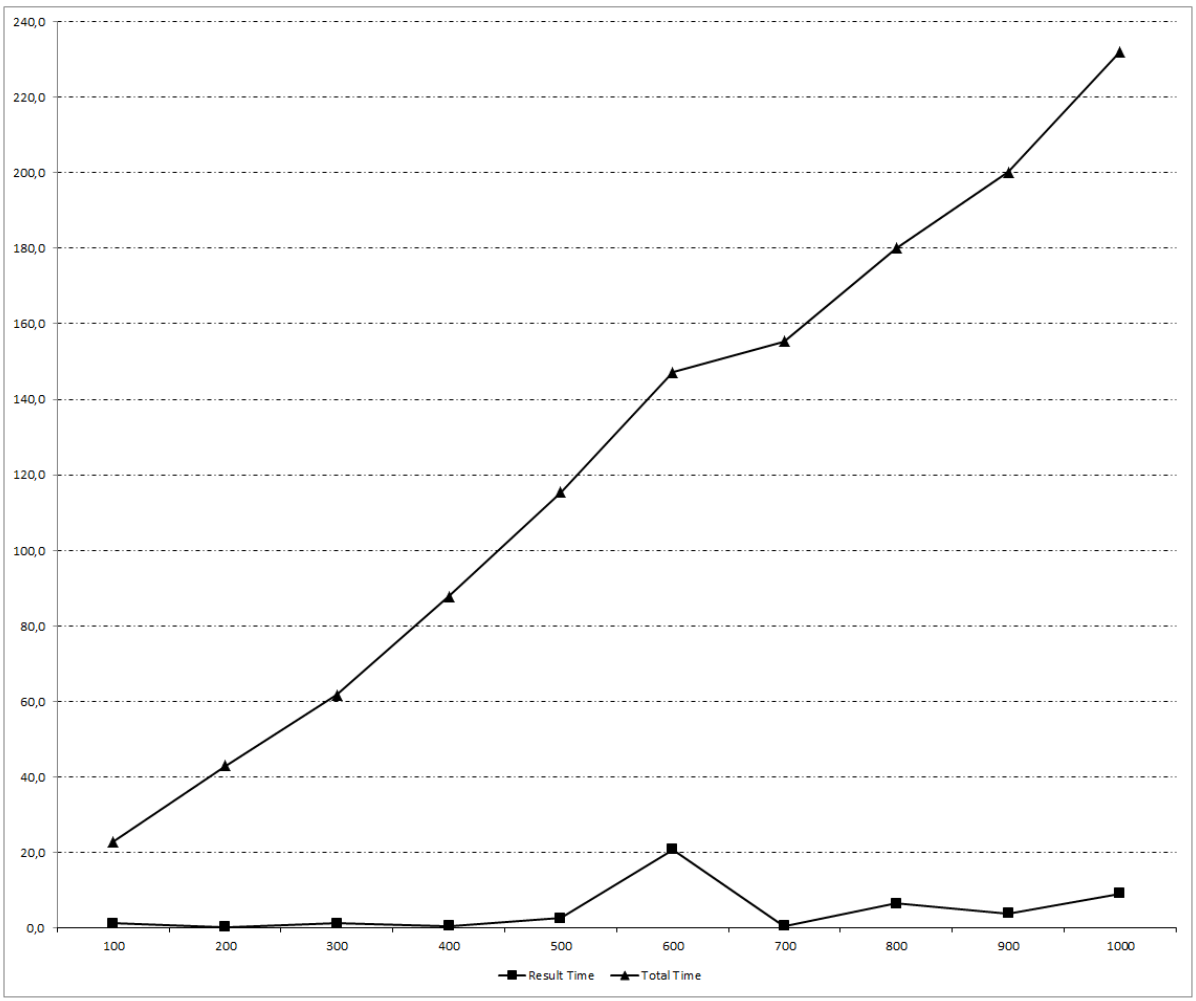

Figure 4.28. Time results of SelParColNum Algorithm in Interval Graphs (100-1000 Number of Clusters).

4.3.5. Selective Partial Coloring Algorithm-Cluster Degree Based

This algorithm is a different version of SelParColNum algorithm mainly by its objective function. In SelParColNum algorithm, we used number of clusters in the pool as an objective function. In this modified version namely Selective Partial Coloring Algorithm-Cluster Degree Based(SelParColDeg), objective function is total cluster degrees in the pool. Cluster degree is the minimum out-degree of vertices in a cluster. Each vertex has a cluster degree depending on their clusters. We thought that it can be a different approach and can lead better results at least in some graph classes.

In this algorithm, starting color classes creation is a little different than SelParColNum too. In SelParColNum algorithm, we take last feasible solution which is colored by k colors and create a new partial solution that has $k-1$ color classes by removing the color class which has least number of vertices(ties are broken randomly). But in SelParColDeg algorithm, we remove color class with minimum total cluster outdegree. In other words, algorithm calculates total cluster outdegrees of each color class and removes the one which has minimum total cluster outdegree and adds their clusters to the pool. Remaining parts of the algorithm exactly works in the same manner as with SelParColNum algorithm.

SelParColDeg algorithm for the selective graph coloring problem operates as we mentioned above. The algorithm is summarized in Algorithm 4.29.

Input: Clustered Graph Instance, Construction Solution, b, B, t, *thresholdValue*;
Output: Tabu Solution;
Build a new starting solution by deleting color class with minimum total cluster degree in last solution and add clusters of deleted vertices to $\mathcal{O}$;
Free the tabu list;
Set *iteration* := 0 *currFS* := *totalClusterDegree*($\mathcal{O}$) *bestFS* := *totalClusterDegree*($\mathcal{O}$) $k := k - 1$;
repeat
Select a cluster V from $\mathcal{O}$ arbitrarily and update $\mathcal{O} := \mathcal{O} \setminus V$;
Find best vertex-color class pair and assign best vertex to best color class;
If best vertex has any conflicts with other vertices in best color class, remove them from the color class and add their clusters to $\mathcal{O}$;
Update *currFS*;
if *currFS* < *bestFS* **then**
bestFS := *currFS*
end if
Update tabu tenure with reactive tabu tenure algorithm
until *iteration* < *maxIteration* and *bestFS* $\neq 0$

Figure 4.29. SelParColDeg Algorithm.

4.3.6. Computational Results of SelParColDeg Algorithm

First of all, we should find a suitable parameter set for reactive tabu scheme. There are 4 different parameters(b, B, C, $thresholdvalue$) that we can fix to find a good t. We fixed $b = 1000$ and $C = 1$ and searched good parameter values for B and $thresholdvalue$.

We used same "DSJC500.5-1", "DSJC500.5-2", "DSJC500.5-3" and "DSJC500.5-4" instances to set our parameter values.

We tried a wide range of parameter values but most of them do not give good results. We showed significant results of parameter tuning experiment for SelParColDeg algorithm in Tables 4.27, 4.28, 4.29 and 4.30. SelParColDeg algorithm is runned 10 times for all different parameter sets for each graph. Maximum iteration number of these instances is 3.000.000.

Table 4.27. SelParColDeg Parameter Tuning (DSJC500.5-1).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-1	65	52	52.9	53	2000	1
DSJC500.5-1	65	52	52.7	53	2000	3
DSJC500.5-1	65	53	53.1	54	3000	1
DSJC500.5-1	65	**53**	**53.3**	**54**	3000	3
DSJC500.5-1	65	53	54.4	55	4000	1
DSJC500.5-1	65	55	55.4	56	4000	3
DSJC500.5-1	65	55	55.3	56	5000	1
DSJC500.5-1	65	55	55.7	56	5000	3

Table 4.28. SelParColDeg Parameter Tuning (DSJC500.5-2).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-2	59	47	47.6	48	2000	1
DSJC500.5-2	59	47	47.8	49	2000	3
DSJC500.5-2	59	47	47.5	48	3000	1
DSJC500.5-2	59	**46**	**46.9**	**47**	3000	3
DSJC500.5-2	59	47	47.4	48	4000	1
DSJC500.5-2	59	46	47.1	48	4000	3
DSJC500.5-2	59	46	47.3	48	5000	1
DSJC500.5-2	59	46	47.3	48	5000	3

Table 4.29. SelParColDeg Parameter Tuning (DSJC500.5-3).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-3	57	46	46.9	48	2000	1
DSJC500.5-3	57	45	45.7	47	2000	3
DSJC500.5-3	57	45	45.3	46	3000	1
DSJC500.5-3	57	**45**	**45.1**	**46**	3000	3
DSJC500.5-3	57	45	45.2	46	4000	1
DSJC500.5-3	57	44	44.9	45	4000	3
DSJC500.5-3	57	45	45.1	46	5000	1
DSJC500.5-3	57	44	44.8	45	5000	3

Table 4.30. SelParColDeg Parameter Tuning (DSJC500.5-4).

	Cons. Result	**Min. Tabu**	**Avg. Tabu**	**Max. Tabu**	**Thres. Value**	**B Value**
DSJC500.5-4	55	45	46.5	48	2000	1
DSJC500.5-4	55	44	44.8	47	2000	3
DSJC500.5-4	55	44	44.3	45	3000	1
DSJC500.5-4	55	**43**	**43.4**	**44**	3000	3
DSJC500.5-4	55	43	43.9	44	4000	1
DSJC500.5-4	55	43	43.5	44	4000	3
DSJC500.5-4	55	42	43.6	44	5000	1
DSJC500.5-4	55	43	43.6	44	5000	3

After these experiments, we decided to use $B = 3$ and $thresholdvalue = 3000$ for our further experiments because in average that parameter set gives better results. By the reason of the fluctuation in the objective function is too much, we should change *thresholdvalue* depending on the number of clusters. Therefore, we used $thresholdvalue = 6 * number\ of\ clusters$ to adjust this value with number of clusters in consequence of our empirical studies. In this way, *thresholdvalue* for 500 clusters will be equal to 3000 as we found in parameter tuning phase.

We used same instance sets that are used in Section 4.3.2. Each instance type has 10 different instances in it and SelParColDeg runs 5 times for each instance. Eventually, average tabu results stand for 50 runs.

Average construction algorithm results, average tabu results and improvement percentage between them are given in Tables 4.31 and 4.33. Their plots are given in Figures 4.30 and 4.32.

Table 4.31. Construction vs SelParColDeg Algorithm (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
500Cl1000Ver01Den	14.0	12.0	14.3%
500Cl1000Ver02Den	24.5	20.3	17.3%
500Cl1000Ver03Den	35.2	28.4	19.3%
500Cl1000Ver04Den	46.9	37.5	20.1%
500Cl1000Ver05Den	59.8	47.3	20.8%
500Cl1000Ver06Den	74.9	58.6	21.8%
500Cl1000Ver07Den	92.7	72.4	21.9%
500Cl1000Ver08Den	116.8	90.4	22.6%
500Cl1000Ver09Den	154.8	121.5	21.5%

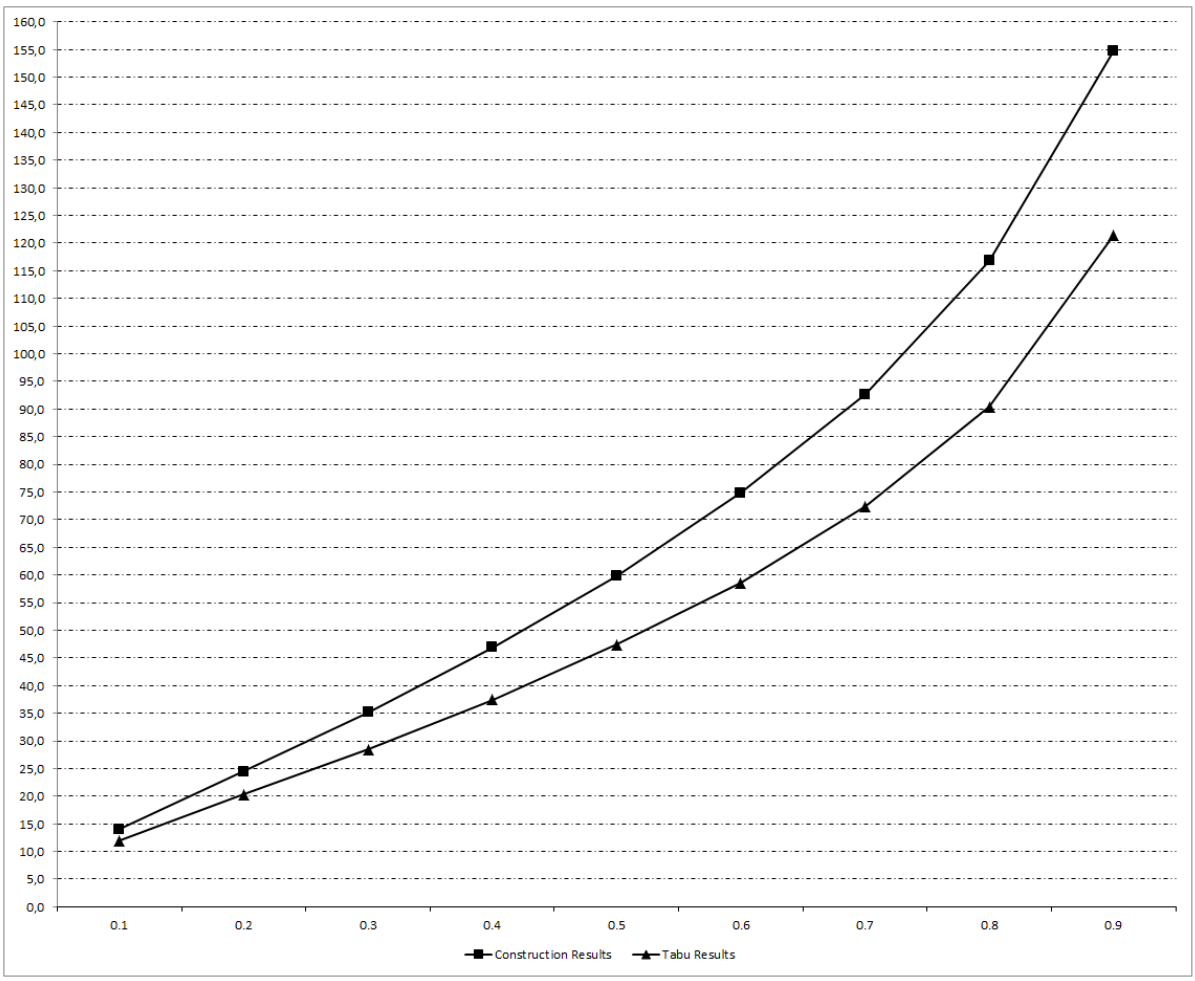

Figure 4.30. Construction vs SelParColDeg Algorithm (0.1-0.9 Edge Density).

Table 4.32. Time results of SelParColDeg Algorithm (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
500Cl1000Ver01Den	0.2	69.8
500Cl1000Ver02Den	16.6	77.8
500Cl1000Ver03Den	17.8	80.6
500Cl1000Ver04Den	23.1	88.3
500Cl1000Ver05Den	35.4	104.2
500Cl1000Ver06Den	55.1	132.3
500Cl1000Ver07Den	68.1	145.4
500Cl1000Ver08Den	75.3	144.7
500Cl1000Ver09Den	71.5	141.6

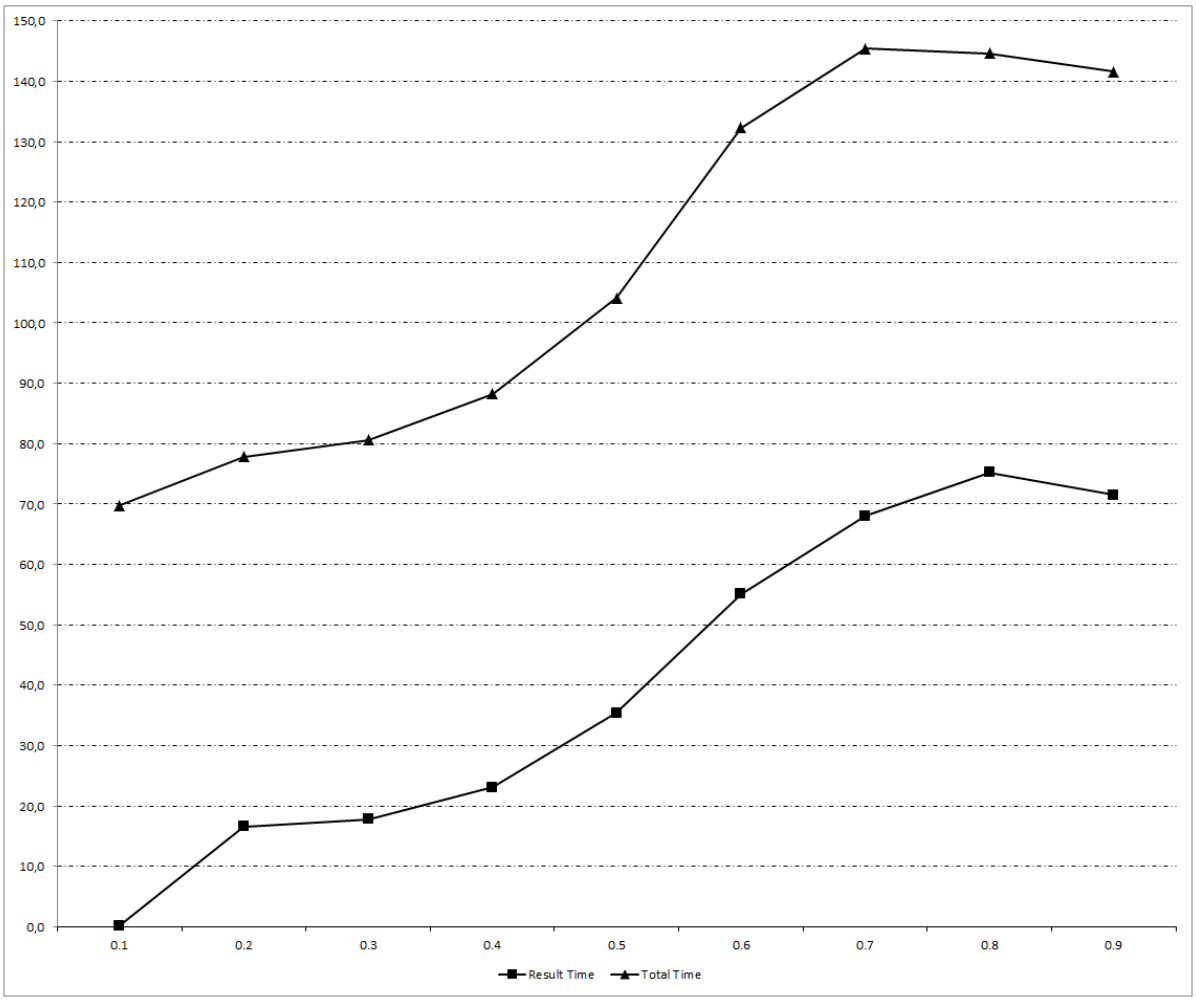

Figure 4.31. Time results of SelParColDeg Algorithm (0.1-0.9 Edge Density).

Table 4.33. Construction vs SelParColDeg Algorithm (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Improvement Percentage**
100Cl200Ver05Den	16.4	12.0	26.8%
200Cl400Ver05Den	28.2	21.6	25.5%
300Cl600Ver05Den	39.4	30.4	24.4%
400Cl800Ver05Den	50.2	39.1	24.4%
500Cl1000Ver05Den	60.2	47.4	23.6%
600Cl1200Ver05Den	70.2	55,6	23.3%
700Cl1400Ver05Den	80.2	63.7	23.0%
800Cl1600Ver05Den	89.0	71.4	22.0%
900Cl1800Ver05Den	98.4	79.5	21.9%
1000Cl2000Ver05Den	107.8	87.1	21.7%

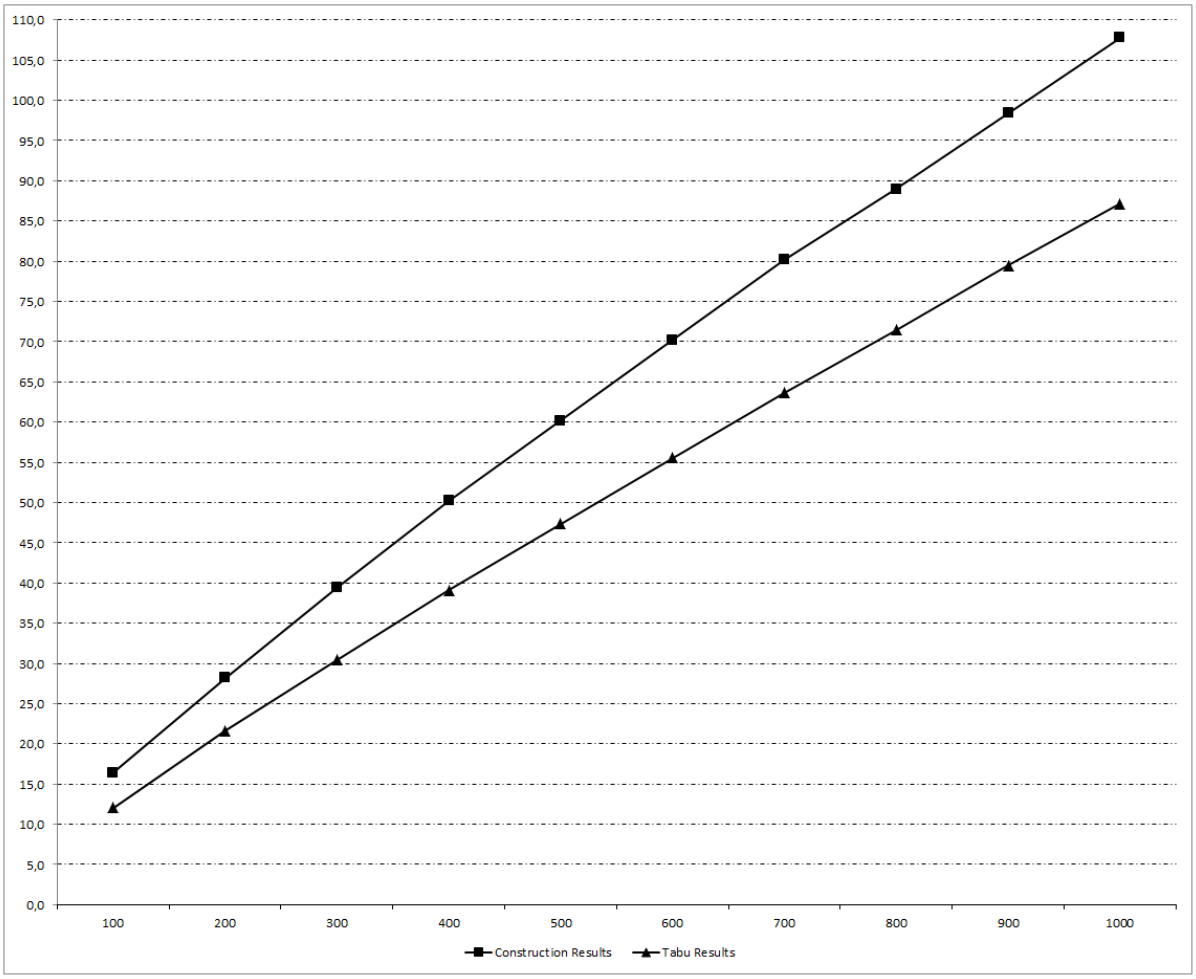

Figure 4.32. Construction vs SelParColDeg Algorithm (100-1000 Number of Clusters).

Table 4.34. Time results of SelParColDeg Algorithm (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
100Cl200Ver05Den	1.8	17.0
200Cl400Ver05Den	5.7	33.8
300Cl600Ver05Den	14.0	55.0
400Cl800Ver05Den	21.2	77.2
500Cl1000Ver05Den	37.5	107.1
600Cl1200Ver05Den	47.8	135.6
700Cl1400Ver05Den	57.1	158.0
800Cl1600Ver05Den	79.3	189.5
900Cl1800Ver05Den	91.3	222.1
1000Cl2000Ver05Den	118.5	264.1

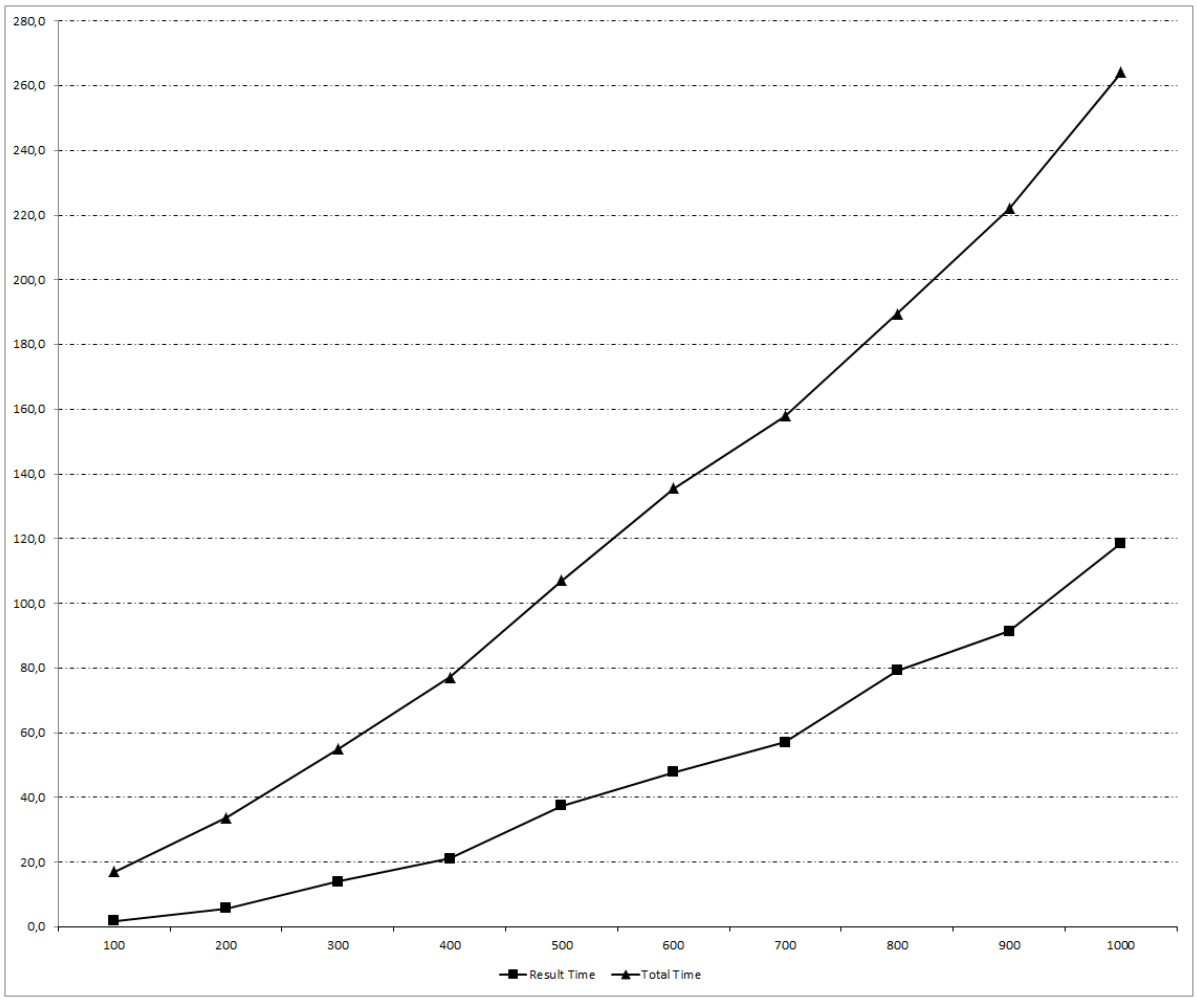

Figure 4.33. Time results of SelParColDeg Algorithm (100-1000 Number of Clusters).

We used same interval instance sets in Section 4.3.2. Each instance type has 3 different instances in it and SelParColDeg runs 5 times for each instance. Eventually, average tabu results stand for 15 runs.

In interval graphs, parameters that we set for SelParColDeg did not work well. We tried to find better parameter set to solve this problem but we could not. Due to objective function is depending on degrees, it fluctuates too much. For this reason, we decided to use dynamic tabu tenure for these instances. In every iteration t randomly takes a value between 0 and 50. This upper bound is a result of our empirical studies to fix this bound.

Average construction algorithm results, average tabu results, exact results, improvement percentage of tabu algorithm and gap between exact result and tabu result are given in Tables 4.35 and 4.37. Their plots are given in Figures 4.34 and 4.36.

Table 4.35. Construction vs SelParColDeg vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improvement Percentage**	**Gap with Exact Result**
Int500Cl1000Ver01Den	29.0	21.8	20.7	24.8%	5.2%
Int500Cl1000Ver02Den	52.7	42.5	39.3	19.2%	7.5%
Int500Cl1000Ver03Den	73.3	63.3	56.3	13.6%	11.1%
Int500Cl1000Ver04Den	93.0	85.4	73.7	8.2%	13.7%
Int500Cl1000Ver05Den	104.7	98.1	91.3	6.2%	6.9%
Int500Cl1000Ver06Den	122.7	118.5	116.0	3.4%	2.1%
Int500Cl1000Ver07Den	157.7	155.4	151.3	1.4%	2.6%
Int500Cl1000Ver08Den	202.3	201.0	199.0	0.7%	1.0%
Int500Cl1000Ver09Den	272.0	271.6	270.3	0.1%	0.5%

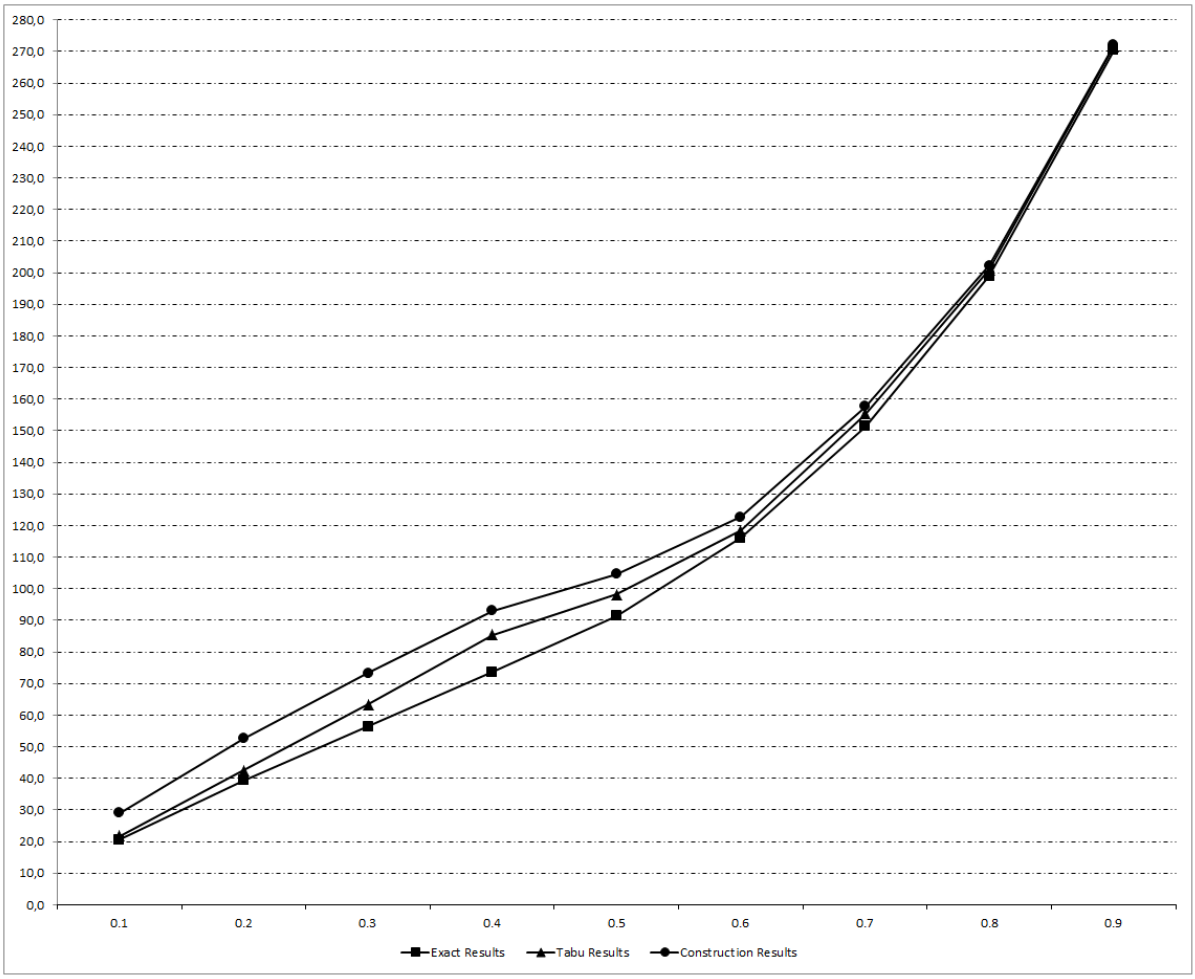

Figure 4.34. Construction vs SelParColDeg vs Exact Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.36. Time results of SelParColDeg Algorithm in Interval Graphs (0.1-0.9 Edge Density).

	Result Time (seconds)	**Total Time (seconds)**
Int500Cl1000Ver01Den	7.1	69.2
Int500Cl1000Ver02Den	12.2	81.8
Int500Cl1000Ver03Den	10.4	78.2
Int500Cl1000Ver04Den	13.3	85.6
Int500Cl1000Ver05Den	12.8	84.3
Int500Cl1000Ver06Den	5.6	76.6
Int500Cl1000Ver07Den	10.3	93.1
Int500Cl1000Ver08Den	0.0	84.3
Int500Cl1000Ver09Den	0.4	92.4

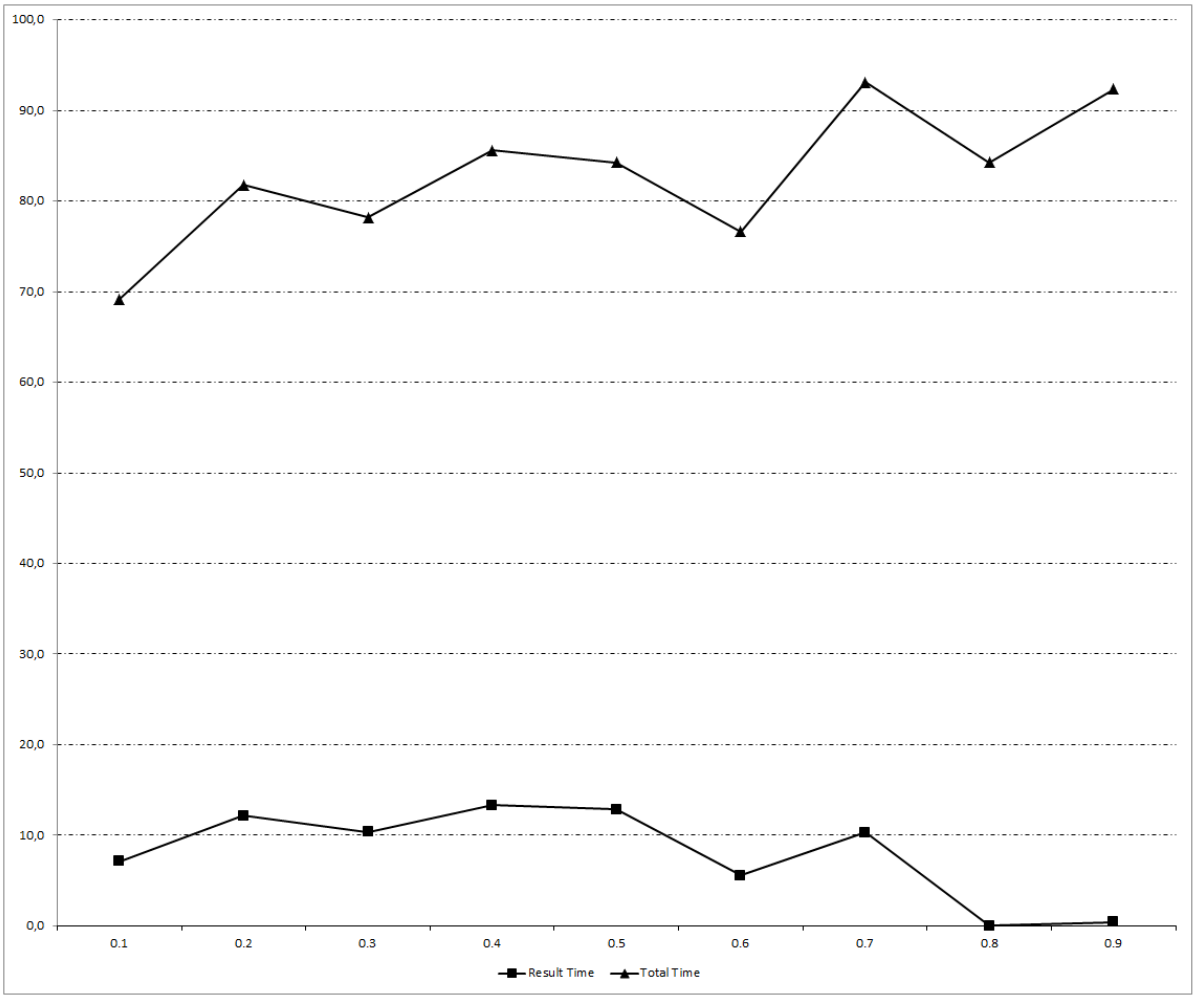

Figure 4.35. Time results of SelParColDeg Algorithm in Interval Graphs (0.1-0.9 Edge Density).

Table 4.37. Construction vs SelParColDeg vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Avg. Cons.	**Avg. Tabu**	**Exact Results**	**Improve-ment Percentage**	**Gap with Exact Result**
Int100Cl200Ver05Den	23.7	20.7	20.7	12.4%	0.3%
Int200Cl400Ver05Den	45.7	41.1	39.7	9.9%	3.6%
Int300Cl600Ver05Den	66.0	60.1	57.3	9.0%	4.6%
Int400Cl800Ver05Den	87.7	79.3	76.3	9.5%	3.8%
Int500Cl1000Ver05Den	106.7	99.8	93.0	6.4%	6.8%
Int600Cl1200Ver05Den	127.7	120.7	108.0	5.5%	10.5%
Int700Cl1400Ver05Den	140.0	131.6	126.7	6.0%	3.7%
Int800Cl1600Ver05Den	167.3	158.3	146.7	5.4%	7.3%
Int900Cl1800Ver05Den	187.0	174.1	161.0	6.9%	7.5%
Int1000Cl2000Ver05Den	213.0	199.3	183.3	6.4%	8.0%

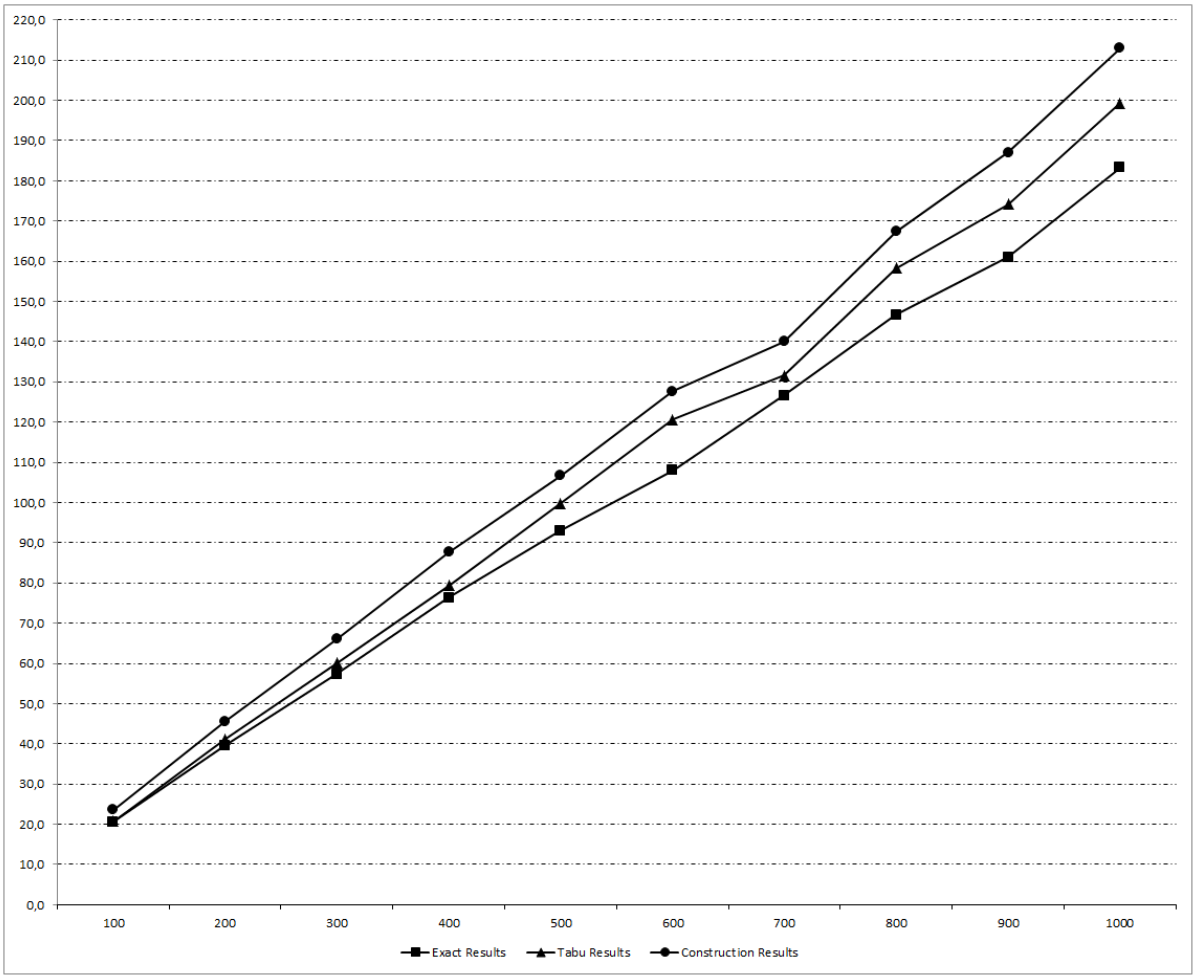

Figure 4.36. Construction vs SelParColDeg vs Exact Algorithm in Interval Graphs (100-1000 Number of Clusters).

Table 4.38. Time results of SelParColDeg Algorithm in Interval Graphs (100-1000 Number of Clusters).

	Result Time (seconds)	**Total Time (seconds)**
Int100Cl200Ver05Den	0.2	16.7
Int200Cl400Ver05Den	3.5	32.7
Int300Cl600Ver05Den	12.7	57.6
Int400Cl800Ver05Den	3.7	61.5
Int500Cl1000Ver05Den	23.3	94.5
Int600Cl1200Ver05Den	21.7	110.8
Int700Cl1400Ver05Den	17.3	116.8
Int800Cl1600Ver05Den	5.2	128.5
Int900Cl1800Ver05Den	38.3	173.3
Int1000Cl2000Ver05Den	56.5	207.2

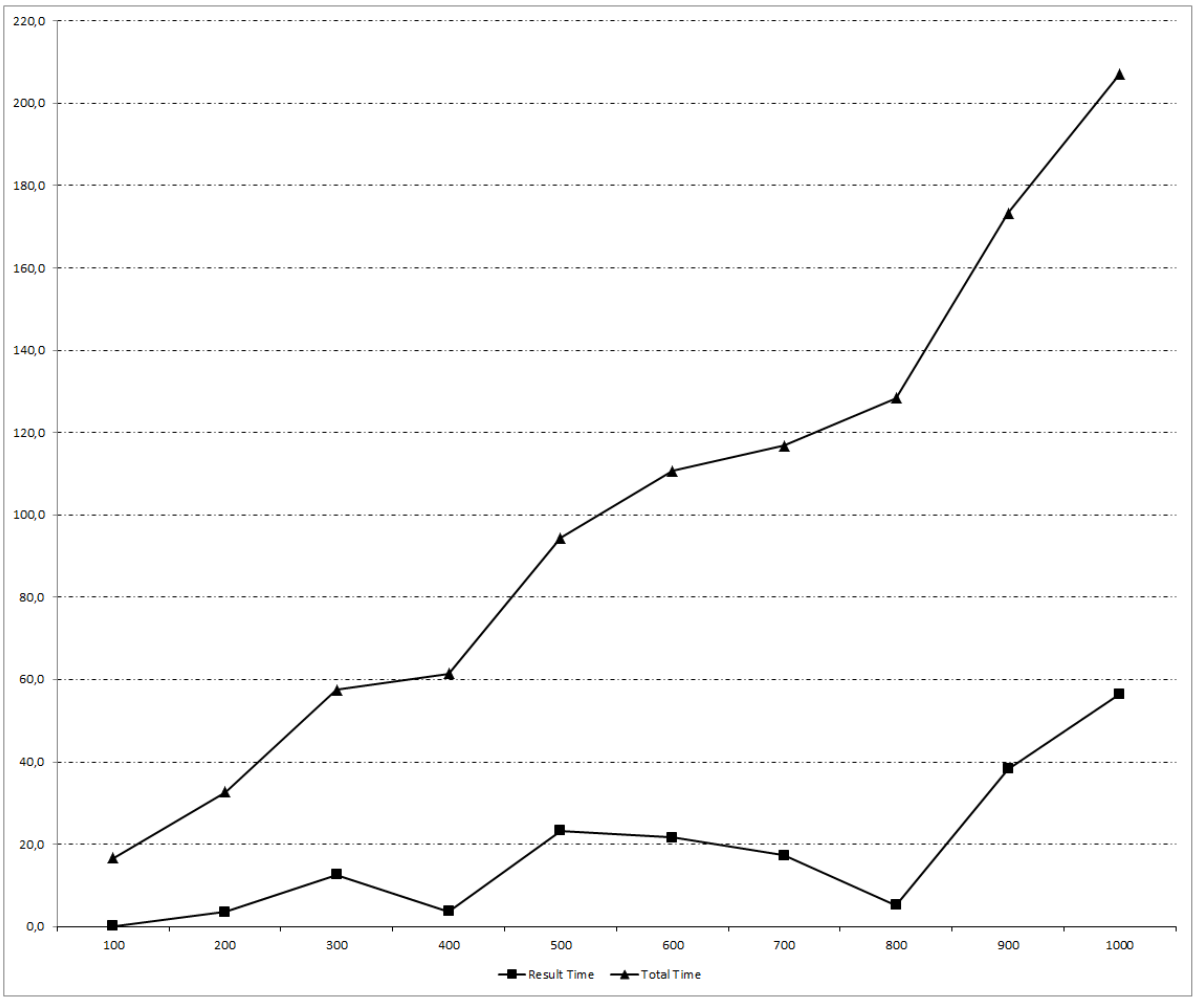

Figure 4.37. Time results of SelParColDeg Algorithm in Interval Graphs (100-1000 Number of Clusters).

4.3.7. Comparison of Tabu Algorithms for The Selective Graph Coloring Problem

We compared our new tabu search algorithms for the selective graph coloring problem with TS-PCP [4]. We used the same instance generation principles with their paper. We used best results of TS-PCP for these instances in the paper and our best results with the selected parameter sets. Algorithms are repeated 10 times likewise in [4]. Minimum values of algorithms are compared in Table 4.39, average values of algorithms are compared in Table 4.40 and maximum values of algorithms are compared in Table 4.41.

Table 4.39. Comparison of TS-PCP vs New Tabu Algorithms (Min. values).

	TS-PCP	**SelImpCol**	**SelParColNum**	**SelParColDeg**
DSJC500.5-1	51	50	50	53
DSJC500.5-2	45	45	45	46
DSJC500.5-3	43	42	43	45
DSJC500.5-4	42	41	41	43

Table 4.40. Comparison of TS-PCP vs New Tabu Algorithms (Avg. values).

	TS-PCP	**SelImpCol**	**SelParColNum**	**SelParColDeg**
DSJC500.5-1	51.3	51.2	51.2	53.3
DSJC500.5-2	45.9	45.3	45.5	46.9
DSJC500.5-3	43.2	43.0	43.1	45.1
DSJC500.5-4	42.0	41.5	41.7	43.4

According to computational results, SelImpCol and SelParColNum has better results than TS-PCP. Results of SelParColDeg algorithm is far from other results. Best algorithm among these algorithms is SelImpCol. It gives best average values in all instances and better minimum values for most of the instances. In other words, SelImpCol algorithm can find solutions that the other algorithms cannot reach.

Table 4.41. Comparison of TS-PCP vs New Tabu Algorithms (Max. values).

	TS-PCP	**SelImpCol**	**SelParColNum**	**SelParColDeg**
DSJC500.5-1	52	52	52	54
DSJC500.5-2	46	46	46	47
DSJC500.5-3	44	44	44	46
DSJC500.5-4	42	42	42	44

In the sequel, we increased the total number of iterations 1.000.000 to 2.000.000 for SelImpCol algorithm to see the difference between SelImpCol and TS-PCP algorithm better.

Table 4.42. Comparison of TS-PCP vs SelImpCol (Min. values).

	TS-PCP	**SelImpCol**
DSJC500.5-1	51	50
DSJC500.5-2	45	45
DSJC500.5-3	43	42
DSJC500.5-4	42	41

The results given in Tables 4.42, 4.43 and 4.44, show that SelImpCol finds better results than best results of TS-PCP.

Table 4.43. Comparison of TS-PCP vs SelImpCol (Avg. values).

	TS-PCP	**SelImpCol**
DSJC500.5-1	51.3	50.6
DSJC500.5-2	45.9	45.0
DSJC500.5-3	43.2	42.3
DSJC500.5-4	42.0	41.0

Table 4.44. Comparison of TS-PCP vs SelImpCol (Max. values).

	TS-PCP	**SelImpCol**
DSJC500.5-1	52	51
DSJC500.5-2	46	45
DSJC500.5-3	44	43
DSJC500.5-4	42	41

5. CONCLUSION

In this study, exact and heuristic algorithms are considered for the selective graph coloring problem. In the first part of the study, we focused on special graph classes that an exact result can be found by using their properties. For this reason, we studied interval and disk graphs. We developed a very fast exact solution method for selective coloring in interval graphs which model several scheduling problems and an exact selective clique algorithm for small instances in disk graphs. Exact selective coloring algorithm for interval graphs can handle big graphs within a short time. However, exact maximum selective clique algorithm for unit disk graphs can only handle small instances in an acceptable time.

In the second part of this study, we focused on developing new heuristics to the selective graph coloring problem. We compared onestepCD algorithm with our 2 different easily generated construction heuristics. onestepCD algorithm gave better results than others. Then, we proposed 3 tabu search algorithms to improve onestepCD algorithm. 2 of these algorithms gave better results than the algorithm of Noronha and Ribeiro [4]. Moreover, we tested our tabu algoritms on interval graphs and compare with the exact results of same instances with the help of our exact selective coloring algorithm for interval graphs.

This study arises several interesting future research directions:

- Is there any other graph classes for which exact algorithms run efficiently?
- Can we improve our exact maximum selective clique algorithm to solve bigger instances?
- Can we find better results than our tabu search heuristic with another heuristic approach?

REFERENCES

1. Golumbic, M. C. and I. B.-A. Hartman, *Graph Theory, Combinatorics and Algorithms*, Springer, New York, NY, USA, 2004.

2. Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, CA, 1979.

3. Li, G. and R. Simha, "The Partition Coloring Problem and its Application to Wavelength Routing and Assignment", *In Proceedings of the First Workshop on Optical Networks*, 2000.

4. Noronha, T. F. and C. C. Ribeiro, "Routing and Wavelength Assignment by Partition Colouring", *European Journal of Operational Research*, Vol. 171, No. 3, pp. 797–810, 2006.

5. Frota, Y., N. Maculan, T. F. Noronha and C. C. Ribeiro, "A Branch-and-Cut Algorithm for Partition Coloring.", *Networks*, Vol. 55, No. 3, pp. 194–204, 2010.

6. Hoshino, E. A., Y. Frota and C. C. de Souza, "A Branch-and-Price Approach for The Partition Coloring Problem.", *Operations Research Letters*, Vol. 39, No. 2, pp. 132–137, 2011.

7. Demange, M., J. Monnot, P. Pop and B. Ries, "Selective Graph Coloring in Some Special Classes of Graphs", A. R. Mahjoub, V. Markakis, I. Milis and V. T. Paschos (Editors), *ISCO*, Vol. 7422 of *Lecture Notes in Computer Science*, pp. 320–331, Springer, 2012.

8. Lovász, L., "A Characterization of Perfect Graphs", *Journal of Combinatorial Theory, Series B*, Vol. 13, No. 2, pp. 95 – 98, 1972.

9. Lovász, L., "Normal Hypergraphs and The Perfect Graph Conjecture", *Discrete*

Mathematics, Vol. 306, No. 10-11, pp. 867–875, 2006.

10. Chudnovsky, M., N. Robertson, P. Seymour and R. Thomas, "The Strong Perfect Graph Theorem", *Annals of Mathematics*, Vol. 164, pp. 51–229, 2006.

11. Porumbel, D., J. K. Hao and P. Kuntz, "DIMACS Graphs: Benchmark Instances and Best Upper Bounds", Online, Accessed at November 2012, `http://www.info.univ-angers.fr/pub/porumbel/graphs/`.

12. Mehrotra, A. and M. A. Trick, "A Column Generation Approach For Graph Coloring", *INFORMS Journal on Computing*, Vol. 8, pp. 344–354, 1995.

13. Clark, B. N., C. J. Colbourn and D. S. Johnson, "Unit Disk Graphs", *Discrete Mathematics*, Vol. 86, No. 1-3, pp. 165–177, 1990.

14. Brélaz, D., "New Methods to Color the Vertices of a Graph", *Communications of the ACM*, Vol. 22, pp. 251–256, 1979.

15. Paschos, V. T., *Paradigms of Combinatorial Optimization*, John Wiley and Sons, 2010.

16. Gendreau, M. and J.-Y. Potvin, "Metaheuristics in Combinatorial Optimization", *Annals OR*, Vol. 140, No. 1, pp. 189–213, 2005.

17. Blöchliger, I. and N. Zufferey, "A Graph Coloring Heuristic Using Partial Solutions and a Reactive Tabu Scheme", *Computers and OR*, Vol. 35, No. 3, pp. 960–975, 2008.

18. Gendreau, M. and J. Potvin, "Tabu Search", M. Gendreau and J. Potvin (Editors), *Handbook of Metaheuristics*, pp. 41–59, Springer, 2010.

19. Hertz, A., E. Taillard and D. D. Werra, *A Tutorial on Tabu Search*, Tech. rep., 1995.