FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Selective Graph Coloring Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Computational Intelligence**

eingereicht von

**Claus-Dieter Volko**
Matrikelnummer 0102122

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.-Prof. Dipl.-Ing. Dr. Günther Raidl
Mitwirkung: Univ.-Ass. Dipl.-Ing. Dr. Bin Hu

Wien, 31.03.2013

(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

# Erklärung zur Verfassung der Arbeit

Claus-Dieter Volko
Hungereckstr. 60/2, 1230 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Unterschrift Verfasser)

# Danksagung

Ich möchte mich vor allem bei Günther Raidl für die Möglichkeit bedanken, meine Diplomarbeit an seiner Abteilung zu verfassen, und bei Bin Hu für die ausgezeichnete Betreuung.

Weiters möchte ich mich bei meinen Eltern bedanken, die mir das Studium finanziell ermöglicht haben. Zudem habe ich ihnen auch zu verdanken, dass sie mein Interesse an Computern frühzeitig erkannt und gefördert haben.

# Abstract

The Selective Graph Coloring Problem (SGCP) is about finding a subgraph of a particular structure whose chromatic number is as low as possible. The original graph is divided into several clusters, and from each cluster the subgraph has to contain exactly one node. This problem is NP-hard and therefore it is usually solved by means of heuristics.

I implemented several variants of an algorithm making use of Variable Neighborhood Search (VNS) to search the space of solution candidates and then evaluating the solution using heuristic or exact methods. Furthermore, each variant can be used with or without a solution archive, i.e. a data structure in which previously found solutions are stored so that duplicates need not be re-evaluated but can be efficiently converted into new solutions instead. For exact computation of the chromatic number integer linear programming was used. To obtain an upper bound a variant of greedy coloring was used. Another variant of the algorithm also counts the number of conflicts that would appear if one color less were used. Finally, two methods were implemented to obtain a lower bound: maximum clique and linear programming using column generation.

The program was tested with various instances from the literature. My algorithm often finished computation within a very short time, but in general it led to slightly worse results.

# Kurzfassung

Beim Selective Graph Coloring Problem (SGCP) geht es darum, einen Teilgraphen mit spezieller Struktur zu finden, dessen chromatische Zahl so niedrig wie möglich ist. Der Ursprungsgraph ist in mehrere Cluster unterteilt, und von jedem Cluster muss der Teilgraph genau einen Knoten enthalten. Dieses Problem ist NP-schwer und wird daher meistens mit Heuristiken gelöst.

Ich habe mehrere Varianten eines Algorithmus implementiert, der Variable Neighborhood Search (VNS) benutzt, um den Lösungsraum zu durchsuchen, und dann die gefundene Lösung mit heuristischen oder exakten Methoden evaluiert. Jede Variante kann mit oder ohne ein Lösungsarchiv verwendet werden. Ein Lösungsarchiv ist eine Datenstruktur, in der bereits gefundene Lösungen gespeichert werden, so dass Duplikate nicht neu evaluiert werden müssen, sondern effizient zu neuen Lösungen konvertiert werden können. Um eine obere Schranke zu errechnen, wurde eine Variante von Greedy Coloring verwendet. Eine weitere Variante des Algorithmus zählt auch die Anzahl der Konflikte, die entstünden, würde eine Farbe weniger verwendet werden. Schließlich wurden zwei Methoden umgesetzt, um eine untere Schranke zu berechnen: maximale Clique und lineare Programmierung mit Spaltengenerierung.

Das Programm wurde mit verschiedenen Instanzen aus der Literatur getestet. Mein Algorithmus beendete die Berechnungen oft schon nach sehr kurzer Laufzeit, führte aber im Allgemeinen zu geringfügig schlechteren Ergebnissen.

# Contents

# Chapter 1

# Introduction

Graphs are a useful tool for modelling real-world problems, as they can serve as an abstraction for various things, such as networks and maps. For this reason, the solution of problems related to graph theory may have an impact in real life. Computer science students usually learn about some of these problems as well as algorithms for solving them in advanced courses on algorithmics. Topics commonly discussed in these courses include shortest path problems, finding the maximal flow in a network, and the Traveling Salesperson Problem. Depth First Search, Breadth First Search, Dijkstra's algorithm, the Bellman-Ford algorithm, the Floyd-Warshall algorithm, Johnson's algorithm, the Ford-Fulkerson method, preflow-push algorithms and other methods belong to the general education of any computer scientist specializing in algorithms.

Graph coloring usually does not appear in these courses, but it is still an important problem about which many papers have been published. Applications of graph coloring include time tabling and various forms of allocation tasks [10]. Since it is an NP-equivalent problem, various heuristics have been proposed to get good results in a reasonable amount of time. The selective graph coloring problem is an extension of graph coloring and for this reason, it is NP-hard as well. It is about finding a subgraph consisting of one node of each cluster that has a chromatic number as low as possible. For most researchers the motivation to study this problem has been its relevance to optical networks [39]. Although some papers have been published that propose efficient solution algorithms for this problem, by far not all possible solution algorithms have been explored yet. This was my motivation for choosing this problem as the topic of my diploma thesis.

In this thesis, I will present diverse variants of a heuristic solution algorithm for the selective graph coloring problem and the results obtained for some test instances. The algorithm is based on variable neighborhood search for scanning the solution space. Each solution is evaluated using exact or heuristic methods.

The exact method is exact but slow. Regarding heuristics mainly an upper bound is computed, but there are also variants calculating a number of conflicts that would arise with one color less and giving a lower bound. In addition a solution archive has been implemented, which makes it possible to avoid duplicates during the search space exploration and easily find new solutions not evaluated as local optima yet.

Figure 1.1 shows a graph with several clusters and a possible solution subgraph, thus demonstrating what the SGCP is all about.

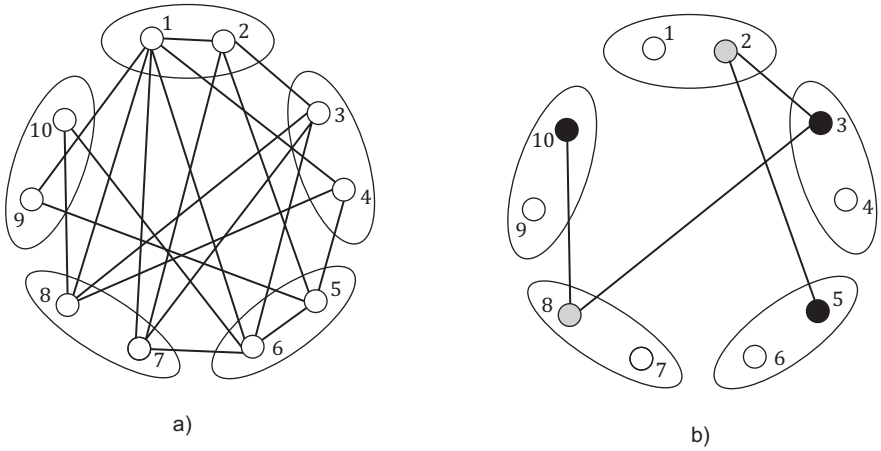


**Figure 1.1:** a) A graph with several clusters, b) A possible solution subgraph

## 1.1 Graph Theory Basics

The selective graph coloring problem (SGCP) is a graph theory problem. Graph theory deals with an abstract formalism that can be used to represent various real-world structures, such as networks (e.g. telephone networks) or traffic structures. A graph is a structure that consists of two sets, a set of nodes and a set of edges. A node is an abstract entity which may represent anything, e.g. a telephone. It is worth mentioning that the spatial structure of the graph is irrelevant. The nodes may be located anywhere in space. It is not important where they are located or how distant they are to each other. Edges are connections between nodes. Each edge connects two distinct nodes. In the real world, an edge may correspond to a line connecting two telephones, for example.

Graphs may be directed or undirected. If a graph is directed, this means that each edge has a node for which it is an outgoing edge, and a node for which it is an incoming edge. In this case, edges are also called arcs. However, the SGCP deals only with undirected graphs. In undirected graphs the two nodes of each

edge are of equal value.

Edges may have weights, that is values assigned to them. This plays a role in many graph theoretical problems, but not in the SGCP. As a consequence we do not consider edge weights in the SGCP.

In the SGCP nodes may be colored. That is, each node is assigned one color. What is important is that in a proper coloring, there must not be a pair of two nodes sharing an edge that have the same color. This is the main obstacle for finding a solution to the SGCP.

In the graph coloring problem, an undirected graph is given, and the objective is to determine the minimal number of colors that is needed to gain a proper coloring of the graph. This number is also called the chromatic number of the graph. The difference to the SGCP is that all nodes must be considered. In the SGCP, by contrast, each node is assigned to a cluster. The objective is to find a subgraph of the given graph which consists of one node per cluster, i.e. from each cluster exactly one node per cluster is taken. This subgraph must have a low chromatic number, in the optimal case the minimal chromatic number possible.

## 1.2 Applications

According to the literature [39], an application of the selective graph coloring problem is the routing and wavelength assignment (RWA) problem in optical networks. In such a network two edges (also called lightpaths) may use the same wavelength if they do not share a common link. Therefore the problem is dual to graph coloring, as edges instead of vertices are assigned colors. The optimal assignment of wavelengths in such a network can thus be obtained by solving a related instance of the selective graph coloring problem.

## 1.3 Formal Definition and Complexity

Given is an unweighted, undirected graph $G =< V, E >$ with a set of nodes $V$ and a set of edges $E$. Each node is assigned to one subgraph, a so-called cluster $C_i$. In total there are $n$ disjoint clusters. The goal is to find a subgraph $S = G(W)$ with $W =< v_1, ..., v_n >$ where $v_i \in C_i, 1 \leq i \leq n$. The subgraph should have a minimal chromatic number. The chromatic number is the minimal number of different colors using which the nodes of a graph can be colored so that there is no single pair of nodes $u$ and $v$ connected by an edge $(u, v)$ that have the same color.

Thus, the SGCP is an extension of the graph coloring problem, which computer scientists and mathematicians have studied for decades. The Graph Coloring Problem is about computing the chromatic number of a given graph, and already

in 1972 it was discovered to be an NP-equivalent problem [3]. Since the Graph Coloring Problem is a subproblem of the selective graph coloring problem, the selective graph coloring problem is NP-hard as well. For this reason it makes sense to use a heuristic approach both to estimate the chromatic number of a possible solution and to find a better solution.

# Chapter 2

# Literature Survey

This chapter provides an overview of the literature about the selective graph coloring problem as well as graph coloring.

## 2.1 Selective Graph Coloring Problem and Routing and Wavelength Assignment Problem

In the literature, the selective graph coloring problem is sometimes also called the partition graph coloring problem.

Frota et al. [39] present a branch-and-cut algorithm for the partition graph coloring problem. It is based on an integer linear programming formulation that generalizes the 0-1 formulation for the graph coloring problem presented in [28] and [29]. With a branching strategy the Partition Graph Coloring Problem is decomposed in two subproblems, and the linear relaxation bound is improved by means of inequalities.

In an earlier publication Li et al. [22] proved that the selective graph coloring problem is as hard as standard vertex coloring. They also proposed extensions of well-known vertex coloring heuristics to the partition coloring problem and applied these heuristics to some instances of the routing and wavelength assignment problem. This paper also cites a lot of papers that deal with theoretical aspects of the routing and wavelength assignment problem.

Noronha et al. [34] propose a heuristic for solving the Partition Graph Coloring Problem based on tabu search.

The paper by Choi et al. [23] reviews various algorithms for solving the routing and wavelength assignment problem. We can learn from this paper that there are actually two types of algorithms: the ones assuming static traffic either have the objectivity to "minimize the required number of wavelengths in order to ac-

commodate a given set of connections" (this corresponds to the SGCP) or to "maximize the number of connections accommodated if the number of wavelengths is limited". The other type of algorithms assumes dynamic traffic, which means that "connection requests arrive to and depart from the network one by one in a random manner" and the objective is to "minimize the blocking probability". This shows that routing and wavelength assignment actually comprises more problems than just selective graph coloring. However, the paper focuses on the problem variants that are related to selective graph coloring. It breaks the problem down into two subproblems, each of them being NP-complete. The "routing problem" is nothing but the problem to search for a subgraph that hopefully yields a small chromatic number. The paper states that diverse variants of shortest path algorithms are most commonly used for this problem. Regarding the problem of selecting a solution, the paper mentions two classes of methods: sequential selection (by means of greedy algorithms) and combinatorial selection.

## 2.2 Graph Coloring

A large number of papers have been published on graph coloring. One of the oldest that is still frequently cited is [3], in which the author proved (among many other things) that the graph coloring problem is NP-equivalent. Another one of the early papers is [5], in which the author proposes a greedy algorithm for graph coloring (recursive largest first algorithm) that yields better results than the algorithms for this problem known before (such as the randomly ordered sequential algorithm, the largest first algorithm, the smallest last algorithm, interchange algorithms and the approximately maximum independent set algorithm). This algorithm is still used nowadays, as the paper [43] shows, which presents an efficient implementation of it. The basic idea of the RLF algorithm is that in each iteration it selects (if possible) a node that is not adjacent to any colored node and that is connected to the largest number of uncolored nodes that are adjacent to some colored node. If that is not possible, the process is repeated recursively on the subgraph induced by the uncolored nodes.

A more recent classic paper is [10], in which Mehrotra and Trick propose an approach to graph coloring that is based on integer linear programming and makes use of a technique called column generation. With this technique it is possible to obtain an exact solution more efficiently since at first only a part of the problem is added to the integer linear program and then, depending on the results of a dual program, it is decided whether the primal program is expanded. This approach can also be used for computing lower bounds by means of a linear relaxation.

The paper [42] is based on Mehrotra's and Trick's approach and presents an implementation of it which is supposed to provide "numerically safe results, independent of the floating-point accuracy of the linear programming software used".

The technique of column generation is also used in [44].

Hertz et al. [37] propose a novel heuristic algorithm for graph coloring which they call variable space search. It is a variant of local search which considers "several search spaces, with various neighborhoods and objective functions". Whenever the search does not manage to overcome a local optimum, the algorithm moves from one search space to another. The algorithm actually does not try to compute the chromatic number, but it tries whether a graph can be colored with a given number of colors $k$. Thus several runs are needed in order to determine a tight upper bound for the chromatic number.

Lue et al. [41] introduce a memetic algorithm for graph coloring, that is a heuristic algorithm that combines an evolutionary algorithm with more traditional types of heuristics. A genetic algorithm is also used by [24], in combination with a column generation technique.

All the algorithms for graph coloring can be used in programs that try to solve the selective graph coloring problem, for the subproblem of determining the chromatic number of a solution. In order to come up with new solutions, a large number of metaheuristics can be used. For this reason, there is an enormous number of different approaches for the selective graph coloring problem that may lead to success. Therefore we can expect that researchers are yet going to publish a lot of papers dealing with the SGCP.

# Chapter 3

# Methods

In this chapter I am going to present all the methods for solving the SGCP I have implemented. This also includes a couple of methods which I decided to abandon after the first test runs because they turned out to be too inefficient.

Basically, I use the variable neighborhood search (VNS) metaheuristic to search the solution space and various exact as well as heuristic methods to evaluate the solutions. Evaluating a solution basically means to obtain either the exact value or an approximation of its chromatic number. For this purpose I have implemented an exact method using integer linear programming as well as several heuristics determining upper and lower bounds.

There are various approaches to estimate the chromatic number of a graph. One of them is to compute an upper bound by means of a graph coloring algorithm. In order to determine whether a new solution is better than the currently best known solution, it makes sense to additionally compute a couple of other parameters apart from the upper bound. One option is to compute a lower bound, so that a solution is considered the better one if it has the same upper bound, but a smaller lower bound. Another possibility is to compute the number of conflicts that would occur if the graph was colored using one color less than estimated by the greedy coloring algorithm. If two solutions have the same upper bound, it makes sense to prefer the one with the lower number of conflicts.

Since I was interested in an efficient exploration of the search space (I aimed for an execution time not exceeding 10 minutes per instance), I chose a greedy coloring algorithm for the upper bound. It has a polynomial run time with regard to the number of nodes since each node is assigned a color only once, and to determine the color of a node only each neighboring node needs to be considered once. For computing the number of conflicts if one color less were used, each node needs to be visited only once after the graph coloring algorithm has been executed, so the conflict determination algorithm has linear run time with respect to the number

of nodes. What is more costly is the computation of a (reasonable) lower bound. One (straight-forward but very inefficient) method for this is to compute the size of the maximal clique in the graph, which has an exponential run time in terms of the number of nodes. Known more efficient approaches utilize linear programming with column generation, but they are not polynomial time algorithms, either.

It is also possible to compute the exact value of the chromatic number by means of integer linear programming, but due to the NP-hardness of this variant it is only feasible for instances with a rather low number of nodes. For other instances it takes far too long.

All variants of the algorithm can be enhanced by a solution archive, which is an efficient way to check whether a solution has already been discovered and evaluated once. If a duplicate is found, it does not have to be regarded again and a new solution can be easily computed using a converting function. During tests, however, the solution archive has not led to a significant improvement of the quality of solutions.

## 3.1 Metaheuristics

What are metaheuristics? In his book [38], Sean Luke defines them as a "major subfield" of stochastic optimization, which is "the general class of algorithms and techniques which employ some degree of randomness to find optimal (or as optimal as possible) solutions to hard problems". They are applied to "I know it when I see it" problems. These are problems in which it is not easy to find the optimal (or even a good) solution, but if you have a solution, you can test it and see how good it is.

Basically metaheuristics are methods of exploring some defined solution space. They do not guarantee that you will find the optimal solution, but they are supposed to make you find solutions that come close to the optimum. Usually the solution space contains various local optima. For a good metaheuristic it is important to be able to overcome local optima since an algorithm that gets stuck with some local optimum will most likely not find the global optimum. According to the No Free Lunch Theorem [11], the outputs of the various kinds of metaheuristics in general are statistically identical. This is because good solutions are usually scattered all about the search space due to the high degree of Kolmogorov randomness almost all objective functions have [25]. Therefore, various metaheuristics can be used for the same problem with similar chances of obtaining good results in a reasonable time. And yet, some metaheuristics may be particularly suitable for some special problems.

### 3.1.1 Solution Representation

A valid solution of the SGCP is a subgraph $S = G(W)$, $W =< v_1, ..., v_n >$ that consists of $n$ nodes which are all elements of the set of nodes $V$ of the graph $G$. Each of these $n$ nodes must be part of a different cluster within $G$. Since there are exactly $n$ clusters, this means that the solution subgraph contains exactly one node per cluster. Moreover, the edges that connect these nodes are part of the solution subgraph.

For this reason, it makes sense to encode the solution as an array of integer values. Bounds and/or the exact value of the chromatic number are stored in separate variables.

### 3.1.2 Initialization

After loading the graph from a file or, alternatively, generating a new, random graph, the program has to find an initial solution to start with. Of course it could simply choose a random node from the set of nodes of each cluster. But this approach would probably not often yield to a good solution. To come up with a better initial solution, I devised and implemented the following greedy construction heuristics:

For the first cluster, select a random node from its set of nodes. Then select the nodes for the other clusters in ascending order. Always compute the degree of each node within the solution subgraph and select one of the nodes that have the lowest degree.

This ensures that the maximal degree of the nodes in the initial solution is rather small. My hypothesis is that the smaller the degree of the maximal nodes, the more likely will it be possible to achieve a good value for the chromatic number. As I will explain later, the maximal degree of a graph plus one is an upper bound of its chromatic number, although usually not a very tight one.

Algorithm 1 provides pseudocode that demonstrates how the initialization procedure works.

**Algorithm 1:** Initialization

```
select random node of cluster C_0;
for i = 1, ..., number of clusters - 1 do
    bestIdx = 0;
    bestCnt = 0;
    for x ∈ nodes of cluster i do
        if degree of node x < bestCnt then
            bestIdx = x;
            bestCnt = degree of node x;
    select node with index bestIdx from the set of nodes of cluster i;
```

### 3.1.3 Local Search

Local search is a very simple metaheuristic which the more sophisticated variable neighborhood descent makes use of. Basically, the heuristic scans through all the solutions that are neighbors to a given initial solution. For example, the set of neighbors may be the set of all solutions in which for one single cluster a different node has been chosen than in the initial solution. The aim of local search is to find a solution that is better than the initial solution. Upon finding such a solution, another iteration of local search may be done, with the new solution acting as the initial solution. The algorithm stops when no further improvement is possible.

In general there are three different strategies for local search. The one I used is called first improvement. As soon as local search finds a solution that is better than the initial solution, it stops. The initial solution is then overwritten by the new (better) solution and a new iteration may be performed. Another strategy is best improvement. An algorithm that is based on this strategy scans the entire neighbor space and stores a pointer to the best solution, i.e. a local optimum. Then this local optimum is used as the initial solution for the next iteration. A third strategy is simply to choose a random neighbor.

In my program the neighborhoods differ by the number of nodes that are exchanged. In neighborhood number $k$, new nodes for $k$ clusters are chosen.

Algorithm 2 shows in pseudocode how local search with first improvement works.

**Algorithm 2:** Local Search with First Improvement

```
while stopping criterion is not fulfilled do
    for S ∈ the set of neighbor solutions of initial solution S_0 do
        if S is better than S_0 then
            S_0 = S;
            exit the for loop;
```

### 3.1.4 Variable Neighborhood Descent

In variable neighborhood descent (VND) [21], we use more than one neighborhood structure. The reason for this is that we can compare various local optima in this way. VND begins with one neighborhood structure and performs a local search. If the local search has been successful, i.e. a better solution than the initial one has been found, another iteration of local search is performed with some neighborhood structure. Otherwise, the neighborhood structure is switched. If the last neighborhood structure has been used and still no improvement has been found, the algorithm exits, since we are stuck in a local optimum.

In my solution algorithm for the SGCP neighborhood structure $k$ comprises all the neighbors that differ from a reference solution in $k$ nodes.

Algorithm 3 shows a general implementation of VND in pseudocode.

**Algorithm 3:** Variable Neighborhood Descent (input: solution L, output: solution L)

```
i = 0;
while i ≤ last neighborhood structure do
    perform local search on solution L with current neighborhood
    structure i;
    if local search was successful then
        i = 0;
    else
        i = i + 1;
```

### 3.1.5 Variable Neighborhood Search

A further improvement with the aim to overcome local optima is variable neighborhood search (VNS) [16]. This metaheuristic makes use of either a local search or a VND. With one of these techniques, it tries to find a solution better than the initial one. If this attempt has been successful, the VNS resets a variable

representing a counter to zero, otherwise it increases it by one. In either case, the VNS modifies the discovered solution by means of a procedure called "shaking". If the counter has reached its maximal value, the algorithm quits, otherwise another iteration of local search or VND is performed, and so on.

Algorithm 4 demonstrates VNS.

**Algorithm 4:** Variable Neighborhood Search (input: solution L, output: solution L)

```
counter = 0;
while counter < maximal counter value + 1 do
    perform shaking of current solution L;
    perform local search or VND on L;
    if local search or VND was successful then
        counter = 0;
    else
        counter = counter + 1;
```

For the SGCP, a reasonable shaking procedure would be to change the selected node for a given number of clusters. In my implementation this number depends on the value of the counter variable used by the VNS, to which I add the maximum value of $k$ used in VND according to the current settings. So if the counter variable is low, only a relatively small number of nodes will be changed. This makes sense as a higher value of the counter variable means that the search procedure has failed to discover a better solution several times, so we have to get away from that local optimum.

Algorithm 5 shows a generic implementation of shaking in pseudocode.

**Algorithm 5:** Shaking (input: solution L, output: solution L)

```
for i = 1, ..., number of nodes to be changed do
    c = random number between 0 and the number of clusters - 1;
    count = 0;
    for x ∈ nodes of cluster c do
        count = count + 1;
    x = random number between 0 and count − 1;
    deselect the node in L that is currently selected from the set of
    nodes of cluster c;
    select node in L with index x from the set of nodes of cluster c;
```

## 3.2 Solution Evaluation

The process of solution evaluation comprises the computation of either the exact chromatic number of a given solution or an upper bound for it, plus a lower bound or the number of conflicts that would occur if one color less were used for coloring the graph.

### 3.2.1 Upper Bound: Maximal Degree

A very simple and fast method for computing an upper bound for the chromatic number is to compute the maximal degree of the graph. The maximal degree of a graph increased by one is an upper bound for the chromatic number of the graph. This works for any graph. However, this bound is not very tight in the general case. Therefore this upper bound should only serve as a temporary value which is to be refined by a more sophisticated algorithm, such as the greedy coloring algorithm I am going to describe in the next section.

The validity of this upper bound can be easily seen by remembering that a valid coloring of a graph is a coloring such that any pair of nodes $u$ and $v$ which are connected by an edge $(u, v)$ have two different colors. The degree of a node is its number of neighbors, and the maximal degree of a graph is the degree of the node that has the largest number of neighbors. If all $d$ neighbors of some node $u$ have different colors, then $u$ must be in yet another color. So if $d$ is the number of neighbors of the node with the largest degree, the number of colors we need in order to obtain a valid coloring of the graph is at most $d + 1$.

In his paper [1] R. L. Brooks proved that for a connected, simple graph $G$, the chromatic number is always at most equal to the maximal degree of $G$, unless $G$ is a complete graph or an odd cycle. So only if $G$ is a complete graph or an odd cycle, it may be necessary to add one to the maximal degree in order to obtain the chromatic number. A complete graph is a graph in which every node $u$ is connected by an edge to every other node $v$. An odd cycle is a cycle (i.e. a structure in which there is a path from any node belonging to this structure to itself) that consists of an odd number of nodes. To determine whether $G$ is a complete graph, it suffices to check whether all elements of the adjacency matrix have been set to true; this can be done in polynomial time with respect to the number of nodes. For checking whether $G$ is an odd cycle, depth first search may be employed, which has a time complexity of $O(|V|+|E|)$, so it is polynomial as well.

However, since the upper bound obtained by computing the maximal degree of the graph will later be refined by means of greedy coloring, it is enough to take the maximal degree plus one. This computation can be done in polynomial time with respect to the number of nodes.

### 3.2.2 Upper Bound: Greedy Coloring

A tighter upper bound for the chromatic number can be obtained by a graph coloring algorithm, such as greedy coloring. There are many variants of greedy coloring. All of them have in common that they determine the color of each node only once and always assign the color with the lowest feasible index. Therefore they have polynomial run-time and thus are very efficient. The result of such an algorithm is always a valid upper bound, but the tightness of this upper bound depends on the order in which the nodes have been chosen. Some variants of greedy coloring have been shown to perform very poorly.

However, initial tests led to the impression that visiting the nodes in the order of breadth first search leads to pretty good results. As the initial node to start with, my algorithm chooses one of the nodes with the largest degree. Then it stores its neighbors in the queue of nodes that yet have to be colored, and after coloring the second node, it adds the neighbors of the second node to the queue, and so on.

Algorithm 6 provides pseudocode for my variant of greedy coloring.

**Algorithm 6:** Greedy Coloring (input: $S = G(W)$, output: chromatic number upper bound

```
queue Q = empty;
u = node in W with maximal degree;
assign color 0 to node u;
maxCol = 0;
for v ∈ all nodes of the graph except u do
    if there is an edge (u, v) then
        add v to queue Q;
stopCondition = false;
while stopCondition == false do
    while Q is not empty do
        u = next node from Q;
        set all elements of array colorFeasible to true;
        for v ∈ W {u} do
            if there is an edge (u, v) then
                if a color c has been assigned to node v then
                    colorFeasible_c = false;
                else
                    if v is not in queue Q then
                        add v to queue Q;
        c = lowest number for which colorFeasible_c == true;
        assign color c to node u;
        if c > maxCol then
            maxCol = c;
    stopCondition = true;
    for u ∈ W do
        if u has not been assigned a color yet then
            add u to queue Q;
            stopCondition = false;
            exit the for loop;
return maxCol + 1;
```

### 3.2.3 Minimal Conflicts Heuristic

After computing an upper bound for the chromatic number by means of greedy coloring, it makes sense to compute the number of conflicts that would arise if one color less were used for coloring the graph. A conflict is an edge $(u, v)$ where nodes $u$ and $v$ have been assigned the same color. To compute the number of

conflicts, the program takes a look at all the nodes that have been assigned the color with the largest index. The least number of neighboring nodes that share their color is the number of conflicts this node generates.

The number of conflicts may be used as an additional criterion whether a solution is better than another one with the same chromatic number.

Algorithm 7 shows a heuristic for the number of conflicts of a solution.

**Algorithm 7:** Number of Conflicts (input: colorization of $S$, output: number of conflicts

```
numConflicts = 0;
for u ∈ W do
    if u has the least often used color then
        minOcc = ∞;
        for c ∈ all possible colors do
            count = 0;
            for v ∈ all neighbors of u do
                count = count + 1;
            if count < minOcc then
                minOcc = count;
        numConflicts = numConflicts + minOcc;
return numConflicts;
```

### 3.2.4 Lower Bound: Maximal Clique

If there were an efficient algorithm to compute a tight lower bound for the chromatic number of a graph, it could be used as an additional evaluation criterion for the solution subgraphs.

As explained in [42], the size of the maximal clique of a graph is a rather tight lower bound for its chromatic number. A clique is a subgraph that, if isolated, would be a complete graph. In other words, for all pairs of nodes $u$ and $v$ that belong to the clique, there exists an edge $(u, v)$. Since every node in a clique is connected to every other node, all the nodes must be assigned pairwise different colors. Therefore, the chromatic number of a graph in which this clique appears cannot be lower than the size of the clique.

Unfortunately, computing the size of the maximal clique has exponential runtime in terms of the number of nodes. Therefore it does not make sense to use it in practice. But there is another method for computing a lower bound,

employing linear programming. I will explain it shortly. First let me introduce linear programming and explain how it can be used to obtain the exact chromatic number.

### 3.2.5 Exact Method: Integer Linear Programming

It is possible to obtain the exact value of the chromatic number of a graph by means of integer linear programming. However, this method is not very efficient.

Linear programming is one of the main methods of operations research. It can be applied for systems of linear inequalities that come along with an optimality criterion, such as the maximization or minimization of some scalar product of two vectors. Linear programming can be solved by means of various algorithms, including the simplex method, the ellipsoid method and the interior point method. There are also techniques for solving some special cases of problems, such as Dantzig-Wolfe decomposition, benders decomposition and linear relaxation.

To solve linear programs efficiently, various commercial libraries exist, such as ILOG CPLEX, which I used in my implementation.

Integer linear programming differs from linear programming in that the variables can only take discrete values. It is generally much harder to solve.

The standard formulation of integer linear programming for graph coloring, which is also the one I implemented, is as follows:

$x_{v,c}$ be a variable that determines whether color $c$ is assigned to node $v$. If so, it is $1$, otherwise it is $0$.

Minimize the number of colors for which $x_{v,c} = 1$

| | | |
|---|---|---|
| subject to | $\sum_{c=1}^{k} x_{v,c} = 1$ | $\forall$ nodes $v \in V$ |
| and | $x_{u,c} + x_{v,c} \geq 1$ | $\forall$ colours $c \in K$ $\forall$ edges $(u, v) \in E$ |
| where | $x_{v,c} = 1$ | if node $v$ is assigned color $c$, otherwise 0 |

The objective is to minimize the number $i$ of colors $c$ for which there exists at least one node $v$ with $x_{v,c} = 1$.

A linear relaxation of this program - that is, a variant in which $x_{v,c}$ is not limited to 0 or 1 and can take any value in between - can be used to obtain a lower bound. However, there is also a more efficient method for computing tight lower bounds which makes use of an alternative formulation of the linear program and the column generation technique.

### 3.2.6 Lower Bound: Linear Programming with Column Generation

This solution was originally proposed by Mehrotra and Trick in [10] for computing the exact value of the chromatic number, but it can also be employed for finding a tight lower bound. This approach is based on the notion of independent sets. An independent set is a subset of the node set $V$ which has the property that no nodes of this subset are connected. Clearly, two nodes belonging to the same independent set can be safely assigned the same color. So if we start with an empty set of nodes and consecutively add independent sets to it until all nodes of the original graph are in this set, we can compute an upper bound for the chromatic number, and if we manage to find the least possible number of independent sets that need to be added to the empty set so that it becomes identical to the set of nodes $V$, this number is the exact value of the chromatic number. This approach can be formulated as a linear program as follows:

The variable $x_s$ equals $1$ if independent set $s$ is included in the solution.

$$\begin{array}{lll} \text{Minimize } \sum_{s} x_s & & \\ \text{subject to } \sum_{s:i \in S} x_s \geq 1 & & \forall i \in V \\ \text{where } x_s \in \{0,1\} & & \forall s \in S \end{array}$$

If we apply linear relaxation on this program, so that $x_s$ may take any value between 0 and 1, this formulation can be used to obtain a tight lower bound. However, it is still not efficient because of the large number of variables it generates. The trick to make it more efficient is to only start with a minimal number of independent sets (thus minimizing the number of constraints), then solve the linear program and decide by means of a dual program whether it makes sense to add another variable.

To determine whether another independent set should be added to the program, the following dual problem has to be solved:

$$\begin{array}{lll} \text{Maximize } \sum_{i \in V} \pi_i z_i & & \\ \text{subject to } z_i + z_j \leq 1 & & \forall (i,j) \in E \\ \text{where } z_i \in \{0,1\} & & \forall i \in V \end{array}$$

Here, $\pi_i$ stands for the dual value of constraint number $i$ in the primal problem.

## 3.3 Solution Archive

Solution archives [45] are a rather new technique. The idea is to store valid solutions that have already been evaluated so that they need not be evaluated again in case the metaheuristic (e.g. an evolutionary algorithm) stumbles across them again. A compact encoding of the solution is achieved via the data structure "trie", which is a tree in which one element of the solution array is stored in each node, so to obtain a complete solution, we have to walk through the tree from the root until the leaf representing the final element of the array. This data structure has the advantage that the look-up time is independent of the number of solutions stored in the trie.

Each trie node consists of a value, a pointer to one of its children and a pointer to its "right" neighbor. In case of our problem, it makes sense to have each level of the trie represent one cluster and have the value of the trie node be the number of the selected node from the original graph.

If a solution already appears in the solution archive, it is possible to derive a new solution. The algorithm has to traverse through the trie until it finds a level that is not complete (i.e., not all nodes of the next level are already children), and then it can build a new solution starting from the subsequent level by random node selection.

To check whether a solution candidate is already in the archive, a hash map is used, as it speeds up retrieval. To compute the hash value, various hash functions can be used. One possibility is to compute $h(x) = (x_0 + x_1 b + x_2 b^2 + ... + x_n b^n)$ mod $k$, where $k$ is the size of the hash table, $b$ is the basis and $x_i$ is the number of the node that has been selected for cluster $i$.

## 3.4 Complete Algorithm

The complete algorithm is a VNS making use of the aforementioned heuristic and exact methods for computing the chromatic number and thus evaluating solution candidates. First an initial solution is computed by means of the initialization method. Then a counter is set to 0, and while this counter is lower than some maximal value (default: 5) and the time limit of 600 seconds has not expired, the VNS calls a VND procedure to search for better solutions. If a better solution is found, the counter is reset to 0, otherwise the counter is incremented by 1. The solution is then either converted to a new solution by means of the solution archive or by means of a shaking procedure. Then the next iteration of the loop happens.

Inside the VND procedure, a counter variable $k$ is initialized with 1. In each

iteration neihgborhood number $k$ is searched. If a solution is found that is better than the currently best known solution, this solution is taken as the new best solution, and $k$ is reset to 1. Once a neighborhood has been completely searched, $k$ is increased by 1. When $k$ equals $maxK$, the VND quits.

Algorithms 8 and 9 give an outline of the complete algorithm.

**Algorithm 8:** Complete algorithm

```
BestSol.Initialize;
ShakedBestSol = BestSol;
Counter = 0;
while Counter < 5 and Expired time < 600 do
    TempSol = VND(ShakedBestSol);
    if TempSol.IsBetter(ShakedBestSol) then
        Counter = 0;
        BestSol = TempSol;
    else
        Counter = Counter + 1;
    if solution archive is used then
        ShakedBestSol = SolutionArchive.Convert(ShakedBestSol);
    else
        ShakedBestSol = Shake(ShakedBestSol);
return BestSol;
```

**Algorithm 9:** Procedure VND

```
k = 0;
while k ≤ maxK do
    TempSol = NextSol(ShakedBestSol, k);
    if solution archive is used then
        while Hashmap.Find(TempSol) do
            TempSol = NextSol(TempSol, k);
    if neighborhood completely explored then
        k = k + 1;
    if TempSol.IsBetter(ShakedBestSol) and not
    SolutionArchive.Find(TempSol) then
        ShakedBestSol = TempSol;
        k = 0;
if solution archive is used then
    SolutionArchive.Add(ShakedBestSol);
    Hashmap.Add(ShakedBestSol);
return ShakedBestSol;
```

# Chapter 4

# Testing Environment

The implementation was done in C++. As metaheuristic for choosing a candidate solution, the program uses VNS with VND in the inner loop. The maximal value of $k$ in VND has been set to 2 by default, and an additional stopping criterion is the elapsing of 10 minutes since the beginning of the execution of the VNS. The shaking procedure modifies a number of nodes equal to the current value of the counter variable plus the parameter $maxK$, which indicates the last neighborhood structure that is used; if this number is greater than the number of clusters, then the number of clusters is taken instead.

For the exact computation of the chromatic number by means of integer linear programming, ILOG CPLEX 12.5 was used. The program can process input files of the format used in the paper [39].

The following algorithm variants have been implemented:

1. Exact computation of the chromatic number by means of integer linear programming.
2. Upper bound by means of greedy coloring combined with an estimation of the number of conflicts that would occur if one color less were used and the computation of a lower bound: If two solutions have the same upper bound, the solution with the lower number of conflicts will be chosen. If both the upper bounds and the numbers of conflicts are the same, the solution with the smaller lower bound will be chosen.
3. Upper bound by means of greedy coloring combined with an estimation of the number of conflicts that would occur if one color less were used.
4. Upper bound by means of greedy coloring.

For each variant there are two sub-variants, one with and one without a solution archive.

The program can be configured by means of a configuration file, in which the following parameters may be specified:

1. printout: determines what should be printed (all steps, some steps or only the best solution of each run and statistics).
2. conflicts: a factor with which the number of conflicts is multiplied to decide whether a solution that has the same upper bound but a lower number of conflicts will be preferred. It sometimes makes sense to set this to values below 100 percent because otherwise the search would take too long.
3. start, step, stop: specify what variants of the algorithm should be used.
4. runs: the number of runs that should be performed and statistically evaluated.
5. maxk: the maximum value of $k$ for the VND.
6. maxlb: the maximum size of a buffer array used to compute the lower bounds.
7. timelimit: the time limit.
8. maxfail: the maximum number of consecutive fails of VND runs (i.e. VND runs that did not lead to an improvement).

# Chapter 5

# Computational Results

The tests were performed on the grid of the Algorithms and Data Structures group at the Vienna UT. The program was compiled using GNU C/C++ 4.6 and ILOG CPLEX 12.5.

To test the diverse variants of the solution algorithm, the PCP instances provided by the authors of the paper [39] were used. These instances can be freely downloaded from the Internet [40]. In particular instances from the subdirectory "Table2 Random Instances" have been evaluated. The name of each instance is composed of the number of nodes, the edge density, the number of nodes per cluster and the number of the instance.

It turned out that the VNS variants employing integer linear programming to obtain the exact chromatic number only terminated within a reasonable time for the smallest instances (twenty nodes), while a single run already needed more than two hours when applied on the second smallest set of instances (forty nodes). Therefore these variants were abandoned and only the six other variants (upper bound plus conflicts plus lower bound by column generation, upper bound plus conflicts, upper bound without conflicts, each with and without solution archive) were thoroughly tested.

## 5.1 Preliminary Results

Each of these variants was tested on several instances of graphs with 20, 40, 60, 80, 100 and 120 nodes provided by Frota, and on 45 instances of graphs with 90 nodes with varying densities. Each variant was run thirty times (except the slow variants with LB, which were only run three times) and then the results were statistically evaluated by the program (mean and standard deviation). The parameter $maxK$ of the VNS (determining the number of neighborhood structures to be explored) was set to 2.

Moreover, a couple of tests were performed using instances from the paper by Noronha and Ribeiro (dsjc500*) [34]. All of these instances use 500 clusters. The first instance has only one node per cluster, the second two and so on. For these tests $maxK = 1$ was used since searching a 2-opt neighborhood would take long time and due to the time limit of ten minutes the neighborhood would only be partially explored.

Tables 5.1 - 5.16 contain the preliminary results. Legend: t = time (seconds), Result = average value and standard deviation, UB = upper bound, C = conflicts, LB = lower bound computed by means of column generation, SA = solution archive, UB same = number of incidents when a new solution has the same upper bound as the currently best solution, UB+C same = same as "UB same" and the number of conflicts is also the same.

We can see that the variants of the algorithm differ regarding the quality of the result they produce. Adding the minimal conflict heuristic often improves the quality, but not always: in some cases, the algorithm variant that just evaluates the upper bounds yields better results. The solution archive hardly has an effect on the results. A solution archive serves two purposes: it enables the algorithm to find duplicates, and it has a solution conversion function to generate new solutions. Neither of these two functions apparently had a significant effect. Regarding the conversion function, it is not better than the shaking procedure that is used in the variants without a solution archive.

The main conclusion is that UB+C usually brings the best results and is more efficient than UB+C+LB. The latter variant takes much processing time and for this reason the time limit usually expires before the search is over. That is why often the best result UB+C finds is neglected by UB+C+LB. On the other hand, UB+C finds better solutions than UB most of the time as UB discards solutions with the same upper bound but a lower number of conflicts. These solutions are however considered by UB+C and in the end this strategy yields better results.

For the Frota instances in tables 5.1 - 5.15, evaluating the lower bound sometimes leads to a slight, but insignificant improvement. This leads to the conclusion that it is more efficient not to evaluate the lower bound. After all, it makes use of linear programming, which is time-consuming.

The Noronha instance dsjc500.5-1 in table 5.16 contains only one node per cluster. For this reason all the variants yield the same result, since effectively nothing else than a run of the greedy coloring heuristic is performed. Thus this instance serves as a benchmark to evaluate the effectiveness of the coloring algorithm.

Apparently the solution archive does not make any difference in table 5.16; I counted the number of duplicates and noticed that no duplicate is ever found.

Perhaps some duplicates would be found if the time limit had been larger and if $maxK$ had been set to a larger value. For these instances, evaluating the lower bound leads to worse results. This is most probably due to the long run time of the lower bound computation, which has the effect that fewer solutions are explored within the time limit.

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n20p5t2s1 | UB+C+LB | 16.0 | 3.0 ± 0.0 | 35.0 | 12.3 |
| | UB+C+LB+SA | 15.3 | 3.0 ± 0.0 | 29.0 | 9.0 |
| | UB+C | 0.0 | 3.0 ± 0.0 | 40.6 | 13.3 |
| | UB+C+SA | 0.0 | 3.0 ± 0.0 | 31.7 | 9.7 |
| | UB | 0.0 | 3.0 ± 0.0 | 18.6 | – |
| | UB+SA | 0.0 | 3.0 ± 0.0 | 15.7 | – |
| n20p5t2s2 | UB+C+LB | 13.7 | 3.0 ± 0.0 | 67.7 | 6.7 |
| | UB+C+LB+SA | 14.3 | 3.0 ± 0.0 | 69.0 | 5.0 |
| | UB+C | 0.0 | 3.0 ± 0.0 | 69.4 | 7.9 |
| | UB+C+SA | 0.0 | 3.0 ± 0.0 | 61.0 | 4.3 |
| | UB | 0.0 | 3.0 ± 0.0 | 70.3 | – |
| | UB+SA | 0.0 | 3.0 ± 0.0 | 58.5 | – |
| n20p5t2s3 | UB+C+LB | 17.3 | 3.0 ± 0.0 | 98.3 | 7.0 |
| | UB+C+LB+SA | 18.3 | 3.0 ± 0.0 | 98.3 | 4.0 |
| | UB+C | 0.0 | 3.0 ± 0.0 | 99.9 | 6.8 |
| | UB+C+SA | 0.0 | 3.0 ± 0.0 | 100.7 | 4.0 |
| | UB | 0.0 | 3.0 ± 0.0 | 89.7 | – |
| | UB+SA | 0.0 | 3.0 ± 0.0 | 77.0 | – |
| n20p5t2s4 | UB+C+LB | 15.3 | 3.0 ± 0.0 | 114.3 | 17.0 |
| | UB+C+LB+SA | 16.0 | 3.0 ± 0.0 | 129.0 | 9.7 |
| | UB+C | 0.0 | 3.0 ± 0.0 | 105.5 | 14.2 |
| | UB+C+SA | 0.0 | 3.0 ± 0.0 | 106.9 | 9.0 |
| | UB | 0.0 | 3.0 ± 0.0 | 70.2 | – |
| | UB+SA | 0.0 | 3.0 ± 0.0 | 75.2 | – |
| n20p5t2s5 | UB+C+LB | 13.7 | 3.0 ± 0.0 | 24.0 | 5.0 |
| | UB+C+LB+SA | 14.7 | 3.0 ± 0.0 | 23.7 | 3.7 |
| | UB+C | 0.0 | 3.0 ± 0.0 | 24.9 | 4.5 |
| | UB+C+SA | 0.0 | 3.0 ± 0.0 | 24.1 | 1.7 |
| | UB | 0.0 | 3.0 ± 0.0 | 19.0 | – |
| | UB+SA | 0.0 | 3.0 ± 0.0 | 22.0 | – |

**Table 5.1:** Preliminary Results: Instances with 20 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n40p5t2s1 | UB+C+LB | 159.7 | 5.0 ± 0.0 | 161.7 | 12.3 |
| | UB+C+LB+SA | 161.3 | 5.0 ± 0.0 | 120.0 | 10.7 |
| | UB+C | 0.0 | 4.7 ± 0.5 | 149.3 | 13.1 |
| | UB+C+SA | 0.1 | 5.0 ± 0.0 | 132.0 | 10.0 |
| | UB | 0.0 | 4.7 ± 0.5 | 139.5 | – |
| | UB+SA | 0.0 | 5.0 ± 0.0 | 134.4 | – |
| n40p5t2s2 | UB+C+LB | 228.7 | 5.0 ± 0.0 | 488.7 | 42.7 |
| | UB+C+LB+SA | 267.3 | 4.7 ± 0.5 | 304.0 | 29.7 |
| | UB+C | 0.1 | 4.9 ± 0.2 | 336.4 | 31.8 |
| | UB+C+SA | 0.1 | 4.8 ± 0.4 | 278.3 | 23.3 |
| | UB | 0.1 | 5.0 ± 0.0 | 246.3 | – |
| | UB+SA | 0.1 | 5.0 ± 0.2 | 195.1 | – |
| n40p5t2s3 | UB+C+LB | 203.7 | 5.0 ± 0.0 | 286.3 | 16.0 |
| | UB+C+LB+SA | 166.0 | 5.0 ± 0.0 | 223.3 | 8.3 |
| | UB+C | 0.0 | 5.0 ± 0.0 | 251.3 | 13.2 |
| | UB+C+SA | 0.1 | 5.0 ± 0.0 | 210.5 | 8.7 |
| | UB | 0.0 | 5.0 ± 0.0 | 87.4 | – |
| | UB+SA | 0.0 | 5.0 ± 0.0 | 77.3 | – |
| n40p5t2s4 | UB+C+LB | 209.3 | 5.0 ± 0.0 | 278.0 | 20.3 |
| | UB+C+LB+SA | 155.7 | 5.0 ± 0.0 | 84.7 | 11.3 |
| | UB+C | 0.0 | 4.8 ± 0.4 | 213.1 | 18.8 |
| | UB+C+SA | 0.0 | 5.0 ± 0.0 | 104.6 | 11.3 |
| | UB | 0.1 | 5.0 ± 0.0 | 59.9 | – |
| | UB+SA | 0.0 | 5.0 ± 0.0 | 45.9 | – |
| n40p5t2s5 | UB+C+LB | 203.7 | 4.7 ± 0.5 | 53.0 | 8.0 |
| | UB+C+LB+SA | 152.7 | 5.0 ± 0.0 | 56.0 | 7.7 |
| | UB+C | 0.0 | 4.8 ± 0.4 | 54.3 | 8.6 |
| | UB+C+SA | 0.1 | 5.0 ± 0.0 | 49.7 | 6.3 |
| | UB | 0.0 | 5.0 ± 0.0 | 28.1 | – |
| | UB+SA | 0.1 | 5.0 ± 0.0 | 35.8 | – |

**Table 5.2:** Preliminary Results: Instances with 40 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n60p5t2s1 | UB+C+LB | 600.0 | 6.0 ± 0.0 | 82.7 | 18.0 |
| | UB+C+LB+SA | 607.3 | 6.0 ± 0.0 | 73.7 | 17.0 |
| | UB+C | 0.3 | 6.0 ± 0.0 | 109.8 | 18.3 |
| | UB+C+SA | 0.3 | 6.0 ± 0.0 | 77.7 | 14.0 |
| | UB | 0.2 | 6.9 ± 0.2 | 77.9 | – |
| | UB+SA | 0.3 | 7.0 ± 0.0 | 42.2 | – |
| n60p5t2s2 | UB+C+LB | 600.0 | 6.0 ± 0.0 | 82.7 | 18.0 |
| | UB+C+LB+SA | 607.3 | 6.0 ± 0.0 | 73.7 | 17.0 |
| | UB+C | 0.3 | 6.0 ± 0.0 | 109.8 | 18.3 |
| | UB+C+SA | 0.3 | 6.0 ± 0.0 | 77.7 | 14.0 |
| | UB | 0.2 | 7.0 ± 0.2 | 77.9 | – |
| | UB+SA | 0.3 | 7.0 ± 0.0 | 42.2 | – |
| n60p5t2s3 | UB+C+LB | 601.0 | 6.0 ± 0.0 | 79.0 | 15.0 |
| | UB+C+LB+SA | 603.3 | 6.0 ± 0.0 | 75.3 | 14.0 |
| | UB+C | 0.3 | 6.0 ± 0.0 | 213.6 | 18.2 |
| | UB+C+SA | 0.3 | 6.0 ± 0.0 | 201.7 | 15.0 |
| | UB | 0.2 | 7.0 ± 0.0 | 162.2 | – |
| | UB+SA | 0.3 | 7.0 ± 0.0 | 125.4 | – |
| n60p5t2s4 | UB+C+LB | 606.3 | 6.0 ± 0.0 | 96.7 | 9.3 |
| | UB+C+LB+SA | 601.3 | 6.0 ± 0.0 | 94.3 | 9.0 |
| | UB+C | 0.3 | 6.0 ± 0.0 | 79.0 | 13.1 |
| | UB+C+SA | 0.3 | 6.0 ± 0.0 | 63.4 | 8.7 |
| | UB | 0.3 | 6.6 ± 0.5 | 222.3 | – |
| | UB+SA | 0.2 | 7.0 ± 0.0 | 182.9 | – |
| n60p5t2s5 | UB+C+LB | 601.3 | 6.0 ± 0.0 | 95.7 | 20.0 |
| | UB+C+LB+SA | 602.0 | 6.0 ± 0.0 | 76.3 | 14.3 |
| | UB+C | 0.3 | 6.0 ± 0.0 | 127.4 | 25.7 |
| | UB+C+SA | 0.3 | 6.0 ± 0.0 | 82.5 | 16.6 |
| | UB | 0.3 | 6.0 ± 0.5 | 330.8 | – |
| | UB+SA | 0.2 | 7.0 ± 0.0 | 175.4 | – |

**Table 5.3:** Preliminary Instances with 60 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n80p5t2s1 | UB+C+LB | 600.7 | 8.0 ± 0.0 | 41.0 | 14.0 |
| | UB+C+LB+SA | 612.0 | 8.0 ± 0.0 | 41.0 | 14.0 |
| | UB+C | 1.0 | 8.0 ± 0.0 | 434.2 | 35.9 |
| | UB+C+SA | 1.1 | 8.0 ± 0.0 | 266.7 | 26.4 |
| | UB | 0.8 | 8.0 ± 0.0 | 144.0 | – |
| | UB+SA | 0.9 | 8.0 ± 0.0 | 95.7 | – |
| n80p5t2s2 | UB+C+LB | 618.7 | 9.0 ± 0.0 | 44.0 | 12.0 |
| | UB+C+LB+SA | 615.0 | 9.0 ± 0.0 | 44.0 | 12.0 |
| | UB+C | 1.0 | 8.0 ± 0.2 | 376.7 | 30.2 |
| | UB+C+SA | 1.0 | 8.0 ± 0.0 | 217.9 | 24.2 |
| | UB | 1.0 | 8.6 ± 0.5 | 265.1 | – |
| | UB+SA | 1.1 | 8.2 ± 0.4 | 164.3 | – |
| n80p5t2s3 | UB+C+LB | 612.0 | 8.0 ± 0.0 | 31.0 | 9.0 |
| | UB+C+LB+SA | 612.3 | 8.0 ± 0.0 | 31.0 | 9.0 |
| | UB+C | 1.1 | 8.0 ± 0.0 | 517.9 | 44.6 |
| | UB+C+SA | 1.2 | 8.0 ± 0.0 | 410.1 | 39.2 |
| | UB | 0.9 | 8.0 ± 0.0 | 177.6 | – |
| | UB+SA | 0.9 | 8.0 ± 0.0 | 142.9 | – |
| n80p5t2s4 | UB+C+LB | 607.0 | 8.0 ± 0.0 | 16.3 | 3.0 |
| | UB+C+LB+SA | 602.7 | 8.0 ± 0.0 | 16.7 | 3.0 |
| | UB+C | 1.1 | 7.9 ± 0.2 | 335.5 | 51.5 |
| | UB+C+SA | 1.0 | 8.0 ± 0.0 | 248.4 | 40.1 |
| | UB | 0.8 | 8.0 ± 0.0 | 92.2 | – |
| | UB+SA | 0.8 | 8.0 ± 0.0 | 79.7 | – |
| n80p5t2s5 | UB+C+LB | 608.7 | 8.0 ± 0.0 | 74.0 | 14.0 |
| | UB+C+LB+SA | 604.7 | 8.0 ± 0.0 | 74.0 | 14.0 |
| | UB+C | 1.0 | 7.0 ± 0.0 | 110.7 | 23.7 |
| | UB+C+SA | 1.0 | 7.0 ± 0.0 | 106.7 | 19.7 |
| | UB | 1.1 | 8.5 ± 0.5 | 325.7 | – |
| | UB+SA | 1.0 | 8.7 ± 0.4 | 334.4 | – |

**Table 5.4:** Preliminary Results: Instances with 80 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n100p5t2s1 | UB+C+LB | 607.7 | 11.0 ± 0.0 | 10.0 | 1.0 |
| | UB+C+LB+SA | 605.0 | 11.0 ± 0.0 | 10.0 | 1.0 |
| | UB+C | 2.6 | 9.0 ± 0.0 | 380.0 | 42.4 |
| | UB+C+SA | 2.6 | 9.0 ± 0.0 | 306.9 | 37.4 |
| | UB | 3.2 | 10.0 ± 0.6 | 521.5 | – |
| | UB+SA | 2.3 | 11.0 ± 0.0 | 228.3 | – |
| n100p5t2s2 | UB+C+LB | 604.3 | 10.0 ± 0.0 | 13.0 | 1.0 |
| | UB+C+LB+SA | 604.3 | 10.0 ± 0.0 | 13.0 | 1.0 |
| | UB+C | 2.3 | 9.0 ± 0.0 | 103.4 | 18.1 |
| | UB+C+SA | 2.2 | 9.0 ± 0.0 | 85.6 | 13.3 |
| | UB | 2.5 | 9.7 ± 0.5 | 193.7 | – |
| | UB+SA | 2.2 | 10.0 ± 0.0 | 98.3 | – |
| n100p5t2s3 | UB+C+LB | 611.0 | 11.0 ± 0.0 | 23.0 | 8.0 |
| | UB+C+LB+SA | 613.7 | 11.0 ± 0.0 | 23.0 | 8.0 |
| | UB+C | 3.0 | 9.0 ± 0.0 | 265.9 | 27.3 |
| | UB+C+SA | 3.0 | 9.0 ± 0.0 | 237.2 | 23.5 |
| | UB | 2.8 | 9.8 ± 0.4 | 403.7 | – |
| | UB+SA | 2.6 | 10.0 ± 0.0 | 324.6 | – |
| n100p5t2s4 | UB+C+LB | 614.3 | 10.0 ± 0.0 | 10.0 | 2.0 |
| | UB+C+LB+SA | 613.7 | 10.0 ± 0.0 | 10.0 | 2.0 |
| | UB+C | 2.5 | 9.0 ± 0.0 | 134.5 | 27.3 |
| | UB+C+SA | 2.5 | 9.0 ± 0.0 | 107.2 | 19.6 |
| | UB | 2.6 | 9.7 ± 0.4 | 296.7 | – |
| | UB+SA | 2.3 | 10.0 ± 0.0 | 197.8 | – |
| n100p5t2s5 | UB+C+LB | 609.0 | 11.0 ± 0.0 | 17.0 | 5.0 |
| | UB+C+LB+SA | 616.3 | 11.0 ± 0.0 | 17.0 | 5.0 |
| | UB+C | 2.4 | 10.0 ± 0.2 | 207.7 | 22.2 |
| | UB+C+SA | 2.3 | 10.0 ± 0.0 | 151.0 | 15.4 |
| | UB | 2.4 | 10.0 ± 0.0 | 150.0 | – |
| | UB+SA | 2.5 | 10.0 ± 0.0 | 128.8 | – |

**Table 5.5:** Preliminary Results: Instances with 100 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n120p5t2s1 | UB+C+LB | 631.3 | 13.0 ± 0.0 | 24.7 | 12.0 |
| | UB+C+LB+SA | 639.0 | 13.0 ± 0.0 | 25.0 | 12.0 |
| | UB+C | 5.2 | 10.9 ± 0.3 | 425.3 | 45.2 |
| | UB+C+SA | 5.0 | 11.0 ± 0.0 | 256.5 | 31.4 |
| | UB | 5.2 | 11.0 ± 0.0 | 393.6 | – |
| | UB+SA | 5.2 | 11.0 ± 0.0 | 233.1 | – |
| n120p5t2s2 | UB+C+LB | 614.7 | 13.0 ± 0.0 | 11.3 | 6.0 |
| | UB+C+LB+SA | 603.0 | 13.0 ± 0.0 | 11.0 | 6.0 |
| | UB+C | 5.0 | 11.0 ± 0.0 | 223.4 | 33.2 |
| | UB+C+SA | 5.1 | 11.0 ± 0.0 | 136.3 | 21.5 |
| | UB | 6.4 | 11.3 ± 0.5 | 526.3 | – |
| | UB+SA | 5.6 | 11.8 ± 0.4 | 272.3 | – |
| n120p5t2s3 | UB+C+LB | 606.7 | 12.0 ± 0.0 | 11.3 | 1.0 |
| | UB+C+LB+SA | 606.3 | 12.0 ± 0.0 | 11.7 | 1.0 |
| | UB+C | 5.0 | 11.0 ± 0.0 | 229.8 | 21.3 |
| | UB+C+SA | 5.1 | 11.0 ± 0.0 | 152.8 | 14.3 |
| | UB | 5.4 | 11.5 ± 0.5 | 166.2 | – |
| | UB+SA | 5.4 | 11.4 ± 0.5 | 148.7 | – |
| n120p5t2s4 | UB+C+LB | 602.7 | 12.0 ± 0.0 | 11.3 | 2.0 |
| | UB+C+LB+SA | 603.7 | 12.0 ± 0.0 | 11.3 | 2.0 |
| | UB+C | 5.9 | 10.8 ± 0.4 | 329.6 | 35.3 |
| | UB+C+SA | 5.6 | 11.0 ± 0.0 | 188.3 | 22.6 |
| | UB | 5.5 | 11.3 ± 0.5 | 338.0 | – |
| | UB+SA | 5.0 | 11.6 ± 0.5 | 165.7 | – |
| n120p5t2s5 | UB+C+LB | 602.0 | 12.0 ± 0.0 | 14.0 | 1.0 |
| | UB+C+LB+SA | 603.7 | 12.0 ± 0.0 | 14.0 | 1.0 |
| | UB+C | 6.2 | 11.0 ± 0.0 | 721.1 | 29.6 |
| | UB+C+SA | 6.3 | 11.0 ± 0.0 | 553.2 | 24.7 |
| | UB | 5.6 | 11.8 ± 0.4 | 397.4 | – |
| | UB+SA | 5.2 | 12.0 ± 0.2 | 215.7 | – |

**Table 5.6:** Preliminary Results: Instances with 120 nodes

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p1t2s1 | UB+C+LB | 600.0 | 3.0 ± 0.0 | 936.7 | 48.3 |
| | UB+C+LB+SA | 600.0 | 3.0 ± 0.5 | 944.7 | 49.0 |
| | UB+C | 0.6 | 3.0 ± 0.0 | 3084.4 | 65.4 |
| | UB+C+SA | 0.7 | 3.0 ± 0.0 | 1846.3 | 55.1 |
| | UB | 0.6 | 3.0 ± 0.0 | 703.9 | – |
| | UB+SA | 0.6 | 3.0 ± 0.0 | 331.2 | – |
| n90p1t2s2 | UB+C+LB | 600.0 | 3.0 ± 0.0 | 764.7 | 31.3 |
| | UB+C+LB+SA | 600.0 | 3.0 ± 0.0 | 725.3 | 30.7 |
| | UB+C | 0.6 | 3.0 ± 0.0 | 2002.2 | 40.6 |
| | UB+C+SA | 0.6 | 3.0 ± 0.0 | 1326.9 | 34.5 |
| | UB | 0.6 | 3.0 ± 0.0 | 464.8 | – |
| | UB+SA | 0.6 | 3.0 ± 0.0 | 401.1 | – |
| n90p1t2s3 | UB+C+LB | 600.0 | 3.0 ± 0.0 | 571.7 | 33.0 |
| | UB+C+LB+SA | 600.0 | 3.0 ± 0.0 | 596.3 | 33.0 |
| | UB+C | 0.8 | 3.0 ± 0.0 | 3361.3 | 92.5 |
| | UB+C+SA | 0.9 | 3.0 ± 0.0 | 1862.9 | 74.5 |
| | UB | 0.6 | 3.0 ± 0.0 | 649.0 | – |
| | UB+SA | 0.6 | 3.0 ± 0.0 | 335.1 | – |
| n90p1t2s4 | UB+C+LB | 600.0 | 3.0 ± 0.0 | 996.3 | 92.0 |
| | UB+C+LB+SA | 600.0 | 3.0 ± 0.0 | 1000.0 | 92.0 |
| | UB+C | 0.6 | 3.0 ± 0.0 | 2767.5 | 51.0 |
| | UB+C+SA | 0.6 | 3.0 ± 0.0 | 1775.3 | 42.7 |
| | UB | 0.6 | 3.0 ± 0.0 | 1020.4 | – |
| | UB+SA | 0.5 | 3.0 ± 0.0 | 992.5 | – |
| n90p1t2s5 | UB+C+LB | 600.0 | 3.0 ± 0.0 | 538.7 | 30.0 |
| | UB+C+LB+SA | 600.0 | 3.0 ± 0.0 | 538.7 | 30.0 |
| | UB+C | 0.7 | 3.0 ± 0.0 | 1259.6 | 36.1 |
| | UB+C+SA | 0.7 | 3.0 ± 0.0 | 598.2 | 28.9 |
| | UB | 0.6 | 3.0 ± 0.0 | 458.0 | – |
| | UB+SA | 0.6 | 3.0 ± 0.0 | 318.4 | – |

**Table 5.7:** Preliminary Results: Instances with 90 nodes, density 0.1

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p2t2s1 | UB+C+LB | 600.0 | 4.0 ± 0.0 | 138.3 | 16.3 |
| | UB+C+LB+SA | 600.0 | 4.0 ± 0.0 | 139.0 | 17.0 |
| | UB+C | 0.9 | 4.0 ± 0.0 | 331.9 | 29.4 |
| | UB+C+SA | 0.9 | 4.0 ± 0.0 | 301.6 | 22.4 |
| | UB | 0.8 | 5.0 ± 0.0 | 1015.6 | – |
| | UB+SA | 0.8 | 5.0 ± 0.0 | 623.9 | – |
| n90p2t2s2 | UB+C+LB | 604.0 | 4.0 ± 0.0 | 146.0 | 41.0 |
| | UB+C+LB+SA | 607.0 | 4.0 ± 0.0 | 146.0 | 41.0 |
| | UB+C | 1.1 | 4.2 ± 0.4 | 1754.4 | 87.5 |
| | UB+C+SA | 1.1 | 4.4 ± 0.5 | 1617.4 | 64.7 |
| | UB | 0.7 | 5.0 ± 0.2 | 1239.0 | – |
| | UB+SA | 0.7 | 5.0 ± 0.0 | 1217.0 | – |
| n90p2t2s3 | UB+C+LB | 600.0 | 4.0 ± 0.0 | 136.7 | 28.0 |
| | UB+C+LB+SA | 600.0 | 4.0 ± 0.0 | 137.0 | 28.0 |
| | UB+C | 0.8 | 4.0 ± 0.0 | 176.2 | 31.7 |
| | UB+C+SA | 0.9 | 4.0 ± 0.0 | 156.9 | 27.7 |
| | UB | 0.8 | 5.0 ± 0.0 | 1211.4 | – |
| | UB+SA | 0.8 | 5.0 ± 0.0 | 727.7 | – |
| n90p2t2s4 | UB+C+LB | 600.0 | 4.0 ± 0.0 | 151.7 | 10.0 |
| | UB+C+LB+SA | 603.7 | 4.0 ± 0.0 | 135.3 | 10.0 |
| | UB+C | 0.7 | 4.0 ± 0.0 | 249.0 | 12.6 |
| | UB+C+SA | 0.8 | 4.0 ± 0.0 | 153.3 | 10.0 |
| | UB | 0.8 | 4.8 ± 0.4 | 1198.3 | – |
| | UB+SA | 0.8 | 5.0 ± 0.0 | 781.7 | – |
| n90p2t2s5 | UB+C+LB | 600.0 | 5.0 ± 0.0 | 283.3 | 42.0 |
| | UB+C+LB+SA | 600.0 | 5.0 ± 0.0 | 282.0 | 42.0 |
| | UB+C | 0.9 | 4.0 ± 0.0 | 359.2 | 49.7 |
| | UB+C+SA | 0.9 | 4.0 ± 0.0 | 348.6 | 46.0 |
| | UB | 0.9 | 5.0 ± 0.0 | 688.5 | – |
| | UB+SA | 0.9 | 5.0 ± 0.0 | 489.8 | – |

**Table 5.8:** Preliminary Results: Instances with 90 nodes, density 0.2

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p3t2s1 | UB+C+LB | 600.7 | 6.0 ± 0.0 | 79.0 | 14.0 |
| | UB+C+LB+SA | 601.0 | 6.0 ± 0.0 | 79.0 | 14.0 |
| | UB+C | 1.1 | 5.0 ± 0.0 | 220.4 | 33.9 |
| | UB+C+SA | 1.1 | 5.0 ± 0.0 | 197.8 | 31.0 |
| | UB | 1.0 | 6.0 ± 0.0 | 188.4 | – |
| | UB+SA | 1.0 | 6.0 ± 0.0 | 165.1 | – |
| n90p3t2s2 | UB+C+LB | 609.7 | 6.0 ± 0.0 | 106.3 | 23.0 |
| | UB+C+LB+SA | 600.0 | 6.0 ± 0.0 | 105.0 | 23.0 |
| | UB+C | 1.3 | 5.9 ± 0.3 | 1649.9 | 85.2 |
| | UB+C+SA | 1.2 | 6.0 ± 0.0 | 1192.0 | 69.7 |
| | UB | 1.0 | 6.0 ± 0.0 | 214.9 | – |
| | UB+SA | 1.0 | 6.0 ± 0.0 | 130.8 | – |
| n90p3t2s3 | UB+C+LB | 608.0 | 6.0 ± 0.0 | 31.0 | 6.0 |
| | UB+C+LB+SA | 607.7 | 6.0 ± 0.0 | 31.0 | 6.0 |
| | UB+C | 1.1 | 6.0 ± 0.0 | 675.9 | 21.6 |
| | UB+C+SA | 1.1 | 6.0 ± 0.0 | 387.7 | 14.7 |
| | UB | 1.3 | 6.3 ± 0.4 | 1421.9 | – |
| | UB+SA | 1.3 | 6.0 ± 0.0 | 932.3 | – |
| n90p3t2s4 | UB+C+LB | 606.3 | 6.0 ± 0.0 | 83.0 | 17.0 |
| | UB+C+LB+SA | 610.7 | 6.0 ± 0.0 | 83.0 | 17.0 |
| | UB+C | 1.3 | 5.7 ± 0.5 | 1033.7 | 20.6 |
| | UB+C+SA | 1.1 | 6.0 ± 0.0 | 658.6 | 13.8 |
| | UB | 1.0 | 6.0 ± 0.0 | 320.5 | – |
| | UB+SA | 1.0 | 6.0 ± 0.0 | 274.5 | – |
| n90p3t2s5 | UB+C+LB | 609.0 | 6.0 ± 0.0 | 66.7 | 5.0 |
| | UB+C+LB+SA | 613.0 | 6.0 ± 0.0 | 66.3 | 5.0 |
| | UB+C | 1.2 | 6.0 ± 0.0 | 353.6 | 14.6 |
| | UB+C+SA | 1.2 | 6.0 ± 0.0 | 192.0 | 7.2 |
| | UB | 1.1 | 6.0 ± 0.0 | 363.6 | – |
| | UB+SA | 1.2 | 6.0 ± 0.0 | 317.8 | – |

**Table 5.9:** Preliminary Results: Instances with 90 nodes, density 0.3

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p4t2s1 | UB+C+LB | 602.0 | 7.0 ± 0.0 | 42.3 | 11.3 |
| | UB+C+LB+SA | 606.7 | 7.0 ± 0.0 | 39.0 | 11.0 |
| | UB+C | 1.4 | 7.0 ± 0.0 | 293.5 | 33.6 |
| | UB+C+SA | 1.4 | 7.0 ± 0.0 | 235.5 | 26.4 |
| | UB | 1.4 | 7.1 ± 0.3 | 458.0 | – |
| | UB+SA | 1.2 | 7.4 ± 0.5 | 338.2 | – |
| n90p4t2s2 | UB+C+LB | 624.7 | 8.0 ± 0.0 | 56.0 | 8.0 |
| | UB+C+LB+SA | 623.0 | 8.0 ± 0.0 | 56.0 | 8.0 |
| | UB+C | 1.4 | 7.0 ± 0.2 | 206.3 | 15.7 |
| | UB+C+SA | 1.4 | 7.0 ± 0.0 | 178.2 | 10.9 |
| | UB | 1.2 | 7.0 ± 0.0 | 245.7 | – |
| | UB+SA | 1.2 | 7.0 ± 0.0 | 154.4 | – |
| n90p4t2s3 | UB+C+LB | 602.3 | 8.0 ± 0.0 | 56.7 | 5.0 |
| | UB+C+LB+SA | 608.3 | 8.0 ± 0.0 | 57.0 | 5.0 |
| | UB+C | 1.4 | 7.0 ± 0.0 | 240.8 | 30.7 |
| | UB+C+SA | 1.4 | 7.0 ± 0.0 | 186.8 | 25.0 |
| | UB | 1.3 | 7.4 ± 0.5 | 354.6 | – |
| | UB+SA | 1.3 | 7.4 ± 0.5 | 192.6 | – |
| n90p4t2s4 | UB+C+LB | 604.7 | 7.0 ± 0.0 | 59.0 | 14.0 |
| | UB+C+LB+SA | 604.0 | 7.0 ± 0.0 | 59.0 | 14.0 |
| | UB+C | 1.2 | 7.0 ± 0.0 | 92.4 | 27.4 |
| | UB+C+SA | 1.2 | 7.0 ± 0.0 | 90.2 | 21.1 |
| | UB | 1.2 | 7.9 ± 0.2 | 453.4 | – |
| | UB+SA | 1.2 | 8.0 ± 0.0 | 260.2 | – |
| n90p4t2s5 | UB+C+LB | 613.7 | 9.0 ± 0.0 | 25.0 | 6.0 |
| | UB+C+LB+SA | 601.0 | 9.0 ± 0.0 | 22.0 | 5.0 |
| | UB+C | 1.5 | 7.9 ± 0.3 | 321.7 | 28.0 |
| | UB+C+SA | 1.3 | 8.0 ± 0.0 | 134.3 | 15.5 |
| | UB | 1.6 | 7.0 ± 0.0 | 247.2 | – |
| | UB+SA | 1.6 | 7.0 ± 0.0 | 239.3 | – |

**Table 5.10:** Preliminary Results: Instances with 90 nodes, density 0.4

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p5t2s1 | UB+C+LB | 609.0 | 10.0 ± 0.0 | 23.0 | 8.0 |
| | UB+C+LB+SA | 610.3 | 10.0 ± 0.0 | 23.7 | 8.0 |
| | UB+C | 1.8 | 9.0 ± 0.0 | 165.6 | 36.3 |
| | UB+C+SA | 1.8 | 9.0 ± 0.0 | 145.2 | 30.7 |
| | UB | 1.6 | 9.0 ± 0.0 | 311.8 | – |
| | UB+SA | 1.6 | 9.0 ± 0.0 | 296.0 | – |
| n90p5t2s2 | UB+C+LB | 609.3 | 9.0 ± 0.0 | 27.0 | 6.0 |
| | UB+C+LB+SA | 609.3 | 9.0 ± 0.0 | 27.0 | 6.0 |
| | UB+C | 1.4 | 8.0 ± 0.0 | 83.5 | 14.5 |
| | UB+C+SA | 1.5 | 8.0 ± 0.0 | 75.3 | 11.3 |
| | UB | 1.9 | 9.0 ± 0.3 | 340.1 | – |
| | UB+SA | 1.8 | 9.1 ± 0.4 | 337.0 | – |
| n90p5t2s3 | UB+C+LB | 612.3 | 9.0 ± 0.0 | 39.0 | 0.0 |
| | UB+C+LB+SA | 600.0 | 9.0 ± 0.0 | 39.0 | 0.0 |
| | UB+C | 1.5 | 8.0 ± 0.0 | 170.1 | 10.8 |
| | UB+C+SA | 1.6 | 8.0 ± 0.0 | 161.5 | 8.0 |
| | UB | 1.4 | 8.9 ± 0.2 | 235.0 | – |
| | UB+SA | 1.4 | 9.0 ± 0.0 | 103.5 | – |
| n90p5t2s4 | UB+C+LB | 603.3 | 10.0 ± 0.0 | 47.7 | 14.7 |
| | UB+C+LB+SA | 605.3 | 10.0 ± 0.0 | 47.7 | 14.7 |
| | UB+C | 1.7 | 8.9 ± 0.2 | 352.3 | 36.7 |
| | UB+C+SA | 1.6 | 9.0 ± 0.0 | 223.0 | 28.4 |
| | UB | 1.4 | 9.0 ± 0.0 | 197.9 | – |
| | UB+SA | 1.4 | 9.0 ± 0.0 | 150.4 | – |
| n90p5t2s5 | UB+C+LB | 610.0 | 10.0 ± 0.0 | 25.3 | 6.0 |
| | UB+C+LB+SA | 609.0 | 10.0 ± 1.0 | 25.0 | 6.0 |
| | UB+C | 1.8 | 9.0 ± 0.0 | 222.6 | 16.4 |
| | UB+C+SA | 1.8 | 9.0 ± 0.0 | 178.3 | 14.0 |
| | UB | 1.7 | 9.7 ± 0.5 | 330.7 | – |
| | UB+SA | 1.6 | 9.8 ± 0.4 | 193.5 | – |

**Table 5.11:** Preliminary Results: Instances with 90 nodes, density 0.5

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p6t2s1 | UB+C+LB | 608.3 | 12.0 ± 0.0 | 15.3 | 3.0 |
| | UB+C+LB+SA | 614.3 | 12.0 ± 0.0 | 15.0 | 3.0 |
| | UB+C | 2.0 | 10.0 ± 0.0 | 330.2 | 71.5 |
| | UB+C+SA | 2.1 | 10.0 ± 0.0 | 291.0 | 62.6 |
| | UB | 1.9 | 10.7 ± 0.5 | 382.6 | – |
| | UB+SA | 2.2 | 10.3 ± 0.5 | 292.9 | – |
| n90p6t2s2 | UB+C+LB | 612.7 | 11.0 ± 0.0 | 13.0 | 4.0 |
| | UB+C+LB+SA | 613.0 | 11.0 ± 0.0 | 13.0 | 4.0 |
| | UB+C | 1.6 | 10.0 ± 0.0 | 99.9 | 30.7 |
| | UB+C+SA | 1.6 | 10.0 ± 0.0 | 90.9 | 26.2 |
| | UB | 1.5 | 11.0 ± 0.0 | 99.3 | – |
| | UB+SA | 1.6 | 11.0 ± 0.0 | 64.1 | – |
| n90p6t2s3 | UB+C+LB | 605.7 | 11.0 ± 0.0 | 28.0 | 14.0 |
| | UB+C+LB+SA | 608.0 | 11.7 ± 1.0 | 24.7 | 10.3 |
| | UB+C | 1.6 | 10.2 ± 0.4 | 274.8 | 32.7 |
| | UB+C+SA | 1.6 | 10.2 ± 0.4 | 155.4 | 19.2 |
| | UB | 1.8 | 10.9 ± 0.6 | 643.4 | – |
| | UB+SA | 1.5 | 11.6 ± 0.7 | 465.0 | – |
| n90p6t2s4 | UB+C+LB | 606.0 | 12.0 ± 0.0 | 34.3 | 7.0 |
| | UB+C+LB+SA | 601.0 | 12.0 ± 0.0 | 34.7 | 7.0 |
| | UB+C | 2.3 | 10.0 ± 0.0 | 219.2 | 39.1 |
| | UB+C+SA | 2.2 | 10.0 ± 0.0 | 197.0 | 34.6 |
| | UB | 1.6 | 11.0 ± 0.2 | 136.0 | – |
| | UB+SA | 1.6 | 11.0 ± 0.0 | 128.4 | – |
| n90p6t2s5 | UB+C+LB | 613.3 | 12.0 ± 0.0 | 18.0 | 5.0 |
| | UB+C+LB+SA | 616.0 | 12.0 ± 0.0 | 18.0 | 5.0 |
| | UB+C | 2.0 | 10.5 ± 0.6 | 378.3 | 50.7 |
| | UB+C+SA | 2.0 | 10.6 ± 0.5 | 263.9 | 31.8 |
| | UB | 1.7 | 10.9 ± 0.3 | 289.1 | – |
| | UB+SA | 1.8 | 10.8 ± 0.4 | 206.1 | – |

**Table 5.12:** Preliminary Results: Instances with 90 nodes, density 0.6

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p7t2s1 | UB+C+LB | 611.0 | 13.0 ± 0.0 | 30.0 | 8.0 |
| | UB+C+LB+SA | 613.3 | 13.0 ± 0.0 | 30.0 | 8.0 |
| | UB+C | 2.0 | 12.7 ± 0.4 | 250.8 | 23.0 |
| | UB+C+SA | 2.0 | 12.8 ± 0.4 | 163.9 | 16.0 |
| | UB | 2.3 | 12.4 ± 0.5 | 270.1 | – |
| | UB+SA | 1.7 | 13.0 ± 0.0 | 164.8 | – |
| n90p7t2s2 | UB+C+LB | 612.3 | 13.0 ± 0.0 | 10.3 | 1.0 |
| | UB+C+LB+SA | 605.3 | 13.0 ± 0.0 | 10.0 | 1.0 |
| | UB+C | 1.8 | 12.9 ± 0.2 | 203.3 | 15.7 |
| | UB+C+SA | 1.7 | 13.0 ± 0.0 | 101.4 | 8.1 |
| | UB | 2.4 | 12.9 ± 0.5 | 399.2 | – |
| | UB+SA | 1.8 | 13.9 ± 0.2 | 284.6 | – |
| n90p7t2s3 | UB+C+LB | 613.0 | 13.0 ± 0.0 | 17.0 | 7.0 |
| | UB+C+LB+SA | 613.7 | 13.0 ± 0.0 | 17.0 | 7.0 |
| | UB+C | 2.1 | 12.7 ± 0.5 | 114.8 | 20.6 |
| | UB+C+SA | 2.0 | 12.9 ± 0.3 | 68.4 | 11.6 |
| | UB | 1.7 | 12.0 ± 0.0 | 92.0 | – |
| | UB+SA | 1.7 | 12.0 ± 0.0 | 72.0 | – |
| n90p7t2s4 | UB+C+LB | 601.7 | 13.0 ± 0.0 | 19.3 | 8.0 |
| | UB+C+LB+SA | 603.7 | 13.0 ± 0.0 | 19.3 | 8.0 |
| | UB+C | 2.2 | 11.9 ± 0.2 | 195.8 | 31.2 |
| | UB+C+SA | 2.1 | 12.0 ± 0.0 | 122.6 | 23.5 |
| | UB | 1.7 | 13.0 ± 0.0 | 77.4 | – |
| | UB+SA | 1.7 | 13.0 ± 0.0 | 52.8 | – |
| n90p7t2s5 | UB+C+LB | 611.0 | 13.0 ± 0.0 | 18.0 | 7.0 |
| | UB+C+LB+SA | 606.7 | 13.0 ± 0.0 | 17.3 | 6.3 |
| | UB+C | 1.7 | 13.0 ± 0.0 | 221.2 | 37.9 |
| | UB+C+SA | 1.8 | 13.0 ± 0.0 | 138.8 | 27.2 |
| | UB | 1.9 | 13.0 ± 0.0 | 103.7 | – |
| | UB+SA | 1.9 | 13.0 ± 0.0 | 88.2 | – |

**Table 5.13:** Preliminary Results: Instances with 90 nodes, density 0.7

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p8t2s1 | UB+C+LB | 615.3 | 15.3 ± 0.5 | 11.3 | 5.0 |
| | UB+C+LB+SA | 616.7 | 15.0 ± 0.0 | 11.3 | 5.0 |
| | UB+C | 1.5 | 15.0 ± 0.0 | 47.0 | 9.9 |
| | UB+C+SA | 1.5 | 15.0 ± 0.0 | 45.3 | 7.0 |
| | UB | 1.8 | 15.4 ± 0.5 | 59.0 | – |
| | UB+SA | 1.8 | 15.4 ± 0.5 | 44.8 | – |
| n90p8t2s2 | UB+C+LB | 602.3 | 15.0 ± 0.0 | 30.0 | 16.0 |
| | UB+C+LB+SA | 605.3 | 15.0 ± 0.0 | 30.0 | 16.0 |
| | UB+C | 1.9 | 15.0 ± 0.0 | 65.7 | 23.2 |
| | UB+C+SA | 1.9 | 15.0 ± 0.0 | 53.5 | 17.2 |
| | UB | 2.2 | 14.8 ± 0.4 | 140.4 | – |
| | UB+SA | 2.0 | 15.0 ± 0.0 | 117.0 | – |
| n90p8t2s3 | UB+C+LB | 613.0 | 16.0 ± 0.0 | 60.0 | 17.0 |
| | UB+C+LB+SA | 609.7 | 16.0 ± 0.0 | 60.3 | 17.0 |
| | UB+C | 2.2 | 15.0 ± 0.0 | 134.2 | 32.9 |
| | UB+C+SA | 2.1 | 15.0 ± 0.0 | 104.4 | 25.2 |
| | UB | 2.2 | 16.3 ± 0.4 | 329.6 | – |
| | UB+SA | 2.0 | 16.5 ± 0.5 | 205.2 | – |
| n90p8t2s4 | UB+C+LB | 609.0 | 15.0 ± 0.0 | 67.3 | 14.0 |
| | UB+C+LB+SA | 614.3 | 15.0 ± 0.0 | 68.0 | 14.0 |
| | UB+C | 2.3 | 15.0 ± 0.0 | 263.1 | 31.4 |
| | UB+C+SA | 2.4 | 15.0 ± 0.0 | 175.5 | 23.1 |
| | UB | 2.8 | 14.7 ± 0.7 | 355.4 | – |
| | UB+SA | 2.3 | 15.6 ± 0.5 | 297.0 | – |
| n90p8t2s5 | UB+C+LB | 610.0 | 15.0 ± 0.0 | 16.0 | 5.0 |
| | UB+C+LB+SA | 608.7 | 15.0 ± 0.0 | 16.0 | 5.0 |
| | UB+C | 2.4 | 15.1 ± 0.8 | 306.8 | 52.2 |
| | UB+C+SA | 1.8 | 16.0 ± 0.0 | 173.9 | 32.1 |
| | UB | 2.0 | 15.7 ± 0.5 | 119.1 | – |
| | UB+SA | 1.8 | 16.0 ± 0.0 | 83.3 | – |

**Table 5.14:** Preliminary Results: Instances with 90 nodes, density 0.8

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n90p9t2s1 | UB+C+LB | 600.0 | 20.0 ± 0.0 | 170.3 | 53.7 |
| | UB+C+LB+SA | 600.0 | 20.0 ± 0.5 | 168.0 | 168.0 |
| | UB+C | 2.3 | 18.0 ± 0.0 | 263.4 | 64.7 |
| | UB+C+SA | 2.3 | 18.0 ± 0.0 | 232.9 | 59.9 |
| | UB | 2.1 | 19.0 ± 0.2 | 108.3 | – |
| | UB+SA | 2.0 | 19.0 ± 0.0 | 89.8 | – |
| n90p9t2s2 | UB+C+LB | 600.0 | 19.0 ± 0.0 | 82.0 | 29.0 |
| | UB+C+LB+SA | 600.0 | 19.0 ± 0.0 | 82.0 | 29.0 |
| | UB+C | 2.6 | 19.0 ± 0.2 | 244.0 | 41.2 |
| | UB+C+SA | 2.4 | 19.0 ± 0.0 | 146.3 | 26.7 |
| | UB | 2.0 | 19.0 ± 0.0 | 42.4 | – |
| | UB+SA | 1.9 | 19.0 ± 0.0 | 27.3 | – |
| n90p9t2s3 | UB+C+LB | 607.3 | 19.0 ± 0.0 | 122.0 | 53.0 |
| | UB+C+LB+SA | 601.0 | 19.0 ± 0.0 | 122.0 | 53.0 |
| | UB+C | 2.7 | 19.2 ± 0.6 | 388.0 | 160.3 |
| | UB+C+SA | 2.2 | 19.7 ± 0.7 | 192.1 | 84.8 |
| | UB | 2.2 | 19.9 ± 0.3 | 163.0 | – |
| | UB+SA | 2.1 | 20.0 ± 0.0 | 107.4 | – |
| n90p9t2s4 | UB+C+LB | 600.0 | 18.0 ± 0.0 | 148.7 | 78.0 |
| | UB+C+LB+SA | 600.0 | 18.0 ± 0.0 | 148.7 | 78.0 |
| | UB+C | 2.7 | 17.8 ± 0.4 | 222.9 | 114.3 |
| | UB+C+SA | 2.3 | 18.0 ± 0.0 | 175.0 | 96.9 |
| | UB | 2.1 | 18.0 ± 0.0 | 50.1 | – |
| | UB+SA | 2.0 | 18.0 ± 0.0 | 41.7 | – |
| n90p9t2s5 | UB+C+LB | 616.7 | 20.0 ± 0.0 | 49.0 | 28.0 |
| | UB+C+LB+SA | 619.3 | 20.0 ± 0.0 | 49.3 | 28.0 |
| | UB+C | 2.4 | 18.0 ± 0.0 | 114.9 | 51.8 |
| | UB+C+SA | 2.2 | 18.0 ± 0.0 | 100.3 | 43.9 |
| | UB | 2.2 | 18.9 ± 0.3 | 143.9 | – |
| | UB+SA | 2.1 | 19.0 ± 0.0 | 99.3 | – |

**Table 5.15:** Preliminary Results: Instances with 90 nodes, density 0.9

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| dsjc500.5-1 | Greedy Coloring | 1.0 | 73 | 0 | 0 |
| dsjc500.5-2 | UB+C+LB | 618.3 | 70.0 ± 0.0 | 6.3 | 1.0 |
| | UB+C+LB+SA | 618.0 | 70.0 ± 0.0 | 7.0 | 1.0 |
| | UB+C | 600.0 | 66.0 ± 0.0 | 231.2 | 87.0 |
| | UB+C+SA | 600.0 | 66.0 ± 0.0 | 214.0 | 81.0 |
| | UB | 528.0 | 66.0 ± 0.0 | 169.2 | – |
| | UB+SA | 533.0 | 66.0 ± 0.0 | 118.2 | – |
| dsjc500.5-3 | UB+C+LB | 613.7 | 68.0 ± 0.0 | 2.7 | 0.0 |
| | UB+C+LB+SA | 609.7 | 68.0 ± 0.0 | 3.0 | 0.0 |
| | UB+C | 600.0 | 65.0 ± 0.0 | 220.4 | 81.8 |
| | UB+C+SA | 600.0 | 65.0 ± 0.0 | 220.7 | 81.8 |
| | UB | 600.0 | 65.9 ± 0.3 | 192.4 | – |
| | UB+SA | 600.0 | 66.0 ± 0.0 | 149.1 | – |
| dsjc500.5-4 | UB+C+LB | 608.0 | 70.0 ± 0.0 | 7.0 | 0.0 |
| | UB+C+LB+SA | 614.0 | 70.0 ± 0.0 | 6.7 | 0.0 |
| | UB+C | 600.0 | 65.0 ± 0.0 | 305.7 | 155.3 |
| | UB+C+SA | 600.0 | 65.0 ± 0.0 | 306.0 | 155.2 |
| | UB | 600.0 | 66.2 ± 0.9 | 286.5 | – |
| | UB+SA | 600.0 | 66.4 ± 0.7 | 264.6 | – |

**Table 5.16:** Preliminary Results: Instances from Noronha's paper

## 5.2 Discussion and Final Results

Since the results of Frota's and Noronha's algorithms are better than the results of my algorithm most of the time, I made investigations for what might be the reason. The first guess was that perhaps the greedy coloring is not optimal. Alas, for test instances with 20, 40 and 60 nodes the upper bound the greedy coloring computed matched the exact chromatic number of the solution which the algorithm considered the best one (see table 5.17). For instances with more nodes it was not possible to make this comparison since the exact computation took too long. But if we take a look at dsjc500.5-1, we see that Noronha's algorithm performed better than my variant of greedy coloring. This leads to the conclusion that greedy coloring is probably good for instances with a low number of nodes, but the more nodes are used, the worse it performs. This opens a possibility for further research, trying the same VNS with a different heuristic. It is also possible that even with instances with a rather small number of nodes, greedy coloring leads to a suboptimal result since it may select a solution although another solution having the same upper bound would have a lower exact chromatic number.

What was also tried was a modification of the parameters: instead of five consecutive failing VND runs (not leading to an improvement), twenty were allowed (table 5.18). The time limit was accordingly increased to 1800 seconds, but it was only reached in Noronha's instances. However, the results are not all too different. This has brought me to the conclusion that a further increase of the number of consecutive failing VND runs will probably not lead to any (significant) improvement; five consecutive failing VND runs seem to suffice.

| Instance | Exact | UB+C |
|---|---|---|
| n20p5t2s1 | 3 | 3 |
| n40p5t2s1 | 5 | 5 |
| n60p5t2s1 | 6 | 6 |

**Table 5.17:** Comparison of exact results and upper bounds

| Instance | Method | t (s) | Result (avg ± sd) | UB same | UB+C same |
|---|---|---|---|---|---|
| n20p5t2s1 | UB+C | 0.0 | 3.0 ± 0.0 | 62.3 | 15.6 |
| n40p5t2s1 | UB+C | 0.2 | 4.6 ± 0.5 | 239.0 | 15.6 |
| n60p5t2s1 | UB+C | 0.0 | 6.0 ± 0.0 | 103.9 | 18.4 |
| n80p5t2s1 | UB+C | 2.6 | 8.0 ± 0.0 | 453.5 | 36.7 |
| n100p5t2s1 | UB+C | 6.6 | 9.0 ± 0.0 | 389.4 | 44.0 |
| n120p5t2s1 | UB+C | 16.0 | 10.8 ± 0.4 | 488.3 | 48.6 |
| dsjc500.5-2 | UB+C | 1800.0 | 66.0 ± 0.0 | 215.0 | 81.0 |
| dsjc500.5-3 | UB+C | 1800.0 | 65.0 ± 0.0 | 369.0 | 156.0 |
| dsjc500.5-4 | UB+C | 1800.0 | 64.0 ± 0.0 | 813.1 | 409.7 |

**Table 5.18:** Preliminary Results: Modified parameters (up to 20 failing VND runs allowed, time limit 1800 seconds)

| Instance(s) | Min. | Max. | My best result (avg ± sd) | t (s) of my result |
|---|---|---|---|---|
| n20p5t2s* | 3 | 3 | 3.0 ± 0.0 | 0.0 |
| n40p5t2s* | 4 | 4 | 4.7 ± 0.5 | 0.0 |
| n60p5t2s* | 5 | 5 | 6.0 ± 0.0 | 0.3 |
| n80p5t2s* | 6 | 6 | 7.0 ± 0.0 | 1.0 |
| n100p5t2s* | 6 | 7 | 9.0 ± 0.0 | 2.2 |
| n120p5t2s* | 7 | 8 | 10.8 ± 0.4 | 5.9 |
| dsjc500.5-1 | 51 | 55 | 73.0 ± 0.0 | 1.0 |
| dsjc500.5-2 | 45 | 49 | 66.0 ± 0.0 | 528.0 |
| dsjc500.5-3 | 43 | 46 | 65.0 ± 0.0 | 600.0 |
| dsjc500.5-4 | 42 | 44 | 65.0 ± 0.0 | 600.0 |

**Table 5.19:** Final Results including data from Frota's and Noronha's papers; for Frota's instances, these are aggregated results

# Chapter 6

# Conclusions

Within this work I developed and implemented diverse variants of an algorithm that solves the selective graph coloring problem. The thesis first defines the problem, introduces a practical application and discusses the complexity of the chosen approach. After a literature survey on both the SGCP and graph coloring in general, the algorithmic approach is discussed in detail. This discussion starts with an introduction to metaheuristics, especially the ones used in this algorithm (variable neighborhood descent and variable neighborhood search). Then the solution representation is described, followed by different ways of evaluating the solution. Both heuristic approaches (upper bound, number of conflicts, lower bound) and an exact method (integer linear programming) are presented. Moreover, the chosen approach for a solution archive is discussed. After an outline of the complete algorithm, the testing environment is described, and then the results of the tests performed on the chosen instances are listed.

The results have in general been not as good as expected: The performance of the algorithm is not quite comparable to the solutions of other researchers. This may be due to the greedy coloring heuristic used, so further research should focus on trying other heuristics. At least the implementation has shown that computing a lower bound of the chromatic number in addition to the upper bound (where the upper bound is the primary evaluation criterion, in case of identical upper bounds the number of conflicts is considered, and if these are still the same the solution with the smaller lower bound is preferred) is time-consuming and does not lead to a significant improvement, if at all. Solution archives have not proven effective either. So the method using upper bound and number of conflicts as evaluation criteria seems to be the best strategy.

# Bibliography


[1] R. Brooks and W. Tutte. On colouring the nodes of a network. *Proceedings of the Cambridge Philosophical Society*, volume 37, issue 2, pages 194-197, 1941.

[2] J. Brown: Chromatic scheduling and the chromatic number problem. *Management Science*, volume 19, issue 4, pages 456-463, 1972.

[3] R. Karp: Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85-103, 1972.

[4] K. Appel and W. Haken: Every Planar Map is Four Colorable. *Illinois Journal of Mathematics*, volume 21, pages 429-567, 1977.

[5] F. Leighton: A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau of Standards*, volume 84, issue 6, pages 489-506, 1979.

[6] J. Peemöller: A correction to Brelaz's modification of Brown's coloring algorithm. *Communications of the ACM*, volume 26, issue 8, pages 593-597, 1983.

[7] M. Kubale and B. Jackowski: A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, volume 28, issue 4, pages 412-418, 1985.

[8] A. Hertz, D. de Werra: Using Tabu Search Techniques for Graph Coloring. *Computing*, volume 39, pages 345-351, 1987.

[9] P. Raghavan and E. Upfal: Efficient routing in all-optical networks. *Proc 26th ACM Symp. on Theory of Computing*, pages 134-143, New York, 1994.

[10] A. Mehrotra and M. Trick: A Column Generation Approach For Graph Coloring. *INFORMS Journal on Computing*, volume 8, pages 344-354, 1995.

[11] D. Wolpert and W. Macready: No Free Lunch Theorems for Search. *Technical Report SFI-TR-95-02-010 (Santa Fe Institute)*, 1995.

[12] C. Morgenstern: Distributed Coloration Neighborhood Search. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 335-357, 1996.

[13] R. Ramaswami and K. Sivarajan: Routing and wavelength assignment in all-optical networks. *IEEE/ACM Trans. Networking*, volume 3, issue 5, pages 489-500, 1995.

[14] C. Fleurent and J. Ferland: Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, volume 63, issue 3, pages 437-461, 1996.

[15] B. Mukherjee: Some principles for designing a wide-area WDM optical network. *IEEE/ACM Trans. Networking*, volume 4, issue 5, pages 684-696, 1996.

[16] N. Mladenovic and P. Hansen: Variable neighborhood search. *Computers and Operations Research*, volume 24, issue 11, pages 1097-1100, 1997.

[17] E. Harder: Routing and wavelength assignment in all-optical WDM wavelength-routed networks, PhD thesis, George Washington University, 1998.

[18] R. Ramaswami and K. Sivarajan: Optical communication networks, Morgan-Kaufmann, 1998.

[19] P. Galinier and J. Hao: Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatoral Optimization*, volume 3, issue 4, pages 379-397, 1999.

[20] B. Narahari et al.: Routing and Scheduling I/O Transfers on Wormhole-Routed Mesh Networks. *J. Parallel and Distributed Computing*, volume 57, issue 1, pages 1-13, 1999.

[21] J. Brimberg et al.: Improvements and comparison of heuristics for solving the multisource Weber problem. *Operational Research*, issue 48, pages 444-460, 2000.

[22] G. Li and R. Simha: The Partition Coloring Problem and its Application to Wavelength Routing and Assignment. *Proceedings of the First Workshop on Optical Networks*, 2000.

[23] J. Choi et al.: A functional classification of routing and wavelength assignment schemes in DWDM networks: Static case. *Proceedings of the 7th International Conference on Optical Communication and Networks*, pages 1109-1115, Paris, 2000.

[24] G. Filho et al.: Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring, paper published on the Internet, 2000.

[25] T. English: Optimization Is Easy and Learning Is Hard in the Typical Function. *Proceedings of the 2000 Congress on Evolutionary Computation: CEC00*, pages 924-931, 2000.

[26] F. Herrmann and A. Hertz: Finding the chromatic number by means of critical graphs. *ACM Journal of Experimental Algorithmics*, volume 7, issue 10, pages 1-9, 2002.

[27] C. Avanthay, A. Hertz and N. Zufferey: A Variable Neighborhood Search for Graph Coloring. *European Journal of Operational Research*, volume 151, pages 379-388, 2003.

[28] R. Correa, M. Campelo and Y. Frota: Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, volume 89, pages 159-164, 2004.

[29] M. Campelo, V. Campos and R. Correa: On the asymmetric representatives formulation for the vertex coloring problem, *Proceedings of the 2th Brazilian Symposium on Graphs, Algorithms and Combinatorics*, volume 19 of Electronic Notes in Discrete Mathematics, pages 337-343, Angra dos Reis, 2005.

[30] C. Desrosiers, P. Galinier and A. Hertz: Efficient algorithms for finding critical subgraphs. *Discrete Applied mathematics*, volume 156, issue 2, pages 244-266, 2005.

[31] A. Hertz, P. Galinier and N. Zufferey: An Adaptive Memory Algorithm for the Graph Coloring Problem. *Discrete Applied Mathematics*, volume 156, issue 2, pages 267-279, 2005.

[32] E. Malaguti, M. Monaci and P. Thot: A Metaheuristic Approach for the Vertex Coloring Problem. *Technical Report OR/05/3*, University of Bologna, Italy, 2005.

[33] I. Diaz and P. Zabala: A Branch-and-Cut Algorithm for Graph Coloring. *Discrete Applied Mathematics*, volume 154, number 5, pages 826-847, 2006.

[34] T. Noronha and C. Ribeiro: Routing and wavelength assignment by partition colouring. *European Journal of Operational Research*, volume 171, issue 3, pages 797-810, 2006.

[35] H. de Fraysseix, P. Ossona de Mendez and P. Rosenstiehl: Tremaux Trees and Planarity. *European Journal of Combinatorics*, volume 33, issue 3, pages 2798-293, 2012.

[36] I. Blöchliger and N. Zufferey: A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research Volume 35*, issue 3, pages 960-975, 2008.

[37] A. Hertz et al.: Variable space search for graph coloring. *Discrete Applied Mathematics*, volume 156, issue 13, pages 2551-1560, 2008.

[38] S. Luke: Essentials of Metaheuristics. Lulu, 2009, available at http://cs.gmu.edu/∼sean/book/metaheuristics/

[39] Y. Frota et al.: A branch-and-cut algorithm for partition coloring. *Networks*, volume 55, issue 3, pages 194-204, 2010.

[40] Y. Frota et al.: Instances for the partition Coloring Problem, www.ic.uff.br/∼celso/grupo/pcp.htm

[41] Z. Lü and J. Hao: A memetic algorithm for graph coloring. *European Journal of Operational Research*, volume 203, issue 1, pages 241-250, 2010.

[42] S. Held et al.: Safe Lower Bounds For Graph Coloring, *IPCO 2011*, pages 261-273, 2011.

[43] M. Chiarandini, G. Galbiati and S. Gualandi: Efficieny issues in the RLF heuristic for graph coloring. *MIC 2011 (The IX Metaheuristics International Conference)*, pages 461-469, 2011.

[44] S. Gualandi and F. Malucelli: Exact Solution of Graph Coloring Problems via Constraint Programming and Column Generation. *INFORMS Journal on Computing*, volume 24, number 1, pages 81-100, 2012.

[45] B. Hu and G. Raidl: An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of LNCS, pages 287-294, Springer, 2012.

# List of Algorithms

# List of Tables