The following page is the Signature Page. The Signature Page needs to be given to the Grad School, but it should not be bound with the thesis.

Don't bind this page either!

We approve the thesis of ThanhVu H. Nguyen.

Date of Signature

Thang Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Sukmoon Chang
Assistant Professor of Computer Science

Qin Ding
Assistant Professor of Computer Science

Linda Null
Assistant Professor of Computer Science
Graduate Coordinator

The Pennsylvania State University
The Graduate School

ON THE GRAPH COLORING PROBLEM AND ITS GENERALIZATIONS

A Thesis in
Computer Science
by
ThanhVu H. Nguyen

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2006

ThanhVu H. Nguyen

The thesis of ThanhVu H. Nguyen was reviewed and approved* by the following:

Thang Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Sukmoon Chang
Assistant Professor of Computer Science

Qin Ding
Assistant Professor of Computer Science

Linda Null
Assistant Professor of Computer Science
Graduate Coordinator

*Signatures are on file in the Graduate School.

Abstract

This thesis presents an agent-based algorithm for the $\mathcal{NP}$-hard Graph Coloring problem and its generalizations, namely the Bandwidth Coloring, Multi Coloring, and Bandwidth Multi Coloring problems.

The main feature of the algorithm is the collaboration of agents to color the vertices of the graph while making decisions based only on local neighborhood information. In addition, the algorithm provides a single framework for solving all four problems by using a preprocessing method to transform different graph coloring problems into a common type. Other ideas such as tabu lists and greedy-based local optimization are also adopted to accelerate the convergence rate and improve solution quality. Experimental results show that this algorithm performs very well compared to other existing approaches.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The foremost gratefulness is owed to my advisor, Dr. Thang Bui. His guidance and expertise provided me the much-needed encouragement and motivation throughout this arduous, exhilarating, and fruitful research. I am convinced that his lucid lectures are exceptional gifts to those willing to learn and regret not being able to continue my Ph.D. study under his supervision.

I am deeply indebted for the patience and generosity in the transmission of knowledge of all Computer Science professors with whom I have taken classes: Dr. Sukmoon Chang, Dr. Qin Ding, Dr. Pavel Naumov, and Dr. Linda Null. My thanks likewise go to all the thesis committee members for not only reviewing this thesis, but also enduring its defense. I am also much obliged to Dr. Marco Chiarandini for providing valuable technical discussions on the thesis topic and Mr. Faisal Zaman for his constructive criticism on the report.

On a more personal basis, I wish to express an immense appreciation to Dr. James Smith, III, my co-op supervisor at the Naval Research Laboratory, for showing me the prestigious value in researching. Undeniably, the time I worked with him at the beginning of my research career has influenced and inspired all of my activities thereafter.

My warmest gratitude goes to my parents for their ultimate, continuous, unfailing love and support over the years. This thesis is dedicated to them.

Chapter 1

Introduction

The Graph Coloring problem (GCP) is the classic $\mathcal{NP}$-hard problem of coloring the vertices of an undirected graph with as few colors as possible, such that no two adjacent vertices have the same color. Its corresponding decision problem, "*Given an undirected graph G = (V,E) and a number k, is there a coloring of G which uses at most k colors?*", is $\mathcal{NP}$-complete, and in fact was one of the 21 $\mathcal{NP}$-complete problems in Richard Karp's landmark paper [21].

The Bandwidth Coloring, Multi Coloring, and Bandwidth Multi Coloring problems are well-known generalizations of GCP. These generalizations share the same objective with the original problem in minimizing the number of colors used; however, they have more constraints placed on either the vertices, edges, or both. Like GCP, all of these generalizations are $\mathcal{NP}$-hard.

Graph colorings are ubiquitous in the modeling of many real-world applications. A wide range of practical problems such as machine assignment problems in production scheduling, register allocation problems in operating systems, and frequency assignment problems in telecommunications can be represented with graph coloring models.

Since graph coloring problems are $\mathcal{NP}$-hard, exact polynomial time solvers for them are not expected to exist, unless $\mathcal{P} = \mathcal{NP}$. Likewise, existing approximation algorithms for GCP in polynomial time are impractical as the margin of error can grow quite high. Not much is known about the approximation complexities for the generalizations. Nevertheless, the problems inspire an ample collection of heuristic algorithms such as local search, tabu techniques, squeaky wheel optimizations,

genetic algorithms, and ant-based algorithms.

Due to distinctive constraints in different graph coloring problems, most existing approaches are biased to a specific problem type. This is not the case with our agent-based algorithm, which proposes a generalized framework for the original problem as well as its variations. In addition to agents, the algorithm also uses a variety of techniques such as local optimization and tabu lists. Extensive experimental results show that our approach is very competitive with other algorithms in both running time and solution quality.

The rest of this thesis is organized as follows. Chapter 2 describes GCP and its generalizations, their applications, complexities, and existing approaches for solving these problems. Chapter 3 discusses our agent-based algorithm in detail. Chapter 4 presents and compares the experimental data to results from other algorithms. Finally, Chapter 5 concludes the thesis and suggests future research directions.

Portions of this thesis also appeared in [7] and [9].

Chapter 2

The Graph Coloring Problem and Its Generalizations

2.1 Problem Descriptions

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. Let $k \in \mathbb{Z}^+$; we denote by $[k]$ the set $\{1, \ldots, k\}$. Let $\mathcal{P}([k])$ refer to the power set of $[k]$, i.e., the set of all subsets of $\{1, \ldots, k\}$.

A *coloring* of G is an assignment of a set of colors to each vertex in G. More formally, a *k-coloring* of G is a mapping $f\colon V \longmapsto \mathcal{P}([k])$, for $k \in \mathbb{Z}^+$. The term *conflict* at a vertex v indicates the number of vertices adjacent to v having colors that are inconsistent with the coloring of v, where the definition of inconsistency depends on the specific coloring problem. A k-coloring is proper or *valid* if no conflict exists among the vertices in the graph.

In the following sections, we formalize the graph coloring problem and its generalizations.

2.1.1 The Graph Coloring Problem (GCP)

GCP is the problem of finding an assignment of colors to the vertices of a graph, using a minimum k number of colors, such that each vertex has a color, and no two adjacent vertices have the same color. For a given graph G, the minimum k such that G has a proper coloring is called the *chromatic* number of G and is denoted

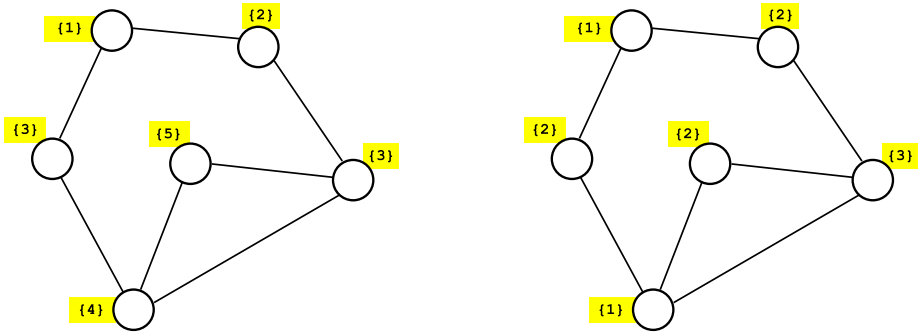
by $\chi(G)$ or simply χ.

Input: An undirected graph $G = (V, E)$.

Output: A minimum k and a mapping $f : V \longmapsto \mathcal{P}([k])$ such that

- $\forall u \in V$, $|f(u)| = 1$, and
- $\forall (u, v) \in E$, $f(u) \neq f(v)$.

In what follows, for simplicity, whenever $|f(u)| = 1$, e.g., $f(u) = \{c\}$, we refer to the color of u as c instead of the singleton set $\{c\}$.



(a) A 5-coloring (b) A 3-coloring

Figure 2.1: GCP example.

Figure 2.1 shows two examples for GCP. The set of colors assigned to a vertex is shown in its associated rectangular box. Notice that in GCP, each vertex has a singleton set of colors (i.e., the cardinality of the set is 1). The example in Figure 2.1a gives a 5-coloring since the highest color value used in this coloring is 5. Figure 2.1b presents an optimal coloring that uses 3 colors.

2.1.2 The Bandwidth Coloring Problem (BCP)

This generalization is similar to GCP, except that each edge in the input graph has a positive integer weight, and the coloring must satisfy an extra constraint. More precisely, the difference between the colors of the two end points of an edge must be at least the weight of that edge.

Input: An undirected graph $G = (V, E)$ with positive integer edge weight $d(u, v)$, $\forall (u, v) \in E$.

Output: A minimum k and a mapping $f\colon V \longmapsto \mathcal{P}([k])$ such that

- $\forall u \in V$, $|f(u)| = 1$, and
- $\forall (u,v) \in E$, $\forall a \in f(u)$, $\forall b \in f(v)$, $|a-b| \geq d(u,v)$.

Note that BCP is reduced to GCP if $d(u,v) = 1, \forall (u,v) \in E$.

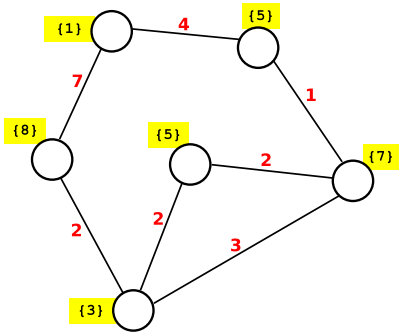


Figure 2.2: BCP example.

Figure 2.2 displays an example for BCP with an 8-coloring using the same format as in Figure 2.1. The numbers on the edges represent the edge weights. Notice that not all values in the set $\{1,\ldots,8\}$ are used to color the graph.

2.1.3 The Multi Coloring Problem (MCP)

This is another generalization of GCP, where each vertex in the input graph has a positive integer weight. The goal is to find a coloring of a vertex set, using as few colors as possible, such that each vertex is colored with not just one color, but with a set of as many colors as the weight of that vertex. Furthermore, for any edge in the graph, the color sets of the two end points of that edge must be disjointed.

Input: An undirected graph $G = (V,E)$ with positive weight $w(u)$, $\forall u \in V$.

Output: A minimum k and a mapping $f\colon V \longmapsto \mathcal{P}([k])$ such that

- $\forall u \in V$, $|f(u)| = w(u)$, and
- $\forall (u,v) \in E$, $f(u) \cap f(v) = \emptyset$.

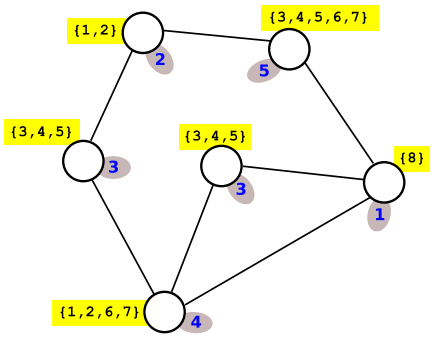


Figure 2.3: MCP example.

Note that MCP is reduced to GCP if $w(u) = 1, \forall u \in V$.

Figure 2.3 shows an example for MCP using an 8-coloring. The weight of the vertex is shown in the associated oval area of that vertex. Notice that each vertex acquires a set of as many colors as the weight of that vertex.

2.1.4 The Bandwidth Multi Coloring Problem (BMCP)

This graph coloring generalization has the constraints of both BCP and MCP.

Input: An undirected graph $G = (V, E)$ with positive integer node weight $w(u), \forall u \in V$, and positive integer edge weight $d(u, v), \forall (u, v) \in E$.

Output: A minimum k and a mapping $f : V \longmapsto \mathcal{P}([k])$ such that

- $\forall u \in V, |f(u)| = w(u)$, and
- $\forall (u, v) \in E, \forall a \in f(u), \forall b \in f(v), |a - b| \geq d(u, v)$.

As the edge weights are positive, the second condition implies that $\forall (u, v) \in E, f(u) \cap f(v) = \emptyset$. It should be noted that input graphs for BMCP may contain self-loops. For example, $d(u, u) = 3$ means the colors assigned to vertex u must have at least a difference of value 3 between any two of them. It is clear that $\forall (u, v) \in E$ and $\forall u \in V$, BMCP is reduced to MCP if $d(u, v) = 1$, to BCP if $w(u) = 1$, and to GCP if $d(u, v) = 1$ and $w(u) = 1$.

Figure 2.4 gives an example for BMCP that uses a 13-coloring. Notice the self-loop requires each color in the color set of the associated vertex differs from each other by at least 2, the weight of the self-loop edge.

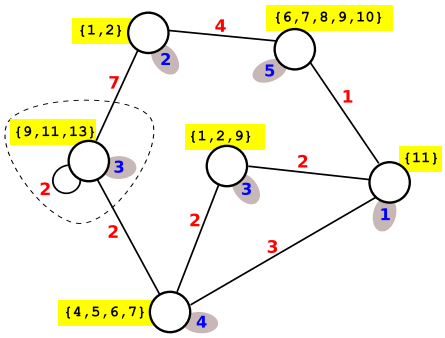


Figure 2.4: BMCP example.

2.2 Applications

Early versions of graph coloring problems date back to 1852, when a college student named Francis Guthrie noticed that all counties in England can be colored using only four colors in such a way that two counties having a common border are never colored with the same color. It was finally proven in 1977 that at most *four* colors are necessary in such planar graphs [2, 3]. Figure 2.5 shows a coloring of the United States map. More practically, the original GCP has served as a paradigm for numerous applications such as timetabling [30] or scheduling problems [22], computer register allocation [1], printed circuit boards testing [15], air traffic flow control [4], and wavelength assignment in optical network design [32].

Figure 2.5: A 4-coloring on the map of the United States.

Suppose that a school needs to schedule a set of classes for a semester. There

are classes that cannot be scheduled concurrently, possibly because a room cannot be occupied by two classes at once, or an instructor cannot be in two places at the same time. The objective is to assign a time period to each class such that the total number of periods used is minimal, since a daily school schedule has a limited number of periods (e.g., six periods in high school). GCP can be used to model this problem as follows. A vertex is associated with each class, an edge joins two vertices if their two associated classes cannot be held at the same time, and the time period assigned to each class is denoted as the color of the associated vertex. Thus, by minimizing the number of colors used in this graph, the school obtains a class schedule with a minimal number of time periods.

More restricted applications can employ BCP, MCP, or BMCP since these generalizations extend the constraints of GCP. For instance, BCP and MCP have been used to model the frequency assignment problem [18] in telecommunications. A mobile cellular network consists of a vast number of cellular phones with each phone operating on a frequency. Phones that are too close to each other (i.e., within a specific radius region) cannot transmit on the same frequency due to interference problems. In addition, certain frequency separations among these phones must be satisfied to reduce the amount of interferences and collisions. The goal is to minimize the number of frequencies being used. This problem can be translated into a BCP in the following way. Each vertex is associated with an individual phone. The phones that require distinct frequencies from each other are joined by edges, and these edges have weights corresponding to the necessary separation distance in the frequencies of the phones. The color assigned to a vertex denotes the frequency of the associated phone. A minimal coloring for this graph achieves the objective of minimizing the number of frequencies in the mobile cellular network.

Under similar conditions, the assignment problem in which each phone having multiple frequencies can be modeled by BMCP [27].

2.3 Existing Algorithms

GCP and its generalizations belong to the $\mathcal{NP}$-hard class of problems, and thus, *exact* polynomial time algorithms for solving them are not expected to exist, unless

$\mathcal{P} = \mathcal{NP}$ [5].

Approximation algorithms are the next attempt in the case of non-existent exact algorithms. Approximation algorithms run in polynomial time, but can only produce an answer within a certain range of the optimal solution. For GCP, the chromatic number, χ, can be approximated within $O\left(|V|\frac{(\log\log|V|)^2}{(\log|V|)^3}\right)$ in polynomial time [16]. However, unless $\mathcal{P} = \mathcal{NP}$, χ is known not to be approximable within $|V|^{1/7-\epsilon}$ for any $\epsilon > 0$.

There is no established approximation algorithm for BCP, MCP, or BMCP. In fact, not much is known about the approximation complexities of the generalized graph coloring problems overall.

Because exact and approximation algorithms that run in polynomial time do not exist for graph colorings in general, most of the focus has been on designing *heuristic* algorithms for these problems. Heuristic algorithms do not guarantee the solution quality; nevertheless, they normally produce excellent solutions in practice.

Among the most prominent heuristic approaches for the classic GCP are constructive methods [6, 22], iterative methods [13], local search methods [26], tabu search methods [17], genetic algorithms [31], and ant-based algorithms [9, 10, 11, 12, 31]. Conversely to the abundance of approaches for GCP, innovations for solving the generalizations are few. The two most well-cited heuristic algorithms for the graph coloring generalizations are Prestwich's local search and constraint propagation [28] and Lim et al.'s Squeaky Wheel Optimization combined with Tabu Search [25].

At the 2002 Graph Coloring Computational Symposium held in New York (Cornell University), Prestwich presented two algorithms [28] for solving the graph coloring problems based on stochastic local search with the space-pruning techniques from constraint programming. The first algorithm, Forward Checking Neighborhood Search (FCNS), is a hybrid of the DSATUR backtracker [6] and the IMPASSE local search algorithm [23, 26]. FCNS is intended specifically for GCP and BCP. To solve BMCP, Prestwich produced another local search and constraint propagation-based algorithm called SATURN. The backtracking and constraint propagation technique in SATURN was based on a Boolean satisfiability (SAT) algorithm called Davis-Logemann-Loveland (DLL). Moreover, the original DLL

was extended into an integer linear program (ILP) model when used in SATURN. We note that [28] does not mention MCP.

Although FCNS achieved very good solutions for BCP, specific parameter tunings were mandatory and depended on different inputs. SATURN, even with the tunings, was unsuccessful among large BMCP graphs. It could not run on half of the tested instances as the ILP model in SATURN consumes an enormous amount of memory. Not much can be said about the performance of FCNS on GCP, as [28] did not provide enough details for this problem. Nonetheless, FCNS and SATURN were among the first endeavors at solving GCP and its generalizations using a universal approach based on backtracking and local constraint search.

In 2003, Lim et al. proposed an algorithm focusing on the graph coloring generalizations using the Squeaky Wheel Optimization (SWO) technique [24]. This approach is laudable for being compatible with the classic GCP, in addition to all of the three generalizations. While outperforming SATURN in BMCP, the preliminary SWO version was not comparable with FCNS in BCP. In 2005, Lim et al. integrated a Tabu Search (TS) optimization [25] to their previous SWO approach. The new algorithm, SWOTS, noticeably enhanced the results of BMCP, but did not improve the quality of others. Not many details were supplied by these SWO-based algorithms for GCP.

In Chapter 3, we present a generalized approach for solving GCP and its variations.

Chapter 3

The Agent-Based Algorithm for Graph Coloring

This chapter describes our agent-based algorithm for the graph coloring problems. The algorithm implements a generalized framework for solving GCP as well as its generalizations. This generic approach embraces code reusability, and hence, allows us to concentrate on a single robust design yet preserves compatibility with different problems.

The principle idea of our Agent-Based Algorithm for Graph Coloring (ABGC) is having a group of agents color the graph. Individual agents work on portions of the graph and together form a coloring for the entire graph. This is distinctive from the traditional Ant Colony Optimization algorithms, where each agent (or ant) finds a complete solution to the problem [14]. Thus, our method encourages cooperation among agents and promotes possibilities for parallel implementation. Another difference in our approach is that the agents do not utilize pheromone as a memory device. In limited experience, we found that adding pheromone to this particular algorithm takes more time and gives no visible quality improvement. Other techniques such as tabu lists and local optimization are also included to help the agents in finding good solutions.

ABGC comprises three main stages: initialization, iteration, and output. In the initialization stage, the algorithm reads the input, applies a preprocessing technique as needed, and finds an initial coloring. Next, a new goal is set that attempts to use fewer colors than what the initial coloring offers for the graph.

The iteration stage features the agents exploring and coloring the graph through successive cycles. Furthermore, this stage contains local optimization to refine the solutions and periodic perturbation to escape local optima. The best coloring found is archived during the running process and finally returned in the output stage.

Figure 3.1 shows the abstract level of ABGC. While the major concepts are generalized for all the mentioned problems, several internal operations are optimized to the specific graph coloring type. For instance, GCP has fewer constraints than the generalizations; hence, its coloring scheme is much simpler. The algorithm details are discussed in the subsequent sections.

$\mathcal{A}$**BGC**$(G = (V, E), d, w)$ //the function weights d, w are optional (for the generalizations)

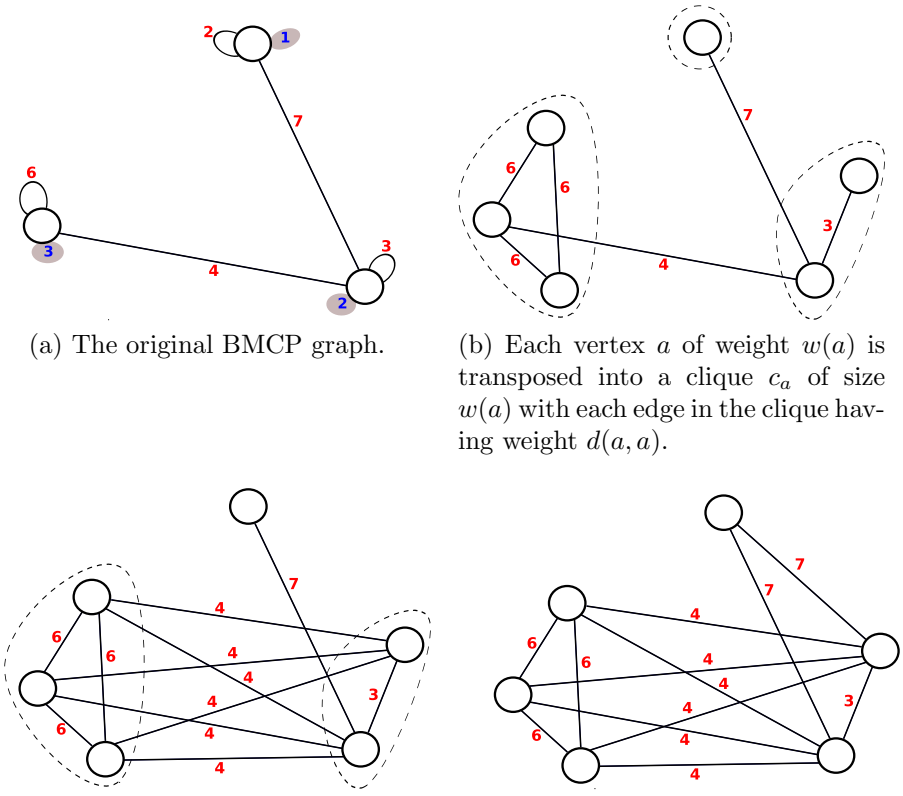
```
1  //Initialization Stage
2  preprocess G if the instance belongs to MCP or BMCP
3  currentColoring ← an initial coloring found by the algorithm described in subsection 3.1.2
4  maxK ← the largest color in currentColoring
5  attemptK ← α * maxK   //attempt new goal, α < 1
6  re-color vertices having colors greater than attemptK
7  update the total conflict in G
8
9  //Iterative Stage
10 for cycle = 1 to nCycles do
11    AgentsOp(G)
12    update the total conflict in G
13    if totalConflict is zero then
14       lColoring ← LocalOpt(currentColoring)   //not necessary for GCP
15       if lColoring uses fewer colors than currentColoring then
16          currentColoring ← lColoring
17       end-if
18       maxK ← the largest color in currentColoring
19       bestColoring ← currentColoring
20       attemptK ← maxK − 1
21       re-color vertices having colors greater than attemptK
22       update the total conflict in G
23    end-if
24    perform perturbation operation periodically to escape local optima
25 end-for
26
27 //Output Stage
28 return bestColoring and maxK
```

Figure 3.1: ABGC algorithm.

3.1 The Initialization Stage

After reading the input graph, the algorithm may apply a preprocessing step on the input. This procedure, given in subsection 3.1.1, effectively eliminates the dissimilarities in the generalizations by transforming a BMCP or MCP instance into a BCP equivalent. Accordingly, the algorithm only needs to solve BCP since all the generalizations are converted to the format of this problem. Note that this preprocessing step is not necessary for GCP or BCP.

3.1.1 Preprocessing the Input Graph



(a) The original BMCP graph.

(b) Each vertex a of weight $w(a)$ is transposed into a clique c_a of size $w(a)$ with each edge in the clique having weight $d(a, a)$.

(c) If $(a, b) \in E$ in the original graph, then each vertex in clique c_a also connects to each vertex in c_b with an edge of weight $d(a, b)$.

(d) The new BCP graph after the transformation.

Figure 3.2: Transformation from BMCP to BCP.

For instances in MCP or BMCP, where each vertex has a positive integer weight, we apply a transformation derived from the one used in [25] to the input graph

to transform it into an instance of GCP or BCP. A vertex a of weight $w(a)$ is transposed into a clique c_a of size $w(a)$, with each edge in the clique having an edge weight equal to $d(a, a)$. Moreover, if (a, b) is an edge in the original graph, then we also connect each vertex in c_a to each vertex in c_b and assign such an edge the weight $d(a, b)$. Figure 3.2 demonstrates how the preprocessing step transforms a BMCP instance into an equivalent BCP instance.

We note that this transformation takes exponential time if vertex weight $w(a)$ is exponential in the number of vertices in the graph. However, the approach is reasonable in practice as the number of vertices in the graph is much more than the weights in real-world applications. For example, the number of different frequencies assigned to a cell phone is much less than the number of cell phones.

Our next objective is to quickly find an upper bound on the optimal solution of the input graph. For this purpose, we run the algorithm described in subsection 3.1.2 to find an initial coloring.

3.1.2 The Initial Coloring

After preprocessing the input graph as needed, ABGC resorts to a greedy-based algorithm to determine an initial coloring upon which the agents can work. The initial coloring generated from this greedy algorithm is valid, although not necessarily optimal or even good. In fact, except for a few trivial graph instances, the agents can always improve upon the initial coloring. We conjecture that the agents are more fruitful when building upon such coloring than when starting from ground zero.

For GCP, we create a greedy-based algorithm based on MXRLF [9], which combines features from the RLF [22] and XRLF [19] methods. The algorithm uses successively larger colors, starting with 1, to color vertices of the graph. For each color x, the algorithm selects a set of vertices to be colored with x. Let W, the white list, be a set of candidate vertices for color x, and B, the black list, be the list of vertices that cannot be colored with x. During its coloring process, MXRLF selects vertices from the white list and avoids those from the black list. For each color x, the algorithm initializes W to contain all uncolored vertices and B as empty. The first vertex chosen is the one with maximum degree from W.

After that vertex is colored, its adjacent vertices cannot have the same color and therefore are black listed (i.e., they are moved from W to B). The algorithm then sequentially selects the next vertex from W having the highest number of adjacent vertices in B. Ties are broken arbitrarily. The selection continues until W is empty or the number of selected vertices reaches a size limit of λ [19]. The current color number, x, is next incremented (i.e., to 2), and this process repeats until all vertices in G are colored. As our intention was to quickly find an upper bound on the optimal solution, we omitted the exhaustive search method for building the coloring from the XRLF algorithm [19].

```
ℐterativeGreedy(G = (V, E), d)
 1  initialize the current coloring C empty
 2  while there is any uncolored vertex in G do
 3    u ← an uncolored vertex selected at random or based on max degree
 4    forbiddenSet(u) ← ∅   //initialize the forbiddenSet of u
 5    for each vertex v that is adjacent to u do
 6      if v already has a color then
 7        update the set of forbidden colors, forbiddenSet(u)
 8          based on the color of v and the weight of the edge (u, v)
 9      end-if
10    end-for
11    assign u a color selected based from forbiddenSet
12      and the rule described in Section 3.2.2
13  end-while
14  return the current coloring C
```

Figure 3.3: The Iterative Greedy (IG) algorithm.

For the generalized coloring problems (more specifically, BCP), a greedy-based algorithm, given in Figure 3.3, called $\mathcal{I}$terative$\mathcal{G}$reedy (IG), is employed. At each step of the algorithm, IG selects a vertex to color next. Selection is done in a random manner or based on the vertex degree, i.e., the vertex of highest degree is selected first among the uncolored vertices. When a vertex u is selected to be colored, IG applies the same scheme as in ABGC. This coloring scheme is described in Section 3.2.2.

ABGC uses IG to find an initial coloring as follows. First, IG runs with the highest degree selection scheme. The coloring of this run is kept. Next, the algorithm runs IG 20 times with the random selection scheme. These colorings are

also saved. Finally, the best coloring found among the 21 runs is returned as the initial coloring for ABGC.

The algorithm sets the initial coloring found as the current coloring of G and then attempts to have the agents find a better coloring. Assuming $maxK$ is the largest color in the current coloring (the initial coloring), an initial number of colors available to the agents for coloring the graph is set to be $attemptK = \lceil \alpha * maxK \rceil$, where $\alpha < 1$. The parameter α and others to follow are specified in Appendix C. The current coloring is changed into an $attemptK$-coloring by assigning randomly selected colors in the $[1 \ldots attemptK]$ interval to vertices with colors greater than $attemptK$. Note that this adjustment may violate some constraints and consequently invalidates the current coloring of G. The current coloring may now have conflict. Nonetheless, it uses fewer colors than what the initial coloring gives. Hence, by resolving all conflict in G without using more than $attemptK$ colors, the agents would have a better coloring, i.e., a valid $attemptK$-coloring.

3.2 The Iterative Stage

The iterative stage contains repeating cycles in which the agents resolve the conflict on the graph in an attempt to find a valid $attemptK$-coloring. Within each cycle, the agents move and re-color the portions of the graph close to their current locations, using only the set of currently available colors, $\{1, \ldots, attemptK\}$. The $attemptK$-coloring goal is found whenever the current coloring of the graph satisfies all the constraints specified by the problem. When this occurs, the algorithm breaks from the cycle, saves the current coloring, attempts a new objective by reducing the $attemptK$ number of colors, creates a new set of available colors for the agents, and starts a new cycle. On the other hand, if a valid coloring cannot be obtained for a specific time period, the algorithm may perturb the current coloring to deter it from getting trapped in a local optimum. This process reiterates until the terminating criteria are met.

Figure 3.4 captures the main operations of agents. In the following subsections we examine various parts of the iterative stage thoroughly.

```
AgentsOp(G = (V, E))
 1  for agent = 1 to nAgents do
 2    if there is no conflict in G then break
 3    agent clears its recentlyVisited tabu list
 4    agent is placed on a max-conflict based vertex of G
 5    agent colors its current vertex
 6    for move = 1 to nMoves do
 7      agent moves to a neighboring vertex by taking two steps
 8      agent colors its current vertex
 9      agent updates its recentlyVisited tabu list
10    end-for
11  end-for
```

Figure 3.4: The operations of agents.

3.2.1 How an Agent Moves

In each cycle, an agent is initially placed at a random vertex and attempts to color that location and its vicinity. To do this, the agent takes two steps, i.e., traverses along a path of length two, and subsequently colors the vertex upon which it lands. The process of taking a number of steps then coloring the vertex is called a *move*. Within a cycle, each agent makes *nMoves* moves.

In the first step of a move, the agent selects and moves to a random adjacent vertex. In the second step, the agent decides on an adjacent vertex having the highest conflict among all adjacent vertices and moves there. Ties are broken arbitrarily. If there is no conflict among the adjacent vertices after the first step, the agent is relocated to the vertex that has the highest conflict in the entire graph. Again, ties are broken arbitrarily. All agents stop moving when there is no conflict remaining, and the algorithm prepares for the next cycle by reducing the number of available colors by one.

Additionally, each agent also has a tabu list containing its recently visited vertices. The tabu list is implemented to help the agent move away from its recent visits. Whenever an agent selects a vertex to move to, it always avoids vertices that are in its current tabu list. The agent also places vertices that it has just visited on its current tabu list. This tabu list has a fixed size, δ, and consequently when the list is full, the new vertex replaces the oldest one. The reason for having the random first step in a move and the tabu list is to induce search space exploration

and alleviate the possibility of being trapped in a local optimum. With an increase in running time, we can gain some improvement in the procedure by allowing each agent to take more than two steps per move. The above algorithm in Figure 3.4 can be easily extended to accommodate this option.

3.2.2 How a Vertex is Colored

Within each cycle, an agent colors only a limited local area of the graph without any global knowledge of the graph and uses only colors from the set of *attemptK* available colors. The purpose of re-coloring a vertex is to resolve any conflict to zero, if possible. Figure 3.5 captures the main ideas of the coloring scheme.

```
Color Vertex(u)
 1  determine forbiddenSet from the adjacent vertices of u
 2  compute the occurrence rate of the members in forbiddenSet
 3  eligibleSet ← {1, ..., attemptK} − forbiddenSet
 4  if eligibleSet is empty then
 5     assign u the least occurred color from forbiddenSet
 6  else
 7     decompose eligibleSet into a union of intervals
 8     assign u the color that is the median of the largest interval
 9  end-if
10  update the conflict at u
```

Figure 3.5: How a vertex is colored.

To color a vertex u, a set of eligible colors that can be used to color u is determined. For each vertex v adjacent to u, a set of colors that is in conflict with the color of v is formed. This set of conflicting colors is computed by examining the color of v and the weight of the edge (u, v). For example, if the color of v is 9 and $d(u, v)$ is 3, then to achieve zero conflict, the coloring of u cannot be in the set $\{7, 8, 9, 10, 11\}$. The conflicting color sets of the adjacent vertices of u form a union called the set of forbidden colors, *forbiddenSet*. Note for GCP, the weight of any edge is one, therefore the *forbiddenSet* of u simply contains the colors of the vertices adjacent to u. This is the same process as that of determining the set of forbidden colors in the IG algorithm of Figure 3.3. In addition, the algorithm monitors how frequently each member in the set of forbidden colors occurs. Specifically, every

color in *forbiddenSet* has an occurrence rate proportional to the number of the conflicting color sets of adjacent vertices containing that color.

Once the set of forbidden colors has been computed, the set of eligible colors, *eligibleSet*, can be easily obtained by taking the difference between the set of forbidden colors and the set of all colors, i.e., $\{1, \ldots, attemptK\}$. These are the colors that resolve the conflict at u to zero.

In the case when the set of eligible colors is empty, the color that causes the least conflict with the adjacent vertices of u, i.e., the color with the lowest occurrence rate, is selected. Ties are broken arbitrarily.

For GCP, when there exists several choices among the available colors satisfying the requirement, the algorithm picks a random one for u. A more complicated coloring scheme is used for the generalizations (specifically BCP). If the cardinality of the set of eligible colors is greater than one, then it can be viewed as a union of intervals. Next, the algorithm selects the color that is the median of the largest interval in the set of eligible colors. If there is more than one interval of the largest size, one of those intervals is chosen at random. Furthermore, when two medians are available, one of them is chosen at random. For example, if the set of eligible colors is $\{1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 15\}$, then this is the same as $[1 \ldots 5] \cup [8 \ldots 9] \cup [11 \ldots 15]$. One of the intervals $[1 \ldots 5]$ or $[11 \ldots 15]$ is selected at random, say, the latter. Therefore, the chosen color is 13, the median of $[11 \ldots 15]$. This selection scheme allows later vertices to have more room to meet their constraints, i.e., the set of eligible colors will be larger for neighboring vertices.

After the coloring of u, the conflict at this vertex is updated, and the agent adds u to its tabu list. Since the list has a fixed size, the newest vertex replaces the oldest one in the case when the list is full. Note that the agent does not have knowledge of the total conflict for the entire graph.

3.2.3 Local Optimization Operation

When the agents identify a valid coloring, it is usually near a local optimum. However, because agents cannot perform deterministic hill-climbing, a local optimization is often used to bring this solution to a nearby local optimum.

For the generalized coloring problems, we implement a local optimization al-

gorithm that performs a compression-like operation to the interval of used colors. This algorithm is not necessary for GCP. The colorings found by the agents for GCP are already in very compacted form since the colors for the adjacent vertices only need to differ from each other by one.

The local optimization in Figure 3.6 is applied to the valid coloring whenever one is found by the agents. Using ideas from the $\mathcal{I}$terative$\mathcal{G}$reedy algorithm in Figure 3.3, the local optimization algorithm first sorts the vertex set into decreasing order of vertex colors, i.e., the colors given in the input coloring. The algorithm subsequently erases all vertex colors from the graph and starts coloring the vertices one at a time in the order of the sorted vertex set, i.e., vertices that have higher color numbers in the original input coloring will be selected first. For each vertex to be colored, the algorithm computes a set of forbidden colors in the same manner as in the algorithm given by Figure 3.3. The vertex is consequently colored using the smallest color number that is not in the forbidden set. Finally, the algorithm terminates and returns the coloring that it found.

$\mathcal{L}$ocal$\mathcal{O}$pt(C) //C is a valid coloring
1 sort the vertices in decreasing order of color numbers from the input coloring C
2 erase the coloring C
3 **for** each vertex u in the sorted order **do**
4 *forbiddenSet*$(u) \leftarrow \emptyset$ //initialize the *forbiddenSet* of u
5 **for** each vertex v that is adjacent to u **do**
6 **if** v already has a color **then**
7 update *forbiddenSet*(u)
8 based on the color of v and the weight of the share edge (u, v)
9 **end-if**
10 **end-for**
11 assign u a color selected based from *forbiddenSet*
12 and the rule described in Section 3.2.2
13 **end-while**
14 **return** the coloring C

Figure 3.6: The Local Optimization algorithm.

If the coloring returned by the local optimization algorithm is better than the current best coloring, it replaces the current best coloring. Otherwise, it is discarded. We note that colorings obtained by repetitively executing the $\mathcal{I}$terative$\mathcal{G}$reedy algorithm followed by the local optimization operation alone are never as good as

those obtained by ABGC. In other words, the explorations of agents discover potential solutions, which are then enhanced by the exploitations from the local optimization.

3.2.4 Perturbation and Stopping Condition

At any time during the algorithm, whenever there is no conflict found in the current coloring, a new goal is set by reducing the number of available colors, *attemptK*, by one. Next each vertex with color greater than *attemptK* is assigned a randomly selected color in the interval $[1 \dots attemptK]$. At this point the current coloring may be an invalid one, i.e., the total conflict is non-zero. The algorithm now starts a new cycle. Notice that even though pheromone was not utilized as a memory device in this algorithm, the coloring of cycle $i+1$ is based on how the agents color in cycle i.

Greedy-based algorithms, such as ABGC, have a natural tendency to get trapped in a plateau or a local optimum. To mitigate this problem, we add a *jolt* procedure analogous to making random jumps in the search space to get out of a local optimum. More specifically, the jolt operation perturbs the current coloring if agents have not been able to improve the number of colors used for the last *nJoltCycles* consecutive cycles. Vertices that have conflict in the top $\beta\%$ are selected, and their neighbors are randomly re-colored using $\gamma\%$ of the current set of available colors. The intention of the jolt operation is to cause enough disturbance in the current coloring to escape the local optimum, but not enough to create havoc in the coloring that has been built up to that point.

The algorithm stops after it has run for a preset number of cycles, *nCycles*, or if it has not made any improvement for a number of *nBreakCycles* consecutive cycles. All parameters are defined in Appendix C.

3.3 The Output Stage

At the end, ABGC simply returns the best coloring stored throughout the course of the program. The largest color in this coloring, *maxK*, denotes the quality of the solution, i.e., a smaller *maxK* value corresponds to a better result quality.

In Chapter 4, we present and compare the performance of ABGC to other algorithms.

Chapter 4

Experimental Results

In this chapter, we describe the results of running ABGC on a collection of benchmark graphs and compare them against other algorithms. We used graph instances from DIMACS [8] (Center for Discrete Mathematics and Theoretical Computer Science, Rutgers, New Jersey) to evaluate our algorithm. Our ABGC algorithm was implemented in C++ and ran on a 3.0 GHz Pentium 4 PC with 2 GB of RAM using the Linux operating system. Since benchmarks are often performed on disparate platforms, the DIMACS community offers the *dfmax* utility for the comparison of different machines. Table A.1 in Appendix A reports the computation time of ABGC running on dfmax.

4.1 Results for the Graph Coloring Problem

For GCP, we tested our algorithm on 119 DIMACS benchmark graphs belonging to various categories given at [8][‡]. These varieties of graphs fall under different categories. Tables B.1 and B.2 in Appendix B give a summary on these benchmark graphs.

Stochastic algorithms involving random variables may yield a dissimilar outcome on each run; thus, they are often tested in a number of trials to obtain more accurate result statistics, such as average solution quality, running time, etc. For

[‡] [8] is a website maintained at Penn State Harrisburg. In addition to archiving the contents of the several DIMACS Challenge Series, it provides updates and bug fixes that were not in the original website.

each of these 119 graphs, we ran ABGC for 100 times (trials). Of the 119 graphs, there are 63 graphs with best-known bound on the chromatic numbers. Our algorithm found matching bound for 57 out of these 63 graphs. There are six graphs for which our algorithm got poorer results, but are within one of the best-known bound.

Tables 4.1, 4.2, 4.3, and 4.4 give experimental results on the 119 graphs. For each instance, we listed the name of the graph, the best-known bound on the chromatic number (χ^*), the best ($\eta_{\min}$), the worst ($\eta_{\max}$), the average (η_{avg}), the standard deviation (σ), and the average running time in seconds (τ_{avg}) of the results produced by our algorithm in 100 trials. The running times for a number of graphs were too small to be recorded and therefore recorded as 0.00. Graphs without best-known bound on the chromatic number have the corresponding entries marked with '–'. The boldfaced values represent the results found by our algorithm that match the best-known bound on the chromatic numbers.

We did not compare the results of GCP against other algorithms since many of them did not use the same set of graph instances from DIMACS. Furthermore, we found no existing algorithms in the literature run on such a comprehensive set of graphs compared to ours. This collection of 119 graphs is the entire list of available DIMACS instances made for GCP. In fact, several large graphs, for example, qg.order100 with 990,000 vertices and 10,000 edges, even caused problems to a few utilities provided on the DIMACS website.

4.2 Results for the Graph Coloring Generalizations

We tested ABGC for the generalizations using 33 geometric (GEOM) benchmark graphs available at [8]. For each graph, the Bandwidth Coloring problem requires an edge weight function, the Multi Coloring problem requires a vertex weight function, and the Bandwidth Multi Coloring problem requires both weight functions. Thus, there are 99 graph instances in total. Information about these graphs is summarized in Table B.3, Appendix B.

For each of the instances, we ran our algorithm for 100 trials. Tables 4.5, 4.6,

and 4.7 provide detailed information on the trials using a similar format as in the GCP trials. In addition, these tables also include comparison data of ABGC against previous algorithms. The boldfaced value denotes the best result quality among the considered algorithms. The columns with the following header notations list the best results (i.e., $\eta_{\min}$) achieved by the corresponding algorithms.

- **FCNS**: Prestwich's algorithm for GCP and BCP [28]
- **SATURN**: Prestwich's algorithm for BMCP [28]
- **SWO**: Lim et al.'s Squeaky Wheel Optimization algorithm [24]
- **SWOTS**: Lim et al.'s Squeaky Wheel Optimization algorithm hybridized with Tabu Search [25]
- **ABGC**: Our agent-based algorithm, ABGC

For the Bandwidth Coloring problem, our results matched or surpassed all results from SWO and SWOTS, but were not as good as those of FCNS. As pointed out in FCNS, their algorithm demands certain parameters to be tuned for each graph or class of graphs. That is not the case for our ABGC algorithm.

Our results matched those of SWO-based algorithms in all 33 cases for the Multi Coloring problem. No results were provided by [28] for MCP. These graph instances for MCP are relatively simple for either ABGC or SWO-based algorithms. For each of these instances, ABGC obtained the standard deviation value of zero in all the trials.

For the Bandwidth Multi Coloring, both ABGC and SWO-based algorithms prevailed over SATURN, which failed to run in half of the instances. Where results were not available, we marked the corresponding entries with '–'. Compared against the algorithms of SWO and SWOTS, ABGC performed better in the majority of the larger graphs while it did worse in a few smaller ones.

Instances	100 runs per instance					
	χ^*	$\eta_{\min}$	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
1-FullIns_3	–	4	4	4.00	0.00	0.03
1-FullIns_4	–	5	5	5.00	0.00	0.35
1-FullIns_5	–	6	6	6.00	0.00	4.53
1-Insertions_4	4	5	5	5.00	0.00	0.12
1-Insertions_5	–	6	6	6.00	0.00	1.75
1-Insertions_6	–	7	7	7.00	0.00	16.85
2-FullIns_3	–	5	5	5.00	0.00	0.06
2-FullIns_4	–	6	6	6.00	0.00	2.10
2-FullIns_5	–	7	7	7.00	0.00	22.86
2-Insertions_3	4	**4**	4	4.00	0.00	0.02
2-Insertions_4	4	**5**	5	5.00	0.00	0.79
2-Insertions_5	–	6	6	6.00	0.00	13.81
3-FullIns_3	–	6	6	6.00	0.00	0.23
3-FullIns_4	–	7	7	7.00	0.00	9.36
3-FullIns_5	–	8	8	8.00	0.00	51.34
3-Insertions_3	4	**4**	4	4.00	0.00	0.08
3-Insertions_4	–	5	5	5.00	0.00	3.14
3-Insertions_5	–	6	6	6.00	0.00	28.20
4-FullIns_3	–	7	7	7.00	0.00	0.56
4-FullIns_4	–	8	8	8.00	0.00	17.66
4-FullIns_5	–	9	9	9.00	0.00	125.94
4-Insertions_3	3	4	4	4.00	0.00	0.13
4-Insertions_4	–	5	5	5.00	0.00	10.14
5-FullIns_3	–	8	8	8.00	0.00	1.08
5-FullIns_4	–	9	9	9.00	0.00	26.12
abb313GPIA	–	9	10	9.53	0.50	55.23
anna	11	**11**	11	11.00	0.00	1.25
ash331GPIA	–	4	5	4.18	0.38	15.74
ash608GPIA	–	4	5	4.31	0.46	24.61
ash958GPIA	–	4	5	4.43	0.50	39.20
david	11	**11**	11	11.00	0.00	0.44

Table 4.1: GCP results on 119 DIMACS graphs.

Instances	100 runs per instance					
	χ^*	$\eta_{\min}$	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
DSJC1000.1	–	21	22	21.47	0.50	51.96
DSJC1000.5	–	91	93	91.95	0.52	199.21
DSJC1000.9	–	228	233	230.85	0.99	416.77
DSJC125.1	–	5	6	5.51	0.50	0.84
DSJC125.5	–	17	18	17.75	0.43	1.75
DSJC125.9	–	44	44	44.00	0.00	2.62
DSJC250.1	–	8	9	8.50	0.50	4.78
DSJC250.5	–	28	30	29.18	0.41	12.27
DSJC250.9	–	72	73	72.36	0.49	23.57
DSJC500.1	–	13	13	13.00	0.00	21.99
DSJC500.5	–	50	52	51.13	0.39	79.94
DSJC500.9	–	127	129	128.38	0.58	125.66
DSJR500.1	–	12	12	12.00	0.00	13.99
DSJR500.1c	–	85	86	85.17	0.38	112.44
DSJR500.5	–	128	130	129.23	0.47	114.47
fpsol2.i.1	65	**65**	65	65.00	0.00	54.00
fpsol2.i.2	30	**30**	30	30.00	0.00	46.53
fpsol2.i.3	30	**30**	30	30.00	0.00	41.72
games120	9	**9**	9	9.00	0.00	0.68
homer	13	**13**	13	13.00	0.00	17.44
huck	11	**11**	11	11.00	0.00	0.23
inithx.i.1	54	**54**	54	54.00	0.00	77.26
inithx.i.2	31	**31**	31	31.00	0.00	63.76
inithx.i.3	31	**31**	31	31.00	0.00	63.37
jean	10	**10**	10	10.00	0.00	0.29

Table 4.2: GCP results on 119 DIMACS graphs.

Instances	100 runs per instance					
	χ^*	$\eta_{\min}$	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
latin_square_10	–	100	103	101.48	0.64	305.21
le450_15a	15	**15**	15	15.00	0.00	31.52
le450_15b	15	**15**	15	15.00	0.00	28.00
le450_15c	15	**15**	21	19.74	1.81	41.70
le450_15d	15	**15**	21	17.02	1.42	42.66
le450_25a	25	**25**	25	25.00	0.00	28.71
le450_25b	25	**25**	25	25.00	0.00	27.14
le450_25c	25	26	26	26.00	0.00	39.55
le450_25d	25	26	26	26.00	0.00	40.71
le450_5a	5	**5**	6	5.32	0.47	16.15
le450_5b	5	**5**	6	5.44	0.50	16.40
le450_5c	5	**5**	5	5.00	0.00	20.44
le450_5d	5	**5**	5	5.00	0.00	20.71
miles1000	42	**42**	42	42.00	0.00	2.55
miles1500	73	**73**	73	73.00	0.00	5.11
miles250	8	**8**	8	8.00	0.00	0.57
miles500	20	**20**	20	20.00	0.00	1.53
miles750	31	**31**	31	31.00	0.00	1.95
mug100_1	4	**4**	4	4.00	0.00	0.25
mug100_25	4	**4**	4	4.00	0.00	0.35
mug88_1	4	**4**	4	4.00	0.00	0.17
mug88_25	4	**4**	4	4.00	0.00	0.16
mulsol.i.1	49	**49**	49	49.00	0.00	7.30
mulsol.i.2	31	**31**	31	31.00	0.00	5.69
mulsol.i.3	31	**31**	31	31.00	0.00	5.86
mulsol.i.4	31	**31**	31	31.00	0.00	5.81
mulsol.i.5	31	**31**	31	31.00	0.00	5.85
myciel3	4	**4**	4	4.00	0.00	0.01
myciel4	5	**5**	5	5.00	0.00	0.01
myciel5	6	**6**	6	6.00	0.00	0.05
myciel6	7	**7**	7	7.00	0.00	0.43
myciel7	8	**8**	8	8.00	0.00	2.29

Table 4.3: GCP results on 119 DIMACS graphs.

Instances	100 runs per instance					
	χ^*	$\eta_{\min}$	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
qg.order30	30	**30**	30	30.00	0.00	44.31
qg.order40	40	**40**	40	40.00	0.00	71.91
qg.order60	60	**60**	60	60.00	0.00	226.36
qg.order100	100	**100**	100	100.00	0.00	1534.70
queen10_10	–	11	11	11.00	0.00	0.99
queen11_11	11	12	13	12.02	0.14	1.34
queen12_12	–	13	14	13.40	0.49	1.84
queen13_13	13	14	15	14.66	0.48	2.56
queen14_14	–	16	16	16.00	0.00	3.59
queen15_15	–	17	17	17.00	0.00	4.90
queen16_16	–	18	18	18.00	0.00	6.45
queen5_5	5	**5**	5	5.00	0.00	0.01
queen6_6	7	**7**	7	7.00	0.00	0.03
queen7_7	7	**7**	7	7.00	0.00	0.06
queen8_12	12	**12**	12	12.00	0.00	0.53
queen8_8	9	**9**	9	9.00	0.00	0.14
queen9_9	10	**10**	10	10.00	0.00	0.37
school1_nsh	–	14	14	14.00	0.00	16.87
school1	–	14	14	14.00	0.00	23.75
wap01a	–	43	43	43.00	0.00	158.15
wap02a	–	42	43	42.80	0.40	145.21
wap03a	–	45	46	45.60	0.49	514.93
wap04a	–	44	45	44.86	0.35	476.18
wap05a	–	50	50	50.00	0.00	67.49
wap06a	–	42	43	42.86	0.35	85.69
wap07a	–	43	44	43.32	0.47	169.88
wap08a	–	42	44	43.02	0.32	175.84
will199GPIA	–	7	7	7.00	0.00	22.44
zeroin.i.1	49	**49**	49	49.00	0.00	8.81
zeroin.i.2	30	**30**	30	30.00	0.00	8.58
zeroin.i.3	30	**30**	30	30.00	0.00	8.23

Table 4.4: GCP results on 119 DIMACS graphs.

Instances	Algorithms				100 runs per instance (ABGC)			
	FCNS	SWO	SWOTS	ABGC	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
geom20	21	21	21	21	21	21.00	0.00	0.03
geom20a	20	22	22	20	24	21.61	0.77	0.03
geom20b	13	14	14	13	14	13.11	0.31	0.04
geom30	28	29	29	28	29	28.04	0.20	0.07
geom30a	27	32	32	27	32	29.10	1.30	0.09
geom30b	26	26	26	26	27	26.07	0.26	0.13
geom40	28	28	28	28	29	28.08	0.27	0.11
geom40a	37	38	38	37	42	38.60	0.92	0.18
geom40b	33	34	34	33	38	35.17	1.39	0.25
geom50	28	28	28	28	31	28.17	0.53	0.24
geom50a	50	52	52	50	56	52.08	1.40	0.39
geom50b	**35**	38	38	36	44	38.92	1.64	0.39
geom60	33	34	34	33	35	33.50	0.54	0.39
geom60a	50	53	53	50	57	52.05	1.21	0.65
geom60b	43	46	46	43	51	46.44	1.58	0.83
geom70	38	38	38	38	42	38.33	0.63	0.66
geom70a	62	63	63	62	71	65.76	2.28	0.84
geom70b	**48**	54	54	50	58	53.72	1.46	1.02
geom80	41	42	42	41	45	42.09	0.94	0.70
geom80a	63	66	66	63	76	69.01	2.59	1.26
geom80b	**61**	65	65	63	74	67.62	1.92	1.60
geom90	46	46	46	46	51	46.88	1.05	0.92
geom90a	**64**	69	69	66	75	69.82	1.88	1.85
geom90b	**72**	77	77	74	85	78.96	2.28	2.57
geom100	50	51	51	50	60	52.24	1.42	1.23
geom100a	**70**	76	76	72	81	75.53	1.89	2.75
geom100b	**73**	88	88	76	93	82.52	2.16	3.59
geom110	50	53	53	50	54	52.14	0.96	1.48
geom110a	**74**	82	82	75	83	79.02	1.58	3.35
geom110b	**79**	88	88	84	97	89.05	2.30	4.22
geom120	60	62	62	**59**	67	61.79	1.46	1.67
geom120a	**84**	92	92	86	95	90.04	1.75	3.93
geom120b	**87**	98	98	90	102	96.43	2.15	5.90

Table 4.5: BCP results on 33 GEOM DIMACS graphs.

Instances	Algorithms				100 runs per instance (ABGC)			
	FCNS	SWO	SWOTS	ABGC	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
geom20	–	28	28	28	28	28.00	0.00	3.33
geom20a	–	30	30	30	30	30.00	0.00	3.49
geom20b	–	8	8	8	8	8.00	0.00	0.13
geom30	–	26	26	26	26	26.00	0.00	3.27
geom30a	–	40	40	40	40	40.00	0.00	8.65
geom30b	–	11	11	11	11	11.00	0.00	0.42
geom40	–	31	31	31	31	31.00	0.00	8.46
geom40a	–	46	46	46	46	46.00	0.00	15.51
geom40b	–	14	14	14	14	14.00	0.00	2.40
geom50	–	35	35	35	35	35.00	0.00	14.63
geom50a	–	61	61	61	61	61.00	0.00	43.05
geom50b	–	17	17	17	17	17.00	0.00	4.09
geom60	–	36	36	36	36	36.00	0.00	16.59
geom60a	–	65	65	65	65	65.00	0.00	53.30
geom60b	–	22	22	22	22	22.00	0.00	3.31
geom70	–	44	44	44	44	44.00	0.00	32.56
geom70a	–	71	71	71	71	71.00	0.00	62.29
geom70b	–	22	22	22	22	22.00	0.00	3.63
geom80	–	63	63	63	63	63.00	0.00	50.61
geom80a	–	68	68	68	68	68.00	0.00	75.08
geom80b	–	25	25	25	25	25.00	0.00	4.64
geom90	–	51	51	51	51	51.00	0.00	67.68
geom90a	–	65	65	65	65	65.00	0.00	73.99
geom90b	–	28	28	28	28	28.00	0.00	7.37
geom100	–	60	60	60	60	60.00	0.00	88.28
geom100a	–	81	81	81	81	81.00	0.00	127.87
geom100b	–	30	30	30	30	30.00	0.00	7.99
geom110	–	62	62	62	62	62.00	0.00	111.81
geom110a	–	91	91	91	91	91.00	0.00	198.22
geom110b	–	37	37	37	37	37.00	0.00	14.00
geom120	–	64	64	64	64	64.00	0.00	136.94
geom120a	–	93	93	93	93	93.00	0.00	308.78
geom120b	–	34	34	34	34	34.00	0.00	13.63

Table 4.6: MCP results on 33 GEOM DIMACS graphs.

	Algorithms				100 runs per instance (ABGC)			
Instances	FCNS	SWO	SWOTS	ABGC	$\eta_{\max}$	η_{avg}	σ	$\tau_{\text{avg (secs)}}$
geom20	159	149	149	149	158	150.86	2.17	6.85
geom20a	175	169	169	169	176	170.78	1.55	11.27
geom20b	44	44	44	44	46	44.38	0.56	0.24
geom30	168	160	160	160	169	160.99	1.44	9.49
geom30a	235	211	**209**	210	225	214.94	3.04	25.39
geom30b	79	77	77	77	79	77.59	0.53	1.24
geom40	189	167	167	167	176	167.65	1.24	24.57
geom40a	260	214	**213**	214	226	216.37	1.91	66.72
geom40b	80	76	74	74	87	77.53	2.35	3.04
geom50	257	224	224	224	232	225.39	1.55	57.48
geom50a	395	326	318	**317**	336	325.68	3.48	379.48
geom50b	89	87	87	**85**	99	89.22	2.06	4.54
geom60	279	258	258	258	264	259.15	1.28	64.39
geom60a	–	368	358	357	369	363.47	2.42	203.23
geom60b	128	119	**116**	117	140	125.59	4.81	10.64
geom70	310	279	273	**267**	278	271.77	1.76	110.96
geom70a	–	478	**469**	470	488	478.27	3.49	276.63
geom70b	133	124	121	121	131	125.61	2.02	12.46
geom80	–	394	383	**382**	393	387.76	2.31	157.88
geom80a	–	379	379	**367**	382	372.92	3.22	239.61
geom80b	152	145	141	**139**	147	142.43	1.56	18.01
geom90	–	335	332	332	339	335.60	1.78	180.91
geom90a	–	382	**377**	378	417	388.28	8.91	387.54
geom90b	–	157	157	**150**	164	155.96	2.60	22.50
geom100	–	413	**404**	405	416	409.10	2.45	292.10
geom100a	–	462	459	**440**	461	449.46	4.27	548.34
geom100b	–	172	170	**164**	178	171.26	2.70	27.85
geom110	–	389	383	**378**	391	384.47	2.76	405.16
geom110a	–	501	494	**487**	502	493.61	3.24	1069.85
geom110b	–	210	**206**	208	228	213.25	3.48	43.86
geom120	–	409	402	**398**	408	401.84	2.36	790.21
geom120a	–	564	556	**548**	565	556.44	3.67	1660.62
geom120b	–	201	199	**198**	209	203.49	2.62	41.26

Table 4.7: BMCP results on 33 GEOM DIMACS graphs.

Chapter 5

Conclusion

In this thesis, we studied the classic Graph Coloring problem (GCP) and its generalizations, namely Bandwidth Coloring, Multi Coloring, and Bandwidth Multi Coloring. These problems are standard models for numerous real-world applications and accordingly motivate people to solve them.

The research in this thesis proposes an agent-based algorithm called ABGC for GCP and its variations. The algorithm features agents making decisions to color portions of the graph based on surrounding information, and this collaboration produces the coloring of the entire graph.

Another notable advantage of ABGC is its *generic* framework. Compared to other algorithms that are either designed to solve each individual problem or solve multiple problems, but only effectively solve one of the problems, ABGC covers GCP and the three generalizations. Moreover, the algorithm is very flexible due to its compatibility with a variety of graph categories. Additional techniques such as graph preprocessing, tabu lists, and greedy-based local optimization also contribute support to the agents. Experimental results show that our algorithm is very competitive with other algorithms.

In possible future work, we envision a pheromone concept to lock the critical colors of several portions on the graph. A general distributed framework is also desirable for agent-based algorithms because it allows separate groups of agents to run on different computer clusters and exchange data periodically.

Bibliography

[1] Allen, M., G. Kumaran, and T. Liu, "A Combined Algorithm for Graph-Coloring in Register Allocation," Proc. Computational Symposium on Graph Coloring and its Generalizations. Ithaca, New York, USA, 2002, pp. 110–111.

[2] Appel, K. and W. Haken "Solution of the Four Color Map Problem," Scientific American, 237(4), 1977, pp. 108–121.

[3] Appel, K., W. Haken, and J. Koch, "Every Planar Map is Four Colorable," Illinois Journal of Mathematics, 21, 1977, pp. 429–567.

[4] Barnier, N. and P. Brisset, "Graph Coloring for Air Traffic Flow," CPAIOR'02: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems, Le Croisic, France, 2002, pp. 133–147.

[5] Bellare, M., O. Goldreich, and M. Sudan, "Free Bits, PCPs and Non-Approximability – Towards Tight Results," SIAM Journal on Computing, 27(3), 1998, pp. 804–915.

[6] Brelaz, D., "New Methods to Color the Vertices of a Graph," Communications of the ACM, 22(4), 1979, pp. 251–256.

[7] Bui, T. N. and T. H. Nguyen, "An Agent-Based Algorithm for Generalized Graph Colorings," Proc. 8th Annual Conference on Genetic and Evolutionary Computation Conference, Seattle, WA, 2006, pp. 19–26.

[8] Bui, T. N. and T. H. Nguyen, Penn State Harrisburg's Dimacs Instances Archive, http://cs.hbg.psu.edu/txn131/. Last access: 09/25/2006.

[9] Bui, T. N., T. H. Nguyen, C. Patel, and K. T. Phan, "An Ant-Based Algorithm for Coloring Graphs," to appear in the Journal of Discrete Applied Mathematics.

[10] Comellas, F. and J. Ozon, "Graph Coloring Algorithms for Assignment Problems in Radio Networks," Proc. International Workshop on Applications of Neural Networks to Telecommunications, Stockholm, Sweden, 1995, pp. 49–56.

[11] Comellas, F. and J. Ozon, "An Ant Algorithm for the Graph Coloring Problem," ANTS'98 – From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization, Brussels, Belgium, 1998.

[12] Costa, D. and A. Hertz, "Ants Can Colour Graphs," Journal of Operational Research Society, 48, 1997, pp. 295–305.

[13] Culberson, J. and F. Luo, "Exploring the k-colorable landscape with Iterated Greedy," Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993, American Mathematical Society, 26, 1996, pp. 245–284.

[14] Dorigo, M. and G. Di Caro, "The Ant Colony Optimization Meta-Heuristic," New Ideas in Optimization, McGraw-Hill, London, England, 1999, pp. 11–32.

[15] Garey, M. R., D. S. Johnson, and H. C. So, "An Application of Graph Coloring to Printed Circuit Testing," IEEE Transactions on Circuits and Systems, 23, 1976, pp. 591–599.

[16] Halldórsson, M. M., "A Still Better Performance Guarantee for Approximate Graph Coloring," Information Processing Letters, 45, 1993, pp. 19–23.

[17] Hertz, A. and D. de Werra, "Using Tabu Search Techniques for Graph Coloring," Computing, 39(4), 1987, pp. 345–351.

[18] Jin, M. H., H. K. Wu, J. T. Horng, and C. H. Tsai, "An Evolutionary Approach to Fixed Channel Assignment Problem with Limited Bandwidth

Constraint," Proc. IEEE International Conference on Communications (ICC 2001), 7, Helsinki, Finland, 2001, pp. 2100–2104.

[19] Johnson, D. S., C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning," Operations Research, 39(3), 1991, pp. 378–406.

[20] Joslin, D. E. and D. P. Clements, "Squeaky Wheel Optimization," Journal of Artificial Intelligence Research, 10, 1999, pp. 353–373.

[21] Karm, R. M., "Reducibility Among Combinatorial Problems," Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, NY, 1972, pp. 85–103.

[22] Leighton, F. T., "A Graph Coloring Algorithm for Large Scheduling Problems," Journal of Research of the National Bureau of Standards, 84(6), 1979, pp. 489–506.

[23] Lewandowski, G., A. Condon, "Experiments with Parallel Graph Coloring Heuristics and Applications of Graph Coloring, Cliques, Coloring and Satisfiability," Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993, American Mathematical Society, 26, 1996, pp. 309–334.

[24] Lim, A., X. Zhang, and Y. Zhu, "A Hybrid Method for the Graph Coloring Problem and Its Generalizations," 5th Metaheuristics International Conference, Kyoto, Japan, 2003.

[25] Lim, A., Y. Zhu, Q. Lou, and B. Rodrigues, "Heuristic Methods for Graph Coloring Problems," Proc. ACM Symposium on Applied Computing, New York, NY, 2005, pp. 933–939.

[26] Morgenstern, C., "Distributed Coloration Neighborhood Search Coloring and Satisfiability," Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993, American Mathematical Society, 26, 1996, pp. 335–357.

[27] Park, E. J., Y. H. Kim, and B. R. Moon. "Genetic Search for Fixed Channel Assignment Problem with Limited Bandwidth," Proc. Genetic and Evolutionary Computation Conference, San Francisco, CA, 2002, pp. 1772–1779.

[28] Prestwich, S. D., "Constrained Bandwidth Multicoloration Neighborhoods," Computational Symposium on Graph Coloring and Generalizations, Cornell University, Ithaca, New York, 2002.

[29] Trick, M. A., COLOR02/03/04: Graph Coloring and its Generalizations, http://mat.gsia.cmu.edu/COLOR04/. Last access: 09/25/2006.

[30] de Werra, D., "An Introduction to Timetabling," European Journal of Operational Research, 19(2), 1985, pp. 151–162.

[31] White, T., B. Pagurek, and F. Oppacher, "ASGA: Improving the Ant System by Integration with Genetic Algorithms," Proc. 3rd Annual Conference on Genetic Programming, Madison, WI, 1998, pp. 610–617.

[32] Zymolka, A., A. Koster, and R. Wessaly, "Transparent Optical Network Design with Sparse Wavelength," Proc. 7th IFIP Working Conference on Optical Network Design and Modelling, Budapest, Hungary, 2003, pp. 61–80.

Appendix A

Machine Benchmark

The DIMACS archive website [8] provides the benchmark utility *dfmax* for the comparison of machines. We report here the results obtained after recompiling dfmax on the machine that runs our algorithm.

Instances	Machine Specs CPU: 3GHz, Ram: 2GB, OS: Linux, Lang: C++			
	Time (Secs)			Best Result
	User	Sys	Real	
r100.5.b	0.00	0.00	0.00	4 57 35 5 61 34 3 62 90
r200.5.b	0.03	0.00	0.00	113 86 147 66 14 134 32 127 161 186 70
r300.5.b	0.25	0.00	0.00	279 222 116 17 39 127 190 158 196 288 263 54
r400.5.b	1.59	0.00	1.00	370 108 27 50 87 275 145 222 355 88 306 335 379
r500.5.b	5.84	0.00	6.00	345 204 148 480 16 336 76 223 260 403 141 382 289

Table A.1: Machine benchmark.

Appendix B

Instance Descriptions

This appendix summarizes the DIMACS instances used to benchmark our algorithm. The name of the instances, their categories (CAT), vertices (V), and edges (E) are shown in the tables below. For each instance, the tables also display the best-known bound on its chromatic number (χ^*), if known, otherwise '–' is displayed. Note that the instances used to test the generalized graph coloring problems have no known bound on their chromatic numbers. Brief descriptions of the categories are also given below.

DSJ: (From David Johnson) Random graphs used in his paper with Aragon, McGeoch, and Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning", Operations Research, 31, 378–406 (1991). DSJC are standard (n, p) random graphs. DSJR are geometric graphs, with DSJR..c being complements of geometric graphs.

CUL: (From Joe Culberson) Quasi-random coloring problem.

REG: (From Gary Lewandowski) Problem based on register allocation for variables in real codes.

LEI: (From Craig Morgenstern) Leighton graphs with guaranteed coloring size. A reference is F.T. Leighton, Journal of Research of the National Bureau of Standards, 84: 489–505 (1979).

Instances	CAT	V	E	χ^*	Instances	CAT	V	E	χ^*
1-FullIns_3	CAR	30	100	–	le450_15d	LEI	450	16750	15
1-FullIns_4	CAR	93	593	–	le450_25a	LEI	450	8260	25
1-FullIns_5	CAR	282	3247	–	le450_25b	LEI	450	8263	25
1-Insertions_4	CAR	67	232	4	le450_25c	LEI	450	17343	25
1-Insertions_5	CAR	202	1227	–	le450_25d	LEI	450	17425	25
1-Insertions_6	CAR	607	6337	–	le450_5a	LEI	450	5714	5
2-FullIns_3	CAR	52	201	–	le450_5b	LEI	450	5734	5
2-FullIns_4	CAR	212	1621	–	le450_5c	LEI	450	9803	5
2-FullIns_5	CAR	852	12201	–	le450_5d	LEI	450	9757	5
2-Insertions_3	CAR	37	72	4	miles1000	SGB	128	3216	42
2-Insertions_4	CAR	149	541	4	miles1500	SGB	128	5198	73
2-Insertions_5	CAR	597	3936	–	miles250	SGB	128	387	8
3-FullIns_3	CAR	80	346	–	miles500	SGB	128	1170	20
3-FullIns_4	CAR	405	3524	–	miles750	SGB	128	2113	31
3-FullIns_5	CAR	2030	33751	–	mug100_1	MIZ	100	166	4
3-Insertions_3	CAR	56	110	4	mug100_25	MIZ	100	166	4
3-Insertions_4	CAR	281	1046	–	mug88_1	MIZ	88	146	4
3-Insertions_5	CAR	1406	9695	–	mug88_25	MIZ	88	146	4
4-FullIns_3	CAR	114	541	–	mulsol.i.1	REG	197	3925	49
4-FullIns_4	CAR	690	6650	–	mulsol.i.2	REG	188	3885	31
4-FullIns_5	CAR	4146	77305	–	mulsol.i.3	REG	184	3916	31
4-Insertions_3	CAR	79	156	3	mulsol.i.4	REG	185	3946	31
4-Insertions_4	CAR	475	1795	–	mulsol.i.5	REG	186	3973	31
5-FullIns_3	CAR	154	792	–	myciel3	MYC	11	20	4
5-FullIns_4	CAR	1085	11395	–	myciel4	MYC	23	71	5
abb313GPIA	HOS	1557	53356	–	myciel5	MYC	47	236	6
anna	SGB	138	493	11	myciel6	MYC	95	755	7
ash331GPIA	HOS	662	4181	–	myciel7	MYC	191	2360	8
ash608GPIA	HOS	1216	7844	–	qg.order30	GOM	900	26100	30
ash958GPIA	HOS	1916	12506	–	qg.order40	GOM	1600	62400	40
david	SGB	87	406	11	qg.order60	GOM	3600	212400	60

Table B.1: Summary of the 119 DIMACS graphs for GCP.

SCH: (From Gary Lewandowski) Class scheduling graphs, with and without study halls.

LAT: (From Gary Lewandowski) Latin square problem.

SGB: (From Michael Trick) Graphs from Donald Knuth's Stanford GraphBase. These can be divided into:

Book Graphs: Given a work of literature, a graph is created where each node represents a character. Two nodes are connected by an edge if the

Instances	CAT	V	E	χ^*	Instances	CAT	V	E	χ^*
DSJC1000.1	DSJ	1000	49629	–	qg.order100	GOM	10000	990000	100
DSJC1000.5	DSJ	1000	249826	–	queen10_10	SGB	100	1470	–
DSJC1000.9	DSJ	1000	449449	–	queen11_11	SGB	121	1980	11
DSJC125.1	DSJ	125	736	–	queen12_12	SGB	144	2596	–
DSJC125.5	DSJ	125	3891	–	queen13_13	SGB	169	3328	13
DSJC125.9	DSJ	125	6961	–	queen14_14	SGB	196	4186	–
DSJC250.1	DSJ	250	3218	–	queen15_15	SGB	225	5180	–
DSJC250.5	DSJ	250	15668	–	queen16_16	SGB	256	6320	–
DSJC250.9	DSJ	250	27897	–	queen5_5	SGB	25	160	5
DSJC500.1	DSJ	500	12458	–	queen6_6	SGB	36	290	7
DSJC500.5	DSJ	500	62624	–	queen7_7	SGB	49	476	7
DSJC500.9	DSJ	500	112437	–	queen8_12	SGB	96	1368	12
DSJR500.1	DSJ	500	3555	–	queen8_8	SGB	64	728	9
DSJR500.1c	DSJ	500	121275	–	queen9_9	SGB	81	1056	10
DSJR500.5	DSJ	500	58862	–	school1_nsh	SCH	352	14612	–
fpsol2.i.1	REG	496	11654	65	school1	SCH	385	19095	–
fpsol2.i.2	REG	451	8691	30	wap01a	KOS	2368	110871	–
fpsol2.i.3	REG	425	8688	30	wap02a	KOS	2464	111742	–
games120	SGB	120	638	9	wap03a	KOS	4730	286722	–
homer	SGB	561	1628	13	wap04a	KOS	5231	294902	–
huck	SGB	74	301	11	wap05a	KOS	905	43081	–
inithx.i.1	REG	864	18707	54	wap06a	KOS	947	43571	–
inithx.i.2	REG	645	13979	31	wap07a	KOS	1809	103368	–
inithx.i.3	REG	621	13969	31	wap08a	KOS	1870	104176	–
jean	SGB	80	254	10	will199GPIA	KOS	701	6772	–
latin_square_10	LAT	900	307350	–	zeroin.i.1	REG	211	4100	49
le450_15a	LEI	450	8168	15	zeroin.i.2	REG	211	3541	30
le450_15b	LEI	450	8169	15	zeroin.i.3	REG	206	3540	30
le450_15c	LEI	450	16680	15					

Table B.2: Summary of the 119 DIMACS graphs for GCP.

corresponding characters encounter each other in the book. Knuth creates the graphs for five classic works: Tolstoy's Anna Karenina (anna), Dicken's David Copperfield (david), Homer's Iliad (homer), Twain's Huckleberry Finn (huck), and Hugo's Les Misérables (jean).

Game Graphs: A graph representing the games played in a college football season can be represented by a graph where the nodes represent each college team. Two teams are connected by an edge if they played each other during the season. Knuth gives the graph for the 1990 college football season.

Miles Graphs: These graphs are similar to geometric graphs in that nodes

Instances	CAT	V	E	Instances	CAT	V	E
geom20	GEO	20	40	geom80	GEO	80	429
geom20a	GEO	20	57	geom80a	GEO	80	692
geom20b	GEO	20	52	geom80b	GEO	80	743
geom30	GEO	30	80	geom90	GEO	90	531
geom30a	GEO	30	111	geom90a	GEO	90	879
geom30b	GEO	30	111	geom90b	GEO	90	950
geom40	GEO	40	118	geom100	GEO	100	647
geom40a	GEO	40	186	geom100a	GEO	100	1092
geom40b	GEO	40	197	geom100b	GEO	100	1150
geom50	GEO	50	177	geom110	GEO	110	748
geom50a	GEO	50	288	geom110a	GEO	110	1317
geom50b	GEO	50	299	geom110b	GEO	110	1366
geom60	GEO	60	245	geom120	GEO	120	893
geom60a	GEO	60	339	geom120a	GEO	120	1554
geom60b	GEO	60	426	geom120b	GEO	120	1611
geom70	GEO	70	337				
geom70a	GEO	70	529				
geom70b	GEO	70	558				

Table B.3: Summary of the 33 GEOM DIMACS graphs for the graph coloring generalizations.

are placed in space with two nodes connected if they are close enough. These graphs, however, are not random. The nodes represent a set of United States cities and the distance between them is given by road mileage from 1947. These graphs are also due to Knuth.

Queen Graphs: Given an n by n chessboard, a queen graph is a graph on n^2 nodes, each corresponding to a square of the board. Two nodes are connected by an edge if the corresponding squares are in the same row, column, or diagonal. Unlike some of the other graphs, the coloring problem on this graph has a natural interpretation: given such a chessboard, is it possible to place n sets of n queens on the board so that no two queens of the same set are in the same row, column, or diagonal? The answer is yes if and only if the graph has coloring number n.

MYC: (From Michael Trick) Graphs based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.

MYC: (From Kuzunori Mizuno) Graphs that are almost 3-colorable, but have a

hard-to-find four clique embedded.

HOS: (From Shahadat Hossain) Graphs obtained from a matrix partitioning problem in the segmented columns approach to determine sparse Jacobian matrices.

CAR: (From M. Caramia and P. Dell'Olmo) k-Insertion graphs and Full Insertion graphs are a generalization of Myciel graphs with inserted nodes to increase graph size but not density.

KOS: (From Arie Koster) From real-life optical network design problems. Each vertex corresponds to a lightpath in the network; edges correspond to intersecting paths.

GOM: (From Carla Gomes) Latin squares (standard encoding).

GEO: (From Michael Trick) Points are generated in a 10,000 by 10,000 grid and are connected by an edge if they are close enough together. Edge weights are inversely proportional to the distance between nodes. Node weights are uniformly generated. The GEOMn instances are sparse, while GEOMa and GEOMb instances are denser. GEOMb requires fewer colors per node.

Appendix C

Parameter Settings

Important parameters used in the algorithm are described in this section. We obtained these parameters by testing the algorithm on a few graphs such as circles, lines, trees, caterpillars, and grids. These parameters were not tuned for any particular classes of graphs. The objective is to balance performance and running time. We assume that $n = |V|$ is the cardinality of the vertex set.

nAgents is the number of agents in the group and was set to 20% of the number of vertices in the graph. For efficiency reason we do not allow *nAgents* to exceed 100.

nCycles is the number of cycles in the entire coloring process and was set to be $\min\{6n, 4000\}$.

nMoves is the number of vertices an agent can visit before it stops. This value is defined as follows:

$$nMoves = \begin{cases} n/4, & \text{if } nAgents < 100 \\ 20 + \frac{n}{nAgents}, & \text{otherwise} \end{cases}$$

nJoltCycles is the number of cycles during which the value of *attemptK* has not improved, before a jolt is applied to the coloring creating a perturbation of the current coloring configuration. This value was set to $\max\{n/2, 600\}$.

nBreakCycles is the number of cycles during which the value of *attemptK* has not improved before the algorithm is terminated. This value was set to $\max\{5n/2, 1600\}$.

α is the percentage of the colors from initial coloring that is made available for the agents to use begin with. This value was set to 80%.

β is the top conflicted percentage of the coloring to be shuffled in the perturbation operation. This value was set to 10%.

γ is the percentage of the current set of available colors that the perturbation operation uses for re-coloring. This value was set to 80%.

δ is the length of the tabu list of recently visited vertices. An agent avoids revisiting those vertices in its tabu list to allow a more diverse exploration of the graph. This value was set to $nMoves/3$.

λ is the maximum size of the vertex set selected to be assigned with a color number in MXRLF. This value was set to $0.7n$.