An Agent-Based Algorithm for Generalized Graph Colorings

Thang N. Bui and ThanhVu H. Nguyen
Computer Science Program
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
{tbui, txn131}@psu.edu

ABSTRACT

This paper presents an algorithm for solving a number of generalized graph coloring problems. Specifically, it gives an agent-based algorithm for the Bandwidth Coloring problem. Using a standard method for preprocessing the input, the same algorithm can also be used to solve the Multicoloring and Bandwidth Multicoloring problems. In the algorithm a number of agents, called *ants*, each of which colors a portion of the graph, collaborate to obtain a coloring of the entire graph. This coloring is then further improved by a local optimization algorithm. Experimental results on a set of benchmark graphs for these generalized coloring problems show that this algorithm performs very well compared to other heuristic approaches.

Categories and Subject Descriptors: G.2.2[Discrete Mathematics]:Graph Theory – Graph algorithms; I.2.8[Artificial Intelligence]:Problem Solving, Control Methods, and Search – Heuristic methods

General Terms: Design, Algorithms

Keywords: Graph Coloring, Bandwidth Coloring, Multicoloring, Bandwidth Multicoloring

1. INTRODUCTION

The *graph coloring problem* (GCP) is the 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. This problem is well known to be NP-hard. Many heuristic approaches have been proposed for it including constructive methods [5][22], iterative methods [12], genetic algorithms [30], local search methods [26], tabu search methods [17], and ant system algorithms [6][8][9][11][30].

Generalizations of the graph coloring problem include the Bandwidth Coloring, Multicoloring, and Bandwidth Multicoloring problems. They have more constraints placed on the vertices and/or edges of the graph. These generalizations model a collection of useful applications. For instance, the Bandwidth Coloring and Multicoloring problems have been used to model the fixed channel assignment problem, which is to assign frequencies to different cells in a mobile cellular network such that certain frequency separations are satisfied while minimizing the amount of interferences. Furthermore, if each cell must use a number of separate frequencies, then the problem can be modeled by the Bandwidth MultiColoring problem [27].

As with GCP, these generalizations are also NP-hard. There are several heuristic algorithms for these generalizations including Prestwich's local search and constraint propagation method [29] and Lim et al's hybrid combinations of Squeaky Wheel Optimization (SWO) and Hill Climbing techniques [21, 23]. Recently, Lim et al proposed another approach for these problems by combining SWO and Tabu Search techniques [24].

In this paper, we propose an agent-based optimization algorithm for these generalized graph coloring problems. We call the agents in our algorithm ants as the agents in our algorithm mimic the collective ability of an ant colony to solve problems. It should be noted that our algorithm is not an ant colony optimization (ACO) algorithm as described in [13]. Among other things, ants in our algorithm do not use pheromone to communicate. Experimental results on a set of 33 benchmark graphs, plus additional constraints for three different generalized coloring problems for a total of 99 problem instances, show that our algorithm produces results that are comparable to those of other algorithms.

The rest of the paper is organized as follows. In Section 2 we give formal definitions for the Graph Coloring problem and its generalizations, and briefly summarize the complexity of finding approximate solutions. We describe our algorithm in Section 3 and present the experimental results in Section 4. The conclusion is given in Section 5.

2. PRELIMINARIES

In this section we describe the Graph Coloring problem and its variations that are considered in this paper. We also give a brief summary of the approximation complexity for the Graph Coloring problem.

The Graph Coloring Problem (GCP): This is the problem of finding an assignment of colors to the vertices of a graph, using a minimum number of colors, such that each vertex has a color and no two adjacent vertices have the same color.

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

Output: A minimum k and a mapping $f : V \longrightarrow \{1, \ldots, k\}$ such that $\forall (u, v) \in E, f(u) \neq f(v)$.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

The Bandwidth Coloring Problem (BCP): This 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 $f : V \longrightarrow \{1, \ldots, k\}$ such that $\forall(u, v) \in E, |f(u) - f(v)| \geq d(u, v)$.

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

The Multicoloring 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 the 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 disjoint.

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

Output: A minimum k and $\forall u \in V$, subsets $S(u) \subset \{1, \ldots, k\}$ such that[1]

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

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

The Bandwidth Multi-Coloring Problem (BMCP): This generalization of the GCP has the constraints of both BCP and MCP.

Input: An undirected graph $G = (V, E)$ with positive integer vertex 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 $\forall u \in V$, subsets $S(u) \subset \{1, \ldots, k\}$ such that

- $\forall u \in V, |S(u)| = w(u)$,
- $\forall(u, v) \in E, S(u) \cap S(v) = \emptyset$, and
- $\forall(u, v) \in E, \forall p \in S(u), q \in S(v), |p - q| \geq d(u, v)$.

We note that input graphs for BMCP may contain self-loops. For example, $d(u, u) = 3$ means that any two colors in the set of colors assigned to vertex u must differ by at least 3. It is clear that BMCP is GCP if $d(u, v) = 1, \forall(u, v) \in E$ and $w(u) = 1, \forall u \in V$.

As mentioned in the previous section, GCP, BCP, MCP, and BMCP are all NP-hard. Thus, we do not expect to have exact polynomial time algorithms for these problems. There is no known approximation algorithm for BCP, MCP or BMCP. In fact, not much is known about the approximation complexity of the generalized graph coloring problems. However, for the graph coloring problem (GCP) it is known that approximating the chromatic number[2] within $O\left(|V|(\log\log|V|)^2/(\log|V|)^3\right)$ can be done in polynomial time [16]. It is also known that the chromatic number cannot be approximated within $|V|^{1/7-\epsilon}$, for any $\epsilon > 0$ unless $P = NP$ [2]. Thus, approximations for the generalized graph coloring problems are at least as difficult.

[1] $|X|$ denotes the cardinality of set X.

[2] The *chromatic number* of a graph is the minimum number of colors needed to color the graph in the GCP.

3. ALGORITHM

In this section we describe our agent-based algorithm for the generalized graph coloring problems described in the previous section. This algorithm is based on and extends the algorithm in [6] for the classical graph coloring problem (GCP). In what follows we use the term *conflict at a vertex* v to denote the number of vertices adjacent to v having color or color sets that are inconsistent with the coloring of v.

The main idea of the algorithm is for a set of agents, called ants, to color the graph. The ants are distributed on the vertices of input graph based on the conflicts at the vertices. Each ant will color a portion of the graph. These local colorings by the ants together form a coloring for the graph. This is different from the traditional Ant Colony Optimization algorithms, where each ant finds a complete solution to the problem [13]. No ant in our algorithm solves the entire problem by itself. Our algorithm is thus more amenable to a distributed implementation. However, in this paper we do not present such an implementation. Another difference in our approach is that the ants do not have pheromone laying capability[3]. In limited experiments we found that with pheromone the algorithm took longer to converge without providing visible or significant improvement on the quality of the solution.

3.1 Overview

Let $\langle G = (V, E), w, d\rangle$ be an instance of the generalized graph coloring problems, where G is a graph with vertex set V, edge set E, w the vertex weight function, and d the edge weight function. We first find an initial coloring of G using a simple greedy algorithm, called IterativeGreedy. We then use a colony of ants to improve on the initial coloring. The algorithm proceeds in a number of cycles. In each cycle, ants are distributed to vertices of the graph based on the conflicts at the vertices. Vertices with higher conflict will be selected first. Each of the ants attempts to color the portion of the graph close to where it is at, using only the set of currently available colors. At the end of a cycle, if there are no conflicts in the current coloring then a local optimization algorithm will attempt to improve the current coloring. If the local optimization algorithm is successful, the number of colors needed will be reduced. This set of colors becomes the current set of available colors. If the local optimization algorithm is not successful, there is no change to the current coloring. Once the local optimization algorithm is finished, the number of colors in the current set of available colors is reduced by one. This is done by deleting the highest number color from the set of available colors. Vertices that were colored with the deleted color are then recolored with another color randomly selected from the reduced set of available colors, and we start another cycle. Note that this recoloring may create a coloring with nonzero conflict.

If the number of conflicts is non-zero for a number of consecutive cycles then the algorithm increases the number of

[3] However, how an ant colors is based partly on the colorings done by previous ants.

```
Algorithm ABGC(G = (V, E), w, d)
preprocess G        // needed only for MCP and BMCP
currentColoring ⟵ IterativeGreedy(G, d)
maxK ⟵ number of colors in currentColoring
attemptK ⟵ αmaxK       //attempt new goal, α < 1
keep random βmaxK color classes    //rename colors as needed
distribute the remaining vertices into γmaxK color classes
update totalConflict cost of G
for cycle = 1 to nCycles do
  for ant = 1 to nAnts do
    if there is no conflict, break
    ant clears its recentlyVisited tabu list
    ant is placed on a vertex having maximum conflict
    ant colors its current vertex
    for move = 1 to nMoves do
      ant moves to a neighboring vertex by taking two steps
      ant colors its current vertex
      ant updates local conflict(s) in its neighborhood
      ant updates its recentlyVisited tabu list
    end-for
  end-for
  update totalConflict cost of G
  if totalConflict = 0 // run local optimization
    maxK ⟵ number of colors in currentColoring
    lColoring ⟵ localOpt(currentColoring)
    lMaxK ⟵ number of colors in lColoring
    if lMaxK < maxK
      maxK ⟵ lMaxK
      currentColoring ⟵ lColoring
    end-if
    bestColoring ⟵ currentColoring
    attemptK ⟵ maxK − 1
    Recolor vertices that have colors greater than attemptK
    update totalConflict cost of G
  end-if
  if attemptK has not improved in nChangeCycles cycles
    attemptK ⟵ attemptK + 1
        (precond: attemptK < maxK)
  if attemptK has not improved in nJoltCycles cycles
    perform a Jolt operation
  if attemptK has not improved in nBreakCycles cycles
    break
end-for
return bestColoring and maxK
```

Figure 1: An agent-based algorithm for generalized graph coloring problems (ABGC)

available colors by one before starting another cycle. Other actions might also be taken by the algorithm to bring it out of a potential local optimum before it starts another cycle. Stopping conditions are described in full below. At the end of the algorithm the best coloring found is returned. The complete algorithm is given in Figure 1. In the following subsections we describe the various parts of the algorithm in details.

3.2 Input Preprocessing

The algorithm described above works for all three generalized coloring problems (Bandwidth Coloring, Multicoloring, and Bandwidth Multicoloring) with only a simple preprocessing of the input graph. Specifically, no preprocessing is needed for the Bandwidth Coloring problem. For the Multicoloring and Bandwidth Multicoloring problems where each vertex has a positive integer weight, we apply a transformation that is used in [24] to the input graph. A vertex a of weight $w(a)$ is transformed into a clique of size $w(a)$, with each edge in the clique having the weight of the edge (a, a), i.e., $d(a, a)$. In addition, if (a, x) is an edge in the original graph, then we also connect each vertex in the clique to x and assign such an edge the weight $d(a, x)$. We note that this transformation takes exponential time if vertex weight $w(a)$ is exponential in the number of vertices in the graph. However, this approach is reasonable in practice as the weights in real world applications are much less than the number of vertices in the graph. For example, the number of different frequencies assigned to a cell phone is much less than the number of cell phones.

3.3 Iterative Greedy algorithm

```
Algorithm IG(G = (V, E), d)
for i = 1 to |V| do    //initialize the coloring
  C[i] ⟵ 0
end-for
for i = 1 to |V| do
  forbiddenSet ⟵ ∅
  u ⟵ an uncolored vertex selected at random or
     based on max degree
  for each vertex v that is adjacent to u do
    if C[v] ≠ 0 then
      L ⟵ max(1, c[v] − d(u, v) + 1)
      U ⟵ c[v] + d(u, v) − 1
      forbiddenSet ⟵ forbiddenSet ∪ [L . . . U]
    end-if
  end-for
  C[u] ⟵ the color selected based on the forbiddenSet
          and the rules described in Section 3.5
end-for
return C
```

Figure 2: Iterative Greedy algorithm

After preprocessing the input graph as needed, ABGC begins by using a greedy algorithm on the graph to obtain an initial coloring that is valid, but not necessarily optimal or even good. This greedy algorithm is called IterativeGreedy (IG) and is given in Figure 2. 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., among the uncolored vertices, the vertex of highest degree is selected first. When a vertex u is selected to be colored, IG uses the same scheme as in ABGC. This coloring scheme is described in Section 3.5.

ABGC uses IG to find an initial coloring as follows. It first runs IG using the highest degree selection scheme. It then runs IG twenty times using the random selection scheme. The best coloring found among the twenty-one runs is then used as the initial coloring for ABCG.

3.4 How An Ant Moves

In each cycle after an ant is placed at a vertex, it attempts to color the neighborhood of that vertex. To do this, the ant takes two steps, i.e., traverses along a path of length two, then colors the vertex that it lands on. The process of taking a number of steps then coloring the vertex is called a *move*. Within a cycle, each ant makes *nMoves* moves.

In the first step of an ant's move, the ant randomly selects an adjacent vertex and moves there. In the second step, the ant selects an adjacent vertex that has 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 ant is relocated to the vertex that has the highest conflict in the entire graph. Again, ties are broken arbitrarily. If there is no conflict left all ants will stop moving and the algorithm prepares for the next cycle by reducing the number of available colors by one.

Additionally, whenever an ant selects a vertex to move to, it always avoids vertices that are in its current tabu list. It also places vertices that it has just visited on its current tabu list. The tabu list is of fixed length. Thus, when the list is full, older vertices on the list will be removed as new vertices are added to the list. The reason for having the random first step in a move and the tabu list is to allow more exploration of the search space and to help the algorithm escape from local optima. With an increase in running time, we can gain some improvement in the algorithm by allowing each ant to take more than two steps per move. The above algorithm can be easily extended to accommodate this option.

3.5 How A Vertex Is Colored

Within each cycle, an ant colors only a limited local area of the graph without any global knowledge of the graph and uses only colors from the set of available colors (in the algorithm of Figure 1, the number of available colors is *attemptK*). The ant's objective is to color or recolor a vertex so that the conflict at that vertex is zero, if possible.

To color a vertex u an ant must first determine the set of eligible colors that can be used to color u. This is done as follows. First, for each vertex v, adjacent to u, a set of colors that is in conflict with the coloring of v is determined. This set of conflicting colors is computed by examining the color of v and the weight of the edge (u, v). The union of the conflicting color sets of the vertices adjacent to u is called the set of forbidden colors. These are the colors that cannot be used to color u. This is the same process as that of computing the *forbiddenSet* in the IG algorithm of Figure 2.

Once the set of forbidden colors has been computed, the set of eligible colors can be easily determined by taking the difference between the set of all colors, i.e., $\{1, \ldots, attemptK\}$, and the set of forbidden colors.

If the cardinality of the set of eligible colors is greater than 1, then it can be viewed as a union of intervals. The ant then chooses the color that is the median of the largest interval in the set of eligible colors. If there are more than one interval of largest size, one of those intervals is chosen at random. Furthermore, if there are two medians then one of them will be 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 $[1 \ldots 5] \cup [8 \ldots 9] \cup [11 \ldots 15]$. Thus, one of the intervals $[1 \ldots 5]$ or $[11 \ldots 15]$ is selected at random, say the latter. Then the selected color is 13, the median of $[11 \ldots 15]$. This selection scheme allows later vertices more room to meet their constraints, i.e., the set of eligible colors will be larger for neighboring vertices.

If the set of eligible colors is empty, i.e., each of the available color is in conflict with the coloring of one or more adjacent vertices, the ant then chooses the color that conflicts with the fewest adjacent vertices. Ties are broken arbitrarily.

After the ant colors, the vertex conflict is updated and it is added to the tabu list of the ant. Note that this vertex will replace the oldest one in case the tabu list is full.

3.6 Local Optimization

Algorithm LocalOpt$(G = (V, E), d, C)$
// C is a valid coloring of G
using the input coloring C sort V
    into decreasing order of color numbers
for $i = 1$ to $|V|$ **do** //erase the input coloring
  $C[i] \longleftarrow 0$
end-for
for $i = 1$ to $|V|$ **do**
  $forbiddenSet \longleftarrow \emptyset$
  $u \longleftarrow$ the next uncolored vertex in the sorted set V
  for each vertex v that is adjacent to u **do**
    if $C[v] \neq 0$ **then**
      $L \longleftarrow \max(1, c[v] - d(u, v) + 1)$
      $U \longleftarrow c[v] + d(u, v) - 1$
      $forbiddenSet \longleftarrow forbiddenSet \cup [L \ldots U]$
    end-if
  end-for
  $C[u] \longleftarrow$ the smallest color not in the *forbiddenSet*
end-for
return C

Figure 3: The local optimization algorithm

The local optimization operation is applied every time a valid coloring is found in ABGC. This optimization algorithm is similar to the IterativeGreedy algorithm in Figure 2. The local optimization algorithm first sorts the vertex set into decreasing order of vertex color, i.e., the color given in the input coloring. The algorithm then 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 of Figure 2. The vertex is then colored using the smallest color number that is not in the forbidden set. When the algorithm terminates it returns the coloring that it found.

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. It should be noted that colorings obtained by just running the IterativeGreedy algorithm followed by the local optimization algorithm alone are never as good as those obtained by ABGC. In other words, operations performed by the ants are essential in finding good colorings.

3.7 Jolt and Stopping Criteria

At the end of a cycle, the total conflict of the current coloring is computed. If the total conflicts is zero, the number of available colors *attemptK* is reduced by 1. Next all vertices with color number greater than *attemptK* are re-colored by assigning each of them a randomly selected color in the interval $[1 \ldots attemptK]$. 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.

Note that although we did not explicitly use pheromone as memory device but how the ants color in cycle $i+1$ is based on the coloring of cycle i. In other words, the coloring done in cycle $i+1$ is built upon the result from cycle i.

It is common that search algorithm such as this agent-based algorithm would get stuck at local optima. To alleviate this problem, we add a procedure, called a *jolt*, for perturbing the current coloring, effectively pushing it out of local optima when needed. More specifically, if the ants have not been able to reduce the number of colors used for the last *nJoltCycles* consecutive cycles, then the jolt operation is performed. Vertices that have conflicts in the top 10% are selected and their neighbors are randomly re-colored using 80% of the current set of available colors. The objective of the jolt operation is to create enough perturbation in the current coloring to push it out of a local optimum, but not enough to completely randomize the coloring that has been built up to that point.

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

3.8 Parameters

In this section we give a description for each parameter used in the algorithm. These parameters were obtained by testing the algorithm on a few graphs. These parameters were not tuned for any particular classes of graphs. The objective is to balance between the performance of ABGC and its running time. We assume that $n = |V|$ is the cardinality of the vertex set.

nAnts is the number of ants in a colony and was set to $2n$. For running time consideration *nAnts* was set to $\max(2n, 150)$.

nCycles is the maximum number of cycles in the algorithm and was set to be $\min(6n, 4000)$.

nMoves is the number of moves each ant makes in a cycle and was set to $n/4$.

rSizeLimit is the length of the tabu list of recently colored vertices and is set to be *nMoves*/3. An ant will avoid revisiting vertices in its tabu list, allowing a more diverse exploration of the graph.

nChangeCycle is the number of consecutive cycles allowed in which there is no improvement before the number of available colors, *attemptK*, is increased. This parameter was set to 5.

nJoltCycles is the number of consecutive cycles during which the value of *attemptK* has not improved before the jolt operation is applied. This value was set to $\max(n/2, 150)$.

nBreakCycles is the number of consecutive cycles during which *attemptK* has not improved before the algorithm is terminated . This value was set to be *nCycles*/5, if it is > 500, it is set to 500, if it is < 50, it is set to 50.

4. EXPERIMENTAL RESULTS

In this section we present the results of our algorithm on the 33 GEOM graphs from [10]. For each graph, the BCP requires an edge weight function, whereas the MCP requires a vertex weight function and the BMCP requires both. Thus, there are 99 problem instances in total. The algorithm was implemented in C++ and run on a 3.0GHz Pentium 4 PC with 2GB of RAM running the Linux operating system[4]. For each problem instance we ran our algorithm for 100 trials. Detailed information are listed in Tables 2, 3, and 4. For each instance, we list the best (Min), worst (Max), average (Avg), and standard deviation (SD), of the colorings found in 100 runs. The average running time, in seconds, is also listed.

Tables 5 and 6 give a comparison of the results of our algorithm against those of the following algorithms.

- **Prestwich**: Prestwich's local search and constraint propagation algorithm [29].
- **Lim**: Lim et al.'s Squeaky Wheel and Tabu Search [23, 24][5].
- **ABGC**: Our agent-based algorithm.

The columns list the best results given by each of the three algorithms. The last column shows the percentage difference between our algorithm and the better of the other two. A positive percentage means that our algorithm is better.

For the BCP problem, our results matched or surpassed all results from [23, 24] but are not as good as those of [29]. As pointed out in [29], their algorithm requires certain parameter to be tuned for each graph or class of graphs. That is not the case for our algorithm, ABGC.

For the Multicoloring problem, we do not give a table comparing the results of our algorithm against those of other algorithms since our results match those of [24] in all 33 cases. Furthermore, no results were provided by [29] for the Multicoloring problem.

For the BMCP problem, our algorithm performs better than the algorithm of [29]. Where results are not available, we mark the corresponding entries with '–'. Comparing against the algorithm of [23, 24], ABGC performs better in most larger graphs while doing worse in a few smaller graphs.

5. CONCLUSION

In this paper we presented an agent-based algorithm for three generalized graph coloring problems. Overall, our algorithm performs very well on a set of benchmark graphs. Agents in our algorithm are called ants, however, they do not have pheromone laying capability, a common component in ant algorithms. We found that in this particular algorithm, simple usage of pheromone to mark individual color increases the algorithm's running time without providing any significant benefit. We are investigating the usage of pheromone to mark coloring patterns as aggregates and not just as individual colors. We expect that pheromone used in this way will retain the constraints in the coloring more accurately, which in turn will help our algorithm's performance.

6. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments.

[4]The *dfmax* benchmark takes 5.91s user time for $r500.5.b$. See [10] for details regarding *dfmax*.

[5]We combine and select the best results from [23] and [24].

Table 1: Summary of the 33 geometric DIMACS graph instances used

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

Table 2: Results of ABGC for BCP

Instance	Min	Max	Avg	SD	Time (sec)
geom20	21	21	21.0	0	0.03
geom20a	20	24	21.61	0.77	0.03
geom20b	13	14	13.11	0.31	0.04
geom30	28	29	28.04	0.2	0.07
geom30a	27	32	29.1	1.3	0.09
geom30b	26	27	26.07	0.26	0.13
geom40	28	29	28.08	0.27	0.11
geom40a	37	42	38.6	0.92	0.18
geom40b	33	38	35.17	1.39	0.25
geom50	28	31	28.17	0.53	0.24
geom50a	50	56	52.08	1.4	0.39
geom50b	36	44	38.92	1.64	0.39
geom60	33	35	33.5	0.54	0.39
geom60a	50	57	52.05	1.21	0.65
geom60b	43	51	46.44	1.58	0.83
geom70	38	42	38.33	0.63	0.66
geom70a	62	71	65.76	2.28	0.84
geom70b	51	58	53.72	1.46	1.02
geom80	41	45	42.09	0.94	0.7
geom80a	64	76	69.01	2.59	1.26
geom80b	64	74	67.62	1.92	1.6
geom90	46	51	46.88	1.05	0.92
geom90a	65	75	69.82	1.88	1.85
geom90b	74	85	78.96	2.28	2.57
geom100	50	60	52.24	1.42	1.23
geom100a	71	81	75.53	1.89	2.75
geom100b	79	93	82.52	2.16	3.59
geom110	50	54	52.14	0.96	1.48
geom110a	75	83	79.02	1.58	3.35
geom110b	83	97	89.05	2.3	4.22
geom120	59	67	61.79	1.46	1.67
geom120a	86	95	90.04	1.75	3.93
geom120b	91	102	96.43	2.15	5.9

7. REFERENCES

[1] Abril, J., F. Comellas, A. Cortes, J. Ozon, and M. Vaquer, "A Multi-Agent System for Frequency Assignment in Cellular Radio Networks," IEEE Trans. on Vehicular Technology, 49(5), September 2000, pp. 1558–1564.

[2] M. Bellare, O. Goldreich and M. Sudan, "Free Bits, PCPs and Non-Approximability - Towards Tight Results", SIAM J. Computing, 27, 1998, pp. 804–915.

[3] Blum A. and D. Karger, "An $O(n^{3/14})-$Coloring Algorithm for 3-Colorable Graphs," Information Processing Letters, 61(1), January 1997, pp. 49–53.

[4] Bonabeau, E., M. Dorigo, and G. Theraulaz, "Inspiration for Optimization from Social Insect Behavior," Nature, 406, July 6, 2000, pp. 39–42.

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

[6] Bui, T. N. and C. Patel, "An Ant system Algorithm for Coloring Graphs," Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002.

[7] Chiarandini, M. and T. Stützle, "An Application of Iterated Local Search to Graph Coloring Problem," Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002.

[8] Comellas, F. and J. Ozon, "Graph Coloring Algorithms for Assignment Problems in Radio Networks," Applications of Neural Networks to Telecommunications 2, 1995, pp. 49–56.

[9] 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, October 15–16, 1998.

[10] Computational Series: Graph Coloring and Its Generalizations. "http://mat.gsia.cmu.edu/COLOR04".

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

[12] 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.

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

[14] Dorigo, M. and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," IEEE Trans. on Evolutionary Computation, 1(1), 1997, pp. 53–66.

Table 3: Results of ABGC for MCP

Instance	Min	Max	Avg	SD	Time (sec)
geom20	28	28	28	0	3.33
geom20a	30	30	30	0	3.49
geom20b	8	8	8	0	0.13
geom30	26	26	26	0	3.27
geom30a	40	40	40	0	8.65
geom30b	11	11	11	0	0.42
geom40	31	31	31	0	8.46
geom40a	46	46	46	0	15.51
geom40b	14	14	14	0	2.4
geom50	35	35	35	0	14.63
geom50a	61	61	61	0	43.05
geom50b	17	17	17	0	4.09
geom60	36	36	36	0	16.59
geom60a	65	65	65	0	53.3
geom60b	22	22	22	0	3.31
geom70	44	44	44	0	32.56
geom70a	71	71	71	0	62.29
geom70b	22	22	22	0	3.63
geom80	63	63	63	0	50.61
geom80a	68	68	68	0	75.08
geom80b	25	25	25	0	4.64
geom90	51	51	51	0	67.68
geom90a	65	65	65	0	73.99
geom90b	28	28	28	0	7.37
geom100	60	60	60	0	88.28
geom100a	81	81	81	0	127.87
geom100b	30	30	30	0	7.99
geom110	62	62	62	0	111.81
geom110a	91	91	91	0	198.22
geom110b	37	37	37	0	14
geom120	64	64	64	0	136.94
geom120a	93	93	93	0	308.78
geom120b	34	34	34	0	13.63

Table 4: Results of ABGC for BMCP

Instance	Min	Max	Avg	SD	Time (sec)
geom20	149	158	150.86	2.17	6.85
geom20a	169	176	170.78	1.55	11.27
geom20b	44	46	44.38	0.56	0.24
geom30	160	169	160.99	1.44	9.49
geom30a	210	225	214.94	3.04	25.39
geom30b	77	79	77.59	0.53	1.24
geom40	167	176	167.65	1.24	24.57
geom40a	214	226	216.37	1.91	66.72
geom40b	74	87	77.53	2.35	3.04
geom50	224	232	225.39	1.55	57.48
geom50a	317	336	325.68	3.48	379.48
geom50b	85	99	89.22	2.06	4.54
geom60	258	264	259.15	1.28	64.39
geom60a	357	369	363.47	2.42	203.23
geom60b	117	140	125.59	4.81	10.64
geom70	267	278	271.77	1.76	110.96
geom70a	470	488	478.27	3.49	276.63
geom70b	121	131	125.61	2.02	12.46
geom80	382	393	387.76	2.31	157.88
geom80a	367	382	372.92	3.22	239.61
geom80b	139	147	142.43	1.56	18.01
geom90	332	339	335.6	1.78	180.91
geom90a	378	417	388.28	8.91	387.54
geom90b	150	164	155.96	2.6	22.5
geom100	405	416	409.1	2.45	292.1
geom100a	440	461	449.46	4.27	548.34
geom100b	164	178	171.26	2.7	27.85
geom110	378	391	384.47	2.76	405.16
geom110a	487	502	493.61	3.24	1069.85
geom110b	208	228	213.25	3.48	43.86
geom120	398	408	401.84	2.36	790.21
geom120a	548	565	556.44	3.67	1660.62
geom120b	198	209	203.49	2.62	41.26

[15] Fleurent, C. and J. Ferland, "Genetic and Hybrid Algorithms for Graph Coloring," Annals of Operations Research, 63, 1996, pp. 437–461.

[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. Werra, "Using Tabu Search Techniques for Graph Coloring," Computing, 39, 1987, pp. 345–351.

[18] Jin, M., H. Wu, J. Horng, and C. Tsai, "An Evolutionary Approach to Fixed Channel Assignment Problems with Limited Bandwidth," Proc. of IEEE International Conference on Communications, 7, 2001, pp. 2100–2104.

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

[20] Johnson, D. S. and M. A. Trick, Editors, *Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993,* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 26, 1996.

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

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

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

Table 5: Comparison of ABGC against other algorithms for BCP

Instance	Prestwich	Lim	ABGC	Improvement (%)
geom20	21	21	21	0.000
geom20a	20	22	20	0.000
geom20b	13	14	13	0.000
geom30	28	29	28	0.000
geom30a	27	32	27	0.000
geom30b	26	26	26	0.000
geom40	28	28	28	0.000
geom40a	37	38	37	0.000
geom40b	33	34	33	0.000
geom50	28	28	28	0.000
geom50a	50	52	50	0.000
geom50b	**35**	38	36	−2.857
geom60	33	34	33	0.000
geom60a	50	53	50	0.000
geom60b	43	46	43	0.000
geom70	38	38	38	0.000
geom70a	62	63	62	0.000
geom70b	**48**	54	51	−6.250
geom80	41	42	41	0.000
geom80a	63	66	64	−1.587
geom80b	**61**	65	64	−4.918
geom90	46	46	46	0.000
geom90a	**64**	69	65	−1.562
geom90b	**72**	77	74	−2.778
geom100	50	51	50	0.000
geom100a	**70**	76	71	−1.429
geom100b	**73**	83	79	−8.219
geom110	50	53	50	0.000
geom110a	**74**	82	75	−1.351
geom110b	**79**	88	83	−5.063
geom120	60	62	**59**	1.667
geom120a	**84**	92	86	−2.381
geom120b	**87**	98	91	−4.598

Table 6: Comparison of ABGC against other algorithms for BMCP

Instance	Prestwich	Lim	ABGC	Improvement (%)
geom20	159	149	149	0.000
geom20a	175	169	169	0.000
geom20b	44	44	44	0.000
geom30	168	160	160	0.000
geom30a	235	**209**	210	−0.478
geom30b	79	77	77	0.000
geom40	189	167	167	0.000
geom40a	260	**213**	214	−0.469
geom40b	80	74	74	0.000
geom50	257	224	224	0.000
geom50a	395	318	**317**	0.314
geom50b	89	87	**85**	2.299
geom60	279	258	258	0.000
geom60a	–	358	**357**	0.279
geom60b	128	**116**	117	−0.862
geom70	310	273	**267**	2.198
geom70a	–	**469**	470	−0.213
geom70b	133	121	121	0.000
geom80	–	383	**382**	0.261
geom80a	–	379	**367**	3.166
geom80b	152	141	**139**	1.418
geom90	–	332	332	0.000
geom90a	–	**377**	378	−0.265
geom90b	–	157	**150**	4.459
geom100	–	**404**	405	−0.248
geom100a	–	459	**440**	4.139
geom100b	–	170	**164**	3.529
geom110	–	383	**378**	1.305
geom110a	–	494	**487**	1.417
geom110b	–	**206**	208	−0.971
geom120	–	402	**398**	0.995
geom120a	–	556	**548**	1.439
geom120b	–	199	**198**	0.503

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

[25] Maniezzo, V. and A. Carbonaro, "Ant Colony Optimization: An Overview," *Essays and Surveys in Metaheuristics*, C. Ribeiro editor, Kluwer Academic Publishers, 2001, pp. 21–44.

[26] Morgenstern, C., "Distributed Coloration Neighborhood Search," *Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993*, American Mathematical Society, 26, 1996, pp. 335–358.

[27] Park, E., Y. Kim, and B. R. Moon, "Genetic Search for Fixed Channel Assignment Problem with Limited Bandwidth," Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2002), pp. 1772–1779, 2002.

[28] Phan, V. and S. Skiena, "Coloring Graphs with a General Heuristic Search Engine," Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002.

[29] Prestwich, S. D. "Constrained Bandwidth Multicoloration Neighborhoods," Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University, Ithaca, NY, 2002.

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