An Ant-Based Algorithm for Coloring Graphs

Thang N. Bui * ThanhVu H. Nguyen Chirag M. Patel
Kim-Anh T. Phan

Penn State Harrisburg, Computer Science Program, Middletown, PA 17057, USA

Abstract

This paper presents an ant-based algorithm for the graph coloring problem. An important difference that distinguishes this algorithm from previous ant algorithms is the manner in which ants are used in the algorithm. Unlike previous ant algorithms where each ant colors the entire graph, each ant in this algorithm colors a small portion of the graph using only local information. These individual coloring actions by the ants form a coloring of the graph. Even with the lack of pheromone laying capacity by the ants, the algorithm performed well on a set of 119 benchmark graphs. Furthermore, the algorithm produced very consistent results, having very small standard deviations over 50 runs of each graph tested.

Key words: graph coloring, ant-based algorithm

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A k-*coloring* of G is a mapping $f : V \longrightarrow C$, where $|C| = k$. The elements of C are called *colors*. A k-coloring is called *proper* if for all $(u, v) \in E$, $f(u) \neq f(v)$, i.e., adjacent vertices must have different colors. The minimum k such that G has a proper k-coloring is called the *chromatic number* of G and is denoted by $\chi(G)$. The *conflict* at a vertex v under f is the number of vertices adjacent to v having the same color as v under f. The total conflict in G under f is the number of pairs of adjacent vertices that have the same color under f. The *graph coloring problem* is the problem of finding the minimum k such that the given

* Corresponding author.
Email addresses: `tbui@psu.edu` (Thang N. Bui), `txn131@psu.edu` (ThanhVu H. Nguyen), `cmp196@psu.edu` (Chirag M. Patel), `kap158@psu.edu` (Kim-Anh T. Phan).

input graph G has a proper k-coloring. It is well known that the graph coloring problem is NP-hard. In fact, determining whether a given graph can be colored with three colors or not is NP-complete. There exists an algorithm that can approximate the chromatic number within $O\left(|V|(\log\log|V|)^2/(\log|V|)^3\right)$[22]. However, it is known that we cannot approximate within $|V|^{1/7-\epsilon}$, for any $\epsilon > 0$ unless P = NP [2]. The graph coloring problem arises in a wide variety of problems such as task scheduling, time tabling, frequency assignment in communication networks, register allocation, short circuit testing in PCB design and traditional map coloring. Since the graph coloring problem is NP-hard and approximation algorithms are not very promising as mentioned above, much work has been concentrated on designing heuristic algorithms for the problem. Heuristics for the graph coloring problem use various algorithm design techniques including constructive methods [5][27], iterative methods [12], genetic algorithms [32], local search methods [30], tabu techniques [21], and ant system algorithms [8][9][11][32].

In this paper we give an ant-based algorithm for the graph coloring problem. Ant-based algorithms and ant systems are optimization techniques that imitate the collective ability of an ant colony to solve problems [4]. A number of ant system algorithms for the graph coloring problem have been proposed in recent years [11][8][9]. Our ant-based algorithm has a number of features that are different from previous ant system algorithms for the coloring problem. Unlike the ant system algorithm of Costa and Hertz [11], where the ants are allowed to move from one vertex to any other vertex in the graph, the ants in our algorithm can only move along the edges of the graph staying closer with the ant colony metaphor. Also the results obtained by each ant in Costa and Hertz's algorithm were based on sequential and traditional methods like RLF [27] or DSATUR [5]. Our method, instead, is based on local search by each ant using new conflict minimization techniques. In comparison to Comellas and Ozon's ant system algorithm [8] for graph coloring, where each ant colors the entire graph and does just one coloring, each of our ants colors a portion of the graph and does this in stages, called *cycles*. The local coloring that each of our ants does in each cycle is affected by the colorings of other ants in previous cycles. Unfortunately, it was difficult to compare our results with those of Comellas and Ozon, as they have not used any standard set of test instances like those used in the DIMACS Second Challenge [25]. We tested our algorithm using graphs provided by the COLOR04 Computational Symposium web site (http://mat.gsia.cmu.edu/COLOR04/). These are the graphs of the DIMACS Second Challenge. The performance of our algorithm on a set of 119 graphs is very encouraging. In this set of benchmark graphs there are 63 graphs with known chromatic numbers or best known upper bound on the chromatic number. Our algorithm matched the best known values for 56 of these instances and came within 1 of the best known values for the remaining 7 instances.

The rest of the paper is organized as follows. In Section 2 we give a brief description of ant system algorithms. We describe our algorithm in Section 3 and present the experimental results in Section 4. The conclusion is given in Section 5.

2 Ant System Algorithms

Ant System (AS) is a heuristic technique that imitates the behavior of a colony of ants and their ability to collectively solve problems. For example, it has been observed that a colony of ants is able to find the shortest path to a food source by marking their trails with a chemical substance called *pheromone* [4][14].

As an ant moves and searches for food, it lays down pheromone along its path. As it decides where to move, it looks for pheromone trails and prefers to follow trails with higher levels of pheromone. Suppose there are two possible paths to reach a food source. Regardless of the path chosen, the ant will lay the same amount of pheromone at each step. However, it will return to its starting point quicker when it takes the shorter path. It is then able to return to the food source to collect more food. Thus, in an equal amount of time, the ant would lay a higher concentration of pheromone over its path if it takes the shorter path, since it would complete more trips in the given time. Ants prefer to follow the path with the most accumulation of pheromone, which happens to be the shortest path. In addition, some pheromone evaporates over time although not a significant amount [4][14].

The Traveling Salesman problem (TSP) was one of the first problems to which the Ant System (AS) technique was applied [4] [14]. TSP is the well-known problem of finding the smallest cost tour in an edge weighted complete graph. The tour must visit each vertex in the graph exactly once, starting and ending at the same vertex. The cost of a tour is the sum of the costs of the edges in the tour. A typical ant system algorithm or more precisely, ant colony optimization (ACO) algorithm, for TSP consists of a number of ants. Each ant takes turn finding a tour in the given input graph. After an ant has found a tour, it deposits an equal amount of pheromone on each edge of the tour. The amount of pheromone deposited is a function of the cost of the tour. Normally, this amount is inversely proportional to the cost of the tour, i.e., the smaller the cost of the tour the more pheromone are deposited. Thus, edges in a low cost tour will have more pheromone deposited on them than edges in a high cost tour. Also, an edge may have more pheromone deposited on it if it is used in one or more tours constructed by the ants. Pheromone effectively acts as a memory device helping later ants to construct their tours. In fact, the amount of pheromone on each edge is an important factor in the construction of a tour by an ant. Generally, ants will tend to pick edges with higher concentration of pheromone in constructing tours in the graph. To mitigate the problem of getting stuck in a local optimum, pheromone is allowed to evaporate. Experimental results reported in [4] and [14] showed that ACO algorithms for TSP are very competitive against existing algorithms for TSP.

Other problems that have been the focus of AS as well as Ant Colony Optimization (ACO) [13] work include the quadratic assignment, network routing, vehicle routing, frequency assignment, graph coloring, shortest common supersequence, machine scheduling, multiple knapsack and sequential ordering problems, graph partitioning, maximum clique [28] [4].

3 An Ant-Based Algorithm for Coloring Graph (ABAC)

In this section we describe an ant system algorithm for the graph coloring problem, called ABAC. The main idea of the algorithm is for a set of ants to color the graph. The ants are randomly distributed to the vertices of the input graph. Each ant follows the same set of rules to color the vertices of the graph. Our approach differs from the ACO approach in that each of our ants does not find a complete solution to the problem as is the case in the ACO algorithms for graph coloring of [8][11] or the ACO algorithm for TSP as described in the previous section. Instead, each ant in our algorithm ABAC colors only a portion of the graph. In this manner, ABAC is more amenable to a distributed implementation. However, in this paper we do not present such an implementation. Another difference in our approach is that ants in our algorithm do not have pheromone laying capability. For our case, this helps reduce the running time and in limited experiments we found that in this algorithm pheromone did not show visible or significant effect on the quality of the solution.

3.1 The General Idea

Let $G = (V, E)$ be the input graph. We first run a slightly modified version of the XRLF algorithm [24][27], which we call MXRLF, on G to obtain a proper k-coloring of G. Note that k is an upper bound on the chromatic number, $\chi(G)$, of G. An initial coloring of G, which may not be a proper coloring, is derived from this proper k-coloring by MXRLF. A colony of ants is then randomly distributed to the vertices of the graph. The algorithm then proceeds in a number of cycles. In each cycle, 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

Input: Graph $G = (V, E)$
Output: A coloring of G. Assume that colors are integers starting from 1.

begin
  Use *MXRLF* to obtain a coloring of G, called *currentColoring*
  Let k be the number of colors in *currentColoring* // $k \geq \chi(G)$
  bestColoring $\longleftarrow$ *currentColoring*, *bestNumColors* $\longleftarrow k$
  availableColors $\longleftarrow \lceil \alpha k \rceil$ // initial number of available colors

  Modify *currentColoring* as follows
    Select $\lceil \beta k \rceil$ color classes at random
    Rename these selected colors with integers from the set $\{1, \ldots, \lceil \beta k \rceil\}$
    Erase the color of vertices not belonging to the above $\lceil \beta k \rceil$ color classes
    Color the uncolored vertices using $\lceil \gamma k \rceil$ color classes

  Compute the conflict at each vertex and the *totalConflict* of G
  Distribute *nAnts* randomly on the vertices of G

  for $cycle = 1$ **to** *nCycles* **do**
    for $ant = 1$ **to** *nAnts* **do**
      for $move = 1$ **to** *nMoves* **do**
        ant colors its current vertex // i.e., *currentColoring* is modified
        ant updates local conflict costs in current neighborhood
        ant updates its *recentlyVisited* tabu list
        ant moves to another vertex using path of length 2
      endfor
    endfor

    update *totalConflict* cost for the entire graph G
    if *totalConflict* $= 0$ and *bestNumColors* $>$ *availableColors* **then**
      bestColoring $\longleftarrow$ *currentColoring*
      bestNumColors $\longleftarrow$ *availableColors*
      availableColors $\longleftarrow$ *availableColors* $- 1$
    endif

    if *availableColors* has not improved for *nChangeCycles* cycles **then**
      availableColors $\longleftarrow$ *availableColors* $+ 1$
    if *availableColors* has not improved for *nJoltCycles* cycles **then**
      perform a jolt operation
    if *bestNumColors* has not improved in the last *nBreakCycles* cycles
      then break
  endfor

  return *bestColoring*
end

Fig. 1. An ant-based algorithm for coloring graphs (ABAC)

coloring then the number of available colors is reduced by one and we start another cycle. Otherwise, we may increase the number of 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. The complete algorithm is given in Figure 1. In the following subsections we describe the various parts of the algorithm in detail.

3.2 Initial Coloring

In the following discussion, a *color class* is a set of vertices having the same color. A coloring of a graph naturally induces a set of color classes.

Our first objective is to quickly find an upper bound on the chromatic number of the input graph. For this purpose we used an algorithm that is mainly based on the RLF algorithm [27] but also uses some features of the XRLF algorithm [24]. We call this algorithm MXRLF. Let P be a set of uncolored vertices not adjacent to any vertices colored with the current color in consideration and R be a set of uncolored vertices adjacent to at least one vertex colored with the current color in consideration. MXRLF avoids using vertices from R, while building a color class. We used the same technique used in RLF [27], i.e., choosing the first vertex with the maximum degree to add to the first color class. Then we sequentially add the next vertex to the current color class from P having the maximum degree in R. Ties are broken by selecting the vertex with the minimum degree in P. Repeat the above process until P is empty or the color class size limit, `MXRLF_SET_LIMIT`, is exceeded [24]. This is repeated recursively until the entire graph is colored. As our intention was to quickly find an upper bound on the chromatic number, we omitted the exhaustive search method for building the color classes of the XRLF algorithm [24].

Let k be the number of color classes produced by MXRLF. The initial number of colors available to the ants, called `availableColors`, for coloring the graph is set to $\lceil \alpha k \rceil$. From the k color classes produced by MXRLF we select at random $\lceil \beta k \rceil$ color classes to be kept. The remaining vertices that do not belong to the selected color classes are then distributed randomly into $\lceil \gamma k \rceil$ color classes, where $0 < \beta \leq \gamma \leq \alpha < 1$. These $\lceil \gamma k \rceil$ color classes include the $\lceil \beta k \rceil$ color classes selected earlier. The parameters α, β and γ, as well as other parameters to follow will be specified in Section 3.5. Note that color classes are renumbered so that all colors are in the set $\{1, \ldots, \lceil \alpha k \rceil\}$. This coloring is then used as a starting point for the ants. To summarize, we now have a coloring of G having $\lceil \gamma k \rceil$ colors. Note that this coloring may not be a proper coloring of G. We also have a total of $\lceil \alpha k \rceil$ colors available for the ants to use initially.

3.3 How Ants Color

We distribute a colony of `nAnts` randomly to the vertices of the graph. The algorithm consists of a number of cycles. In each cycle ants are activated one at a time. When activated an ant colors a limited local area of the graph without any global knowledge of the graph and using only colors from the set of available colors, i.e., the set $\{1, \ldots, \texttt{availableColors}\}$. When an ant is at a vertex its objective is to color or re-color that vertex so that the conflict at that vertex is zero, if possible. If it is not possible, the ant will select the smallest number color from the list of available colors that will minimize the conflict at that vertex. Furthermore, if zero conflict is not possible, the ant will try to select a color that it has not used in a previous location. This will prevent additional conflicts to previous vertices that the ant has colored. When there is a choice among several available colors satisfying the requirement, the ant just picks one at random. The conflict at this vertex is then updated. Note that the ant does not have knowledge of the total conflicts for the entire graph.

After an ant finishes coloring its current vertex it moves to another vertex and tries to color it. Each ant will make `nMoves` such moves before it stops. The ant moves to another vertex by taking a path of length two. The first edge in that path is selected at random among all edges connected to its current vertex. The second edge in the path is selected so that the ant will end up in a vertex that has the maximum conflict among all vertices adjacent to the vertex at the end of the first edge. Ties are broken arbitrarily. Additionally, each ant also has a tabu list containing recently visited vertices that they cannot revisit. The tabu list helps prevent ants from getting stuck in a loop.

3.4 Perturbation and Stopping Condition

When all the ants finish coloring at the end of a cycle we have a coloring of G. We then compute the total conflict of the current coloring. If the total conflict of the current coloring is zero, we reduce the number of available colors `availableColors` by 1 and continue with the next cycle. On the other hand, we increment `availableColors` by 1 (up to `bestColors`) if `availableColors` has not been changed for the last `nChangeCycles`. We also maintain the best coloring found so far and update that value after each cycle, if appropriate.

To assist ants in escaping local optima, we perturb the current coloring of the graph by a method that we call a *jolt*. More specifically, if there is no reduction in the number of colors used for the last `nJoltCycles` cycles, then the current coloring is perturbed as follows. The vertices in the graph 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 idea of the jolt is to inject enough disturbances into the current coloring to move it out of the current local optimal but not enough to destroy 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.

3.5 Parameters

In what follows, we give a brief description for each important parameter used in the algorithm. These parameters were obtained by testing the ABAC algorithm on a small number of graphs such as circles, lines, trees, caterpillars and grids. A few instances of the DIMACS Second Challenge were also used in these tests. These parameters were not tuned for any particular classes of graphs. The objective is to balance between performance and 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 20% of the number of vertices in the graph. For efficiency reason we do not allow `nAnts` to exceed 100.

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

α is the percentage of color classes produced by MXRLF that is made available for the ants to use initially. We set $\alpha = 80\%$.

β is the percentage of color classes from MXRLF that are kept to create an initial coloring for the graph before the ants start. We set $\beta = 50\%$.

γ is the percentage of color classes from MXRLF that are to be used for coloring vertices that have not been colored in the initial $\lceil \beta k \rceil$ color classes. We set $\gamma = 70\%$.

`MXRLF_SET_LIMIT` is the color class/partition size limit in MXRLF. We set `MXRLF_SET_LIMIT` $= 0.7n$.

`nMoves` is the number of vertices an ant can visit before it stops. We define `nMoves` as follows.

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

`R_SIZE_LIMIT` is the length of a tabu list of recently visited vertices. An ant will avoid revisiting those vertices in its tabu list allowing a more diverse exploration of the graph. We set `R_SIZE_LIMIT` = `nMoves`/3.

`nChangeCycle` is the number of consecutive cycles allowed in which there is no improvement before the number of available colors, `availableColors`, is increased. We set `nChangeCycle` = 20.

`nJoltCycles` is the number of consecutive cycles during which the number of colors used, i.e, `availableColors`, has not improved, before a jolt is applied to the coloring creating a perturbation of the current coloring configuration. We set `nJoltCycles` = $\max\{n/2, 600\}$.

`nBreakCycles` is the number of consecutive cycles during which the value of `availableColors` has not improved before the algorithm is terminated. We set `availableColors` = $\max\{5n/2, 1600\}$.

4 Experimental Results

In this section we present the results of our algorithm on 119 benchmark graphs given at the web site `http://mat.gsia.cmu.edu/COLOR04/`. Information about these graphs is summarized in Table 1. The algorithm was implemented in C++ and run on a 3.2 GHz Mobile Pentium4 PC with 1 GB of RAM running the Linux operating system. The machine benchmark is given at the end of the paper in Figure 2. For each of the 119 graphs in Table 1 we ran our algorithm for 50 trials. Of the 119 graphs there are 63 graphs with either known chromatic number or best known bound on the chromatic numbers. Of these 63 graphs, our algorithm found matching bounds for 56 of them. There are 7 graphs for which our algorithm got poorer results, but are within 1 of the best known bound. The results are summarized in Tables 2 and 3. For each graph, we list the name of the graph, the chromatic number or the best known bound on the chromatic number, the minimum, maximum, average and standard deviation of the results produced by our algorithm in 50 trials. We also list the average running time (in seconds) out of the 50 runs of each graph. For a number of graphs the running times were too small to be recordable and were recorded as 0. It should be noted that the standard deviations of the results are quite small, less than 1 for all but three graphs. For the remaining three graphs the standard deviations are less than 2.

Table 1
Summary of the 119 test graphs.

Instances $G = (V, E)$	$\lvert V \rvert$	$\lvert E \rvert$	Best Known	Instances $G = (V, E)$	$\lvert V \rvert$	$\lvert E \rvert$	Best Known
1-FullIns_3.col.b	30	100	?	le450_15d.col.b	450	16750	15
1-FullIns_4.col.b	93	593	?	le450_25a.col.b	450	8260	25
1-FullIns_5.col.b	282	3247	?	le450_25b.col.b	450	8263	25
1-Insertions_4.col.b	67	232	4	le450_25c.col.b	450	17343	25
1-Insertions_5.col.b	202	1227	?	le450_25d.col.b	450	17425	25
1-Insertions_6.col.b	607	6337	?	le450_5a.col.b	450	5714	5
2-FullIns_3.col.b	52	201	?	le450_5b.col.b	450	5734	5
2-FullIns_4.col.b	212	1621	?	le450_5c.col.b	450	9803	5
2-FullIns_5.col.b	852	12201	?	le450_5d.col.b	450	9757	5
2-Insertions_3.col.b	37	72	4	miles1000.col.b	128	3216	42
2-Insertions_4.col.b	149	541	4	miles1500.col.b	128	5198	73
2-Insertions_5.col.b	597	3936	?	miles250.col.b	128	387	8
3-FullIns_3.col.b	80	346	?	miles500.col.b	128	1170	20
3-FullIns_4.col.b	405	3524	?	miles750.col.b	128	2113	31
3-FullIns_5.col.b	2030	33751	?	mug100_1.col.b	100	166	4
3-Insertions_3.col.b	56	110	4	mug100_25.col.b	100	166	4
3-Insertions_4.col.b	281	1046	?	mug88_1.col.b	88	146	4
3-Insertions_5.col.b	1406	9695	?	mug88_25.col.b	88	146	4
4-FullIns_3.col.b	114	541	?	mulsol.i.1.col.b	197	3925	49
4-FullIns_4.col.b	690	6650	?	mulsol.i.2.col.b	188	3885	31
4-FullIns_5.col.b	4146	77305	?	mulsol.i.3.col.b	184	3916	31
4-Insertions_3.col.b	79	156	3	mulsol.i.4.col.b	185	3946	31
4-Insertions_4.col.b	475	1795	?	mulsol.i.5.col.b	186	3973	31
5-FullIns_3.col.b	154	792	?	myciel3.col.b	11	20	4
5-FullIns_4.col.b	1085	11395	?	myciel4.col.b	23	71	5
abb313GPIA.col.b	1557	53356	?	myciel5.col.b	47	236	6
anna.col.b	138	493	11	myciel6.col.b	95	755	7
ash331GPIA.col.b	662	4181	?	myciel7.col.b	191	2360	8
ash608GPIA.col.b	1216	7844	?	qg.order30.col.b	900	26100	30
ash958GPIA.col.b	1916	12506	?	qg.order40.col.b	1600	62400	40
david.col.b	87	406	11	qg.order60.col.b	3600	212400	60
DSJC1000.1.col.b	1000	49629	?	qg.order100.col.b	10000	990000	100
DSJC1000.5.col.b	1000	249826	?	queen10_10.col.b	100	1470	?
DSJC1000.9.col.b	1000	449449	?	queen11_11.col.b	121	1980	11
DSJC125.1.col.b	125	736	?	queen12_12.col.b	144	2596	?
DSJC125.5.col.b	125	3891	?	queen13_13.col.b	169	3328	13
DSJC125.9.col.b	125	6961	?	queen14_14.col.b	196	4186	?
DSJC250.1.col.b	250	3218	?	queen15_15.col.b	225	5180	?
DSJC250.5.col.b	250	15668	?	queen16_16.col.b	256	6320	?
DSJC250.9.col.b	250	27897	?	queen5_5.col.b	25	160	5
DSJC500.1.col.b	500	12458	?	queen6_6.col.b	36	290	7
DSJC500.5.col.b	500	62624	?	queen7_7.col.b	49	476	7
DSJC500.9.col.b	500	112437	?	queen8_12.col.b	96	1368	12
DSJR500.1.col.b	500	3555	?	queen8_8.col.b	64	728	9
DSJR500.1c.col.b	500	121275	?	queen9_9.col.b	81	1056	10
DSJR500.5.col.b	500	58862	?	school1_nsh.col.b	352	14612	?
fpsol2.i.1.col.b	496	11654	65	school1.col.b	385	19095	?
fpsol2.i.2.col.b	451	8691	30	wap01a.col.b	2368	110871	?
fpsol2.i.3.col.b	425	8688	30	wap02a.col.b	2464	111742	?
games120.col.b	120	638	9	wap03a.col.b	4730	286722	?
homer.col.b	561	1628	13	wap04a.col.b	5231	294902	?
huck.col.b	74	301	11	wap05a.col.b	905	43081	?
inithx.i.1.col.b	864	18707	54	wap06a.col.b	947	43571	?
inithx.i.2.col.b	645	13979	31	wap07a.col.b	1809	103368	?
inithx.i.3.col.b	621	13969	31	wap08a.col.b	1870	104176	?
jean.col.b	80	254	10	will199GPIA.col.b	701	6772	?
latin_square_10.col.b	900	307350	?	zeroin.i.1.col.b	211	4100	49
le450_15a.col.b	450	8168	15	zeroin.i.2.col.b	211	3541	30
le450_15b.col.b	450	8169	15	zeroin.i.3.col.b	206	3540	30
le450_15c.col.b	450	16680	15				

"Best Known" columns indicate the best known upper bound on the chromatic number.
A '?' indicates an unknown value.

Table 2
Performance of ABAC

Instances	Best	50 runs of ABAC on each instance				
	Known	Min	Max	Avg	SD	Avg. Time (s)
1-FullIns_3.col.b	?	4	4	4	0	0.01
1-FullIns_4.col.b	?	5	5	5	0	0.31
1-FullIns_5.col.b	?	6	6	6	0	4.54
1-Insertions_4.col.b	4	5	5	5	0	0.1
1-Insertions_5.col.b	?	6	6	6	0	1.64
1-Insertions_6.col.b	?	7	7	7	0	18.6
2-FullIns_3.col.b	?	5	5	5	0	0.07
2-FullIns_4.col.b	?	6	6	6	0	2.03
2-FullIns_5.col.b	?	7	7	7	0	29
2-Insertions_3.col.b	4	4	4	4	0	0.02
2-Insertions_4.col.b	4	5	5	5	0	0.74
2-Insertions_5.col.b	?	6	6	6	0	17.82
3-FullIns_3.col.b	?	6	6	6	0	0.22
3-FullIns_4.col.b	?	7	7	7	0	11.22
3-FullIns_5.col.b	?	8	8	8	0	68.78
3-Insertions_3.col.b	4	4	4	4	0	0.07
3-Insertions_4.col.b	?	5	5	5	0	4.69
3-Insertions_5.col.b	?	6	6	6	0	36.68
4-FullIns_3.col.b	?	7	7	7	0	0.73
4-FullIns_4.col.b	?	8	8	8	0	22.53
4-FullIns_5.col.b	?	9	9	9	0	170.05
4-Insertions_3.col.b	3	4	4	4	0	0.17
4-Insertions_4.col.b	?	5	5	5	0	12.9
5-FullIns_3.col.b	?	8	8	8	0	1.38
5-FullIns_4.col.b	?	9	9	9	0	33.5
abb313GPIA.col.b	?	9	10	9.32	0.47	62.78
anna.col.b	11	11	11	11	0	1.14
ash331GPIA.col.b	?	4	4	4	0	17.45
ash608GPIA.col.b	?	4	5	4.24	0.43	28.97
ash958GPIA.col.b	?	4	5	4.46	0.5	50.68
david.col.b	11	11	11	11	0	0.38
DSJC1000.1.col.b	?	21	22	21.42	0.5	74.37
DSJC1000.5.col.b	?	91	93	91.9	0.7	285.27
DSJC1000.9.col.b	?	229	233	230.84	1.05	503.29
DSJC125.1.col.b	?	5	6	5.7	0.46	0.92
DSJC125.5.col.b	?	17	18	17.8	0.4	1.69
DSJC125.9.col.b	?	44	44	44	0	3.51
DSJC250.1.col.b	?	8	9	8.5	0.5	4.33
DSJC250.5.col.b	?	29	30	29.14	0.35	13.11
DSJC250.9.col.b	?	72	73	72.4	0.49	23.57
DSJC500.1.col.b	?	13	13	13	0	28.92
DSJC500.5.col.b	?	50	52	51.2	0.6	98.55
DSJC500.9.col.b	?	127	129	128.36	0.56	145.03
DSJR500.1.col.b	?	12	12	12	0	18.62
DSJR500.1c.col.b	?	85	86	85.1	0.3	154.96
DSJR500.5.col.b	?	128	130	129.24	0.51	147.03
fpsol2.i.1.col.b	65	65	65	65	0	63.18
fpsol2.i.2.col.b	30	30	30	30	0	61
fpsol2.i.3.col.b	30	30	30	30	0	54.43
games120.col.b	9	9	9	9	0	0.72
homer.col.b	13	13	13	13	0	20.75
huck.col.b	11	11	11	11	0	0.2
inithx.i.1.col.b	54	54	54	54	0	97.49
inithx.i.2.col.b	31	31	31	31	0	78.9
inithx.i.3.col.b	31	31	31	31	0	78.35
jean.col.b	10	10	10	10	0	0.35
latin_square_10.col.b	?	100	103	101.48	0.64	305.21
le450_15a.col.b	15	15	15	15	0	31.52
le450_15b.col.b	15	15	15	15	0	28
le450_15c.col.b	15	15	21	19.74	1.81	41.7

Table 3
Performance of ABAC (cont.)

Instances	Best Known	50 runs of ABAC on each instance				
		Min	Max	Avg	SD	Avg. Time (s)
le450_15d.col.b	15	15	21	17.02	1.42	42.66
le450_25a.col.b	25	25	25	25	0	28.71
le450_25b.col.b	25	25	25	25	0	27.14
le450_25c.col.b	25	26	26	26	0	39.55
le450_25d.col.b	25	26	26	26	0	40.71
le450_5a.col.b	5	5	6	5.32	0.47	16.15
le450_5b.col.b	5	5	6	5.44	0.5	16.4
le450_5c.col.b	5	5	5	5	0	20.44
le450_5d.col.b	5	5	5	5	0	20.71
miles1000.col.b	42	42	42	42	0	2.55
miles1500.col.b	73	73	73	73	0	5.11
miles250.col.b	8	8	8	8	0	0.57
miles500.col.b	20	20	20	20	0	1.53
miles750.col.b	31	31	31	31	0	1.95
mug100_1.col.b	4	4	4	4	0	0.25
mug100_25.col.b	4	4	4	4	0	0.35
mug88_1.col.b	4	4	4	4	0	0.17
mug88_25.col.b	4	4	4	4	0	0.16
mulsol.i.1.col.b	49	49	49	49	0	7.3
mulsol.i.2.col.b	31	31	31	31	0	5.69
mulsol.i.3.col.b	31	31	31	31	0	5.86
mulsol.i.4.col.b	31	31	31	31	0	5.81
mulsol.i.5.col.b	31	31	31	31	0	5.85
myciel3.col.b	4	4	4	4	0	0
myciel4.col.b	5	5	5	5	0	0
myciel5.col.b	6	6	6	6	0	0.05
myciel6.col.b	7	7	7	7	0	0.56
myciel7.col.b	8	8	8	8	0	2.49
qg.order30.col.b	30	30	30	30	0	44.31
qg.order40.col.b	40	40	40	40	0	71.91
qg.order60.col.b	60	60	60	60	0	226.36
qg.order100.col.b	100	100	100	100	0	1534.7
queen10_10.col.b	?	11	11	11	0	0.99
queen11_11.col.b	11	12	13	12.02	0.14	1.34
queen12_12.col.b	?	13	14	13.4	0.49	1.84
queen13_13.col.b	13	14	15	14.66	0.48	2.56
queen14_14.col.b	?	16	16	16	0	3.59
queen15_15.col.b	?	17	17	17	0	4.9
queen16_16.col.b	?	18	18	18	0	6.45
queen5_5.col.b	5	5	5	5	0	0.01
queen6_6.col.b	7	7	7	7	0	0.03
queen7_7.col.b	7	7	7	7	0	0.06
queen8_12.col.b	12	12	12	12	0	0.53
queen8_8.col.b	9	9	9	9	0	0.14
queen9_9.col.b	10	10	10	10	0	0.37
school1_nsh.col.b	?	14	14	14	0	16.87
school1.col.b	?	14	14	14	0	23.75
wap01a.col.b	?	43	43	43	0	158.15
wap02a.col.b	?	42	43	42.8	0.4	145.21
wap03a.col.b	?	45	46	45.6	0.49	514.93
wap04a.col.b	?	44	45	44.86	0.35	476.18
wap05a.col.b	?	50	50	50	0	67.49
wap06a.col.b	?	42	43	42.86	0.35	85.69
wap07a.col.b	?	43	44	43.32	0.47	169.88
wap08a.col.b	?	42	44	43.02	0.32	175.84
will199GPIA.col.b	?	7	7	7	0	22.44
zeroin.i.1.col.b	49	49	49	49	0	8.81
zeroin.i.2.col.b	30	30	30	30	0	8.58
zeroin.i.3.col.b	30	30	30	30	0	8.23

5 Conclusion

In this paper we presented an ant-based algorithm that seems to perform well on a set of 119 DIMACS benchmark graphs. This ant-based algorithm has not given ants the ability to leave pheromone which generally helps improve the performance of ant-based algorithms. In limited experiments we found that in this particular algorithm, adding pheromone laying capability increases running time without providing visible or significant performance improvement.

6 Acknowledgements

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

References

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

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

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

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

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

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

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

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

[9] F. Comellas 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).

[10] C. Coritoru, H. Luchian, O. Gheorghies and A. Apetrei, A New Genetic Graph Coloring Heuristic, Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University (2002).

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

[12] J. Culberson 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) 245–284.

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

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

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

[16] P. Galinier, A. Hertz and N. Zufferey, Adaptive Memory Algorithms for Graph Coloring, Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University (2002).

[17] P. Galinier and J. Hao, Hybrid Evolutionary Algorithms for Graph Coloring, Journal of Combinatorial Optimization 3(4) (1998) 379–397.

[18] L. M. Gambardella and M. Dorigo, An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem, INFORMS Journal on Computing 12(3) (2000) 237–255.

[19] F. Glover, M. Parker and J. Ryan, Coloring by Tabu Branch and Bound, *Cliques, Coloring and Satisfiability – Second DIMACS Implementation Challenge 1993*, American Mathematical Society 26 (1996) 285–307.

[20] C. Gomes and D. Shmoys, Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem, Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University (2002).

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

[22] M. M. Halldórsson, A Still Better Performance Guarantee for Approximate Graph Coloring, Information Processing Letters 45 (1993) 19–23.

[23] T. R. Jensen and B. Toft, *Graph Coloring Problems,* Wiley-Insterscience Series in Discrete Mathematics and Optimization, 1995.

[24] D. S. Johnson, 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) (1991) 378–406.

[25] D. S. Johnson 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).

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

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

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

[29] K. Mizuno and S. Nishihara, Toward Ordered Generation of Exceptionally Hard Instance for Graph 3-Colorability, Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University (2002).

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

[31] V. Phan and S. Skiena, Coloring Graphs with a General Heuristic Search Engine, Computational Symposium on Graph Coloring and Its Generalizations, COLOR02, Cornell University (2002).

[32] T. White, B. Pagurek and F. Oppacher, ASGA: Improving the Ant System by Integration with Genetic Algorithms, Proceedings of the 3rd Conference on Genetic Programming (GP/SGA 98) (1998) 610–617.

The following data was obtained after DFMAX was recompiled on the machine that we tested our algorithm.

DFMAX(r500.5.b)

5.67 (user)   0.00 (sys)   6.00 (real)

Best: 345 204 148 480 16 336 76 223 260 403 141 382 289

Fig. 2. Machine Benchmark