Analele Universităţii din Timişoara
Vol. XLII, Fasc. special, 2004
Seria Matematică–Informatică

# ACODYGRA: AN AGENT ALGORITHM FOR COLORING DYNAMIC GRAPHS

*Presented at 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*

Davy PREUVENEERS, Yolande BERBERS

**Abstract.** The graph coloring problem is a well-known constraint satisfaction problem. It is a generalization of the map-coloring problem for non-planar graphs. Dynamic graphs are graphs subject to discrete changes and encountered in many applications in science and engineering, such as communication networks. In this paper, we propose an agent-based algorithm to color dynamic graphs with the property that edges between vertices may be added or removed. We compare the coloring results with other classic and agent-based algorithms and conclude that for non-fast evolving graphs the proposed algorithm is superior performance-wise.

**Key words:** graph coloring problem, agent-based algorithm, dynamic graph, recursive largest first

**AMS Subject Classification:** 68W05, 68T99

**1. Introduction** The graph coloring problem (GCP) is an assignment problem where each vertex in a graph $G$ is labeled with a color. In a proper coloring no two vertices having the same color share an edge. We say that graph G is $k-colored$ if every two adjacent vertices have been assigned a different color from a set of $k$ unique colors. The graph coloring problem is known to be NP-hard. Therefore heuristic-based approaches must be used to find a non-optimal but acceptable solution.

An interesting graph coloring problem is the frequency assignment problem in cellular phone networks (Borndörfer et al., 1998) where frequencies have to be assigned to base stations so that interference is induced as little as possible. As users are mobile, these dynamic interference graphs need to be recolored over and over again.

Constructive methods, such as the DSATUR (Brélaz, 1979) and RLF (Leighton, 1979) methods, start from an uncolored graph and color each vertex until the whole graph is colored. Other agent or ant algorithms (Costa et al., 1997) (Comellas et al., 1998) (Mermet et al., 2002) iterate over a previous coloring to find a feasible or better solution.

Our algorithm *Agent algorithm for Coloring Dynamic Graphs* (ACODYGRA) focuses on the problem of coloring a dynamic graph with edges between vertices being removed and inserted. The algorithm starts from an initial valid RLF coloring and lets agents recompute the solution when the graph evolves due to discrete changes.

This paper is structured as follows: In section 2 we present some notations and definitions that are used in the following sections. Sections 3 and 4 describe experiments on classic constructive and agent-based algorithms to solve the assignment problem. The dynamic graph coloring problem and our agent-based algorithm for dynamic graphs ACODYGRA is discussed in section 5. We conclude and discuss future work in section 6.

**2. Definitions** A graph $G = (V, E)$ consists of 2 sets $V$ and $E$. The elements in $V$ are the vertices, the elements in $E$ are the edges. Each edge is defined by a pair of vertices $(u, v) \in V^2$. The adjacent vertices $u$ and $v$ are *neighbors*. The set of all neighbors of $u$ is defined as $N(u)$, while the cardinality of $N(u)$ is defined as the degree of $u$, $deg(u)$. The color that has been assigned to $u$ is defined as $C(u)$, while the set of colors that has been assigned to its neighbors is defined as $C(N(u))$. The number of unique colors in $C(N(u))$ is the degree of saturation of $u$, $deg_s(u)$. The number of colors in an optimal coloring of the graph $G$ is defined as the chromatic number of the graph $\chi(G)$. Cliques are completely connected subgraphs that for a proper optimal coloring require as many colors as there are vertices in the subgraph. The size of the largest clique in the whole graph $G$ is a lower bound for the chromatic number of the graph $\chi(G)$. For example, a simple circular graph with 5 vertices requires 3 colors, while the largest clique only contains two vertices. The average edge density $\rho_{edge}(G)$ of a graph $G$ defines the average number of edges per vertex and determines the coloring complexity of the graph:

$$\rho_{edge}(G) = \frac{|E|}{|V|} = \frac{\sum_{i=1}^{i=|V|} |N(v_i)|}{2.|V|} \qquad v_i \in V$$

**Table 1.** Average number of colors and time for constructive algorithms on graphs with $n$ vertices and edge probability $p$.

| algorithm | p | n=100 | | n=300 | | n=500 | | n=1000 | |
|---|---|---|---|---|---|---|---|---|---|
| | | col. | time | col. | time | col. | time | col. | time |
| Greedy | **0.1** | 7.4 | .002 | 14.4 | .004 | 19.6 | .008 | 31.7 | .023 |
| | **0.3** | 14.1 | .001 | 30.3 | .006 | 44.5 | .013 | 75.9 | .057 |
| | **0.5** | 20.7 | .001 | 48.9 | .007 | 72.7 | .023 | 126.6 | .090 |
| | **0.7** | 30.5 | .001 | 71.7 | .011 | 109.8 | .024 | 193.1 | .121 |
| DSATUR | **0.1** | 5.9 | .031 | 11.4 | .406 | 16.1 | 1.82 | 27.0 | 15.8 |
| | **0.3** | 12.1 | .033 | 27.1 | .789 | 39.9 | 3.79 | 69.3 | 32.3 |
| | **0.5** | 18.8 | .036 | 44.0 | 1.11 | 65.5 | 5.26 | 116.9 | 49.4 |
| | **0.7** | 28.1 | .041 | 66.4 | 1.47 | 102.1 | 7.24 | 181.5 | 70.3 |
| RLF | **0.1** | 5.8 | .041 | 10.7 | .117 | 14.8 | .477 | 24.7 | 3.65 |
| | **0.3** | 11.1 | .035 | 24.4 | .306 | 35.8 | 1.31 | 63.0 | 9.53 |
| | **0.5** | 17.3 | .044 | 40.2 | .531 | 60.1 | 2.18 | 107.9 | 16.6 |
| | **0.7** | 25.4 | .060 | 61.2 | .816 | 93.0 | 3.38 | 167.7 | 26.5 |

**3. Constructive methods for graph coloring** Recursive Largest First (RLF) (Leighton, 1979) is a constructive method that starts from an empty solution and inserts new colored vertices into the current partial solution. It tries to color as many vertices as possible with the same color before continuing with a new color. The vertex selection depends on the degree of connectivity of uncolored vertices to previously colored ones. We will use this algorithm to compare the quality of the ant and agent-based algorithms, as it is one of the classic approaches that leads to better colorings in a reasonable time. Other constructive algorithms that have been tested are Greedy (Costa et al., 1997) and DSATUR (Brélaz, 1979). The Greedy method visits all vertices in no specific order and colors them by using the first color from a set that does not result in two neighbors having the same color. DSATUR is also a constructive method like the previous, but here the order in which vertices are selected for coloring is based on their degree of saturation. Vertices $u$ with the highest degree of saturation $deg_s(u)$ are colored first.

The previous algorithms have been tested on random graphs $G_{n,p}$, with $n$ the number of vertices and $p$ the edge probability between each pair of vertices, for $n = 100, 300, 500, 1000$ and $p = 0.1, 0.3, 0.5, 0.7$. The edge probability $p$ determines the average edge density $\rho_{edge}(G) \approx p.n(n-1)/2$ of graph $G$. The test results were obtained by taking the average of 10 test runs on random graphs $G_{n,p}$ with the same parameters $n$ and $p$. These algorithms were implemented in the Java language and executed on an Athlon XP 2.4GHz. The average number of colors and computing time can be found in Table 1. The computing time in all tables is expressed in seconds, unless otherwise denoted. When comparing the algorithms, we can conclude that RLF results in solutions using the least number of unique colors.

**4. Ant and agent-based algorithms for graph coloring** Ant or agent-based algorithms are not new in the field of graph coloring. The method of ant systems (AS) was proposed in 1991 by Colorni et al. (Dorigo et al., 1991) in an Assignment Type Problem-framework (ATP). Later on, other researchers have adapted the principles of ant systems to solve the graph coloring problem (Vesel et al., 2000). However, most of these algorithms focus on coloring a single static graph. We will show in Table 1 and Table 2 that these algorithms require much more time to achieve the same (sub)optimal coloring of a classic constructive algorithm such as RLF (Leighton, 1979) or DSATUR (Brélaz, 1979).

**4.1. Costa and Hertz** The previously discussed constructive algorithms, RLF and DSATUR, are also used by Costa and Hertz (Costa et al., 1997) for comparing their ant algorithm ANTCOL. ANTCOL is actually a family of 8 proposed variants with 6 of them based on the RLF technique and 2 on DSATUR. We will use the RLF-based implementation that resulted in the best colorings to compare with the other algorithms. A large difference with their classic equivalents is that the vertex and color selection is probability-based. During each cycle in the iterative coloring process each ant colors the full graph in a constructive way, and the experience of each ant after each iteration is memorized in a periodically updated $n \times n$ matrix $M$, with $n = |V|$. This matrix represents the pheromone trails that real ants use to communicate indirectly with each other. The evaporation of these pheromone trails is simulated by multiplying the entries $M_{u,v}$ with a factor smaller than 1. When vertices $u$ and $v$ were given the same color by an ant coloring, the matrix entry $M_{u,v}$ is increased. The best graph coloring found so far will be remembered. In this RLF-based implementation of ANTCOL each uncolored vertex has a certain probability to be colored next. This probability is derived from the values in the experience matrix $M$. Another difference is that the first uncolored vertex is randomly chosen, instead of being the one with the highest degree. This random selection would result in a beneficial diversification effect.

**4.2. Comellas and Ozón** In the algorithm proposed by Comellas and Ozón (Comellas et al., 1998) all ants work together on a single coloring. They move around and color vertices based on local heuristics by selecting a color from a fixed set of $k$ colors. Hence the graph will not be properly colored until a feasible $k$-coloring is found. When an ant jumps to a vertex, it will recolor the vertex if it has the same color as one of its neighbors. It selects the color that gives the least conflicts with its adjacent vertices. Then the ant jumps to the neighbor with the most color conflicts. Random jumps and colorings are used to avoid ants getting stuck in endless loops or separated graphs not getting colored. While easier to understand than ANTCOL, this algorithm lacks any physical similarity with the pheromone trails in the ant world.

**4.3. Mermet, Simon and Flouret** Mermet et al. propose an algorithm (Mermet et al., 2002) that starts with an initial proper coloring and that tries to reduce the number of unique colors. The authors define the *local chromatic number (lcn)* for each vertex $u$ as the size of the largest clique to which vertex $u$ belongs. The *lcn* is then used as a lower bound for the *current chromatic number (ccn)*, the number of unique colors currently used by vertex $u$ and its neighbors $N(u)$. Each vertex is managed by an agent that will try to recolor its own vertex as long as the *ccn* of one of its neighbors is larger than its *lcn*. It does not change the color of its neighbors directly, but changes the color of its own vertex $u$ to reduce the *ccn* of one of its neighbors, e.g. vertex $v \in N(u)$, by selecting a color from $C(N(v)) \setminus \{C(N(u)) \cup C(u)\}$. If this is not possible, vertex $u$ *attacks* another adjacent vertex $w \in N(u)$ that in turn will change its color to one of $C(N(N(w))) \setminus C(N(w))$.

**4.4. Experimental results** The first test approach compares the time required for finding a valid $k$-coloring with $k$ the number of colors in a RLF coloring. The time required by each algorithm can be found in Table 2. The second test method gives each algorithm a fixed amount of time − here 250 times the required time for a coloring by the RLF algorithm − and compares the color quality by counting the number of unique colors. These results can be found in Table 3. In both tests the algorithm of Comellas and Ozón uses 10 ants with a random jump probability of 0.2 and a random coloring probability of 0.01 to introduce some diversification.

[1] No improvement upon the initial coloring was found when the deadline expired.

[2] The algorithm was still busy finding all cliques. No attempt was made to improve the initial coloring.

**Table 2.** Average time for a $k$-coloring with $k$ the number of colors required by the RLF algorithm.

| algorithm | p | n=50 time | n=100 time | n=300 time | n=500 time |
|---|---|---|---|---|---|
| **Costa** | **0.1** | .574 | 196 | $>24h$ | $>24h$ |
| **and** | **0.3** | 1.55 | 761 | $>24h$ | $>24h$ |
| **Hertz** | **0.5** | 1.58 | 4326 | $>24h$ | $>24h$ |
| | **0.7** | 1.51 | 4543 | $>24h$ | $>24h$ |
| **Comellas** | **0.1** | .060 | .208 | 12.1 | 45.8 |
| **and** | **0.3** | .253 | .595 | 136 | 2066 |
| **Ozón** | **0.5** | .448 | 3.13 | 515 | 6623 |
| | **0.7** | 1.01 | 7.94 | 1270 | 25682 |
| **Mermet** | **0.1** | 4.25 | $>24h$ | $>24h$ | $>24h$ |
| **Simon** | **0.3** | 3.14 | 16596 | $>24h$ | $>24h$ |
| **and** | **0.5** | 1.56 | 919 | $>24h$ | $>24h$ |
| **Flouret** | **0.7** | 1.07 | 212 | $>24h$ | $>24h$ |

These parameters were manually fine-tuned, but were by no means meant to always guarantee optimal coloring results. The method of Costa and Hertz uses 50 ants. The initial coloring for the approach of Mermet et al. was given by a Greedy coloring.

The test results in Table 1 and 2 show that ant and agent algorithms need much more time to achieve the $k$-coloring results of RLF. The algorithm of Mermet et al. does not scale well. The initialization step requires knowledge about the largest clique for each vertex to calculate the *lcn*, but this problem is NP-hard as well. Finding all largest cliques for a graph with $n = 300$ and $p = 0.7$ using a branch-and-bound technique takes more than a day. The method of Comellas and Ozón is the "winner" in the first challenge, although it takes more than 1000 times as long as RLF in some cases.

Looking at the results in Table 3, it seems that for a fixed amount of time, the coloring quality achieved by the method of Comellas and Ozón is slightly better then the method of Costa and Hertz. The algorithm of Mermet et al. is not up to par with its competitors. This is due to the long computation times for calculating the *lcn* for each vertex.

**5. Coloring dynamic graphs** In the previous sections, we have shown that ant and agent algorithms are able to color graphs. However, their

**Table 3.** Average number of unique colors after fixed running time.

| **algorithm** | **p** | **n=50 colors** | **n=100 colors** | **n=300 colors** | **n=500 colors** |
|---:|---:|---:|---:|---:|---:|
| **Costa** | **0.1** | 4.0 | 6.2 | 13.4 | 18.9 |
| **and** | **0.3** | 7.4 | 12.4 | 28.9 | 43.1 |
| **Hertz** | **0.5** | 10.6 | 19.1 | 46.5 | 69.9 |
| | **0.7** | 15.2 | 27.4 | 68.3 | 105.5 |
| **Comellas** | **0.1** | 4.0 | 5.0 | 10.3 | 15.2 |
| **and** | **0.3** | 6.7 | 10.9 | 26.7 | 40.9 |
| **Ozón** | **0.5** | 10.1 | 17.5 | 45.9 | 70.7 |
| | **0.7** | 14.4 | 26.0 | 69.8 | 107.8 |
| **Mermet** | **0.1** | 4.7 | 7.4 | *14.4*[1] | *19.6*[1] |
| **Simon** | **0.3** | 7.7 | 14.1 | *30.3*[1] | *44.5*[1] |
| **and** | **0.5** | 10.4 | 19.7 | *48.9*[2] | *72.7*[2] |
| **Flouret** | **0.7** | 14.3 | *30.5*[2] | *71.7*[2] | *109.8*[2] |

solutions require more colors or much more time than a classic dynamic constructive algorithm such as RLF.

We then investigated how an agent-based approach would perform in a dynamic graph coloring problem, where multiple agents are able to quickly respond to changes in the graph structure. Dynamic graphs are the result of discrete changes in the graph, i.e. the appearing and disappearing of vertices and edges. The method by Mermet et al. is not a good candidate for dynamic graph colorings. After each update all cliques need to be recalculated and the algorithm would in this case wrongly assume that the dynamic graph is always correctly colored. The method by Costa and Hertz does not look promising either, since each ant actually performs an RLF-like coloring, while the coloring experience of older graphs is rendered useless. The algorithm by Comellas and Ozón seems more feasible than the others with the only drawback that it tries to find a proper $k$-coloring. When a graph is subject to changes, a valid $k$-coloring with $k$ defined for the previous graph, is not guaranteed to exist.

The method *Agent algorithm for Coloring Dynamic Graphs* (ACO-DYGRA) we propose in this paper is fast and simple: only recolor vertices when necessary and try to keep the degree of saturation as low as possible. In our proposal, we will only test on graphs with changing edges. Thus, no new vertices are inserted or old ones removed. This allows us to implement dynamic graphs as a fixed user-defined fraction of all edges being replaced by other edges. By keeping the average edge density $\rho_{edge}(G)$ in the graph

**Algorithm 1** RecolorGraph(**in**: graph $G_{old,new}(V, E)$)

$E_{add} \leftarrow E_{new} \setminus E_{old}$
$E_{remove} \leftarrow E_{old} \setminus E_{new}$
$W \leftarrow \{\}$
**for all** $(u, v) \in E_{add}$ **do**
  **if** $C(u) = C(v)$ **then**
    **if** $deg_s(u) < deg_s(v)$ **then**
      $RecolorVertex(u)$
      $W = W \cup \{u\}$
    **else**
      $RecolorVertex(v)$
      $W = W \cup \{v\}$
**for all** $w \in \{u, v \mid (u, v) \in E_{remove}\} \setminus W$ **do**
  $RecolorVertex(w)$

the same, the average RLF color usage more or less remains the same as well. Inserting and removing vertices would make it more complicated to preserve the complexity of the graph.

As RLF outperforms the above agent algorithms, we will use it to initialize the first coloring and let agents recompute the solution when the graph is updated, based on the current coloring. We hope that this will lead to good colorings, while taking less time than some constructive algorithms. In our current implementation each vertex is occupied by an agent which is able to recolor its own vertex and its neighbors. There are two reasons for a vertex to be recolored:

- A new edge is inserted connecting 2 vertices with the same color. One of both vertices clearly needs to be recolored.
- Although removing an edge causes no color conflict, it decreases the number of constraints on the color of a vertex. This could be used to lower the color usage in the graph.

We therefore introduce two sub-algorithms, one for selecting the vertex to be recolored (Algorithm 1), and another for selecting a new feasible color (Algorithm 2).

**5.1. Algorithm outline** The most simple approach would be to only recolor conflicting vertices by selecting the first available and feasible color, just as in the greedy coloring algorithm. However, this yields to a quickly increasing color usage. A first improvement for recoloring is to select the

**Algorithm 2** RecolorVertex(**in**: vertex $v$)

$i \leftarrow 1$
**while** $c_i \in C(N(v))$ **do**
  $i \leftarrow i + 1$
**if** $\exists j : c_j \in C(N(v)) \ \wedge \ j > i$ **then**
  recolor vertex $v$ with color $c_i$
**else**
  **for** $k \leftarrow 1$ to $q$ **do**
    $satur[k] \leftarrow 0$
  **for all** $u \in N(v)$ **do**
    $c_k \leftarrow C(u)$
    $satur[k] \leftarrow max(satur[k], deg_s(u))$
  select $j$ with $\forall c_k \in C(u) : satur[j] \leq satur[k]$
  **if** $satur[j] < i - 1$ **then**
    recolor vertex $v$ with color $c_j$
    **for all** $u \in \{w \in N(v) | C(w) = c_j\}$ **do**
      recolor vertex $u$ using greedy method
  **else**
    recolor vertex $v$ with color $c_i$

best candidate from the pair of conflicting adjacent vertices by taking the one with the lowest degree of saturation. This vertex has the lowest chance of increasing the total color usage. The second improvement also recolors vertices with a disappearing edge to reduce the color usage. This last step is only necessary if the vertex has not already been recolored due to a new edge with another vertex. This process is outlined in Algorithm 1.

When recoloring a vertex $v$, we search for the first available color $c_i$ that does not result in a color conflict with one of the adjacent nodes $N(v)$. Suppose the ordered set of colors C is given as

$$C = \{c_1, c_2, \ldots, c_i, \ldots, c_q\}$$

We then try to find the color with lowest index $i$ so that

$$c_i \notin C(N(v))$$

If there is an adjacent vertex with a higher ranked color than $c_i$, we have not increased the number of colors. If however $c_i$ is larger in rank than all colors in $C(N(v))$, a new color might have been introduced. The index $i$

**Table 4.** ACODYGRA on a graph with $n$=1000 vertices with $i$=1000 updates and edge update fraction $f$=0.1%

| **n=1000 i=1000 f=0.1%** | | **p=0.1 col.** | **time** | **p=0.3 col.** | **time** | **p=0.5 col.** | **time** | **p=0.7 col.** | **time** |
|---|---|---|---|---|---|---|---|---|---|
| | **avg:** | 31.7 | .023 | 76.2 | .060 | 127 | .099 | 194 | .136 |
| **Greedy** | **min:** | 30 | .018 | 72 | .046 | 121 | .076 | 188 | .107 |
| | **max:** | 34 | .130 | 80 | .108 | 131 | .173 | 200 | .197 |
| | **avg:** | 24.6 | 3.70 | 63.2 | 9.57 | 108 | 16.7 | 168 | 26.2 |
| **RLF** | **min:** | 24 | 3.31 | 61 | 8.65 | 105 | 15.3 | 163 | 24.0 |
| | **max:** | 25 | 6.53 | 65 | 11.1 | 111 | 19.8 | 172 | 28.1 |
| | **avg:** | 27.9 | .009 | 69.5 | .080 | 117 | .251 | 179 | .598 |
| **ACODYGRA** | **min:** | 25 | .000 | 63 | .007 | 108 | .022 | 170 | .042 |
| | **max:** | 29 | .126 | 71 | .228 | 119 | .781 | 183 | 1.63 |

of this new color reflects the degree of saturation of this vertex

$$deg_s(v) = ccn(v) - 1 = i - 1$$

In this case, we check if it is possible to reduce this degree of saturation by selecting a color of one of its neighbors and recolor those if necessary. Therefore, we calculate the degree of saturation for all neighbors and store for each color the highest degree of saturation. In the next step we use this information to select the color with the lowest maximum degree of saturation. These steps result in Algorithm 2 for recoloring a vertex.

**5.2. Experimental results** We have tested the combination of these 2 algorithms on graphs with 1000 vertices and varying edge probabilities and compared the coloring results with the Greedy (the fastest) and RLF (the best coloring) constructive algorithms. Our algorithm ACODYGRA does not start from an uncolored graph, but uses the previous coloring. As mentioned before, the initial coloring of the graph was given by an RLF coloring. Graphs were updated $i$=1000 times with edge update fractions $f$=0.1% and 1%. For a graph with $n$=1000 vertices and edge probability $p$=0.3, we have on average 150 000 edges. With an edge update fraction of 1%, on average 1500 edges are replaced after each update cycle. This means that on average each vertex is involved with 2 or 3 new edges. The results of these experiments can be found in Tables 4 and 5. The average time and color usage after 1000 update iterations are given for all algorithms, as well as the minimum and maximum value. Note that for the minimum

**Table 5.** ACODYGRA with on a graph $n$=1000 vertices with $i$=1000 updates and edge update fraction $f$=1%

| **n=1000 i=1000 f=1%** | | **p=0.1 col** | **time** | **p=0.3 col** | **time** | **p=0.5 col** | **time** | **p=0.7 col** | **time** |
|---|---|---|---|---|---|---|---|---|---|
| | avg: | 31.7 | .025 | 76.2 | .063 | 126 | .107 | 193 | .148 |
| **Greedy** | min: | 30 | .019 | 73 | .047 | 122 | .076 | 188 | .109 |
| | max: | 34 | .028 | 80 | .069 | 131 | .385 | 200 | .203 |
| | avg: | 24.6 | 3.84 | 63.2 | 10.3 | 108 | 17.9 | 168 | 28.1 |
| **RLF** | min: | 24 | 3.41 | 61 | 8.73 | 104 | 15.3 | 164 | 24.8 |
| | max: | 26 | 4.30 | 65 | 10.9 | 111 | 30.8 | 172 | 29.5 |
| | avg: | 28.5 | .074 | 71.2 | .587 | 120 | 1.85 | 184 | 4.94 |
| **ACODYGRA** | min: | 26 | .038 | 67 | .284 | 114 | .904 | 175 | 2.43 |
| | max: | 30 | .127 | 73 | 1.05 | 122 | 3.38 | 187 | 7.84 |

and maximum values, the color usage and processing time not necessarily correspond.

For an edge update fraction of $f$=0.1% the coloring efficiency of ACODYGRA lies somewhere in between the RLF and Greedy method. It is comparable with the DSATUR algorithm in Table 1, but ACODYGRA requires much less time. For low edge probabilities ACODYGRA is even faster than the Greedy method. Another remark is that the maximum color usage of ACODYGRA is below the overall minimum color usage of the Greedy method.

For higher edge update fractions, such as $f$=1%, the coloring results of ACODYGRA are slightly worse, but the algorithm takes more time than for the $f$=0.1% experiment. This is understandable since a higher edge update fraction involves more vertices that need to be recolored. These new edges may introduce new color conflicts. However, processing time is still below the DSATUR method with similar coloring results.

**6. Conclusion** After recalling the graph coloring problem and giving an overview of some constructive coloring algorithms and ant or agent-based methods, we have presented experimental results that allow us to conclude that ant and agent algorithms are not up to par with the classic approaches for coloring an uncolored graph. These algorithms require much more time to achieve the same coloring quality or give worse results after a lengthy predefined running time.

We have proposed the ACODYGRA algorithm for coloring dynamic graphs using an agent based approach. By starting from an RLF coloring, experiments in a simulation with discrete events have shown that agents are capable of recoloring a dynamic graph where edges are replaced at the same time. If the set of edges being replaced at each update is small, the coloring results are comparable to the DSATUR method while the performance is slightly lower than for the Greedy algorithm. When many edges are replaced at once, the structure of the graph differs too much from the previous to allow agents to make use of previous coloring results.

Future work will focus on coloring dynamic graphs that include the adding and removing of vertices as well, while preserving the complexity of the graph to be able to compare our method with other classic algorithms. When this problem is solved, we will look in to applying our algorithm on continuously changing dynamic graphs, as this will allow us to test the algorithm on real world dynamic graph coloring problems.

## REFERENCES

Borndörfer, R., Eisenblätter, A., Grötschel, M., Martin, A. (1998). *Frequency assignment in cellular phone networks.* Annals of Operations Research, **76**, 73-93

Brélaz, D. (1979). *New methods to color the vertices of a graph.* Communications of the ACM, **22**, 251-256

Leighton, F.T. (1979). *A graph coloring algorithm for large scheduling problems.* Journal of Research of the National Bureau of Standards, **84**, 489-506

Costa, D., Hertz, A. (1997). *Ants can colour graphs.* Journal of the Operational Research Society, 295-305

Comellas, F., Ozón, J. (1998). *An ant algorithm for the graph colouring problem.* ANTS'98 - From Ant Colonies to Artificial Ants: First International Workshop on Ant Colony Optimization, Brussels, Belgium.

Mermet, B., Simon, G., Flouret, M. (2002). *A multi-agents approach for a graph colouring problem.* SCI 2002, Orlando, USA.

Dorigo, M., Maniezzo, V., Colorni, A. (1991). *The ant system: An autocatalytic optimizing process.* Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy

Vesel, A., Zerovnik, J. (2000). *How good can ants color graphs?* Journal of computing and Information Technology - CIT, **8**, 131-136

Katholieke Universiteit Leuven,
Department of Computer Science
Celestijnenlaan 200A
B-3001 Leuven, BELGIUM