Fast Heuristics for the Edge Coloring of Large Graphs

Mario Hilgemeier Nicole Drechsler Rolf Drechsler

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{mh,nd,rd}@informatik.uni-bremen.de

Abstract

Heuristic algorithms for coloring the edges of large undirected single-edge graphs with (or very close to) the minimal number of colors are presented. Compared to simulated annealing and a grouping genetic algorithm for small graphs, the heuristics were not only faster by orders of magnitude, but almost all solutions had the optimal color number; the rest differed by at most two colors.

For large graphs, the heuristics were validated by an evolutionary algorithm. Here, the heuristics often found an optimum or a solution very close to it.

1. Introduction

Edge coloring of a graph means that the edge colors at each node are all different, which means that no two edges of the same color have a node in common. This is needed in problems where two or more disjunct sets of pairs of neighboring nodes have to be found to complete a given task. Edge colorings are used for partitioning, scheduling, timetabling, and wavelength-routing in optical networks [15, 16], electrical networks [8], and optimal design (especially VLSI circuits [4]).

Modern electronic circuit designs may consist of millions of elements. Hence, the algorithmic complexity to solve the coloring problem has to be as low as possible.

The edge-coloring problem is to color a given graph with the minimal number of colors necessary. With the exception of special cases [14], this problem is known to be NP-complete [12].

Evolutionary algorithms (EAs) have been applied successfully to edge coloring of graphs: For graphs with up to 750 nodes a grouping genetic algorithm was employed by [15] which found optimal or near-optimal solutions. A simulated annealing algorithm was used with graphs of up to 450 nodes in [7]. This method always determined the optimal number of colors, also in cases where the grouping genetic algorithm of [15] did not find it. But both methods suffer from high run-times.

A variety of algorithms, ranging from linear time heuristics to an evolutionary algorithm will be applied to a set of small and large edge coloring problems. This work will show more simpler and significantly faster heuristic methods to find the optimal number of colors for small (< 500) node numbers. For large node numbers, these heuristics find optimal or near-optimal solutions. To show the quality of these approaches, these results were compared to an EA.

The paper is structured as follows: Section 2 explains basic definitions, properties, and notations. The heuristics are described in Section 3. Section 4 presents the EA that was used for comparison for large graphs. In Section 5 the benchmark selection is explained and results for small and large graphs are shown. Section 6 concludes and summarizes the results.

2. Preliminaries

In this section, basic definitions, properties, and notations concerning the graph edge coloring problem are explained.

- Let $G(N, E)$ be a graph, where its number of edges is designated by $|E|$, the number of nodes by $|N|$.
- Considered are only *simple* graphs, i.e. undirected graphs with no multiple edges between any two nodes, and no edges that have the same node as start and end (loops).
- The *edge multiplicity* μ of a graph is the maximum number of edges between any two nodes. Here only simple graphs are considered, hence $\mu = 1$ in all cases.
- The number of incident edges of a node is called its *degree*. For any graph, its *maximum degree* is symbolized by Δ.
- An edge-coloring of a graph is called $correct$, iff the colors of the edges incident at any node are all different.
- The minimal number of colors necessary to color the edges of a graph $G(N, E)$ correctly is called *edge chromatic number* or alternatively *chromatic index*, denoted χ'. Obviously, the lower bound for χ' is Δ. The upper bound for χ' is known from Vizing's Theorem [11, 17]:

  Theorem 1. $\Delta \leq \chi' \leq \Delta + \mu$,

- A simple graph in which every node is connected by an edge with every other node is called a *complete graph*. In complete graphs, all nodes have maximum degree, and this equation holds:

  Theorem 2. $\chi'_{complete} = \Delta + (|N| \bmod 2)$

  Which means that if we add a node to a complete graph with an odd number of nodes, we need no extra color (see Example 1 below). Hence, $\chi'_{complete}$ is always odd (for $|N| \geq 2$).

- The number of edges of a complete graph is given by

  Theorem 3. $|E|_{complete} = \frac{|N| \cdot (|N|-1)}{2}$

- The ratio δ is called the *density* of the graph.

  Definition 1. $\delta := |E|/|N|$

  The mean degree of a graph is twice the density, since each edge is incident on two nodes. The density is ≥ 1 in all interesting cases. Note that simple complete graphs have maximum density.

- In this paper, graphs that have 500 or more nodes are called *large*.

Edge-coloring problems can be transformed into node-coloring problems by replacing each edge by a node and drawing the new edges according to the graph topology [6].

But this edge-to-node transformation normally comes at the cost of an enlarged size of the graph: after the transformation there are as many nodes as there were edges, and more new edges than before in most cases. Therefore we do not consider this transformation here because of the rise in problem size and hence in run-time.

3. Heuristics

A heuristic usually consists of a simple set of carefully chosen rules that are applied in the search for a desired goal or optimum. For the edge-coloring application, each color assigned by a heuristic should be as near to correct as possible. Moreover, each heuristic should strive to keep the number of colors low because we search for a minimum.

In the following, six algorithms for the edge-coloring problem are presented, namely the heuristics random, cyclic, linE, antN, ant1, and the method symC for the coloring of complete graphs.

Heuristics can be classified by their focus of attention into three types: blind, local, and global.

- A **blind heuristic** disregards its search environment. Examples are the random and cyclic heuristics below which do not consider neighboring edges when assigning a color. The method symC (Section 3.6) can also be put into this class.

- A **local heuristic** only takes note of its immediate surroundings. The linE, antN, and ant1 heuristics in the following sections are examples for local heuristics; they only consider edges in the vicinity of the edge to be colored.

- **Global heuristics** do take the total of the search space into account. For each step in the search they have to look at all parts of the intermediate solution, often involving sorting.

Six heuristics will now be presented. The best three of these will be compared to three different evolutionary algorithms in Section 5.

3.1. Heuristic: cyclic

In the cyclic heuristic, the number of colors to be used is set to the minimal number of necessary colors (Δ). Then the edge with number i in the edge list is colored with color $((i-1) \bmod \Delta) + 1$. (Color and edge numbers start at 1.) The edge order used is identical to the order given in the benchmark file.

The cyclic method is the fastest way of coloring used. It runs in linear time, i.e. $O(|E|)$ (the run-time is proportional to the number of edges).

A disadvantage of this method is that it tries to distribute the colors evenly over all edges, which means that some colorings (which might be optimal colorings) are impossible to generate. Moreover, it is not guaranteed to find a correct solution. The number of colors to be used is fixed to Δ, although the correct χ' may be $\Delta + 1$.

3.2. Heuristic: random

In this algorithm, each edge is visited once and assigned a random color. This color assignment method also runs in linear time ($O(|E|)$) but is slower than the cyclic method by a factor of about two, since the generation of random numbers by the program takes additional time. It is still faster than most other heuristics by orders of magnitude.

The random heuristic is not limited to near-evenly distributed colors like the cyclic heuristic but can generate any coloring, although very uneven color distributions are less probable. The random heuristic is not predictable like cyclic, each solution is different. As in cyclic, the number of colors is fixed; thus the minimum for χ' may be missed and incorrect solutions are often generated.

3.3. Heuristic: linE

The algorithm visits each edge once (in the order given in the benchmark file). It starts at an edge that is incident on a node with maximum degree and proceeds from there towards increasing edge numbers, continuing at the start after the end is reached (wraparound). Thus the name linE because the algorithm linearly visits each edge. The colors of other edges that are incident at start and end nodes of the current edge are read. Considering the colors already present at edge start and end, the current edge is assigned the minimal possible color number.

In this way, each edge is visited many times, once for its own coloring, plus the times for coloring of the other edges at start and end nodes, depending on their degree. Obviously, the mean number of visits for each edge depends on the density δ of the graph to be colored.

We now estimate the run-time of this heuristic by assuming that each node has the same degree, namely the mean degree 2δ. This assumption covers all graphs, even complete graphs, which have maximally possible δ. Therefore this assumption includes the worst case for the run-time T. Now the mean number of neighboring edges ν_{mean} of any edge is the sum of the degrees of start and end node of the edge minus the incidences of the edge itself (using Definition 1):

Lemma 1. $\nu_{mean} \;=\; 4\delta - 2 \;=\; 4(\;|E|\;/\;|N|\;) - 2$

For the coloring of each edge all neighboring edges must be observed. Since we have $|E|$ edges, and using Lemma 1 we get

$$\begin{aligned} T_{linE} &= |E| \cdot \nu_{mean} \\ &= |E| \cdot [4(|E|/|N|) - 2] \\ &= 4|E|^2/|N| - 2|E| \end{aligned}$$

From this, Theorem 3 for the worst case yields

$$T_{linE} \;=\; 2 \cdot |N| \cdot (|N| - 1)^2 - (|N| \cdot (|N| - 1))$$

Omitting the terms of lower order, we arrive at

Theorem 4. $T_{linE} \;=\; O(\;|N|^3)$

The worst case run-time of this heuristic is proportional to the cube of the node number.

3.4. Heuristic: antN

Contrary to linE, which works edge-by-edge, the antN heuristic goes node-by-node. The naming metaphor is an ant that visits each node.

Starting at a node with maximum degree, the antN algorithm visits each node once (following the node numbering given by the benchmark, with wraparound at the end similar to linE). Taking into account the colors already present at the current node, each of the edges incident on that node is assigned the minimal possible color number. Like in the linE method, to find this minimally allowable color number for an edge, the edges incident at the start and end nodes of that edge must be considered.

Estimating the run-time T of this heuristic, we proceed similar as in Section 3.3: We assume the same mean degree 2δ for all nodes, and go over all $|N|$ nodes. For each node, all 2δ incident edges must be considered. Like in linE, all edges incident at each edge's start and end nodes must be checked for their color, which gives a factor of ν_{mean}. Thus, for a single node, $2\delta \cdot \nu_{mean}$ edge visits must be made. Multiplying this by $|N|$ we get the estimate, substituting δ and ν_{mean} using Definition 1 together with Lemma 1.

$$\begin{aligned} T_{antN} &= |N| \cdot 2\delta \cdot \nu_{mean} \\ &= |N| \cdot 2(|E|/|N|) \cdot [4(|E|/|N|) - 2] \\ &= 8|E|^2/|N| - 4|E| \\ &\leq 8|E|^2 \;/\; |N| \end{aligned}$$

Finally, the expression is simplified and Theorem 3 is applied.

$$T_{antN} \;\leq\; 2|N| \cdot (|N| - 1)^2$$

Theorem 5. $T_{antN} \;=\; O(|N|^3)$

That is, in the worst case the antN heuristic run-times should vary like the linE heuristic. Although, the order in which the edges are visited is quite different.

3.5. Heuristic: ant1

This was the first ant algorithm conceived, thus the name. In the ant1 method, a virtual ant visits all nodes and leaves a "scent" each time a node is visited, i.e. the number of visits of the node is increased. The number of allowed colors is limited to Δ. The ant starts at a node with maximum degree Δ and colors all edges differently. Then the next neighbor node is visited. If that node has uncolored edges, these are colored with the smallest color number that is either correct or - if that is not possible - is the smallest of the color numbers that are present at start and end node. If a dead end is met (i.e. a fully edge-colored node), the ant jumps to the next node in the node list that has already been visited and has uncolored edges.

Remark 1. *As a side effect, at the end all nodes not reachable from the start node are now identifiable, because they bear no "scent".*

The above heuristic alone does not always yield a correct solution after its completion. Therefore, linE is invoked afterward to mend wrong color assignments. This may increase the number of edge colors and guarantees a correct solution.

The heuristic ant1 differs from antN in the order in which the nodes are visited. Where antN uses the order given by the benchmark, ant1 employs the neighboring relations of nodes and thus uses a more local approach.

Apart from this locally inspired node ordering, the heuristics antN and ant1 differ only in the additions of scent checking and the search for a new node after a dead end is reached.

Hence, an estimate of the run-time of the ant1 heuristic can be based on the estimate for antN. Only the additions need to be factored in. The scent checking is done for all neighbors of each node, yielding a constant factor, since the neighbor nodes have to be looked at anyhow. The time for the linear search of a new node after a dead end depends on $|N|$. Omitting the constant factor from scent checking, and using Theorem 5, we get for the worst case

Theorem 6. $T_{ant1} \;=\; O(\;|N|^4\;)$

3.6. Method: symC

This algorithm is designed to color complete graphs. It results in a symmetric pattern for an odd $|N|$ and in a near-symmetric form for an even $|N|$. Thus the name from 'symmetric' and 'complete'. This method runs in O($|N|^2$) time, similar to the method described in [3].

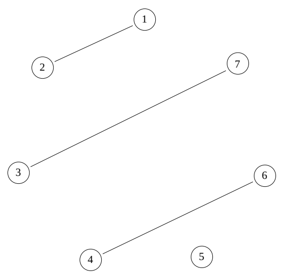

Figure 1. First step in symC coloring

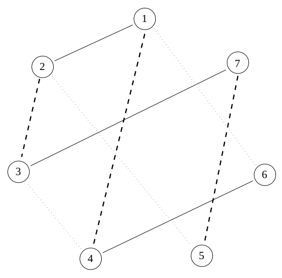

Figure 2. Third step in symC coloring

The algorithm works by assigning the same color to parallel edges. For each color, the start points for the parallels are rotated by one node.

Here is how the node pairs are generated that define these parallel edges (assuming $|N|$ is odd): The first pair consists of the number i of the current start point of the color group and the number $[(i+1)\ mod\ |N|]\ +\ 1$. For each following pair of the group, 1 is subtracted from the left node number of the previous pair, and 1 is added to the right part of that pair. Both node numbers of the new pair are corrected modulo $|N|$ as before if necessary. For each color group, $(|N|-1)\ /\ 2$ edges are drawn.

Note that one node is left out in each group. This node can be connected to an $(|N|+1)$th node with the group color, yielding the solution for the complete graph with $|N|+1$ nodes (an even node number).

Example 1. *How does symC color the complete graph with seven nodes? In symC, the same color is assigned to parallel edges. In the first step, we start with color 1 at node 1. The connected node pairs are 1-2, 7-3, and 6-4 (see Figure 1).*

In the second step, the start point is node 2. The pairs to be connected are 2-3, 1-4, and 7-5. The third step starts at node 3 and the pairs are 3-4, 2-5, and 1-6.

In Figure 2, the first color group of parallels is shown as straight lines, the second color as dashed lines, and the third group as dotted lines.

For odd node numbers, symC yields a rotationally symmetric solution (Figure 3). Now it is possible to add an eigth node without using more colors because at each node a different color is 'missing' (as explained above).

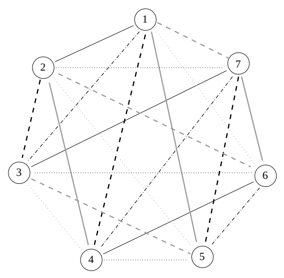

Figure 3. Complete graph colored by symC

4. Evolutionary Algorithm

For the larger graph benchmarks no exact values were available. Thus an EA was employed that the results of the heuristics could be compared to. We built a new EA for the edge coloring of large graphs.

The functions of the GAME library [10] were used for this EA. An object-oriented data structure to represent a large graph was developed.

4.1. Fitness Function

There are two obvious fitness functions for the coloring. First, the number of colors used shall be as low as possible (i.e. χ'). Second, the coloring must be correct, which means that the number of edge faults shall be minimized (with minimum at zero).

The edge faults in the initialization tests below (Section 5.2) were computed in this way: for each node, the number of wrong incident edges was counted. The overall sum of the counts at each node is the number of edge faults wce (= wrong colored edge-ends).

Example 2. *At a node of degree 5, the incident edge color numbers are 1,1,2,2,2. There are only two different colors - the other three edges are wrong. Hence, the wce for this node is 3.*

Since each edge could cause a fault at each of its two ends, an upper limit for wce is $\Lambda = 2|E|$. The ratio of edge faults was computed by dividing the number of edge faults by Λ.

The number of edge faults wce was not used as the only fitness function, although it is a more fine-grained measure and was successful in finding χ' for small graphs. But while wce measures correctness, what is sought is optimality, which is measured by the number of colors. Both functions have to be minimized simultaneously. Therefore the sum of wce and the number of colors was chosen as fitness function.

4.2. Solution Representation

The solution was represented in the EA as the list of all edge colors in the order of the edge numbering. This coding does not guarantee correctness of the solution but it can represent all possible correct solutions. This solution representation has been used in [7].

4.3. Operators and Selection

Two standard operators were used, each with a probability of 50 % during reproduction.

The first operator was mutation which randomly changed one edge color in the list. Roulette-wheel selection was applied to choose the parent for mutation. This means, the better the fitness of a given solution, the larger the probability to be chosen as a parent.

The second operator was a one-point crossing-over. Two different parents were chosen by roulette-wheel selection. The edge list of the generated child started identical to the first parent but continued like that of the second parent after the randomly chosen crossing-over point.

4.4. Reproduction

The population had a size of 35 individuals with 7 children in each generation. This size was chosen as a compromise between speed (small population, few children) and search breadth (large population, many children). Of the resulting population of 42 individuals, only the 35 with the smallest values of the fitness function survived.

If the fitness function of the best solution generated by the EA had not improved after 500 generations, a correction attempt was performed. This attempt consisted of a reinitialization of the worst individual by either the random or the ant1 heuristic starting at a random node. Each heuristic had a probability of 50% of occurring. After that the linE heuristic was applied to assure correctness. This is analogous to the 'kick' operator in the algorithm of [7] and can also be described as a big mutation.

This 'lazy' correction attempts were done for two reasons:

- They were run seldomly to avoid an early cut-off of exploration.

- But correction can lead the EA out of local minima.

The first population was initialized by the heuristics described above to ensure a start population that is better than random. The heuristics cyclic, linE, antN, and ant1 produced one parent individual each. The rest of the parents and the initial children population were generated by the random heuristic.

The EA stops if a fitness function value of Δ (the lower bound for χ') is reached or if either a run-time or a generation limit is exceeded.

5. Benchmarks

Benchmarks were selected for largeness and comparability. Disconnected graphs were excluded from the tests.

The benchmarks used were taken from a list of 119 graphs given at CP2002 [13]. This list also included the DIMACS benchmark graphs [1] which were employed by [7, 15].

Before going into the detailed results of the benchmark runs, some interesting findings from the initialization tests are shown that concern the heuristic coloring of complete graphs.

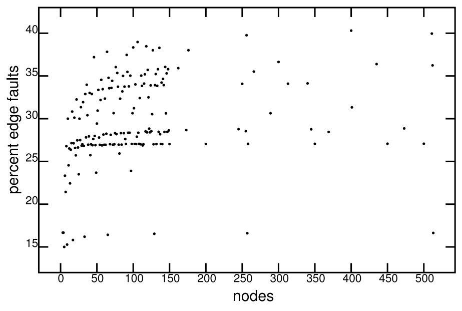

Figure 4. Edge faults of cyclic

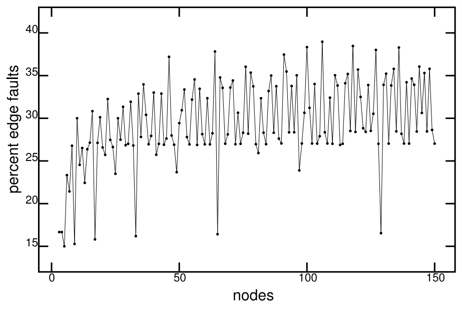

Figure 5. Edge faults of cyclic (enlarged)

5.1. Computing Environment

All tests were run on Linux machines equipped with AMD Athlon XP2200+ processors (1.8 GHz, 256kB cache) and 512 MB RAM. The EA is based on the C++ library for evolutionary algorithms GAME [10]. The graph data structure and its associated methods (including the heuristics) were implemented as a set of C++ classes. Test control programs were written in Python.

5.2. Initialization Tests

Three methods for the initialization (random, cyclic, and ant1) were tested on a wide range of complete graphs with sizes up to 513 nodes. The heuristics linE and antN were not tested on the full range of complete graphs because they needed too much colors in most cases (see Example 4 below).

Although a method to color the edges of a complete graph in linear time exists [3] (see symC above), complete graphs were chosen for initialization tests because they provide maximum density and are easily reproduced.

The edges of these complete graphs can be represented by pairs of different node numbers i-j, in which the smaller number appears on the left. These edge pairs are lexically ordered.

Example 3. *For the complete graph of five nodes, the edges appear in the order 1-2, 1-3, 1-4, 1-5, 2-3, 2-4, 2-5, 3-4, 3-5, 4-5. Edge number 1 is 1-2, number 2 is 1-3, ... and edge number 10 is 4-5.*

This ordering leads to interesting effects in the plots of wce versus node number.

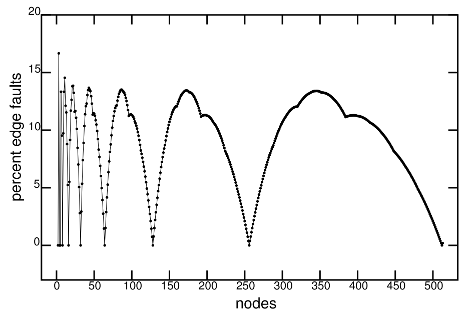

Figure 6. Edge faults of uncorrected ant1

The cyclic heuristic's wce ratio for complete graphs lies in a range between 15 and 40 percent. This is shown in Figures 4 and 5.

In Figure 4 the percentage of edge faults of the cyclic heuristic in the initialization of complete graphs is plotted as a function of node number for the range 3 to 513 nodes. Above 150 nodes only some graphs have been initialized.

In the similar Figure 5 the region from 3 to 150 nodes is enlarged. Here, points are connected with a line that shows the sequence of results for increasing node number.

The percentages show marked horizontal 'bands' of preferred values with minima of 15 to 17 percent for node numbers $2^n + 1$ with $n \geq 2$.

The ant1 heuristic without the linE correction typically has less than 15% wce. Figure 6 shows wce in the initialization of complete graphs as a function of node number for a range of 3 to 513 nodes.

The regular ordering of the edges leads to repeated 'Pão de Açúcar' (sugar loaf) forms, with maxima at graph sizes of 3, 6, 11, 22, 43, 86, 173, and 345 nodes. Up to 512, each maximum appears within ± 1 of the double node number of the previous maximum. There are zero faults if the node number is 5 or a power of 2.

The wce ratio of the random heuristic typically lies at about $1/e$ (36.8%).

The heuristics linE and antN have also been tested on complete graphs. They generate feasible solutions but can increase the color number up to nearly the double of the minimal number. As an effect of the edge ordering, these heuristics find solutions with $2^n - 1$ colors for complete graphs with node numbers $|N|$ from $2^{(n-1)} + 1$ to 2^n , where $n \geq 3$.

Example 4. *Complete simple graphs with 14, 15 or 16 nodes are colored with 15 colors by linE ($n = 4$), but graphs with 17 nodes get 31 edge colors ($n = 5$).*

If a randomized order of the edges is used, the number of used colors normally gets smaller.

5.3. Benchmark Graphs

Table 1 shows the graph qualities for the benchmarks used. If a benchmark name in any table is set in *italics*, this indicates that the graph in question is a multiple-edge graph that was interpreted as a single-edge graph. For these graphs, only one of the occurring edges between two nodes was used, thus avoiding edge multiplicities greater than one and reducing the density δ of the original benchmark. The maximum degree Δ is the lower bound for an estimate of χ'. If the minimal color number χ' is not known, its upper bound is given. The upper half of Table 1 contains benchmarks used in [7, 15] that will be used for comparison in Section 5.3.1. The lower half of Table 1 contains large graphs not considered in these papers. The coloring of these large graphs by heuristics compared to the EA of Section 4 will be shown in Section 5.3.2.

Table 1. Benchmark graph qualities

benchmark	**graph qualities**				
name	nodes	edges	δ	Δ	χ'
games120	120	638	5.32	13	13
david	87	406	4.67	82	82
anna	138	493	3.57	71	71
miles500	128	1170	9.14	38	38
miles1000	128	3216	25.13	86	86
miles1500	128	5198	40.61	106	106
queen11_11	121	2596	18.03	43	43
queen12_12	144	2596	18.03	43	43
queen13_13	169	3328	19.69	48	48
queen14_14	196	4186	21.36	51	51
queen15_15	225	5180	23.02	56	56
queen16_16	256	6320	24.69	59	59
myciel6	95	755	7.95	47	47
myciel7	191	2360	12.36	95	95
le450_5c	450	9803	21.78	66	66
le450_15a	450	8168	18.15	99	99
le450_15c	450	16680	37.07	139	139
le450_15d	450	16750	37.22	138	138
le450_25c	450	17343	38.54	179	179
le450_25d	450	17425	38.72	157	157
DSJC500.1	500	12458	24.92	68	≤ 69
ash331GPIA	662	4185	6.32	23	23
ash958GPIA	1916	12506	6.53	24	24
will199GPIA	701	6772	9.66	38	≤ 39
4-FullIns_4	690	6650	9.64	119	119
5-FullIns_4	1085	11395	10.50	160	160
qg.order30	900	26100	29.00	58	58
qg.order60	3600	212400	59.00	118	118
qg.order100	10000	990000	99.00	198	≤ 199
wap04a	5231	294902	56.38	351	351
latin_square_10	900	202081	224.53	512	≤ 513

5.3.1 Heuristics Performance for Small Graphs

Table 2 gives the results of the heuristics compared to algorithms of other authors, namely the grouping genetic algorithm (GGA) of [15] and the simulated annealing algorithm (SA) of [7]. The original results of the SA had to be adapted, since the SA did not reduce the graphs with edge multiplicity $\mu \geq 2$. Therefore the color numbers for the SA are set in

Table 2. Benchmarks for small graphs

benchmark	**GGA**	**SA**		**linE**		**antN**		**ant1**		**SA / ant1**
name	colors	colors	secs	colors	secs	colors	secs	colors	secs	secs/secs
games120	**13**	(**13**)	2	17	< 0.005	15	< 0.005	15	< 0.005	> 400
david	**82**	(**82**)	1	**82**	< 0.005	**82**	0.01	**82**	< 0.005	> 200
anna	**71**	(**71**)	2	**71**	< 0.005	**71**	< 0.005	**71**	< 0.005	> 400
miles500	**38**	(**38**)	4	**38**	0.01	**38**	< 0.005	**38**	< 0.005	> 800
miles1000		(**86**)	133	87	0.04	**86**	0.04	**86**	0.07	1900
miles1500		(**106**)	171	108	0.13	112	0.13	108	0.24	712
queen11_11	**40**	(**40**)	15	**40**	0.01	**40**	0.01	**40**	0.01	1500
queen12_12	**43**	(**43**)	33	44	0.01	44	0.02	44	0.02	1650
queen13_13		(**48**)	56	**48**	0.02	**48**	0.02	**48**	0.04	1400
queen14_14		(**51**)	96	53	0.03	53	0.03	**51**	0.05	1920
queen15_15		(**56**)	143	57	0.04	57	0.04	**56**	0.08	1788
queen16_16		(**59**)	243	60	0.06	60	0.06	60	0.10	2430
myciel6	**47**	**47**	2	**47**	< 0.005	**47**	< 0.005	**47**	0.01	200
myciel7		**95**	3	**95**	0.02	**95**	0.01	**95**	0.03	100
le450_5c		**66**	93	**66**	0.08	**66**	0.08	**66**	0.15	620
le450_15a		**99**	35	**99**	0.06	**99**	0.07	**99**	0.12	292
le450_15c		**139**	234	**139**	0.40	**139**	0.42	**139**	0.76	308
le450_15d		**138**	231	**138**	0.42	**138**	0.42	**138**	0.78	296
le450_25c		**179**	243	**179**	0.52	**179**	0.52	**179**	0.96	253
le450_25d		**157**	440	**157**	0.53	**157**	0.51	**157**	0.96	458

parentheses for certain benchmarks. Optimal colorings are shown in boldface.

The heuristics perform well but sometimes they do not find the minimum color number. The algorithms linE and antN are almost identical in run-time; for certain benchmarks, however (e.g. games120 and miles1500), they differ markedly in their color numbers.

Especially ant1 does very well in finding the minimal color number. Only in three cases (games120, miles1500, and queen12_12) it is off by at most two colors.

The heuristic ant1 is distinctly faster than the SA of [7]. This will now be discussed in detail.

The SA algorithm run-times were expected to be longer because of a different hardware. In [7] a 700 MHz machine was used with Java[1], the heuristics of this article were written in C++ and tested on a 1800 MHz machine.

If the clocking frequency of the processors is considered as a measure of the processor speed, the run-time ratio SA / ant1 would be about 2.6, provided SA was as fast as ant1. Using the SPEC results [2] as run-time ratio, we get $738/230 \approx 3.2$. Thus we expect a run-time ratio of about 3 if the algorithms are comparable.

But the run-time ratio in the last column of Table 2 shows that ant1 is generally much faster than SA; the run-time ratios for the 20 benchmarks vary between 100 and 2430, with a median value of 539 and a mean of 881.

Because the results of the heuristics linE, antN, and ant1 were near-optimal (i.e. $\approx \Delta$) in most cases, the EA was not run - it could only confirm or improve little on the solutions found by the heuristics at the cost of high run-times.

[1]It is known that Java runs slower than C++ by a factor between 3 and 30, depending on the benchmarks used for comparison [5, 9].

5.3.2 Heuristics Compared to EA for Large Graphs

Table 3 gives results for larger graph sizes compared to the EA described in Section 4. These benchmarks were not tested in [7, 15] (where only small graphs are considered), hence the GGA and SA columns are omitted. These benchmarks were chosen to show the usefulness of the heuristics for large graphs.

The linE and antN heuristics almost always differ more from the minimum than ant1. Also obvious are near-identical run-times of linE ant antN. However, in the benchmarks with the largest node numbers (qg.order60 and qg.order100), antN runs significantly longer than linE.

For these large graphs, only the ant1 heuristic really excels. In two cases (qg.order30 and qg.order60) it finds the minimum. In these cases the EA was not run, since it uses ant1 for initialization and hence immediately stops. The EA was run in cases when there was room for improvement. Only in three of these benchmarks the EA could improve on the near-minimal solutions found by the ant1 heuristic. In all other cases, the color numbers found are identical or come close to those of the EA. Note the enormous run-times of the EA compared to the speed of ant1.

Since χ' was not known for DSJC500.1, the upper bound found by ant1 and our EA is not set in boldface.

Table 3. Benchmarks for large graphs

benchmark	**linE**		**antN**		**ant1**		**EA**	
name	colors	secs	colors	secs	colors	secs	colors	secs
DSJC500.1	80	0.13	80	0.14	69	0.25	69	158043.00
ash331GPIA	25	< 0.005	26	0.01	**23**	0.01		
ash958GPIA	**24**	0.01	**24**	0.07	**24**	0.07		
will199GPIA	45	0.02	43	0.01	40	0.03	40	119038.00
4-FullIns_4	120	0.03	121	0.04	120	0.07	**119**	2332.60
5-FullIns_4	161	0.09	163	0.09	161	0.16	**160**	38117.30
qg.order30	60	0.29	60	0.33	**58**	0.59		
qg.order60	122	9.27	122	10.21	**118**	17.94		
qg.order100	226	134.55	226	140.65	212	248.38	212	143542.00
wap04a	**351**	24.98	**351**	26.25	**351**	46.75		
latin_square_10	733	147.15	742	147.28	554	268.85	530	1048789.79

5.4. Complete Graphs

Although the symC heuristic currently runs only in O($|N|^4$) time because of implementation details, a complete graph of 200 nodes and 19900 edges is correctly colored with the minimal number of colors in 5.05 seconds. That is nearly three orders of magnitude faster than the simulated annealing (SA) algorithm of [7] which needed 3341 seconds for this graph. A complete graph with 512 nodes and 131328 edges is colored in 220 seconds by symC.

6. Conclusions

In this paper, heuristics for the edge coloring of graphs were explored using a set of standard benchmarks.

From the known benchmarks, 20 small graphs and 11 large graphs plus 175 complete graphs were edge-colored by five different heuristics. Compared to two EAs of recent works and a newly designed EA, the speed and small color numbers of the better heuristics are remarkable.

Below benchmark graph sizes of 500 nodes, the ant1 heuristic yields minimal solutions for graph edge coloring in almost all cases considered. In the cases where ant1 does not find the optimum, it differs by at most two colors.

The heuristic ant1 is faster than the SA employed in [7] by orders of magnitude. Although ant1 is slower than the heuristics linE and antN, its run-time typically differs only by a constant factor of 1.8 from antN, even for large graphs.

For 500 or more nodes the ant1 heuristic finds solutions that are optimal or differ not much from the color numbers found by our EA.

7. Acknowledgements

The authors like to thank Hans-Jörg Kreowski for helpful discussions.

References

[1] *DIMACS Center for Discrete Mathematics and Theoretical Computer Science ftp://dimacs.rutgers.edu/ pub/ challenge/ graph/ benchmarks/ color/.* Rutgers University, Piscataway, New Jersey, 1993.

[2] All spec cpu2000 results published by standard performance evaluation corporation. *http:// www.specbench.org/ cpu2000/ results/ cpu2000.html*, 2003.

[3] C. Berge. *Graphs and Hypergraphs.* North Holland, 1973.

[4] E. Caspi, R. Huang, and C. Kozyrakis. On detailed routing for a hierarchical scalable reconfigurable array with constrained switching capability. *CS-270 Course Project http://www.cs.berkeley.edu/ ~eylon/ cs270/*, 1998.

[5] M. Connell. Python vs. perl vs. java vs. c++ runtimes. *http://www.flat222.org/ mac/ bench/*, 2002.

[6] R. Diestel. *Graph Theory, Electronic Edition, ftp:// ftp.math.uni-hamburg.de/ pub/ unihh/ math/ books/ diestel/ GraphTheoryII.pdf.* Springer, New York, 2000.

[7] B. Enochs and R. Wainwright. 'forging' optimal solutions to the edge-coloring problem. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pages 313–317. Morgan Kaufmann Publishers, San Francisco, 2001.

[8] S. Fiorini and R. Wilson. *Edge-colourings of graphs. Research Notes in Mathematics 16.* Pitman, London, 1977.

[9] E. Galyon. C++ vs java performance. *http:// www.cs.colostate.edu/ ~cs154/ PerfComp/*, 1998.

[10] N. Göckel, R. Drechsler, and B. Becker. GAME: A software environment for using genetic algorithms in circuit design. In *Applications of Computer Systems*, pages 240–247, 1997.

[11] R. Gupta. The chromatic index and the degree of a graph. *Not. Amer. Math. Soc.*, 13:719, 1966.

[12] I. Holyer. The np-completeness of edge coloring. *SIAM Journal of Computing*, 10:718–720, 1980.

[13] D. Johnson, A. Mehrotra, and M. Trick. Graph coloring instances, constraint programming 2002. In *http://mat.gsia.cmu.edu/COLOR03/.*

[14] A. Kapoor and R. Rizzi. Edge coloring bipartite graphs. *University of Trento, Technical Report # DIT-02-0040, http:// eprints.biblio.unitn.it/ archive/ 00000120/ 01/ 40.pdf*, 1999.

[15] S. Khuri, T. Walters, and Y. Sugono. A grouping genetic algorithm for coloring the edges of graphs. In *Proceedings of the 2000 ACM symposium on Applied computing 2000, Como, Italy.* ACM Press New York, NY, USA, http://portal.acm.org/citation.cfm?id=335880&coll=portal &dl=ACM&ret=1.

[16] S. Skiena. *The Algorithm Design Manual http:// www2.toki.or.id/ book/ AlgDesignManual/ BOOK/ BOOK4/ NODE179.HTM#17445.* Springer, New York, 1993.

[17] V. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3:23–30, 1964.