

A Range-compactness Heuristic for Graph Coloring

Andrea Di Blas¹ Arun Jagota Richard Hughey

Department of Computer Engineering
University of California, Santa Cruz
1156 High St. Santa Cruz 95064 CA

February 22, 2005

¹Corresponding author: andrea@soe.ucsc.edu

Abstract

This paper presents a novel heuristic for graph coloring that works on a range of colors and iteratively tries to make this range more compact. This range-compaction heuristic also has a “pressure” component and an annealing schedule for it. The value of this component is empirically quantified. This algorithm is evaluated on a wide range of DIMACS benchmark graphs, and found to be competitive with state-of-the-art algorithms in terms of solution quality and run time.

Keywords: graph coloring, iterative-greedy algorithm, DIMACS benchmarks

Introduction

Given an undirected graph $G = \{V, E\}$, where $V = \{v_i, i = 1 \dots N\}$ is the set of vertices and $E = \{e_{ij}\}$ is the set of edges, a k -coloring of G is a partition of V into k sets (*color classes*) $V_1 \dots V_k$ such that $\forall e_{ij}, v_i \in V_l \Rightarrow v_j \notin V_l$. The *chromatic number* $\chi(G)$ is the smallest value of k such that G is k -colorable.

Coloring all nodes in a graph G using exactly $\chi(G)$ colors is referred to as the *minimum coloring* problem, and is known to be NP-hard [GJ79]. The graph coloring problem is interesting not only because it is computationally intractable, but also because it has numerous practical applications, for instance: register assignment in code compilation [GC93], several CAD problems [De 94, GDWL92], timetabling [de 85], and frequency channel assignment [RL93, SP97]. For these reasons, the graph coloring problem has been addressed by a number of approaches over the past few decades. However, no one approach has turned out to be a clear winner.

This paper presents a novel heuristic method for the minimum coloring problem. The method starts with a certain proper coloring and iteratively tries to narrow the range of colors used by up-coloring low-valued colors and down-coloring high-valued ones. This method has some randomization built into it. This, coupled with its iterative nature, allows the solution quality to improve as the running time is increased.

In this paper, we compare this method with state-of-the-art methods for graph coloring. The experimental results show that the solution quality is comparable and sometimes better than results for state-of-the-art algorithms, with a similar amount of run time.

1 Related work

The graph coloring problem is known to be a very difficult NP-hard problem, and has been attacked in many different ways. In this section we give a brief overview of some of the major approaches.

The graph coloring problem can be solved exactly with a branch-and-bound or other exhaustive search algorithm. Coudert presents a branch-and-bound algorithm that uses, for pruning, the size of a clique in a graph to lower-bound its chromatic number [Cou97]. He also gives a branch-and-bound algorithm to find a maximum clique in an arbitrary graph. In a *1-perfect* graph — a graph in which the chromatic number equals the size of the maximum clique — he uses this combination of algorithms to find an optimal coloring very efficiently, as he claims and demonstrates, as follows. He first uses the clique algorithm to find the largest clique in the graph, and then uses the size of this clique as a lower bound on the chromatic number, for his graph coloring algorithm. While this appears to work well for 1-perfect graphs, it is obvious that there is no hope of it working well on arbitrary graphs. It is also worth noting that computing the chromatic number of 1-perfect graphs is in P [Knu94].

The approach of Coudert — or any other exact algorithm — can not possibly work efficiently on large graphs with arbitrary connectivity. One can choose to truncate the search early, but then one is not assured of retrieving an optimal coloring. The well-known heuristics DSATUR [Bre79] and *Recursive Largest First* (RLF) [Lei79] may be

regarded as truncated branch-and-bound algorithms.

Apart from the issue of whether a search algorithm is exact or approximate, is the issue of the order in which nodes are picked. The choice of ordering mechanism can greatly impact the run time of the algorithm in either case. The ordering mechanism can be static or dynamic. The algorithm by Welsh and Powell, for example, statically pre-orders the nodes according to their degree [WP67]. DSATUR and *Recursive Largest First* instead perform a dynamic reordering of nodes based on their degree and on the colors used by their neighbors.

The RLF heuristic has spawned other algorithms. The algorithm proposed by Costa and Hertz embeds the RLF idea into an evolutionary algorithm that is based on the real-life model of the behavior of ants [CHD95, CH97]. Though interesting, this algorithm has been shown by Vesel and Žerovnik [Vv00] to be less effective in practice than an extension of the antivoter algorithm by Petford and Welsh [DW84, PW89], which is a randomized algorithm based on a statistical mechanics model of a particle system. Kirovski and Potkonjak propose an algorithm called “least-constraining most constrained eXtended RLF” (ImXRLF) that tries to improve the solution quality by combining many different methods and heuristics [KP98].

General optimization techniques of different kinds have also been tried on the graph coloring problem. Approaches based on simulated annealing include the one by Johnson et al. [JAMS91] and the one by Chams et al. [CHdW87]. Two evolutionary formulations of the graph coloring problem have been proposed by Barbosa et al. [BA04]. Joslin and Clements offer experimental results on the graph coloring problem of their “Squeaky Wheel Optimization” algorithm, a general approach to optimization based on repetitively finding a solution with a greedy method, analyzing the solution found, and re-formulating the problem in such a way that the following application of the greedy solver is likely to find a solution of better quality [JC99].

Neural network-based approaches also abound. The earliest efforts were focused on solving the 4-colorability problem on planar graphs [Dah87]. Philipsen and Stok [PS91] transformed the graph coloring problem into the maximum weight stable set problem, and to enforce feasibility of the solution they used a previous approach by Peterson and Soderberg based on the Mean Field Theory Potts Neuron [PS89]. Takefuji and Lee used a Hopfield network with binary neurons to solve the four- and k-coloring problem on planar geographical maps [TL91]. Their algorithm was later modified by Gassen and Carothers [GC93] to deal with the more general problem of k-coloring an arbitrary graph, and applied it to register allocation optimization. Another extension to the k-coloring problem of the Takefuji and Lee algorithm was later devised by Berger [Ber94]. Later, one of us (Jagota [Jag96]) proposed a reduction of the coloring problem to the Maximum Independent Set problem, followed by a mapping onto a Hopfield network. More recently we have proposed a different energy-function formulation based on a “quasi-Hopfield” neural network that encodes the color of a vertex in the state of a single neuron [DJH02]. The latter formulation is much more compact than the former. A rather peculiar network-based algorithm that also encodes the color of a vertex in the state of a single cell is described in the work by Wu, where an array of coupled oscillators is considered as a graph by associating oscillators to vertices and couplings to edges [Wu98].

Other older and newer approaches include: the iterated-greedy algorithm by Culberson and Luo [CL96]; the tabu branch-and-bound algorithm by Glover et al. [GPR96]; the tabu search algorithms by Hertz and De Werra [HdW87] and by Dorne and Hao [DH98]; the distributed neighborhood search algorithm by Morgenstern [Mor96] and the one by Lewandowski and Codon [LC96]; the algorithm based on linear programming by Karger et al. [KMS94] and the one by Mehrotra and Trick [MT96]; and the object-oriented implementation by Fleurent and Ferland [FF96].

The algorithm presented in this paper can be considered an iterated-greedy algorithm. It obviously shares some basic ideas with many of the above-cited methods. However, to the best of our knowledge, our particular formulation is unique.

In Sec. 3 we offer a comparison of our experimental results with those of some of the most famous graph coloring algorithms, and for which the published results include tests on the official benchmark graphs from the 1993 DIMACS Graph Coloring Challenge [JT96, Hom], or for which we could obtain the source code and run the experiments ourselves.

2 Algorithm description

The algorithm starts from any proper coloring of the graph — for instance, the coloring that assigns color i to vertex i — and works in “sweeps”, iteratively trying to improve the coloring by narrowing the range of colors used. Let Q_L and Q_H denote the lowest and highest values of colors used in the coloring immediately before the start of a sweep. There are two types of sweeps: up-sweeps and down-sweeps, and they are performed alternatively.

A down-sweep works as follows. While there is a vertex whose color value can be lowered to a value that is no less than a certain bound $Q_{L\text{limit}} \geq Q_L$ while maintaining the properness of the coloring, this vertex is recolored with the lowest such value.

This quantity $Q_{L\text{limit}}$ is set to $Q_L + \alpha_t$, where α_t is a nonnegative integer. Here t represents the sweep number, and α_t follows a schedule that we describe later. When α_t is chosen to be greater than zero, this has the effect of exerting “pressure”, as illustrated in Fig. 1. A down-sweep thus tends to shift the coloring of the graph toward low color values. The aim of applying pressure is to, additionally, prevent the number of nodes in the lowest α_t color classes from increasing, in the hope that in the following up-sweep nodes in these classes will be easier to recolor. Note that when a node is recolored, some lower-valued color might become available to recolor some other node in its neighborhood. So nodes can be recolored more than once during a single sweep. When no node is found that can be recolored in this way, the down-sweep ends. At this point, the highest color value still used in the graph, Q_H , is certainly not greater than it was before the down-sweep, and may very well be less.

Now the value Q_H is used to set the high coloring limit, $Q_{H\text{limit}}$, for the up-sweep that follows. $Q_{H\text{limit}}$ is set to $Q_H - \alpha_t$, where $\alpha_t > 0$ corresponds to applying pressure. The up-sweep works in the exact same way as the down-sweep, except that nodes are recolored with the highest possible color that maintains a proper coloring and is not greater than

$Q_{H\text{limit}}$. When no such recoloring is possible, a new value for Q_L — the lowest color still used — is found, $Q_{L\text{limit}}$ is set accordingly, and a new down-sweep starts. Figure 2 shows the pseudo-code for the algorithm.

Repeating this process tends to shift the lowest color used to higher values, and the highest color used to lower values, thereby reducing the total number of colors used. The best coloring found is always the last one, since the number of colors can never increase, as stated in Prop. 1.

We call a coloring *gapless* if all colors in the range Q_L to Q_H are used.

Lemma 1 *When a sweep terminates, the coloring is gapless in the sense that all colors in the range Q_L to Q_H are used.*

Proof: Suppose there is a gap in the coloring at the end of a sweep. Let c be an unused color. Color c is clearly greater than Q_L and less than Q_H , by definition of Q_L and Q_H .

First consider the case when the sweep is an up-sweep. There is a node v whose present color $c(V)$ has a value less than c . Since c is unused, v can be recolored with c , contradicting that the sweep has ended.

The proof is similar for the case of a down-sweep. □

Proposition 1 *The number of colors used after a sweep is no greater than the number of colors used before the sweep.*

Proof: This is an immediate corollary of Lemma 1 and the fact that the range of used colors can be no wider after a sweep than before it. □

Like in other stochastic optimization approaches, the more times this process is repeated, the higher the probability of getting a solution of better quality. In our implementation, the stopping criteria that the user can specify are the maximum number of iterations, the desired coloring size, or both. Figure 3 shows an example of coloring progress on a sample graph (flat1000_50_0) where the higher color value, Q_H , the lower color value, Q_L , and the total number of colors used, Q , are reported as a function of the run time (in number of sweeps). In this case, after a sufficiently large number of sweeps (about 60000, or $60N$) we actually reach the optimum coloring, $\chi = 50$. This behavior — that solution quality keeps improving as the number of sweeps is increased — was observed in quite a few other graphs, but not in the majority of them (Fig. 4).

The algorithm’s name, Range-Compaction Coloring, is chosen because in each sweep the method tries to make the range of colors used more compact. Moreover, it does this in alternating down- and up-sweeps, which is reminiscent of the *shakesort* sorting algorithm, with the added feature of pressure.

Pressure schedule

One possibility is to use no pressure at all, i.e. set α_t to zero. One would expect a non-zero pressure to work better, however, and as we will see in the experimental results section, this turns out to be the case. At the other extreme, a high constant pressure will typically not work either. The high pressure will usually prevent the range of colors

from becoming narrower from sweep to sweep by limiting the choices of colors a node can be recolored with. A sensible middle ground is to initially start with a high pressure in the hope of quickly “squeezing out” a compact coloring, should one be easy to find, and then relax the pressure if this does not happen, to try to at least make more conservative improvements. We found this scheme to work well in our experiments, with the particular choices $\alpha_t = 2, t = 0, \dots, \approx N/10$; $\alpha_t = 1, t \approx N/10 + 1, \dots, \approx N$; $\alpha_t = 0, t > \approx N$. This is the default pressure schedule we use. Other pressure schedules could be investigated. For instance, adding noise to the pressure schedule might help getting unstuck from local minima. At the present time, however, we have not tried this.

Variants

We now look at other variants of our approach. One part of the algorithm where it is easy to come up with different heuristics is the way nodes are selected. In the following we will describe the ones that we experimented with, and present a quantitative comparison in the experimental results section. We evaluated these variants not only to find if some might work better, but also to get a better sense for how good our original method was by evaluating other possibilities. We found that, of the four variants we evaluated, only two yielded a better solution quality in some cases, but at the expense of a much longer run time. On average, the algorithm implementing the random selection had the best solution quality for a reasonable run time.

The node-ordering heuristics that we evaluated are to select nodes: randomly (S1), in non-increasing degree order (S2), in order from high- to low-valued color in present coloring (S3), in order from least-used color to most-used color in present coloring (S4), and in order from most-used color to least-used color in present coloring (S5).

3 Experimental results

Our programs were written in C, compiled with the GNU `gcc` compiler (-O3 optimization) and run under Solaris 8 on a SUN Blade 100 workstation (Sun UltraSPARC-IIe at 500 MHz) with 640 MB RAM. The programs were tested on the official benchmarks from the 1993 DIMACS graph coloring challenge [Hom, JT96]. The graphs in the tables include several random graphs (DSJCx.y), geometric random graphs (DSJRx.y and Rx.y[c]), “quasi-random” graphs (flatx.y_0), graphs from real-life applications (fpsol2.i.x, inithx.i.x, multsol.i.1, and zeroin.i.1 map register allocation problems, school1 and school1_nsh are class scheduling graphs), and artificial graphs (lex_y). This specific subset of the complete set of DIMACS coloring benchmarks was chosen to maximize the overlap with the published results of other methods reported in Table 3 since none of them offers results on the complete set. The values for the chromatic number are in parentheses when they are the lowest value reported in the literature but not proven to be the minimum.

Comparison of the different variants

Table 1 compares solution quality and timings of the five node-selection heuristics. In all cases, the total number of sweeps (up and down counted separately) was set to $10N$, where N is the number of nodes in each graph, and pressure was used as described earlier. The results reveal the following: S1 runs the fastest; S2 returns slightly better solutions than S1 but is twice as slow; S3, S4, and S5 are not competitive with S1 or S2 neither in solution quality nor in run time. In view of the above, we restrict ourselves to S1 from now on.

Graph		S1		S2		S3		S4		S5	
name	χ	k	t [s]	k	t [s]	k	t [s]	k	t [s]	k	t [s]
DSJC125.5	(17)	19.3	0.23	19.0	0.34	20.0	0.32	22.0	0.44	20.0	0.59
DSJC250.5	(28)	32.4	1.82	32.6	3.01	35.0	3.46	38.0	5.14	33.0	6.14
DSJC500.5	(48)	55.6	12.08	55.7	24.10	62.0	38.58	63.0	39.37	57.0	45.32
DSJC1000.5	(84)	97.2	185.73	97.2	340.06	108.0	554.07	109.0	584.85	99.0	580.20
DSJR500.1	12	12.0	7.60	12.0	15.11	12.0	79.48	14.0	93.37	13.0	145.22
DSJR500.5	(123)	130.9	36.30	129.8	119.39	129.0	164.81	138.0	136.29	128.0	114.52
R125.1	5	5.0	0.22	5.0	0.31	5.0	0.67	5.0	0.59	5.0	1.05
R125.1c	46	46.0	0.47	46.0	0.59	48.0	0.78	46.0	0.53	47.0	0.78
R125.5	36	37.2	0.32	37.0	0.81	37.0	0.78	41.0	1.02	38.0	0.85
R250.1	8	8.0	1.47	8.0	2.52	8.0	10.71	9.0	10.27	8.0	14.84
R250.1c	64	64.0	5.02	64.0	5.27	66.0	7.70	67.0	10.65	67.0	10.67
R250.5	65	69.6	3.75	68.6	9.84	68.0	13.57	74.0	13.46	68.0	10.90
R1000.1	20	21.0	65.51	20.70	137.59	21.0	856.70	24.0	1093.84	20.0	1165.84
R1000.5	(238)	255.5	585.79	254.8	2701.36	248.0	2427.13	267.0	2273.88	250.0	1789.28
flat300_20_0	20	20.3	1.81	20.0	3.18	39.0	6.78	41.0	10.4	32.0	8.95
flat300_26_0	26	36.1	3.06	36.0	4.70	39.0	6.99	40.0	9.00	36.0	9.14
flat300_28_0	28	36.2	3.12	36.1	4.89	39.0	7.33	40.0	7.49	36.0	8.38
flat1000_50_0	50	95.1	191.63	95.4	345.79	107.0	522.54	107.0	498.46	96.0	520.76
flat1000_60_0	60	95.4	180.93	95.7	333.99	105.0	447.86	108.0	616.335	96.0	523.04
flat1000_76_0	76	96.5	195.76	96.0	330.03	105.0	407.23	108.0	534.63	99.0	520.90
le450_15a	15	17.6	7.17	17.2	22.19	18.0	59.43	20.0	53.79	18.0	79.83
le450_15b	15	17.3	6.64	17.4	23.67	17.0	53.01	19.0	44.13	17.0	61.87
le450_15c	15	23.2	8.18	23.1	16.00	25.0	51.38	27.0	47.60	21.0	47.62
le450_15d	15	23.3	8.26	23.3	15.84	24.0	33.69	27.0	38.79	24.0	55.01
musol.i.1	49	49.0	1.88	49.0	37.18	49.0	8.41	49.0	5.52	49.0	9.19
school1	14	14.0	1.58	14.0	9.59	15.0	10.58	26.0	21.29	21.0	26.42
school1_nsh	14	14.2	1.60	14.1	7.62	24.0	22.19	30.0	24.78	14.0	15.59
Average of the above		k/χ	$t[s]$	k/χ	$t[s]$	k/χ	$t[s]$	k/χ	$t[s]$	k/χ	$t[s]$
		1.178	6.648	1.174	15.370	1.289	25.596	1.404	27.959	1.222	31.026

Table 1: A comparison of the different selection heuristics. Times are in seconds and all values are averaged over 10 different experiments. In each case the total number of sweeps (up and down counted separately) was set to $10N$. Average values: k/χ is the arithmetic mean of the solution quality (the size of the coloring found with respect to the chromatic number), and $t[s]$ is the geometric mean of run times, to minimize the effect of large variances and to preserve the performance ratio among all timings.

Pressure versus no pressure

To quantify the value of using pressure in our algorithm, we also compared results with a version without pressure. On the same graph set of Tab. 1, using the random selection

function S1, and with the same number of up- and down-sweeps ($10N$), not applying pressure gives an average solution quality $k/\chi = 1.181$ and an average run time $t = 7.976s$. Comparing these numbers with the average values shown in Tab. 1, column S1, we find that applying pressure marginally improves solution quality but speeds up the run time by almost 20%. The benefits of applying pressure become more apparent when a lower number of sweeps is allowed. When allowing $5N$ sweeps, for instance, we found that the average solution quality was 2% higher with pressure, and run time was decreased by more than 23%. This suggests that if one wants a solution fast, use pressure.

Solution quality versus run time

Our algorithm has a parameter, the number of sweeps, which allows one to control the tradeoff between solution quality and run time. In this section, we report experiments that explore this tradeoff. We run RCC on DIMACS graphs. On each, we run RCC for $100N$ sweeps (up and down counted separately) and plotted the evolution of the solution quality as a function of sweeps/ N . Figure 4 displays these curves, one per graph, grouped by graph class. The initial trivial coloring that uses N colors is not shown in the plots. We start plotting after the first (down) sweep. When a curve stops and its solution quality is greater than one, this means that there was no further improvement from where it stopped to $100N$.

From these plots it is clear that in many cases the solution quality keeps improving as the number of sweeps is increased. The plots also reveal that $10N$ is a reasonable default setting for the sweeps parameter — in most curves, most of the solution improvement happens before one reaches this number.

Comparison with other methods

We offer two sets of comparisons of our algorithm with other algorithms. In one set, comparisons are based on algorithms that have all been compiled and run under the same conditions. In the other set, we compare our results with those of published works.

Table 2 — which corresponds to the first set — compares our implementation with the experimental results from other well-known graph coloring algorithms. We took the code for all these from Joe Culberson’s web page [Cul]. All programs were compiled and run under the conditions described at the beginning of this section.

“RCC” is our Range-Compaction Coloring described in this paper, as in version S1 with pressure (Tab. 1). The only parameter the user has to set (besides a “seed” for the random number generator) is the total number of sweeps before stopping. Based on the experimental results on a broader data set (Fig. 4), we opted for a uniform setting of $10N$ total sweeps for all graphs, as mentioned earlier.

“MAXIS” is an evolution by Joe Culberson of the algorithm by Bollobás and Thomason that colors a graph by finding MAXimal Independent Sets using a depth first search [BT85]. It is known to work well with random graphs with $p \sim 0.5$. For other graph instances — like sparser graphs, for instance — the user has to be careful with the parameter settings or the run time will increase by orders of magnitude. The parameter settings

that we used are: no cheat; random seed = 1234; Vertex Num, Cutlim = 0, 1; Backtrack Limit, UpSort Limit, MidSort Limit = 1, 66, 33.

“G” is the greedy algorithm, also known as the “sequential” algorithm. Here, multiple heuristics can be applied to the order in which vertices are selected and to the color assignment when there is more than one choice. The parameter settings that we used are: no cheat; random seed = 1234; Greedy Type Selection = simple greedy; Initial Vertex Ordering = inorder; Use Kempe Reductions = no.

“IG” is the “Iterated Greedy” algorithm by Culberson and Luo [CL96]. This program starts from any coloring and repeatedly calls the greedy program to find a new coloring. Each new coloring is guaranteed to have no more colors than the previous one. The possible improvement comes from a re-ordering of the vertices based on the previous coloring. The parameter settings that we used are: no cheat; random seed = 1234; Target color = 1; weight for Simple Greedy = 100; weight for Largest First Greedy = 100; weight for Smallest First Greedy = 0; weight for Random Sequence Greedy = 50; weight for Reverse Order Greedy = 0; weight for Stir Color Greedy = 0; frequency for Kempe Reductions = 30; number of iterations before quitting = 1000; reorder control information = 100 1, 100 2, 50 5; no starting coloring.

“TABU-C” is the tabu search algorithm by Hertz and De Werra [HdW87] as modified and improved by Culberson and Luo [CL96]. It is a local improvement search based on partitioning the vertices into color classes that may not represent a legal coloring, and then attempting to reduce the number of coloring violations by moving vertices from a class to another. The parameter settings that we used are: no cheat; random seed = 1234; maximum iterations = N ; number of neighbors = N ; minimum number of neighbors = 2; tabu list size = 7; target coloring = 2; increases allowed = N ; no starting coloring.

“DSATUR” is the sequential algorithm with the added heuristic that the order in which vertices are chosen is determined dynamically based on the vertices’ color saturation (the number of colors that cannot be used to properly color a vertex). It was first described in the work by Brélaz [Bre79]. The parameter settings that we used are: no cheat; random seed = 1234; random initial vertex ordering.

What we consider a convenient feature of our algorithm is that it requires just a single parameter, the number of sweeps. Of the other programs, only DSATUR requires just a single parameter, the initial vertex ordering. MAXIS, TABU, and especially Iterated Greedy all require quite a few control parameters. Performance on specific graph instances or graph classes can therefore be improved on a case-by-case basis, with multiple retries with different parameter settings. In our experiments, though, we selected a fixed set of parameters for all instances for each program — in TABU-C these were a function of the graph size — as we did for RCC.

Under these assumptions, from the results in Tab. 2 we observe the following. Three algorithms, MAXIS, G, and DSATUR, are faster than RCC by 2 orders of magnitude, but their average solution quality is clearly poorer. IG’s solution quality is almost identical to RCC’s on average — even though it can be much better or much worse in specific cases — for about half as much run time, on average. TABU-C’s solution quality is almost consistently better than RCC’s, but at the price of four times as much computation.

Table 3, which corresponds to the second set of comparisons, compares RCC with published results of other graph coloring algorithms. Since the programs were run on different machines, we normalized their timings to ours using the benchmark program `dfmax` provided by DIMACS [DIM]. Timing comparisons therefore can be only approximate, since algorithms different from `dfmax` on different machines do not necessarily scale in the same way. Despite this limitation, the combination of the reported solution quality and run times does give the reader a sense of how these algorithms compare to ours.

Graph		RCC		MAXIS		G		IG		TABU-C		DSATUR	
name	χ	k	t [s]	k	t [s]	k	t [s]	k	t [s]	k	t [s]	k	t [s]
DSJC125.5	(17)	19.3	0.23	22	0.01	26	0.01	19	0.54	19	0.87	21	0.01
DSJC250.5	(28)	32.4	1.82	37	0.01	43	0.01	33	2.54	31	9.20	37	0.01
DSJC500.5	(48)	55.6	12.08	61	0.11	72	0.01	58	13.6	53	72.24	67	0.02
DSJC1000.5	(84)	97.2	185.73	107	0.68	127	0.02	106	29.93	93	912.35	114	0.09
DSJR500.1	12	12.0	7.60	14	0.13	15	0.01	12	3.77	12	29.91	12	0.01
DSJR500.5	(123)	130.9	36.30	140	0.18	143	0.01	129	13.19	131	107.89	126	0.02
R125.1	5	5.0	0.22	5.0	0.01	5	0.01	5	0.36	5	0.17	5	0.01
R125.1c	46	46.0	0.47	47	0.01	51	0.01	46	0.45	46	1.70	46	0.01
R125.5	36	37.2	0.32	38	0.01	44	0.01	37	0.63	39	1.32	38	0.01
R250.1	8	8.0	1.47	8	0.02	9	0.01	8	0.60	8	2.42	8	0.01
R250.1c	64	64.0	5.02	68	0.04	76	0.01	64	1.32	65	10.17	66	0.01
R250.5	65	69.6	3.75	75	0.04	79	0.01	69	1.45	71	11.90	69	0.01
R1000.1	20	21.0	65.51	25	0.75	26	0.03	20	20.42	20	382.83	21	0.05
R1000.5	(238)	255.5	585.79	271	1.36	275	0.03	254	52.80	259	1060.85	253	0.11
flat300_20_0	20	20.3	1.81	38	0.05	47	0.01	20	1.18	26	12.74	41	0.01
flat300_26_0	26	36.1	3.06	40	0.02	45	0.01	38	1.61	34	13.27	43	0.01
flat300_28_0	28	36.2	3.12	41	0.03	46	0.01	38	2.25	33	13.84	43	0.01
flat1000_50_0	50	95.1	191.63	104	0.71	126	0.02	50	74.16	90	783.64	115	0.09
flat1000_60_0	60	95.4	180.93	103	0.74	125	0.02	104	42.80	91	864.22	115	0.09
flat1000_76_0	76	96.5	195.76	103	0.72	122	0.04	105	23.33	91	917.59	113	0.10
le450_15a	15	17.6	7.17	21	0.07	22	0.01	18	1.63	17	41.12	17	0.01
le450_15b	15	17.3	6.64	21	0.08	22	0.01	18	3.76	16	36.86	17	0.01
le450_15c	15	23.2	8.18	28	0.05	30	0.01	25	2.08	22	38.88	25	0.02
le450_15d	15	23.3	8.26	28	0.06	31	0.01	25	3.68	19	38.58	25	0.01
musol.i.1	49	49.0	1.88	49	0.02	49	0.01	49	1.18	49	4.23	49	0.01
school1	14	14.0	1.58	21	0.02	42	0.01	14	1.84	14	17.99	21	0.01
school1_nsh	14	14.2	1.60	34	0.04	39	0.01	14	1.85	14	11.74	14	0.01
Average of the above		k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]
		1.178	6.648	1.395	0.067	1.612	0.012	1.172	3.392	1.149	26.343	1.320	0.017

Table 2: Solution quality and run time of different graph coloring algorithms all compiled and run under the same conditions. Like in Tab. 1, average values are arithmetic mean of solution quality and geometric mean of run times.

“RCC” is our Range Compaction Coloring algorithm as in Tab. 2. Our machine ran `dfmax r500.5` in 29.0s.

“SWO” is the general approach to optimization applied to the graph coloring problem by Joslin and Clements [JC99]. Times for SWO are originally on a 333 MHz Pentium Pro, on which `dfmax r500.5` ran in 86.0s.

“TABU-G” is the tabu branch-and-bound algorithm by Glover et al. [GPR96]. Times were obtained on a DEC Alpha 3000, on which `dfmax r500.5` ran in 86.9s.

“IG” is the Iterated Greedy algorithm by Culberson and Luo [CL96]. On the SUN

Graph		RCC		SWO		TABU-G		IG		IMPASSE		MC	
name	χ	k	t [s]	k	t [s]	k	t [s]	k	t [s]	k	t [s]	k	t [s]
DSJC125.5	17	19.3	0.23	18.3	0.54	20.0	53.54	18.9	0.85	17.0	2.14	21.3	0.41
DSJC250.5	28	32.4	1.82	31.9	2.80	35.0	1202.40	32.8	2.31	28.0	90.42	36.5	1.97
DSJC500.5	48	55.6	12.08	56.3	13.78	65.0	1202.40	58.6	6.18	49.0	2731.36	63.6	12.64
DSJC1000.5	84	97.2	185.73	101.5	70.30	117.0	1202.40	104.2	22.87	89.0	13974.41	111.5	95.00
DSJR500.1	12	12.0	7.60	12.0	0.67	12.0	0.16	12.0	7.15	12.0	0.15	13.0	0.62
DSJR500.5	123	130.9	36.30	124.1	23.15	126.0	138.01	129.6	8.84	123.0	59.06	133.5	18.23
R125.1	5	5.0	0.22	5.0	0.07	5.0	0.13	5.0	0.69	5.0	0.15	5.0	0.10
R125.1c	46	46.0	0.47	46.0	1.72	46.0	0.32	46.0	0.37	46.0	0.15	46.2	1.04
R125.5	36	37.2	0.32	36.0	0.94	36.0	0.23	36.9	0.64	36.0	0.15	38.8	0.52
R250.1	8	8.0	1.47	8.0	0.17	8.0	0.08	8.0	2.37	8.0	0.15	8.0	0.21
R250.1c	64	64.0	5.02	64.0	10.31	65.0	16.21	64.0	1.55	64.0	0.15	65.8	5.39
R250.5	65	69.6	3.75	65.0	4.95	66.0	20.60	68.4	2.81	65.0	27.69	70.4	2.59
R1000.1	20	21.0	65.51	20.0	2.70	20.0	0.57	20.6	29.49	20.0	2.75	22.3	2.69
R1000.5	238	255.5	585.79	238.9	193.61	248.0	1202.40	253.2	34.79	241.0	317.93	260.4	162.03
flat300_20_0	20	20.3	1.81	25.3	5.53	39.0	1202.40	20.2	1.28	20.0	0.15	38.7	3.00
flat300_26_0	26	36.1	3.06	35.8	4.04	41.0	1202.40	37.1	2.59	26.0	3.37	40.4	3.00
flat300_28_0	28	36.2	3.12	35.7	4.01	41.0	1202.40	37.0	3.24	31.0	644.74	40.3	2.80
flat1000_50_0	50	95.1	191.63	100.0	68.71	-	-	65.6	49.50	50.0	0.15	108.4	88.27
flat1000_60_0	60	95.4	180.93	100.7	66.73	-	-	102.5	29.54	60.0	0.15	109.2	97.28
flat1000_76_0	76	96.5	195.76	100.6	70.23	-	-	103.6	26.92	89.0	3716.52	109.7	104.53
le450_15a	15	17.6	7.17	15.0	1.85	16.0	6.23	17.9	5.74	15.0	0.15	18.0	1.14
le450_15b	15	17.3	6.64	15.0	2.06	15.0	9.93	17.9	5.49	15.0	0.15	18.0	1.14
le450_15c	15	23.2	8.18	21.1	2.70	23.0	1202.40	25.6	4.90	15.0	19.28	25.9	2.18
le450_15d	15	23.3	8.26	21.2	2.63	23.0	1202.40	25.8	4.55	15.0	12.24	25.9	2.18
musol.i.1	49	49.0	1.88	49.0	1.99	49.0	0.11	49.0	1.41	49.0	0.15	49.0	0.52
school1	14	14.0	1.58	14.0	2.83	29.0	31.67	14.0	3.55	14.0	0.15	28.3	2.69
school1_nsh	14	14.2	1.60	14.0	2.43	26.0	10.90	14.1	3.02	14.0	0.15	23.5	2.07
Average of the above		k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]	k/χ	t[s]
Without flat1000_X		1.127	4.374	1.098	3.040	1.264	24.471	1.148	3.330	1.008	3.116	1.303	2.268

Table 3: Solution quality and run time of RCC and other graph coloring algorithms as reported in the literature. To allow a meaningful — though approximate — comparison, timings for these were taken from the respective papers and scaled based on the performance in the `dfmax r500.5` DIMACS benchmark program.

Sparc 10/40 on which it was run, the `dfmax r500.5` benchmark took 192.6s.

“IMPASSE” is the Distributed Coloration Neighborhood Search by Morgenstern [Mor96]. Even though it runs on five identically configured 80 MHz SUN SPARCstation 2s, the times are the sum of all computers’ times. This gives an acceptable qualitative indication of the time that would be required by a sequential simulation of the distributed algorithm. The reported execution time for `dfmax r500.5` is 189.26s.

“MC” is the *MinColoring* approximate algorithm recently proposed by us, based on neural networks [DJH02]. The original timings for MC are on a 466 MHz DEC Alpha 21164, on which `dfmax r500.5` runs in 28.0s.

Keeping in mind that the run time comparisons are only approximate, from Tab. 3 and 4 we see the following. IMPASSE stands out for the best solution quality and the lowest run time. However, as Morgenstern himself says: “The hidden cost not reflected by these times is the effort required to find the values for k , I , and T [the parameters]. This hidden cost doubles or even triples the actual coloring time. On the other hand, this cost

Graph	RCC	SWO	TABU-G	IG	IMPASSE	MC
DSJC125.5	4	2	5	3	1	6
DSJC250.5	3	2	5	4	1	6
DSJC500.5	2	3	5	4	1	6
DSJC1000.5	2	3	6	4	1	5
DSJR500.1	1	1	1	1	1	6
DSJR500.5	5	2	3	4	1	6
R125.1	1	1	1	1	1	1
R125.1c	1	1	1	1	1	6
R125.5	5	1	1	4	1	6
R250.1	1	1	1	1	1	1
R250.1c	1	1	5	1	1	6
R250.5	5	1	3	4	1	6
R1000.1	5	1	1	4	1	6
R1000.5	5	1	3	4	2	6
flat300_20_0	3	4	5	2	1	6
flat300_26_0	3	2	6	4	1	5
flat300_28_0	3	2	6	4	1	5
flat1000_50_0	3	4	—	2	1	5
flat1000_60_0	2	3	—	4	1	5
flat1000_76_0	2	3	—	4	1	5
le450_15a	4	1	3	5	1	6
le450_15b	4	1	1	5	1	6
le450_15c	4	2	3	5	1	6
le450_15d	4	2	3	5	1	6
musol.i.1	1	1	1	1	1	1
school1	1	1	6	1	1	5
school1_nsh	4	1	6	3	1	5

Table 4: Ranks by solution quality of the results in Tab. 3 (1 is best).

is removed for a class of graphs once the parameters have been determined.” [Mor96] It is worth noting that k is the target number of colors. This parameter alone is obviously difficult to find (the best value is the chromatic number of the graph). Also, all these parameter settings are computed on each individual graph.

The results in Tab. 4 suggest that RCC, SWO, TABU-G, and IG all seem competitive with each other in solution quality. On average, though, SWO yields a little better solution quality than RCC for a shorter run time (Tab. 3). About TABU and IG, it is interesting to notice that average solution quality as well as the average run time seem to be better in the experiments we run ourselves (Tab. 2), even though we simply followed Culberson’s suggestion for the parameter settings. The solution quality of the neural approach, MC, is clearly not competitive, even though its run time is comparatively short.

4 Conclusion

This paper has presented a novel heuristic for graph coloring. The heuristic works by making the range of colors used more compact over “up-” and “down-sweeps”. The heuristic also has a pressure component. An annealing schedule is used in which pressure is periodically applied and relaxed. The value of this component was quantified empirically. The algorithm with annealed pressure was found to perform better than a version without

pressure. We evaluated our heuristic on a broad range of DIMACS benchmark graphs and compared results with the following methods: a simple greedy algorithm, “Iterated Greedy” by Culberson and Luo [CL96], “MAXIS” originally by Bollobás and Thomason [BT85], in the improved version by Culberson [Cul], “TABU-C” by Hertz and De Werra [HdW87] as modified and improved by Culberson and Luo [CL96], “TABU-G” by Glover et al. [GPR96], “DSATUR” by Bréaz [Bre79], “IMPASSE” by Morgenstern [Mor96], “MC” by Di Blas et al. [DJH02], “SWO” by Joslin and Clements [JC99].

Our results indicate that IMPASSE is a clear winner in solution quality. However, IMPASSE is a complex algorithm with multiple parameters which need to be set on each input graph. These settings are very costly to compute. RCC is competitive with the remaining algorithms and in fact among the best of them. Finally, whereas all the other algorithms use multiple parameters, many of which are difficult to set, RCC uses just one simple and easy-to-set parameter.

Acknowledgments

This work was supported in part by the NSF grant EIA-9722730.

References

- [BAd] Valmir C. Barbosa, Carlos A. G. Assis, and Josina O. do Nascimento. Two novel evolutionary formulations of the graph coloring problem. *J. of Combinatorial Optimization (to appear)*. available at citeseer.nj.nec.com/479842.html.
- [Ber94] Matthias Oliver Berger. k-coloring vertices using a neural network with convergence to valid solutions. In *IEEE Int. Conf. on Neural Networks*, volume 7, pages 4514–4517. IEEE Press, 1994.
- [Bre79] Daniel Bréaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, April 1979.
- [BT85] Béla Bollobás and Andrew Thomason. Random graphs of small order. In Michał Karoński and Andrzej Ruciński, editors, *Random Graphs '83*, volume 28 of *Annals of Discrete Mathematics*, pages 47–97. North-Holland Publishing Co., Amsterdam, 1985.
- [CH97] D. Costa and A. Hertz. Ants can color graphs. *J. of the Operational Research Society*, 48:295–305, 1997.
- [CHD95] D. Costa, A. Hertz, and O. Dubuis. Embedding of a sequential algorithm within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1:105–128, 1995.
- [CHdW87] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32:260–266, 1987.
- [CL96] Joseph C. Culberson and Feng Luo. Exploring the k-colorable landscape with iterated greedy. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 245–284. American Mathematical Society, 1996.
- [Cou97] O. Coudert. Exact coloring of real-life graphs is easy. In *Proc. of the 34th Design Automation Conf.*, pages 121–126. ACM Press, 1997.
- [Cul] Joseph Culberson. *Joseph Culberson’s Coloring Page* . <http://web.cs.ualberta.ca/~joe/Coloring>.
- [Dah87] E. Danning Dahl. Neural network algorithm for an NP-complete problem: Map and graph coloring. In *IEEE Int. Conf. on Neural Networks*, volume 3, pages 113–120. IEEE Press, 1987.

- [de 85] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19:151–162, 1985.
- [De 94] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.
- [DH98] R. Dorne and J. Hao. Tabu search for graph coloring, t-coloring and set t-colorings, 1998.
- [DIM] DIMACS benchmark program, `dfmax.c`, and documentation, `README`, can be found in the `ftp` site. <ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/>.
- [DJH02] Andrea Di Blas, Arun Jagota, and Richard Hughey. Energy function-based approaches to graph coloring. *IEEE Trans. on Neural Networks*, 13(1):81–91, January 2002.
- [DW84] P. J. Donnelly and D. J. A. Welsh. The antivoter problem: Random 2-colourings of graphs. In B. Bollobas, editor, *Graph Theory and Combinatorics*, pages 133–144. Academic Press, 1984.
- [FF96] Charles Fleurent and Jaques A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique and satisfiability. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 619–652. American Mathematical Society, 1996.
- [GC93] David W. Gassen and Jo Dale Carothers. Graph color minimization using neural networks. In *IEEE Int. Joint Conf. on Neural Networks*, pages 1541–1544. IEEE Press, 1993.
- [GDWL92] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-level Synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
- [GPR96] Fred Glover, Mark Parker, and Jennifer Ryan. Coloring by tabu branch and bound. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 285–308. American Mathematical Society, 1996.
- [HdW87] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
- [Hom] Home page of Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). <http://dimacs.rutgers.edu>.
- [Jag96] A. Jagota. An adaptive, multiple restarts neural network algorithm for graph coloring. *European J. of Operational Research*, 93:257–270, 1996.
- [JAMS91] David S. Johnson, Cecilia R. Aragon, Lyle A Mcgeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May-June 1991.
- [JC99] David E. Joslin and David P. Clements. ‘Squeaky Wheel’ optimization. *J. of Artificial Intelligence Research*, 10:353–373, 1999.
- [JT96] David S. Johnson and Michael A. Trick, editors. *Clique, Coloring, and Satisfiability — Second DIMACS Implementation Challenge*. American Mathematical Society, 1996.
- [KMS94] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. In *Proc. of 35th IEEE Symp. on the Foundations of Computer Science*, pages 2–43. IEEE Comp. Soc. Press, 1994.
- [Knu94] Donald E. Knuth. The sandwich theorem. *Electronic J. of Combinatorics*, 1, 1994. Article 1, approx. 48 pp. (electronic).
- [KP98] Darko Kirovski and Miodrag Potkonjak. Efficient coloring of a large spectrum of graphs. In *Proc. of the 35th Design Automation Conf.*, pages 427–432. ACM Press, 1998.
- [LC96] Gary Lewandowski and Anne Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 309–334. American Mathematical Society, 1996.

- [Lei79] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *J. of Research of the National Bureau of Standards*, 84(6):489–503, 1979.
- [Mor96] Craig Morgenstern. Distributed coloration neighborhood search. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 335–358. American Mathematical Society, 1996.
- [MT96] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.
- [PS89] C. Peterson and B. Soderberg. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(1):3–22, 1989.
- [PS91] W. J. M. Philipsen and L. Stok. Graph coloring using neural networks. In *IEEE Int. Symp. on Circuits Systems*, volume 3, pages 1597–1600, 1991.
- [PW89] A. D. Petford and D. J. A. Welsh. A randomised 3-colouring algorithm. *Discrete Mathematics*, 74:253–261, 1989.
- [RL93] S. Ramanathan and E. L. Lloyd. Scheduling broadcasts in multi-hop radio networks. *IEEE/ACM Transactions on Networking*, 1(2):166–172, 1993.
- [SP97] Kate Smith and Marimuthu Palaniswami. Static and dynamic channel assignment using neural networks. *IEEE Journal on Selected Areas in Communications*, 15(2):238–249, February 1997.
- [TL91] Yoshiyatsu Takefuji and Kuo Chun Lee. Artificial neural networks for four-coloring map problems and k-colorability problems. *IEEE Trans. on Circuits and Systems — I*, 38(3):326–333, March 1991.
- [Vv00] Aleksander Vesel and Janez Žerovnik. How good can ants color graphs? *J. of Computing and Information Technology - CIT*, 8:131–136, 2000.
- [WP67] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10:85–86, 1967.
- [Wu98] Chau Wah Wu. Graph coloring via synchronization of coupled oscillators. *IEEE Trans. on Circuits and Systems — I*, 45(9):974–978, September 1998.

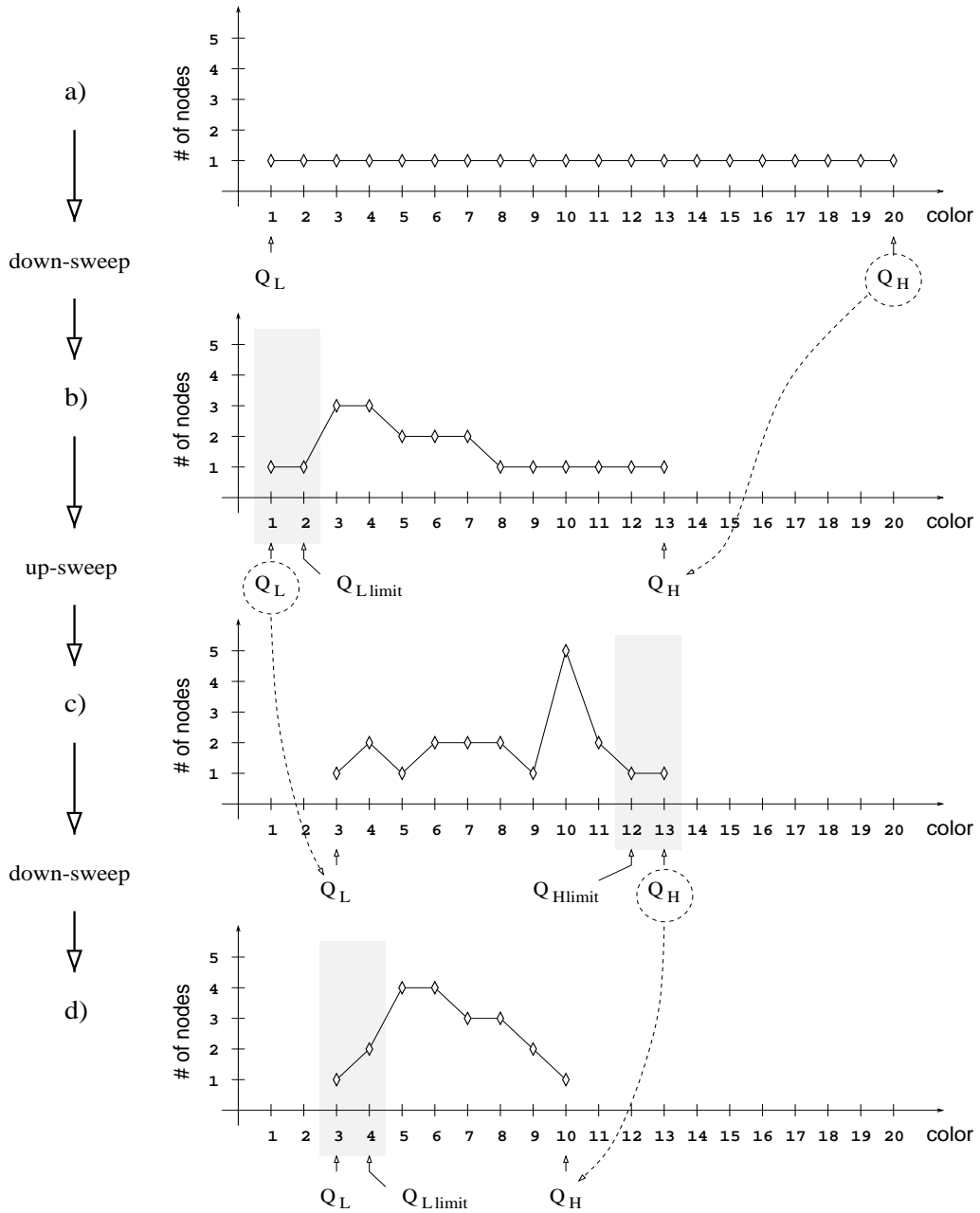


Figure 1: A sample progress diagram: in a), a graph with 20 nodes is initially colored with $c(v) = v$, so $Q_L = 1$ and $Q_H = 20$. In b), the recolored graph at the end of the first down-sweep is shown. The highest color used, Q_H , is now 13; pressure has been applied with $\alpha = 2$ during the down-sweep, therefore nodes in the shadowed areas have not been recolored. In c) and d) we see how repetitively performing up- and down-sweeps in this way tends to reduce the color range.

```

RangeCompactionColoring(G)
{
  for all nodes  $v$ , set  $c(v) = v$ 
  set initial  $\alpha_t$ , set  $Q_L = 1$  and  $Q_H = N$ 

  repeat alternating up-sweeps and down-sweeps
  {
    if (down-sweep)
    {
      set  $Q_{Llimit} = Q_L + \alpha_t$ 
      while(  $\exists v$  such that  $c(v) > Q_{Llimit}$  AND
             recoloring  $v$  with the lowest color  $c$  such that  $Q_{Llimit} \leq c < c(v)$ 
             would keep the coloring proper )
        recolor  $v$  with  $c$ ;
      set  $Q_H$  to highest color used}

    if (up-sweep)
    {
      set  $Q_{Hlimit} = Q_H - \alpha_t$ 
      while(  $\exists v$  such that  $c(v) < Q_{Hlimit}$  AND
             recoloring  $v$  with the highest color  $c$  such that  $Q_{Hlimit} \geq c > c(v)$ 
             would keep the coloring proper )
        recolor  $v$  with  $c$ ;
      set  $Q_L$  to lowest color used}

    adjust  $\alpha_t$  according to its schedule

  } until (stop)
}

```

Figure 2: The Range-Compaction Coloring algorithm. The *stop* condition can be a limit on the number of iterations, a desired coloring size, or both.

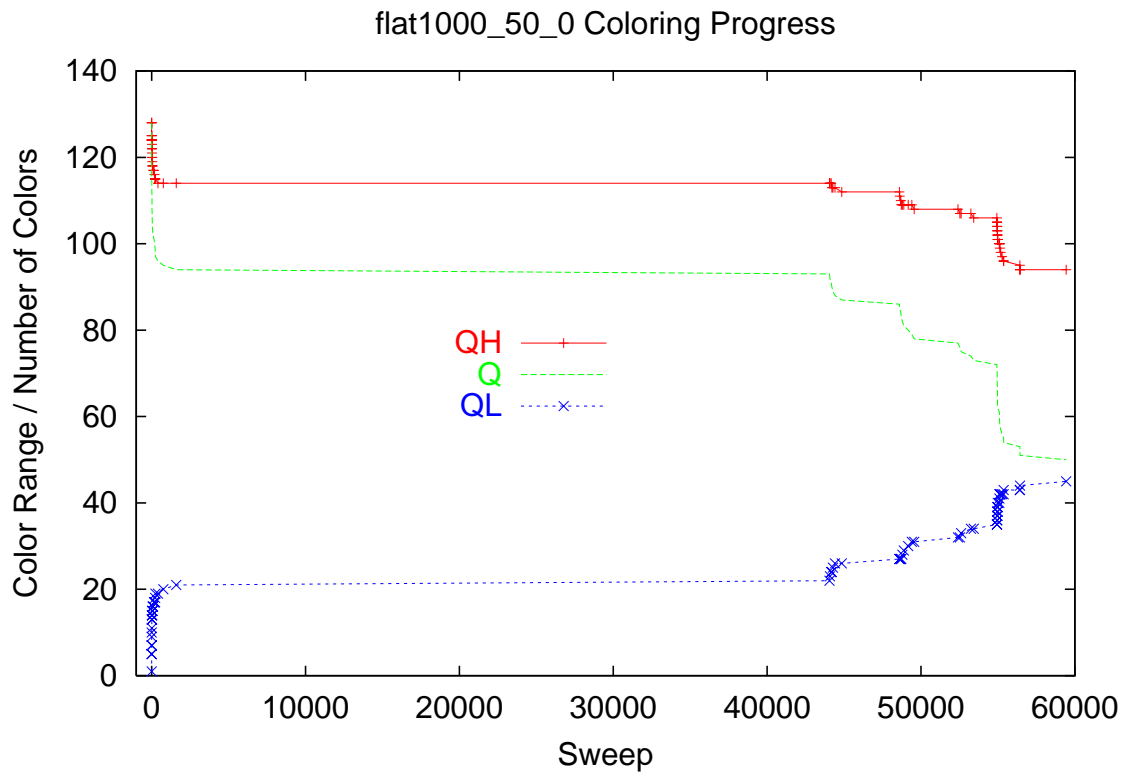


Figure 3: Coloring evolution on a sample graph (flat1000_50_0) showing how the range of colors progressively narrows (Q_H to Q_L) reducing the total number of colors Q . Up- and down-sweeps are counted separately. The initial coloring that uses $N = 1000$ colors is not shown.

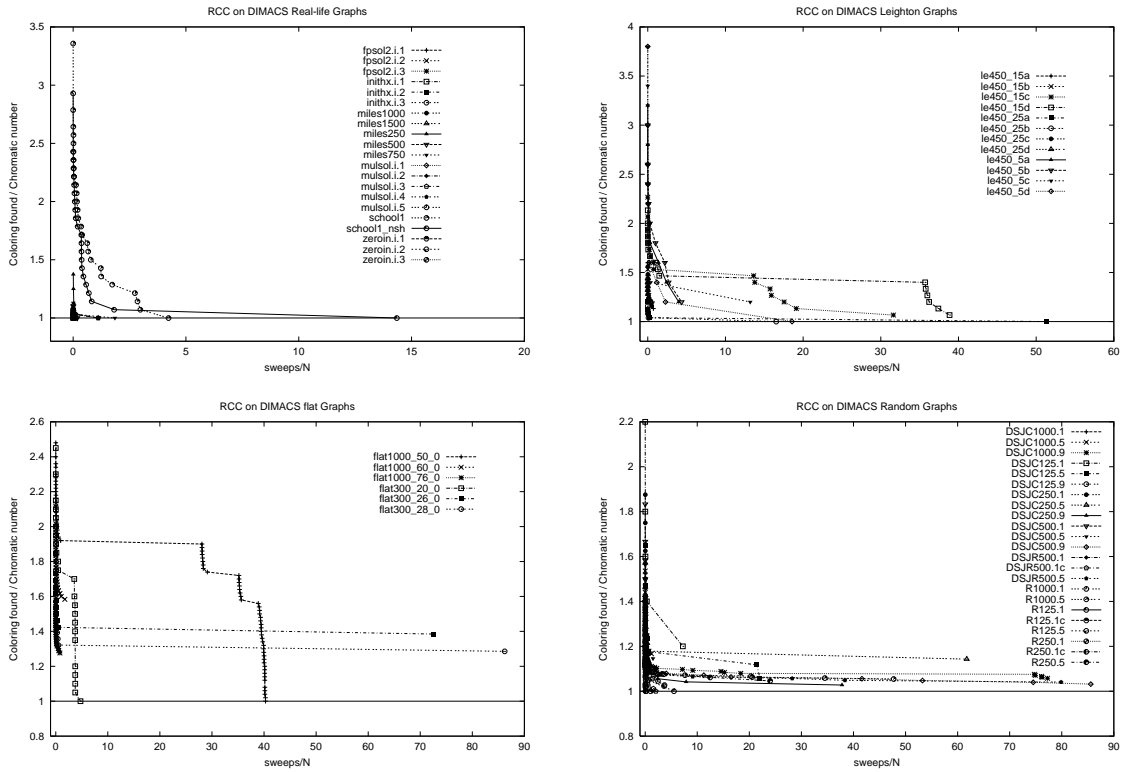


Figure 4: Solution quality as a function of run time for different classes of DIMACS graphs. The run time here is expressed as number of sweeps (up and down counted separately) normalized with respect to the graph size (number of nodes).