Proceedings of GECCO 2001 San Francisco, CA, Morgan Kaufmann, pp. 313-319. July 2001

"Forging" Optimal Solutions to the Edge-Coloring Problem

Brandon P. Enochs
Math and Computer Science Dept.
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
brandon-enochs@utulsa.edu

Roger L. Wainwright
Math and Computer Science Dept.
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
rogerw@utulsa.edu

Abstract

The applications of a modified version of the simulated annealing algorithm for solving the edge-coloring problem, which consists of partitioning the edges of a graph into the minimum number of disjoint subsets such that no two edges in a given subset are adjacent, was investigated. With the exception of coloring bipartite graphs, finding a minimum edge coloring of a graph is NP-Complete. The traditional simulated annealing algorithm was modified to incorporate an additional perturbation (disruptive) operator. The details of the modifications to the simulated annealing algorithm, and the technique we chose to represent solutions for the edge-coloring problem are presented in the paper. We also describe two perturbation operators used by our modified technique, along with the heuristic function used to evaluate solutions. Our algorithm for solving the edge-coloring problem was tested on over sixty test graphs from the literature. In every case, our algorithm found the optimal edge coloring. We compared our algorithm to a grouping genetic algorithm technique for solving this problem published recently in the literature. Our results over a wide variety of test graphs were vastly superior to the grouping genetic algorithm technique.

1 INTRODUCTION

Generally, for NP-Hard partitioning problems like the vertex-coloring problem, heuristic optimization techniques, such as simulated annealing, are well suited for approximating solutions. The edge-coloring problem, which is another NP-Hard partitioning problem, is no exception. Solutions to the edge-coloring problem have numerous practical applications including timetable problems (Bondy & Murty, 1976), partitioning problems (Fiorini & Wilson, 1977), and certain types of scheduling problems. The edge-coloring problem consists of partitioning the edges of a graph G into the minimum number of disjoint subsets such that all edges in a given subset are not adjacent. The smallest number of subsets into which the edges of a graph can be partitioned is called the chromatic index of the graph G, denoted as $\chi'(G)$ (Fiorini & Wilson, 1977). The chromatic index of any graph is bounded by $\Delta_{max}(G) \leq \chi'(G) \leq \Delta_{max}(G) + \mu$, where $\Delta_{max}(G)$ is the maximum degree of the graph and μ is the maximum edge multiplicity of the graph (Vizing, 1964).

Khuri *et al.* (2000) provides the most recent and comprehensive work on various techniques for solving the edge-coloring problem. Khuri developed and tested three approaches for approximating solutions to instances of the edge-coloring problem. By far, Khuri's most successful approach to approximating solutions to the edge-coloring problem came from his use of a grouping genetic algorithm. Grouping genetic algorithms, which are a subclass of genetic algorithms, focus on approximating solutions to problems with grouping properties, such as the edge-coloring problem. Using a grouping genetic algorithm, Khuri was able to find the optimal solution for nearly all of the graphs he tested. However, as the density, (which is the ratio of the number of edges to the number of vertices in a graph), increased, the performance of his approach faltered, producing approximations using as many as three additional colors beyond the optimal number.

In order to overcome the deficiencies experienced by Khuri's algorithm, we developed a different approach to the edge-coloring problem. In our approach, solutions to the edge-coloring problem were "forged" using simulated annealing rather than "evolved" using genetic algorithms. We applied a slight modification to the traditional simulated annealing technique. This modification consisted of applying a secondary perturbation operator at key times during the execution of the algorithm. This modified technique was then applied to 63 simple graphs and 15 multigraphs from various sources in the literature such as Knuth's Stanford GraphBase (1993) and the DIMACS ftp site (2000), and compared to the results obtained by Khuri (2000).

The rest of this paper is organized as follows: Section 2 presents the modifications we made to the traditional simulated annealing algorithm. Section 3 depicts the representation we used to encode a solution to the edge-coloring problem, and the definition of the heuristic function used to evaluate candidate solutions. The algorithms used by the perturbation, and the additional perturbation operators to improve solutions, as well as the details of how solutions are initially generated are also presented in Section 3. Section 4 presents the results of several runs of our modified simulated annealing algorithm on various the test graphs from the literature. Section 5 describes the conclusions of our research, and comparisons to results obtained Khuri's grouping genetic algorithm. Acknowledgements and references are given at the end of the paper.

2 SIMULATED ANNEALING AND MODIFICATIONS

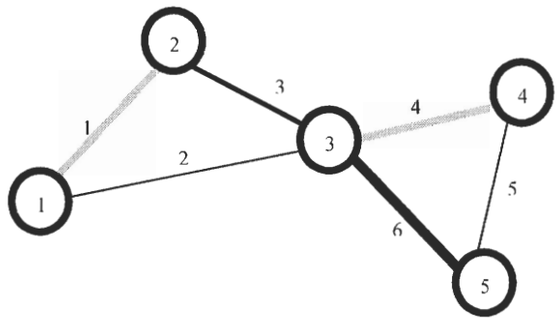
Traditionally, simulated annealing uses only one operation—perturbation. Perturbation, which is similar in functionality to mutation in genetic algorithms, attempts to improve a solution by making small modifications. The resultant solution created by perturbation is accepted or rejected based on several parameters given at runtime. For the edge-coloring problem, perturbation alone was not able to solve certain graphs in a reasonable amount of time; therefore, an additional operator was implemented to overcome this weakness. The new operator, which we call the "kick" operator, is similar in functionality to perturbation. The kick operator functions as follows: if a specified number of iterations has passed during which there has been no heuristic improvement, then some perturbation on the current solution is performed. This perturbation is intended to disrupt the current solution such that further iterations of the modified simulated annealing algorithm will lead to heuristic values exceeding those found before the kick was executed. Unlike the perturbation operation, the modifications made by the kick operator are accepted regardless of any benefits or losses that result. The modified simulated annealing algorithm is described as follows (it is assumed that the reader is familiar with simulated annealing):

1. Generate an initial solution and set the change counter to zero.
2. Given the parameters T, the initial temperature, α, the rate at which the temperature decreases, N, the initial number of iterations to perform before decreasing the temperature and increasing the number of iterations to be performed, β, the rate at which N increases, and C, the number of iterations allowed to occur in which the current solution does not improve before the kick operator is executed. Execute the following three steps N times.
   a. Execute the perturbation operator on the current solution. This will produce a new solution.
   b. Choose a new current solution by weighing the new solution against the current solution with the acceptance function. The acceptance function will choose one of the two solutions based on the current temperature, T, the heuristic value of the current solution, and the heuristic value of new solution. The acceptance function behaves as follows: If the new solution is better than the current solution, then the new solution is chosen and the change counter is reset to zero; however, if the new solution is worse than the current solution, a random real number is generated and compared against a value based on the current temperature and the heuristic difference between the current and new solutions. If the random real number is greater than or equal to the calculated value, then the new solution is chosen and the change counter is reset; otherwise, the current solution is chosen and the change counter is incremented by one (except for the change counter logic, this a traditional simulated annealing algorithm).
   c. After the acceptance function has chosen a new current solution, test the following condition: if the change counter equals C, then perform the kick operation and reset the change counter to zero.
3. Increase N to $N*\beta$, and decrease T to $T*\alpha$..
4. Perform the following test: if the total number of iterations performed equals the maximum number of iterations allowed, the maximum value of the heuristic is found, or a specified time limit is exceeded, then return the current solution, otherwise, repeat step 2a.

3 SOLUTION REPRESENTATION, HEURISTIC EVALUATION, AND OPERATOR ALGORITHMS

3.1 SOLUTION REPRESENTATION, HEURISTIC EVALUATION, AND OPERATOR ALGORITHMS

The solution representation is extremely important to the success of any algorithm. Our solution representation of the edge coloring of a graph is simply an array of integers representing the color assigned to each edge with the edges of a graph numbered *a priori*. This proved to be extremely successful. The first number represents the color of edge 1; the second number represents the color of edge 2; and so on. This representation lends itself to efficient color lookup, efficient heuristic evaluation, and efficient memory usage. Figure 1 shows an example graph and the solution representation for that graph.



SOLUTION

EDGE	1	2	3	4	5	6
COLOR	3	1	2	3	1	4

Figure 1: Solution Representation

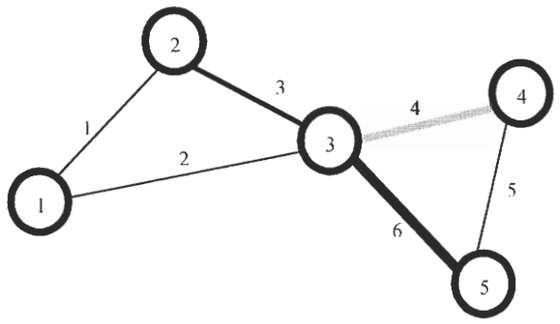
3.2 HEURISTIC FUNCTION

The heuristic function used for this problem is defined as $\Sigma\ [d(i)-invalid(i)]$ for $i = 1..n$, where $d(i)$ is the degree of the ith node, $invalid(i)$ is the number of edges with the same color that are connected to the ith node, and n is the number of nodes in the graph. It is important to note that $\Sigma\ d(i)$ for $i=1..n$ is a fixed constant for a given graph. Thus, when the value of the heuristic function is equal to $\Sigma\ d(i)$ for $i=1..n$, a solution is feasible. Therefore, the heuristic function measures a solution's distance to feasibility; consequently, higher heuristic values indicate solutions that are closer to feasibility. In addition, the number of usable colors is a fixed constant specified at run-time. Thus, annealing can stop when the heuristic value of the current solution is equal to $\Sigma\ d(i)$ for $i=1..n$, indicating that no two adjacent edges share the same color.

3.3 PERTURBATION FUNCTION

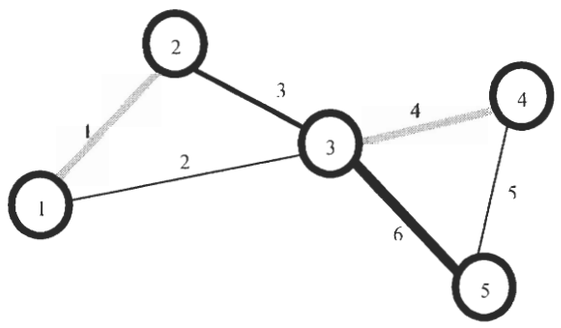
The pertubation function is a greedy operation. The perturbation operator randomly selects one node in which infeasible edges are incident. The operator then randomly selects one of those incident infeasible edges. Next, the operator attempts to assign a non-conflicting color to the selected edge. If a non-conflicting color cannot be found, then the selected edge is assigned a random color.

For example, consider Figure 2a. Assume that node 1 is randomly selected. Observe that edges 1 and 2, which are incident to node 1, have the same color, (color 1). Assume that edge 1 has been randomly selected. Then a free color (a color other than color 1 in this case) is randomly selected. Assume the free color selected was color 3, then color 3 is assigned to edge 1 (see figure 2b). If no free color exists, then edge 1 receives a random color that was not the color of any edge connected to node 3 to which edge 1 is adjacent (node 3 has the highest degree).



(a) BEFORE

SOLUTION						
EDGE	1	2	3	4	5	6
COLOR	1 (invalid)	1	2	3	1	4



(b) AFTER

SOLUTION						
EDGE	1	2	3	4	5	6
COLOR	3 (valid)	1	2	3	1	4

Figure 2: Perturbation Example with Node 1 Selected.

3.4 KICK FUNCTION

The kick function is a disruptive operation that reassigns a new color to a randomly selected edge with an invalid color. Empirical results support the claim that this operator is able to overcome many of the local hurdles that the heuristic function presents, producing the optimal solution on certain graphs in ¼ of the time required for traditional simulated annealing to produce the same results. On every graph we tested, traditional simulated annealing was able to produce an optimal solution. However, on many graphs, depending upon the seed used, traditional simulated annealing required as many as four times the number of iterations compared to our modified simulated annealing algorithm. As a kick operator example, consider again Figure 2a. In this case, edge 1 has been assigned an invalid color, and is, therefore, a candidate for the kick operation. Assume that the kick operator randomly chose edge 1. Edge 1 would be reassigned a random valid color, in this case, 3. If no color for edge 1 had existed, which was not in conflict with other colors assigned to adjacent edges, then edge 1 would have been assigned a random color.

3.5 INITIAL SOLUTION GENERATION

The initial solution is constructed using the following algorithm: first, find the node with the highest degree. Second, assign a different color to each edge connected to the node with the highest degree. Finally, the remaining edges are then assigned random colors. Note the range of valid colors for the remaining edges is reduced by the colors of edges connected to the node with the highest degree. For example, in Figure 1, node 3 has the highest degree; thus, edges 2, 3, 4, and 6 must always have different colors. As a result, this significantly reduces the search space for any given graph. Furthermore, the colors of the edges connected to the vertex with the highest degree do not change! This is an application of the reduction principle presented in (Corcoran & Wainwright, 1996), and proved to be significant to the success of our algorithm. Also the number of usable colors is a fixed constant specified by the user at runtime. Fixing the number of usable colors in this manner turned out to be a critical aspect of the success of our algorithm. Fixing the usable colors allows the user to specify a minimum quality for solutions. Thus, all the user has to do is specify the optimal number of colors to obtain an optimal solution

4 RESULTS

The performance of our modified simulated annealing algorithm and the grouping genetic algorithm were compared on 63 problem instances taken from various sources in the literature such as the DIMACS challenge ftp site (2000) and the Stanford GraphBase (1993). Graph density values ranged from 2 edges per node all the way up to 99.5 edges per node. In addition, graphs with a wide range of other properties, such as completeness, were tested. Table 1 depicts the results obtained for the 63 problem instances from our modified simulated annealing algorithm and the grouping genetic algorithm described by Khuri (2000). For the modified simulated annealing algorithm, the following parameters were used: $T = 0.0, \alpha = 0.0, \beta = 1.0, N = 1000$, and C=500...1250. For Khuri's grouping genetic algorithm, the following parameters were used: mutation rate = 0.2, crossover rate = 0.6, and population size = 20. The Problem Instance (left half) portion of Table 1 lists the datasets used and various properties of those datasets. The first 21 datasets are complete graphs named complete*x*.col, where *x* is the number of nodes in the graph. The remaining test graphs were taken from Donald Knuth's Stanford GraphBase (1993) and the DIMACS ftp site (2000). The column headed by Δ indicates the maximum degree for each graph, and, for many graphs, the minimum number of colors that can be used to color that graph.

The results shown in the right portion of Table 1 record the *# of iterations*, the *# of colors used,* and the *time required* (in seconds) to reach the optimal solution for our modified simulated annealing algorithm. We ran our modified simulated annealing algorithm on the test datasets using an Intel Pentium III 700Mhz processor running Microsoft Windows 2000 Professional and Sun's JDK version 1.3. Optimal solutions, which are indicated as a **bold** entry in the column (# of colors used), are solutions where the number of colors used were in the range of [Δ...Δ+μ], where μ is the maximum edge multiplicty of the graph.

Khuri *et al.* (2000) research represents the most recent work on this topic. In his paper he developed three techniques for solving the edge-coloring problem, the best of which was his GGA algorithm. The results from Khuri's GGA algorithm for the 63 test graphs are given in the last column under the heading *GGA results*. Similarly, entries in the column (GGA Results) that are depicted in **bold** represent optimal solutions. Our algorithm found the optimal solution in all 63 cases. Khuri's grouping genetic algorithm found the optimal solution in 21 of the 63 cases.

5 CONCLUSIONS

The results indicate that the edge-coloring problem lends itself very nicely to our modified simulated annealing algorithm. Impressively, our modified simulated annealing algorithm generated optimal solutions for every test data set. This includes all of the multigraphs from the

Stanford GraphBase, the simple graphs from the DIMACS ftp site, and the generated complete graphs. The optimal coloring was even obtained for a complete graph with 200 nodes. For every dataset except those that formed a complete or regular graph with an odd number of nodes, the number of colors used in the final solution was equal to Δ, the optimal number. In the cases in which datasets formed a complete or regular graph with an odd number of nodes, the number of colors used increased to $\Delta+1$; however, the chromatic index of a complete or regular graph, K_n, for which n is odd, is $\Delta+1$, thus, all results in which $\Delta+1$ colors were used were optimal as well (Rosen, 2000). The empirical results obtained using the modified simulated annealing algorithm support the claim that it is superior in performance when compared to recently developed algorithms for the edge-coloring problem. The use of only a fixed number of colors was of critical importance to the success of the modified simulated annealing technique in finding optimal solutions to the graphs tested. Limiting the number of usable colors allowed the modified simulated annealing technique to determine with absolute certainty whether or not the best solution, given the initial parameters, had been found. Also, by fixing the number of usable colors to the optimal number, the modified simulated annealing algorithm worked only towards finding the optimal solution while retaining the ability to produce feasible solutions that are not optimal. Also, the masking performed in assigning colors of edges reduced search space and lead to faster discoveries of optimal solutions. The incorporation of the kick operator, while unnecessary in producing optimal solutions, greatly reduced the average number of iterations required to find solutions to instances of the edge-coloring problem. Finally, the modified simulated annealing algorithm proved to be an excellent algorithm for solving the edge-coloring problem over a wide variety of graphs.

Acknowledgments

The authors wish to thank Sami Khuri and Tim Walters for providing us with additional results of their algorithm not published in their original paper, and for providing us with their software to run their GGA algorithm for solving the edge-coloring problem.

References

Bondy, J.A., & Murty, U.S.R. (1976). *Graph Theory with Applications*. New York: McMillan.

Corcoran, A.L. & Wainwright, R.L. (1996). Reducing Disruption of Superior Building Blocks in Genetic Algorithms. *Proceedings of the 1996 ACM/SIGAPP Symposium on Applied Computing* (pp. 269-276). Philadelphia, PA. ACM Press.

DIMACS: Center for Discrete Mathematics and Theoretical Computer Science (2000). ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color.

Fiorini, S. & Wilson, R.J. (1977). *Edge-Colourings of Graphs*. London: Pitman.

Holyer, I. (1980). The NP-completeness of edge coloring. *SIAM Journal of Computing*, 10, 718-720.

Khuri, S., Walters T., & Sugono, Y. (2000). A Grouping Genetic Algorithm For Coloring The Edges of Graphs. *Proceedings of the 2000 ACM Symposium on Applied Computing* (pp. 422-427). Villa Olmo, Como, Italy. ACM Press.

Knuth, D. (1993). *The Stanford GraphBase: A Platform for Combinatorial Computing*. New York: ACM Press.

Rosen, K. (2000). *Handbook of Discrete and Combinatorial Mathematics*. New York: CRC Press.

Vizing, V. (1964). On an estimate of the chromatic class of a p-graph (in Russian). *Diskret. Analiz.*, 3, 25-30.

Proceedings of GECCO 2001 San Francisco, CA, Morgan Kaufmann, pp. 313-319. July 2001

Table 1: Edge Coloring Results

PROBLEM INSTANCE					RESULTS[1]			
File Name	# of nodes	# of edges	Density[2]	Δ	Iterations	# of colors used	Time (seconds)	GGA results
complete20.col	20	190	9.5	19	795	**19**	1	20
complete25.col	25	300	12	24	1321	**25**	2	26
complete30.col	30	435	14.5	29	1966	**29**	3	31
complete35.col	35	595	17	34	2697	**35**	4	37
complete40.col	40	780	19.5	39	4116	**39**	4	41
complete45.col	45	990	22	44	6106	**45**	5	47
complete50.col	50	1225	24.5	49	6870	**49**	6	52
complete55.col	55	1485	27	54	7233	**55**	7	58
complete60.col	60	1770	29.5	59	9973	**59**	11	63
complete65.col	65	2080	32	64	12817	**65**	17	70
complete70.col	70	2415	34.5	69	16789	**69**	23	75
complete75.col	75	2775	37	74	19892	**75**	35	78
complete80.col	80	3160	39.5	79	22407	**79**	43	83
complete85.col	85	3570	42	84	25563	**85**	52	88
complete90.col	90	4005	44.5	89	26638	**89**	77	93
complete95.col	95	4465	47	94	31124	**95**	92	98
complete100.col	100	4950	49.5	99	36267	**99**	127	104
complete105.col	105	5460	52	104	40065	**105**	169	109
complete110.col	110	5995	54.5	109	37493	**109**	180	114
complete115.col	115	6555	57	114	40967	**115**	233	119
complete200.col	200	19900	99.5	199	174191	**199**	3341	203
anna.col	138	986	7.14	142	224	**142**	2	**142**
david.col	87	812	9.33	164	113	**164**	1	**164**
homer.col	561	3258	5.80	198	590	**198**	5	**198**
huck.col	74	602	8.14	106	118	**106**	1	**106**
jean.col	80	508	6.35	72	125	**72**	1	**72**
mile250.col	128	774	3.02	32	470	**32**	1	**32**
miles500.col	128	2340	9.14	76	1985	**76**	4	77
miles1000.col	128	6432	50.25	172	6326	**172**	133	173
miles1500.col	128	10396	81.22	212	16563	**212**	171	213
queen5_5.col	25	320	12.8	32	377	**32**	1	33
queen6_6.col	36	580	16.11	38	867	**38**	1	39
queen7_7.col	49	952	19.43	48	1289	**48**	2	**48**
queen8_8.col	64	1456	22.75	54	2215	**54**	3	**54**
queen9_9.col	81	2112	26.07	64	2841	**64**	5	**64**
queen8_12.col	96	2736	28.5	64	5249	**64**	9	65
queen10_10.col	100	2940	29.4	70	4643	**70**	9	**70**

1 Optimal solutions are indicated in bold.
2 Density is the ratio of the # of edges to the # of vertices in a graph

Table 1: Continued

PROBLEM INSTANCE					RESULTS[3]			
File Name	# of nodes	# of edges	Density[4]	Δ	Iterations	# of colors used	Time (seconds)	GGA results
queen11_11.col	121	3960	32.73	80	5540	**80**	15	**80**
queen12_12.col	144	2596	36.05	86	8136	**86**	33	**86**
queen13_13.col	169	6656	39.38	96	9897	**96**	56	97
queen14_14.col	196	8372	42.71	102	12869	**102**	96	**102**
queen15_15.col	225	10360	46.05	112	15047	**112**	143	113
queen16_16.col	256	12640	49.38	118	19731	**118**	243	119
myciel3.col	11	20	1.81	5	12	**5**	1	**5**
myciel4.col	23	71	3.09	11	25	**11**	1	**11**
myciel5.col	47	236	5.02	23	119	**23**	1	**23**
myciel6.col	95	755	7.95	47	326	**47**	2	**47**
myciel7.col	191	2360	12.36	95	820	**95**	3	**95**
games120.col	120	1276	10.63	26	1836	**26**	2	27
le450_15a.col	450	8168	18.15	99	4051	**99**	35	**99**
le450_15b.col	450	8169	18.15	94	4293	**94**	36	**94**
le450_15c.col	450	16680	37.06	139	12153	**139**	234	**139**
le450_15d.col	450	16750	37.22	138	12297	**138**	231	140
le450_25a.col	450	8260	18.35	128	3558	**128**	33	129
le450_25b.col	450	8263	18.36	111	4018	**111**	38	112
le450_25c.col	450	17343	38.54	179	9887	**179**	243	180
le450_25d.col	450	17425	38.72	157	12030	**157**	440	159
le450_5a.col	450	5714	12.7	42	4764	**42**	25	43
le450_5b.col	450	5734	12.74	42	4677	**42**	25	43
le450_5c.col	450	9803	21.8	66	9175	**66**	93	67
le450_5d.col	450	9757	21.68	68	8485	**68**	88	69
fpsol2.i.3.col	325	8688	26.73	346	3082	**346**	51	347

3 Optimal solutions are indicated in bold.
4 Density is the ratio of the # of edges to the # of vertices in a graph.