A Comparison of Graph Colouring Techniques

E Parkinson P R Warren
Department of Computer Science, University of Port Elizabeth, P.O. Box 1600, Port Elizabeth, 6000

Abstract

Many scheduling problems can be modelled as graph colouring problems. This paper gives a survey of heuristic algorithms used to colour graphs by describing a number of such algorithms found in the literature using an uniform notation. We then compare these algorithms in terms of the time used and the quality of the colourings produced, on the basis of empirical results.

Keywords: *Graph Colouring, Heuristic Algorithms, NP-hard, Random Graphs.*
Computing Review Categories: *A.1, G.2.2*

1 Introduction

The graph colouring problem has many practical applications such as scheduling exams, the storing of parser tables and the assignment of radio frequencies [15]. Much research has been done to develop efficient exact and heuristic algorithms to colour graphs. The graph colouring problem is NP-hard, however and finding an exact solution is not always practical. Because of the exponential time complexity of all known exact algorithms, heuristic algorithms are usually used for problems encountered in practice. It is unknown whether a polynomial time algorithm for this problem exists. Thus far, however very few experimental results have been published comparing the large number of heuristic graph colouring algorithms known. This paper makes a uniform presentation of these algorithms. Experiments were done to test their performance on generated random graphs of different sizes. We then compare the algorithms in terms of time used and quality of the colourings produced on average.

2 Notation

We define a (simple, undirected) graph $G = (V, E)$ as a set of vertices V together with a set of edges $E = \{\{v_1, v_2\} \mid v_1, v_2 \in V\}$. Two vertices are said to be adjacent, or neighbours, in G if there is an edge between those two vertices. The degree of a vertex v in G is denoted by $d(v)$ and is defined as the number of vertices in G adjacent to v. A subgraph $H = (U, F)$ of G is defined as a graph such that $U \subseteq V$ and $F \subseteq E$. The subgraph induced by vertices $v_1, v_2, ..., v_n$ of G we denote by $\langle\{v_1, v_2, ..., v_n\}\rangle$ and it is defined as the graph $G' = (V', E')$ where $V' = \{v_1, v_2, ..., v_n\}$ and $E' = \{\{w_1, w_2\} | w_1, w_2 \in V' \wedge \{w_1, w_2\} \in E\}$. A clique or complete subgraph K of a graph G is a subgraph of G such that there is an edge between every two vertices in K. A graph containing n vertices and m edges is said to be of order n and its density is defined as $\frac{2m}{n(n-1)}$. This is the ratio between the maximum number of edges an order n graph can have and the number of edges the graph actually has.

A path of length r between vertices u and w in a graph $G = (V, E)$ is a sequence of vertices $v_0 - v_1 - ... - v_r$, such that $u = v_0$ and $w = v_r$, and $(v_{i-1}, v_i) \in E : \forall i, 1 \leq i \leq r$. A graph is connected if there is a path between every two vertices in the graph. A graph is disconnected if it is not connected. A subgraph H of G is a connected component, or a component, of G if it is a maximal connected subgraph of G, that is, there is no other connected subgraph of G of which H is a subgraph. A connected graph consists of only one connected component.

Given a graph $G = (V, E)$ a k-colouring is an assignment of the set of colours, $C = \{1, 2, ..., k\}$ to the vertices of G such that no two adjacent vertices are assigned the same colour. The chromatic number $\chi(G)$ of the graph is defined to be the smallest k for which G has a k-colouring and any such $\chi(G)$-colouring is known as an optimal colouring.

3 Exact algorithms

The problem of finding an optimal colouring of an arbitrary graph is known to be NP-hard. All known algorithms for this problem are therefore exponential time algorithms and can only be applied to small graphs without computationally heavy cost. Christofides [3], Brown [2] and Brelaz [1] describe three such algorithms and Peemoller [14] points out and corrects an error in Brelaz's algorithm. Kubale and Jackowski [11] compare the efficiency of these algo-

rithms on the basis of empirical experiments. They restrict themselves to graphs of order 60 and less and for the 60 vertex graphs fail to find optimal colourings except for the sparse and very dense graphs, which are easier to colour than graphs of medium density.

4 Heuristic algorithms

Because it is not always practical to find optimal colourings, many researchers have focused on developing heuristic algorithms that find near optimal colourings of graphs. For practical applications near optimal solutions for optimization problems are usually sufficient when the computational complexity of the problem is such that an exact solution is infeasible. Unfortunately, there are also theoretical limits on how closely we can approximate optimal solutions for graph colouring. Garey and Johnson [5] showed that constructing colourings using fewer than $r \cdot \chi(G)$ colours, with $r < 2$, is also NP-hard in the sense of [6]. The best guarantee at present is that we can colour any order n graph in polynomial time, using at most $O(\frac{n \cdot (\log\log n)^2}{\log^3 n} \cdot \chi(G))$ colours [7]. This means that at present no known polynomial time algorithm will produce, for all graphs, a colouring using less than some constant multiple of $\chi(G)$ colours, for *any* constant. Fortunately, NP-hardness is a worst case measure and on average heuristic algorithms can produce colourings much better than those given by the worst case limits. We will now describe these algorithms and compare them empirically.

Sequential algorithm

This is perhaps the most intuitive of heuristic algorithms. It is also known as the greedy algorithm because it colours the vertices of a graph in sequence in a greedy fashion.

1 Order the vertices of the input graph in some sequence, $v_1, v_2, ..., v_n$.
2 Colour v_1 with colour 1.
3 For all $i, 1 < i \leq n$, if vertices $v_1, v_2, ..., v_{i-1}$ have been coloured, colour v_i with the smallest colour c (a positive integer) not assigned to any vertex $v_1, v_2, ..., v_{i-1}$ that is adjacent to v_i.

Many refinements of this algorithm have been proposed to reduce the number of colours used. These all order the vertices, in step 1 of the above algorithm, according to some heuristic, instead of starting with a random sequence of vertices, in an attempt to force this colouring algorithm to first colour vertices that might prove difficult to colour later on.

Largest first ordering

This vertex ordering was suggested in [16]. It orders the vertices of the graph in decreasing order of degrees. The initial ordering of the vertices, before the sequential algorithm is applied, is such that $d(v_1) \leq d(v_2) \leq ... \leq d(v_n)$. The resulting algorithm is known as the largest first algorithm.

Smallest last ordering

This vertex ordering which can be used with the sequential algorithm was first described in [13]. It orders the vertices of the input graph $G = (V, E)$, in order $v_1, v_2, ..., v_n$ such that v_n is a vertex of minimum degree in G and for each $i, 1 \leq i < n$, v_i is a vertex of minimum degree in the subgraph $\langle\{v_1, v_2, ..., v_i\}\rangle$.

Dynamic largest first ordering

This is a version of the largest first ordering that orders vertices according to the number of neighbours it has in the uncoloured subgraph remaining at each stage and was suggested in [4]. The vertices are ordered $v_1, v_2, ..., v_n$ such that each v_i is a vertex of maximum degree in the subgraph $\langle\{v_i, v_{i+1}, .., v_n\}\rangle$.

Largest first with tie-breaking

This is a refinement of the largest first ordering and was first described in [4]. In the largest first ordering, when two vertices have the same degree, ties were broken, implicitly, by selecting randomly between the vertices of equal degree the order in which they will appear in the ordering. The largest first with tie-breaking ordering, when encountering vertices of equal degree, considers the sum of the degrees of all vertices adjacent to the given vertex and breaks ties by placing vertices with a higher sum before vertices with lower such sums.

Dsatur Algorithm

The Dsatur algorithm, due to [1], differs slightly from a sequential algorithm in that an initial ordering is not constructed before colouring starts, but instead the order in which the vertices are coloured is decided dynamically as the vertices are being coloured.

Brelaz defines the colour degree of a vertex in a partially coloured graph as the number of different colours used to colour adjacent vertices. This concept is used in the Dsatur algorithm below.

1 Colour a vertex of largest degree with colour 1.
2 Select a vertex v of maximum colour degree. If there is a tie, choose between these vertices by selecting any vertex of largest degree in the subgraph induced by the uncoloured vertices.
3 Colour the vertex v with the smallest colour not assigned to any vertex adjacent to v.
4 Stop if all vertices are coloured, else go to step 2.

The interchange techniques

All the algorithms described so far have in common that they colour each vertex, as it is encountered, with the smallest possible colour. Once a colour is assigned to a vertex that colour is not changed, regardless of whether that assignment later causes problems.

As a way of improving the performance of his smallest last algorithm, Matula [13] suggested trying interchanges on colours of neighbouring vertices if, at some stage of the colouring process, a new colour has to be introduced when

trying to colour a vertex. Using this idea he constructed a new algorithm called Smallest Last with Interchanges (SLI).

If G is a partially coloured graph and C_i and C_j are the sets of vertices assigned colours i and j respectively, called the colour classes of i and j, then let the subgraph $H_{i,j} = \langle C_i \cup C_j \rangle$. This subgraph need not be connected and a connected component of $H_{i,j}$ is called an i,j-component. Since there are no edges between vertices in one such component and any other vertices in C_i or C_j (by definition of a connected component), colours i and j can be interchanged for all vertices in this component and the resulting assignment would still be a legal colouring. Such an interchange is called an i,j-interchange on the component. Matula's smallest last with interchanges algorithm makes use of this concept:

1 Order the vertices in smallest last order.
2 Colour the graph with the sequential algorithm, but if a new colour, c, has to be introduced to colour some vertex v_k then do the following:
    3 Let C^1 be the set of those colours that were used to colour exactly one neighbour of v_k.
    4 Find two distinct colours $i, j \in C^1$ and find the two neighbours v_i and v_j of v_k that were coloured with i and j respectively, such that v_i and v_j are in different i,j-components.
    5 If such an i and j were found, perform an i,j-interchange on the component containing v_i. This will free colour i so that it could be used to colour v_k.
6 If such an (i,j)-pair was not found, colour v_k with colour c.

The above algorithm can be adapted and the interchange technique can be used with any of the previous colouring algorithms, including Dsatur. This is done by using the above algorithm whenever a new colour has to be introduced to colour a vertex. If an i,j-interchange is found then a new colour would not have to be introduced to colour the vertex.

To investigate the effect of attempting more interchanges, we have extended this algorithm somewhat by slightly changing step 4. When searching for two colours i and j with which to perform an i,j-interchange, instead of restricting both i and j to come from C^1, consider all i,j pairs where $i \in C^1$ and j is from the set of all colours used so far. A successful i,j-interchange will still free colour i so that it can be used to colour v_k. By increasing the number of colours from which j is chosen, the chance of finding an interchange that will avoid introducing a new colour is increased, but the complexity of the algorithm also increases. As with the interchange technique, this interchange2 technique can be combined with any of the previous algorithms.

Dunstan's algorithm

The following variation of the largest first algorithm was suggested by Dunstan, see [1]:

1 Order the vertices in a largest first order.
2 Set $c = 1$.
3 Visit the vertices in the order they appear in the vertex ordering. If a vertex has no neighbours that has been coloured with c, then colour it with colour c.
4 Order the uncoloured vertices in non-increasing order of degrees in the uncoloured subgraph.
5 If all the vertices are coloured, stop. Otherwise set $c = c + 1$ and go to step 3.

If the vertices are not reordered in step 4 of the above algorithm, then this would be equivalent to the largest first colouring algorithm. This algorithm differs from largest first in that the vertex ordering is not fixed before colouring begins, but is determined as the vertices are being coloured.

COSINE algorithm

The COSINE algorithm was developed by Hertz [8] and it produces optimal colourings, in polynomial time, for a class of graphs known as perfect graphs. A perfect graph is a graph with its chromatic number equal to the order of its largest clique. COSINE can also be used to construct good, but non-optimal, colourings for general graphs.

The algorithm below makes use of the contraction operation. The contraction of two non-adjacent vertices u and v in a graph G is obtained by deleting u and v from G and replacing it by a single vertex (uv) that is adjacent to all neighbours of u and to all neighbours of v.

1 Set $G_0 = G$ and $k = 0$.
2 Let $(xy)_0$ be any vertex in graph.
3 While G_k is not a clique do
    4 If there is a vertex not adjacent to $(xy)_k$ then set $x_k = (xy)_k$. Else choose for x_k any vertex not adjacent to all other vertices in G_k.
    5 Choose for y_k any vertex not adjacent to x_k for which the number of common neighbours of y_k and x_k is a maximum.
    6 Form G_{k+1} by contracting x_k and y_k into a vertex $(xy)_{k+1}$.
    7 Set $k = k + 1$.
8 Colour G_k (a clique) by assigning a different colour to each vertex.
9 Colour G by assigning to each vertex v the same colour as the vertex in G_k into which v has been contracted.

Recursive Largest First algorithm

Leighton [12] designed an graph coloring algorithm that is especially efficient, in terms of time required, for large graphs with low densities - the type of graphs usually encountered in practical scheduling problems. Leighton's largest first algorithm also produces good colourings for graphs of all densities.

1 Set $k = 1$.
2 While there are uncoloured vertices in G do:
    3 Let V' be the set of uncoloured vertices in G.
    4 Set $C = U = \phi$.
    5 Choose a vertex $v_0 \in V'$ that has the maximum number of neighbours in V'.

Table 1. Average number of colours used by each algorithm on graphs with density .25

Order	125	250	500
Algorithm	Colors used.		
SEQ	14.4	22.8	38.6
SEQI	13.4	21.5	36.0
SEQI2	12.5	20.6	35.3
LF	13.4	21.5	36.3
LFI	12.6	20.6	35.1
LFI2	11.9	19.4	33.3
DLF	12.6	21.0	34.8
DLFI	12.1	20.2	34.2
DLFI2	12.0	19.4	33.5
LFTB	13.7	21.6	36.2
LFTBI	12.2	20.5	35.0
LFTBI2	11.8	19.6	33.4
DUN	12.4	20.2	33.8
SL	13.8	21.8	37.2
SLI	12.5	20.6	35.3
SLI2	11.8	19.6	33.2
DS	12.1	19.3	32.9
DSI	11.8	19.9	33.7
DSI2	11.5	19.0	33.2
COS	11.5	18.6	31.0
RLF	11.2	18.3	30.3
BEST	11.1	18.3	30.3

6 Move v_0 from V' to C.
7 Move all neighbours of v_0 in V' to U.
8 While $V' \neq \phi$ do:
9 Choose a vertex $v \in V'$ that has a maximum number of neighbours in U. Ties are broken, if possible, by selecting a v with the minimum number of neighbours in V'.
10 Move v from V' to C.
11 Move all neighbours of v in V' to U.
12 Colour all vertices in C with colour k and set $k = k + 1$.

Experimental results

The algorithms described above were compared in terms of time and number of colours used to colour a number of random graphs. Graphs were generated with densities, 0.25, 0.5 and 0.75 and orders, 125, 250 and 500. For each order-density pair 10 graphs were generated and the average number of colours and time used by each of the algorithms tested to colour these graphs were recorded. In addition 2 graphs of order 1000 and density 0.5 were also used. The results are indicated in Tables 1–6. We have used the the common method of generating a random test graph with density around the desired density d [8, 10, 15]: For each possible edge in the graph a pseudo-random number r is selected from the interval $(0, 1]$ with uniform distribution. If $r < d$ then that edge is included in the random graph.

The algorithms appearing in Tables 1–6 are abbreviated as follows:

Table 2. Average number of colours used by each algorithm on graphs with density .5

Order	125	250	500	1000
Algorithm	Colors used.			
SEQ	24.4	42.0	72.5	125.0
SEQI	22.8	40.2	69.4	123.0
SEQI2	22.2	38.0	66.3	117.5
LF	23.7	39.7	69.1	122.5
LFI	22.3	38.3	67.6	120.5
LFI2	21.2	36.7	64.9	115.5
DLF	22.7	39.0	67.8	120.0
DLFI	21.8	37.9	66.2	119.0
DLFI2	21.0	36.6	63.7	114.0
LFTB	23.4	40.0	69.6	122.5
LFTBI	21.9	38.5	67.3	121.0
LFTBI2	20.9	36.3	64.2	115.0
DUN	22.2	37.8	66.4	117.5
SL	23.8	40.9	69.9	123.5
SLI	21.8	38.5	67.7	121.5
SLI2	20.9	36.3	64.4	114.0
DS	21.8	37.0	65.8	115.0
DSI	21.3	37.4	66.9	118.5
DSI2	20.7	36.5	64.5	116.0
COS	20.8	35.4	61.0	110.0
RLF	20.2	34.3	60.3	107.5
BEST	20.0	34.3	60.2	107.5

Table 3. Average number of colours used by each algorithm on graphs with density .75

Order	125	250	500
Algorithm	Colors used.		
SEQ	39.7	70.0	121.7
SEQI	36.7	64.2	115.5
SEQI2	34.9	62.3	110.0
LF	38.2	65.6	117.8
LFI	35.9	63.0	113.3
LFI2	34.0	60.4	107.9
DLF	37.2	65.4	115.2
DLFI	35.7	62.6	111.4
DLFI2	34.2	59.8	107.6
LFTB	38.1	66.2	117.7
LFTBI	35.4	62.8	113.2
LFTBI2	33.9	60.1	108.0
DUN	36.4	63.3	113.4
SL	38.2	67.1	119.3
SLI	35.9	63.1	113.8
SLI2	34.1	59.9	108.8
DS	34.7	62.2	111.5
DSI	34.7	62.4	113.5
DSI2	34.5	59.9	109.6
COS	34.9	59.6	106.0
RLF	33.1	58.4	104.1
BEST	32.8	58.2	104.1

Table 4. Average time used (in seconds) by each algorithm on graphs with density .25

Order	125	250	500
Algorithm	Time used.		
SEQ	0.11	0.26	1.12
SEQI	0.43	2.31	14.03
SEQI2	1.64	13.55	117.13
LF	0.09	0.32	1.24
LFI	0.48	2.49	14.65
LFI2	2.01	16.97	144.42
DLF	0.21	0.80	3.33
DLFI	0.46	2.38	11.23
DLFI2	1.76	13.33	106.40
LFTB	0.15	0.58	2.26
LFTBI	0.58	2.76	13.83
LFTBI2	2.47	17.92	145.36
DUN	0.40	2.07	12.45
SL	0.20	0.80	3.31
SLI	0.70	3.90	20.24
SLI2	2.66	21.30	175.21
DS	1.47	8.02	51.41
DSI	3.23	27.85	271.79
DSI2	6.07	63.42	648.64
COS	6.51	34.94	220.69
RLF	6.45	34.68	209.47

Table 5. Average time used (in seconds) by each algorithm on graphs with density .5

Order	125	250	500	1000
Algorithm	Time used.			
SEQ	0.12	0.53	2.22	9.10
SEQI	1.12	6.83	39.22	239.46
SEQI2	6.46	54.57	442.19	4144.65
LF	0.17	0.58	2.38	9.42
LFI	1.44	7.35	39.90	265.27
LFI2	8.55	68.63	563.57	4541.41
DLF	0.33	1.30	5.59	23.23
DLFI	1.22	6.43	35.88	200.01
DLFI2	6.08	53.93	456.25	3845.99
LFTB	0.27	1.04	4.43	17.88
LFTBI	1.63	7.75	43.71	266.00
LFTBI2	8.34	69.27	594.02	4997.40
DUN	0.92	5.32	33.45	230.36
SL	0.33	1.30	5.54	22.99
SLI	1.99	10.05	53.18	305.06
SLI2	10.51	77.15	626.12	5191.48
DS	3.66	23.97	176.66	1313.54
DSI	11.53	109.76	1042.34	10679.52
DSI2	24.89	221.61	2105.17	19339.29
COS	8.57	50.38	337.35	2364.18
RLF	8.01	46.38	328.88	2237.11

Table 6. Average time used (in seconds) by each algorithm on graphs with density .75

Order	125	250	500
Algorithm	Time used.		
SEQ	0.20	0.79	3.36
SEQI	3.06	16.24	98.17
SEQI2	18.77	157.14	1274.72
LF	0.23	0.85	3.51
LFI	3.75	18.63	106.99
LFI2	24.65	196.45	1619.22
DLF	0.46	1.84	7.83
DLFI	2.73	14.44	79.79
DLFI2	19.57	148.70	1332.99
LFTB	0.40	1.58	6.61
LFTBI	3.99	20.37	107.22
LFTBI2	25.36	189.58	1617.27
DUN	1.87	11.38	74.91
SL	0.47	1.82	7.76
SLI	5.75	27.42	142.68
SLI2	32.38	222.48	1789.94
DS	7.56	53.74	411.86
DSI	26.85	244.50	2494.09
DSI2	62.98	564.91	5283.24
COS	9.16	58.75	390.72
RLF	8.52	52.34	364.73

SEQ – sequential colour,
LF – largest first,
DLF – dynamic largest first,
LFTB – largest first with tiebreaking,
SL – smallest last,
DS – Dsatur,
DUN – Dunstan's algorithm,
COS – COSINE and
RLF – recursive largest first.

Appending an I or I2 to some of the above abbreviations indicates that the interchange or interchange2 technique was used with the algorithm. BEST shows the average number of colours used in the best colouring found for each graph by any of the algorithms tested.

From these tables we can see that RLF produces the best colourings with COS producing only slightly worse colourings, even though it is slower. RLF, in addition to producing the best colourings on average, almost always produced the best colouring for each graph. This is illustrated by the small differences between the average colours used by RLF and BEST, especially for the larger graphs. The sequential colouring algorithms use the most colours on average, but these algorithms are very fast compared to the others. The benefits of ordering the vertices according to some heuristic before applying SEQ can also be seen clearly; all these algorithms produced better colourings than SEQ, on average. The interchange algorithms produce colourings better than the corresponding sequential algorithms. The exception to this is the DS algorithm that often out performs DSI and DSI2 on the larger graphs. The interchange2 algorithms are computationally very ex-

pensive with running times larger than that of the RLF and COS algorithms while producing inferior colourings. This shows that there are more efficient ways to find improved colourings than to resort to extensive, uncontrolled, interchanging of colours. DUN also produces fairly good colourings quickly - much faster than the I2 algorithms and RLF, but uses slightly more colours.

Conclusions

Selecting a best algorithm depends on the specific application it is to be used for. Usually we want the best colourings possible, provided that we can find it in a reasonable time. RLF produces the best colourings, but for very large graphs it might be too slow and a faster algorithm might be needed. If speed is an issue than LF is a good algorithm to use. On average it uses only slightly more time than the fast SEQ but uses fewer colours. The two algorithms RLF and LF would appear to be the best 2 from those tested. LF would be preferable for for large graphs and in casses where timing is critical while RLF would be better for application where good colourings are of more importance than fast running times.

Recently it has been shown that general search heuristics like simulated annealing [10] and tabu search [9] perform very well when applied to the graph colouring problem. Both these techniques start with some initial colouring and then proceed to modify that colouring in an attempt to reduce the number of colours used. Hertz [8], for example shows that applying tabu search to an initial colouring generated by the COSINE algorithm usually results in a significant decrease in colours used to colour large graphs. Even though both these techniques can be started from an arbitrary initial colouring, they produce much better colourings when started from a good initial colouring. These general search heuristics are very time consuming compared to the heuristic algorithms described here. Simulated annealing is in the order of 100 times slower than the algorithms described here on graphs of order 1000 and density 0.5 [10]. They are useful when trying to improve on colourings found by other, faster, algorithms given sufficient time. Even though tabu search and simulated annealing is not discussed in any detail in this paper, the results given here are also useful for deciding on an algorithm with a suitable trade off between time used and quality of colourings produced to be used to create an initial colouring for one of these general search heuristics. For such algorithms, according to the results in this paper, the best algorithms to consider for generating an initial colouring would be LF or RLF.

References

1. D Brelaz. 'New methods to color the vertices of a graph'. *Comm. ACM*, **22**:251–256, (1979).
2. J R Brown. 'Chromatic scheduling and the chromatic number problems'. *Management Science*, **19**:456–463, (1972).
3. N Christofides. 'An algorithm for the chromatic number of a graph'. *Comput. J.*, **14**:38–39, (1971).
4. N Christofides. *Graph Theory, An Algorithmic approach*. Academic Press, London, 1975.
5. M R Garey and D S Johnson. 'The complexity of near optimal graph coloring'. *J. ACM*, **23**:43–49, (1976).
6. M R Garey and D S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
7. M M Halldórsson. 'A still better guarantee for approximate graph coloring'. *Information Processing Letters*, **45**:19–23, (1993).
8. A Hertz. 'COSINE: A new graph coloring algorithm'. *Operations Research Letters*, **10**:345–351, (1991).
9. A Hertz and D de Werra. 'Using tabu search techniques for graph coloring'. *Computing*, **39**:345–351, (1987).
10. D S Johnson, C R Aragon, L A McGeoch, and C Schevon. 'Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning'. *Operations Research*, **39**:378–406, (1991).
11. M Kubale and B Jackowski. 'A generalized implicit enumeration algorithm for graph coloring'. *Comm. ACM*, **28**:412–418, (1985).
12. F T Leighton. 'A graph coloring algorithm for large scale scheduling problems'. *Journal of Research of the National Bureau of Standards*, **85**:489–506, (1979).
13. D W Matula. *Graph Coloring Algorithms*. Academic Press, New York, 1972.
14. J Peemoller. 'A correction to Brelaz's modification of Brown's coloring algorithm'. *Comm. ACM*, **26**:595–597, (1983).
15. J Peemoller. 'Numerical experiences with graph coloring algorithms'. *European Journal of Operations Research*, **24**:146–151, (1986).
16. D J A Welsh and M B Powel. 'An upper bound to the chromatic number of a graph and its application to time-table problems'. *Comput. J.*, **10**:85–86, (1967).

Received: 12/93, Accepted: 2/94, Final copy: 12/94.