Binary Merge Model Representation of the Graph Colouring Problem

István Juhos[1], Attila Tóth[2], and Jano I. van Hemert[3]

[1] Dept. of Computer Algorithms and Artificial Intelligence, University of Szeged
[2] Department of Computer Science, University of Szeged (JGYTFK)
[3] National Institute for Mathematics and Computer Science, Amsterdam (CWI)

Abstract. This paper describes a novel representation and ordering model that, aided by an evolutionary algorithm, is used in solving the graph k-colouring problem. Its strength lies in reducing the number of neighbors that need to be checked for validity. An empirical comparison is made with two other algorithms on a popular selection of problem instances and on a suite of instances in the phase transition. The new representation in combination with a heuristic mutation operator shows promising results.

1   Introduction

Constraint satisfaction problems form a well studied class of problems [1]. Within this class the graph k-colouring problem is one of the most popular problems. The second DIMACS challenge [2] has been devoted to it, which has led to a standard file format for representing problem instances. Here we will use this file format, together with problem instances from the challenge.

In this paper we introduce a novel approach to representing candidate solutions for the graph k-colouring problem in an evolutionary algorithm. The approach is based on effective reduction rules. The order in which these reduction rules are executed determines the quality of the solution. This order will be determined by an evolutionary algorithm.

To show that the approach based on reduction rules has a performance that is competitive with other approaches, we perform a comparison with two algorithms that are known to perform well. The comparison is based on the effectiveness and efficiency of the algorithms.

The structure of the remainder of the paper is as follows. In the next section we give a definition and short overview of graph k-colouring. In Section 3, we introduce the new representation. This will be incorporated into an evolutionary algorithm described in Section 4, which we will compare empirically to other methods in Section 5. In Section 6 we provide conclusions.

2   Graph k-colouring

The problem class known as graph k-colouring is defined as follows, given a graph $\langle V, E \rangle$ where $V = \{v_1, ..., v_n\}$ is a set of nodes and $E = \{(v_i, v_j)|v_i \in V \wedge v_j \in V \wedge i \neq j\}$ is a set of edges, where every edge lies between two nodes. The graph

k-colouring problem is to colour every node in V with one of k colours such that no two nodes connected with an edge in E have the same colour.

Many algorithms to solve this problem were created and studied. Early algorithms are complete, i.e., they always find a solution if one exists. However, it was proven that such algorithms exhibit an exponential effort to either find a solution or proof that no solution exists, when the problem is scaled up. As a result many algorithms studied today use a stochastic process to guide them towards suboptimal solutions or, hopefully, towards an optimal solution. Examples of such methods include Tabu-search [3], simulated annealing [4] and ant colony optimisation [5].

One popular approach for dealing with graph k-colouring is evolutionary computation [6] Unfortunately, evolutionary algorithms are not necessarily very good in solving constraint satisfaction problems [7] as they may suffer from a number of flaws that keeps them from reaching optimal solutions [8]. One problem that is appropriate for graph k-colouring is when the problem at hand contains symmetry [9]. For the latter, symmetry is defined as the invariance to the solution when permuting he colours. Besides symmetry, other structural properties make it difficult to find solutions efficiently. As we know that graph k-colouring exhibits a phase transition depending on the ratio of constraints to the maximum number of possible constraints $\binom{n}{2}$ [10], where the number of solvable problem instances quickly drops to zero. At that transition constraint solvers require the most search effort to find solutions for solvable problem instances. Recent investigations [11] have shown that the size of the backbone in graph k-colouring may be a good explanation towards explaining the rise in difficulty at the phase transition.

3 Representing the Graph k-Colouring Problem

3.1 Binary Merge Model

Graph colouring algorithms make use of adjacency checking during colouring, which plays a significant role in performance. The number of checks depends on the representation of the problem. The model introduced here directly addresses this issue. Generally, when assigning a colour to a node all adjacent nodes must be scanned to check for an equal colouring, i.e., constraint check needs to be performed. Thus, we have to perform at least as many checks as the number of coloured neighbours and at most as many as $|V| - 1$. In the new model, this number of checks is between one and the number of colours used up to this point.

The *Binary Merge Model* (BMM) implicitly uses hyper-nodes and hyper-edges (see Figure 1(b)). A hyper-edge connects a hyper-node with other nodes, regardless whether it is normal or hyper. A hyper-node is a set nodes that have the same colour. If h is a hyper-node and v is a normal node that are connected by a hyper-edge then, and only then, h contains at least one normal node connected to v. Our algorithm concentrates on the relation between hyper-nodes and normal nodes. Thus, the adjacency checks can be done along hyper-edges instead of edges, whereby we spare many checks because during colouring, the initial set of edges is folded into hyper-edges.

The colouring of a graph is stored in a Binary Merge Table (BMT) (see Figure 2). Every cell (i,j) in this table takes as value either 0 or 1. The columns refer to the nodes and the rows refer to the colours. The value in cell (i,j) is 1 if and only if node j cannot be assigned colour i because of the edges in the original graph $\langle V, E \rangle$. In the initial colouring every node is assigned a unique colour by setting the (i,i) cells to 0.

If the graph is not a full graph, then it might be possible to decrease the number of necessary colours. This amounts to reducing rows in the BMT. To make reduction of the rows possible we introduce a Binary Merge Operation, which attempts to merge two rows whereby reducing the number of colours with one. It is successful only when the two nodes are not connected by an edge or by a hyper-edge. An example is found in Figure 1(b) and the related Figure 2.

Definition 1. *Binary Merge Operation $\cup$ merges an initial row i to an arbitrary (initial or merged) row r if and only if $(r,i) = 0$ (i.e., the hyper-node r is not connected to the node i) in BMT. If rows i and r can be merged then the result is the union of i and r.*

Formally, let I be the set of initial rows of BMT and R be the set of all possible $|V|$ size binary rows (vectors), then a binary merge operation is defined as,

$$\cup : R \times I \to R$$
$$r' := r \cup i, \text{ where } r', r \in R, i \in I, \text{ or by components}$$
$$r'_k := r_k \vee i_k, k = 1, \dots, |V|, \text{ where } \vee \text{ is the logical OR operator}$$

Regarding the time complexity of this operation, we can say that it is proportional to a binary OR operation on a register of l bits. If l is the number of bits in one such operation and under assumption that the time complexity of that operation is one, the merge of two rows of length n by l length parts takes $\lceil n/l \rceil$ to complete. If m is the number of rows left in MT, then $n - m$ binary merge operations are performed, where $m \in \{\chi, \dots, n\}$.

3.2 Permutation Binary Merge Model

The result, i.e., the colouring of a graph, after two or more binary merge operations, depends on the order in which these operations were performed. Consider the hexagon in Figure 1(a). In Figure 2 the BMTs are shown that correspond to the graphs in Figure 1. Now let the sequence of the rows $1, 4, 2, 5, 3, 6$ be the order in which rows are considered for the binary merge operation and consider the following merging procedure. Take the first two rows in the sequence, then attempt to merge row 4 with row 1. As these can be merged the result is $1 \cup 4$ (see Figure 1(b)). Now take row 2 and try to merge this with the first row, i.e., $(1 \cup 4)$. This is unsuccessful, so row 2 remains unaltered. The merge operations continues with the next rows 5, 3, and finally, 6. The allowed merges are $1 \cup 4$, $2 \cup 5$, and $3 \cup 6$. This sequence of merge operations results in the 3-colouring of the graph depicted in Figure 1(c). However, if we use the sequence $1, 3, 5, 2, 4, 6$ then the result is the 2-colouring in Figure 1(d) with the merges $1 \cup 3$, $(1 \cup 3) \cup 5$, $2 \cup 4$, and $(2 \cup 4) \cup 6$. The defined merge is greedy, it takes a row and tries to

find the first row from the top of the table that it can merge. The row remains unaltered if there is no suitable row. After performing the sequence P of merge operations, we call the resulting BMT the *merged* BMT.
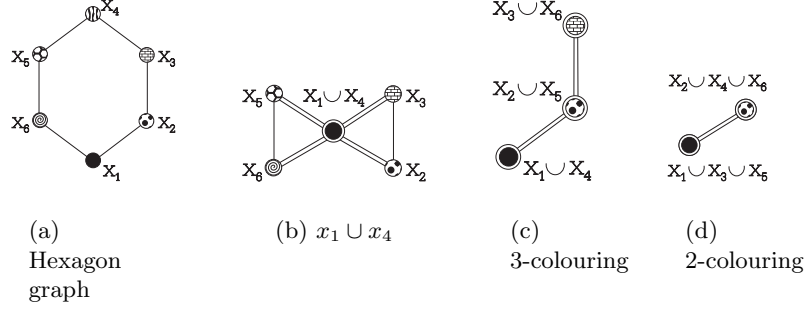

(a) Hexagon graph   (b) $x_1 \cup x_4$ (c) 3-colouring   (d) 2-colouring

Fig. 1. Example of the result of two different merge orders. Double-lined edges are hyper-edges and double-lined nodes are hyper-nodes.

(a)	x_1	x_2	x_3	x_4	x_5	x_6
r_1	0	1	0	0	0	1
r_2	1	0	1	0	0	0
r_3	0	1	0	1	0	0
r_4	0	0	1	0	1	0
r_5	0	0	0	1	0	1
r_6	1	0	0	0	1	0

(b)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
r_2	1	0	1	0	0	0
r_3	0	1	0	1	0	0
r_5	0	0	0	1	0	1
r_6	1	0	0	0	1	0

(c)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_4$	0	1	1	0	1	1
$r_2 \cup r_5$	1	0	1	1	0	1
$r_3 \cup r_6$	1	1	0	1	1	0

(d)	x_1	x_2	x_3	x_4	x_5	x_6
$r_1 \cup r_3 \cup r_5$	0	1	0	1	0	1
$r_2 \cup r_4 \cup r_6$	1	0	1	0	1	0

Fig. 2. The Binary Merge Tables that correspond to the graphs in Figure 1

Finding a minimal colouring for a graph k-colouring problem using the BMT representation and binary merge operations comes down to finding the sequence of merge operations that leads to that colouring. This can be represented as a sequence of candidate reduction steps using the greedy approach described above. The permutations of this representation form the Permutation Binary Merge Model. We proof that this model is a valid representation of the graph k-colouring problem, i.e., that any valid k-colouring can be represented with it.

Lemma 1 (Equivalence). *An arbitrary valid k-colouring of a graph G can be associated with a merged Binary Merge Table.*

Proof (constructive). Consider a $k \times n$ BMT, where n is the number of the nodes and k is the number of colours used in the colouring. Let a cell $(c, i) = 1$ if and only if node i has an adjacent node that is coloured with c, otherwise let it contain 0.

Lemma 2 (Solvability). *If O is an optimal colouring of the graph G, i.e., it uses the k colours, then there exists a sequence of merge operations P that reduces the initial Binary Merge Table such that O is obtained by P.*

Proof (constructive). Suppose O is an optimal solution. Construct a merged BMT
*B according to O, this exists because of Lemma 1. The binary merge operation
is reversible if we keep track of the merges. Let this reverse operation be called
"unmerge". Unmerge all merged rows in B until the number of rows is equal to
the number of nodes of G. At first, apply unmerge to the last row of B as many
times as it contains nodes. Then, continue to unmerge the previous row and so
on. By applying the unmerge operation sequences we get a sequence of merge
operators P that leads back to O. Note that there may exist several sequence that
lead to O because unmerge can be applied in any optional order.*

4   Evolutionary algorithm to guide PBMM

The search space of the model in Section 3.2 is of size $n!$ and contains many
local optima. We use an evolutionary algorithm (PBMM-EA)[4] to search through
the space of permutations. The genotype is thus the permutations of the rows.
The phenotype is a valid colouring of the graph after using the greedy method
on the permutation to select the order of binary merge operations.

An intuitive way of measuring the quality of an individual p in the population
is by counting the number of rows remaining in the final BMT. This equals to
the number of colours $k(p)$ used in the colouring of the graph, which needs to be
minimised. If we know that the optimal colouring is χ then we may normalise
this fitness function to $g(p) = k(p) - \chi$.

This function gives a rather low diversity of the individuals in a population
because it cannot distinguish between two individuals that use an equal number
of colours. We address this problem by introducing a new multiplier. This mul-
tiplier is based on the heuristic that we want to get rid of highly constraint rows
in order to have more chance of successful merges later. This involves merges of
rows where many 1s are merged. Let $z(p)$ denote the number of 1s in the final
BMT B then the fitness function becomes $f(p) = (k(p) - \chi)z(p)$.

We use a steady-state evolutionary algorithm with 2-tournament selection,
which incorporates elitism. The stop condition is that either an individual p exists
with $f(p) = 0$ or that the maximum number of 3 000 generations is reached.
The latter means that the run was unsuccessful, i.e., the optimal colouring was
not found. The population size depends on the problem and will be set in the
experiments.

Two variation operators are used to provide offspring. First, the 1-point
order-based crossover (OX1) [13, in Section C3.3.3.1] is applied. Second, the
other variation operator is a mutation operator. We test two different mutation
operators yielding two variants of the evolutionary algorithm. The probability of
crossover depends on the choice of mutation operator. During tuning, we tried
probabilities from 0 to 1 in several combinations. In case of DGS mutation, the
results showed that a higher probability of mutation and a lower probability of
crossover lead to better solutions.

PBMM-EA/SWAP This variant first applies with a probability of 0.6 OX1 and
then with a probability of 0.3 the simple swap mutation, which selects at
random two different items in the permutation and then swaps them.

---

[4] PBMM-EA is developed using the EO [12] framework

PBMM-EA/DGS *Difficulty Guided Swap.* This variant applies with a probability of 0.3 OX1 and then always applies a heuristic mutation operator that is similar to the simple swap mutation, but it always chooses a node related to the last merged row and forces it to be ahead of its position in the permutation. To accomplish this it chooses at random a previous row identifier for swap. The idea being that these last merged rows are the most difficult to merge.

5 Empirical comparison

5.1 Benchmark algorithms

Stepwise Adaptation of Weights SAW is put forth in [6] as a very promising technique for colouring graph 3-colouring problems by showing its competitiveness against an early version of DSATUR and the Grouping Genetic Algorithm by Falkenauer [14]. We provide a brief overview of the concept of SAW and refer to [6] for a detailed description of the algorithm and specific versions of it.

The basic idea behind SAW is to learn on-line about the difficulty of constraints in a problem instance. This is achieved by keeping a vector of weights that associates the weight w_i with constraint i. In the context of graph k-colouring every edge $i \in E$ is assigned a weight w_i. These weights are initialised as one. A basic evolutionary algorithm is used to solve a given problem instance. Every T generations it is interrupted in order to change the vector of weights using the best individual of the current population. Every constraint i violated by this individual is incremented by one. Then, the evolutionary algorithm continues using this new vector. The fitness of an individual equals the sum of the weights of all the constraints it violates. By adapting this fitness function using the vector of weights, a dynamic process occurs that may prevent the evolutionary algorithm to get stuck in local optima. The SAW-EA is the same version as described in [6].

CLQ-DSATUR is an improvement [15] over the original DSATUR, which originates from Brélaz [16]. It uses a heuristic to dynamically change the ordering of the nodes and then applies the greedy method to colour the nodes. It works as follows, one node with the highest saturation degree, i.e., number of adjacent colours, is selected from the uncoloured subgraph and it is assigned the lowest indexed colour that still yields a valid colouring. If there exists more than one most saturated nodes, the algorithm chooses a node with the highest degree among them. At first CLQ-DSATUR tries to find the largest clique, i.e., the subgraph that is a full-graph, in the graph. It allocates different colours for the nodes in the clique. Then, uncoloured nodes are dynamically ordered by saturation of colour and subproblems are created as in the original DSATUR algorithm. This uses backtracking to discover a valid colouring. More details about this algorithm is found in [17].

5.2 Means of comparisons

The performance of an algorithm is expressed in its effectiveness and its efficiency in solving a problem instance. The first is measured using the success ratio, which

is the amount of runs where an algorithm has found the optimum divided by the total number of runs. The second is measured by keeping track of how many constraint checks are being performed on average, for a successful run. This measure is independent of hardware and programming language as it counts the number of times an algorithm requests information about the problem instance, e.g., it checks if an edge exists between two nodes in the graph. This check, or rather the number of times it is performed, forms the largest amount of time spend by any constraint solver. A *constraint check* is defined for each algorithm as checking whether the colouring of two nodes is allowed (satisfied) or not allowed (violated).

The two variants of PBMM-EA and the SAW-EA are both stochastic algorithms. Therefore we perform 10 independent runs with different random seeds for each problem instance. The number of constraint checks are averaged over these 10 runs. The complete method CLQ-DSATUR suffices with 1 run.

5.3 DIMACS Challenge

We would like to present the performance of the PBMM-EA variants on different kinds of real life problem instances. The test suite consists from problem instances taken from "The Second DIMACS Challenge" [2] and Trick's graph colouring repository [18]. In Table 1 we show the results for 34 problem instances.

The second goal is to test the supposed benefit of the difficulty guided mutation operator over the simple swap mutation. For all runs we set the population size for the PBMM-EA variants to two, but for the queens instances this gives very poor results due to the equal degree in the graph, which is also visible for the benchmark algorithm. Consequently we increase the population size to 12 for queen6_6 and to 70 for queen7_7 and queen8_8 to show the difference on such graph structures. This sets a guideline for the next test suite.

For large graphs the PBMM-EA variants are much faster than the CLQ-DSATUR and SAW-EA. The two PBMM-EA variants do not differ significantly except for the miles and queens graphs where the difficulty guided swap variant shows its improvement over the simple swap mutation. Also, the latter is not able to always find a solution for two of the queen graphs.

5.4 Phase transition

To test the algorithms on really difficult graphs, we generate, using Culberson's generator [10], a test suite of 3-colourable graphs with 200 nodes that show a phase transition. The graphs are equipartite, i.e., for each colour the same amount of nodes, to some approximation, will be coloured. It consists of 9 groups of graphs where each group has 25 instances. The connectivity is changed from 0.020 to 0.060 by steps of 0.005 over the groups. The population size for PBMM-EA/DGS is set to 100 for these difficult graphs.

In Figure 3 we show the results for SAW-EA, PBMM-EA/DGS and CLQ-DSATUR. The latter is clearly the most useful algorithm as it always finds a solution and it takes the least number of constraint checks to do so. The results for the three

Table 1. Average number of constraint checks required for solving various problem instances using CLQ-DSATUR, SAW-EA, PBMM-EA/SWAP and PBMM-EA/DGS. Entries with "–" refer to where the algorithm never found the chromatic number, in all other cases the success ratio is one. The last three entries are used to show the difference in the two mutation operators.

Graph	$|V|$	$|E|$	χ	CLQ-DSATUR	SAW	PBMM-EA SWAP	DGS
fpsol2.i.1	496	11 654	65	1 311 037	49 028	13 831	17 793
fpsol2.i.2	451	8 691	30	1 078 861	1 745 560	12 196	11 513
fpsol2.i.3	425	8 688	30	973 200	1 414 220	69 964	10 276
inithx.i.1	864	18 707	54	3 640 097	155 996	15 552	21 490
inithx.i.2	645	13 979	31	2 328 921	739 201	8 447	12 475
inithx.i.3	621	13 969	31	2 150 844	304 910	9 725	12 135
mulsol.i.1	197	3 925	49	292 942	6 265	5 964	8 525
mulsol.i.2	188	3 885	31	229 907	21 707	4 110	5 667
mulsol.i.3	184	3 916	31	224 934	51 042	4 874	5 619
mulsol.i.4	185	3 946	31	228 474	128 130	4 084	5 606
mulsol.i.5	186	3 973	31	232 835	11 120	4 141	5 536
zeroin.i.1	211	4 100	49	289 622	13 165	6 670	8 040
zeroin.i.2	211	3 541	30	289 191	65 053	11 870	4 942
zeroin.i.3	206	3 540	30	277 504	52 493	22 556	11 197
anna	138	493	11	89 024	15 579	2 903	1 242
david	87	406	11	34 557	56 872	9 957	2 493
homer	561	1 629	13	1 484 278	11 906 400	71 572	5 038
huck	74	301	11	25 783	1 210	788	1 015
jean	80	254	10	29 939	11 390	746	949
miles250	128	387	8	76 961	983 894	21 556	3 051
miles500	128	1 170	20	89 025	9 724 950	191 011	20 398
miles750	128	2 113	31	136 866	7 922 930	946 683	103 376
miles1000	128	3 216	42	462 986	15 476 000	1 551 235	164 312
miles1500	128	5 198	73	404 051	886 155	167 487	67 721
myciel3	11	20	4	798	32	49	73
myciel4	23	71	5	4 273	135	120	186
myciel5	47	236	6	22 104	447	265	447
myciel6	95	755	7	117 163	5 920	708	955
myciel7	191	2 360	8	638 536	52 997	4 074	3 245
games120	120	638	9	101 070	3 227	1 492	1 926
queen5_5	25	160	5	8 797	8 835	4 630	2 000
queen6_6	36	290	7	202 170	–	740 550	139 711
queen7_7	49	476	7	1 015 209	3 195 320	6 326 912[1]	744 273
queen8_8	64	728	9	57 989 844	–	30 412 779[2]	9 558 255

[1] success ratio of 0.9
[2] success ratio of 0.4
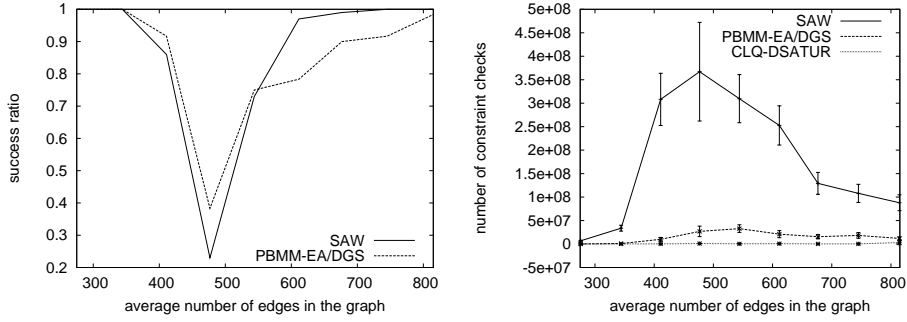
(a) Success ratio, note that CLQ-DSATUR always finds a solution

(b) Average number of constraint checks with 95% confidence intervals

Fig. 3. Results for 225 random equipartite graph 3-colouring problems of size 200 where for each problem instance 10 independent runs are performed

algorithms in Figure 3(b) are significantly different and allow us to make a clear ranking on efficiency for the three algorithms.

Both evolutionary algorithms show a sharp dip in the success ratio in the phase transition (see Figure 3(a)), which is accompanied with a rise in the average number of constraint checks. PBMM-EA/DGS starts out over 34 times faster than SAW-EA. When the number of edges increases this difference decreases to 7 times as fast. CLQ-DSATUR seems to have the least problems with this phase transition. However, it shows a sudden rise in constraint checks at 814 edges.

6   Conclusions

We introduced a new model to tackle the graph k-colouring problem that serves as an efficient representation that helps to reduce constraint checks. We verify this efficiency by embedding it in an evolutionary algorithm and performing an empirical comparison on two test suites. The results from the first test suite show a performance in speed and accuracy that is quite favourable, especially on the large real world problem instances with more than 400 nodes. However in the second test, where we look at equipartite graphs in the phase transition, the success ratio shows the typical dip we often observe for stochastic algorithms.

Set against one other popular evolutionary algorithm for constraint satisfaction, the stepwise adaptation of weights method, and against an improved version of a popular complete method, CLQ-DSATUR, the model combined with an evolutionary algorithm that also uses a heuristically guided mutation operator yields promising results. For larger problem instances it is even up to 50 times faster than SAW-EA and CLQ-DSATUR. For difficult equipartite graphs, i.e., that lie in the peak of the phase transition, it is slower and less effective than CLQ-DSATUR, but it is much faster than SAW-EA.

The benefits of the new representation depend highly on the quality of the evolutionary algorithm that is guiding it and the structure of the graph. Further work is needed to tune this evolutionary algorithm and to map the performance against different graph properties.

Acknowledgements

References

1. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)
2. Johnson, D., Trick, M.: Cliques, Coloring, and Satisfiability. American Mathematical Society, DIMACS (1996)
3. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39** (1987) 345–351
4. Chams, M., Hertz, A., Werra, D.D.: Some experiments with simulated annealing for coloring graphs. European Journal of Operational Research **32** (1987) 260–266
5. Vesel, A., Zerovnik, J.: How good can ants color graphs? Journal of computing and Information Technology - CIT **8** (2000) 131–136
6. Eiben, A., van der Hauw, J., van Hemert, J.: Graph coloring with adaptive evolutionary algorithms. Journal of Heuristics **4** (1998) 25–46
7. van Hemert, J.: Comparing classical methods for solving binary constraint satisfaction problems with state of the art evolutionary computation. In *et al.*, S.C., ed.: Applications of Evolutionary Computing. Volume 2279 of LNCS. (2002) 81–90
8. van Hemert, J., Bäck, T.: Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction. In *et al.*, J.M., ed.: Proceedings of Parallel Problem Solving from Nature VII. Number 2439 in LNCS (2002) 23–32
9. Marino, A., Damper, R.: Breaking the symmetry of the graph colouring problem with genetic algorithms. In Whitley, D., ed.: Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference. (2000) 240–245
10. Culberson, J.: Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, University of Alberta, Dept. of Computing Science (1992)
11. Culberson, J., Gent, I.: Frozen development in graph coloring. Theoretical Computer Science **265** (2001) 227–264
12. Keijzer, M., Merelo, J., Romero, G., Schoenauer, M.: Evolving objects library. Internet Publication (2002)
13. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York (1997)
14. Falkenauer, E.: Genetic Algorithms and Grouping Problems. John Wiley (1998)
15. Trick, M.: Easy code for graph coloring. Internet Publication (1995) Last Modified: November 2, 1994.
16. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM **22** (1979) 251–256
17. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. INFORMS Journal on Computing **8** (1996) 344–354
18. Trick, M.: Computational series: Graph coloring and its generalizations (2003) http://mat.gsia.cmu.edu/COLORING03.
19. Juhos, I., Tóth, A., Tezuka, M., Tann, P., van Hemert, J.: A new permutation model for solving the graph k-coloring problem. In: Kalmár Workshop on Logic and Computer Science. (2003) 189–199