An Analysis of Heuristics for Vertex Colouring

Marco Chiarandini[1] and Thomas Stützle[2]

[1] University of Southern Denmark, Campusvej 55, Odense, Denmark
marco@imada.sdu.dk
[2] IRIDIA, Université Libre de Bruxelles, Av. Roosevelt 50, Brussels, Belgium
stuetzle@ulb.ac.be

Abstract. Several heuristics have been presented in the literature for finding a proper colouring of the vertices of a graph using the least number of colours. These heuristics are commonly compared on a set of graphs that served two DIMACS competitions. This set does not permit the statistical study of relations between algorithm performance and structural features of graphs. We generate a new set of random graphs controlling their structural features and advance the knowledge of heuristics for graph colouring. We maintain and make all algorithms described here publically available in order to facilitate future comparisons.

Keywords: graph coloring, heuristics, experimental analysis.

1 The Graph Colouring Problem

The graph-vertex colouring problem (GCP) is a central problem in graph theory [1]. It consists in finding an assignment of colours to vertices of a graph in such a way that no adjacent vertices receive the same colour. Graph colouring problems arise in many real life applications like register allocation, air traffic flow management, frequency assignment, light wavelengths assignment in optical networks, and timetabling [4].

In the GCP, one is given an undirected graph $G = (V, E)$, with V being the set of $|V| = n$ vertices and E being the set of $|E| = m$ edges. A *k-colouring* of G is a mapping $\phi : V \mapsto \Gamma$, where $\Gamma = \{1, 2, \ldots, k\}$ is a set of $|\Gamma| = k$ integers, representing the colours. A k-colouring is *proper* if for all $[u, v] \in E$ it holds that $\varphi(u) \neq \varphi(v)$; otherwise it is *improper*. If for some $[u, v] \in E$ it is $\varphi(u) = \varphi(v)$, the vertices u and v are said to be *in conflict*. A k-colouring can also be seen as a partitioning of the set of vertices into k disjoint sets, called *colour classes*. In the decision version of the GCP, also called the *(vertex) k-colouring problem*, we are asked whether for some given k a proper k-colouring exists. In the optimisation version, we are asked for the smallest number k, called the *chromatic number* $\chi(G)$, for which a proper k-colouring exists. For general graphs, the decision version of the GCP problem is NP-complete.

In this work, we consider algorithms for solving the optimisation version of the GCP on general graphs. This version can be algorithmically approached by solving a sequence of k-colouring problems: an initial value of k is considered and

P. Festa (Ed.): SEA 2010, LNCS 6049, pp. 326–337, 2010.

each time a proper k-colouring is found, the value of k is decreased by one. The chromatic number is found when for some k the answer to the decision version is no. In this case, $\chi(G) = k + 1$. If a proper colouring cannot be found but no proof of its non-existence is given, as it is typically the case with heuristic algorithms, $k + 1$ is an upper bound on the chromatic number. For reviews of heuristic approaches to the GCP we refer to [3,4,9,15].

New heuristic and exact algorithms are commonly tested on a set of graphs maintained by M. Trick [19] and used for the two DIMCAS challenges on the GCP in 1996 and 2002 [12,13]. The results of algorithm comparisons, above all those concerning heuristics, are often hard to interpret. Apparently, in most articles on the GCP published in major journals [4], a newly presented heuristic outperforms some others on some instances. Nevertheless, generally, no insight is given on which instance features are separators of algorithm performance. Culberson et al. [7] provided an interesting analysis about what makes a graph hard to colour. The aim of our research is to exploit the insights on instance hardness from Culberson's work, to extend the analysis to some of the best known and best performing algorithms for the GCP, and to give indications on which algorithms perform best for specific classes of graphs. In this way we hope to advance both the understanding of algorithm behaviour and the dependence of algorithmic performance on structural properties of graphs. We use Culberson's generator [7] to produce new random graphs. We make the code of most of the algorithms studied available in an online compendium `http://www.imada.sdu.dk/~marco/gcp-study`. All experiments were run on a 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB of RAM.

2 Instance Generation

Culberson's random generator [7] allows us to create families of graphs of different structure and size. Structure may be induced by (i) imposing a graph to have a given number of independent sets (i.e., hiding a k-colouring) and by (ii) influencing the variability of the size of these independent sets. In our analysis, we control these two structural aspects of graphs together with the edge distribution. In future studies it might be possible to control also the limit of the size of an induced clique or the girth of the graph.

Hidden colour classes and variability of their size. In its *smooth k-colourable* modality, the generator controls the variability of the size of the colour classes. A graph is generated by assigning each vertex to the independent set with label $\lfloor kx(ax + 1 - a) \rfloor$, where $a \in [0, 1]$ is a parameter and x is a random number from the interval $[0, 1)$. For $a = 0$, the size of the independent sets tends to be nearly equal and the graph be quasi equi-partite, while for $a = 1$ the size tends to vary considerably. This structural feature of the graph was shown to be relevant for understanding differences in the instance hardness for algorithms [7]. We generated graphs using `variability` $\in \{0, 1, \texttt{no}\}$, where 0 and 1 are the values assigned to a and `no` indicates that no hidden colouring and biased size of colour classes is enforced.

Edge distribution. We considered the following three types. ***(i) Uniform graphs***. An edge between a pair of vertices (u, v) is included with a fixed probability p. These graphs are denoted by G_{np}, and by G_{knp} if a k-colouring is hidden. ***(ii) Geometric Graphs***. These graphs are created by disposing n points uniformly at random in a two dimensional square with coordinates $0 \leq x, y < 1$. Vertices in the graph correspond to the n points and an edge is associated to a pair of vertices (u, v) if their Euclidean distance, $d(u, v)$ is smaller or equal to a value r. We denote these graphs by U_{nr}, and by U_{knr} if a k-colouring is hidden. (The use of the letter G for uniform graphs and U for geometric graphs, instead of the viceversa, is common in the literature, see for example [11].) According to [11], geometric graphs have "inherent structure and clustering". ***(iii) Weight Biased Graphs***. These graphs are generated by first assigning vertices to independent sets. Then, a weight w is assigned to all $\binom{n}{2}$ pairs of vertices, except those in the same hidden independent set, which are assigned weight zero. Vertex pairs are selected as edges with a probability proportional to their weight. When an edge is added to the graph, weights are decreased in such a way that the formation of large cliques becomes unlikely. This procedure is controlled by two parameters, α and γ, that we set equal to 0 and 1, respectively, as recommended in [7]. The process terminates when either all weights are zero, or when $\lfloor p\binom{n}{2} \rfloor$ edges have been selected. As a side effect, the vertices may have small variability in vertex degree. These graphs are among the hardest to colour [7]. The edge density of the resulting graph, measured as the ratio between number of present edges and the number of edges in the corresponding complete graph, depends on the parameters p and w. In order to attain graphs of edge density $\{0.1, 0.5, 0.9\}$, we set p equal to $\{0.1, 0.5, 0.9\}$ and w equal to $\{2, 114, 404\}$ if $n = 500$ and equal to $\{4, 230, 804\}$ if $n = 1000$ (see [7] for details on the choice of these values). We denote these graphs by W_{np}, and by W_{knp} if a k-colouring is hidden.

We generated 1260 graphs of size 500 and 1000. We summarise the characteristics of the graphs in Table 1a, organised by five factors: size, type, edge density, variability and hidden colouring. These factors are parameters of the instances and, using a statistical terminology, stratify the experimental units in subgroups. In each subgroup, corresponding to specific combinations of the five factors, we have 5 graphs constructed with different random seeds in the generator. Table 1b gives aggregate statistics on the number of graphs considered. Note that due to random decisions in the generation of the edges, the value of k in a hidden colouring is only an upper bound to the chromatic number of the graph.

Selection of a time limit. For a comparison of heuristic algorithms, stopping criteria need to be fairly defined. We decided to use a classical local search algorithm as benchmark: our implementation of TabuCol by de Werra (1990) [20] that we refer to as TS_{N_1}. It uses the improved dynamic tabu list by [8]. In our implementation, this algorithm performs the evaluation of a neighbour in $O(1)$ by means of an auxiliary matrix that stores delta values and that can be updated in $O(n)$ by a simple scan of the vertices adjacent to the vertex that changed colour. In addition, we maintain both an adjacency matrix and an adjacency list for the representation of the graph, using either of the two at

Table 1. Table (a) shows how the 1260 graphs generated are distributed over the features. The number of graphs of size 500 is 520 and those of size 1000 is 740. Table (b) and (c) show the number of graphs per class given by size, edge density and graph type. Table (d) gives the time limits in seconds for the instances differentiated by size (rows) and density (columns).

Size	Type	Density	Variability	Hidden colouring	Tot. graphs
1000	G	0.1	0	$\{5, 10, 20\}$	15
			1	$\{5, 10, 20\}$	15
			no	–	10
		0.5	0	$\{10i : i = 2, \ldots, 14\}$	75
			1	$\{10i : i = 2, \ldots, 14\}$	75
			no	–	10
		0.9	0	$\{20, 50i : i = 1, \ldots, 5\}$	30
			1	$\{20, 50i : i = 1, \ldots, 5\}$	30
			no	–	10
	U	0.1	0	$\{10i : i = 2, \ldots, 5\}$	20
			1	$\{10i : i = 2, \ldots, 5\}$	20
			no	–	10
		0.5	0	$\{10i : i = 2, \ldots, 20\}$	95
			1	$\{10i : i = 2, \ldots, 20\}$	95
			no	–	10
		0.9	0	$\{100i : i = 1, \ldots, 6\}$	30
			1	$\{100i : i = 1, \ldots, 6\}$	30
			no	–	10
	W	0.1	0	$\{10, 20\}$	10
			1	$\{10, 20\}$	10
			no	–	10
		0.5	0	$\{10i : i = 2, \ldots, 9\}$	40
			1	$\{10i : i = 2, \ldots, 9\}$	40
			no	–	10
		0.9	0	$\{30, 90, 150, 220\}$	20
			1	$\{30, 90, 150, 220\}$	20
			no	–	10

(a)

$n = 500$	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
G	60	90	90
U	60	150	150
W	30	75	60

(b)

$n = 1000$	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
G	90	180	180
U	120	150	180
W	30	120	60

(c)

	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
$n = 500$	60	120	180
$n = 1000$	155	465	720

(d)

convenience. Similarly, we represent a colouring both as a mapping by means of an array and as a collection of colour classes, each one being implemented by a binary search tree. The other algorithms we implement use, as far as possible, the same data structures.

A common stopping criterion for TS_{N_1} is after $I_{\max} = 10^4 n$ iterations [8], which is also the one we used. In fact, we found this stopping criterion to be large enough so that the algorithm obtains limiting behaviour and that further improvements become unlikely. In particular, we found that setting the termination criterion to $I_{\max}$ corresponds to missing 26% of cases where a better colouring could still be found. This empirical probability drops below 3% after $10 \times I_{\max}$ and below 0.02% after $50 I_{\max}$. However, $10 \times I_{\max}$ implies an increase of a factor of 10 in computation time. For more detailed results on this analysis we refer to `http://www.imada.sdu.dk/~marco/gcp-study`. The actual time limits used for the heuristic algorithms are given in Table 1d; we divided them according to edge density since this property has a strong impact on computation time. For each density we took the median time observed.

3 Experimental Analysis

Exact Algorithms. The famous Brelaz's backtracking search algorithm with forward checking [2,17], here denoted as Ex-DSATUR, was shown to be substantially competitive with a more involved column generation approach [16].

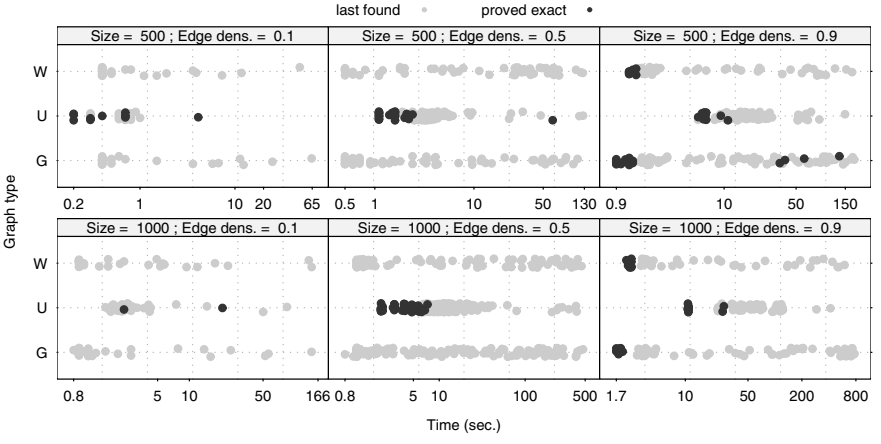

Fig. 1. The figure represents the exit status of Ex-DSATUR on the 1260 random graphs when the computation is truncated at the time limits of Table 1d. For each graph, the corresponding point indicates the computation time at which a last proper colouring was found. If the solution is proved optimal, the point is black, otherwise it is grey. A jitter effect is added to the y coordinates of the data to make points distinguishable. The plots are logarithmic in the x-axis.

An implementation of Ex-DSATUR by Mehrotra and Trick [16] including clique pre-colouring can be found online [18]. We tested this implementation of the algorithm on the 1260 instances using the same time limit as reported in Table 1d. In Figure 1, we summarise the behaviour of Ex-DSATUR. The plots give an account of the time at which the best proper colouring is found. If the colouring is proved optimal, the corresponding point is plotted in black.

The first observation is that an exact solution is found either quickly (in less than 10 sec.), or it is hardly found afterwards. Despite the large size, some graphs are easily solvable by Ex-DSATUR. Chances to find solvable graphs are highest for Geometric graphs and in particular for edge density equal to 0.5. Graphs of type G and W are solvable exactly only for the high edge density and graphs become harder to be solved exactly if the number of hidden colours increases. A closer look at the data reveals that the graphs solved to optimality have $\omega(G)$, the size of the largest clique, very close to $\chi(G)$. A graph for which $\omega(G) = \chi(G)$ is called *perfect* and can be recognised and coloured in polynomial time [5,10]. For perfect and quasi-perfect graphs, the large clique found heuristically by Ex-DSATUR can be used to prune effectively the search tree.

Construction heuristics. Construction heuristics are fast single pass heuristics that construct a proper colouring and finish. We compared the performance in terms of run-time and solution quality of the two most famous ones: DSATUR [2] and RLF [14], which we implemented to run in $O(n(n+k)+m)$ and $O(n^2+km)$, respectively. We include in the analysis also a random order greedy heuristic, ROS, that runs in $O(nk + m)$. Note that for dense graphs, like those in this

work, and assuming $k = O(n)$, the complexity of these algorithms becomes $O(n^2)$, $O(n^3)$ and $O(n^2)$, respectively.

We run the three algorithms once on each instance. For run-times, we compute the 95% confidence intervals of the mean value by means of the Tukey test for all pairwise comparisons. For quality, we rank the results in terms of colours within each instance and we compute the 95% confidence intervals of the mean rank value by means of the Friedman test for unreplicated two-way designs with Bonferrori correction [6].

On Uniform and Weight Biased graphs, RLF is clearly, and by a large margin, the best algorithm in terms of solution quality. We do not show the results. This indication is somehow interesting because we might expect to see differences between RLF and DSATUR varying the size of the hidden colour classes, but this is not the case. On Geometric graphs, instead, differences in solution quality with respect to DSATUR are not always statistically significant. In Figure 2, we report the all-pairwise comparisons with an indication of the computation time on the y-axis. Only Geometric graphs of size 1000 are used to generate the plot because differences in computation time are more pronounced in this setting and no difference in solution quality due to size was observed. Interestingly, there are classes of graphs in which DSATUR performs better than RLF, but a clear pattern does not arise. Since Geometric graphs have a clique number close to the chromatic number, this result may indicate that coloring looking at the saturation number of vertices (DSATUR) may be a sufficiently good strategy when cliques are relevant for the final result. In addition, we observe that the performance of DSATUR with respect to RLF improves as edge density increases. Finally, in terms of run-time, RLF exhibits a much stronger dependency on edge density, as captured by the asymptotic analysis. This effect can be evinced in Figure 2 observing the row-wise growth of computation time for RLF. The same pattern is present also on Uniform and Weight Biased graphs thus trading off in those graphs with the outperformance in quality terms.

Stochastic Local Search algorithms. We implemented heuristic algorithms based on stochastic local search (SLS) concepts. Beside TS_{N_1}, we include $\mathsf{TS}_{\mathsf{VLSN}}$, a tabu search on a very large scale neighbourhood [3], SA_{N_6}, the simulated annealing by Johnson et al. (1991) based on Kempe chains [11], $\mathsf{MC\text{-}TS}_{N_1}$, a min-conflict heuristic [3], Nov^{+}, inspired by Novelty algorithms for satisfiability problems [3], HEA, the hybrid evolutionary algorithm by Galiner and Hao (1999) [8], GLS, a guided local search algorithm [3] and ILS, an iterated local search algorithm [3]. In addition, we include our reimplementation of XRLF [11], a parametrised version of RLF that uses an exact colouring when the set of remaining colours is sufficiently small. In order to reduce the variance of the results and to isolate the effect of these heuristics, we start all them, except XRLF, from the same initial solution produced by RLF.

The influence of hidden colours. For several generated graphs, the SLS algorithms were able to easily find much better colorings than the hidden colouring. In a few cases the difference reached 200 colours while, with few exceptions,

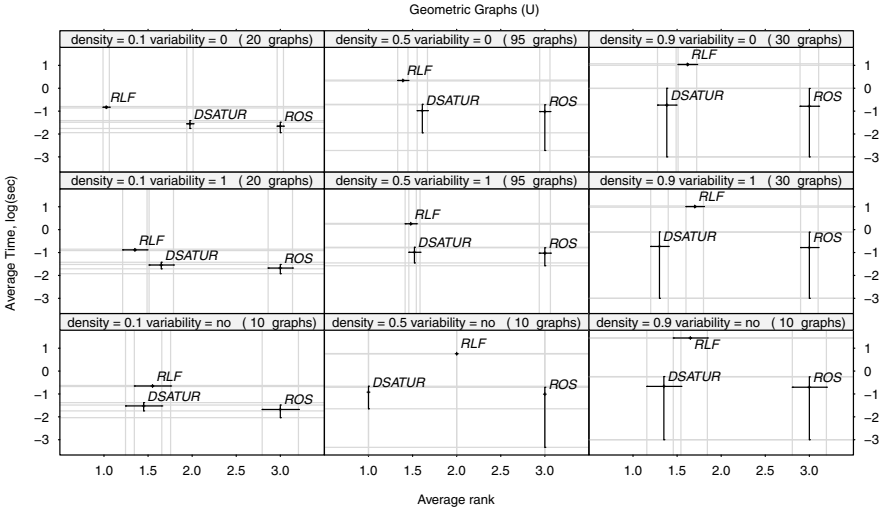

Fig. 2. All-pairwise comparisons in a run-time versus solution quality plot for construction heuristics on Geometric Graphs. Two algorithms are statistically significantly different in one of the two criteria if their intervals in the corresponding axis do not overlap. A base-10 logarithm transformation is applied on the time axis.

the best SLS algorithms always reach the hidden upper bound. Hence, the attempt to use hidden colourings to measure hardness of an instance is somehow a failure, due to the difficulty of hiding colourings. One interesting phenomenon is, however, worth reporting. On two of the 38 classes, we observed the phenomenon depicted in Figure 3. The plot shows the error relative to the hidden upper bound attained by the SLS heuristics on graphs of size 1000, density 0.5, variability 0, and type G and W. Since a hidden colouring exists, it should be possible to reach it on all those graphs, i.e., all curves should reach a zero error or approximate it. Nevertheless, the curves peak for some values of the hidden colourings, indicating that there are some values of k for which the instances are much harder to solve than usual. The region, which exhibits this phenomenon, arises at about 80 hidden colours for graphs of type G and between 70 and 80 colours for graphs of type W. It is interesting to note that this phenomenon affects also algorithms such as SA_{N_6} and XRLF, which do not solve sequences of k-colouring problems. The same effect was not observed in the corresponding graph classes of size 500 and it is unclear whether it exists for graphs of sizes not considered here.

Interaction between algorithms and graph features. We visualize the interactions between algorithms and graph features in Figure 4. Lines are added to emphasize the trend and not to interpolate data. The less parallel these lines are the stronger the interaction effect is. The strongest effect seems to be produced by edge density. Note that we omit hidden colours from now since this feature cannot be easily recognised a priori.

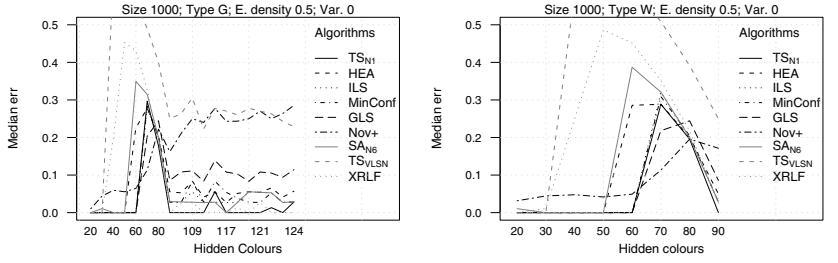

Fig. 3. The figure depicts the algorithm performance (expressed in terms of relative error computed over the hidden upper bound) and the number of hidden colours. A hidden colour indicates that a proper colouring with that or a lower number of colours certainly exists.

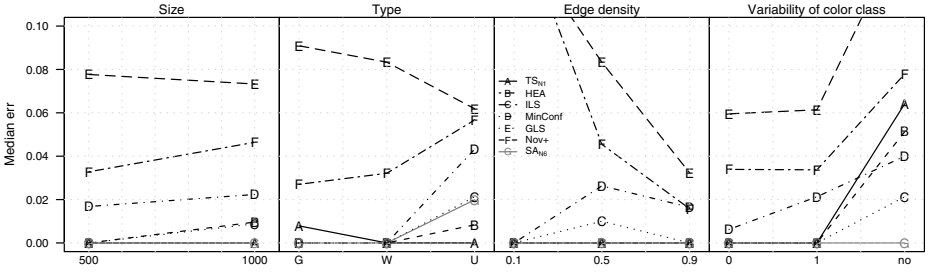

Fig. 4. Interaction plots between algorithms and stratification variables. The worst algorithms, XRLF and TS$_{VLSN}$, are removed in order to gain a clearer insight on the high performing algorithms.

Nonparametric analysis. We run all heuristics once on each graph until the corresponding time limit was exceeded. As a first step, we intended a parametric analysis fitting a linear model on a relative error transformation of the response. The analysis hinted at a strong significance of the interactions summarized in Figure 4. However, the necessary assumptions for a parametric analysis were found to be violated and therefore we proceeded with a nonparametric analysis by means of a rank transformation as mentioned above. An analysis based on permutation tests was also considered and results were in line with those presented below. Our preference for the rank-based analysis is due to its higher power. Due to the situation depicted in Figure 4, we separated the analysis into graph classes determined by the features: type, edge density, and variability. Graphs of size 500 and 1000 were instead aggregated, since the influence on the relative order of the algorithms is negligible. The results shown by means of confidence intervals on ranks derived by the Friedman test are reported in Figure 5. The following are the main conclusions from the analysis.

(i) On Uniform graphs, TS$_{N_1}$ is the best algorithm on four scenarios and, except for one scenario, no other algorithm does significantly better. Therefore,

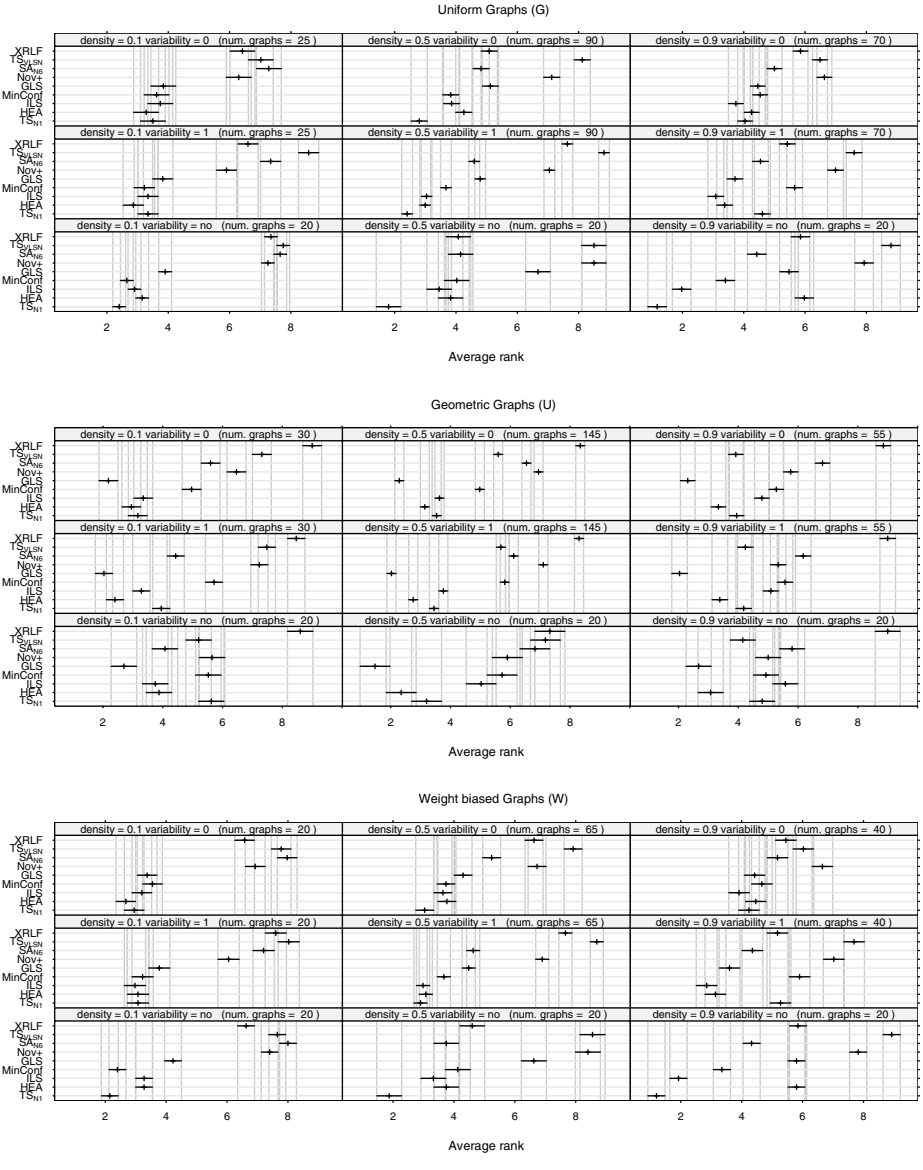

Fig. 5. The diagrams show average ranks and corresponding confidence intervals for the 9 algorithms considered. The analysis is divided into three main classes of graphs: Uniform graphs, Geometric graphs and Weight Biased graphs. Inside each class, subclasses are determined by the combinations of the stratification variables: edge density and independent set variability. Graphs of size 500 and 1000 are aggregated and the number of instances considered is reported in the strip text of the plots. Two algorithms are statistically significantly different if their intervals do not overlap.

we indicate TS_{N_1} as the preferable method for this class. It appears particularly powerful with graphs of density 0.5. The second best is ILS, which outperforms TS_{N_1} in the scenarios with density 0.9 and variability 1.

(ii) On the Weight Biased graphs, results are very similar to Uniform graphs. The best algorithm is TS_{N_1}, which is significantly the best in 3 scenarios, while ILS is the second best and again outperforms TS_{N_1} on graphs with density 0.9 and variability 1.

(iii) On the Geometric graphs, the overall best algorithm is clearly GLS. Differences from the second best are significant in 6 out of 9 scenarios, while in the other 3 scenarios it is not significantly dominated. The second best algorithm is HEA. The variability of the hidden colour classes and the edge density have a weak impact on these graphs, at least for the performance of the best algorithms, and results could be aggregated in a unique diagram, in which case GLS is significantly the best algorithm.

(iv) Scenarios with variability 0 and 1 exhibit very similar performance of the algorithms suggesting that this effect is not relevant.

Improvement over Ex-DSATUR. Ex-DSATUR is an exact algorithm with exponential worst case, but it can be stopped at any time and it returns a feasible solution. In Figure 6 we compare the approximate solutions returned by SLS algorithms and Ex-DSATUR after the same run-time as that for TS_{N_1}. There is evidence that the SLS algorithms obtain colourings that are much better than those of Ex-DSATUR, reaching improvements by even more than 150 colours.

Improvement over RLF. The use of SLS heuristics gives a significant improvement over the initial solution of RLF. The results, reported in the online compendium, show that on Uniform and Weight Biased graphs the improvement increases considerably with size and edge density. In the case of Weight Biased graphs, the improvement can reach 105 colours. On Geometric graphs, the improvement is smaller, above all on graphs with edge density 0.9; a possible explanation is that RLF finds near-optimal solutions on these graphs.

4 Discussion

We reported our computational experience on algorithms for colouring large, general graphs. We considered 1260 graphs that were constructed by controlling several structural parameters in order to gain a better insight into the relationship between algorithm performance and graph features. We took into account graph size, edge density, type of graph, and characteristics of colour classes. These are features that may be recognised a priori in practical contexts.

We showed that a straightforward backtracking algorithm can solve instances of even 1000 vertices if these have the favourable property of being nearly *perfect*. The simple and fast construction heuristic RLF is considerably better than DSATUR in terms of solution quality, although its running time is more strongly affected by the number of edges in the graph. If a faster heuristic is needed and the graph is Geometric, then probably DSATUR is the best choice. If we can

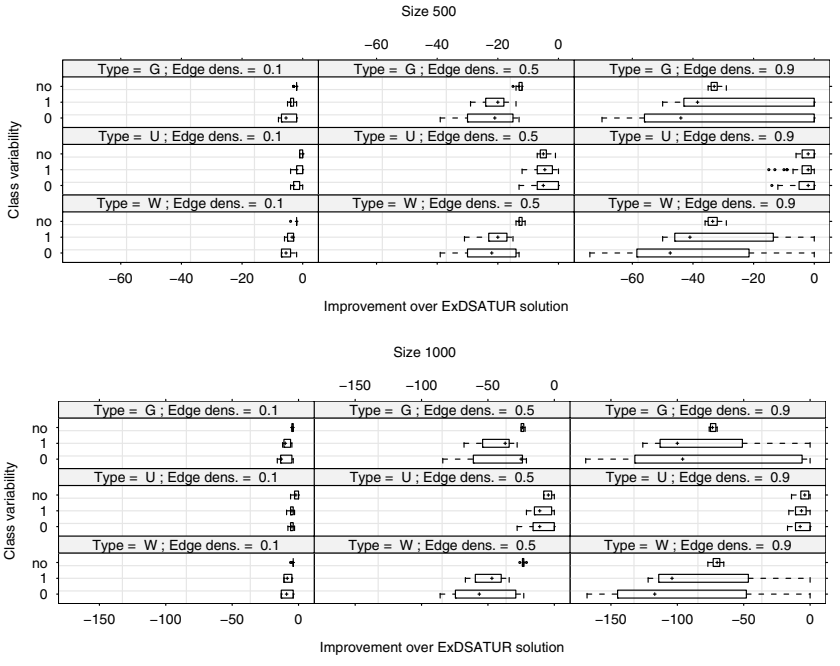

Fig. 6. Box-plots of differences for each graph class between the best solutions found by the SLS algorithms and the solution produced by the Ex-DSATUR heuristic

trade computation time for solutions quality, then SLS algorithms can lead to a large improvement. Our comparison includes some well known, high performing SLS algorithms for GCP. On the *Uniform and Weight Biased random graphs*, TS_{N_1} is the best algorithm with graphs of density 0.5 and it is not dominated at edge density 0.1 and 0.9. This result is important because it indicates TS_{N_1} as a relevant benchmark to be used when new algorithms are introduced and therefore we make its implementation available online. On *Geometric random graphs*, GLS is clearly the best algorithm. This class of graphs has the clique number very close to the chromatic number and the result leads us to conjecture that GLS works well for graphs with this property. Weight Biased graphs are, instead, graphs in which large cliques are avoided and indeed GLS is performing poorly.

References

1. Bondy, J., Murty, U.: Graph Theory. Graduate Texts in Mathematics, vol. 244. Springer, Heidelberg (2008)
2. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM 22(4), 251–256 (1979)
3. Chiarandini, M., Dumitrescu, I., Stützle, T.: Stochastic local search algorithms for the graph colouring problem. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics, pp. 63-1–63-17. Chapman & Hall/CRC, Boca Raton (2007)

4. Chiarandini, M.: Bibliography on graph-vertex coloring (2010), http://www.imada.sdu.dk/~marco/gcp
5. Chudnovsky, M., Cornuéjols, G., Liu, X., Seymour, P., Vušković, K.: Recognizing Berge graphs. Combinatorica 25(2), 143–186 (2005)
6. Conover, W.: Practical Nonparametric Statistics, 3rd edn. John Wiley & Sons, New York (1999)
7. Culberson, J., Beacham, A., Papp, D.: Hiding our colors. In: Proceedings of the CP 1995 Workshop on Studying and Solving Really Hard Problems, Cassis, France, September 1995, pp. 31–42 (1995)
8. Galinier, P., Hao, J.: Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
9. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Computers & Operations Research 33, 2547–2562 (2006)
10. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2), 169–197 (1981)
11. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. Operations Research 39(3), 378–406 (1991)
12. Johnson, D.S., Mehrotra, A., Trick, M.A.: Special issue on computational methods for graph coloring and its generalizations. Discrete Applied Mathematics 156(2), 145–146 (2008)
13. Johnson, D.S., Trick, M. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993. DIMACS Series in DMTCS, vol. 26. American Mathematical Society, Providence (1996)
14. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. Journal of Research of the National Bureau of Standards 84(6), 489–506 (1979)
15. Malaguti, E., Toth, P.: A survey on vertex coloring problems. International Transactions in Operational Research, 1–34 (2009)
16. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS Journal on Computing 8(4), 344–354 (1996)
17. Peemöller, J.: A correction to Brelaz's modification of Brown's coloring algorithm. Communications of the ACM 26(8), 595–597 (1983)
18. Trick, M.: Network resources for coloring a graph (1994), http://mat.gsia.cmu.edu/COLOR/color.html (last visited: February 2005)
19. Trick, M.: ROIS: Registry for optimization instances and solutions (2009), http://mat.tepper.cmu.edu/ROIS/ (last visited: December 2009)
20. de Werra, D.: Heuristics for graph coloring. Computing Supplement 7, 191–208 (1990)