A Wide-Ranging Computational Comparison of High-Performance Graph Colouring Algorithms

R. Lewis[1], J. Thompson[1], C. Mumford[2], J. Gillard[1]
[1]Cardiff School of Mathematics,
email: lewisR9|thompsonJM1|gillardJW@cf.ac.uk
[2]Cardiff School of Computing,
email: c.l.mumford@cs.cf.ac.uk

January 13, 2012

Abstract

This paper reviews the current state of the literature surrounding methods for the general graph colouring problem and presents a broad comparison of six high-performance algorithms, each belonging to one of the main algorithmic schemes identified. Unlike many previous computational studies in graph colouring, a large range of both artificially generated and real-world graphs are considered, culminating in over 40,000 individual trials that have consumed more than a decade of computation time in total. The picture painted by the comparison is complex, with each method outperforming all others on at least one occasion; however, general patterns are also observed, particularly with regards to the advantages of hybridising local-search techniques with global-based operators.

Keywords: Graph Colouring; Metaheuristics; Combinatorial Optimisation.

1 Introduction

The graph colouring problem is a classical NP-hard combinatorial optimisation problem with practical applications in various branches of operational research including school and university timetabling [14, 16, 56], sports scheduling [59], frequency assignment [6, 78], compiler register allocation [18], and short-circuit testing [40]. Graph colouring involves assigning a "colour" to each vertex in a graph such that adjacent vertices are allocated different colours, with the number of colours being minimised [46, 65]. Formally, given an undirected simple graph $G = (V, E)$, with vertex set V and edge set E, the task is to assign each vertex $v \in V$ an integer $c(v) \in \{1, 2, \ldots, k\}$ such that:

- $c(v) \neq c(u)$ $\forall \{v, u\} \in E$; and
- k is minimised

In addition to its underlying computational complexity, the graph colouring problem presents considerable challenges due to the sheer variety of graph structures that can be encountered in practice. Some graphs, for example, are generated so that the degrees of the vertices are modelled by a particular probability function, or such that the vertices and edges correspond to specific geometric objects and configurations. Graphs can also be produced so that their solutions fulfil certain properties such as the number of colours that are needed, and/or the number of vertices assigned to each colour class. Graphs arising from practical problems can also exhibit certain features: e.g. graphs that model timetabling problems may contain various large cliques,[1] or

[1]In graph terminology, a clique is a subset of vertices $V' \subseteq V$ that are mutually adjacent (i.e. $\forall u, v \in V', \{u, v\} \in E$)

exhibit some clustering of the vertices [72]. If the existence of such structures is known beforehand then algorithms might be tailored to exploit this information, perhaps leading to improvements in performance. However, the identification of such structures is not always easy and, indeed, is often a computationally intractable task in itself [49].

The past four decades have seen very many articles proposing algorithms for the graph colouring problem, ranging from simple constructive techniques [79] to sophisticated hybrid metaheuristics [37, 45, 61, 74]. Comparisons of such algorithms have generally been conducted empirically, and a useful development occurred in 1992 with the organisation of the Second DIMACS Implementation Challenge [2], which resulted in a suite of differently structured graph colouring problems being placed into the public domain. In addition, various problem generators have also been made available online for practitioners who are interested in testing their algorithms on particular types of problem instance [4]. However, despite these developments we suggest that it is often difficult to assess from the literature the relative benefits of different graph colouring algorithms, particularly due to the following reasons.

- Although many studies make use of publicly available problem instances or generators, experimental conditions usually alter from paper to paper. In particular, researchers often perform different numbers of trials-per-instance, use different performance metrics, and/or impose different computation limits. In cases where sets of benchmark instances are used, researchers also often direct their efforts towards gaining good results on just a subset of problems.

- Related to this, many research papers only report results achieved at the imposed cut-off point, giving little indication of how an algorithm progresses from the initial solution towards the final reported solution.

- In some cases researchers also enhance results using existing knowledge on the structure of a problem instance to tune their algorithms. For example, the minimal number of colours needed to colour the graph might be known in advance, or details may be available as to how the problem was generated. Of course, on the one hand tuning an algorithm may help to give readers a true appreciation of its capabilities; however, such actions also go against the general aim of an approximation algorithm, which is to produce good quality results while assuming no such specialist knowledge on behalf of the user.

Given these issues, in this work our aim is to provide an instructive comparative analysis on a range of different graph colouring algorithms, some of which are currently claimed to be "state of the art". To achieve this, we consider the performance of six methods, using at least one from each of the main algorithmic schemes identified in the next section. We also avoid some of the pitfalls above by (a) analysing algorithm performance using a platform-independent performance metric; (b) using a very large sample of problem instances (over 5000); and (c) by executing algorithms blindly, meaning that no problem knowledge is assumed on behalf of the user before a run is started.

In the next section we start by reviewing some of the more prominent algorithms proposed for the graph colouring problem over the past 40 years, identifying common schemes. Other reviews can be found in [38, 45, 54]. Section 3 then contains a more detailed description of the six algorithms considered in our comparison, with Section 4 detailing the results. Section 5 concludes the paper.

2 Existing Methods for General Graph Colouring

Before conducting a review of the literature, we outline some terms and notation used in this paper.

- A candidate solution to a graph colouring problem is said to be *complete* if all of its vertices $v \in V$ are assigned a colour $c(v) \in \{1, \ldots, k\}$, otherwise it is a *partial* solution.

- If in a solution there exists a pair of nodes $u, v \in V$ such that $\{u, v\} \in E$ and $c(v) = c(u)$, then this is termed as a *clash* between v and u. If a solution contains no clashes, then it is a *proper* solution, otherwise it is *improper*.

- A solution is considered *feasible* if and only if it is both complete and proper.
- The *chromatic number* of a graph, denoted χ, refers to the minimal number of colours needed for a solution to be feasible. (Note that the problem of identifying χ is itself an NP-hard problem [49]).
- A subset of vertices $U \subseteq V$ is termed an *independent set* if and only if $\forall u, v \in U$, $c(u) = c(v)$ and $\{u, v\} \notin E$ (i.e. an independent set is a set of mutually non-adjacent vertices assigned to the same colour).

A feasible solution can thus be viewed as a set of independent sets $\mathcal{U} = \{U_1, \ldots, U_{|\mathcal{U}|}\}$ such that $\bigcup U_i = V$. A feasible solution with $|\mathcal{U}| = \chi$ is *optimal*.

High-performance graph colouring algorithms can be loosely grouped into two classes: *constructive techniques*, which build solutions in a step-by-step manner by assigning each vertex to a colour in turn, perhaps using various heuristic and backtracking operators; and *optimisation-based techniques*, which navigate their way through a space of candidate solutions, attempting to optimise an objective function defined on this space. The latter class might also be further divided into a number of sub-classes according to how these spaces and objective functions are defined. These are considered in turn below.

2.1 Constructive Methods

The earliest proposed methods for the general graph colouring problem used construction heuristics to quickly build feasible, though not necessarily optimal solutions. Perhaps the most simple of these heuristics is the *first-fit* (or *greedy*) algorithm which operates by taking each vertex in turn and assigning it to the lowest indexed colour where no clash is induced, creating new colours when necessary. The number of colours in solutions produced in this way depends on the way in which the vertices are ordered, although it is known that there will always be at least one ordering that results in an optimal solution [57]. The earliest heuristic for choosing such an ordering is due to Welsh and Powell [79] who suggest sorting the vertices according to their degrees, colouring those of highest degree first.

Later, an improvement on this heuristic was suggested by Brélaz [10] in his DSATUR algorithm, where the next vertex to be coloured is determined dynamically by choosing the uncoloured node that currently has the largest number of distinct colours assigned to adjacent vertices (a worst case analysis of this heuristic on 3-colourable graphs can be found in [73]). Around the same time a different constructive approach was also suggested by Leighton [55], called the Recursive Largest First (RLF) algorithm. In contrast to the above methods, this algorithm operates by constructing each independent set $U_i \in \mathcal{U}$ one by one, using the process shown in fig. 1. In this pseudocode X defines the set of uncoloured vertices, and $Y \subseteq X$ is the set of uncoloured vertices that can be feasibly assigned to the independent set U_i currently being constructed. It can be seen that the only decision to be made in each stage of construction is which vertex $v \in Y$ to add next (line (4)). In Leighton's case, the first vertex chosen for each colour U_i is the vertex with the largest degree in the sub-graph induced by set Y; the remaining vertices are then chosen by selecting $v \in Y$ with the largest degree in the sub-graph induced by $B = X \setminus Y$. Other heuristics might also be applied in practice, however.

Other early methods for graph colouring were based around tree search (backtracking) strategies such as the methods of Brown [11] and Korman [52]. Such schemes may ultimately involve enumerating a significant proportion of the solution space which, of course, is unlikely to be achievable in reasonable time for many graphs. However, the performance of these algorithms can improve if they are augmented with additional operators such as the dynamic reordering of uncoloured vertices in each forward step of the algorithm [52], or by using appropriate heuristics to prioritise the traversal of more promising regions of the search tree. A comparison of such algorithms is presented by Kubale and Jackowski in [53]. Later, a complete algorithm based on column generation was proposed by Mehrotra and Trick [62], and more recently an incomplete algorithm based on backtracking has been presented by Prestwich [71].

```
    RLF (U = ∅, X ← V, i ← 0)
(1) while X ≠ ∅ do
(2)     i ← i + 1, U_i ← ∅, Y ← X
(3)     while Y ≠ ∅ do
(4)         Choose v ∈ Y
(5)         X ← X \ {v}
(6)         U_i ← U_i ∪ {v}
(7)         Y ← Y \ (Γ_Y(v) ∪ {v})
(8)     U ← U ∪ {U_i}
```

Figure 1: The RLF algorithm for graph colouring. Here, $\Gamma_Y(v)$ denotes the subset of vertices in Y that are adjacent to vertex v.

2.2 Optimisation-based Methods

2.2.1 Feasible-only Search Spaces

Over the past 15 years-or-so one optimisation-based strategy for graph colouring has been to use simple constructive algorithms as the underlying search operators for navigating the space of feasible solutions. One prominent example is the Iterated Greedy algorithm of Culberson and Luo [26], which operates by making repeated applications of the first-fit algorithm. As the authors point out, given a feasible solution $\mathcal{U}$, if a permutation of the $|V|$ vertices is formed such that those belonging to the same independent sets are placed in adjacent positions, then feeding this permutation into the first-fit algorithm will result in a new feasible solution $\mathcal{U}'$ such that $|\mathcal{U}'| \leq |\mathcal{U}|$. The algorithm thus operates by taking a feasible solution, forming a permutation as described, and then applying the first-fit algorithm to produce a new feasible solution before repeating the process indefinitely. Culberson and Luo [26] investigate different schemes for ordering the independent sets before copying them one-by-one into the permutation, including ordering the sets according to their size, or due to their ordering in previous solutions (see also Section 3.5).

Other approaches based in this scheme include the ant colony optimisation algorithm of Costa and Hertz [23] and the evolutionary-based methods of Mumford [67] and Erben [33]. Erben also proposes the use of the following heuristic-based cost function:

$$f_1 = \frac{\sum_{U_i \in \mathcal{U}} \left(\sum_{v \in U_i} d(v) \right)^2}{|\mathcal{U}|} \tag{1}$$

where $\sum_{v \in U_i} d(v)$ gives the total degree of all vertices assigned to the independent set U_i. Such a function is purported to allow selection pressure to be sustained in a population for longer during a run compared to the more obvious choice of using the number of colours $|\mathcal{U}|$ as a quality measure. Most recently, a hill-climbing technique based on the Iterated Greedy approach of Culberson and Luo but also complemented with a local search operator was proposed by Lewis [57] (see Section 3.5). This method was shown to outperform the algorithms of Erben [33] and Culberson and Luo [26] in most cases.

2.2.2 Spaces of Complete, Improper Colourings

Perhaps the most widely used optimisation scheme for graph colouring involves exploring spaces of complete, improper colourings. Such methods typically start by proposing a fixed number of colours k, with each vertex then being assigned to one of these colours using heuristics, or possibly at random. During this assignment there may exist vertices that cannot be assigned to any colour without inducing a clash, but these will be assigned to one of the colours anyway. Thus we are left with a complete, improper k-colouring, whose cost can be evaluated:

$$f_2 = \sum_{\forall \{v,u\} \in E} g(v,u) \qquad \text{where} \qquad g(v,u) = \begin{cases} 1 & \text{if } c(v) = c(u) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In such algorithms the aim is to make alterations to a solution such that the number of clashes, defined by f_2, is reduced to zero. If this is achieved, k might then be decremented and the process restarted; alternatively if all clashes cannot be eliminated, k can be increased. Note that at each setting for k, we are thus attempting to solve the NP-complete decision variant of the graph colouring problem: "Given a positive integer k, is there a feasible k-colouring of the vertices?".

Perhaps the first algorithm to make use of the above strategy was due to Chams et al. in 1987 [19], where simulated annealing was combined with a simple neighbourhood operator that altered the colour of a single vertex at each iteration. Soon after this paper, Hertz and de Werra [44] proposed a similar algorithm called TabuCol based on tabu search [43], which was then a relatively new optimisation technique. This method used the same neighbourhood structure as [19] but generally produced better results. Indeed as is noted by Galinier and Hertz [38], where a good exposition of this method is given, TabuCol is still used in contemporary algorithms for graph colouring. Another early approach, again using simulated annealing but this time under a variety of different search spaces and resultant neighbourhood operators was also suggested by Johnson in 1991 [47].

In recent years many other methods for exploring the space of complete, improper k-colourings have been proposed, including techniques based on evolutionary algorithms (EAs) [30, 32, 36, 37], iterated local search [21, 69], GRASP algorithms [54], variable neighbourhood search (VNS) [8], and ant colony optimisation [74].[2] Two of the most notable examples of these, particularly due to the quality of results reported, are the hybrid evolutionary algorithm of Galinier and Hao [37] and the AntCol algorithm of Thompson and Dowsland [74], both which use population-based methods combined with a variant of Hertz and de Werra's TabuCol algorithm. The idea in both of these cases is to use the population-based elements to guide the search over the long term, gently directing it towards favourable regions of the search space (global search), while the local search element is used to identify high quality solutions within these regions. We consider these particular algorithms further in Section 3.

2.2.3 Spaces of Partial, Proper Colourings

A further strategy for graph colouring, which has perhaps received less attention historically, involves exploring the space of proper, *partial* solutions. This scheme again involves stipulating a fixed number of colours k at the outset; however, when vertices are encountered that cannot be feasibly assigned to a colour they are transferred to a set of uncoloured vertices S. Thus, the aim is find a solution with $S = \emptyset$ and, if this goal is achieved, k can be reduced as before. An early algorithm of this type was proposed by Morgenstern [66] who used a simulated annealing-based algorithm that operated on a population of candidate solutions using combinations of different neighbourhood operators. Later an effective, though somewhat simpler, method was proposed by Blochliger and Zufferey [9], who used tabu search as the basis for their PartialCol algorithm. The latter approach operates in a very similar fashion to the TabuCol algorithm of Hertz and de Werra [44] and uses a cost function $f_3 = |S|$ together with a neighbourhood operator suited to the underlying solution space (see Section 3.2). Most recently, high-quality results based on this scheme have also been reported by Malaguti et al. [61] who, like Galiner and Hao [38], hybridise an exploitative local search method with a more exploratory, evolutionary-based approach. One notable feature of this algorithm its use of the heuristic-based cost function $f_4 = \sum_{v \in S} d(v)$. This is based on the proposals of Morgenstern [66], and like eq. (1) is purported to sustain selection pressure. Note that Blochliger and Zufferey [9] also experimented with this cost function with their earlier PartialCol algorithm; however, they found that it only bought about better solutions in few cases, while in the remainder f_3 gave better results.

[2] Note that some of these listed works also present other exploration schemes in addition to those for complete improper k-colourings, though the latter schemes are generally the most prominent in the papers.

2.2.4 Further Points

Recently, interesting work has also been carried out by Hertz, Plumettaz and Zufferey [45] who propose a method that operates in different solution spaces during different stages of a run. Specifically, TabuCol is used to explore the space of complete, improper k-colourings, and PartialCol is used for the space of partial, proper solutions. The main idea here is that a local optimum in one solution space is not necessarily a local optimum in another; hence, when the search is deemed to have stagnated in one space, a procedure is used to alter the incumbent solution so that it becomes a member of another space. The search can then be continued in this new space where further improvements might be made, with the process being repeated as long as necessary. The authors also propose a third solution space based on the idea of assigning orientations to edges in the graph and then trying to minimise the length of the longest paths within the resultant directed graph (see also [41]). The authors note, however, that improvements are rarely achieved during exploration of the latter space, but that its inclusion is still useful because it tends to make large alterations to a solution, helping to diversify the search.

Concluding this review, it is relevant to note that many of the schemes mentioned above are also commonly used in algorithms tackling problems related to graph colouring. For example, we observe the existence of timetabling algorithms that use constructive heuristics with backtracking [17]; algorithms that allow additional timeslots (colours) in a timetable and then only deal with feasible solutions [13, 24, 33, 58]; methods that fix the number of timeslots in advance and then allow constraints to be violated (i.e. clashes to occur) [15, 22, 28]; and also algorithms that deal with partial timetables, never allowing constraint violations to occur [14, 68]. Similar examples can also be noted in other related problems such as the frequency assignment problem [6, 78].

3 Algorithms for this Comparison

In this section we review in more detail the six algorithms considered in our comparison. To gain a holistic view, we purposely choose at least one high-performing method from each algorithmic scheme outlined above. Specifically, TabuCol, AntCol and the hybrid evolutionary algorithm (HEA) operate by searching spaces of complete, improper k-colourings; PartialCol searches spaces of partial, proper colourings; the hill climbing (HC) method explores spaces of feasible colourings; and the backtracking algorithm makes use of constructive techniques. To a certain extent, choice of algorithm has also been influenced by the ease at which we found we could reproduce results according to details given in the literature.

3.1 The TabuCol Algorithm [37, 44].

The first algorithm considered in our comparison is the TabuCol algorithm. Since its proposal in 1987 by Hertz and de Werra [44] TabuCol has been used as a local search subroutine in various high-performing hybrid algorithms [8, 30, 37, 39, 74], though experiments reported by Blochliger and Zufferey [9] also suggest that this method can be quite competitive in its own right. As mentioned earlier, TabuCol operates in the space of complete improper k-colourings, using cost function f_2. The specific version of the algorithm that we consider is the so-called "improved" variant, initially proposed Galinier and Hao [37]. The reader may find a detailed account of this particular method in [38], but we now summarise the main characteristics.

During a run, a move in the search space is performed using a neighbourhood operator that identifies a vertex v whose assignment to colour i is currently causing a clash, and then assigns it to a new colour $j \neq i$. The tabu list is stored in a $|V| \times k$ matrix T, and upon performing this move, the element $T(v, i)$ is marked as tabu for the next t iterations of the algorithm (we note that this has the effect of making *all* solutions containing the assignment of vertex v to colour i tabu, not just the current one). In each iteration the complete set of $(f_2 \times (k-1))$ moves is considered, and the move chosen is the one that invokes the largest decrease (or failing that, the smallest increase) in cost of any non-tabu move. Ties are broken randomly, and tabu moves are also permitted if they

are seen to improve on the best solution found so far.

In the version of TabuCol that we use, an initial candidate solution is constructed by taking a random ordering of the vertices and applying a modified first-fit algorithm (Section 2) in which only k colours are permitted. Thus if vertices are encountered that cannot be assigned to any of the k colours without inducing a clash, these are assigned to one of the existing colours randomly. Of course, we could use more sophisticated constructive methods here, but it is stated in both [38] and [9] that the method of initial solution generation is not critical in this algorithm's performance. With regards to the tabu tenure, Galinier and Hao [37] suggest making t a random variable that is proportional to its cost. Specifically, they suggest using $t = 0.6f_2 + r$, where r is an integer uniformly selected from the range 0 to 9 inclusive. These particular settings have been used in previous algorithms [9, 37, 74] and are generally thought to give good results, though it is noted in [38] that other schemes for determining t could be more appropriate for certain graphs.

3.2 The PartialCol Algorithm [9].

The PartialCol algorithm, due to Blochliger and Zufferey [9], operates in a similar fashion to TabuCol in that it seeks to achieve a proper colouring at a particular setting for k. However in contrast to the latter, PartialCol does not allow improper solutions to be considered: instead, vertices that cannot be assigned to any of the k colours without causing a clash are put into a set S. The aim of PartialCol is thus to make alterations to both the partial solution and S such that $f_3 = |S| = 0$ is achieved, leaving a feasible k-coloured solution.

To try and achieve its goals PartialCol makes use of tabu search, though due to its use of partial solutions, its neighbourhood operator is rather different to TabuCol. Specifically, a move in the search space is achieved by selecting a vertex $v \in S$ and assigning it to a colour i ($1 \leq i \leq k$). Next, any vertices u that are coloured with i and adjacent to v are uncoloured and moved to S. The corresponding elements $T(u, i)$ in the tabu list are then marked as tabu for the next t iterations. At each iteration of the algorithm, the complete neighbourhood of $(|S| \times k)$ moves is examined and the move to be performed is chosen using the same criteria as TabuCol.

The algorithm version used in our comparison is known as Foo-PartialCol, which Blochliger and Zufferey stated in 2008 "should be considered as one of the best local search colouring heuristics developed to date". Here, Foo abbreviates "Fluctuation Of the Objective-function", which indicates the use of a mechanism that alters the tabu tenure t based on the progress of the current state of the search. In essence, if during a run the objective function has not altered for a lengthy period of time, it is assumed that the search has stagnated in a particular region of the search space and so t is increased to try to encourage an escape from this region. Similarly, when the objective function is seen to be fluctuating, t is slowly reduced, counteracting these effects. Note that this scheme requires values to be assigned to various parameters, the meanings of which are described in [9]. In our case, we choose to use the settings supplied in the authors' original source code, available at [5]. Finally, an initial solution is generated in the same way as TabuCol, except in this case vertices for which there exists no clash-free colour are added to S.

3.3 Hybrid Evolutionary Algorithm (HEA) [37].

The hybrid evolutionary algorithm of Galinier and Hao [37] was originally proposed in 1999 and is currently understood to be one of the best performing algorithms for graph colouring [42, 61]. The HEA operates by maintaining a (typically small) steady-state population of candidate solutions which are evolved via a problem-specific recombination operator and a local search method. Like TabuCol, the HEA operates in the space of complete improper k-colourings using cost function f_2.

The algorithm begins by creating an initial population of candidate solutions via a modified version of the Dsatur algorithm. Specifically, each individual in the population is formed by taking each vertex in turn according to the Dsatur heuristic (Section 2.1) and then assigning it to the lowest indexed colour i ($1 \leq i \leq k$) where no clash occurs. Vertices for which no clash-free colour exists are kept to one side and are assigned to a random colour at the end of this process. Ties in the

		Parent p_1	Parent p_2	Child	Comments
a)	$c(1)=$	{1, 2, 3}	{3, 4, 5, 7}	{}	Select the colour with most vertices in p_1 and copy to the child.
	$c(2)=$	{4, 5, 6, 7}	{1, 6, 9}	{}	Remove the copied vertices from p_1 and p_2.
	$c(3)=$	{8, 9, 10}	{2, 8, 10}	{}	
b)	$c(1)=$	{1, 2, 3}	{3}	{4, 5, 6, 7}	Select the colour with most vertices in p_2 and copy to the child.
	$c(2)=$	{}	{1, 9}	{}	Remove the copied vertices from p_1 and p_2.
	$c(3)=$	{8, 9, 10}	{2, 8, 10}	{}	
c)	$c(1)=$	{1, 3}	{3}	{4, 5, 6, 7}	Select the colour with most vertices in p_1 and copy to the child.
	$c(2)=$	{}	{1, 9}	{2, 8, 10}	Remove the copied vertices from p_1 and p_2.
	$c(3)=$	{9}	{}	{}	
d)	$c(1)=$	{}	{}	{4, 5, 6, 7}	Having formed k colours, assign any vertices not in the child
	$c(2)=$	{}	{9}	{2, 8, 10}	to random colours.
	$c(3)=$	{9}	{}	{1, 3}	
e)	$c(1)=$	{}	{9}	{4, 5, 6, 7}	A complete (though not necessarily proper) solution has
	$c(2)=$	{9}	{}	{2, 8, 10, 9}	been formed.
	$c(3)=$	{}	{}	{1, 3}	

Figure 2: Example application of Galinier and Hao's GPX operator using $|V| = 10$ and $k = 3$.

DSATUR heuristic are also broken randomly, which is thought to provide sufficient randomisation to form a diverse initial population [37]. Upon construction of the population, an attempt is then made to improve each individual by applying the local search routine to each member.

For the remainder of the run, the algorithm evolves the population using recombination, local search, and replacement pressure. In each iteration two parent solutions p_1 and p_2 are selected at random, and copies of these are used in conjunction with the recombination operator to produce one child solution c. This child is then improved via the local search operator, and is inserted into the population by replacing the weaker of its two parents. Note that there is no bias towards selecting fitter parents in each iteration; rather evolutionary pressure only exists due to the offspring replacing their weaker parent (regardless of whether the parent has a better cost than its child).

The recombination operator proposed by Galinier and Hao [37] is the so-called Greedy Partition Crossover (GPX). An illustration of this operator using an example taken from [37] is provided in fig. 2. As is the norm for evolutionary computation, the idea is to produce a child solution by attempting to combine useful substructures (building blocks) from both parents. In this case, the building blocks are thought to be the colours themselves; however, as is illustrated in fig. 2 it is not always possible to pass complete colour classes from parent to offspring since the vertices assigned to a particular colour in one parent may be spread across many colours in the other. This implies that some reorganisation and/or repair may need to take place, possibly introducing substructures in an offspring that do not occur in either parent. Though this may not be ideal (since it could result in rather large amounts of uninherited information in an offspring) such a feature seems to be inevitable when using colour classes as building blocks in recombination and it is certainly a phenomenon noted elsewhere [33, 34, 35, 58]. Note, however, that by alternating between the parents and by favouring large colour classes, the operator seems to be making efforts to maximise the amount of information inherited from the parents, though in the worst case, only one colour class in an offspring will exactly match a colour class of its parents.

Considering the local search element of the HEA, Galinier and Hao use TABUCOL for a fixed number of iterations I whenever a new candidate solution is generated.[3] In [37] Galinier and Hao present results for a subset of the DIMACS instances, manually tuning I and k in each case. As noted earlier however, we choose not to follow this particular strategy and require a setting for I to be determined automatically by the algorithm. We also need to be wary that if I is set too low, then insufficient local search will be carried out on each newly created solution, while an I that is too high will result in too much effort being placed on local search as opposed to the global search carried out by the evolutionary operators. Ultimately we settled on $I = 16|V|$, which corresponds roughly to the settings used on the lowest values for k considered for each instance in [37].[4] In all

[3]The tabu-tenure scheme used is the same as that described in Section 3.1.

[4]In the more recent work of Glass and Prugel-Bennett [42], it is shown that Galinier and Hao's algorithm produces

```
     AntCol (G = (V, E))
(1)  t_{v,u} ← 1 ∀v, u ∈ V, v ≠ u
(2)  k = |V|
(3)  while (not stopping condition) do
(4)      δ_{v,u} ← 0 ∀v, u ∈ V, v ≠ u
(5)      best← k
(6)      madeFeasible←false
(7)      for (ant ← 1 to nants) do
(8)          U ←BuildSolution(k)
(9)          if (U is a partial solution) then
(10)             Assign uncoloured vertices to existing colours in U and run TabuCol
(11)         if (U is feasible) then
(12)             madeFeasible←true
(13)             if (|U| ≤ best) then
(14)                 best← |U|
(15)         δ_{v,u} ← δ_{v,u} + F(U) ∀v, u : c(v) = c(u), v ≠ u
(16)     t_{v,u} ← ρt_{v,u} + δ_{v,u} ∀v, u ∈ V, v ≠ u
(17)     if (madeFeasible=true) then
(18)         k ←best−1
```

Figure 3: The AntCol algorithm. At termination, the best feasible solution found uses $k + 1$ colours.

experiments, we also used a population size of 10, as recommended by the authors.

3.4 The AntCol Algorithm [74].

The AntCol algorithm of Thompson and Dowsland [74] is another metaheuristic-based method combining global and local search operators. The idea in this approach is to use "ants" to produce individual candidate solutions to the problem. During a run each ant produces their solution in a non-deterministic manner, using probabilities based on heuristics and also on the quality of solutions produced by previous ants. In particular, if previous ants have identified features that seem to lead to better-than-average solution quality, the current ant is more likely to include these features in their own solution, generally leading to a reduction in the number of colours during the course of a run. Further information on ant colony optimisation can be found in [29].

The AntCol algorithm proposed by Thompson and Dowsland is based on earlier work of Costa and Hertz [23]. However, the former also introduce a number of additional operators, some of which are shown to considerably improve performance. In our case, we focus on the best-performing version they report, a description of which is provided in fig. 3. As shown in the pseudocode, in each cycle of the algorithm (lines (3)-(18)), a number of ants each produce a complete, though not necessarily feasible, solution $\mathcal{U}$. In line (15) the details of each of these solutions are then added to a trail update matrix δ, and at the end of a cycle the contents of δ are used to update the global trail matrix t before moving on to the next cycle.

To start, each individual ant attempts to construct a solution using the procedure BuildSolution, which is based on the RLF method (fig. 1), albeit with a few modifications:

- In the procedure a maximum of k colours are permitted, and once these have been constructed any remaining vertices are left uncoloured;
- The first vertex to be assigned to each colour U_i $(1 \leq i \leq k)$ is chosen randomly from the set of currently uncoloured vertices X;
- In remaining cases, each vertex v is assigned to colour U_i with probability:

$$P_{v,i} = \begin{cases} \frac{{\tau_{v,i}}^{\alpha}.{\eta_{v,i}}^{\beta}}{\sum_{u \in Y} {\tau_{u,i}}^{\alpha}.{\eta_{u,i}}^{\beta}} & \text{if } v \in Y \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

similar results when the tabu search element of the algorithm is replaced by a simple steepest decent method. However, it is also found that this alteration results in longer run times and so we do not consider this variant here.

where $Y \subseteq X$ denotes the set of currently uncoloured vertices that can be feasibly assigned to colour U_i. Note that the value $\tau_{v,i}$, calculated:

$$\tau_{v,i} = \frac{\sum_{u \in U_i} t_{v,u}}{|U_i|}, \tag{4}$$

makes use of the global trail matrix t. Thus, higher values of τ are associated with combinations of vertices that have been assigned the same colour in previous solutions. The value $\eta_{v,i}$ meanwhile, is the value associated with a heuristic rule, which in this case is the degree of vertex v in the graph induced by the set of currently uncoloured vertices X. Higher values for $\tau_{v,i}$ and $\eta_{v,i}$ thus result in higher values for $P_{v,i}$, encouraging vertex v to be assigned to colour U_i. The parameters α and β are used to control the balance between τ and η.

In their algorithm Thompson and Dowsland also make use of a "muti-sets" operator in the BUILDSOLUTION procedure, based on heuristics presented by Laguna and Marti [54]. Since the process of constructing a colour class is stochastic, the operator makes m separate attempts to construct each colour class. It then selects the one that results in the minimum number of edges in the graph induced by the remaining uncoloured vertices (since such graphs will tend to feature lower chromatic numbers).

On completion of BUILDSOLUTION, the generated solution $\mathcal{U}$ will be proper, but could be partial. If the latter is true, all uncoloured vertices are assigned to colours to form a complete, improper solution, and TABUCOL is run for I iterations. Details on the solution are then written to the trail update matrix δ using the evaluation function:

$$F(\mathcal{U}) = \begin{cases} 1/f_2 & \text{if } f_2 > 0 \\ 3 & \text{otherwise.} \end{cases} \tag{5}$$

This means that higher-quality solutions contribute larger values to δ, encouraging their features to be included in solutions produced by future ants.

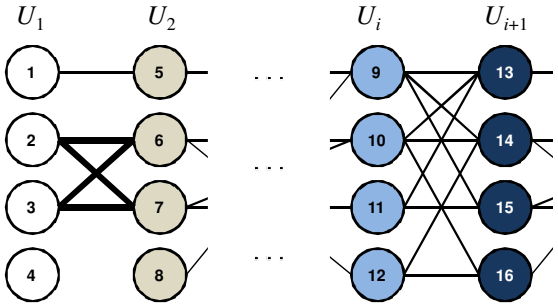
The parameters used in our application, and recommended by Thompson and Dowsland are as follows: $\alpha = 2$, $\beta = 3$, $\rho = 0.75$, *nants*$= 10$, $I = 2|V|$, and $m = 5$. The tabu tenure scheme of TABUCOL is the same as previous descriptions.

3.5 Hill-Climbing (HC) algorithm [57]

The next algorithm considered in our comparison is adapted from a general-purpose method for grouping problems, outlined by Lewis in [57]. This method operates in the space of feasible solutions with the initial solution being formed using the DSATUR heuristic. During a run the algorithm operates on a single solution $\mathcal{U} = \{U_1, \ldots U_{|\mathcal{U}|}\}$ with the aim of minimising $|\mathcal{U}|$. To start, a small number of independent sets are removed from $\mathcal{U}$ and are placed into a second set $\mathcal{W}$. Then, a local-search procedure is run for I iterations, which attempts to feasibly transfer vertices from independent sets in $\mathcal{W}$ into existing independent sets in $\mathcal{U}$ such that $\mathcal{U}$ remains proper. If successful, this has the effect of increasing the cardinality of one or more independent sets in $\mathcal{U}$ and may also eliminate some independent sets from $\mathcal{W}$, reducing the number of colours being used.

A single iteration of the local search procedure operates as follows. First, each vertex v in $\mathcal{W}$ is considered in turn to see if it can be transferred to any of the independent sets in $\mathcal{U}$. If so, such transfers are performed. Next, an alteration is made to a randomly selected pair of independent sets $U_i, U_j, \in \mathcal{U}$ by interchanging some of their vertices. The purpose of making this alteration is that it changes the make-up of two independent sets and raises the possibility that other vertices in $\mathcal{W}$ can be transferred into them. This repeats for I iterations.

In the version of the algorithm used in our experiments, alterations to independent sets U_i and U_j are usually made by identifying a Kempe chain between a randomly selected vertex $v \in U_i$ and set U_j, and then interchanging the colours of all the nodes within this chain (see fig. 4(a)). Note that in some cases a Kempe chain will contain all vertices in both independent sets (fig. 4(b)). These interchanges serve no purpose here and are ignored. Also note that certain moves are not included



(a) Kemp chain (in bold) formed from e.g. vertex 2 and independent set U_2.

(b) Example of two independent sets where all Kemp chain moves result in a full colour interchange.

Figure 4: The Kempe chain operator and related issues.

in the Kempe chain neighbourhood: for example, they do not allow the swapping of a pair of non-adjacent vertices, such as vertices 4 and 8 in fig. 4. Consequently, during a run alterations can also be made to independent sets $U_i, U_j \in \mathcal{U}$ by identifying two non-adjacent vertices $v \in U_i, u \in U_j$ and swapping their colours if such a swap is not seen to induce a clash. Such a move is performed by going through each pair of non-adjacent vertices in a random order, and stopping when an appropriate swap has been performed.

On completion of the local search procedure, the independent sets in $\mathcal{W}$ are copied back into $\mathcal{U}$, forming a feasible solution. The independent sets in $\mathcal{U}$ are then ordered according to some (possibly random) heuristic, and a new solution $\mathcal{U}'$ is formed by constructing a permutation of the vertices in the same manner as Culberson and Luo's Iterated Greedy algorithm ([26], Section 2.2.1) and applying the first-fit procedure. This operation is intended as a macro-move operator that generates large alterations to the incumbent solution, which is then passed back to the local search procedure for further optimisation. Note that none of the stages of this algorithm allow the number of independent sets $|\mathcal{U}|$ to increase, thus providing its hill-climbing characteristics.

As with other approaches, a number of parameters have to be set with this algorithm, each that can influence its performance. The values used in our experiments were determined in preliminary tests and during experimental work in a previous paper [57]. For the local search procedure, independent sets are moved into $\mathcal{W}$ by considering each $U_i \in \mathcal{U}$ in turn and transferring it with probability $1/|\mathcal{U}|$. The local search procedure is then run for $I = 1000$ iterations, and in each iteration the Kempe-chain and swap neighbourhoods are called with probabilities 0.99 and 0.01 respectively. Finally, when constructing the permutation of the vertices for passing to the first-fit algorithm, the independent sets are ordered using the following heuristics (a) *largest first* (where independent sets are arranged in order of decreasing size), (b) *reverse ordering* (where sets are arranged in the the reverse order of their labelling the previous solution), and (c) *random ordering*. These heuristics are selected randomly according to the ratio 5:5:3 (respectively), a setting suggested by Culberson and Luo [26].

3.6 Backtracking Dsatur algorithm [10, 52].

Finally, a tree-search algorithm is also considered in our comparison. We use a backtracking version of the Dsatur heuristic which also includes an operator for dynamically re-ordering the vertices when a node in the search tree is revisited [52]. This algorithm was implemented by Culberson and is available for download at [4]. A number of parameters need to be set when running this algorithm, each that can alter the quality of solutions and run-times, sometimes drastically. These include specifying the maximum number of branches that can be considered at each node, and the option of prohibiting branching at certain levels of the tree. In practice, it is not obvious how these settings might be chosen *a priori* for individual graphs, so in our case we opt for the most natural

configuration which is to simply attempt a complete exploration of the search tree. This means that the algorithm is exact under excess time, though of course such run-lengths are not achievable in most cases.

4 Experimental Analysis

In this section we detail the various graph types considered in our experiments and analyse the results achieved by the six algorithms. First we consider performance with two well-known types of artificially generated graph, random graphs and flat graphs, produced using the publicly available software of Culberson [4]. Section 4.2 then compares the algorithms on graphs arising in three practical situations, namely university timetabling (4.2.1), social networking (4.2.2), and sports scheduling (4.2.3).

In terms of comparing the algorithms, many previous graph colouring studies have chosen to measure computational effort using CPU time, often imposing time limits specified for their own machines [8, 9, 45, 54, 61, 74]. However, such an approach seems problematic because it depends on various local factors relating to the available hardware, the coding of the algorithm, and the compiler options used. In our case, we use the number of *constraint checks* as the atomic measure of computational effort [25, 48, 76]. A "check" occurs when an algorithm requests some information about a problem instance, including checking whether two vertices are adjacent (by accessing an adjacency list or adjacency matrix), and referencing the degree of a vertex. In addition, the algorithms employing tabu search also maintain additional data structures that allow changes in cost for each considered move to be evaluated in constant time. References to these structures are also counted as checks, and further information can be found in [9, 38].

Due to the operational differences of the algorithms, during a run solution quality is measured by simply observing the smallest number of colours used in a feasible solution up until that point. Note that because TabuCol, PartialCol and the HEA operate using infeasible solutions, settings for k are also required which might then be modified during a run. In our case initial values were determined by executing Dsatur on each instance and setting k to the number of colours used in the resultant solution. During runs, k was then decremented by 1 each time a feasible k-colouring was found, with the algorithms being restarted.

In all trials a cut-off point of 5×10^{11} checks was imposed. This corresponds to roughly double the longest run that was performed by Galinier and Hao in [37] and is set deliberately high to provide some notion of excess time in the trials.[5]

Finally for space reasons, the data presented in most of our figures focusses on mean algorithm performance – that is, the quality of the best solution produced in a run averaged across a particular set of runs. Unless stated otherwise, however, best and worst performance from these sets can be considered to be consistent with these patterns. For completeness, a complete listing of all figures including best and worse performance is available online [3].

4.1 Comparison on Artificially Generated Graphs

Random graphs are a class of artificially generated graph where each of pair of vertices are made adjacent with probability p. Such graphs are nearly always used in analyses of graph colouring algorithms, though typically using only a limited number of instances (e.g. the DIMACS set contains just one instance for $|V| \in \{125, 250, 500, 1000\}$ and $p \in \{0.1, 0.5, 0.9\}$). Here we use a much larger set, using values of p ranging from 0.05 (sparse) to 0.95 (dense), incrementing in steps of 0.05, with $|V| \in \{250, 500, 1000\}$. Twenty five instances were generated in each case.

The second type of artificial graph we consider are flat graphs. These are constructed by partitioning the vertices into K almost equi-sized sets: that is, each set contains either $\lfloor |V|/K \rfloor$ or $\lceil |V|/K \rceil$ vertices. Edges are then added between pairs of vertices in different sets with probability

[5] All algorithms were implemented by the authors in C++ (with the exception of the Backtracking algorithm [4], programmed in C) and were compiled using the `Gnu` compiler under the `-O3` optimisation option. In various cases, modified versions of the publicly available tabu search-based code of Blochliger and Zufferey [5] were also used.

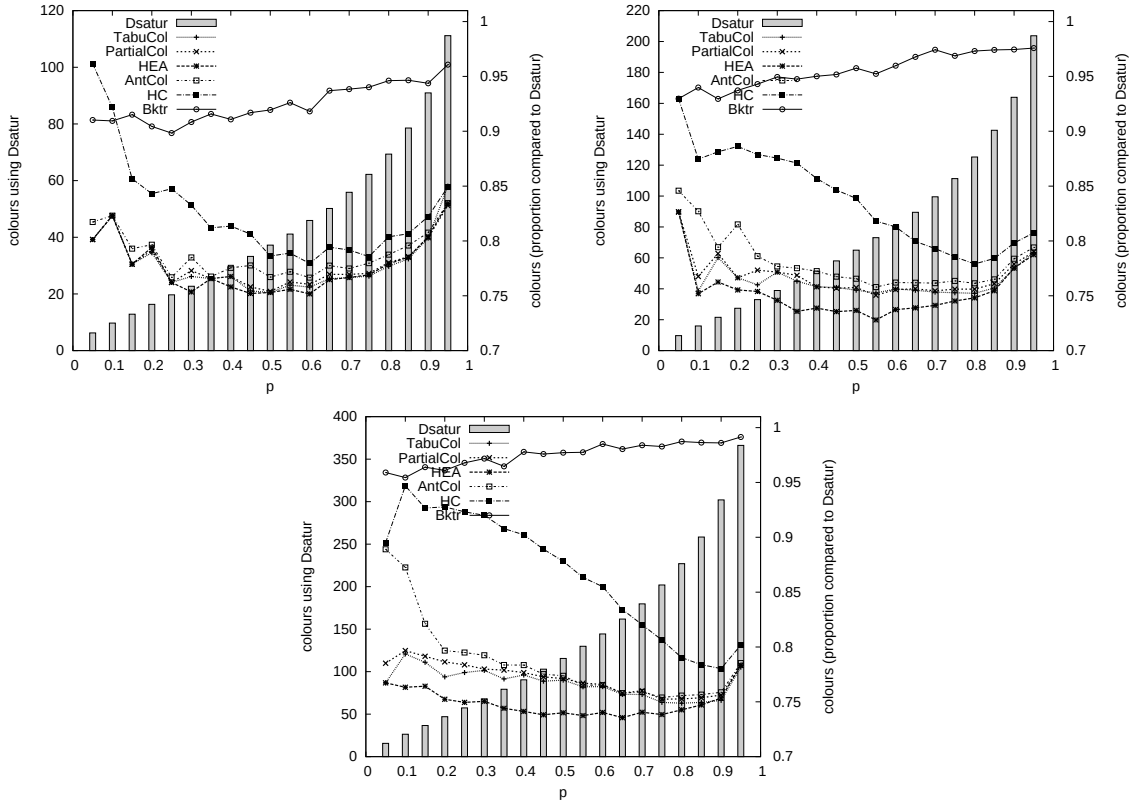


Figure 5: Mean quality of solution achieved on random graphs of $|V| = 250$, 500, and 1000 (respectively) for various edge probabilities p. All points are the mean of 25 runs on 25 different instances.

p in such a way that the variance in vertex degrees is kept to a minimum. It is well-known that K-coloured solutions to flat graphs are quite easy to achieve for most values of p. This is because for lower p's, problems will be under-constrained, perhaps giving $\chi < K$, making K-coloured solutions easily identifiable. On the other hand, high p's result in over-constrained problems with prominent global optima (with strong basins of attraction) also allowing easy discovery. Hard-to-solve K-colourable graphs occur for a region of p's at the boundary of these extremes, commonly termed the *phase transition region* [20, 32, 57, 77]. Flat graphs, in particular, are known to have rather pronounced phase transition regions because each colour class and vertex degree is deliberately similar, implying a lack of heuristic information for algorithm exploitation [26, 57]. For our experiments, flat graphs were generated for $K = \{10, 50, 100\}$ using various settings for p in and around the phase transition regions. In each case we used $|V| = 500$, implying 50, 10, and 5 vertices per colour respectively. Twenty instances were generated in each case.

Note that according to the structure of random graphs, vertex degrees are characterised by the binomial distribution $D \sim \mathrm{B}(|V| - 1, p)$. This means that the standard deviation of the vertex degrees $\sigma = \sqrt{(|V| - 1)p(1 - p)}$ does not exceed 15.8 in our set of random graphs. This also implies that the degree coefficient of variation (CV), which is defined as the ratio of the standard deviation to the mean $\frac{\sigma}{\mu}$, never exceeds 28% (being maximised at $|V| = 250$, $p = 0.05$). In a similar fashion, flat graphs are constructed such that variance in degrees is minimised, and for our generated instances this means that the CV never exceeds 28.5%. Compared to many of the more real-world graphs considered later, these values imply rather a high level of vertex homogeneity (i.e. vertices tend to "look the same"), helping to explain some of our results.

Figure 5 summarises the mean solution quality achieved by the six algorithms on the random graphs. In each figure, the bars show the number of colours used in solutions produced by DSATUR and the lines then give the proportion of this number used in the solutions of the six algorithms.

Note that all algorithms achieve a reduction in the number of colours realised by DSATUR, though in all but the smallest, sparsest graphs, the backtracking algorithm exhibits the smallest margins of improvement, apparently due to the high levels of vertex homogeneity in these instances, which makes it difficult for favourable regions of the search tree to be identified by the algorithm. It is clear that TABUCOL, PARTIALCOL, and the HEA in particular, produce the best results for such graphs. For $|V| = 250$ these algorithms produce mean results that, across the range of p's, show no significant difference to one another,[6] perhaps indicating that the achieved solutions are consistently close to the optima. For larger graphs however, the HEA's solutions are seen to be significantly better, though its rates of improvement are slightly slower than TABUCOL, and PARTIALCOL, as shown in fig. 6. Similar behaviour during runs was also witnessed with the smaller random instances (not shown here).

The patterns in fig. 5 indicate that the HEA's strategy of exploring the space of infeasible solutions via use of both global and local search operators is the most beneficial of those considered here. Indeed, although the HC algorithm also uses both global and local search operators, here its insistence on preserving feasibility implies a lower level of connectivity in its underlying search space, making navigation more restricted and resulting in noticeably inferior solutions.

Figure 5 also reveals that ANTCOL does not perform well with large sparse instances, though it does become more competitive with denser instances. The reasons for this are twofold. First, the degrees of vertices in sparse graphs are naturally lower, reducing the heuristic bias provided by η (eq. (3)) and perhaps implying an over-dominant role of pheromone τ during applications of BUILDSOLUTION. Second, sparse graphs also feature greater numbers of vertices per colour – thus, even if very promising independent sets are identified by ANTCOL, their reconstruction by later ants will naturally depend on a longer sequence of random trials, making them less likely. To back these assertions, we also repeated the trials of ANTCOL using the same local search iteration limit as the HEA, $I = 16|V|$. However, though this brought slight improvements for denser graphs, the results were still observed to be significantly worse than the HEA, suggesting the difference in performance indeed lies with the global-search element of ANTCOL in these cases.

Similar patterns are also revealed in the six algorithms' performance with flat graphs, shown in fig. 7. Again, we see that HEA, TABUCOL, and PARTIALCOL exhibit the best performance on instances within the phase transition regions, with the HC and backtracking algorithms proving the least favourable. One pattern to note is that for all values of K, the HEA tends to produce the best-quality results on the left side of the phase transition region, but PARTIALCOL produces better results for a small range of p's on the right-side. However, this difference is not due to the FOO tabu tenure mechanism of PARTIALCOL, because statistically equivalent results were achieved when we repeated our experiments using PARTIALCOL under TABUCOL's tabu tenure scheme. Thus, it seems that PARTIALCOL's strategy of only allowing solutions to be built from independent sets is favourable in these cases, presumably because this restriction facilitates the formation of independent sets of size $|V|/K$ – structures that will be less abundant in denser graphs, but which also serve as the underlying building blocks in these cases.

Another striking feature in fig. 7 is the poor performance of ANTCOL on the right-side of the phase transition regions. This again seems due to the diminished effect of heuristic value η, which in this case is because variance in vertex degrees is (deliberately) low, making it difficult to distinguish between vertices. Further, in denser graphs fewer combinations of $|V|/K$ vertices form independent sets, decreasing the chances of an ant constructing one. This reasoning is also backed by the fact that ANTCOL's poor performance lessens with larger values of K where, due to there being fewer vertices per colour, the reproduction of independent sets is dependent on shorter sequences of random trials.

[6] According to a Wilcoxon signed-rank test at the 1% significance level

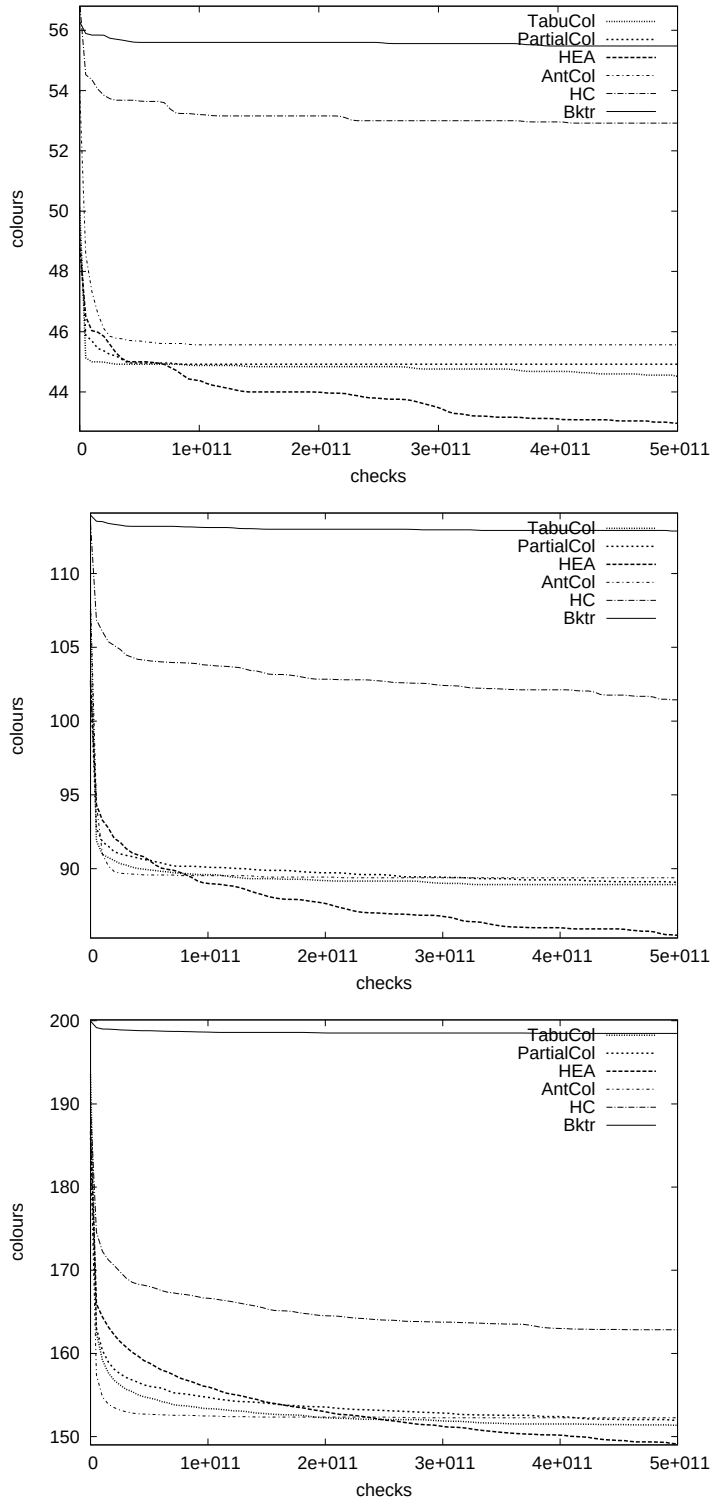


Figure 6: Run profiles on random graphs of $|V| = 1000$ with edge probabilities $p = 0.25$, 0.5, and 0.75 respectively. Each line represents a mean of 25 runs on 25 different instances.

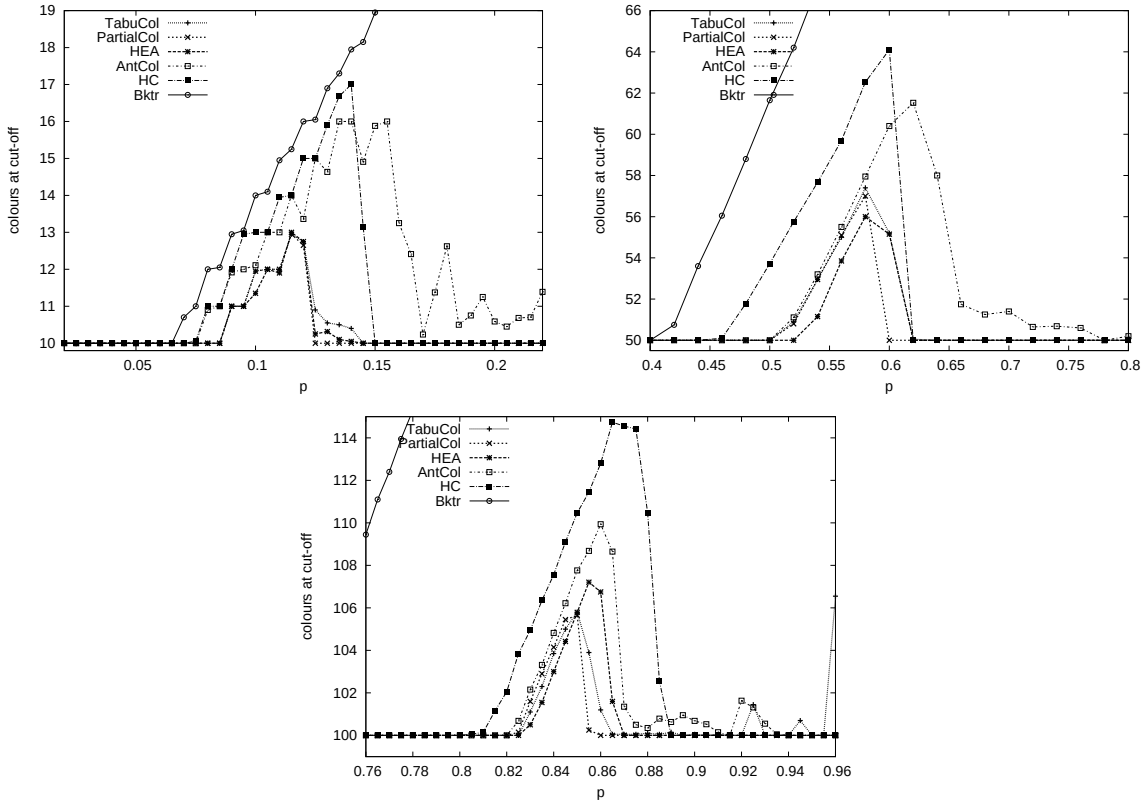


Figure 7: Mean quality of solution achieved with flat graphs of $|V| = 500$ with $K = 10$, 50, and 100 (respectively) for various edge probabilities p. All points are the mean of 20 runs on 20 different instances.

4.2 Comparison on Graphs From Real-World Problems

4.2.1 Exam Timetabling Problems

Our first set of "real world" problems concerns graphs representing university timetabling problems. Such problems involve assigning a set of "events" (exams, lectures etc.) to a fixed number of "timeslots" such that a collection of constraints are obeyed. One ubiquitous constraint in timetabling concerns "conflicting" events. A pair of events conflict when they require the same single resource: e.g. there may be a student or lecturer who needs to attend both events, or the events may require use of the same room. As a result conflicting events need to be scheduled to different timeslots. Under this constraint, timetabling problems can be modelled through graph colouring by considering each event as a vertex, with edges occurring between pairs of events that conflict. Each colour then represents a timeslot, and a feasible colouring corresponds to a complete timetable with no conflict violations [12, 14, 17, 58].

In practice, universities usually have a pre-defined number of timeslots in their timetable, and the task is to determine a feasible solution with fewer or equal colours. In many cases, however, it might be difficult to ascertain whether a target is achievable for a given problem, or it may be desirable to use as few timeslots as possible, particularly if it provides extra time for marking, or gives a shorter teaching day. In our case we concern ourselves with the latter problem, and use a well-known set of real-world timetabling problems compiled by Carter et al. [1, 17]. This contains 13 exam timetabling problems encountered during the 1980s and 90s in various international universities and features problems ranging in size from $|V| = 81$ to 2419.

A summary of instance details and algorithm performance is given in Table 1. Note that in contrast to the artificial instances from the previous section, the worst overall performance now occurs with those methods relying solely on local search: that is, TabuCol and to a lesser extent

			Vertex degree			Colours at Cut-off: Mean and best (in parenthesis)					
Name	$\|V\|$	Density	Min;Med;Max	Mean	CV	TabuCol	PartialCol	HEA	AntCol	HC	Bktr
hec-s-92	81	0.415	9; 33; 62	33.7	36.3%	17.22 (17)	17.00 (17)	17.00 (17)	17.04 (17)	17.00 (17)	19.00 (19)
sta-f-83	139	0.143	7; 16; 61	19.9	67.4%	13.35 (13)	13.00 (13)	13.00 (13)	13.13 (13)	13.00 (13)	13.00 (13)
yor-f-83	181	0.287	7; 51; 117	52	35.2%	19.74 (19)	19.00 (19)	19.06 (19)	19.87 (19)	19.00 (19)	20.00 (20)
ute-s-92	184	0.084	2; 13; 58	15.5	69.1%	10.00 (10)	10.00 (10)	10.00 (10)	11.09 (10)	10.00 (10)	10.00 (10)
ear-f-83	190	0.266	4; 45; 134	50.5	56.1%	26.21 (24)	22.46 (22)	22.02 (22)	22.48 (22)	22.00 (22)	22.00 (22)
tre-s-92	261	0.18	0; 45; 145	47	59.6%	20.58 (20)	20.00 (20)	20.00 (20)	20.04 (20)	20.00 (20)	23.00 (23)
lse-f-91	381	0.062	0; 16; 134	23.8	93.2%	19.42 (18)	17.02 (17)	17.00 (17)	17.00 (17)	17.00 (17)	17.00 (17)
kfu-s-93	461	0.055	0; 18; 247	25.6	120.0%	20.76 (19)	19.00 (19)	19.00 (19)	19.00 (19)	19.00 (19)	19.00 (19)
rye-s-93	486	0.075	0; 24; 274	36.5	111.8%	22.40 (21)	21.06 (21)	21.04 (21)	21.55 (21)	**21.00** (21)	22.00 (22)
car-f-92	543	0.138	0; 64; 381	74.8	75.3%	39.92 (36)	32.48 (31)	28.50 (28)	30.04 (29)	27.96 (27)	**27.00** (27)
uta-s-92	622	0.125	1; 65; 303	78	73.7%	41.65 (39)	35.66 (34)	30.80 (30)	32.89 (32)	30.27 (30)	**29.00 (29)**
car-s-91	682	0.128	0; 77; 472	87.4	70.9%	39.10 (32)	30.20 (29)	29.04 (28)	29.23 (29)	29.10 (28)	**28.00** (28)
pur-s-93	2419	0.029	0; 47; 857	71.3	129.5%	50.70 (47)	45.48 (42)	33.70 (33)	33.47 (33)	33.87 (33)	**33.00** (33)
					Total	341.05 (315)	302.36 (294)	280.16 (277)	286.84 (281)	**279.20 (276)**	282.00 (282)
					Rank	(6)	(5)	(2)	(4)	(1)	(3)

Table 1: Details of the 13 timetabling instances of Carter et al. [17] and summary of algorithm performance. All statistics collected from 50 runs on each instance.

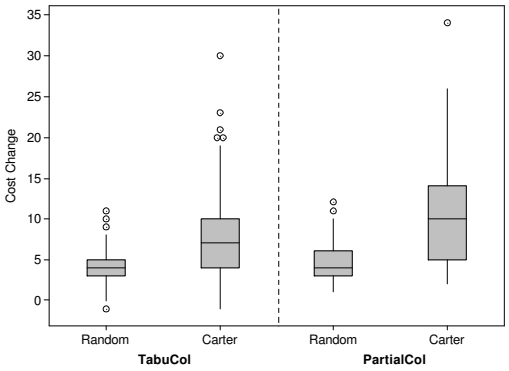


Figure 8: Cost-change distributions for a random graph ($|V| = 500$, $p = 0.15$, CV= 10.7%, using $k = 16$) and timetable graph car-f-92 ($|V| = 543$, $p = 0.138$, CV= 75.3%, using $k = 27$). In all cases samples were taken from candidate solutions with costs of 8.

PartialCol. Indeed, in our tests we found that these methods were often incapable of achieving feasible solutions even using the initial setting for k determined by Dsatur, compelling us to use a higher values in order to gain any comparative results.[7] The cause of this poor performance seems to lie in the fact that, as shown in Table 1, the degree CV of these 13 problems are considerably higher than any of the artificially generated instances seen earlier. The effects of this are shown in fig. 8 where, compared to a random graph of a similar size and density, the differences in cost between neighbouring solutions varies more widely. This suggests a more "spiky" cost landscape in which the use of local search mechanisms in isolation is insufficient, exhibiting a susceptibility for becoming trapped at local optima.

On the other hand, the most consistent performance with these instances is achieved by the HC and HEA algorithms,[8] demonstrating that the issues of using local search can be alleviated via hybridisation with global search operators. On individual instances, the relative performances of HC and HEA do seem to vary, however. With `car-f-92` and `car-s-91`, for example, the HEA's best observed solutions are determined within ≈1% of the run limit, while HC's progress is much slower, taking a few minutes to achieve similar results. On the other hand, with instances such as `rye-s-93`, HC consistently produces the best observed results in less that 0.3% of the cut-off, suggesting that its operators are somehow suited to this instance. This issue is considered further

[7]The reported results for TabuCol and PartialCol in Table 1 were thus produced using an initial k generated by executing the first-fit algorithm with a random permutation of the vertices.

[8]No significant difference between these two methods across the instances was observed

in Section 5.

In contrast to the artificially generated graphs seen earlier, we also observe that the backtracking algorithm is competitive with these instances, featuring the third lowest mean-total in the table, and even producing the best average performance with the four largest problems. It seems that the large degree CVs of these problems characterises an abundance of heuristic information that is often successfully exploited by the algorithm. Indeed for the four largest instances, the solutions reported in Table 1 were actually found in less than 2% of the computation limit, implying that the algorithm quickly identified the correct regions of the search-tree. However, counterexamples in which the backtracking algorithm consistently produces the worst performance can also be seen in the table.

Finally, we also note the sporadic performance of AntCol with these instances. For all but the four largest problems, AntCol's best solutions equal those of the other algorithms; however its averages are less favourable, particularly compared to the HEA and HC algorithms. Consider, for example, the results of `ute-s-92` in the table. This problem is consistently solved using 10 colours by all methods except AntCol, which often requires 11 or 12 colours. In fact, we find that for instances such as these, AntCol's performance depends very much on the quality of solutions produced in the first cycle of the algorithm. Due to the low vertex degrees (and reduced influence of η that results), eq. (3) is predominantly influenced by the pheromone trails τ; however, if an 11 or 12-colour solution is produced during the first cycle, features of these sub-optimal solutions are still used to update the pheromone matrix t, making their re-occurrence in later cycles more likely. The upshot is that AntCol is rarely seen to improve upon solutions found in the initial cycle of the algorithm with these instances.

To demonstrate the described features, run profiles for three contrasting timetabling instances are shown in fig. 9.

4.2.2 Social Networks

Our next set of experiments involves graphs representing social networks. Social networks consist of "nodes" (usually individual people) that are "tied" by some sort of inter-dependency such as friendship or belief. Here we consider the social networks of school-friends, compiled as part of the USA-based National Longitudinal Study of Adolescent Health project [64]. The colouring of such networks might be required when we wish to partition students into groups such that individuals are kept separate from their friends, e.g. for group work and team-building exercises (see also [50]). To construct these networks, surveys were conducted in various schools, with each student being asked to list all of their friends. In some cases, students were only allowed to nominate friends attending the same school, while in others they could include friends attending a "sister school" (e.g. middle-school students could include friends in the local high-school), giving either single-cluster or double-cluster networks respectively. In the resultant graphs, each student is represented by a vertex, with edges signifying a claimed friendship between the associated individuals (see fig. 10).[9]

We took a random sample of 10 single-cluster networks and 10 double-cluster networks from the publicly available data set, with graph sizes ranging from 380 to 2250 and 291 to 1246 vertices respectively. Each algorithm was then run 50 times on each instance. The outcomes of these experiments were quite straightforward, with the HEA, HC, TabuCol, and PartialCol methods quickly producing the best observed solutions for each instance in all runs. We also saw that the number of colours used in each case ranged from 5 to 10, but no correlations were observed to suggest that instance size or the presence of clusters affected this.

One feature of these graphs is that they are quite sparse, with all instances featuring mean degrees of less than 8 and edge densities of less than 0.03. This might make their solving relatively "easy" with the above four algorithms; however for AntCol these features also often invoked the negative performance features noted in the previous section, with a high-quality solution either

[9]Note that in the original data, edges signifying friendships are both directed and weighted; however, in our case directions and weights have been removed to form a simple graph.

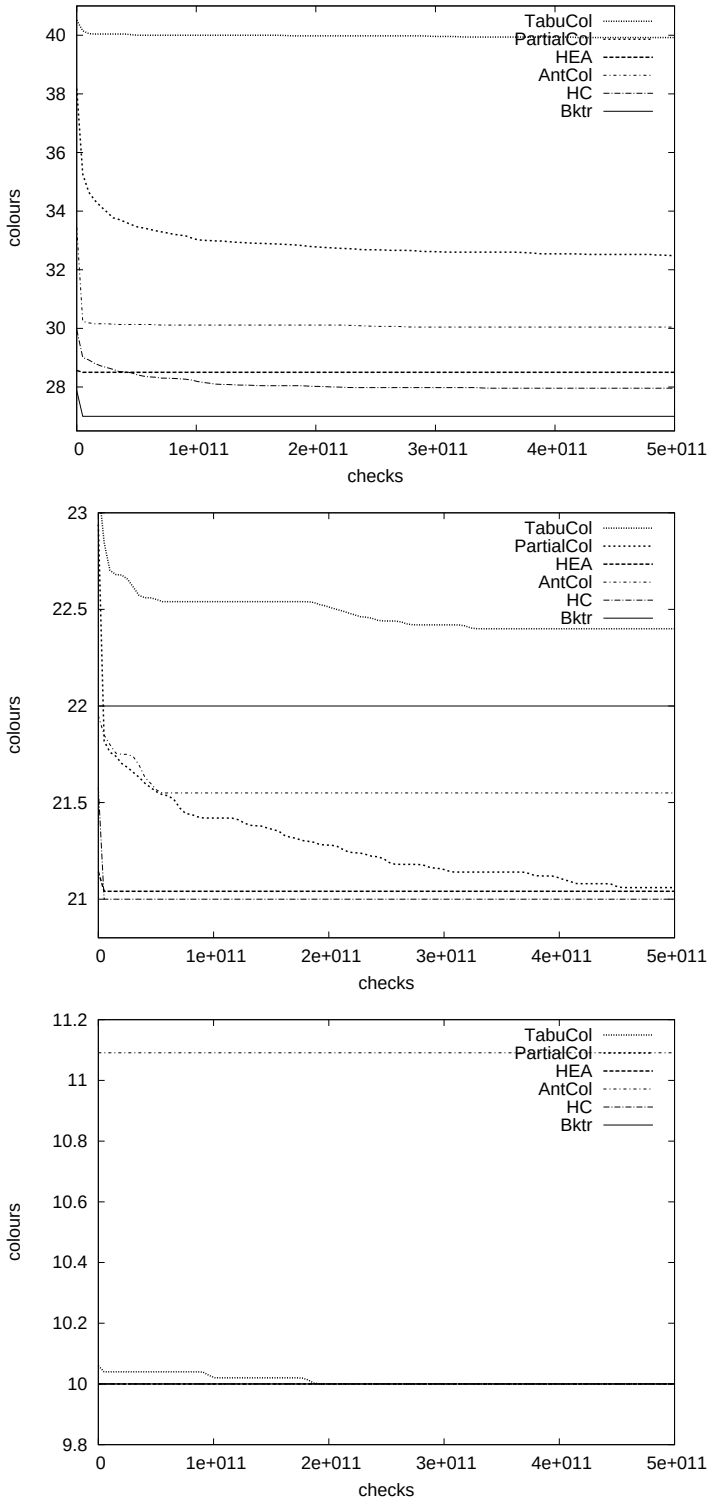


Figure 9: Run profiles for car-f-92, rye-s-93, and ute-s-92 respectively.

being produced very quickly (in the first cycle), or not at all. Indeed, across all trials, AntCol was seen to require an average of 0.3 additional colours across the instances, which was also worse than the backtracking algorithm, which required 0.05 extra colours.

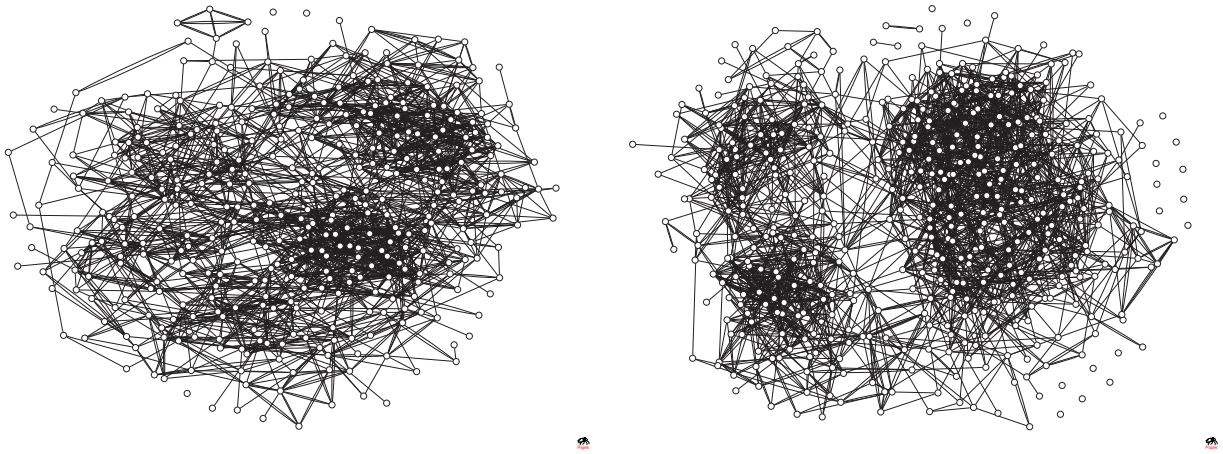


Figure 10: Illustration of a single-cluster (left) and double-cluster (right) social network from the Add Health data set.

4.2.3 Comparison using Round-Robin Scheduling Problems

Our final comparison concerns graphs representing *round-robin* tournaments used in sports leagues [7, 31, 59, 75, 80]. In such problems, we are given an even number of "teams" n, and each team is required to participate in a match against all other teams m times in $m(n-1)$ *rounds* [27]. The problem of constructing a round-robin tournament can be transformed into graph colouring by considering each match as a vertex, with edges being imposed between any pair of matches that cannot be assigned to the same round (e.g. matches featuring a common team). Colours then represent the rounds of the schedule, and the task is to colour the graph using $K = r(n-1)$ colours. Here we specifically consider cases where $m = 2$, commonly termed "double round-robins". The number of vertices $|V|$ in such problems is $n(n-1)$ and, as illustrated in fig. 11(a), each has an equal degree of $4(n-2)+1$. The chromatic number of such problems is also known to be $\chi = 2(n-1)$ and, since each team needs to appear exactly once per round, a χ-coloured solution necessarily features $n/2 = |V|/\chi$ vertices-per-colour [51, 63]. Such graphs are thus a highly structured type of flat graph in which the vertices associated with each team constitutes a clique of size $2(n-1)$.[10]

Previous research [59] has demonstrated that, due to the very specific structures of round-robin graphs, optimal solutions for values of n up to (and in some cases exceeding) 50 can be found quite quickly by algorithms such as the six considered here. For this comparison we thus augment the above model by defining additional constraints that specify when certain matches can and cannot occur within a schedule. Such constraints are added as follows:

First, let V_1 be the set of $n(n-1)$ vertices representing matches, constructed as above. Next, add a second set of K vertices V_2 to the graph, each representing a round. Now add edges between all pairs of vertices in V_2 to form a clique of size K, implying that each vertex in V_2 will be assigned a different colour in a feasible K-coloured solution. Finally, impose additional constraints by adding edges between vertices in V_1 and vertices in V_2, signifying that certain matches cannot occur in certain rounds (see fig. 11(b)).

Graphs of size $|V| = |V_1| + |V_2| = n(n-1) + 2(n-1)$ were produced in the above manner for our comparison. Additional constraints were imposed by considering each match-vertex/round-vertex pair in turn, and adding edges between them with probability p. This means for example that if $p = 0.5$, each match can only be assigned to approximately half of the available rounds. Graphs were also generated in two ways: one where $K = \chi$ was ensured, and one where this matter was ignored, possibly resulting in graphs where $\chi > K$. Note that by adding edges in this binomially

[10] The clique arising due to team 0 is shown in bold in the left graph of fig. 11(a)).

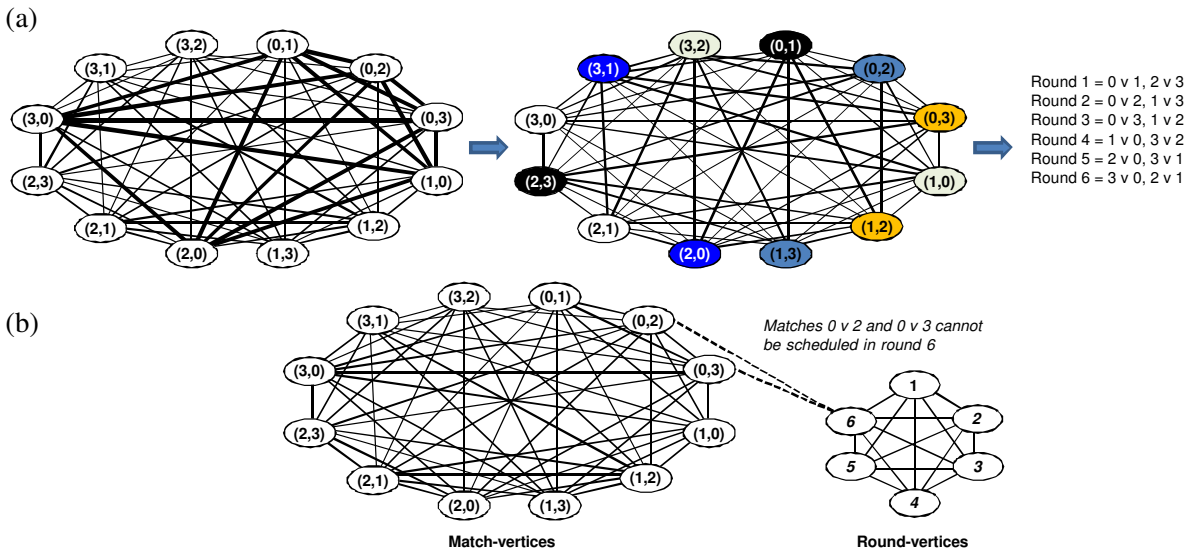


Figure 11: (a) A double round-robin problem graph for $n = 4$, an optimal graph colouring solution, and the corresponding round-robin schedule; (b) An illustration of how the model incorporates additional constraints.

distributed manner, the expected degrees of each vertex $v \in V_1$ and $u \in V_2$ are:

$$\begin{aligned} \mathbb{E}(d(v)) &= 4(n-2) + 1 + p|V_2| \quad \forall v \in V_1, \text{ and} \\ \mathbb{E}(d(u)) &= 2(n-1) - 1 + p|V_1| \quad \forall u \in V_2. \end{aligned} \tag{6}$$

The expected variance in degree across all vertices in V is thus approximated:

$$\frac{|V_1|\mathbb{E}(d(v))^2 + |V_2|\mathbb{E}(d(u))^2}{|V|} - \left(\frac{|V_1|\mathbb{E}(d(v)) + |V_2|\mathbb{E}(d(u))}{|V|}\right)^2. \tag{7}$$

The effect that p therefore has on the overall degree CV of a graph is demonstrated in fig. 12. As p is increased from zero, $\mathbb{E}(d(v))$ and $\mathbb{E}(d(u))$ initially become more similar, resulting in a slight drop in the CV. However, as p is increased further, $\mathbb{E}(d(u))$ rises more quickly than $\mathbb{E}(d(v))$, resulting in large increases in the CV.

The consequences of these specific characteristics help to explain the performance of the six algorithms across the instances, summarised in fig. 13. As with earlier results, the quality of solution achieved by TabuCol and PartialCol are observed to be substantially worse than the other approaches when p, and thus degree CV, is high. In particular TabuCol shows very disappointing performance, providing the worst-quality results for both sizes of graph in all cases where CV$\gtrsim 40\%$.

In contrast, the best performance across the instances is once again due to the HEA, again illustrating the importance of using global operators in conjunction with local search. Surprisingly, AntCol also performs well here, with no significant difference being observed in the mean results of HEA and AntCol across the set. The reasons for the improved performance of AntCol, particularly with denser graphs, seems due to two factors: (a) the higher degrees of the vertices in the graphs, and (b) the high variance in degrees. In AntCol's BuildSolution procedure the first factor naturally increases the influence of the heuristic value η in eq. (3), while the second allows a greater discrimination between vertices. In these cases it seems that a favourable balance between heuristic and pheromone information is being struck, allowing AntCol's global operator to effectively contribute to the search. Note, however, that if we examine individual values of p, the picture becomes more complicated. Figure 14, for example, shows run profiles of AntCol and

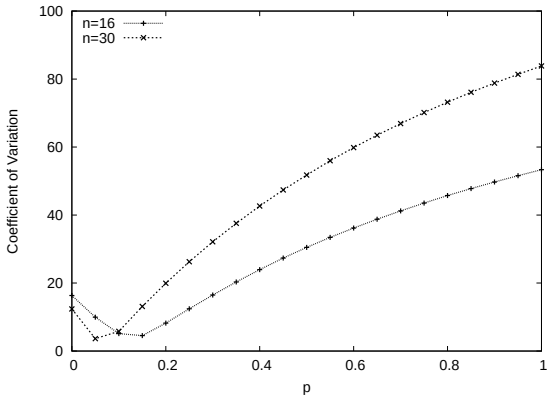


Figure 12: Effect of varying p on the degree coefficient of variation with round-robin graphs of size $n = 16$, 30.

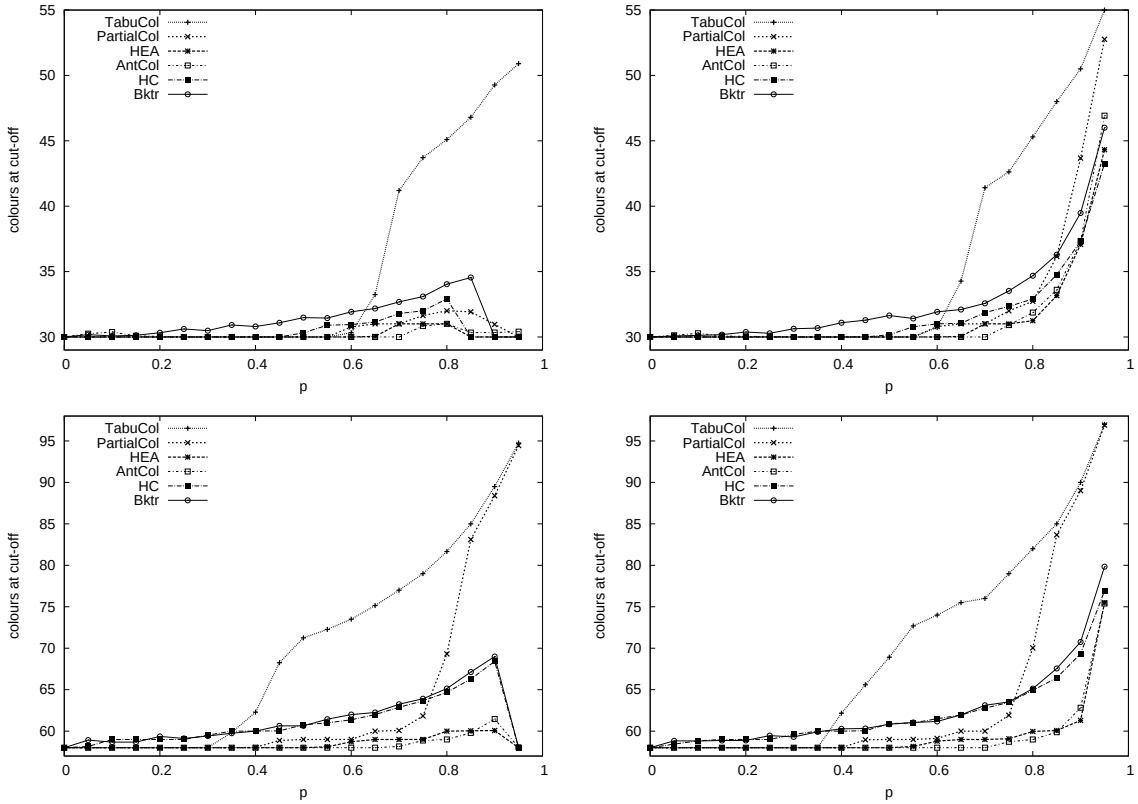


Figure 13: Mean quality of solutions achieved with round-robin graphs using (respectively): $n = 16$, ($|V| = 270$, $K = 30$) $\chi = 30$; $n = 16$, $\chi \geq 30$; $n = 30$, ($|V| = 928$, $K = 58$), $\chi = 58$; and $n = 30$, $\chi \geq 58$. All points are the average of 25 runs on 25 graphs.

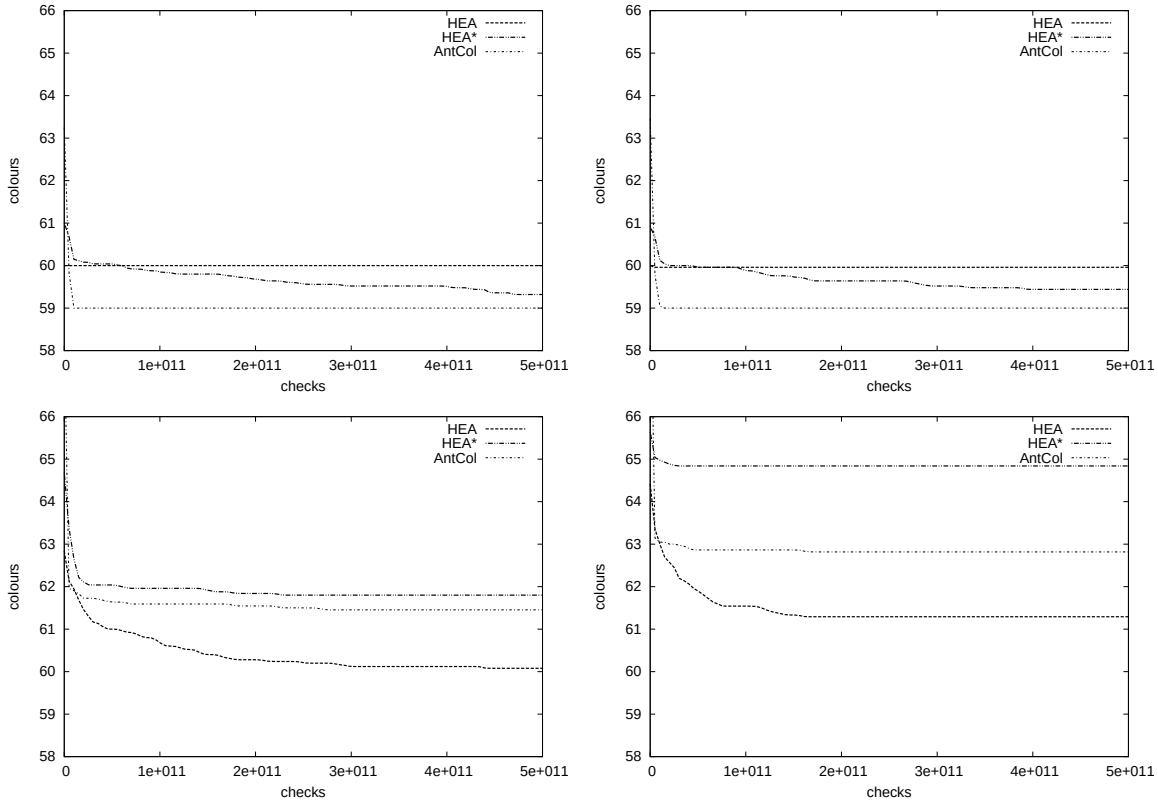


Figure 14: Run profiles for $n = 30$ ($|V| = 928$) using respectively: $p = 0.8$, $\chi = 58$; $p = 0.8$, $\chi \geq 58$; $p = 0.9$, $\chi = 58$; and $p = 0.9$, $\chi \geq 58$. HEA* denotes the HEA algorithm with a reduced local-search limit of $I = 2|V|$

HEA on four sets of large, highly constrained round-robin graphs. For comparison, the variant HEA* is also included here, which features a local-search iteration limit equal to AntCol's (i.e. I reduced from $16|V|$ to $2|V|$). For $p = 0.8$ we see that AntCol quickly produces the best observed results, with HEA consistently requiring an additional colour. However, reducing the local search element of the HEA – thus placing more emphasis on global search – seems to improve performance, though the results of HEA* are still inferior to those of AntCol. On the other hand, for $p = 0.9$ the picture is reversed, with HEA (using $I = 16|V|$) clearly producing the best results, suggesting increased amounts of local search are beneficial in this case.

5 Conclusions and Discussion

This paper has presented a broad comparison of six graph colouring algorithms, detailing the results of over 40,000 individual trials. An analysis of these results reveals a complicated picture, with each algorithm outperforming all others on at least one occasion. This suggests the underlying structures of graphs (themselves not always easy to gauge) are often critical in subsequent algorithm performance. Indeed, in some cases we have seen that even very slight adjustments to graph parameters can significantly alter the relative performances of the algorithms.

In terms of overall patterns we offer the following observations:

As our experiments have shown, methods that rely solely on local search (in this case TabuCol and PartialCol) often struggle with instances whose cost-landscapes are "spiky", commonly characterised by high CVs in vertex degrees. On the other hand, such methods do fare better in cases where the CV is quite low, such as random and flat graphs, suggesting that they have a natural aptitude for navigating spaces in which neighbouring solutions often feature costs that are close or equal to the incumbent. The latter assertion is also backed by research of Blochliger and Zufferey [9]

who find that PartialCol produces more consistent results under cost function $f_3 = |S|$, whose cost landscape features more plateaus than the alternative cost function $f_4 = \sum_{v \in S} d(v)$.

Across the trials, HEA has proved by far the most consistent of the six approaches. We suggest this is due to a combination of positive attributes:

- *The HEA operates in the space of infeasible solutions.* Unlike the HC algorithm, which only permits changes to a solution that maintain feasibility, the strategy of allowing infeasible solutions offers higher levels of connectivity (and thus less restriction of movement) in the search space, helping the algorithm to navigate its way towards high-quality solutions more effectively.

- *The HEA makes use of global as well as local search operators.* On many occasions TabuCol performs poorly when used in isolation; however, HEA's use of a global search operator in conjunction with TabuCol seems to alleviate these problems by allowing the algorithm to escape from local optima.

- *HEA's global search operators are robust.* Unlike AntCol's global search operator, which sometimes hinders performance, HEA's use of crossover in conjunction with a small population of candidate solutions, seems beneficial across the instances. Note in particular that the GPX crossover operator does not consider any problem-specific information in its operations (such as the connectivity or degree of vertices), yet it still seems to strike a useful balance between (a) altering the solution sufficiently, while (b) maintaining useful substructures formed in earlier iterations of the algorithm. Recent research on operators of this nature has been disseminated by Lü and Hao [60] and Porumbel et al. [70].

One surprising result of these experiments is the consistent performance of the HC algorithm with the timetabling problems, particularly as it has performed relatively poorly in other cases. To examine why this is the case, let us define a metric for measuring the distance between two solutions: Given a solution $\mathcal{S}$, let $P_{\mathcal{S}} = \{\{u, v\} | c(u) = c(v)\}$, for $\forall u, v \in V$, $u \neq v$. The *distance* between two solutions $\mathcal{S}_1$ and $\mathcal{S}_2$ can be defined:

$$D(\mathcal{S}_1, \mathcal{S}_2) = \frac{|P_{\mathcal{S}_1} \cup P_{\mathcal{S}_2}| - |P_{\mathcal{S}_1} \cap P_{\mathcal{S}_2}|}{|P_{\mathcal{S}_1} \cup P_{\mathcal{S}_2}|}. \tag{8}$$

This measure thus gives the proportion of vertex pairings (assigned to the same colour) that exist in just one of the two solutions. Consequently, if $\mathcal{S}_1$ and $\mathcal{S}_2$ are identical, then $P_{\mathcal{S}_1} \cup P_{\mathcal{S}_2} = P_{\mathcal{S}_1} \cap P_{\mathcal{S}_2}$, giving $D(\mathcal{S}_1, \mathcal{S}_2) = 0$. Conversely, if no vertex pair is assigned the same colour in both solutions, $P_{\mathcal{S}_1} \cap P_{\mathcal{S}_2} = \emptyset$, implying $D(\mathcal{S}_1, \mathcal{S}_2) = 1$.

Figure 15 presents the distributions of distances between solutions before and after applications of HC's macro-move operator for a variety of graph-types, including five timetabling problems where HC is seen to perform well compared to other approaches. For the timetabling problems, observe that the changes made are clearly larger than for other types of graph. This is due to the fact that solutions to these problems feature independent sets whose sizes vary widely, which often allows vertices to be moved into different independent sets during applications of the first-fit algorithm.[11] Recall, however, that an application of this operator does not allow the number of colours being used to increase – thus these large moves are not associated with any degradation in solution quality. For these particular instances it thus seems that HC's macro move operator provides a useful mechanism for exploring new regions of the search space, discouraging stagnation at any particular local optimum. However, this does not seem to be the case for the other types of graph considered, where the changes induced by first-fit tend to be smaller.

Given such observations, one useful avenue of future research would be to investigate whether *combinations* of different global search operators might be used to improve the performance of an algorithm. It would also be helpful to investigate whether other neighbourhood moves can also

[11] Note that the largest distances occur with `rye-s-93`, which is also the problem instance for which HC has outperformed all others.

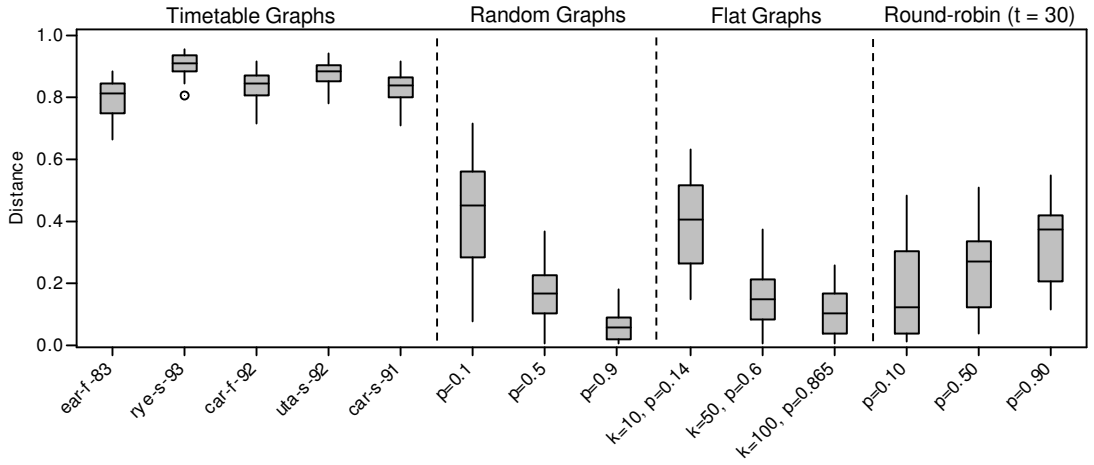


Figure 15: Boxplot showing the distance between solutions before and after applications of the HC algorithm's first-fit procedure. For each instance, samples were formed using solutions featuring one colour more than HC's observed best. For random and flat graphs $|V| = 500$ was used.

	$\|V\| = 250$	500	1000
TabuCol	3467	3581	3512
PartialCol	3986	3653	3435
HEA	3348	3425	4195
AntCol	8644	7224	9505
HC	11799	9596	8397
Bktr	8201	6578	5771

Table 2: Time (in seconds) to complete runs of 5×10^{11} checks using random graphs with $p = 0.5$.

be built into a TabuCol-based method, such as vertex-swaps and Kempe chains, as this would certainly increase the connectivity of the underlying search space. We might also consider applying a Kempe chain operator to the proper partial solutions generated by PartialCol because, while their application will not worsen a solution (i.e. the set of unplaced vertices S will not change), the alterations made might be sufficient to allow the algorithm to sample new areas of the search space, allowing better solutions to be returned over the long term.

As mentioned in Section 1, one of our intentions in this comparison has been to test the robustness of the various algorithms by executing them blindly on each instance. For each algorithm this has involved using the same parameter-values (or methods for calculating them) across all trials. However, while we acknowledge that different settings may have led to better results in some cases, tuning the algorithms for each individual case would have been far too excessive. Also mentioned earlier, unlike many previous studies computational effort in our trials has been viewed using a platform independent measure, though ultimately the algorithms were still compiled under the same conditions (and in all but one case in the same language). In terms of CPU time, Table 2 shows the relative run-times of the algorithms for a small sample of graphs (executed on a PC under Windows XP with a 3.0 GHz processor and 3.18 GB RAM). Perhaps the most striking feature is that the HEA is among one of the quickest to execute, a fact that further endorses the method. On the other hand, the AntCol and the HC algorithms seem to require significantly more CPU time, apparently due to the computational overheads associated with their BuildSolution and Kempe chain operators respectively.

6 Acknowledgements

This work was partially carried out using the computational facilities of the Advanced Research Computing @ Cardiff (ARCCA) Division, Cardiff University.

This research also uses data from Add Health, a program project directed by Kathleen Mullan Harris and designed by J. Richard Udry, Peter S. Bearman, and Kathleen Mullan Harris at the University of North Carolina at Chapel Hill, and funded by grant P01-HD31921 from the Eunice Kennedy Shriver National Institute of Child Health and Human Development, with cooperative funding from 23 other federal agencies and foundations. Special acknowledgment is due to Ronald. R. Rindfuss and Barbera Entwhisle for assistance in the original design. Information on how to obtain the Add Health data files is available on the Add Health website (www.crc.unc.edu/addHealth). No direct support was received from grant P01-HD31921 for this current publication.

References

[1] ftp://ftp.mie.utoronto.ca/pub/carter/testprob/all_file.zip.

[2] http://dimacs.rutgers.edu/challenges/.

[3] http://rhydlewis.eu/papers/gcolsisterpaper.pdf.

[4] http://web.cs.ualberta.ca/∼joe/coloring/.

[5] http://www.bloechligair.ch/science/.

[6] K. I. Aardel, S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino, and A. Sassano. Models and solution techniques for the frequency assignment problems. *4OR : Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4):1–40, 2002.

[7] A. Anagnostopoulos, L. Michel, P. van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

[8] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operations Research*, 151:379–388, 2003.

[9] I. Blochliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers and Operations Research*, 35:960–975, 2008.

[10] D Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[11] R. Brown. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4):451–463, 1972.

[12] E. Burke, D. de Werra, and J. Kingston. *Handbook of Graph Theory.*, chapter Applications to timetabling, pages 445–474. CRC Press, 2003.

[13] E. Burke, D. Elliman, and R. Weare. Specialised recombinative operators for timetabling problems. In *The Artificial Intelligence and Simulated Behaviour Workshop on Evolutionary Computing*, volume 993, pages 75–85. Springer-Verlag, Berlin, 1995.

[14] E. K. Burke and J. P Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

[15] M. P. Carrasco and M. V. Pato. A multiobjective genetic algorithm for the class/teacher timetabling problem. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling (PATAT) III*, volume 2079, pages 3–17. Springer-Verlag, Berlin, 2001.

[16] M. Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, 1986.

[17] M. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47:373–383, 1996.

[18] G. Chaitin. Register allocation and spilling via graph coloring. *SIGPLAN Not.*, 39(4):66–74, 2004.

[19] M. Chams, A. Hertz, and O. Dubuis. Some experiments with simulated annealing for coloring graphs. *European Journal of Operations Research*, 32:260–266, 1987.

[20] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331–337, 1991.

[21] M. Chiarandini and T. Stützle. An application of iterated local search to graph coloring. In In D. Johnson, A. Mehrotra, and M. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, New York, USA, 2002.

[22] A. Colorni, M. Dorigo, and V Maniezzo. Metaheuristics for high-school timetabling. *Computational Optimization and Applications*, 9(3):277–298, 1997.

[23] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[24] P. Cote, T. Wong, and R. Sabourin. Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling (PATAT) V*, volume 3616, pages 294–312. Springer-Verlag, Berlin, 2005.

[25] B. G. W. Craenen. *Solving Constraint Satisfaction Problems with Evolutionary Algorithms*. PhD thesis, Vrije Universiteit Amsterdam, 2005.

[26] J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 245–284. American Mathematical Society, 1996.

[27] D. de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21:47–65, 1988.

[28] L. Di Gaspero and A. Schaerf. Multi-neighbourhood local search with application to course timetabling. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling (PATAT) IV*, volume 2740, pages 263–287. Springer-Verlag, Berlin, 2002.

[29] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[30] R. Dorne and J-K. Hao. A new genetic local search algorithm for graph coloring. In A. Eiben, T. Back, M. Schoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN) V*, volume 1498, pages 745–754. Springer-Verlag, Berlin, 1998.

[31] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem: description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 580–585. Springer, Berlin/Heidelberg, 2001.

[32] A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[33] E. Erben. A grouping genetic algorithm for graph colouring and exam timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling (PATAT) III*, volume 2079, pages 132–158. Springer-Verlag, Berlin, 2001.

[34] E. Falkenauer. Solving equal piles with the grouping genetic algorithm. In L. J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 492–497. Morgan Kaufmann Inc, 1995.

[35] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[36] C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph colouring. *Annals of Operational Research*, 63:437–461, 1996.

[37] P. Galinier and J-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.

[38] P. Galinier and A. Hertz. A survey of local search algorithms for graph coloring. *Computers and Operations Research*, 33:2547–2562, 2006.

[39] P. Galinier, A. Hertz, and N. Zufferey. An adaptive memory algorithm for the k-coloring problem. *Discrete Applied Mathematics*, 156(2), 2008.

[40] M. Garey, D. Johnson, and H. So. An application of of graph coloring to printed circuit testing. *IEEE Transactions on Circuits and Systems*, CAS-23:591–599, 1976.

[41] B. Gendron, A. Hertz, and P. St-Louis. On edge orienting methods for graph coloring. *Journal of Combinatorial Optimization*, 13(2):163–178, 2007.

[42] C. Glass and A. Prugel-Bennett. Genetic algorithms for graph coloring: Exploration of galnier and hao's algorithm. *Journal of Combinatorial Optimization*, 7:229–236, 2003.

[43] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[44] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[45] A. Hertz, M. Plumettaz, and N. Zufferey. Variable space search for graph coloring. *Discrete Applied Mathematics*, 156(13):2551–2560, 2008.

[46] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley-Interscience, first edition, 1994.

[47] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part ii graph colring and number partitioning. *Operations Research*, 39:378–406, 1991.

[48] I. Juhos and J. I. van Hemert. Increasing the efficiency of graph colouring algorithms with a representation based on vector operations. *Journal of Software*, 1(2):24–33, August 2006.

[49] M. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85–103. Plenum, New York, 1972.

[50] M. Kearns, S. Suri, and N. Montfort. An experimental study of the coloring problem on human subject networks. *Science*, 313(5778):824–827, 2006.

[51] T. Kirkman. On a problem in combinations. *Cambridge Dublin Math Journal*, 2:191–204, 1847.

[52] S. Korman. *Combinatorial Optimization.*, chapter The Graph-Colouring Problem, pages 211–235. Wiley, New York, 1979.

[53] M. Kubale and B. Jackowski. A generalized implicit enumeration algorithm for graph colouring. *Commun. ACM*, 28(28):412–418, 1985.

[54] M. Laguna and R. Marti. A grasp for coloring sparse graphs. *Computational Optimization and Applications*, 19:165–78, 2001.

[55] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.

[56] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

[57] R. Lewis. A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers and Operations Research*, 36(7):2295–2310, 2009.

[58] R. Lewis and B. Paechter. Finding feasible timetables using group based operators. *IEEE Transactions on Evolutionary Computation*, 11(3):397–413, 2007.

[59] R. Lewis and J. Thompson. On the application of graph colouring techniques in round-robin sports scheduling. *Computers and Operations Research*, 38(1):190–204, 2010.

[60] Z. Lü and J-K Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241 – 250, 2010.

[61] E. Malaguti, M. Monaci, and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

[62] A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.

[63] E. Mendelsohn, R. Rosa, A.Melo, S. Urrutia, and C. Riberiro. One-factorizations of the complete graph – a survey. *Journal of Graph Theory*, 9:43–65, 1985.

[64] J. Moody and D. White. Structural cohesion and embeddedness: A hierarchical concept of social groups. *American Sociological Review*, 68(1):103–127, 2003.

[65] C. Morgenstern. *Algorithms for General Graph Coloring*. PhD thesis, University of New Mexico, 1989.

[66] C. Morgenstern. Distributed coloration neighborhood search. *Discrete Mathematics and Theoretical Computer Science*, 26:335–358, 1996.

[67] C. Mumford. New order-based crossovers for the graph coloring problem. In *Parallel Problem Solving from Nature IX (PPSN IX)*, volume 4193 of *LNCS*, pages 880–889. Springer-Verlag, 2006.

[68] B. Paechter, R. Rankin, A. Cumming, and T. Fogarty. Timetabling the classes of an entire university with an evolutionary algorithm. In T. Baeck, A. Eiben, M. Schoenauer, and H. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN) V*, volume 1498, pages 865–874. Springer-Verlag, Berlin, 1998.

[69] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Gunther Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279, pages 121–130, Kinsale, Ireland, 3-4 2002. Springer-Verlag.

[70] D. Porumbel, J-K Hao, and P. Kuntz. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and operations research*, 37:1822–1832, 2010.

[71] S. Prestwich. Using an incomplete version of dynamin backtracking for graph colouring. *Electronic Notes in Discrete Mathematics*, 1:61–73, 1999.

[72] P. Ross, D. Corne, and H. Terashima-Marin. The phase-transition niche for evolutionary algorithms in timetabling. In E Burke and P Ross, editors, *Practice and Theory of Automated Timetabling (PATAT) I*, volume 1153, pages 309–325. Springer-Verlag, Berlin, 1996.

[73] P. Spinrad and G. Vijayan. Worse case analysis of a graph colouring algorithm. *Discrete Applied Mathematics*, 12:89–92, 1984.

[74] J. Thompson and K. Dowsland. An improved ant colony optimisation heuristic for graph colouring. *Discrete Applied Mathematics*, 156:313–324, 2008.

[75] M. A. Trick. A schedule-then-break approach to sports timetables. In E. Burke and W. Erben (editors), editors, *The Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 242–253. Springer-Verlag, 2001.

[76] A. Tucker, J. Crampton, and S. Swift. Rgfga: An efficient representation and crossover for grouping genetic algorithms. *Evolutionary Computation*, 13(4):477–499, 2005.

[77] J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.

[78] C. M. Valenzuela. A study of permutation operators for minimum span frequency assignment using an order based representation. *Journal of Heuristics*, 7:5–21, 2001.

[79] D. Welsh and M. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *Computer Journal*, 12:317–322, 1967.

[80] M. Wright. Scheduling fixtures for basketball New Zealand. *Computers and Operations Research*, 33(7):1875–1893, 2006.