Stochastic Local Search Algorithms for the Graph Colouring Problem

Marco Chiarandini, Irina Dumitrescu, and Thomas Stützle

Forschungsbericht AIDA-05-03

Darmstadt, August 2005

Fachgebiet Intellektik, Fachbereich Informatik
Technische Hochschule Darmstadt
Alexanderstraße 10
D-64283 Darmstadt
Germany

Intellektik!

Stochastic Local Search Algorithms for the Graph Colouring Problem

Marco Chiarandini[†], Irina Dumitrescu[‡], Thomas Stützle[⋆]

[†]Department of Mathematics and Computer Science, University of Southern Denmark
Campusvej 55, DK-5230 Odense, Denmark
marco@imada.sdu.dk

[‡]Canada Research Chair in Distribution Management, HEC Montréal
C.P.6128, succursale Centre-ville, Montreal, Canada H3J 3J7
irina@crt.umontreal.ca

[⋆]IRIDIA, Université Libre de Bruxelles
CP 194/6, Avenue Franklin Roosevelt 50, 1050 Brussels, Belgium
stuetzle@ulb.ac.be

1 Introduction

The (vertex) graph colouring problem (GCP) is a central problem in graph theory [36] and it arises in many real life applications like register allocation [2], air traffic flow management [4], frequency assignment [28], light wavelengths assignment in optical networks [58], or timetabling [19, 44].

In the GCP, one is given an undirected graph $G = (V, E)$, with V being the set of $|V| = n$ vertices and E being the set of edges, we call a *k-colouring* of G, a mapping $\phi : V \mapsto \Gamma$, where $\Gamma = \{1, 2, \ldots, k\}$ is the set of $|\Gamma| = k$ integers, each one representing one colour. We say that a colouring is *feasible* or *legal* if for all $[u, v] \in E$ we have that $\varphi(u) \neq \varphi(v)$; otherwise we say that the colouring is *infeasible*. If for some $[u, v] \in E$ we have that $\varphi(u) = \varphi(v)$, the vertices u and v are *in conflict*. The *conflict set* V^C is the set of all vertices that are in conflict. A k-colouring can also be seen as a partitioning of the set of vertices into k disjoint sets, called *colour classes* and represented as a partitioning of V, $C = \{C_1, \ldots, C_k\}$. Finally, if some vertices are assigned to colour classes while others are not, we say that we have a *partial colouring*.

The graph (vertex) colouring problem (GCP) can be posed as a decision or an optimisation problem. In the decision version, also called the *(vertex) k-colouring problem*, the question to be answered is whether for some given k a feasible k-colouring exists. The optimisation version of the GCP asks for the smallest number k such that a feasible k-colouring exists; for a graph G, this number is called the *chromatic number* χ_G. The problem of finding the chromatic number is often approached by solving a sequence of k-colouring problems: an initial value of k is considered and each time a feasible k-colouring is found, the value of k is decreased by one. The chromatic number is found when for some k the answer to the decision version is no, that is, a feasible k-colouring does not exist. In this case, $\chi_G = k + 1$. If a feasible colouring cannot be found but no proof of its non-existence is given, as it is typically the case with heuristic algorithms, $k + 1$ is an upper bound on the chromatic number.

It is well known that the k-colouring problem for general graphs is $\mathcal{NP}$-complete and that the chromatic number problem is $\mathcal{NP}$-hard[41]; only for a few special cases polynomial time algorithms are known. In some sense, the GCP is among the hardest problems in $\mathcal{NP}$, since approximation ratios of $n^{1-\epsilon}$ cannot be achieved in polynomial time, unless $\mathcal{NP} = \mathcal{ZPP}$ [24]; that is, it is very unlikely to find polynomial time algorithms that guarantee a constant approximation ratio. The best absolute performance guarantee of $\mathcal{O}(n(\log\log n)^2/\log^3 n)$ is given for an approximation algorithm presented in [32]. More details on the approximability of graph colouring can be found in [55].

Due to its importance, many attempts have been made to tackle the GCP algorithmically. Various exact algorithms, including specialised branch–and–bound algorithms [6, 7] or approaches based on general integer programming formulations of the GCP have been tested [21, 30, 49]. Probably the best known exact algorithm is Brélaz' modification of Randall-Brown's colouring algorithm [6]. While exact algorithms can be very effective on specific classes of graphs, their performance for many large graphs is often rather poor [49, 11]. Therefore, a significant amount of research has focused on stochastic local search (SLS) algorithms for the GCP (see Chapter R-17 for an overview of SLS methods). The available SLS approaches range from rather simple but very fast construction methods over iterative improvement algorithms to sometimes rather complex SLS algorithms, which are currently the best performing approximate algorithms for many classes of GCP instances. In this chapter, we give an overview of available SLS algorithms and present some indicative comparison of the performance of several such algorithms.

2 Benchmark instances

The available benchmark instances for the GCP are either randomly generated graphs or derived from some practically relevant application. The most frequently used benchmark instances are available from the webpage *COLOR02/03/04: Graph Coloring and its Generalizations* at `http://mat.gsia.cmu.edu/COLOR04` and the earlier DIMACS challenge [39]. Next, we describe some instance classes which will be used in the experimental analysis presented in Section 7.

Uniform random graphs: This class comprises graphs of a variable number of vertices, n, where each of the $n(n-1)/2$ possible edges is generated independently at random with probability p. The instance have identifiers `DSJC`$n.p$, with $n \in \{125, 250, 500, 1000\}$ and $p \in \{0.1, 0.5, 0.9\}$. They stem from one of the first thorough experimental studies of SLS algorithms for the GCP [37]. For these graphs, estimates of the chromatic number exist [1, 40, 47].

Flat graphs: These graphs are random graphs generated according to an equi-partitioning of vertices into k sets. Edges are then generated to respect some constraints on the resulting degrees of the vertices until a given edge density is obtained [16]. k is an upper bound on the chromatic number of the graph. These instances are denoted as `flat`n_k, with $n = 300$; $k \in \{20, 26, 28\}$ and $n = 1000$; $k \in \{50, 60, 76\}$.

Leighton graphs: Leighton graphs are random graphs of density below 0.25, which are constructed by first partitioning vertices into k distinct sets representing the colour classes and then assigning edges only between vertices that belong to different sets. The chromatic number is guaranteed to be k by implanting cliques of sizes ranging from 2 to k into the graph. The graphs are denoted as `le450_`kx, where 450 is the number of vertices, k is the chromatic number of the graph, $x \in \{a, b, c, d\}$ is a letter used to distinguish different graphs with the same characteristics, with c and d graphs having higher edge density than the a and b ones.

Quasigroup Graphs: A Quasigroup is an algebraic structure on a set with a binary operator. The constraints on this structure define a Latin square, that is a square matrix with elements such that entries in each row and column are distinct. A Latin square of order n has n^2 vertices and $n^2(n-1)$ edges, corresponding to $2n$ cliques, a clique per row/column, each of size n. The chromatic number of Latin square graphs is n. We denote Latin square graphs originated by Quasigroups by `qg.order`n. Latin square graphs arise also in experimental design for statistical analysis. The instance `latin_square_10`, which has an unknown chromatic number, is one such example with preassigned experiments [45].

Queens graphs: The n-queens problem asks whether it is possible to place n queens on a $n \times n$ grid such that no pair of queens is in the same row, column, or diagonal. This problem can be posed as a graph colouring problem and a feasible solution with n queens exists if, and only if, the resulting queen graphs have a feasible colouring with n colours. Queen graphs are highly structured instances and their edge density decreases with their size; they are denoted by `queen`n_n.

WAP graphs: These graphs arise in the design of transparent optical networks [58] and are denoted by `wap0`m`a`, where $m = \{1, \ldots, 8\}$. They have between 905 and 5231 vertices. All instances have a clique of size 40.

Jacobian estimation graphs: These graphs stem from a matrix partitioning problem in the segmented columns approach to determine sparse Jacobian matrices [35]. They range in size from 662 to 1916 vertices and they are identified with the following names `abb313GPIA`, `ash331GPIA`, `ash608GPIA`, `ash958GPIA`, and `will199GPIA`.

The DIMACS benchmark repository contains some other instances that we identified as *easy*, because some combination of graph reduction rules and simple construction heuristics, which are introduced in the next two sections, are enough to find a colouring with the known chromatic number. We identified 45 instances as easy. They are the Mycielski graphs, the graphs from register allocation for variables in compiled code [45], graphs from Knuth's Stanford GraphBase [42], and the almost 3-colourable graphs with embedded four cliques [51]. Two further classes of graphs that are a generalisation of Mycielski graphs (the insertions and full insertions graphs due to Caramia and Dell'Olmo, pers. comm.) and graphs for course scheduling [45] are not useful to determine differences among algorithms.

3 Preprocessing and heuristic reduction rules

In the chromatic number problem, preprocessing can be applied to reduce a graph G to a graph G' such that a feasible k-colouring for G can be derived by construction rules from any feasible k-colouring of G'. Next we give the two reduction rules presented in [10].

Rule 1: Remove all vertices in G that have a degree less than the size of the largest known clique $\widehat{\omega}(G)$. Knowing that the degree of a vertex u is less than $\widehat{\omega}(G)$ guarantees that at least one colour that is not used in the set of adjacent vertices can be assigned to u without breaking feasibility. (This rule can be applied when solving the k-colouring problem by replacing $\widehat{\omega}(G)$ by k).

Rule 2: Remove any vertex $v \in V$ for which there is a $u \in V$, $v \neq u$ and $[u, v] \notin E$, such that u is connected to every vertex to which v is connected (subsumption). In this case, any colour that can be assigned to u can also be assigned to v.

These two rules can be applied in any order and if one rule applies, it may make possible further reductions by the other rule. Hence, in the preprocessing stage, the two rules are applied iteratively until no vertex can be removed anymore. The rules are easy to apply. Rule 1 requires $\mathcal{O}(|V|)$ operations once a clique has been found heuristically and the degree of each vertex is known. Rule 2 is more costly and its time-complexity is $\mathcal{O}(|V|^3)$. The overall reduction time is, however, insignificant in practice. Among the challenging instances that we consider, graph reduction is effective only for the WAP instances.

The graph can also be reduced heuristically. One such procedure consists of reducing the graph by removing maximal independent sets [25, 34, 53]. Typically, this procedure is accomplished with the companion strategy of minimising the number of edges in the residual graph. In contrast to the previously mentioned preprocessing rules, this procedure may make it impossible to find the chromatic number of the original graph because it is a priori unknown how the independent sets should be constructed such that an optimal colouring is obtained from a colouring of the residual graph and the colouring for the independent sets. Nevertheless, for some graphs such a heuristic procedure has contributed significantly to the improvement of the solution quality for SLS algorithms [15, 25, 34, 52].

4 Construction heuristics

The fastest methods to generate a feasible colouring are construction heuristics. Most of these heuristics for the GCP belong to the class of *sequential* heuristics that start from a set of empty colour classes $\{C_1, \ldots, C_k\}$, where $k = |V|$, and then iteratively add vertices to colour classes until a complete colouring is reached. Each iteration of the heuristic consists of two steps: first, the next vertex to be coloured is chosen, and next this vertex is assigned to a colour class. The order in which the vertices are coloured corresponds to a permutation π of the vertex indices that can be determined either *statically* before assigning the colours, or *dynamically* by taking into account the partial colouring for the choice of the next vertex. The choice of the colour class at each construction step is typically based on the *greedy heuristic* that adds at iteration i the vertex $\pi(i)$

to the colour class with the lowest possible index such that the partial colouring obtained after colouring vertex $\pi(i)$ remains feasible, therefore trying to minimise the number of non-empty colour classes.

Clearly, the result of the greedy heuristic depends on the permutation π. The simplest way to derive π in a static way is by using the *random order sequential heuristic* (ROS), which simply generates a random permutation. Several other ways for generating π statically exist, like largest degree first, smallest degree last, etc. However, static sequential methods are typically dominated by dynamic ones [40]. Probably the most widely spread dynamic heuristic is the DSATUR *heuristic* that is derived from the exact DSATUR algorithm of Brélaz [6]. In DSATUR, the vertices are first arranged in decreasing order of their degrees and a vertex of maximal degree is inserted into C_1. Then, at each construction step the next vertex to be inserted is chosen according to the *saturation degree*, that is, the number of differently coloured adjacent vertices. The vertex with the maximal saturation degree is chosen and inserted according to the greedy heuristic. Ties are broken preferring vertices with the maximal number of adjacent, still uncoloured vertices; if further ties remain, they are broken randomly. Other dynamic heuristics for determining π were studied in [9, 20].

A different strategy for generating colourings is to iteratively extract independent sets from the graph. The most famous such heuristic is the *recursive largest first (*RLF*)* heuristic proposed by Leighton [44]. RLF iteratively constructs a colour class C_i by first assigning to it a vertex v with maximal degree from the set V' of still uncoloured vertices; initially we have $V' = V$. Next, all the vertices in V' that are adjacent to v are removed from V' and inserted into a set U, which is initially empty; U is the set of vertices that cannot be added to colour class C_i anymore. Then, while V' is not empty, the vertex $v' \in V'$ that has the largest number of edges $[v', u]$, with $u \in U$, is chosen; v' is added to C_i, removed from V', and all vertices in V' adjacent to v' are moved into U. Ties are broken, if possible, choosing the vertex that has minimal degree in V', otherwise randomly. These steps are iterated until V' is empty and the same steps are repeated with the residual graph consisting of the vertices in U.

We compared the three construction heuristics DSATUR, RLF, and ROS empirically. Since they are stochastic procedures, due to the random tie breaking employed, we ran each of the algorithms ten times on all 125 instances of the two previously mentioned benchmark sets as of end 2004. The conclusion was that RLF performs statistically significantly better than DSATUR for most instance classes and both heuristics are by a large margin better than ROS. RLF is not significantly better than DSATUR only on the insertions, full insertions, and course scheduling graphs. With respect to computation time, although RLF and DSATUR have the same time complexity $\mathcal{O}(|V|^3)$, RLF is in practice much more time-consuming and ROS with a complexity of $\mathcal{O}(|V|^2)$ is the fastest. For example, on the largest instance tested (10000 vertices), RLF takes on average about 18.5 seconds to colour the graph with 101 colours, DSATUR 3 seconds to colour it with 103 colours, and ROS less than 1 second to produce a colouring with 106 colours (the computer used was a 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB of RAM memory). A more detailed analysis showed that the computation time of RLF depends not only on the instance size, as it is the case for DSATUR, but it is also affected by the graph density and the final number of colours used [11].

Note that the construction heuristics described above can also be used if a fixed number $k < |V|$ of colour classes is available. In that case, the usual steps of the construction algorithms are followed as long as the number of used colour classes remains below k. Then, if a vertex cannot be coloured legally, it is added to a colour class randomly or according to some other heuristic that tries to minimise the number of arising conflicts. The result of these so modified construction heuristics is typically an infeasible k-colouring, which can serve as an initial solution for improvement methods.

5 Neighbourhoods for local search

Once an initial (feasible or infeasible) colouring is generated by construction algorithms, one may try to improve it through the application of iterative improvement algorithms. These algorithms may be used within two different approaches to solve the GCP: as a sequence of decision problems or directly as an optimisation problem. The first approach corresponds to leaving the numbers of colours *fixed* to some value k at each stage. Subsequently, once a feasible colouring with k colours is found, the number of colours is reduced by one. The second approach allows the number of used colours to *vary* throughout one single trial of the local search algorithm. To apply iterative improvement algorithms one needs to define the a set of candidate solutions, a neighbourhood relation, and an evaluation function. Two choices are possible for

the set of *candidate solutions*: one may opt for the algorithm to work on *complete colourings*, where each candidate solution represents a possibly infeasible partitioning of the set of vertices into colour classes, or to work on *partial colourings*, where candidate solutions are also those with a subset of the vertices left without any colour assigned. Next, we present several neighbourhood relations and evaluation functions of candidate solutions, differentiating among possible choices of how to tackle the GCP.

Strategy 1: k fixed, complete colourings

This approach works on a partitioning of V into k colour classes: $C = \{C_1, \ldots, C_k\}$. The most widely used evaluation function counts the number of edges that have their end points in the same colour class. Formally, the function can be described as $g(C) = \sum_{i=1}^{k} |E_i|$, where E_i is the set of edges with both end points in C_i. A candidate solution with an evaluation function value of zero corresponds to a legal k-colouring. An alternative evaluation function would be $g(C) = |V^C|$. However, this function is much less used, possibly because it would lead to a large number of neighbours with the same evaluation function value, therefore inducing large plateaus.

1-exchange neighbourhood. The most frequently used neighbourhood structure in this setting is the 1-exchange neighbourhood, in which two colourings are neighbours if they differ in the colour class assignment of exactly one vertex. That is, to obtain a neighbour of C, one vertex u is moved from a colour class C_j, $j \in \{1, \ldots, k\}$ into a different colour class C_l, $l \neq j$. Often, the 1-exchange neighbourhood is further restricted to change only the colour class assignment of vertices that are in conflict, since only these modifications can lead to a decrease of the evaluation function; we call this the *restricted 1-exchange neighbourhood*.

Swap neighbourhood. In the swap neighbourhood exactly one vertex $v \in V^C$ exchanges the colour class with another vertex $u \in V$. This neighbourhood is of quadratic size and it is used only very rarely.

Cyclic exchange and path exchange neighbourhoods. An extension of the 1-exchange and swap neighbourhoods are the path and cyclic exchange neighbourhoods. Both the cyclic and the path exchange are sequences of 1-exchanges. A *cyclic exchange* of length m acts on a sequence of distinctly coloured vertices $(u_1, \ldots, u_m)$. For simplicity, we will denote the colour class of any u_i, $i = 1, \ldots, m$, by C_i. The cyclic exchange moves any u_i, $i = 1, \ldots, m$, from C_i into C_{i+1}. We use the convention $C_{m+1} = C_1$. A cyclic exchange does not change the cardinality of the colour classes involved in the move. In a *path exchange* instead, u_m remains in C_m. Hence the sequence of exchanges is not closed and the cardinality of C_1 and C_m is modified. The cyclic and path exchange neighbourhoods are examples of very large scale neighbourhoods (VLSN). The problem of finding a good neighbour within these neighbourhoods can be reduced to searching the least cost cycle or cost path in an improvement graph. If these problems can be solved exactly, then the best neighbour is found. We refer to chapter R-18 for more details on how an improvement graph is built and can be searched exactly and heuristically. Details on the implementation of a VLSN for the GCP can be found in [11].

Other neighbourhoods. Similar to the path exchange neighbourhood is the ejection-chain neighbourhood defined in [31], where the length of the sequences was limited to 3. Other neighbourhoods, rather fancy at times, were proposed in the context of a variable neighbourhood approach [3]. However, the overall contribution of these neighbourhoods is somewhat unclear, especially when they are used inside a more complex SLS algorithm.

Strategy 2: k fixed, partial colourings

This approach works on a partitioning of V into $k+1$ colour classes: $C = \{C_1, \ldots, C_k, C_{imp}\}$. The partial colouring $\bar{C} = \{C_1, \ldots, C_k\}$ is usually required to be feasible. The colour class C_{imp} is called the *impasse* class [52] and it contains the "uncoloured" vertices. The goal of the local search is to try to empty colour class C_{imp}, while maintaining the partial colouring $\bar{C}$ feasible.

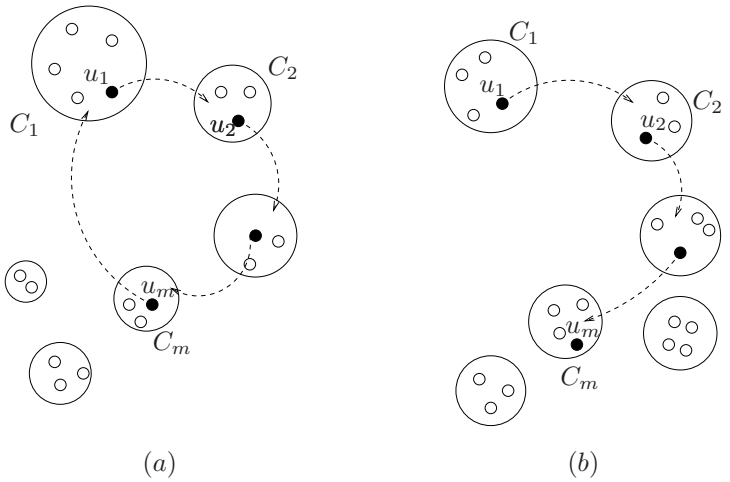

Figure 1.1: The black vertices are the vertices involved in the cyclic and path exchange; (a) cyclic exchange, (b) path exchange.

The most widely used evaluation function in this case is $g(C) = \sum_{v \in C_{imp}} d(v)$, where $d(v)$ is the degree of vertex v. Analogously to complete colourings, an alternative would be to minimise $g(C) = |C_{imp}|$. However, this appears to lead to worse performance [53].

***i*-swap neighbourhood.** A neighbour of C in the i-swap neighbourhood is obtained by moving a vertex v from C_{imp} into another colour class C_i, followed by moving all vertices of C_i that share an edge with v into C_{imp} so that the partial colouring $\bar{C}$ remains feasible [52].

Other neighbourhood. The neighbourhoods discussed in the section dedicated to Strategy 1 could, at least in principle, also be applied for algorithms using strategy 2; however, it would probably be difficult to maintain the partial colouring feasible and additional penalties for infeasibility may be required in the evaluation function. So far, such an approach appears not to have been tried.

Strategy 3: *k* variable, complete colourings

The final strategy we discuss allows the number of colour classes to vary during the local search. In almost all these approaches, the current candidate colouring is forced to be complete and feasible. The local search then tries to minimise the number of colour classes. Hence, the simplest evaluation function would be to count the number of colours currently used; however, since moves that remove one colour class completely will be certainly very rare, the guidance provided by this evaluation function would be minor. As an alternative, Johnson *et al.* [37] proposed to use $g(C) = -\sum_{i=1}^{k}(|C_i|)^2$, which biases the search towards a small number of colour classes. The most widely used neighbourhood structure for this strategy is based on Kempe chains.

Kempe-chains neighbourhood. A *Kempe chain* K is a set of vertices that form a maximal connected graph in the subgraph G' of G induced by the vertices that belong to two (disjoint) colour classes C_i and C_j of C. A Kempe chain exchange applied to a feasible colouring produces again a feasible colouring (a *Kempe chain neighbour*) by moving all vertices of C_i that belong to the Kempe chain K into the colour class C_j and vice versa. An example of a Kempe chain is given in Figure 1.2.

An alternative to enforcing colouring feasibility would be to also allow infeasibilities together with leaving k variable. In that case, the same neighbourhoods as when keeping k fixed can be applied, but the evaluation function needs now to also guide the search towards feasible candidate solutions. One such evaluation function is $g'(C) = -\sum_{i=1}^{k}|C_i|^2 + \sum_{i=1}^{k} 2|V_i||E_i|$, where E_i is the set of vertices with both end points in C_i [37]. The second term in g' is used to penalise conflicts between connected vertices.

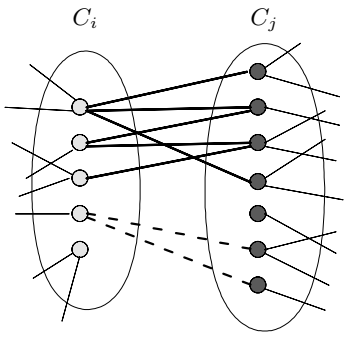

Figure 1.2: Two Kempe chains are available between colour classes C_i and C_j and are indicated by the thick and dashed lines, respectively.

Finally, note that we are not aware of any approach that leaves k variable and uses partial colourings, although such an approach would be conceivable.

Neighbourhood examination and data structures

The various neighbourhoods we have described can be restricted and explored in various ways. It is intuitively clear that neighbourhood restrictions are more important for large neighbourhoods than for the small ones. Nevertheless, computational results with SLS algorithms suggest that such restrictions are also essential even for the 1-exchange neighbourhood.

Regarding the neighbourhood search strategy, we can distinguish mainly between best- and first-improvement strategies. In best-improvement, the best neighbouring candidate solution (best w.r.t. neighbourhood restrictions, if these are used) is accepted, breaking ties randomly; in first-improvement, the first improving candidate solution found when scanning the neighbourhood replaces the current one. Somehow intermediate between these two is the strategy followed in the *min-conflicts heuristic* [50], which searches the restricted 1-exchange neighbourhood in a two-stage process. In a first stage, a vertex is chosen uniformly at random from the conflict set; in a second stage, this vertex is moved into a colour class such that the number of conflicts is minimised, breaking ties randomly.

Finally, let us note that the local search algorithms for the GCP require the usage of appropriate data structures to support caching and updating strategies to make the evaluation of the neighbouring candidate solutions as efficient as possible. Typically, the use of elaborated data structures is more important for best-improvement local search algorithms and those making use of large neighbourhoods. We refer to [11] for a short discussion of efficient caching and updating strategies.

6 Stochastic local search algorithms

In the case of the GCP, iterative improvement algorithms have received rather little attention, probably because of their generally poor performance when used as stand-alone algorithms. Much more attention has focused on the application of more complex SLS algorithms. In what follows, we present a selection of some of the best-performing SLS algorithms, as well as a few recently developed ones. The selection was biased by the desire to cover different approaches and methods. Where appropriate, we also give a short overview of algorithms that are related to the ones we describe, covering in this way the majority of the SLS algorithms proposed so far for the GCP. If nothing else is said in the text, the evaluation functions used are the standard ones given for the various strategies described in the previous section.

6.1 Strategy 1: k fixed, complete colourings

Tabu search with 1-exchange neighbourhood. Tabu search algorithms based on the 1-exchange neighbourhood ($\mathsf{TS}_{\mathsf{1\text{-}ex}}$) are probably the most frequently applied SLS algorithms for the GCP (see Chapter R-23 for details on tabu search). Such an algorithm was first proposed by Hertz and de Werra [34] and was later

improved leading to the most performing tabu search variant to date by Dorne, Galinier, and Hao [22, 26]. $\mathsf{TS_{1\text{-}ex}}$ chooses at each iteration a best non-tabu or tabu but aspired neighbouring candidate solution from the restricted 1-exchange neighbourhood. If a 1-exchange move puts vertex v from colour class C_i into C_j, it is forbidden to re-assign vertex v to C_i in the next tt steps; the tabu status of a neighbouring solution is overruled if it would improve over the best candidate solution found so far (aspiration criterion). If more than one move produces the same effect on the evaluation function, one of those moves is selected uniformly at random. The tabu list length in $\mathsf{TS_{1\text{-}ex}}$ is set to $tt = random(A) + \delta \cdot |V^C|$, where $random(A)$ is an integer uniformly chosen from $\{0, \ldots, A\}$ and δ and A are parameters. Since tt depends on $|V^C|$, the tabu list length varies dynamically with the evaluation function value.

Tabu search with very large scale neighbourhood. In this algorithm, a neighbourhood obtained from the composition of the path exchange and cyclic exchange neighbourhoods is used, referred to as the *cyclic and path exchange neighbourhood*. The embedding of this neighbourhood into a tabu search algorithm analogous to $\mathsf{TS_{1\text{-}ex}}$ results in an algorithm called $\mathsf{TS_{VLSN}}$ [11]. Several variants of $\mathsf{TS_{VLSN}}$ have been studied and the best performing one of these first selects the best non-tabu move in the restricted 1-exchange neighbourhood as in $\mathsf{TS_{1\text{-}ex}}$. If this move is a plateau-move, (i.e. the best neighbouring solution has the same evaluation function value as the current one) then the cyclic and path exchange neighbourhood is searched. In all other cases the 1-exchange move is applied and the tabu list is updated. The tabu mechanism is applied to the search for cyclic and path exchanges by discarding any neighbouring candidate solution that involves the reassignment of a vertex to some colour class that is currently tabu. The tabu list is updated by considering the path or the cyclic exchange as a composition of 1-exchanges and the tabu duration tt for a specific vertex–colour class pair (v, i) is chosen using the rule of $\mathsf{TS_{1\text{-}ex}}$. Yet, contrarily to $\mathsf{TS_{1\text{-}ex}}$, $\mathsf{TS_{VLSN}}$ does *not* use an aspiration criterion.

Min-Conflicts heuristics. One of the most effective extensions of the basic min-conflicts heuristic is a tabu search variant of it ($\mathsf{MC\text{-}TS_{1\text{-}ex}}$) [56]. It uses the same two-stage selection process as the min-conflicts heuristic, which was described in the previous section, but in the second stage it only allows to move the vertex into a colour class that is not tabu, analogously to $\mathsf{TS_{1\text{-}ex}}$. If all colour classes are tabu, one is chosen randomly. One advantage of this neighbourhood examination strategy is that it does not require the usage of sophisticated caching and updating schemes as required, for example, by $\mathsf{TS_{1\text{-}ex}}$; hence it allows for an easier implementation. In addition, the chances of cycling are reduced due to the random choices especially in the first stage of the selection process, allowing for shorter tabu lists.

Guided local search. Guided local search (GLS) is an SLS method that modifies the evaluation function in order to escape from local optima [57]. An application of GLS to the GCP was proposed in [11]. In this algorithm, GLS uses an augmented evaluation function g' defined as

$$g'(C) = g(C) + \lambda \cdot \sum_{i=1}^{|E|} w_i \cdot I_i(C),$$

where $g(C)$ is the usual evaluation function, λ a parameter that determines the influence of the penalties on the augmented cost function, w_i the penalty weight associated to edge i, and $I_i(C)$ an indicator function that takes the value 1 if the end points of edge i are in conflict in C and 0 otherwise. The penalties are initialised to 0 and are updated each time an iterative improvement algorithm reaches a local optimum of g'. The modification of the penalty weights is done by first computing a utility u_i for each violated edge, $u_i = I_i(s)/1 + w_i$, and then incrementing the penalties of all edges with maximal utility by one. The underlying local search is a best-improvement algorithm in the restricted 1-exchange neighbourhood. Once a local optimum is reached, the search continues for a maximum number of sw plateau moves before the evaluation function g' is updated.

Iterated local search. $\mathsf{TS_{1\text{-}ex}}$ can be used as a local search inside hybrid SLS methods like iterated local search (ILS) [46]. In the ILS algorithm presented in [12], $\mathsf{TS_{1\text{-}ex}}$ is run until the best solution found does not change for l_{LS} iterations. A perturbation is then applied to the best colouring found so far and $\mathsf{TS_{1\text{-}ex}}$

is run again. In the perturbation, a number k_r, $k_r < k$, of colour classes is randomly chosen and the colour class membership of all vertices in those colour classes is changed. The ROS heuristic bounded by k and with the further strong constraint of avoiding the re-insertion of a vertex into its previous colour class is used to accomplish this task. The tabu list of $\mathsf{TS}_{\text{1-ex}}$ is emptied before applying the perturbation, while the exchanges caused by the perturbation are inserted in the tabu list.

Other approaches that are based on the same or similar SLS methods have been studied: a predecessor of the ILS algorithm described above is presented in [54], an ILS algorithm that uses a permutation of the colour classes in subgraphs as a perturbation is given in [29]. Similar in spirit is also the iterated greedy solution reconstruction [17].

Evolutionary algorithms. The first evolutionary algorithm (EA) for the GCP is reported in [18]. The most successful EAs are hybrid methods that use $\mathsf{TS}_{\text{1-ex}}$ to improve candidate solutions [15, 22, 25, 26, 48]. Among them, the best results so far have been reported for the hybrid evolutionary algorithm (HEA) [26]. HEA starts with a population P of candidate solutions, which is initialised by using the DSATUR construction heuristic restricted to k colours, and then iteratively generates new candidate solutions by first re-combining two members of the current population that are improved by local search. For the recombination, the greedy partition crossover (GPX) is used [26]. Starting with two candidate partitionings (parents) $C^1 = \{C_1^1, \ldots, C_k^1\}$ and $C^2 = \{C_1^2, \ldots, C_k^2\}$, GPX generates a candidate solution (offspring) by alternately selecting colour classes of each parent. At step i of the crossover operator, $i = 1, \ldots, k$, GPX chooses a colour class with maximal cardinality from parent C^1 (if i is odd) or from parent C^2 (if i is even). This colour class will become colour class C_i of the offspring. Once C_i is chosen, the vertices that belong to it are removed from both parents. The vertices that remain in C^1 and C^2 after step k are added to a colour class of the child, which for each vertex is chosen uniformly at random. The new candidate partitioning returned by GPX is then improved by $\mathsf{TS}_{\text{1-ex}}$, run for l_{LS} iterations, and it is inserted in the population P replacing the worse parent. The population is re-initialised if the average distance between colourings in the population falls below a threshold of 20. Responsible for the high performance reported for the algorithms appears to be mainly the GPX crossover operator [29].

The adaptive search algorithm of Galinier, Hertz and Zuffrey [27] also makes use of the GPX crossover, but it does not further improve on the performance of HEA. Another evolutionary algorithm was proposed in [23] and a scatter search algorithm was proposed in [33]; however, none of these algorithms appears to reach the performance of HEA.

Other methods. A greedy randomised adaptive search procedure was proposed in [43]. It uses a randomisation of RLF for the candidate solution construction and an iterative improvement algorithm in the 1-exchange neighbourhood. The reported results for low degree graphs appear to be good. Another SLS method, ant colony optimisation (ACO), has also been applied to the graph colouring problem in [14] (see Chapter R-24 for more details on ACO). In that approach several ways of defining the heuristic information were studied; the computational results appear to be worse than state-of-the-art, however, no local search was used to improve candidate solutions.

6.2 Strategy 2: *k* fixed, partial colourings

Distributed coloration neighbourhood search. The distributed coloration neighbourhood algorithm proposed by Morgenstern [52] can be seen as an ILS algorithm that uses a simulated annealing (SA) algorithm for the local search. The SA algorithm is based on the i-swap neighbourhood and is run for I iterations or until a certain solution quality threshold is passed. Upon termination of the SA algorithm, a perturbation is applied that is defined by a random *s-chain exchange* that moves from the current colouring configuration to a new one with the same solution quality. An s-chain exchange can be seen as a generalisation of Kempe chain exchange. It is defined through a vertex v and an ordered sequence of non-empty colour classes $C_1, \ldots C_s$, where we have that $v \in C_1$, all colour classes are distinct, and $s \leq k$. From this sequence, a directed graph with vertex set $V' = C_1 \cup \ldots \cup C_s$ and arc set $A = \{(u, w) \mid (u, w) \in E, u \in C_i \text{ and } w \in C_{i+1}\}$ is derived (we use the convention that $C_{s+1} = C_1$). In an s-chain, each vertex that is reachable from v in the digraph (V', A) is moved from colour class C_i to C_{i+1}. Note that an s-chain, where all vertices in V' are reachable, would simply correspond to a relabelling of the colour classes and, hence, would result in the very same

partitioning; therefore, such a *total* s-chain is to be avoided and the neighbourhood is restricted to non-total s-chains. Different ways of combining these two steps and including them into a distributed computational environment were studied in [52].

More recently, a tabu search algorithm based on the i-swap neighbourhood was proposed [5]. The tabu criterion in this algorithm forbids adding into C_i vertices adjacent to a vertex v that was moved into a colour class C_i. This interdiction acts for the tt iterations successive to the move of v.

6.3 Strategy 3: k variable, complete colouring

Simulated annealing with Kempe chain neighbourhood. Simulated annealing was among the first SLS methods applied to the GCP [8, 37]. A comprehensive study of three simulated annealing algorithms was presented in [37]; among these three variants, two work with the number of colours k variable. The more promising one of the two allows only feasible colourings and uses the Kempe chain neighbourhood ($\mathsf{SA}_{\mathsf{Kempe}}$), while the other allows infeasible colourings and uses the 1-exchange neighbourhood. $\mathsf{SA}_{\mathsf{Kempe}}$ uses the evaluation function $g(C) = -\sum_{i=1}^{k}(|C_i|)^2$ and starts from an initial partitioning generated by the random order sequential heuristic. At each iteration of $\mathsf{SA}_{\mathsf{Kempe}}$, a neighbouring solution is generated in three steps; firstly, a non-empty colour class C_i, a vertex $v \in C_i$, and a non-empty colour class C_j are chosen uniformly at random but avoiding that C_i and C_j form a full Kempe chain, which would result simply in a re-labelling of the colour classes; secondly, the Kempe chain K_{ij} of colour classes C_i and C_j that contains vertex v is determined; thirdly, the Kempe chain exchange is applied. The generated neighbouring candidate solution $C'$is always accepted if it improves over the current candidate solution C and otherwise it is accepted with a probability of $\exp((g(C) - g(C'))/T)$, where T is a parameter called temperature. The parameter T is modified according to a rather standard cooling schedule [37].

7 Experimental comparison of SLS algorithms

In this section, we give numerical results for the SLS algorithms that were described in detail in Section 6. For this purpose, we use the challenging benchmark instances described in Section 2, that is, those that are not recognised as easy; the benchmark instances were not treated by the preprocessing rules given in Section 3. We compare $\mathsf{TS}_{\mathsf{1\text{-}ex}}$, $\mathsf{TS}_{\mathsf{VLSN}}$, $\mathsf{MC\text{-}TS}_{\mathsf{1\text{-}ex}}$, ILS, GLS, HEA, $\mathsf{SA}_{\mathsf{Kempe}}$, and XRLF [37], an extension of RLF. All algorithms were implemented under the same environment, sharing the same data structures as much as reasonable. We used a specific experimental set-up considering the GCP in its optimisation version: each algorithm started from the number of colours k^{RLF} returned by the RLF heuristic. When a feasible k-colouring is found, a new colouring with $k-1$ colours is created by uncolouring the vertices assigned to one selected colour and re-colouring each of the vertices by randomly choosing any of the remaining colours. Finally, all algorithms were allowed the same maximum computation time t_{max}; after preliminary experiments, we decided to use $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ as the reference algorithm and to set t_{max} to the average time $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ needs to perform $I_{max} = 10^4 \times |V|$ iterations (averages are taken across 10 trials per instance). For each of the algorithms, we tried to link parameter settings to instance features where possible and set the remaining ones to some constant value that resulted in good performance across the whole benchmark set. Thus, the results presented below give rather an indication of the algorithms' robustness than necessarily their true peak performance.

Each of the algorithms was run 10 times on each instance. For each trial we measured the minimal number of colours found by the algorithm and we analysed the resulting data using rank-based statistical methods. In particular, the measured results on each instance are transformed into a rank value in $1, \ldots, 80$ (we compare eight algorithms and each algorithm is run ten times) and then we aggregated the ranks within the instance classes described in Section 2. Note that applying the statistical tests to each of the instance classes separately avoids the incorrect bias towards problem classes that comprise more instances and it may help in distinguishing the particular strength or weakness of the algorithms for the various instance classes.

In Figure 1.3, we report for each instance class the analysis produced by the non-parametric rank-based Friedman test for an all-pairwise comparison [13]. The graphs are obtained by attaching error bars to a scatter plot of the estimated average rank versus algorithm labels. The length of the bars is adjusted so that the average rank of a pair of algorithms can be inferred to be different with statistical significance at the

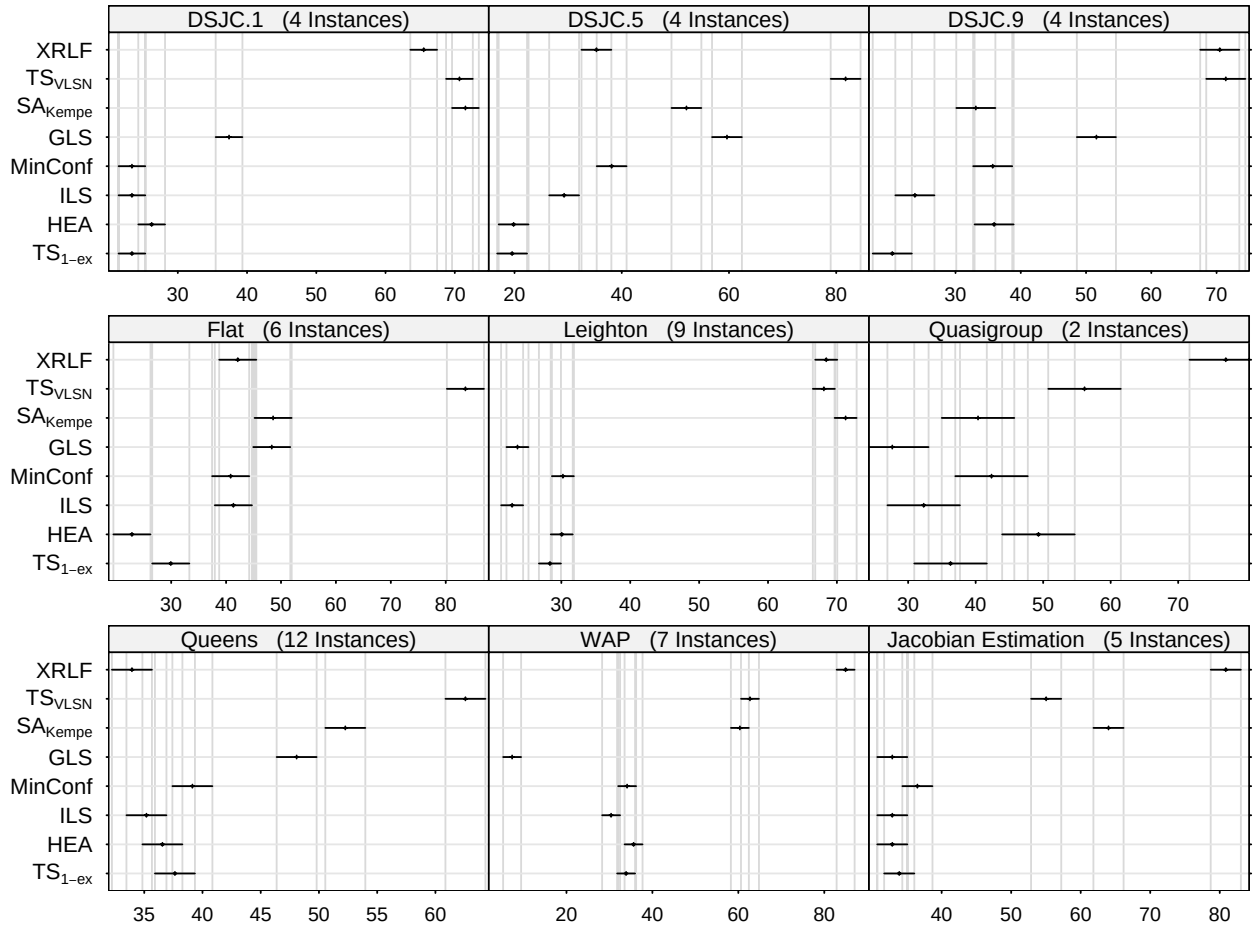

Figure 1.3: All-pairwise comparisons through confidence intervals for the SLS algorithms discussed in the text. The x-axis gives the average rank for the SLS algorithms.

level of $\alpha = 0.05$ if their bars do not overlap. Numerical results on the performance of the algorithms are given in Table 1.1.

From these results we can draw the following conclusions. Most importantly, there are strong differences in the relative order of the algorithms among the various instance classes and, hence, it is not possible to declare any single algorithm to be the best performing one. On the uniform random graphs $\mathsf{TS}_{1\text{-}ex}$, ILS, and HEA are the most competitive algorithms. HEA is the significantly best performing algorithm on the Flat graphs, while on the Leighton graphs ILS and GLS are the best ones. The largest variation in the relative order of the algorithms appears to be due to GLS; GLS is best or among the best algorithms for Leighton, Jacobian estimation, and WAP graphs, but on the other classes of graphs its performance is significantly worse than, for example, that of $\mathsf{TS}_{1\text{-}ex}$. Two further results are interesting. Firstly, the exploration of large neighbourhoods in TS_{VLSN} does not pay off; in fact, it is among the worst performing algorithms. Further analysis showed that this is mainly due to the higher computational cost per search step [11]. Secondly, on most instance classes XRLF performs rather poorly and it is among the best algorithms only on the Queens graphs. This contradicts somehow the reputation it gained, which, however, is mainly due to its very good performance on large uniform random graphs with edge density 0.5. Finally, note that the performance of HEA is worse than reported in [26]. We verified that this difference is mainly due to our experimental setup

Instance	Bench.	time	$TS_{1\text{-ex}}$			HEA			ILS			MC–$TS_{1\text{-ex}}$			GLS			SA_{Kempe}			TS_{VLSN}			XRLF		
	$(\chi, \hat{\chi}^{best})$	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.
DSJC125.1	(–,5)	10	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	6	6	0	5	6	0	5	6	0
DSJC250.1	(–,8)	30	8	8	0.1	8	8	0	8	8	0.2	8	8	0.3	8	9	0	9	9	0.2	9	9	0	9	9	24.4
DSJC500.1	(–,12)	37	13	13	0.1	13	13	0.1	13	13	0.1	13	13	0.2	13	13	0.2	14	14	1.6	14	14	4.5	14	14	29.9
DSJC1000.1	(–,20)	174	21	21	2.8	21	21	164.3	21	21	5.9	21	21	10.4	21	22	0.8	23	23	46.7	23	23	90.5	22	22	169.9
DSJC125.5	(–,17)	14	17	17	0.5	17	17	1.3	17	17	1.6	17	17	6.6	18	18	0	18	18	0.4	18	19	2.1	18	18	2.5
DSJC250.5	(–,22)	47	28	28	22.3	28	28	30.8	28	28	33.6	29	29	2.8	29	30	0.9	29	30	2.7	32	32	6.2	29	30	5.4
DSJC500.5	(–,48)	168	49	50	35	50	50	100.3	50	50	105.8	50	51	20.4	52	52	81.4	51	51	47.3	55	55	138.5	50	50	123
DSJC1000.5	(–,83)	1102	89	90	309.7	89	90	962.5	90	91	303.5	90	91	496.9	93	93	546.3	90	91	409.7	97	98	981.1	86	86	514.8
DSJC125.9	(–,20)	12	44	44	0.1	44	44	0.1	44	44	0.1	44	44	0.2	44	44	0.3	44	44	0.3	44	44	9.9	44	45	3.4
DSJC250.9	(–,72)	90	72	72	3.8	72	72	28.2	72	72	5.6	72	72	29.7	72	73	6.3	72	73	26.6	74	74	39	75	75	12
DSJC500.9	(–,126)	398	127	127	234.4	128	129	189.8	127	128	83.3	128	129	227	129	130	154	127	128	377.2	134	135	340.3	132	132	204.4
DSJC1000.9	(–,224)	2693	226	227	1983.5	230	232	1869.2	227	228	2245	230	230	2382.1	233	234	1621.1	226	229	2401.4	245	247	2143.7	232	233	125.4
flat300_20_0	(20,20)	113	20	20	0.3	20	20	0.4	20	20	0.4	20	20	0.6	20	20	0.6	20	20	1.1	33	34	76.9	20	20	2.9
flat300_26_0	(26,26)	89	26	26	5.5	26	26	16.1	26	26	16.6	26	26	19.8	29	33	4.3	33	33	4.3	35	35	53.5	33	34	2.9
flat300_28_0	(28,31)	61	31	32	3.4	31	32	54.6	31	32	3.3	31	32	7.8	33	33	7.2	33	33	5.1	35	36	17.3	33	34	2.9
flat1000_50_0	(50,50)	1076	85	86	957.4	50	78	1004.9	88	88	729.5	87	88	713.3	50	50	636.3	86	88	470.3	95	96	939.4	84	86	359.3
flat1000_60_0	(60,60)	1119	88	89	245.2	87	88	918.5	89	90	128.2	89	90	372.3	90	91	719.2	88	89	1014.6	96	97	624.5	87	87	235.4
flat1000_76_0	(76,83)	1147	88	89	618.1	88	89	957.6	89	90	188.7	90	90	712	92	92	605	89	90	399	96	97	869	87	87	306
le450_5a	(5,5)	230	5	5	0.1	5	5	0.1	5	5	0.1	5	5	0.9	5	5	0.2	5	7	0	6	7	0	6	7	47.4
le450_5b	(5,5)	232	5	5	0.3	5	5	0.5	5	5	0.6	5	5	0.6	5	5	0.3	6	7	0	6	6	59.6	7	7	37.1
le450_5d	(5,5)	191	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	6	16.4
le450_15a	(15,15)	68	15	15	0.2	15	15	3.4	15	15	0.1	15	15	15	15	15	2.2	16	16	0	16	16	0	16	17	9.3
le450_15b	(15,15)	76	15	15	0.1	15	15	0.2	15	15	0.1	15	15	5.8	15	15	0.3	16	16	0	16	16	0	16	16	18.9
le450_15c	(15,15)	45	16	16	13.5	15	15	19.5	15	15	19.1	15	16	8	15	15	5.9	23	23	0	23	23	0	19	21	216.4
le450_15d	(15,15)	42	16	16	21.7	15	16	13	15	15	20.3	15	16	7.1	15	15	7.8	22	23	0	22	23	0	20	21	189.6
le450_25c	(25,26)	56	26	26	0.7	26	27	0	26	26	2	26	27	0.1	26	26	18	27	28	0	27	28	0	27	28	32.3
le450_25d	(25,26)	59	26	26	0.5	26	27	0	26	26	0.8	26	27	0.2	26	26	4.7	28	28	0	27	28	0	27	27	38.1
latin_square_10	(–,99)	1242	103	104	617.8	106	107	889.4	103	104	510.4	104	105	458.4	102	103	214.9	101	102	369.9	111	114	798.4	117	118	970.7
qg.order100	(100,100)	12102	100	100	17.9	100	100	18.5	100	100	18.3	100	100	19.9	100	100	36.6	100	101	14.8	100	100	875.1	100	101	3971.9
queen6_6	(7,7)	2	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	1.5	7	7	0
queen7_7	(7,7)	4	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	2.1	7	7	0
queen8_12	(12,–)	7	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0.1	12	12	0.4	12	12	0.9
queen8_8	(9,9)	5	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0.1	9	10	0	9	9	0.2
queen9_9	(10,10)	6	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0.1	10	11	0	10	10	0.4
queen10_10	(11,11)	10	11	11	0.1	11	11	0.1	11	11	0	11	11	0.1	11	11	0.8	11	12	0.1	12	12	0.1	11	11	0.9
queen11_11	(12,12)	14	12	12	0.1	12	12	0.2	12	12	0.2	12	12	0.2	12	13	0	12	13	0.1	13	13	0.4	12	13	2
queen12_12	(13,14)	28	13	13	0.1	13	14	0.9	13	14	1.3	13	14	13.2	13	15	0	14	14	0.2	14	15	1.1	13	14	21.3
queen13_13	(14,–)	22	14	14	2.9	14	14	2.3	14	14	20	15	16	0	15	16	0	15	15	0.3	15	15	2.7	14	14	32.2
queen14_14	(15,17)	24	15	16	0	16	17	0	16	16	23.9	16	17	0	17	17	0	17	17	0.5	16	16	5.7	15	15	23.9
queen15_15	(16,18)	24	18	18	0	18	18	0	18	18	0	18	18	0	18	18	0	17	17	0.8	17	17	5.7	16	17	23.9
queen16_16	(–,–)	24	18	18	0	18	18	0	18	18	0	18	18	0	18	18	0	17	18	1.2	18	18	19.5	17	17	33.4
wap01a	(–,–)	4172	43	44	1.2	43	44	1.6	43	44	1.5	42	42	217.1	42	42	55	44	45	107.8	44	46	30.8	47	48	131.6
wap02a	(40,–)	318	42	43	0.7	42	43	0.8	42	42	251.6	41	42	4.9	41	41	159.9	43	43	97.8	43	44	0.5	46	47	150.8
wap03a	(–,–)	1395	42	43	3.8	46	47	4.5	46	46	365.2	45	47	5.8	44	44	782	46	47	198.9	47	48	339.1	50	51	884.3
wap04a	(40,–)	2125	44	44	170	45	45	2.8	44	44	484.3	43	44	31.2	43	43	833.6	45	46	1.6	45	46	1.8	47	49	1073.5
wap06a	(–,–)	138	41	42	0.5	42	42	0.7	42	42	0.5	42	43	0.2	40	41	8.1	42	44	0.2	42	43	4.7	44	44	24.3
wap07a	(–,–)	341	43	44	0.7	43	43	1.9	43	44	0.7	43	43	30.9	42	42	215.2	44	45	9.1	44	45	39	47	47	80.7
wap08a	(40,–)	373	42	43	10.3	42	43	1.4	43	43	56.1	42	44	0.7	42	42	41.4	45	45	0.4	44	45	0.4	46	47	89.4
abb313GPIA	(9,10)	328	9	9	4.1	9	9	26.4	9	9	0.9	9	9	30.7	9	9	1.1	11	11	0	11	11	0.1	12	13	36.4
ash331GPIA	(4,4)	200	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	5	5	25.8
ash608GPIA	(4,4)	633	4	4	0.1	4	4	0.2	4	4	0.1	4	4	0.2	4	4	0.1	4	5	0	4	4	409.3	5	5	27.4
ash958GPIA	(5,4)	1627	4	4	0.5	4	4	0.5	4	4	0.6	4	4	0.9	4	4	0.4	5	5	0	4	5	0	5	5	121.5
will199GPIA	(7,7)	31	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	8	43.9

Table 1.1: Numerical results on the DIMACS graph colouring instances.

and the usage of a single parameter setting; in [26], HEA was tuned for each specific graph and even the value of k when solving the k-colouring problem. When we fine-tuned our implementation of HEA, we could roughly match the results presented in [26]. Across all the instances, the very good performance of $\mathsf{TS}_{1\text{-ex}}$ is most noteworthy, because it is also one of the algorithms that are among the most easy ones to implement.

8 Summary

This chapter gives an overview of the main SLS algorithms described in the literature dedicated to the graph colouring problem. Most of these SLS algorithms follow the strategy of keeping the number of colours fixed to a value k and trying to minimise the number of conflicts. The optimisation of the graph colouring problem is then tackled by solving a series of k-colouring problems. Among the algorithms that follow this strategy, a simple tabu search algorithm based on the restricted 1-exchange neighbourhood is a very robust and fast approach that achieves competitive results for many instance classes. Other SLS algorithms either show better performance only on a few instance classes (for example, GLS on the WAP instances or Jacobian estimation graphs) or when very high computation times are available (for example, HEA when appropriately tuned). On most classes of graphs, SLS algorithms also outperform exact algorithms for the GCP. However, exact algorithms appear to be competitive or even preferable, if the chromatic number is equal or very close to the clique number, the size of the largest clique, of a graph [11].

A promising direction for future research on the graph colouring problem appears to be the integration of exact and SLS algorithms, given that they show particular advantages for different instance classes. Another challenge for research on SLS algorithms for the graph colouring problem is to get a better understanding of the performance of these algorithms in dependence of instance features. Insights into this relationship may help to increase the robustness of the algorithms across the various instance classes and may finally lead to new developments and possibly even better performing algorithms.

Acknowledgments

This work was supported in part by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106 and by a European Community Marie Curie Fellowship, contract HPMF-CT-2001. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. Irina Dumitrescu acknowledges support of the Canada Research Chair in distribution management, HEC Montréal and Thomas Stützle support of the Belgian FNRS, of which he is a research associate.

References

[1] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. To appear in *Annals of Mathematics*, 2005.

[2] M. Allen, G. Kumaran, and T. Liu. A combined algorithm for graph-coloring in register allocation. In Johnson et al. [38], pages 100–111.

[3] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[4] N. Barnier and P. Brisset. Graph coloring for air traffic flow management. In *Proc. of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 133–147, Le Croisic, France, March 2002.

[5] I. Blöchliger and N. Zufferey. A reactive tabu search using partial solutions for the graph coloring problem. Technical Report 04/03, Ecole Ploytechnique Fédérale de Lausanne, Recherche Opérationelle Sud-Est, Lausanne, Switzerland, 2004.

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

[7] M. Caramia and P. Dell'Olmo. Bounding vertex coloring by truncated *multistage* branch and bound. *Networks*, 44(4):231–242, 2004.

[8] M. Chams, A. Hertz, and D. De Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[9] A. Chand. A constraint based generic model for representing complete university timetabling data. In E. K. Burke and M. Trick, editors, *Proceedings of PATAT'05*, pages 125–150, Pittsburgh, USA, 2004.

[10] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI'91*, pages 331–337. Morgan Kaufmann Publishers, USA, 1991.

[11] M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimization Problems*. PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, 2005.

[12] M. Chiarandini and T. Stützle. An application of iterated local search to the graph coloring problem. In Johnson et al. [38], pages 112–125.

[13] W.J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, USA, third edition, 1999.

[14] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48(3):295–305, 1997.

[15] D. Costa, A. Hertz, and O. Dubois. Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105–128, 1995.

[16] J. Culberson, A. Beacham, and D. Papp. Hiding our colors. In *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, pages 31–42, Cassis, France, September 1995.

[17] J. C. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In Johnson and Trick [39], pages 245–284.

[18] L. Davis. Order-based genetic algorithms and the graph coloring problem. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 72–90. Van Nostrand Reinhold, USA, 1991.

[19] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

[20] D. de Werra. Heuristics for graph coloring. *Computing Supplement*, 7:191–208, 1990.

[21] I. Mendez Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. In Johnson et al. [38], pages 55–62.

[22] R. Dorne and J. K. Hao. A new genetic local search algorithm for graph coloring. In A. E. Eiben et al., editors, *Proceedings of PPSN-V*, volume 1498 of *LNCS*, pages 745–754. Springer Verlag, Germany, 1998.

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

[24] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Science*, 57(2):187–199, 1998.

[25] C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–464, 1996.

[26] P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[27] P. Galinier, A. Hertz, and N. Zuffrey. Adaptive memory algorithms for graph colouring. In Johnson et al. [38], pages 75–82.

[28] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.

[29] C. A. Glass and A. Prügel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3):324–330, 2005.

[30] C. Gomes and D. Shmoys. Completing quasigroups or latin squares: A structured graph coloring problem. In Johnson et al. [38], pages 22–39.

[31] J.L. González-Velarde and M. Laguna. Tabu search with simple ejection chains for coloring graphs. *Annals of Operations Research*, 117(1-4):165–174, 2002.

[32] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993.

[33] J.-P. Hamiez and J.-K. Hao. Scatter search for graph coloring. In P. Collet et al., editors, *Artificial Evolution*, volume 2310 of *LNCS*, pages 168–179. Springer Verlag, Germany, 2001.

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

[35] S. Hossain and T. Steihaug. Graph coloring in the estimation of mathematical derivatives. In Johnson et al. [38], pages 9–16.

[36] T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, USA, 1994.

[37] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation: Part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[38] D. S. Johnson, A. Mehrotra, and M. Trick, editors. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, Ithaca, New York, USA, 2002.

[39] David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, USA, 1996.

[40] A. Johri and D.W. Matula. Probabilistic bounds and heuristic algorihms for coloring large random graphs. Technical Report 82-CSE-6, Southern Methodist University, Dallas, Texas, USA, 1982.

[41] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, USA, 1972.

[42] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM press, USA, 1993.

[43] M. Laguna and R. Martí. A GRASP for coloring sparse graphs. *Computational Optimization and Applications*, 19(2):165–178, 2001.

[44] 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.

[45] G. Lewandowski and A. Condon. Experiments with parallel graph coloring heuristics and applications of graph coloring. In Johnson and Trick [39], pages 309–334.

[46] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, USA, 2002.

[47] T. Luczak. The chromatic number of random graphs. *Combinatorica*, 11(1):45–54, 1991.

[48] A. Marino and R. I. Damper. Breaking the symmetry of the graph colouring problem with genetic algorithms. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 GECCO Conference*, pages 240–245, Las Vegas, Nevada, USA, 8 2000.

[49] A. Mehrotra and M. A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[50] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1–3):161–205, 1992.

[51] K. Mizuno and S. Nishihara. Toward ordered generation of exceptionally hard instances for graph 3-colorability. In Johnson et al. [38], pages 1–8.

[52] C. Morgenstern. Distributed coloration neighbourhood search. In Johnson and Trick [39], pages 335–357.

[53] C. Morgenstern and H. Shapiro. Coloration neighbourhood structures for general graph coloring. In *First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 226–235. Society for Industrial and Applied Mathematics, USA, 1990.

[54] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni et al., editors, *Applications of Evolutionary Computing*, volume 2279 of *LNCS*, pages 122–131. Springer Verlag, Germany, 2002.

[55] V. T. Paschos. Polynomial approximation and graph-coloring. *Computing*, 70(1):41–86, 2003.

[56] Thomas Stützle. *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications*, volume 220 of *DISKI*. Infix, Sankt Augustin, Germany, 1999.

[57] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. PhD thesis, University of Essex, Department of Computer Science, Colchester, UK, 1997.

[58] A. Zymolka, A. M. C. A. Koster, and R. Wessäly. Transparent optical network design with sparse wavelength conversion. In *Proceedings of the 7th IFIP Working Conference on Optical Network Design & Modelling*, pages 61–80, Budapest, Hungary, 2003.