An Experimental Investigation of Iterated Local Search for Coloring Graphs

Luis Paquete and Thomas Stützle

Darmstadt University of Technology, Computer Science Department, Intellectics Group
Alexanderstr. 10, 64283 Darmstadt

Abstract. Graph coloring is a well known problem from graph theory that, when attacking it with local search algorithms, is typically treated as a series of constraint satisfaction problems: for a given number of colors k one has to find a feasible coloring; once such a coloring is found, the number of colors is decreased and the local search starts again. Here we explore the application of Iterated Local Search on the graph coloring problem. Iterated Local Search is a simple and powerful metaheuristic that has shown very good results for a variety of optimization problems. In our research we investigated several perturbation schemes and present computational results on a widely used set of benchmarks problems, a sub-set of those available from the DIMACS benchmark suite. Our results suggest that Iterated Local Search is particularly promising on hard, structured graphs.

1 Introduction

The Graph Coloring Problem (GCP) is a well known combinatorial problem defined as follows: Given a directed graph $G = (V, E)$, where V is the set of $|V| = n$ vertices and $E \subseteq V \times V$ is the set of edges, and an integer k (number of colors), find a mapping $\Psi : V \mapsto 1, 2, ..., k$ such that for each $[u, v] \in E$ we have $\Psi(v) \neq \Psi(u)$. In fact, this problem statement corresponds to the decision version, where we try to find a solution, satisfying an additional constraint on the value of the objective function. In the optimization counterpart, the GCP can be defined as to find the minimum k, that is to find the chromatic number χ_G of G.

The GCP is an interesting problem for theory and practice. In fact, several classes of real-life problems such as examination timetabling [1] and frequency assignment [2] can be modelled as GCPs. Yet, it cannot be expected that an algorithm can find, in polynomial time, a solution to any GCP instance, because it is $\mathcal{NP}$-hard [3]. In fact, exact algorithms can solve only small size instances [4]. For larger instances approximate algorithms have to be used and a large number of such algorithms has been proposed [5–10].

This article explores the application of Iterated Local Search (ILS) [11] to the GCP. ILS consists in the iterative application of a local search procedure to starting solutions that are obtained by the previous local optimum through a solution perturbation. So far, ILS has only been applied to optimization problems. In this article we apply ILS to the decision variant of the GCP. This is no limitation, because the optimization variant can be stated as a sequence of constraint satisfaction problems, where k is being decremented sequentially by one until no admissible mapping exists, meaning that

$\chi_G = k + 1$. In this case, the problem is attacked as a constraint satisfaction problem. The good performance of our ILS especially on structured graphs suggests that it may be worthwhile to also consider applications of ILS to other types of constraint satisfaction problems or to the well known satisfiability problem in propositional logic.

The article is structured as follows. Section 2 presents a review of approximation algorithms for the GCP and Section 3 introduces available benchmark sets. Section 4 introduces ILS and describes some details of the ILS implementation for the GCP. Section 5 presents the experimental results and we conclude in Section 6.

2 Approximate algorithms for graph coloring

Approximate algorithms for the GCP fall into two main classes, construction heuristics and local search algorithms.

Construction heuristics start from an empty solution and successively augment a partial coloring until the full graph is colored. During the solution construction these algorithms maintain feasibility that is, they return a conflict free coloring. Well known construction heuristics are the Brelaz heuristic [5], the Recursive Largest First (RLF) heuristic [10] or iterated, randomized construction heuristics like the Iterated Greedy algorithm [12].

Local search for the GCP starts at some initial, inconsistent color assignment and iteratively moves to neighboring solutions, trying to reduce the number of conflicts. A pair of vertices (i, j) is in conflict, if both are assigned the same color and $(i, j) \in E$. In fact, local search applied to the GCP iteratively tries to *repair* the current color assignment guided by an evaluation function that counts the total number of conflicts. In case a candidate solution with zero conflicts is encountered, this candidate solution corresponds to a feasible coloring of the graph.

When treating the GCP as a decision problem, a commonly used neighborhood is the 1-opt neighborhood that in each step changes the color assignment of exactly one vertex. For searching the 1-opt neighborhood, two different local search architectures exist. The first is based on the *min-conflicts* heuristic (A1): In each local search step, a vertex that is in conflict is chosen at random. To this vertex a color is assigned that minimizes the number of conflicts [13]. The second scheme (A2) examines all pairs of vertices and colors (v_i, j) and performs the move with the maximal reduction of the number of conflicts; if several such moves exist, one is chosen randomly [14, 6]. This latter neighborhood is often further reduced by considering moves that only affect vertices that are currently involved in a conflict (A2'). Note that A2 and A2' architectures are greedier than the min-conflicts architecture, because at each step one among a larger set of candidate moves is choosen.

In the simplest case, local search algorithms accept only improving moves and they terminate in local optima. The most successful technique to avoid this problem is a family of algorithm schemata that is often called metaheuristics. Among the first metaheuristic approaches to the GCP were Simulated Annealing implementations. Simulated Annealing was first applied to the GCP by Chams et al. [15] and was intensively tested by Johnson et al. [9] on random graphs. Among the most widely applied metaheuristics to the GCP are Tabu Search implementations. The first implementations of

Tabu Search were due to Hertz and de Werra [8]. More recently, Hao, Dorne, and Galinier [14, 7] presented the most performing Tabu Search implementations, based on the A2' local search. The solution of the GCP by Evolutionary Algorithms was proposed by Davis [16], who reported several crossover operators combined with several ordering of vertices. Eiben et al. [17] applied an Adaptive Evolutionary Algorithm to the GCP; this algorithm changes periodically the evaluation function to avoid local optima. More recently, Laguna and Martí [18] proposed an application of GRASP to the GCP and presented good results for sparse graphs. Finally, several hybrid approaches were proposed. These typically combine Evolutionary Algorithms with Tabu Search implementations. The first such approach for the GCP was proposed by Fleurent and Ferland [19].

The best computational results so far have been obtained by the Hybrid Evolutionary Algorithm (HEA) by Galinier and Hao [7]. In their algorithm, the initial population is generated by a greedy saturation algorithm. After initialization, two solutions are chosen randomly and a specific crossover operator generates a new solution based on the information of the latter. This operator builds a partial solution by exchanging subsets of the color class sets of the two chosen solutions and fills the unassigned vertices in a random fashion to obtain a complete solution. The new solution is then improved by applying the Tabu Search algorithm for a certain number of L iterations and then re-inserted into the set of solutions. Galinier and Hao have reported very good performance and outperformed the Tabu Search, which previously was reported to be among the most effective local search algorithms for the GCP, on a number of hard benchmark instances.

3 Benchmark Problems

Many different benchmark problems and instance sets for the GCP are available at Joseph Culberson's Graph Coloring Page (http://www.cs.ualberta.ca/~joe/Coloring/) and Michael Trick's Graph Coloring Page (http://mat.gsia.cmu.edu/COLOR/color.html). The most prominent benchmark set is that of the Second DIMACS Implementation Challenge on Cliques, Coloring, and Satisfiability, which is available on the web site http://mat.gsia.cmu.edu/challenge.html. Many of these instances were generated in way such that the chromatic number is known, like the Leighton and the Flat graphs. The Leighton graphs were generated by a procedure that uses the number of vertices, the desired chromatic number, the average vertex degree and a random vector of integers to generate a certain number of cliques. The Flat graphs were generated in a way such that the set of vertices is partitioned into k almost equal sized sets. The number of edges is close to the expected number of edges given a certain probability p and partitioning. A flatness parameter controls the variation of the vertex degree. The DIMACS instances have a flatness parameter equal to 0, which means that the graph is uniform.

Another prominent class of benchmark instances available from the DIMACS site are random graphs, where each edge is included into the graph with a probability p (the chromatic number of these graphs is not known). These graphs were initially proposed by Johnson et al. in their experimental evaluation of Simulated Annealing [9] and were used since then in a large number of studies.

```
procedure Iterated Local Search
    s_0 = GenerateInitialSolution()
    s = LocalSearch(s_0)
    repeat
        s' = Perturbation(s, history)
        s'' = LocalSearch(s')
        s = AcceptanceCriterion(s, s'', history)
    until termination condition met
end
```

Fig. 1. Pseudocode of an iterated local search procedure (ILS)

4 Iterated Local Search for coloring graphs

ILS [11] is based on the simple idea of improving a local search procedure by providing new starting solutions which are obtained from perturbations of the current solution. This perturbation must be sufficiently strong to allow the local search to explore different solutions, but also weak enough to prevent random restart. ILS is appealing both due to its simplicity and due to the very good results obtained, for example, the Traveling Salesman Problem [20], Graph-Partitioning [21], and Scheduling Problems [11].

To apply an ILS algorithm, four components have to be specified. These are a procedure GenerateInitialSolution() that generates an initial solution s_0, a procedure Perturbation, that modifies the current solution s leading to some intermediate solution s', a procedure LocalSearch that returns an improved solution s'', and an AcceptanceCriterion that decides to which solution the next perturbation is applied. An algorithmic scheme for ILS is given in Figure 1.

In principle, any local search algorithm can be used, but the performance of the ILS algorithm with respect to solution quality and computation speed depends strongly on the one chosen. Very often an iterated descent algorithm is taken, but it is also possible to apply more sophisticated local search algorithms like Tabu Search, as done in our case.

The perturbation mechanism should be chosen *strong enough* to allow to leave the current local minimum and to allow the local search to explore different solutions. At the same time, the modification should be *weak enough* to keep enough characteristics of the current local minimum.

The procedure AcceptanceCriterion is used to decide from which solution the search is continued by applying the next pertubation. One important aspect of the acceptance criterion and the perturbation is to introduce a bias between intensification and diversification of the search. Intensification of the search around the best found solution is achieved, for example, by applying the perturbation always to the best found solution and using small perturbations. Diversification is achieved, in the extreme case, by accepting every new solution s'' and applying large perturbations.

4.1 Iterated Local Search operators

In our application of ILS to the GCP we focused mainly on the choice of the local search and the perturbation operator.

Local Search For the local search we considered to apply one based on local search architecture A1 (see Section 2) and one based on A2 and A2', respectively. In addition to a plain iterative improvement local search, we implemented Tabu Search versions for both local search architectures. In fact, after some initial experiments it was found that our ILS with the Tabu Search implementation based on A2' performed significantly better than the other variants. Yet, this improved performance is only possible because of the use of speed-up techniques for the neighborhood evaluation [19]: The implementation uses a two-dimensional table of size $n \cdot k$ where each entry $\gamma(i, j)$ stores the effect on the evaluation function incurred by changing the color of vertex i to color j. Each time a move is performed, only the part of the table that is affected by the move is updated. This table has to be initialised in $O(n^2 \cdot k)$, but each update of the matrix then only takes $O(n \cdot k)$ in the worst case (for sparse graphs even faster); in fact the complexity of each iteration is the same as for implementations of architecture A1. For the setting of the tabu list lenght we followed the scheme in [7]: the length of tabu list was taken as $Random(A) + \alpha \times |n_c|$, where n_c is the set of conflicting vertices.

Perturbation For the perturbation we considered four different possibilities.

P1, **random moves**: Each perturbation consists of assigning to some vertices randomly chosen colors.

P2, **adding edges**: P2 consists of adding a certain number of edges during some number of iterations, leading to a modification of the instance definition. After the perturbation, the additional edges are removed again.

P3, **conflict vertices**: Assignments of randomly chosen colors to conflicting vertices.

P4, **directed diversification**: We perform p_{iter} moves using the Tabu Search procedure with long tabu list settings and mix this strategy with random moves. In fact, at each step a random choice is made whether we execute a random search step (such a step is done with a probability w_p) or a Tabu Search step (with probability 1-w_p).

One particularity of all the perturbations is that they also use the move table from the local search in the perturbation. The reason is that, depending on the length of the perturbation, this can save a significant amount of computation time, because the re-initialisation of the move table is avoided in this case.

Preliminary experiments showed that P4 was the most promising perturbation. An investigation of the tradeoffs regarding the parameter settings for P4 showed that the larger is p_{iter}, the lower should be w_p to avoid that the final solution becomes too close to a random initial solution. Two more details are note-worthy: First, the tabu list is maintained during the perturbation as during the standard Tabu Search procedure. This has the effect that the perturbation moves (including random moves) cannot directly be undone after the perturbation. Second, the perturbation is triggered, if the local search is deemed to be stuck as indicated by the fact that for ls_{iter} iterations the Tabu Search could not find improved solutions.

Initial solution For the initial solution we considered random initial solutions and greedy solutions. Since for the hardest instances we could not observe a significant difference in performance, we used random initial solutions for the following experiments.

Acceptance criteria The Acceptance criteria is a straightforward procedure that uses one of the following rules: accept every new solution or always apply the perturbation to best solution found so far in the search. Some preliminary experiments were done to compare both acceptance criteria, and no significant difference was detected. Thus, we choose random walk which is less expensive from a computational point of view.

5 Experimental results

5.1 Peak performance

In this section we give some performance results of the ILS obtained by a limited parameter tuning effort. Here we report peak performance, that is, the best performance we obtained for the studied parameter settings. Observing peak performance is important, because one hint on the usefulness of a technique is that at least for some instance classes the state-of-the-art performance can be matched or even be improved upon. In fact, many other articles report peak performance after significant parameter tuning efforts like it is the case for the Hybrid Evolutionary Algorithm (HEA) and the Tabu Search presented in [7]. Since in that article the best results so far were reported for several hard benchmark problems, we compare our results to HEA as far as possible. We report results on some Leighton and Flat graphs, which are known to be very hard, and few Random graphs.

In some preliminary experiments, we first identified good parameter settings for the Tabu Search algorithm that is embedded into our ILS. We found that best performance was obtained by the setting of $A = 10$ and α = 3.5 with a A2' architecture. Then, at a next step, the Tabu Search procedure was kept fixed (as a black box) and we performed experiments with different parameter settings for the perturbation. Regarding the perturbation, we found that P4 gave the apparently best performance and we decided to run detailed experiments with all possible combinations of parameters settings governing perturbation P4 taken from $ls_{iter} = \{10|V|, 20|V|, 30|V|\}$, $p_{iter} = \{\frac{|V|}{5}, \frac{|V|}{2}, |V|, 2|V|\}$, $w_p = \{0.25, 0.5, 0.75\}$. This results in a total of 36 experiments.

Table 1 presents the results obtained in 50 runs of the Tabu Search (as said above, these results were obtained after fine-tuning parameters A and α) and the best results found in 25 runs of each combination of parameter settings for the ILS. Each trial was given a maximum of ten million local search iterations. For a given number of colors, the results indicated are the average number of iterations in the successful runs and the fraction of successful runs. The ILS presents an additional column with the best parameter settings. The HEA results are taken from [7].

When comparing the peak performance obtained by ILS with Tabu Search, we observe that ILS obtains complete colorings in less iterations than Tabu Search in almost all instances. The most marked difference in favor of ILS is observed for the instance le450_15c.col which is solved in all 25 runs by ILS, while the Tabu Search in a large fraction of the runs stopped unsuccessfully. However, on the runs on instances DSJC250.5.col and flat300_28_0.col, ILS obtained an inferior number of colorings than Tabu Search. One possible reason could be the efficiency of the Tabu Search, giving no opportunity to ILS to outperform its underlying local search.

Table 1. Experimental results. Given are the instance name, the number of colors allowed, and for each algorithm the average number of iterations in the successful runs and the percentage of successful runs.

		Tabu Search		ILS			HEA	
Instances	k	Average	% succ	Average	% succ	ls_{iter}, p_{iter}, w_p	Average	% succ
DSJC125.5.col	17	244382	100	186185	100	$(20\|V\|, \frac{\|V\|}{5}, 0.25)$	-	-
DSJC250.5.col	28	4179466	88	3550914	76	$(30\|V\|, \|V\|, 0.50)$	490000	90
flat_26_0.col	26	957811	100	649107	100	$(10\|V\|, \frac{\|V\|}{2}, 0.25)$	-	-
flat_28_0.col	31	5375378	26	5354077	20	$(30\|V\|, \frac{\|V\|}{5}, 0.75$	637000	60
le450_15a.col	15	113165	100	95246	100	$(30\|V\|, \frac{\|V\|}{2}, 0.50)$	-	-
le450_15b.col	15	66165	100	63920	100	$(30\|V\|, \frac{\|V\|}{5}, 0.50)$	-	-
le450_15c.col	15	3881149	22	451714	100	$(10\|V\|, 2\|V\|, 0.25)$	194000	60
le450_15d.col	15	6399520	8	2128483	100	$(10\|V\|, 2\|V\|, 0.25)$	-	-
le450_25c.col	26	120720	100	108312	100	$(20\|V\|, \frac{\|V\|}{5}, 0.50)$	800000	100
le450_25d.col	26	108685	100	99785	100	$(30\|V\|, \frac{\|V\|}{5}, 0.25)$	-	-

For some of the hardest instances, as le450_15c.col and le450_25c.col, the ILS was able to obtain colorings in less iterations than HEA. This is remarkable, since HEA was reported the best results among the large number of approximation algorithms for the GCP. However, on the instances DSJC250.5.col and flat_28_0.col, HEA shows better average behavior.

Finally, it should be remarked that, when compared to the Tabu Search results of [7] (not reported here), our Tabu Search was able to find complete coloring with an inferior chromatic number on the instances flat_28_0.col and le450_15c.col, with 31 and 15 colors, respectively. However, on the instance DSJC250.5.col our Tabu Search performed worse.

The peak performance results clearly indicate that the ILS algorithm is able to compete with state-of-art algorithms. Yet, this approach does not use any operator that uses the knowledge of the problem to guide the search, as the crossover operator in [7].

5.2 Perturbation parameters

We analyzed the dependence of the ILS performance from the parameters that determine the perturbation. Tables 2, 3, and 4 show the average results obtained from 25 runs on the instance le450_25c.col and the percentage of successful runs. Each table is grouped by the ls_{iter} values.

The computational results suggest that higher values for ls_{iter} could be a good choice. Additionally, low values for the parameters w_p and p_{iter} seem to help. Hence, a good perturbation strength for this instance is rather different from random restart, which confirms our conjecture that a perturbation, as the one reported here, brings more advantages than a random restart. This observation holds for most instances. One exception is the instance le450_15c.col, for which the results obtained indicate that the perturbations should be disruptive enough to move the search to different regions of the search space.

Table 2. Average values for ILS with $ls_{iter} = 10|V|$ on le450_25c.col

	p_{iter}			
w_p	$\frac{\|V\|}{5}$	$\frac{\|V\|}{2}$	$\|V\|$	$2\|V\|$
0.25	157298 (100%)	254323 (100%)	376345 (100%)	2498161 (100%)
0.50	153709 (100%)	192482 (100%)	278668 (100%)	- (0%)
0.75	131303 (100%)	320049 (100%)	- (0%)	- (0%)

Table 3. Average values for ILS with $ls_{iter} = 20|V|$ on le450_25c.col

	p_{iter}			
w_p	$\frac{\|V\|}{5}$	$\frac{\|V\|}{2}$	$\|V\|$	$2\|V\|$
0.25	117866 (100%)	163488 (100%)	179836 (100%)	709729 (100%)
0.50	108312 (100%)	144848 (100%)	196596 (100%)	872861 (100%)
0.75	127268 (100%)	155794 (100%)	1051493 (100%)	1973531 (100%)

Table 4. Average values for ILS with $ls_{iter} = 30|V|$ on le450_25c.col

	p_{iter}			
w_p	$\frac{\|V\|}{5}$	$\frac{\|V\|}{2}$	$\|V\|$	$2\|V\|$
0.25	116881 (100%)	167439 (100%)	163045 (100%)	300516 (100%)
0.50	165227 (100%)	128923 (100%)	153511 (100%)	597886 (100%)
0.75	130700 (100%)	184779 (100%)	411852 (100%)	737571 (100%)

5.3 Run-time Distributions

Our ILS algorithm makes strong use of randomized decisions during the search and therefore the run-time required to find a coloring is a random variable; this is notable by the fact that the time needed to find such a solution varies between runs of the algorithm. We estimate the distribution the solution times by analyzing empirical run-time distributions (RTDs). RTDs give the empirical probability of finding a solution as a function of the run-time [22, 23]. RTDs can be used to ease the comparison of local search algorithms, characterize the run-time behavior of local search algorithms on specific problem classes and to give valuable hints in which situations local search algorithms can be improved [22].

Here we analyze the run-time behavior of ILS and compare it with Tabu Search. We ran 100 times the Tabu Search and the ILS using the best parameter settings found on the instances le450_15b.col, le450_15c.col. Figures 2 and 3 plot the RTDs of ILS and Tabu Search on these instances.

Observing the RTDs on instance le450_15b.col on the Figure 2 (left), it is possible to conclude that both algorithms perform similarly, as also reported in Table 1. However, the behavior of the two algorithms is different beyond 100000 iterations. The curvature of Tabu Search's RTD indicates worse performance than ILS on long search paths. This different behavior could mean that ILS is capable of getting out of some local optimum in a more successful way than Tabu Search and this may be the main reason for improved performance. Figure 2 (right) also demonstrates the higher performance of ILS on the instances le450_15c.col (the RTDs on le450_15d.col look similar). The

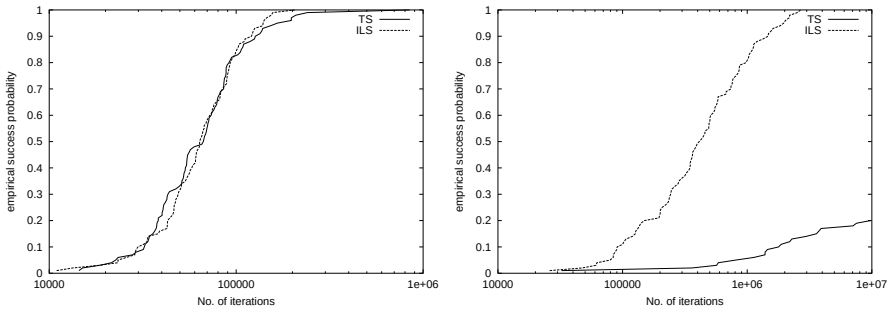

Fig. 2. Run-time distributions of Tabu Search and ILS on instance le450_15b.col (left) and le450_15c.col (right).

RTDs obtained from these two instances prove their hardness, since Tabu Search was not able to reach more than a success rate of 20%.

6 Conclusion

In this paper we have presented an initial study of the application of ILS to the graph coloring problem. When looking at peak performance, some very promising results were obtained compared to state-of-the-art algorithms. There are a number of ways how this research will be extended in the future. First, we will extend this approach to larger instances, random and structured, to verify if the overall behaviour is similar to the one reported here. Second, we will to do a more detailed analysis of the parameter space for the perturbation. Such a study may identify even better parameter settings. Additionally, a refined study using methods from experimental design may help to better understand the interaction between the perturbation parameters. Third, additional ways of perturbing solutions will be tested, because this seems to be the key to further enhance ILS performance for the graph coloring problem. Future work will also include ways of how to automatize the adjustment of the parameters for the instance under solution.

Acknowledgments This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. 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.

References

1. M.W. Carter. A survey of pratical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, 1986.
2. D.J. Castelino, S. Hurley, and N.M. Stephens. A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 63:301–320, 1996.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, USA, 1979.

4. A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal On Computing*, 8(4):344–354, 1996.
5. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
6. C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–464, 1996.
7. P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
8. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.
9. 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.
10. F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 85:489–506, 1979.
11. H.R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, MA, USA, 2002. to appear.
12. J.C. Culberson. Iterated greedy graph coloring and the difficulty landscape. Technical Report 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, June 1992.
13. 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*, 52:161–205, 1992.
14. R. Dorne and J.K. Hao. Tabu search for graph coloring, t-colorings and set t-colorings. In I.H. Osman S. Voss, S. Martello and C. Roucairol, editors, *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer Academic Publishers, Boston, MA, USA, 1999.
15. M. Chams, A. Hertz, and D. De Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32:260–266, 1987.
16. L. Davis. Order-based genetic algorithms and the graph coloring problem. In *Handbook of Genetic Algorithms*, pages 72–90. Van Nostrand Reinhold; New York, 1991.
17. A.E. Eiben, J.K. Hauw, and J.I. Van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4:25–46, 1998.
18. M. Laguna and R. Martí. A GRASP for coloring sparse graphs. *Computational Optimization and Applications*, 19(2):165–178, 2001.
19. C. Fleurent and J. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique and satisfiability. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, pages 619–652. American Mathematical Society, 1996.
20. D.S. Johnson and L.A. McGeoch. The travelling salesman problem: A case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.
21. O. Martin and S. W. Otto. Partitoning of unstructured meshes for load balancing. *Concurrency: Practice and Experience*, 7:303–314, 1995.
22. H.H. Hoos and T. Stützle. Evaluating Las Vegas algorithms — pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann, San Francisco, 1998.
23. H.H. Hoos and T. Stützle. Characterising the behaviour of stochastic local search. *Artificial Intelligence*, 112:213–232, 1999.