An application of Iterated Local Search to Graph Coloring

Marco Chiarandini and Thomas Stützle
Fachgebiet Intellektik, Fachbereich Informatik
Technische Universität Darmstadt, Darmstadt, Germany
{machud,stuetzle}@intellektik.informatik.tu-darmstadt.de

Abstract

Graph coloring is a well known problem from graph theory that, when solving 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. In this article we explore the application of Iterated Local Search to the graph coloring problem. Iterated Local Search is a simple, yet powerful, metaheuristic that has shown very good results for a variety of optimization problems. In our research we investigate different perturbation schemes and present computational results on some hard instances from the DIMACS benchmark suite.

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 look for a solution that satisfies all constraints. In the optimization counterpart, the GCP consist of finding the minimum k satisfying the constraints. Such k is called the chromatic number of G and is denoted by χ_G.

The GCP is an interesting problem for theory and practice. In fact, several classes of real-life problems such as examination timetabling [3] and frequency assignment [4] can be modeled as GCP extensions. Yet, it cannot be expected that an algorithm can find, in polynomial time, a solution to an arbitrary GCP instance, because the GCP is $\mathcal{NP}$-hard [15]. In fact, exact algorithms can solve only small size instances [22]. For larger instances approximate algorithms have to be used and a large number of such algorithms has been proposed [10, 14, 16, 17, 20].

In this work we explore the application of Iterated Local Search (ILS) [21] to the GCP. ILS consists in the iterative application of a local search procedure to initial solutions obtained by modifying some previous local optima. The optimization variant of GCP 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 solved by ILS as a constraint satisfaction problem.

The article is structured as follows. Section 2 presents a review of approximation algorithms for the GCP and show which one we use. Section 3 introduces available benchmark sets. Section 4 presents ILS and describes some details of the ILS implementation for the GCP. Section 5 gives experimental results. Conclusions are presented 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 [2] and the Recursive Largest First (RLF) heuristic [20] or iterated, randomized construction heuristics like the Iterated Greedy algorithm [6].

Local search for the GCP starts with some initial, infeasible color assignment and iteratively moves to neighboring solutions, trying to reduce the number of conflicts until a feasible solution is found or a stopping criterion is met. Two vertices i, j are in conflict, if they are connected by an edge and both are assigned the same color. 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.

Often, in articles, only results for obtaining an a priori determined number of colors are given, that is, the number of colors is fixed and the constraint satisfaction problem solved. The number of colors chosen a priori often corresponds to a very good one. But in real applications, one can not solve the GCP by first guessing a very good coloring and then running the algorithm; rather, one determines a good initial feasible coloring and then tries to reduce the number of colors. In our work we follow this latter approach. In particular, we can distinguish three different phases in which an algorithms from one of the two classes, previously introduced, is applied:

- *Initialization phase*: the exact coloring algorithm implemented by Trick[1] [22] based on DSATUR [2] is used to construct an initial coloring. We stop the algorithm when coloring the graph with a lower number of colors becomes difficult (in the sense of CPU time required). The output of this phase is a feasible coloring.

- *Color number decreasing phase*: once DSATUR or local search (described in the next point) provided a feasible coloring, one color is removed and the vertices that were assigned to the removed color are reassigned to another available color with the *dsat* heuristic which colors first the vertex most constrained, i.e. the vertex with the highest number of different colors used in its neighborhood (ties are broken in favor of the highest vertex degree, that is, the number of incident edges) [2].

- *Local Search phase*: it takes place when the coloring returned by the decreasing phase is not feasible; local search based algorithms are used for finding a feasible coloring. In this sense local search is used for solving the decision version of the problem.

The whole procedure ends when local search is not able to solve feasibility for the given number of colors.

The main aim of this work is to improve the local search phase. In the simplest case, local search algorithms accept only improving moves and they terminate in local optima. Metaheuristics are techniques which are intended to avoid the disadvantages of early termination in local optima. Among the first metaheuristic approaches to the GCP were Simulated Annealing implementations. Simulated Annealing was first applied to the GCP by Chams et al. [5] and was intensively tested by Johnson et al. [17] on random graphs. Among the most widely applied metaheuristics to the GCP are Tabu Search algorithms. The first implementations of Tabu Search are due to Hertz and de Werra [16]. The best peak performance is apparently obtained by the variant proposed by Hao and Dorne [8, 14]. Solving the GCP by Evolutionary

[1]Available via http://mat.gsia.cmu.edu/COLOR/color.html, June 2002

Algorithms was proposed by Davis [7], who reported several crossover operators combined with several ordering of vertices. Eiben et al. [9] applied an Adaptive Evolutionary Algorithm to the GCP; this algorithm changes periodically the evaluation function to avoid local optima. More recently, Laguna and Martí [19] 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 [11, 12] and the currently most performing of these approaches seems to be the one of Galinier and Hao [13]. The first application of Iterated Local Search to the graph coloring is that of Paquete and Stützle [25], for which interesting performance results were obtained, though significant room for further investigation remains.

3 Benchmark Problems

We tested our approaches on some of the benchmark instances proposed for the Symposium at `http://mat.gsia.cmu.edu/COLOR02`. We first divided the instances into two groups: those which are solved to optimum by the DSATUR algorithm in less than 8 seconds on our machine, a Pentium III 700 MHz processor with 256 KB cache and 512 MB RAM, and those for which the DSATUR algorithm requires longer time. We then divided this latter group in three sub-groups: (i) the instances for which the exact optimum is known, (ii) the instances for which the optimum is not known and (iii) the instances which are of large size (more than 1000 vertices). A table with this classification is given in Appendix. For the experimental analysis of our algorithms we took a sample from the first and the second group while in the Appendix results also for instances of the third group are reported. More specifically we used the following graphs:

- *Random graphs.* These are instances from Johnson [17] in which for a given set of vertices the graph is obtained by including each possible edge $[u, v]$ with a probability p for each pair. The chromatic number for these instances is unknown.
- *Leighton graphs.* These are structured graphs 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 [20]. The chromatic number of these instances is known.
- *Queen graphs.* Given an $n \times n$ chess board, a queen graph is a graph with n^2 vertices, each corresponding to a square of the board. Two vertices are connected by an edge if the corresponding squares are in the same row, column or diagonal. The chromatic number is not always known, but since the maximum clique in the graph is no more than n the chromatic number is not less than n.
- *Full Insertion graphs.* These are obtained by a generalization of Mycielski transformation [24] with inserted vertices to increase graph size but not density.
- *Latin square graphs.* These are derived from the problem of completing Latin Squares.

4 Iterated Local Search for coloring graphs

The essence of ILS [1, 26] is to build a biased randomized walk in the space of the local optima (local optima with respect to some local search algorithm). This walk is built by iteratively perturbing a locally optimal solution, next applying a local search algorithm to obtain a new locally optimal solution, and finally using an acceptance criterion for deciding from which of these solutions to continue the search. The perturbation must be sufficiently strong to allow

```
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
```

Figure 1: Pseudocode of an iterated local search procedure.

the local search to effectively escape from local optima and to explore different solutions, but also weak enough to prevent the algorithm from reducing to a simple random restart algorithm, which is known to typically perform poorly. ILS is appealing both for its simplicity and for the very good results it provided, for example, in the Traveling Salesman Problem [18] or Scheduling Problems [21].

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. An iterated descent algorithm is often used, but it is also possible to apply more sophisticated local search algorithms like Tabu Search, as we do in our case.

The perturbation mechanism should be chosen *strong enough* to allow to leave the current local optimum 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 optimum.

The procedure AcceptanceCriterion is used to decide from which solution the search is continued by applying the next perturbation. 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

4.1.1 Local Search

The commonly used neighborhood when treating the GCP as a decision problem 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 are commonly applied.

1. The first one is based on the *min-conflicts* heuristic, which was proposed to solve constraint satisfaction problems of which graph coloring is a particular case. In each local search step, a vertex that is in conflict is chosen at random; then the color that minimizes the number of conflicts [23] is assigned to this vertex [23].

2. The second scheme examines all possible combinations of vertices and colors (i, c), where i is a vertex and c is a color, to discover the maximal reduction of the number of conflicts; if several such pairs i, c exist, one is chosen randomly [8, 10]. In [8] this neighborhood is

further reduced by considering only moves that affect vertices that are currently involved in a conflict.

The second architecture is greedier than the first, because at each step a larger set of candidate moves is examined. Since a straightforward implementation of a local search using the second neighborhood examination scheme is quite costly, we speed-up the neighborhood evaluation by means of a two-dimensional table T of size $n \cdot k$, where each entry $t_{i,j}$ 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 initialized in $\mathcal{O}(n^2 \cdot k)$, but each update of the matrix then only takes $\mathcal{O}(n \cdot k)$ in the worst case (for sparse graphs the update is much faster) [12].

In addition to this we enhanced both local search architectures with the use of a Tabu Search metaheuristic in order to avoid to get stucked in a local optimum. For the setting of the tabu list length we adopted two different schemes as suggested in the respective literature. Thus for the first local search architecture we used a fixed value, named *tabu length* (tl), and keep it as the parameter to tune; for the second architecture we followed, instead, a dynamic scheme [8, 14] where the length of tabu list is taken as $Random(A) + \delta \times |n_c|$, with n_c the set of conflicting vertices and δ a parameter to tune.

Our stopping criterion is the maximum number of iterations without improvement for the lowest number of constraint violations between vertices reached so far. More specifically, this number of maximum iteration is given by $|V| \times k \times f_{end}$ where f_{end} is a parameter to be decided.

4.1.2 Perturbation

The main part of our research concerns the examination of different ways of perturbing solutions. Here, we considered three different possibilities:

P1, **directed diversification**: we perform p_{iter} moves using the Tabu Search procedure with long tabu list settings and mix this strategy with random moves: 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).

P2, **random recoloring**: a certain number of colors is removed and vertices are recolored with a randomly chosen color different from the one they had previously.

P3, **dsat recoloring**: we use the same type of perturbation as in the random recoloring except that the recoloring of the vertices affected directly by the perturbation is done using the *dsat* heuristic.

The first kind of perturbation was already studied in [25] were it performed when compared to three other schemes which were (i) to assign to some vertices randomly chosen colors, (ii) to add a certain number of edges during some number of iterations, leading to a modification of the instance definition, and (iii) to assign randomly chosen colors to vertices that are in conflict.

Let us note that for all the perturbations the modifications introduced by the solution perturbation are recorded in the tabu list to prevent the search to immediately undo the perturbation.

4.1.3 Initial solution

As stated in Section 2, the initial solution is the solution provided by DSATUR (with a CPU time limit of 8 seconds on our machine). The number of colors is then reduced by one and the vertices that were assigned to the removed color are recolored with the *dsat* heuristic. This decreasing procedure ends when the coloring becomes infeasible.

4.1.4 Acceptance criterion

The acceptance criterion uses one of the following rules: accept every new solution or always apply the perturbation to the best solution found so far in the search. This corresponds to a strategy for the diversification and the intensification of the search, respectively.

5 Experimental results

5.1 Local search

In a preliminary analysis we compare the two local search architectures. In Table 1 we show the results obtained by using only the two different local searches extended by the Tabu Search. For each instance, values of $tl \in \{1, 2, 3, 4, 5\}$ and of $\delta \in \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$ were tested, while as stopping criterion we used $f_{end} = 100$. We report in Table 1 the best results obtained among the different values for the parameters. We run both algorithms 10 times on each instance and we report the lowest number of colors found, the percentage of success (indicating the number of runs in which the instance was solved), the mean number of iterations for solving only the best k-coloring found and the mean time. For some instances the algorithms were unable to find a better coloring than the one found by DSATUR (which may be already optimal). In this case only the minimum k-coloring found is reported.

The results clearly indicate that the second architecture for the local search appears to outperform the first one: typically the number of colors found is less than for the first one.

For the next experiments we consider only the second local search and we take the values of δ for which the best solutions were obtained.

5.2 Perturbations

For the first kind of perturbation proposed we refer to [25] and we use the parameters settings and results presented there. The second and the third kind of perturbation have only one parameter to be tuned, that is, the number of colors to remove. To normalize this number among different instances, we decided to put it in relation with the current number of colors, by multiplying k by a coefficient $\gamma \in (0, 1)$. We tested $\gamma \in \{0.05, 0.15, 0.25\}$.

Further parameters to be considered are the number of local search iterations before the next perturbation and which acceptance criterion should be applied. For the former we use $ls_{iter} \times |V|$ and we tested $ls_{iter} = \{1, 2, 4, 8\}$ and for the latter we considered a percentage of new solutions accepted indicated by Div and we tested: $Div \in \{80\%, 100\%\}$ where $Div = 100\%$ corresponds to the case in which every new solution is accepted.

Table 2 shows the results obtained in 25 runs for the best combination of parameters on different instances. In the table we report the starting k-coloring which is the last one solved by DSATUR, the best k-coloring found at the end of the search, the percentage of successful runs, the mean of the total number of iteration needed by the algorithm to solve to feasibility all the k-coloring (from the starting one down to the best), the mean number of iterations for solving the best k-coloring found and the mean time for the whole run. Last columns indicate the combination of parameters which performed best.

From the experimental results we can observe the following results:

- on Leighton and Queens instances the third type of perturbation can reach the same or higher success rates in slightly less iterations than the second type.
- on Leighton and Queens instances the second and third type of perturbation shows better performance than the first one, because they are able to reach the same success rate in much less iterations.

	Local Search 1					Local Search 2				
inst.	*tl*	*k*	% succ.	iter.	time	δ	*k*	% succ.	iter.	time
1-Insertions-5	-	6	-	-	-	-	6	-	-	-
2-Insertions-4	-	5	-	-	-	-	5	-	-	-
3-Insertions-4	-	5	-	-	-	-	5	-	-	-
4-Insertions-4	-	5	-	-	-	-	5	-	-	-
1-FullIns-4	-	5	-	-	-	-	5	-	-	-
2-FullIns-4	-	6	-	-	-	-	6	-	-	-
3-FullIns-4	-	7	-	-	-	-	7	-	-	-
4-FullIns-4	-	8	-	-	-	-	8	-	-	-
5-FullIns-3	-	8	-	-	-	-	8	-	-	-
DSJC250.5	1	31	60	369,228	15.3	0.5	28	20	153,608	7.2
DSJC500.1	1	14	100	11,307	1.5	1	12	10	564,769	17.5
DSJC500.9	3	132	10	2,343	80.7	1	127	10	3,573	37.5
DSJR500.1	-	12	-	-	-	-	12	-	-	-
DSJR500.5	1	126	40	204,312	31.1	0.5	124	20	1,322,263	74.4
le450-5a	3	6	40	2,605	1.4	1	5	100	43,976	2.4
le450-5b	2	6	30	5,713	1.4	0.5	5	60	39,755	2.1
le450-5d	3	6	100	14,511	1.9	0.5	5	100	2,383	3,0
le450-15a	-	17	-	-	-	0.5	15	100	69,575	2.6
le450-15b	-	16	-	-	-	1	15	100	55,241	2.3
le450-15c	2	17	20	2,570	9.3	3	15	60	28,034	3.7
le450-15d	5	18	50	3,911	5.9	2	15	20	65,606	4.9
le450-25c	-	29	-	-	-	0.5	26	100	100,595	3.8
le450-25d	-	28	-	-	-	0.5	26	100	85,563	3.3
queen13-13	-	16	-	-	-	0.5	14	50	104,607	2.4
queen16-16	-	19	-	-	-	0.5	18	100	1,267	1.0

Table 1: Comparison between two local search schemes enhanced by Tabu Search. Results are averaged over 10 runs. We report for each instance the value of δ which gave best results, the best k-coloring found, the percentage of successful runs, the mean number of iterations needed for solving only the best k-coloring and the time expressed in seconds.

P1										
inst.	start k	best k	% succ.	tot. iter.	iter.	time	ls_{iter}	Div	p_{iter}	w_p
DSJC250.5	-	28	76	-	3,550,914	-	30	100	1	0.5
DSJC500.1	-	-	-	-	-	-	-	-	-	-
le450-15a	-	15	100	-	95,246	-	30	100	0.5	0.5
le450-15b	-	15	100	-	63,920	-	30	100	0.2	0.5
le450-15c	-	15	100	-	451,714	-	10	100	2	0.25
le450-15d	-	15	100	-	2,128,483	-	10	100	2	0.25
le450-25c	-	26	100	-	108,312	-	20	100	0.2	0.5
le450-25d	-	26	100	-	99,785	-	30	100	0.2	0.25
queen13-13	-	-	-	-	-	-	-	-	-	-

P2									
inst.	start k	best k	% succ.	tot. iter.	iter.	time	ls_{iter}	Div	γ
DSJC250.5	36	28	40	439,151	367,213	27.9	4	100	0.05
DSJC500.1	15	12	8	389,887	385,389	17.2	4	80	0.15
le450-15a	17	15	100	60,375	60,302	4.24	8	100	0.05
le450-15b	16	15	100	72,287	72,287	4.9	8	80	0.05
le450-15c	29	15	100	262,250	107,173	41.3	4	80	0.25
le450-15d	23	15	100	237,346	168,355	27.4	2	80	0.05
le450-25c	29	26	96	109,264	103,636	13.5	2	100	0.05
le450-25d	28	26	100	85,889	85,374	6.4	4	100	0.05
queen13-13	16	14	64	70,711	70,410	2.9	4	80	0.05

P3									
inst.	start k	best k	% succ.	tot. iter.	iter.	time	ls_{iter}	Div	γ
DSJC250.5	36	28	40	390,408	324,157	39.6	2	100	0.05
DSJC500.1	15	12	4	285,815	274,567	11.3	8	100	0.15
le450-15a	17	15	100	57,918	57,821	4.0	8	100	0.05
le450-15b	16	15	100	67,492	67,492	6.1	4	80	0.15
le450-15c	29	15	100	180,764	84,967	31.6	4	80	0.25
le450-15d	23	15	100	355,683	95,958	77.1	1	80	0.25
le450-25c	29	26	100	100,107	94,839	8.2	4	100	0.05
le450-25d	28	26	100	102,457	101,887	5.5	8	100	0.05
queen13-13	16	14	88	72,887	72,623	2.6	4	80	0.15

Table 2: Comparison between three perturbation schemes. Results are obtained on 25 runs with $f_{end} = 100$. We report for each instance the starting k-coloring corresponding to the best k-coloring solved by DSATUR in 8 seconds, the best k-coloring found by the ILS algorithm, the percentage of successful runs, the mean number of iterations for solving all k-colorings, the mean number of iterations for solving only the best k-coloring, the mean CPU time needed expressed in seconds, and the winning parameters characterizing the strength of perturbation (percentage of diversification, Div, number of local search iterations before perturbing the solution, ls_{iter}, number of perturbing moves of kind P1, p_{iter}, walk probability w_p, and fraction of current k colors to remove, γ). All means are computed only over the successful runs.

- the results on Random Instances (DSJC250.5 and DSJC500.1) seem to indicate that the first kind of perturbation is better than the second and third one. It has to be considered, however, that in [25] a number of ten millions iterations was used as stopping criterion while in our case this number is much lower.

In general, there seems to be a slight advantage for the third type of perturbation. The advantage of this type of perturbation lies in the possibility of combining two different ways of guiding the search, each of them exploiting a different type of information. In particular, the *dsat* heuristic reconstructs part of a solution according to a heuristic which is different from the one used by the local search. Concerning the strength of the perturbation, some conclusion can be drawn by considering the best combinations of parameters. In particular, the number of colors to remove before applying the *dsat* heuristic seems to be low. Hence, the part of solution to be reconstructed is small and the strength of the perturbation is far from the extreme case of a random restart. Only instances le450-15c and le450-15d seem to require more disruptive perturbations.

Another positive aspect of the *dsat* recoloring (and also the random recoloring) when compared to the directed diversification is that the recoloring perturbation appears to be more robust and is much easier to tune. The latter is mainly due to the fact that the recoloring perturbation only involves one parameter, while the directed diversification uses two.

When comparing the results of the ILS with those of the simple local search in Table 1, we notice that, with the only exception of instance DSJC500.1, the success rate increases. This indicates that, under equal conditions, the application of ILS helps to improve the performance of local search and it gains further robustness.

5.3 Comparison to available metaheuristics

We report the peak performance of ILS in Table 3 and we compare the results with the best results published so far which are those obtained by the Hybrid Evolutionary Algorithm (HEA) of Galinier and Hao [14], which is currently one of the best performing metaheuristics for the graph coloring problem. They also present peak performance, intended as consequence of tuning efforts. The HEA algorithm applies a recombination of two solutions which are randomly chosen from the population and then improves the new solution by applying the same Tabu Search algorithm we used here. Therefore comparison can be made properly comparing the number of iterations of Tabu Search, because the overhead due to the perturbation or the crossover is minor. In the table we report the best k-coloring found and the number of Tabu Search iterations needed for solving only that specific number of colors.

For the structured graphs we just report the results of Table 2. Indeed, already with $f_{end} = 100$ ILS requires less iterations for finding feasibility and clearly appears more robust. Here, times reported are the times relative only to the last coloring, as done in [14].

For the random graphs the results of Table 2 are worst than those presented for HEA. However the conditions in which they were generated were different. We tried, therefore, to reproduce the conditions as equal as possible by solving only the given number of colors and fixing $f_{end} = 300$. Results improve and become comparable with those of HEA, at least in terms of speed. However a larger and more rigorous experimental analysis has to be conducted to draw statistically significant conclusions.

In Appendix we report the results on a larger number of instances from the benchmark suite obtained with one reasonable parameter setting.

6 Conclusions

In this work we studied the application of ILS to graph coloring problem focusing on large or hard instances, where we assumed as “hard” the instances which are not easily solvable by

	ILS			HEA		
inst.	best k	% succ.	iter.	best k	% succ.	iter.
DSJC250.5	28	80	557,949	28	90	490,000
le450-15c	15	100	84,967	15	60	194,000
le450-25c	26	100	94,839	26	100	800,000

Table 3: Experimental results of ILS compared with those of Hybrid Evolutionary Algorithm (HEA) presented in [14]. ILS was run 10 times on each instance. For the instance DSJC250.5 f_{end} was set equal to 300 while for the other instances was 100. We report the best k-coloring found, the percentage of successful runs and the mean number of iterations for finding the best k taken over the successful runs.

the DSATUR exact algorithm. The main contribution of this abstract is the analysis of the performance of different kinds of perturbations for the Iterated Local Search metaheuristics. The comparison was made after a phase of tuning so that the best configuration of parameters were considered. Among the perturbations tested the most robust approach seems to be to remove a few number of colors and reconstructing the solution by means of some heuristic information. Compared to the perturbation applied in [25], this perturbation also has the advantage of requiring only one parameters instead of three.

In this extended abstract we compared algorithms based on peak performance, that is, the performance of the algorithms when the parameters are reasonably fine-tuned (experiments on the larger instances in the benchmark suite are currently under way and the results will be presented at the Symposium). In this process, however, we noted that parameter settings may be sensitive to the particular instance at hand. It is our intention to extend our algorithms by some type of reactive mechanism, where parameters are tuned automatically during the algorithm run. To this aim, we are currently analyzing the relationship between distance of solutions, diversification and search performance in order to gain deeper insight in the problem and devise a reasonable strategy to automatically adjust the parameters.

Acknowledgments We would like to thank Marco Pranzo, Luis Paquete and Irina Dumitrescu for suggestions and discussions which contributed to the development of this work.
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] E.B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. Manuscript, 1986.

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

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

[4] D.J. Castelino, S. Hurley, and N.M. Stephens. A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 63:301–320, 1996.

[5] 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.

[6] J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. In D.S. Johnson and M.A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 499–520. American Mathematical Society, 1996.

[7] 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.

[8] R. Dorne and J. Hao. Tabu search for graph coloring, t-coloring and set t-colorings. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Metaheuristics: advances and trends in local search paradigms for optimization*, pages 77–92. Kluver Academic Publishers, 1998.

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

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

[11] C. Fleurent and J.A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In D. S. Johnson and eds. M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652. American Mathematical Society, 1996.

[12] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. In et P. L. Hammer G. Laporte, I. H. Osman, editor, *Annals of Operations Research*, volume 63, pages 437–461. Baltzer Science Publishers, 1996.

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

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

[15] 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.

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

[17] 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.

[18] 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.

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

[20] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 85:489–506, 1979.

[21] 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.

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

[23] 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.

[24] J. Mycielski. Sur le coloriage des graphes. *Colloquim Mathematiques*, 3:161–162, 1955.

[25] L. Paquete and T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G.R. Raidl, editors, *Applications of Evolutionary Computing*, volume 2270 of *LNCS*, pages 122–131. Springer-Verlag, Berlin, Germany, 2002.

[26] T. Stützle. Applying iterated local search to the permutation flow shop problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt, August 1998.

A Instances classification

Our instances classification. We divided the instances into four groups: (i) those which are solved to optimum by the DSATUR algorithm in less than 8 seconds on our machine, a Pentium III 700 MHz processor with 256 KB cache and 512 MB RAM; (ii) the instances for which the exact optimum is known but it can not be found by DSATUR in short time; (iii) the instances for which the optimum is not known and (iv) the instances of size larger than 1000 vertices.

solved by DSATUR	opt-known	opt-unknown	large
inithx.i.1	1-Insertions-4	1-FullIns-3	5-FullIns-4
2-Insertions-3	2-Insertions-4	1-FullIns-4	3-FullIns-5
3-Insertions-3	4-Insertions-3	1-FullIns-5	3-Insertions-5
anna	le450-15a	1-Insertions-5	4-FullIns-5
david	le450-15b	1-Insertions-6	abb313GPIA
fpsol2.i.1	le450-15c	2-FullIns-3	ash608GPIA
fpsol2.i.2	le450-15d	2-FullIns-4	ash958GPIA
fpsol2.i.3	le450-25c	2-FullIns-5	qg.order100
games120	le450-25d	2-Insertions-5	qg.order60
homer	le450-5a	3-FullIns-3	wap01
huck	le450-5b	3-FullIns-4	wap02
inithx.i.2	le450-5d	3-Insertions-4	wap03
inithx.i.2	queen11-11	4-FullIns-3	wap04
inithx.i.3	queen13-13	4-FullIns-4	wap05
inithx.i.3	queen8-8	4-Insertions-4	wap06
jean	queen9-9	5-FullIns-3	wap07
le450-25a		ash331GPIA	wap08
le450-25b		DSJC1000.1	
miles1000		DSJC1000.5	
miles1500		DSJC1000.9	
miles250		DSJC125.1	
miles500		DSJC125.5	
miles750		DSJC125.9	
mug100-1		DSJC250.1	
mug100-25		DSJC250.5	
mug88-1		DSJC250.9	
mug88-25		DSJC500.1	
mulsol.i.1		DSJC500.5	
mulsol.i.2		DSJC500.9	
mulsol.i.3		DSJR500.1	
mulsol.i.4		DSJR500.1c	
mulsol.i.5		DSJR500.5	
myciel3		latin-square-10	
myciel6		queen10-10	
myciel7		queen12-12	
queen5-5		queen14-14	
queen8-12		queen15-15	
zeroin.i.1		queen16-16	
zeroin.i.2		school1-nsh	
zeroin.i.3		school1	
myciel4		will199GPIA	
le450-5c			
myciel5			
qg.order30			
qg.order40			
queen6-6			
queen7-7			

B Results on the benchmark suite

We report the results of the ILS described in the text on instances taken from the benchmark suite. The parameter setting is kept as constant as possible and is the one that guarantees on average best performance. Values of $\delta = \{0.5, 1\}$ for the tabu length parameter and $\gamma = \{0.1, 0.001\}$ for the perturbation parameter are used. As stopping criteria $f_{end} = 200$ is maintained in all the experiments as also the number of tabu search iterations between two perturbations which is $k \cdot |V|$. Given are the start coloring number given by DSATUR, the best k-coloring found, the percentage of successful runs, the mean number of iterations for solving all k-colorings, the mean number of iterations for solving only the best k-coloring and the mean CPU time needed expressed in seconds. Means are taken over the successful runs. A last column indicate when parameters are changed. Where only the minimum k-coloring found is reported it means that ILS was not able to improve the results found by DSATUR (the optimal solution for those instances is unknown). We include also results on real-life optical network design problems.

inst.	nodes	density	start k	best k	% succ.	tot. iter.	par. iter	time	(δ, γ)
DSJC250.5	250	0.50	36	28	80	691,710	642,703	37 sec	(0.3,0.001)
DSJC500.1	500	0.10	15	13	100	7,999	7,408	4 sec	(0.3,0.001)
DSJC500.5	500	0.50	64	49	20	3,929,822	2,287,199	498 sec	(0.3,0.001)
DSJC500.9	500	0.90	163	126	10	5,872,162	2,615,735	19 min	(0.3,0.001)
DSJR500.5	500	0.47	130	124	30	5,889,767	5,456,854	369 sec	(0.3,0.001)
DSJC1000.5	1000	0.50	114	89	10	26,468,228	14,571,427	1.6 h	(0.3,0.001)
le450-15a	450	0.08	17	15	100	153,242	153,114	5 sec	(0.3,0.001)
le450-15b	450	0.08	16	15	100	83,354	83,354	3 sec	(0.3,0.001)
le450-15c	450	0.17	23	15	90	754,240	568,940	41 sec	(1,0.1)
le450-15d	450	0.17	24	15	30	624,473	520,935	42 sec	(1,0.1)
le450-25c	450	0.17	29	26	100	186,314	180,518	12 sec	(0.3,0.001)
le450-25d	450	0.17	28	26	100	243,991	243,528	11 sec	(0.3,0.001)
2-Insertions-4	149	0.05	5	5					(0.3,0.001)
3-Insertions-4	281	0.03	5	4	20	13,677	13,677	1 sec	(0.3,0.001)
3-Insertions-5	1406	0.01	6	6					(0.3,0.001)
3-FullIns-4	405	0.04	7	7					(0.3,0.001)
3-FullIns-5	2030	0.02	8	8					(0.3,0.001)
4-FullIns-5	4146	0.01	9	9					(0.3,0.001)
latin-square-10	900	0.76	129	99	60	14,807,059	10,194,314	1.6 h	(0.3,0.1)
qg.order60	3600	0.03	63	60	100	1,175	1,101	568 sec	(0.3,0.001)
queen13-13	169	0.47	16	14	100	76,583	76,347	2 sec	(0.3,0.001)
queen16-16	256	0.39	19	18	100	1,391	1,391	1 sec	(0.3,0.001)
wap01	2368	0.04	48	42	20	213,005	81,545	45 sec	(1,0.1)
wap02	2464	0.04	46	42	100	297,935	285,218	72 sec	(1,0.1)
wap03	4730	0.03	55	45	60	621,568	253,392	335 sec	(1,0.1)
wap04	5231	0.02	48	44	70	15,757,256	15,466,696	2 h	(1,0.1)
wap05	905	0.11	51	50	100	206	206	2 sec	(1,0.1)
wap06	947	0.10	45	42	100	69,397	56,012	10 sec	(1,0.1)
wap07	1809	0.06	46	43	20	351,929	217,320	71 sec	(1,0.1)
wap08	1870	0.06	45	43	100	42,589	34,802	16 sec	(1,0.1)

C Benchmark code

We report the results obtained by running the benchmark code available from `http://mat.gsia.cmu.edu/COLOR/color.html` on our machine.

```
DFMAX(r100.5.b)
0.01 (user) 0.00 (sys) 0.00 (real)
Best: 4 57 35 5 61 34 3 62 90

DFMAX(r200.5.b)
0.13 (user) 0.00 (sys) 0.00 (real)
```

```
Best: 113 86 147 66 14 134 32 127 161 186 70

DFMAX(r300.5.b)
1.16 (user) 0.00 (sys) 1.00 (real)
Best: 279 222 116 17 39 127 190 158 196 288 263 54

DFMAX(r400.5.b)
7.15 (user) 0.00 (sys) 7.00 (real)
Best: 370 108 27 50 87 275 145 222 355 88 306 335 379

DFMAX(r500.5.b)
27.26 (user) 0.01 (sys) 28.00 (real)
Best: 345 204 148 480 16 336 76 223 260 403 141 382 289
```