A Study of Evaluation Functions for the Graph K-Coloring Problem

Daniel Cosmin Porumbel[1], Jin-Kao Hao[1] and Pascale Kuntz[2]

[1] LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France
[2] LINA, PolytechNantes, 44306 Nantes, France

Abstract. The evaluation or fitness function is a key component of any heuristic search algorithm. This paper introduces a new evaluation function for the well-known graph K-coloring problem. This function takes into account not only the number of conflicting vertices, but also inherent information related to the structure of the graph. To assess the effectiveness of this new evaluation function, we carry out a number of experiments using a set of DIMACS benchmark graphs. Based on statistic data obtained with a parameter free steepest descent, we show an improvement of the new evaluation function over the classical one.

1 Introduction

Heuristic algorithms are known to be a very powerful tool for solving hard and large combinational optimization problems. It is now well recognized that the performance of a heuristic algorithm is strongly conditioned by the design of a number of key components. For instance, for local search algorithms, the neighborhood relation constitutes an element that must be carefully studied. Similarly, for evolutionary algorithms, it is important to seek problem specific operators such as crossover and mutation in order to obtain a better search performance. For both local and genetic search paradigms, another indispensable key component is the evaluation or fitness function. Indeed, it is this function that guides the search process to explore an arbitrarily large search space.

There are different approaches to design an informative evaluation function [14]. First, the static or dynamic penalty approach is a well established technique for constrained problems. Here relaxed constraints are integrated into the evaluation function with special penalty terms, which can be fixed statically or tuned dynamically. Second, in a hierarchical approach, the evaluation function is decomposed into several ordered components; the evaluation is realized according to that order. The third approach, less studied in the literature, consists in designing specific evaluation function specially adapted to the problem at hand. Contrary to the penalty or hierarchical approaches which are general techniques and thus applicable to different problems, the problem-specific approach requires a fine analysis of the target problem in order to identify particular properties that are useful for the design of the evaluation function. Such a problem-specific approach has demonstrated its effectiveness for several NP-hard problems [4, 8, 13].

In this paper, we consider the well-known graph K-coloring (K-COL) problem and try to devise a new and more informative evaluation function for a heuristic solving of this problem. Informally, for a given graph, K-COL requires to find a conflict-free vertex coloring using only K different colors such that two adjacent vertices receive two different colors. As one of the three problems chosen for the DIMACS second implementation challenge [9], K-COL is certainly one of the most studied NP-complete problems with a large number of solution algorithms.

The long-term goal of our study is the development of high performance algorithms able to produce competitive results across a large range of benchmark instances. For this purpose, we here focus our study on the discovery of a new evaluation function that will provide a better guidance than the classical penalty based evaluation function for local or genetic search algorithms. Indeed, the conventional evaluation function (called f in the paper) simply counts, for a given K-coloring, the number of conflicting vertices and consequently cannot distinguish two K-colorings of equal number of conflicts having different potential for further improvements.

2 Heuristic Search for Graph Coloring

2.1 Graph K-coloring and graph coloring

Definition 1. *(K-COL) Given a graph $G = (V, E)$ (V and E are respectively the vertex and edge set) and a positive integer K such that $K < |V|$, the graph K-coloring problem is to determine whether there exists a conflict-free vertex coloring using K colors or less, i.e. a function $c : V \to \{1, 2, \cdots, K\}$ such that $\forall \{i, j\} \in E$, $c(i) \neq c(j)$. If such a coloring exists, G is said to be $K-colorable$. In the following, we denote a coloring by $C = (c(1), c(2), \cdots, c(|V|))$.*

Definition 2. *(COL) Given a graph G, the graph coloring problem is to determine the smallest K such that G is $K-$colorable. The smallest K is the chromatic number of G.*

From a theoretical viewpoint, K-COL is a very important NP-complete problem as it is one of the 21 NP-complete problems listed [10]. Coloring problems are also at the heart of numerous applications including for instance, scheduling, register allocation in compilers and frequency assignment in mobile networks.

The graph coloring problem COL is an *optimization* problem (to minimize K), while K-COL is its corresponding decision problem (to determine whether there exists a K-coloring or not). Notice that if one can solve K-COL, one can also solve COL by the following iterative approach: find a $K-$coloring of G for a fixed K, $K < |V|$ (solve K-COL), then decrease K ($K = K - 1$) until no conflict-free K-coloring can be found.

2.2 Local search for K-coloring

To solve K-COL by local search, we consider K-COL from an optimization point of view. For a given K-COL instance, i.e., a graph $G = (V, E)$ and an integer K, we define the following optimization problem (S, f) where:

- S is the search space composed of all the $|V|^K$ possible K-colorings, i.e. $S = \{C | C : V \to \{1, 2, \cdots, K\}\}$;
- f is an "artificial" objective counting the number of conflicting edges, i.e. $\forall C \in S, C = (c(1), c(2), \cdots, c(|V|))$,

$$f(C) = \sum_{\{i,j\} \in E} p_{ij}, \text{ where } p_{ij} = \begin{cases} 1 & \text{if } c(i) = c(j) \\ 0 & \text{if } c(i) \neq c(j) \end{cases}$$

Accordingly, any $C^* \in S$ such that $f(C^*) = 0$ corresponds to a conflict-free K-coloring and thus represent a solution to the given K-COL instance. To solve this optimization problem by local search, three main components need to be defined: an evaluation function, a neighboring relationship and an exploration strategy of the neighborhood.

- Evaluation function: One convenient *evaluation function* is the above objective function f which counts the number of conflicting edges of a given K-coloring. Indeed, this evaluation function is largely used by many well-known coloring algorithms [7, 5, 2, 6, 1]. We will show in this paper that this evaluation function is not discriminating enough and can be improved.
- Neighborhood: Given a configuration (K-coloring) $C = (c(1), c(2), \cdots, c(|V|))$, a neighboring configuration C' of C is a K-coloring $C' = (c'(1), c'(2), \cdots, c'(|V|))$ with *exactly* one conflicting color $c(i)$ being changed to $c'(i)$.
- Exploration strategy: The exploration strategy used in this paper is a steepest descent detailed in section 4.1.

3 A New Evaluation Function

For the graph K-coloring problem, many previous algorithms use f as their evaluation function, although other functions are also proposed (see for instance [12, 3]). However, the function f is not sufficiently discriminating since it cannot distinguish configurations having the same number of conflicts while these colorings may have different possibilities for further improvement. To overcome this difficulty, we are trying to define another evaluation function able to identify the configurations which, despite of having the same conflict number, are more promising for further exploration.

3.1 The new evaluation function

Let us consider two configurations (3-colorings) C_1 (figure 1.a)) and C_2 (figure1.b)) of the same graph for which one needs to obtain a 3-coloring without conflicts.

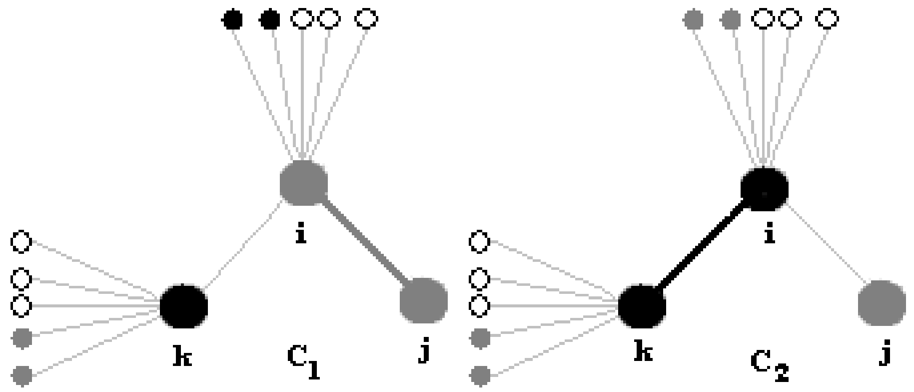

Fig. 1. Two 3-colorings with one conflict. The conflicting edge is marked in larger thickness; it is easier to solve the gray one (C_1, left) than the black one (C_2, right) even if both configurations have just a single conflict

We denote by E_{confl} the set of edges in conflict, by $\delta(i)$ the degree of vertex i and by $confl(i)$ the number of conflicts for vertex i.

The E_{confl} set has just one element for both examples: $\{i, j\}$ for C_1 and $\{i, k\}$ for C_2. Since $\delta(j) < K < \delta(k)$, we can assign to j a color not used by its $\delta(j)$ neighbors (i.e black or white) to solve the $\{i, j\}$ conflict; it requires just one more step. Since this is not necessarily the case for the conflict $\{i, k\}$ in C_2, C_1 is here preferable to C_2. Furthermore, it is natural to consider that C_1 is preferable to C_2 only because $\delta(j) < \delta(k)$ (without considering the value of K), since the more neighbors a vertex has, the more difficult it is to change its color without perturbing the rest of the configuration. In order to consider all conflicting vertices in a single formula, we propose the following evaluation function:

$$\hat{f}(C) = f(C) - \sum_{i \in V} confl(i) \frac{1}{\delta(i)}. \tag{1}$$

In all practical cases, we have $f(C) - 1 < \hat{f}(C) < f(C)$ since, for a non-trivial problem the final number of conflicts (the number of terms in the sum) is considerably smaller than the average degree of conflicting vertices (average denominator $\overline{\delta(i)}$). The second part of the function allows us to discriminate the configurations *having the same number of conflicts*; note that $\hat{f}$ preserves the f ordering: $\hat{f}(C) < \hat{f}(C')$ whenever $f(C) < f(C')$. In other words, we have two components each one with a different goal: (a) the first counts the number of conflicts (f more precisely), (b) the second is a quantity of the form $\sum_{i \in V_{conflict}} \frac{1}{\delta(i)}$ (which is less than 1) that better discriminates colorings unable to be distinguished by f. All reported values of $\hat{f}$ in this paper will be rounded to the nearest greater integer since all encountered values of $\hat{f}(C)$ satisfy $\left\lceil \hat{f}(C) \right\rceil = f(C)$.

3.2 Computational complexity

The computational efforts required by f and $\hat{f}$ are equivalent. To see this, we re-write the formula of $\hat{f}$ in a computationally convenient way. Let us remark that:

$$\sum_{i \in V} confl(i) \frac{1}{\delta(i)} = \sum_{\{i,j\} \in E_{confl}} \left(\frac{1}{\delta(i)} + \frac{1}{\delta(j)}\right).$$

Consequently, we have:

$$\hat{f}(C) = f(C) - \sum_{\{i,j\} \in E_{confl}} \left(\frac{1}{\delta(i)} + \frac{1}{\delta(j)}\right) = \sum_{\{i,j\} \in E_{confl}} \left(1 - \frac{1}{\delta(i)} - \frac{1}{\delta(j)}\right)$$

In our implementation, both functions are constructed by adding constant coefficients ($E[i,j] = 1 - \frac{1}{\delta(i)} - \frac{1}{\delta(j)}$) for each conflicting edge $\{i,j\}$. All values from E are computed before starting the main algorithm and they have a time complexity of $(O(|V|^2))$. In our numerical experiments, the computing time for E is always less than 1 second on a Pentium 4 with a CPU at 2.8GHz. The only non-negligible difference is the data types we are manipulating: instead of integer values we use double values to store the table E and to perform all operations.

4 Experimental Comparisons of the Two Evaluation Functions

In this section, we perform an extensive experimental analysis of the effect of $\hat{f}$ on the steepest descent (SD) algorithm. We start by detailing the algorithm, then we present the test instances, the comparison criteria and we analyze the results.

4.1 Steepest descent

The steepest descent (SD) algorithm starts from a random initial configuration and iteratively chooses, from the whole neighborhood, the *best* neighbor according to the evaluation function. When there exists several equally best neighbors, one of these neighbors is chosen at random. The algorithm stops when there exists no improving neighbor - i.e. when the current coloring is a local optimum with respect to the given neighborhood relation and evaluation function. For all the experimental results reported in this paper, this simple SD algorithm is considered with the two evaluation functions f and $\hat{f}$.

The choice of the SD algorithm for our experimentations is here justified by the fact that it is one of the few algorithms to ensure a complete neutrality in the final results. In any algorithm which closely depends on a given parameter (e.g. temperature in simulated annealing, tabu list length in Tabu Search, etc), the tuning of this parameter might significantly skew the results favoring one method or another.

We also started performing experiments with other more advanced algorithms (especially Tabu Search), but however, the aim of this paper is not to compete with the best graph coloring algorithms. The goal of our experiments is to study the influence of evaluation functions for the graph coloring algorithms from a completely neutral point of view.

4.2 The experimental conditions

Instances The following graphs from the well-known second DIMACS challenge benchmarks are used:

- Five uniform random graphs generated by Johnson *et. al* in their state-of-the-art papers about simulated annealing [8] and used extensively afterwords in testing graph coloring algorithms: $dsjc250.5$, $dsjc500.5$, $dscj1000.1$, $dsjc1000.5$ and $dscj1000.9$. They have 250, 500 and 1000 vertices respectively and the density p is denoted by the last digit (i.e. .5 for the first graph).
- Three Leighton graphs: $le450.15a$, $le450.25a$ and $le450.25c$, these graphs have each 450 vertices and a known chromatic number (15 for the first, 25 for the others) [11]. The last two graphs are generated in the same manner, but with different random seeds.

Comparison criteria The main indicator of *solution quality* is the number of conflicts of the configuration obtained at the end of the search (Other criteria are explained in Section 5. For each graph, we set K to be the smallest number of colors for which a coloring has ever been found or the chromatic number when it is known (i.e. for the Leighton graphs). Consequently, all these instances are difficult to solve. For the first five graphs (dsjc*.*) we use the least K found either by a hybrid algorithm [6] or by a population based local search algorithm combining two specific neighborhoods and using the strategy of successive building of color classes [12].

Experimental protocol The experimental evaluation was carried out by considering 1000 independent runs, with different random seeds, for both functions and by statistically analyzing the solutions obtained. We examine the extremal conflict numbers found with the two evaluation functions and precisely analyze the distributions of the values obtained on the run set.

4.3 Results

For each instance, we show in table 1 the minimum, the mean and the maximum solution quality (columns 2,3,4 and 6,7,8 respectively) computed separately

	Classic Function(f)				New Function($\hat{f}$)			
Graph (colors)	Min	Max	Mean	Std. Dev.	Min	Max	Mean	Std. Dev.
$dsjc250.5(28)$	60	106	83.0	7.4	36	71	54.1	5.6
$dsjc500.5(49)$	140	209	173.1	10.7	89	136	112.2	8.2
$dsjc1000.1(20)$	260	355	307.2	15.2	152	231	191.8	11.7
$dsjc1000.5(83)$	364	478	424.9	16.6	249	333	290.0	13.3
$dsjc1000.9(224)$	301	402	347.5	14.4	183	253	218.9	9.5
$le450.15c(15)$	270	345	310.3	10.6	216	284	250.3	9.9
$le450.25a(25)$	11	28	18.2	2.8	0	10	4.6	1.6
$le450.25c(25)$	87	128	107.7	6.1	51	78	64.1	4.8

Table 1. The results of 1000 runs of the SD algorithm on all the tested graphs. $\hat{f}$ allows SD to obtain a better local optimum with a smaller number of conflicts for each graph and even to find an optimal coloring for $le450.25a$. The numbers between the parenthesis in Column 1 correspond to the smallest K reported in the literature.

for both functions. We also compute the standard deviation (columns 5 and 9 respectively), as it is an indicator of the algorithm's precision and robustness.

The first observation is that the distributions of f and $\hat{f}$ are disjoint in 75% of cases: the quality of the solutions obtained with $\hat{f}$ is always better (with smaller numbers of conflicts) for absolutely all runs. And moreover, even for the rest of 25% cases, the general tendency remains the same.

More surprisingly, let us remark that $\hat{f}$ leads one time to a proper coloring (no conflicts) for the $le450.25a$ graph with K=25 (its chromatic number). In fact, even some state-of-the-art algorithms like HGA ([6]) fails to find a conflict free coloring with 25 colors for graphs in the $le450.25a$ family.

Furthermore, we collected for each graph the final values obtained by our SD algorithm with the two evaluation functions f and $\hat{f}$ into a single sample and depicted in Figure 2 the distribution of the results according to two axes: quality (number of conflicts) and the frequency (number of colorings having a given quality). The distribution confirms once again the superiority of $\hat{f}$ in the search process. Indeed, the distribution with $\hat{f}$ is more on the left than the distribution with f, meaning that the solutions with $\hat{f}$ have a smaller number of conflicts.

Additionally, it is important to remark that all 1000 configurations found for each method and each graph are pairwise different, i.e. the algorithm never comes to two identical solutions. We consider two configurations $(c(1), c(2), \cdots c(|V|))$ and $(c'(1), c'(2), \cdots, c'(|V|))$ to be identical if and only if there exists a permutation σ of the set $\{1, 2, 3, \cdots, K\}$ such that the first configuration is mapped into the second by σ (i.e $\sigma(c(i)) = c'(i), \forall i \in \{1..|V|\}$).

5 Why the New Evaluation Function Works?

In this section, we try to understand why the new evaluation function $\hat{f}$ works better than the classical one f. For this purpose, we analyze the dynamics of the

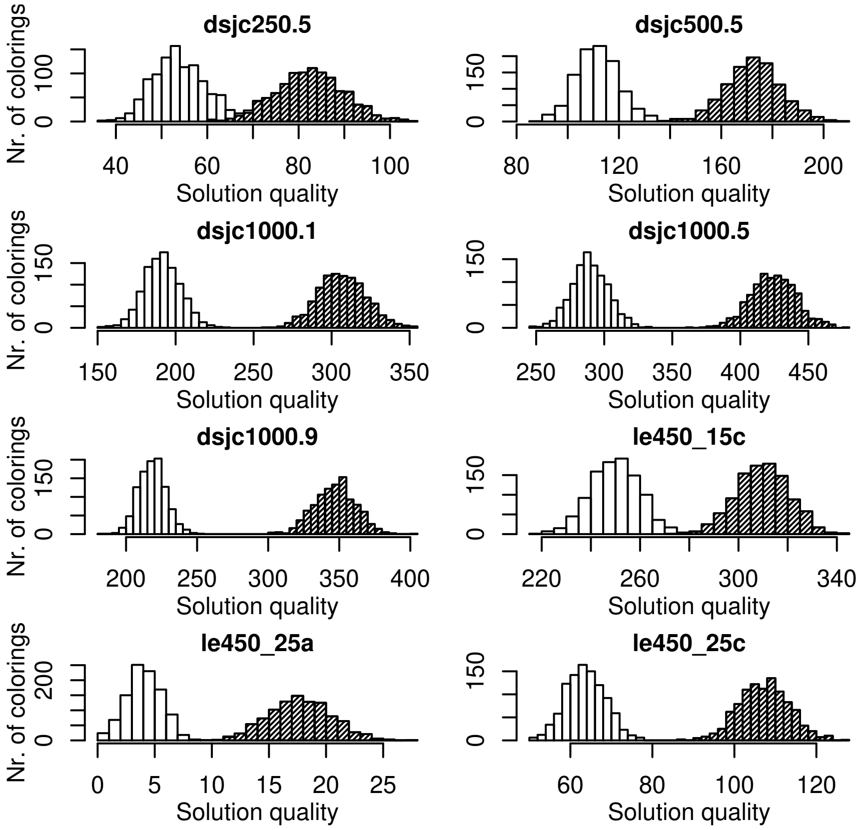

Fig. 2. The solution quality (evaluation function value) distribution for all graphs considering 1000 random steepest descents with the new function $\hat{f}$ (denoted by simple bars) and the classic one f (denoted in shading lines)

steepest descent with f and $\hat{f}$ and consider three indicators: a) the convergence of the SD algorithm with f and $\hat{f}$, b) the number of quality-improving neighbors induced by each evaluation function and c) the cardinality of equivalence classes of configurations.

5.1 Convergence

Table 2 indicates the total number of iterations performed by typical search processes using both functions. Figure 3 depicts the evolution of solution quality with both functions along the same scale. These results show that the SD with f is trapped earlier in a local optimum than with $\hat{f}$. For instance, for the graph ($dsjc250.5$), while the descent with f stops at 159^{th} iteration, the search con-

Graph (colors)	New Function($\hat{f}$)	Classic Function(f)
$dsjc250.5(28)$	245	158
$dsjc500.5(49)$	465	353
$dsjc1000.1(20)$	941	613
$dsjc1000.5(83)$	1003	703
$dsjc1000.9(224)$	921	635
$le450.25a(25)$	191	154

Table 2. The number of iterations performed by the steepest descent using f and $\hat{f}$. The descent process with $\hat{f}$ lasts always longer than the descent with f.

tinues with $\hat{f}$. This is possible because $\hat{f}$ offers improving neighbors for a longer time.

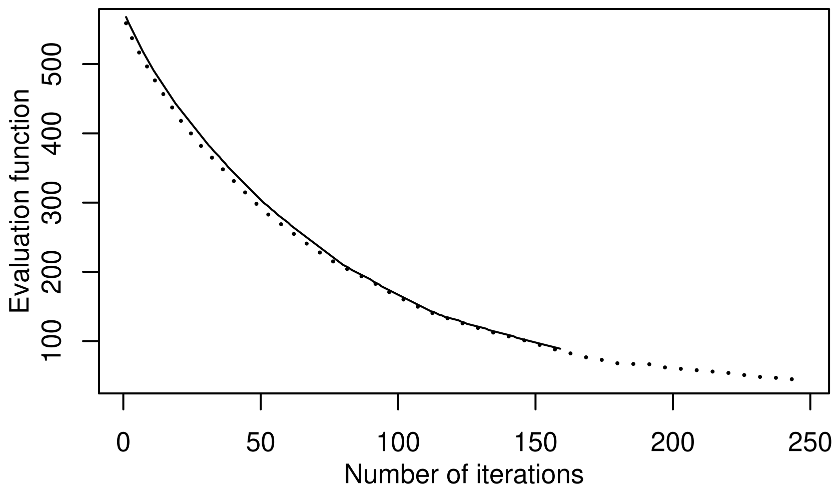

Fig. 3. The solution quality (number of conflicts) evolution for a classic steepest descent run ($G = dsjc250.5, K = 28$); $\hat{f}$ is depicted in dotted line, f in continuous line. The descent process with f stops at 159^{th} iteration while the search continues with $\hat{f}$.

5.2 Neighborhood analysis

An important indicator about the dynamics of a search process is the evolution of the number of improving neighbors during the search. Intuitively, if this number decreases rapidly, the search process might easily get blocked in a local optima. This is particularly true for a descent algorithm. Indeed, in a SD algorithm, the number of improving neighbors tends to monotonically decrease until this number drops to zero thus triggering the stop condition.

Let Δ denote the improvement added by a neighbor vertex V_{next} to the current vertex $V_{current}$ according to f or $\hat{f}$: $\Delta f = |f(V_{next}) - f(V_{current})|$ or $\Delta \hat{f} = |\hat{f}(V_{next}) - \hat{f}(V_{current})|$. For each coloring, we are interested to study: 1) how many improving neighbors are there at each step and 2) what is the actual improvement these neighbors can add (what values Δ can take).

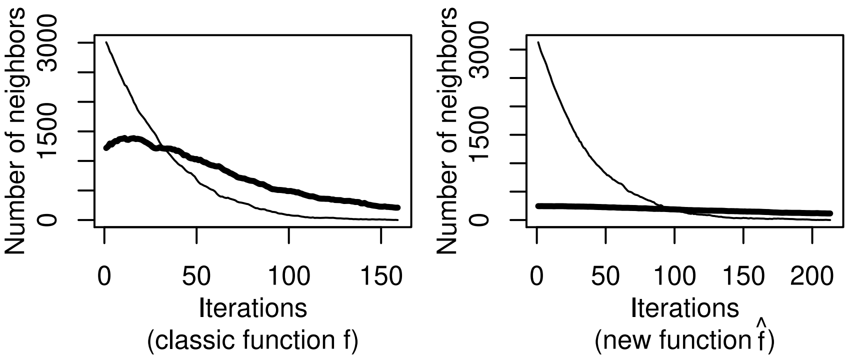

Fig. 4. A typical evolution of the number of quality-improving neighbors ($\Delta > 0$, in thin line) and of quality-stagnation neighbors ($\Delta = 0$, in thick line) for f (left) and $\hat{f}$(right)

Figure 4 depicts the first indicator on a typical run by drawing the curves for the number of neighbors satisfying $\Delta > 0$ (thin lines) and $\Delta = 0$ (thick lines). These curves confirm that $\hat{f}$ is more discriminating than f: the thick curves present significantly lower values for $\hat{f}$ than for f and, at each step, there are numerous equivalent neighbors for f especially.

Figure 5 depicts the distribution of Δ values, showing the degree of the improvements according to the two evaluation functions. An intriguing observation is that the improvement at each iteration is rather small; the algorithm performs many small steps rather than few large steps. In figure 5, one can see that the number of neighbors improving the current solution by more than 3 is usually very low. In most cases, the actual possible improvement of both f and $\hat{f}$ is 1 or 2. Furthermore, note that at the end of the search using $\hat{f}$ there are some improvements of less than 1 (in fact of almost 0). The existence of these improvements is justified only by the second part of the function $\hat{f}$ and not by the conflict number; and this explains why $\hat{f}$ can well distinguish between these colorings having the same number of conflicts.

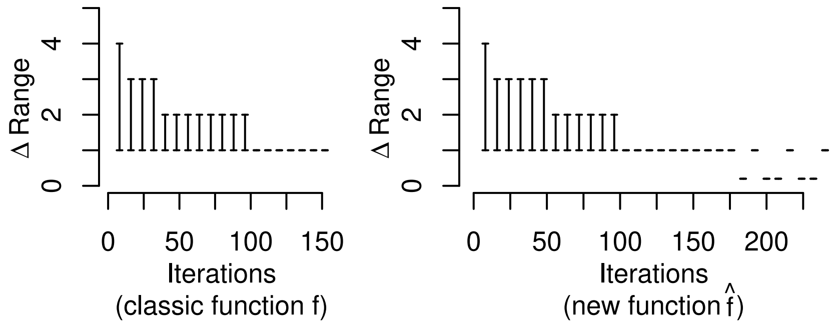

Fig. 5. A typical evolution of the possible improvement (values of Δ) in the neighborhood according to f (left) and $\hat{f}$ (right)

5.3 Classes of configurations in the landscape

An analysis of equivalence classes of configurations may be also useful for a better understanding of the dynamics of the search process. In our case, an equivalence class is a set of configurations that are evaluated at the same value by f or $\hat{f}$. Thus, two colorings are in the same f-class if they have the same conflict number and in the same $\hat{f}$-class if they have, in addition, the same degree distribution of their conflicting vertices. Consequently, the cardinal of a $\hat{f}$-class is considerably smaller and this is another indicator why the $\hat{f}-$search is more discriminating.

The cardinal of an $\hat{f}$-class is strongly influenced by the degree distribution of the considered graph. In regular graphs (all vertices have the same degree), any f-class is also an $\hat{f}$-class (so there is no practical difference between the two functions) since all edges $\{i, j\}$ generate the same values $E[i, j] = 1 - \frac{1}{\delta(i)} - \frac{1}{\delta(j)}$. At the opposite, in a heterogeneous graph with few vertices having the same degree, there are statistically very few edges generating equal values in the table E.

6 Conclusions and Further Work

In this paper we have introduced a new evaluation function $\hat{f}$ for the graph K-coloring problem. This evaluation function is based not only on the conflicts induced by a K-coloring, but also on information related to the structure of a graph. The experimental results with a steepest descent algorithm show that this function outperforms the classic evaluation function which is based only on the conflict number. To explain the good performance of the new evaluation function, some empirical justifications were proposed, based on the distribution of improving neighbors induced by each evaluation function and an analysis of the equivalent classes of configurations.

To further assess the practical usefulness of the proposed evaluation function, we are experimenting this function within a Tabu Search (TS) algorithm. The preliminary results show that the new evaluation function boosts the Tabu Search algorithm. Indeed, for a large number of the DIMACS graphs, the TS algorithm using $\hat{f}$ (as well as some other simple improvements) finds the best known colorings. Moreover, it is even able to improve on half of the results published by previous Tabu Search methods

More generally, we believe that the evaluation function introduced in this paper may be useful for other heuristic coloring algorithm and shed light on the design of other informative evaluation functions for the graph coloring problem.

Acknowledgments: This work is partially supported by the CPER project "Pôle Informatique Régional" (2000-2006) and the Régional Project MILES (2007-2009). We would like to thanks our referees for their useful comments.

References

1. C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.
2. R. Dorne and J.K. Hao. Tabu search for graph coloring, T-colorings and set T-colorings. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92, 1998.
3. A.E. Eiben and J.K. van der Hauw. Adaptive penalties for evolutionary graph coloring. *LNCS* 1365: 95–106, 1998.
4. E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:530, 1996.
5. C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.
6. P. Galinier and J.K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
7. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.
8. 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.
9. D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, USA, 1996.
10. R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 43:85–103, 1972.
11. F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503, 1979.
12. C. Morgenstern. Distributed Coloration Neighborhood Search. In [9], pages 335–357, 1996.
13. E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez. An improved evaluation function for the bandwidth minimization problem. *LNCS* 3242: 650-659, 2004.
14. E. Rodriguez-Tello and J.K. Hao. On the role of evaluation functions for heuristic search. *Working Paper*, 2007.