Intensification/Diversification-Driven ILS for a Graph Coloring Problem

Samir Loudni

▶ **To cite this version:**

Samir Loudni. Intensification/Diversification-Driven ILS for a Graph Coloring Problem. 12th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2012), Apr 2012, Malaga, Spain. pp.160-171. hal-01022859

HAL Id: hal-01022859
https://hal.science/hal-01022859

Submitted on 11 Jul 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Intensification/Diversification-Driven ILS for a Graph Coloring Problem

Samir Loudni

Université de Caen Basse-Normandie,
UMR 6072 GREYC, F-14032 Caen, France

Abstract. This paper presents an extension of the ILS algorithm, called `ID-ILS`, by introducing new local search devices that enforce an efficient tradeoff of intensification and diversification. Experiments performed on the DIMACS benchmarks show that our method is competitive with the best coloring algorithms.

1 Introduction

The *Graph Coloring Problem* (GCP) is to find the minimum number of colors required to color the vertices of a graph so that no edge has both endpoints with the same color. The GCP has received much attention in the literature, not only for its direct applications to many other real world problems [1,2], but also for its difficulty from complexity point of view. In fact, although many exact algorithms have been proposed for this problem (see [3]), such algorithms can only be used to solve small instances (up to 100 vertices). Therefore, heuristic algorithms are needed for larger instances. The best performing heuristic algorithms are local search methods (e.g., [4,5,6,7]) and hybrid algorithms that combines a local search with a population based method (e.g. [8,9,10,11]).

Iterated local search (ILS) [12] is a simple and effective type of metaheuristic that has been successfully applied on a wide range of problems. The basic principle of ILS consists in successively applying perturbations and local search to the current solution. The perturbation step plays a primary role because it drives ILS to explore different regions of the search space, in order to escape from the basin of attraction of the most recently visited local optima (*diversification effort*), while the goal of local search step is to focus more intensively within each promising region to converge towards a local optimum (*intensification effort*). However, as explained in [13], most LS algorithms handle diversity and intensity as two opposite objectives : as one gets more intensity, one can lose diversity. So, more coordination/balance is required between these two main objectives. The `Aspiration Plus` CLS (*Candidate List Strategy*) [14,15] is a promising mechanism proposed for the Tabu Search. It restricts the number of neighbors to examine for the next move, in order to control the intensification effort.

The goal of this work is to propose a new extension of the ILS algorithm, noted `ID-ILS`, by introducing in both steps of ILS new local search devices that enforce an efficient tradeoff of intensification and diversification. We performed

experiments on a set of challenging DIMACS graphs [16], and we have compared our results to six local search methods, as well as to three hybrid evolutionary algorithms HCA [9], MACOL [10] and MMT [11]. These results show that, our method clearly dominates the local search approaches and is competitive compared to the hybrid ones. Section 2 gives a synthetic overview of the GCP and presents the best performing algorithms for solving it. Section 3 describes our resolution approach ID-ILS and details their main components. Section 4 is devoted to experimentations. Finally, we conclude and draw some perspectives.

2 Graph Coloring Problem

2.1 Definitions and Notations

Given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a k-coloring of G is a function $c : V \rightarrow \{1, \ldots k\}$. The value $c(u)$ of a vertex u is called the color of u. An edge $(u, v) \in E$ is said *conflicting* if its vertices u and v have the same color. A k-coloring without conflicting edges is said *legal*, otherwise it is *illegal*. Let $s = [c(1), \ldots, c(|V|)]$ be a legal k-coloring, s can be represented by a partition of V into k disjoint subsets V^1, ..., V^k. We say that V^r is the *color class* r induced by s (i.e., the set of vertices having a color r in s). The objective function f counts the number of conflicting edges induced by s. The GCP is to determine the chromatic number $\chi(G)$ of G, i.e. the minimum value of k for which there is a k-coloring s of G such that $f(s) = 0$.

2.2 Metaheuristic Approaches to the GCP

TABUCOL [5] is one of the most famous local search algorithms proposed for the GCP. Morgenstern [7] proposed a complex algorithm, called MOR, based on *partial k-colorings*. A solution is a partition of vertices of G into k disjoint color classes $\{V^1, \ldots, V^k\}$. A specific class (i.e., V^{k+1}) is used to represent the set of uncolored vertices. A neighbor solution is obtained by moving an uncolored vertex u from V^{k+1} to a pre-existing color class V^h, and by moving to V^{k+1} all vertices in V^h that are adjacent to u. The complete legal k-coloring is obtained by emptying V^{k+1}.

In [9], an evolutionary algorithm, called HCA, combining an improved version of TABUCOL with a *Greedy Partitioning Crossover* operator (GPX) was proposed. GPX builds a partial legal k-coloring $\{V^1, \ldots, V^k\}$ by alternatively selecting from two parent solutions the class of maximum size to become color class V^i of the offspring. All vertices in this color class are then deleted from the parents. The remaining vertices are then assigned to a class randomly chosen. In [10], a similar approach was proposed, called MACOL, which extends GPX to use more than two parents for generating color classes of the offspring.

In [4], two TS methods (DYN-P.COL and FOO-P.COL), based on *partial k-colorings* were proposed. Finally, Hertz et al. [6] proposed an extension of the VNS algorithm, called *Variable Search Space* (VSS). The idea of VSS is to completely change the search space and to consider different objective functions for each space. They proposed VSS-Col, which moves between three different search spaces.

Algorithm 1. Pseudo-code algorithm for `ID-ILS`

```
   function ID-ILS(maxIter, maxneigh, maxMoves, nextneigh) ;
   begin
1      s ← genRandomSol(), i ← 1, b ← b_max;
2      while (i ≤ maxIter) do
3          i ← i + 1;
4          s' ← Perturbation(s, b) ;
5          s' ← LS(s', maxneigh, maxMoves, nextneigh) ;
6          if f(s') < f(s) then
7              s ← s', i ← 1;
8              b ← b_max;
9          else b ← updatePertubationSize(i);
10     return s;
   end
```

3 Intensification/Diversification-Driven ILS

3.1 Main Scheme of `ID-ILS`

`ID-ILS` extends `ILS` [12] by introducing new local search devices that enforce an efficient tradeoff of intensification and diversification. To achieve this goal, we first define an adaptive scheme to control the size for the perturbation (cf. Sect 3.2). Second, we make use a *candidate list strategy*, endowed with a diversification mechanism to exit from local minima (cf. Sect. 3.3). Algorithm 1 presents its pseudo-code. We denote by b_{max} the maximum perturbation size. It starts from an initial solution s which is randomly generated. The loop in lines 2 to 9 is performed until i number of consecutive iterations performed without improving s reaches `maxIter`. A new local optimum s' is obtained by the combination of a perturbation move of size b applied to the current solution s (line 4) with a local search procedure applied to the so obtained perturbed solution (line 5). If s' is better than s, it becomes the new current solution and i is reset to 1 (line 7);

3.2 Perturbation Step

Adaptive Perturbation. As explained in [12], the perturbation is just a collection of moves that complement those carried out by the local search. A weak perturbation is likely to get rapidly stuck in a deep local optima, whereas a strong perturbation is prone to be slow in convergence and similar to a randomized search. Achieving such a delicate balance is a challenge and certainly it is the key to success in ILS. We propose to exploit the search history to determine the perturbation size (i.e., b). In our approach, b proportionally decreases according to the value of i (see Algorithm 1). The main idea is to perform large perturbations each time the current solution s is improved, and to favor gradually small perturbations when s has not been improved for a long time. In fact, in our experiments we observed that the space of solutions has very distant solutions

that are nearly as good as the optimum. So, after getting a best coloring in one's neighbor solution by the local search procedure (i.e., intensification effort), one must go explore other regions of locally optimal solutions. This is achieved by using *large* perturbations (i.e., diversification effort). Initially, b is set to b_{max}. During the search, each time s is improved b is reset to b_{max} (line 8); otherwise it is decreased whenever i is increased, until reaching the value b_{min} (line 9).

Perturbation Schemes. Our perturbation operator consists of changing the color of some *conflicting vertices* in s. Let us note by $neighbors(u)$ the set of all vertices adjacent to u and by $\mathcal{X}(s)$ the set of conflicting vertices in s. We randomly select a first vertex v_O in $\mathcal{X}(s)$ and move its original color to the best possible other one (i.e. the new color is chosen among those producing the smallest number of conflicts). Let s_1 be the new perturbed solution. If s_1 increases the number of conflicts, we randomly select a new vertex among conflicted ones in $\mathcal{X}(s_1) \setminus \mathcal{X}(s)$ and assign to it the best possible new color. This process is repeated until a non-deteriorating move (i.e., that does not increase the number of conflicts) is found. In this way, we only accept moves that deacrese as small as possible the solution quality. To favor the diversification capability, we prevent changing the color of a vertex more than once. This sequence of moves are successively applied with b different v_O. This perturbation operator is noted *ConflictVar* (P_1). This operator, which is based on random choices, will change the current solution in an unpredictable way. The result will be, most likely, a worse solution, and many times, much worse. We propose to exploit information from the topology of the constraints graph to guide the perturbation operator towards more promising regions [17,18]. We propose new perturbations :

a) *ConflictVar Chain* (P_2): We first randomly select an initial vertex v_{init} in $\mathcal{X}(s)$ and move it into the best possible other color class V^i. Let s_1 be the new perturbed solution. If s_1 increases the number of conflicts, we randomly select a new vertex u among conflicted ones in $(\mathcal{X}(s_1) \setminus \mathcal{X}(s)) \cap V^i$ and assign to it the best possible new color class V^j. This sequence of moves is achieved until a non-deteriorating move is found. This process is repeated by successively applying such sequences of changes with b different v_{init}.

b) *ConflictVar Connected Centers* (P_3): We randomly select a first vertex v_{cc} noted "connected center" in $\mathcal{X}(s)$ and move it into the best possible other color class V^i. Then, for each conflicting vertex v_c in $neighbors(v_{cc}) \cap V^i$, we move it into the best possible other color class V^j, and we assign the best possible color to every new conflicting vertex in $neighbors(v_c) \cap V^j$. This sequence of moves are successively applied with b different v_{cc}.

c) *Conflict ColorClass* (P_4): We first select a color class V^i having the highest number of conflicting vertices (ties are randomly broken) and move each of its vertices into the best possible other color class. Let s_1 be the new perturbed solution. Then, we select randomly b new conflicting vertices from $\mathcal{X}(s_1) \setminus \mathcal{X}(s)$ and move each of them into a best possible other color. A *tabu list* is used to forbid selecting the color class V^i for the next l iterations of `ID-ILS`, with $l = 0.6 \times |\mathcal{X}(s)| + rand(0, 9)$ [9], where $rand(0, 9)$ is a function providing a random number in $\{0, \ldots, 9\}$.

3.3 Local Search Step

Our local search procedure is an extension of `TABUCOL`. Algorithm 2 shows its pseudo-code. We use a neighborhood defined by the 1-flip move, which consists of changing the original color class $V^{c(v)}$ of a single conflicting vertex v to its *best* possible new color class V^i ($c(v) \neq i$). (s, v, i) will denote this move (lines 7 to 8). Once a move is performed, vertex v is forbidden to move back to its previous color $c(v)$ for the next T iterations (line 19). After preliminary experiments, the *tabu tenure* T was fixed to 20. At each iteration, it determines the best neighbor s' of the current solution s such that either s' is a non-tabu solution or $f(s') < f(s^*)$, where s^* is the best solution found so. Our stopping condition is based on a total number of iterations (`stopIter`). Initially, `stopIter` = `maxMoves` (line 1). Each time a best solution is found, `stopIter` is increased by the current number of iterations performed (lines 20 to 21).

The most critical part for local search methods concerns the neighborhood exploration, and more exactly: (i) the number of candidate neighbors to visit, and (ii) the way of selecting the next move among these candidates. Indeed, the number of candidates should be large enough to focus more intensively on regions found to be good. However, it should be small enough to prevent examining a large set of candidates and thus to allow the search to exit from local optima more quickly. Moreover, the way of selecting the next move should drive the search towards unexplored regions in the solution space. To address these issues, we make use a *candidate list strategy* (CLS) to manage the neighborhood exploration. Two parameters are defined : (a) `maxneigh` which is the maximum number of candidate neighbors studied in every move. This CLS manages the `maxneigh` candidates so as to obtain a good tradeoff between intensification and diversification efforts; and (b) `nextneigh` which is a diversification devise to jump out of local minima. Two variants perform this diversification process. In the first variant, `ID-ILS(first)`, where `nextneigh` is set to `first`, the first neighbor among the `maxneigh` non-accepted candidates is selected (i.e., `s_first`). In the second variant, `ID-ILS(best)`, where `nextneigh` is set to `best`, the best neighbor with a lowest cost among the `maxneigh` non-accepted candidates is selected (i.e., `s_best`). The loop in lines 6 to 16 is performed until a better solution s' which improves s^* is obtained or no improvement has been made after `maxneigh` iterations. According to the value of `nextneigh`, the next neighbor solution is selected in lines 17 and 18 from the rejected candidates.

4 Experimental Results

In this section, we report experimental results over different graph types from the DIMACS benchmarks. For some very difficult graphs, we considered a set of k-coloring instances for different values of k. As experiments have been run on various machines, we will report (when it is possible), normalized[1] CPU times.

[1] For a machine κ times slower than ours, reported CPU times will be divided by κ.

Algorithm 2. Pseudo-code algorithm for LS

```
function LS (s, maxneigh, maxMoves, nextneigh) ;
begin
1   stopIter ← maxMoves;
2   s_first ← ∅,  s_best ← ∅,  f(s_best) ← ⊤,  i ← 1,  s* ← s;
3   while (i ≤ stopIter) do
4       i ← i + 1,  nbtries ← 1;
5       firstfound ← false, done ← false;
6       while (nbtries ≤ maxneigh) and not(done) do
7           v ← randomConflictVertex(s);
8           s' ← getNeighbor(s, v);
9           if not(firstfound) then
10              firstfound ← true,  s_first ← s';
11          if (not(done) and (v, s'[v]) ∉ tabulist) or (f(s') < f(s*)) then
12              s ← s',  done ← true;
13          else
14              if not(done) and (nextneigh = best) and (f(s') < f(s_best))
                then
15                  s_best ← s' ;
16          nbtries ← nbtries + 1;
17      if not(done) and (nextneigh = first) then s ← s_first ;
18      if not(done) and (nextneigh = best) then s ← s_best ;
19      insert (v, c(v)) in tabulist and make (v, c(v)) tabu for T iterations;
20      if f(s) < f(s*) then
21          stopIter ← i + maxMoves,  s* ← s;
22  return s*
end
```

4.1 Problem Instances and Experimental Protocol

We experimented our algorithm on the following difficult graphs [16]:

- **8 DSJC$n.y$ graphs:** DSJCs are random graphs with n vertices and a density equal to $0.y$. We selected those with $n \in \{250, 500, 1000\}$ and $y \in \{1, 5, 9\}$.
- **2 DSJR$n.r$ graphs:** DSJRs are geometric random graphs. We selected those with $n = 500$ and $r \in \{1, 5\}$.
- **5 flatn_x_0 graphs:** flat graphs are quasi-random graphs. We selected the $flat300\_x\_0$, with $x \in \{26, 28\}$ and the $flat1000\_x\_0$, with $x \in \{50, 60, 76\}$.
- **4 len_x graphs:** the Leighton graphs are derived from scheduling, and have 450 vertices. We selected instances c and d, with $x \in \{15, 25\}$.
- **one latin square** graph ($latin\_square\_10$).

Based on preliminary testing, we used the following parameter settings: maxIter=$2*n$, maxMoves=$200,000$, maxneigh$\in \{50, 100, 150, 175, 200\}$, b_{min}=20,

and b_{max}=50 (except for P_1, where b_{max}=100). A set of 20 (or 10) runs per k-coloring instance has been performed on a 2GHz Intel Core 2 DUO with 2GB of RAM. We report the value of k for which a k-coloring was found, the number of successful runs ("succ. runs/total runs"), the average CPU time in seconds for successful runs and the average cost over the total runs. `ID-ILS` has been implemented in C++.

Table 1. Comparing the different perturbation schemes. The best results are in bold.

Instance	k	P_i	MaxN.	Succ.	Time	Avg
DSJC250.5	28	P_1	200	18/20	113.5	0.1
n=250		P_2	175	18/20	148.1	0.1
m=15668		P_3	150	20/20	118.9	0
k^*=28		**P_4**	**150**	**20/20**	**76.4**	**0**
DSJC250.9	72	**P_1**	50	20/20	8.5	0
n=250		P_2	100	20/20	9.8	0
m=27897		**P_3**	**50**	**20/20**	**8.4**	**0**
k^*=72		P_4	50	20/20	10.86	0
DSJC500.1	12	P_1	175	17/20	142.4	0.15
n=500		P_2	175	17/20	448.9	0.2
m=24916		P_3	100	13/20	303.4	0.5
k^*=12		**P_4**	**200**	**20/20**	**121.9**	**0**
DSJC500.5	48	P_1	200	1/10	2077.6	1.5
n=500		**P_2**	**100**	**2/10**	**4762.6**	**1.2**
m=125248		P_3	100	2/10	6445.6	1.6
k^*=48		P_4	50	1/10	2820.6	2.1
DSJC500.9	126	P_1	**150**	**18/20**	**2451**	**0.15**
n=500		P_2	50	11/20	3513	0.45
m=224874		P_3	175	14/20	5348.8	0.3
k^*=126		P_4	175	12/20	6093.4	0.4
DSJR500.1c	85	P_1	150	0/10	-	2
n=500		P_2	200	0/10	-	2
m=121275		P_3	50	1/10	152.2	1.8
k^*=85		**P_4**	**50**	**10/10**	**1713.5**	**0**
DSJR500.5	125	P_1	150	0/10	-	2.7
n=500		P_2	100	0/10	-	3.1
m=58862		P_3	100	0/10	-	1.4
k^*=122		**P_4**	**100**	**9/10**	**2702.6**	**0.15**
	124	P_1	175	0/10	-	3.6
		P_2	100	0/10	-	3.9
		P_3	175	0/10	-	1.8
		P_4	**175**	**1/10**	**297.1**	**1.4**
DSJC1000.1	21	P_1	50	20/20	3.67	0
n=1000		P_2	50	20/20	3.1	0
m=49629		P_3	50	20/20	3.3	0
k^*=20		**P_4**	**50**	**20/20**	**3.08**	**0**
DSJC1000.5	88	P_1	150	20/20	1546.4	0
n=1000		P_2	50	20/20	1322.9	0
m=499652		**P_3**	**100**	**20/20**	**852.6**	**0**
k^*=83		P_4	50	20/20	1116.1	0
	86	P_1	150	0/10	-	5.5
		P_2	**100**	**2/10**	**4998.6**	**1.7**
		P_3	50	0/10	-	19.3
		P_4	175	2/10	27,071.2	4.8
le450_15c	15	P_1	25	20/20	0.6	0
n=450		P_2	25	20/20	0.6	0
m=16680		P_3	25	20/20	0.4	0
k^*=**15**		**P_4**	**25**	**20/20**	**0.3**	**0**

Instance	k	P_i	MaxN.	Succ.	Time	Avg
le450_15d	15	P_1	150	20/20	2.9	0
n=450		P_2	50	20/20	6.4	0
m=16750		P_3	150	19/20	6.7	0.05
k^*=**15**		P_4	50	20/20	3.2	0
le450_25c	26	P_1	100	7/25	19.9	0.8
n=450		**P_2**	**100**	**25/25**	**10**	**0**
m=17343		P_3	150	18/25	12	0.32
k^*=**25**		P_4	100	12/25	19.4	0.68
le450_25d	26	P_1	100	11/20	61	0.56
n=450		**P_2**	**100**	**20/20**	**11.3**	**0**
m=17425		P_3	100	17/20	9	0.4
k^*=**25**		P_4	150	7/20	17.2	0.88
flat300_26_0	26	P_1	50	20/20	4.6	0
n=300		P_2	50	20/20	4.2	0
m=21633		**P_3**	**50**	**20/20**	**3.2**	**0**
k^*=**26**		P_4	50	20/20	10.3	0
	31	P_1	150	15/20	192	0.25
		P_2	150	17/20	160.9	0.15
		P_3	175	17/20	239.1	0.15
		P_4	**200**	**20/20**	**185**	**0**
Flat300_28_0	30	P_1	150	10/20	1399.43	1.7
n=300		**P_2**	**100**	**10/20**	**1273.6**	**2**
m=21695		P_3	150	9/20	1932.9	2
k^*=**28**		P_4	150	8/20	1344.8	2.2
	29	P_1	50	2/20	547.3	9.35
		P_2	200	4/20	2075.8	8.3
		P_3	**200**	**4/20**	**1612.7**	**8.2**
		P_4	200	3/20	1190.9	8.75
flat1000_50	50	P_1	50	0/20	2379.8	-
n=1000		P_2	50	0/20	2305.1	-
m=224874		P_3	50	0/20	1977.4	-
k^*=**50**		**P_4**	**50**	**20/20**	**2858.1**	**0**
flat1000_60	60	P_1	50	0/20	-	-
n=1000		P_2	50	0/20	-	-
m=245830		P_3	50	0/20	-	-
k^*=**60**		**P_4**	**50**	**20/20**	**13,854**	**0**
flat1000_76_0	86	P_1	100	2/20	5750.6	**1.77**
n=1000		P_2	100	1/20	29,765	2.8
m=246708		P_3	150	0/20	-	9.88
k^*=82		**P_4**	100	**17/20**	**20,579**	0.15
DSJC1000.9	224	P_1	175	8/10	31,461	0.2
n=1000		P_2	50	5/10	13,384	0.7
m=449449		P_3	50	3/10	20,671	1.2
k^*=223		**P_4**	**150**	**9/10**	**32,598**	**0.1**
latin_square	100	P_1	50	0/20	-	3.35
n=900		P_2	50	0/20	-	3.35
m=307350		P_3	50	0/20	-	3.75
k^*=98		**P_4**	**100**	**15/20**	**12,812.1**	**0.3**

4.2 Comparing the Different Perturbation Schemes

Our first experiment aims to evaluate the effectiveness of our perturbation schemes. Table 1 reports the detailed results of `ID-ILS(first)`. Column 1 gives the features of each instance: its name, the number of vertices (n), the number of edges (m) and the value of the best known coloring (k^*) (in bold when it is the proven optimal value). Column 3 denotes the different perturbations (P_i). Column 4 reports the best setting for `maxneigh`. A score (b-s-w) is assigned to each P_i, corresponding to the number of k-colorings for which P_i gets

Table 2. Comparison among ID/TS and VSS-Col. Best results are in bold. Column P_2 (resp. P_4) refers to the results obtained by ID-ILS(first) with P_2 (resp. P_4).

Instance	k^*	k	ID-ILS(first+P_4)		ID-ILS(first+P_2)		VSS-Col		ID/TS		
			Succ.	Time	Succ.	Time	Succ.	Time	k_{best}	Time	Avg.
DSJC250.5	28	28	20/20	76	18/20	148	-	-	28 (1)	1241	0.8
DSJC250.9	72	72	20/20	11	20/20	10	-	-	72 (5)	15	0
DSJC500.1	12	12	20/20	122	17/20	449	10/10	97	12 (5)	1465	0
DSJC500.5	48	48	1/10	2820	2/10	4762	**3/10**	1331	50 (3)	2378	0.4
		49	12/20	1894	14/20	889	**10/10**	162			
DSJC500.9	126	126	12/20	6094	11/20	3513	**8/10**	1686	127 (1)	3435	1
		127	20/20	194	20/20	173	10/10	169			
DSJR500.1c	85	85	**10/10**	1713	0/10	-	**9/10**	736	85 (0)	-	1.4
DSJR500.5	122	124	**1/10**	297	0/10	-	0/10	-	125 (0)	-	3.4
		125	**9/10**	2702	0/10	-	0/10	-			
DSJC1000.1	20	20	0/20	-	0/20	-	**3/10**	2396	21 (5)	4	0
		21	20/20	3	20/20	3	10/10	11			
DSJC1000.5	83	86	**2/10**	27,071	**2/10**	4998	0/10	-	90 (1)	2711	1.2
		88	20/20	1116	20/20	1323	8/10	2028			
DSJC1000.9	223	224	**9/10**	32,598	**5/10**	13,384	1/10	3326	228 (1)	5707	1.2
		225	**20/20**	20,816	**20/20**	1546	5/10	1484			
le450_15c	15	15	20/20	0	20/20	0	10/10	6	15 (5)	3	0
le450_15d	15	15	20/20	3	20/20	6	10/10	44	15 (5)	5	0
le450_25c	25	26	12/25	19	**25/25**	10	10/10	1	26 (3)	6	0.4
le450_25d	25	26	7/20	17	**20/20**	11	10/10	1	26 (1)	279	0.8
flat300_28_0	28	29	**3/20**	1191	**4/20**	2075	1/10	867	31 (1)	486	1.4
		30	**8/20**	1344	**10/20**	1273	2/10	2666			
		31	**20/20**	185	17/20	160	10/10	39			
flat1000_50_0	50	50	**20/20**	2858	0/20	-	10/10	318	60 (0)	-	484.4
flat1000_60_0	60	60	**20/20**	13,854	0/20	-	10/10	694	70 (0)	-	223.8
flat1000_76_0	82	86	**17/20**	20,579	1/20	29,765	0/10	-	90 (5)	1190	0
		87	**20/20**	1204	**20/20**	1780	4/6	1689			
		88	NA	NA	NA	NA	10/10	1155			
			P_2		P_4		P_4	P_2	P_4	P_2	
Better			**4**		0		**3**	2	**10**	8	
Equal			**14**		14		**12**	9	8	8	
Worse			**0**		4		**1**	4	**0**	0	

respectively better (2^{nd} better in parentheses), equal (100%) and worse success rates than the other perturbations. From Table 1, the following remarks are drown :

- Perturbations based on the topology of the constraints graph (except P_3) are clearly more relevant. Both P_1 (*score*: 2(5)-6-10) and P_3 (*score*: 3(6)-6-9) perform similarly, with a slight advantage to P_3. This can be explained by the fact that P_3 performs perturbations only on a very limited part of the graph, whereas the random character of P_1 allows more diversification in the perturbation step, which helps to find better solutions.
- P_2 (*score*: 6(3)-6-7) and P_4 (*score*: 12(2)-6-4) outperform P_3. Both perturbations find solutions with better success rates respectively for 6 and 12 coloring instances among 24, whereas P_3 obtains best success rates for 3 coloring instances. Indeed, perturbations P_2 and P_4 allow a more "aggressive" diversification by performing perturbations on different connected subparts of the graph.
- Finally, P_4 clearly dominates P_2: P_2 obtains better success rates on 5 k-coloring instances while P_4 outperforms P_2 on 11 k-coloring instances.

4.3 Comparison with Two Local Search Methods

We have compared ID-ILS(first) using the two best perturbations P_2 and P_4, with two local search methods:

(i) `ID/TS`, a variant of TS endowed with our CLS. We made experiments with `nextneigh` set to `first` and `maxneigh` $\in \{50, 100, 150, 175, 200\}$. For each value of k, and each trial, `ID/TS` is run 20 times with `maxMoves` set to $1,500,000$. If no legal k-coloring is found, then it is run 10 times with `maxMoves` equal to $5,000,000$. A set of 5 trials per k-coloring is performed.

(ii) `VSS-Col` which is one of the most performing among local search coloring algorithms [6]. The reason for comparing `ID-ILS` with `VSS-Col` is that both methods are very close. However, `VSS-Col` considers different search spaces, each one being associated with a set of neighborhoods.

Table 2 compares performances of the four methods. Results for `VSS-Col` are taken from [6] and correspond to those obtained with a time limit of 1h on a 2 GHz Pentium 4, with 512 MB of RAM. For `ID/TS`, we report the best value of k (k_{best}) found, the number of successful runs (in parentheses), the average CPU time in seconds for successful runs and the average cost among the five trials. The last three rows show the summary of the comparisons. The rows *better*, *equal* and *worse* gives respectively the number of graphs for which our method gets better, equal and worse colorings than the other algorithms.

`ID-ILS(first+P4)` is clearly the best one. From these results, we observe that the two variant of `ID-ILS(first)` outperform `ID/TS`, particularly on large graphs, where better colorings have been found on at least eight large graphs. For the two flat1000_50&60, `ID/TS` was not able to find a legal coloring even for high values of k. On the eight remaining graphs, the two methods find solutions of the same quality, but `ID-ILS(first)` provides better success rates.

When comparing with the results of `VSS-Col`, `ID-ILS(first+P4)` gets better solutions on three graphs, with colorings using respectively 2, 2 and 1 less colors, and it is worse on one graph. Both methods obtain the same colorings on 12 graphs. However, if we compare the success rates, `ID-ILS(first+P4)` performs better than `VSS-Col` on three graphs. Both algorithms find the same colorings, with the same success rate on five graphs, but `VSS-Col` is generally faster, except for le450_15c&15d, where `ID-ILS(first+P4)` find optimal colorings very quickly. So, `ID-ILS(first+P4)` can be considered as more effective than `VSS-Col`.

4.4 Comparison with the Most Effective Algorithms

In this section we compare our method with the most performing algorithms for the GCP: four local search algorithms (`MOR` [7], `ILS` [19] and `DYN/FOO-P.COL` [4]) and three hybrid evolutionary methods (`HCA` [9], `MACOL` [10] and `MMT` [11]). However, we do not report the CPU times because the conditions of experimentation are not equivalent. So, comparisons must be done with care. Results are given so that the reader may have a baseline by which he may evaluate `ID-ILS`.

If we compare the results of `ID-ILS(first+P4)` with those of local search methods, one easily observes that our method clearly dominates these local search algorithms (see last three rows of Table 3). Indeed, our method obtains worse results for **at most three graphs** while better results are obtained for **at least six graphs**, except for `ILS`, where our approach obtains better results

Table 3. Comparison with the state-of-the-art algorithms

Instance	MOR	ILS	HCA		FOO-P.COL		DYN-P.COL		MACOL		MMT	ID-ILS	
	k_{best}	k_{best}	Succ.	k	Succ.	k	Succ.	k	Succ.	k	k_{best}	Succ.	k
DSJC250.5	28	28	9/10	28					20/20	8	28	20/20	28
DSJC250.9	-	-							10/10	72	72	20/20	72
DSJC500.1	12	12			23/50	12	50/50	12	20/20	12	12	20/20	12
DSJC500.5	**49**	**49**	5/10	48	50/50	**50**	1/50	**49**	20/20	48	48	1/10	48
DSJC500.9	**128**	126			48/50	**128**	1/50	127	20/20	126	**127**	12/20	126
DSJR500.1c	85	-			50/50	85	3/50	85	20/20	85	85	10/10	85
DSJR500.5	*123*	124			24/50	**128**	28/50	**126**	11/20	*122*	*122*	1/10	124
DSJC1000.1	21	-	-	*20*	50/50	21	3/50	*20*	20/20	20	*20*	20/20	21
DSJC1000.5	**88**	**89**	-	*83*	5/50	**89**	6/50	**89**	20/20	*83*	*84*	2/10	86
DSJC1000.9	**226**	-	-	224	30/50	**228**	30/50	**228**	18/20	*223*	**225**	9/10	224
le450_15c	15	15	6/10	15	50/50	15	50/50	15	20/20	15	15	20/20	15
le450_15d	15	15			50/50	15	50/50	15	20/20	15	15	20/20	15
le450_25c	*25*	26	10/10	26	50/50	**27**	50/50	**27**	20/20	*25*	*25*	12/20	26
le450_25d	*25*	26			50/50	**27**	50/50	**27**	20/20	*25*	*25*	7/20	26
flat300_26_0	26	26							20/20	26	26	20/20	26
flat300_28_0	**31**	**31**	6/10	**31**	35/50	*28*	13/50	*28*	15/20	29	**31**	3/20	29
flat1000_50_0	50	-			50/50	50	50/50	50	20/20	50	50	20/20	50
flat1000_60_0	60	-			50/50	60	50/50	60	20/20	60	60	20/20	60
flat1000_76_0	**89**	-	4/5	*83*	10/50	**88**	9/50	**88**	20/20	*82*	*83*	17/20	86
latin_square	-	*99*							5/20	*99*	**101**	15/20	100
Better	**6**	**3**		**1**		**8**		**8**		**0**	**4**		
Equal	9	9		5		7		6		12	10		
Worse	**3**	**1**		**3**		**1**		**2**		**8**	**6**		

for **three graphs**. For DSJC500.9 (resp. DSJC1000.9), ID-ILS(first+P_4), ILS and VSS-Col are the only algorithms that can reach 126-coloring. Detailed comparisons are given bellow:

- ID-ILS(first+P_4) is better than MOR on six graphs and worse on three graphs (DSJR500.5, le450_25c and le450_25d).
- ID-ILS(first+P_4) is better than ILS on three graphs and worse on one graph (latin_square). This comparison is very informative as well and shows the importance of our perturbation scheme P_4 as well as of the CLS.
- ID-ILS(first+P_4) is better than DYN/FOO-P.COL on eight graphs and worse on two/one graphs. For flat300_28, there are only few algorithms in the literature that can reach 28-coloring.
- ID-ILS(first+P_4) is better than VSS-Col on three graphs (six graphs if we consider the success rates) and worse on two graphs.

When comparing with the results of the two hybrid evolutionary algorithms HCA and MMT, one observes that ID-ILS(first+P_4) is competitive. Indeed, our method is better than HCA on flat300_28_0 and worse on three graphs, better than MMT on four graphs (DSJC500.9, DSJC1000.9, flat300_28_0 and latin_square) and worse on six graphs. If we compare with the results of MACOL, one easily observes

Table 4. Impact of the CLS and the `nextneigh` parameter on the performance of `ID-ILS`. We report in parentheses the best cost for the unsuccessful runs.

Instance	k	P_i	ID-ILS(first)			ID-ILS(best)			ILS/TS		
			Succ.	Time	Avg.	Succ.	Time	Avg.	Succ.	Time	Avg.
le450_15c	15	P_2	20/20	0.6	0	10/20	9.1	2.15	5/20	286	7.5
		P_4	20/20	0.3	0	7/20	17.4	3.1	0/20	-	24.7(2)
le450_15d	15	P_2	20/20	6.4	0	11/20	76.4	1.05	2/20	468.5	11.7
		P_4	20/20	3.2	0	5/20	43.6	2.7	0/20	-	31.5(13)
le450_25c	26	P_2	25/25	10	0	0/20	-	6.1(4)	5/20	295.6	2.2
		P_4	12/25	19.4	0.68	0/20	-	6.6(5)	0/20	-	6.6(2)
le450_25d	26	P_2	20/20	11.3	0	0/20	-	5.4(3)	7/20	233	2.2
		P_4	7/20	17.2	0.88	0/20	-	6.6(5)	2/20	79.5	5.6
DSJC250.5	28	P_2	18/20	148.1	0.1	0/20	-	3.9(2)	3/20	1436	2.7
		P_4	20/20	76.4	0	0/20	-	5(4)	1/20	149	5.5
DSJC500.9	127	P_2	20/20	173.3	0	0/20	-	3.95(2)	1/20	5523	3.85
		P_4	19/20	194.2	0.05	0/20	-	5.05(3)	1/20	687	5.3

that `MACOL` clearly outperforms `ID-ILS(first+`P_4`)`. However, our method should be considered as a simple local search method which uses very simple diversification devices, while `HCA`, `MMT` and `MACOL` are much more complex algorithms, with sophisticated ingredients finely tuned.

4.5 Analysis of the Parameters of `ID-ILS`

The aim of this section is to study the impact of the two local search devices, on the performance of `ID-ILS`.

(a) Impact of `nextneigh`. Table 4 compares the results of the two variants of `ID-ILS`. The impact of setting `nextneigh` to `first` has a strong influence on the effectiveness of `ID-ILS`, particularly on le450_25 and DSJC500.9, for which several orders of magnitude are gained. The poor results of `ID-ILS(best)` are probably due to the fact that selecting the best neighbor among the `maxneigh` non-accepted candidates leads `ID-ILS` to get stuck in a deep local optimum looping between already visited areas. This surprising result shows that setting `nextneigh` to `first` clearly favors diversification.

(b) Impact of the CLS. In this study, we compared `ID-ILS` with a version of `ILS` using `TABUCOL` without any CLS (noted `ILS/TS`). As showed in Table 4, `ID-ILS(first)` is clearly more relevant. We can observe that the impact of the CLS when `nextneigh` = `best` is not systematic. Indeed, compared to `ILS/TS`, `ID-ILS(best)` performs very well on le450_15c&d, but on the other instances, `ILS/TS` is very effective. These results highlight the importance of the CLS device for enforcing a tradeoff between intensification and diversification.

5 Conclusions

In this paper, we have proposed a new extension of the `ILS` algorithm, noted `ID-ILS`, by introducing new devices that enforce an efficient tradeoff between intensification and diversification. For the graph coloring problem, we have defined new perturbation schemes that exploit information from the topology of the constraints graph. Experimentations, carried out on a set of DIMACS graphs show that our method is very competitive with the current best hybrid approaches. Let

us mention that our approach is generic (with some adaptations for perturbation schemes) and could be applied to other difficult optimization problems. We are currently investigating such a direction on Radio link frequency assignment (RLFAP), and Car Sequencing problems. We also intend to study the impact of the perturbation size on intensification/diversification.

References

1. Burke, E.K., Marecek, J., Parkes, A.J., Rudová, H.: A supernodal formulation of vertex colouring with applications in course timetabling. Annals OR 179(1), 105–130 (2010)
2. Gamache, M., Hertz, A., Ouellet, J.O.: A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. Computers & OR 34(8), 2384–2395 (2007)
3. Malaguti, E., Toth, P.: A survey on vertex coloring problems. Intl. Trans. in Op. Res. 17(1), 1475–3995 (2010)
4. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers & OR 35(3), 960–975 (2008)
5. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing 39(4), 345–351 (1987)
6. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Applied Mathematics 156(13), 2551–2560 (2008)
7. Morgenstern, C.: Distributed coloration neighborhood search. DIMACS Series, vol. 26, pp. 335–357. Providence, RI (1996)
8. Fleurent, C., Ferland, J.: Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 63(3), 437–461 (1996)
9. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. 3(4), 379–397 (1999)
10. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. European Journal of Operational Research 203(1), 241–250 (2010)
11. Malaguti, E., Monaci, M., Toth, P.: A metaheuristic approach for the vertex coloring problem. INFORMS Journal on Computing 20(2), 302–316 (2008)
12. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of Metaheuristics, vol. 146, pp. 363–397. Springer, New York (2010)
13. Linhares, A., Yanasse, H.: Search intensity versus search diversity: a false trade off? Appl. Intell. 32(3), 279–291 (2010)
14. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
15. Neveu, B., Trombettoni, G., Glover, F.: ID Walk: A Candidate List Strategy with a Simple Diversification Device. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 423–437. Springer, Heidelberg (2004)
16. Trick, M.: Computational symposium: Graph coloring and its generalizations. Cornell University, Ithaca, NY (2002), http://mat.gsia.cmu.edu/COLOR02/
17. Avanthay, C., Hertz, A., Zufferey, N.: A variable neighborhood search for graph coloring. European Journal of Operational Research 151(2), 379–388 (2003)
18. Loudni, S., Boizumault, P., Levasseur, N.: Advanced generic neighborhood heuristics for vns. Eng. Appl. of AI 23(5), 736–764 (2010)
19. Chiarandini, M., Stützle, T.: An application of iterated local search to graph coloring. In: Proceedings of the Comput. Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA, pp. 112–125 (2002)