ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/225214687

Hybrid Evolutionary Algorithms for Graph Coloring

Article *in* Journal of Combinatorial Optimization · January 1999

DOI: 10.1023/A:1009823419804 · Source: DBLP

CITATIONS
427

READS
901

2 authors:

Philippe Galinier
Polytechnique Montréal
58 PUBLICATIONS **1,594** CITATIONS

SEE PROFILE

Jin-Kao Hao
University of Angers
387 PUBLICATIONS **8,068** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Sports league scheduling View project

Metaheuristic Optimisation View project

All content following this page was uploaded by Jin-Kao Hao on 03 June 2014.

The user has requested enhancement of the downloaded file.

Hybrid Evolutionary Algorithms for Graph Coloring

Philippe Galinier and Jin-Kao Hao*
LGI2P
EMA-EERIE
Parc Scientifique Georges Besse
F-30000 Nîmes
France
email: {galinier,hao}@eerie.fr

Submitted to the *Journal of Combinatorial Optimization*

October 14, 1998

Abstract

A recent and very promising approach for combinatorial optimization is to embed local search into the framework of evolutionary algorithms. In this paper, we present such hybrid algorithms for the graph coloring problem. These algorithms combine a new class of highly specialized crossover operators and a well-known tabu search algorithm. Experiments of such a hybrid algorithm are carried out on large DIMACS Challenge benchmark graphs. Results prove very competitive with and even better than those of state-of-the-art algorithms. Analysis of the behavior of the algorithm sheds light on ways to further improvement.

Keywords: Graph coloring, solution recombination, tabu search, combinatorial optimization.

1 Introduction

A recent and very promising approach for combinatorial optimization is to embed local search into the framework of population based evolutionary algorithms, leading to hybrid evolutionary algorithms (HEA). Such an algorithm is essentially based on two key elements: an efficient local search (LS) operator and a highly

*Corresponding author

specialized crossover operator. The basic idea consists in using the crossover operator to create new and potentially interesting configurations which are then improved by the LS operator. Typically, a HEA begins with a set of configurations (population) and then repeats an iterative process for a fixed number of times (generations). At each generation, two configurations are selected to serve as parents. The crossover operator is applied to the parents to generate a new configuration (offspring). Then the LS operator is used to improve the offspring before inserting the latter into the population.

To develop a HEA for an optimization problem, both the LS and crossover operators must be designed carefully. Nevertheless, it is usually more difficult to develop a meaningful crossover operator than a LS operator. Indeed, while various good LS algorithms are available for many well-known optimization problems, little is known concerning the design of a good crossover. In general, designing a crossover requires first the identification of some "good properties" of the problem which must be transmitted from parents to offspring and then the development of an appropriate recombination mechanism.Note that random crossovers used in standard genetic algorithms are meaningless for optimization problems.

HEAs have recently been applied to some well-known NP-hard combinatorial problems such as the traveling salesman problem [23, 10], the quadratic assignment problem [10, 21] and the bin-packing problem [8]. The results obtained with these HEAs are quite impressive. Indeed, they compete favorably with the best algorithms for these problems on well-known benchmarks. These results constitute a very strong indicator that HEAs are among the most powerful paradigms for hard combinatorial optimization.

In this paper, we are interested in tackling with HEAs a well-known combinatorial optimization problem: the graph coloring problem (GCP). The GCP consists in coloring the vertices of a given graph with a minimal number of colors (chromatic number, denoted by $\chi(G)$) with the constraint that two adjacent vertices receive different colors. This problem allows one to model many practical and important applications [20, 11, 2]. Unfortunately, the problem is very difficult to solve because it belongs to the NP-complete family [12]. Even approximating the problem prove to be difficult: no polynomial algorithm is able to color any graph using a number of colors smaller than $2 * \chi(G)$ (unless P=NP). Empirical results on coloring random graphs confirmed the hardness of the problem. For instance, Johnson et al. observed that no known exact algorithm is able to color optimally even relatively small (90 vertices) random graphs of density 0.5 [18]. Given that there is almost no hope to find an efficient exact algorithm, various heuristic algorithms have been developed including greedy algorithms, local search algorithms, and hybrid algorithms.

This paper aims at developing powerful hybrid evolutionary algorithms to find sub-optimal solutions of high quality for the GCP. To achieve this, we devise a new class of highly specialized crossover operators. These crossovers, combined with a well-known tabu search algorithm, lead to a class of HEAs. We choose to experiment a particular HEA. Results on large benchmark graphs prove very competitive with and even better than those of state-of-the-art algorithms. Also, we study the influence of some critical parameters of the HEA and analyze its behavior, shedding light on several ways to further improve the performance of

these HEAs.

The paper is organized as follows. We first review heuristic methods for graph coloring (Section 2). Then, we present the general principles of the specialized crossovers (Section 3). Section 4 presents the different components of our hybrid algorithms. Sections 5-6 present computational results and analyze the behavior of the algorithm. The last two sections give some discussions and conclusions.

2 Heuristic methods for graph coloring

Let $G = (V, E)$ be an indirected graph with a vertex set V and an edge set E. A subset of G is called an independent set if no two adjacent vertices belong to it. A k-coloring of G is a partition of V into k independent sets (called proper color classes). An optimal coloring of G is a k-coloring with the smallest possible k (the chromatic number $\chi(G)$ of G). The graph coloring problem is to find an optimal coloring for a given graph G.

Many heuristics have been proposed for this problem. We review below the main classes of known heuristics.

Greedy constructive methods: The principle of a greedy constructive method is to color successively the vertices of the graph. At each step, a new vertex and a color for it are chosen according to some particular criteria. These methods are generally very fast but produce poor results. Two well-known techniques are the saturation algorithm DSATUR [1] and the Recursive Largest First algorithm [20].

Genetic algorithm (GA): The main principles of GAs are presented in [17, 14]. Davis proposed to code a coloring as an ordering of vertices. The ordering is decoded (transformed into a coloring) by using a greedy algorithm. This algorithm gives poor results [5]. No other work using pure GA has been reported since then. It is believed that pure GAs are not competitive for the problem. Recently, Falkenauer introduced a set of genetic operators for the so-called "grouping problems". These operators are applicable to the GCP which is a special grouping problem, but no results are available on benchmark graphs.

Local search (LS): Several LS algorithms have been proposed for the GCP and prove very successful. These algorithms are usually based on simulated annealing or tabu search. They differ mainly by their way of defining the search space, the cost function and the neighborhood function. Hertz and de Werra proposed a penalty approach and a simple neighborhood. They tested simulated annealing and tabu search [16, 3]. Johnson et al. compared three different neighborhoods using simulated annealing and presented extensive comparisons on random graphs [18]. A more powerful LS algorithm was recently reported in [22], which is based on a quite different neighborhood from the previous ones.

Hybrid algorithms: Hybrid algorithms integrating local search and crossovers were recently proposed in [9, 4]. [9] shows that crossovers can improve on the performance of LS, but this happens only for a few graphs and needs important computational efforts. Hybrid algorithms using a population and local search but without crossover are proposed in [22], producing excellent results on benchmark graphs.

Successive building of color classes: This particular strategy consists in

building successively different color classes by identifying each time a maximal independent set and removing its vertices from the graph. So the graph coloring problem is reduced to the maximal independent set problem. Different techniques to find maximal independent sets have been proposed for building successive color classes [18, 16, 22], including local search methods [9]. This strategy proves to be one of the most effective approaches for coloring large random graphs.

3 New crossovers for graph coloring

Standard genetic algorithms were initially defined as a general method based on blind genetic operators (crossover, mutation, ...) able to solve any problem whose search space is coded by bit strings. But it is now admitted that the standard GA is generally poor for solving optimization problems and that genetic operators, notably crossover, must incorporate specific domain knowledge [15]. More precisely, designing crossovers requires to identify properties that are meaningful for the problem and then to develop a recombination mechanism in such a way that these properties are transmitted from parents to offspring.

Concerning graph coloring, there are essentially two different approaches according to whether a coloring is considered to be an assignment of colors to vertices or to be a partition of vertices into color classes. Before presenting these approaches, the table below show their main characteristics.

	assignment approach	partition approach
configuration	assignment of colors to vertices $s : V \rightarrow \{1, \cdots, k\}$	partition of the vertices $s = \{V_1, \cdots, V_k\}$
elementary characteristic	couple vertex-color (v, i): color i assigned to vertex v	pair $\{v_1, v_2\}$ or set $\{v_1, \cdots, v_q\}$: vertices all included in a same V_i
crossover	assignment crossover $s(v) := s_1(v)$ or $s_2(v)$	partition crossover

Table 1: Assignment approache *v.s.* partition approach

The *assignment approach* considers a configuration (a proper or improper) coloring as an assignment of colors to vertices, *i.e.* a mapping $s : V \rightarrow \{1 \cdots k\}$ from the vertex set V into a set of k colors. So, it is natural to present a configuration s by a vector $(s(v_1), \cdots, s(v_n))$ of size $n = |V|$, each $s(v_i)$ being the color assigned to the vertex v_i. It is now possible to define an uniform assignment crossover as follows: given two parents $s_1 = (s_1(v_1), \cdots, s_1(v_n))$ and $s_2 = (s_2(v_1), \cdots, s_2(v_n))$, build the offspring $s = (s(v_1), \cdots, s(v_n))$ by performing for each vertex v either $s(v) := s_1(v)$ or $s(v) := s_2(v)$ with an equal probability 1/2.

Assignment crossovers have been proposed for coloring problem in [9, 4]. In [9], the uniform assignment crossover is enhanced in the following way: if a vertex v is conflicting in one of the two parents, it systematically receives the color assigned to v in the other parent.

We can observe that with the assignment approach, the property transmitted by crossovers is a couple (vertex, color): a particular vertex receives a particular

color. Nevertheless, such a couple, considered isolatedly, is meaningless for graph coloring since all the colors play exactly the same role.

For the coloring problem, it will be more appropriate and significant to consider a *pair* or a *set of vertices* belonging to a same class. This leads to a different approach, that we call *partition approach*. With this approach, a configuration is considered as a partition of vertices into color classes and a crossover is required to transmit color classes or subsets of color classes. Given this general principle, many different crossovers are now imaginable.

One possibility is to build first a partial configuration of maximum size from subclasses of the color classes of the two parents and then to complete it to obtain a complete configuration. More precisely, given two parents (colorings or partial colorings) $s_1 = \{V_1^1, \cdots, V_k^1\}$ and $s_2 = \{V_1^2, \cdots, V_k^2\}$, the partial configuration will be a set $\{V_1, \cdots, V_k\}$ of disjoint sets of vertices having the following properties:

- each subset V_i is included in a class of one of the two parents: $\forall i (1 \leq i \leq k)$ $\exists j : V_i \subseteq V_j^1$ or $\exists j : V_i \subseteq V_j^2$, hence all V_i are independent sets.
- the union of the V_i has a maximal size: $|\cup_{1 \leq i \leq k} V_i|$ is maximal.
- about the half of the V_i is imposed to be included in a class of parent 1 and the other half in a class of parent 2, because it is desirable to equilibrate the influence of the two parents.

One way to construct the partial configuration is to build each V_i successively in a greedy fashion: both parents are considered successively and we choose in the considered parent the class with the maximal number of unassigned vertices to become the next class V_i. Note that this crossover, named Greedy Partition Crossover (GPX), generates each time one offspring. The GPX crossover is presented in detail in Section 4. More partition crossovers will be presented in Section 7.1.

Let us mention now two other partition crossovers. The first one is presented in [7] where each V_i is built from the union of two classes, each class coming from one parent. The second one is a crossover defined for grouping problems and applicable to the GCP [8]. This crossover is different from the GPX: to build a V_i, two classes are *randomly* taken from the parents, contrary to the GPX where this choice is carefully realized with particular optimization criteria.

Finally, let us mention that we have carried out experiments to confirm the pertinence of the characteristic considered by partition crossovers (*i.e.* set of vertices), limiting here to pairs of vertices. We take a (random) graph and a fixed k and collect a sample of k-colorings. We look at the frequencies that two non-adjacent vertices are grouped into a same color class. Analysis of these frequencies discloses that some pairs of non-adjacent vertices are much more frequently grouped into a same class than others. This constitutes a positive indicator of the pertinence of the approach.

4 The hybrid coloring algorithm

To solve the GCP, *i.e.* to color a graph with a minimum possible number of colors, a particular approach consists in applying a coloring algorithm to look for

a k-coloring for a (sufficiently high) number of colors $k = k_0$. Then whenever a k-coloring is found, one re-applies the same algorithm to look for k-colorings with decreasing numbers of colors ($k = k_0 - 1, k_0 - 2, \cdots$). Therefore, the graph coloring problem is reduced to solving increasingly difficult k-coloring problems. In this section, we present the components of our Hybrid Coloring Algorithm (HCA) for finding k-colorings.

4.1 Search space and cost function

Recall that the k-coloring problem consists in coloring a graph $G = (V, E)$ with a fixed number of k colors. To solve a k-coloring problem, we consider the set of all possible partitions of V in k classes including those which are not proper k-colorings. Each partition is attributed a penalty equal to the number of edges having both endpoints in a same class. Therefore, a partition having a 0 penalty corresponds to a proper k-coloring (a solution). The purpose of the search algorithm is then to find a partition having a penalty as small as possible. So the k-coloring problem can be solved by solving the minimization problem (S, f) where:

- A configuration $s \in S$ is any partition $s = \{V_1, \cdots, V_k\}$ of V in k subsets.
- $\forall s \in S, f(s) = |\{e \in E : \text{both endpoints of } e \text{ are in the same } V_i \in s|\}$.

In the following we call configuration any element of the search space S and reserve the word solution for a proper k-coloring.

4.2 The general procedure

In HCA, the population P is a set of configurations having a fixed constant size $|P|$. We present below the general algorithm:

Data : *graph* $G = (V, E)$, *integer* k
Result : *the best configuration found*
begin
  P=*InitPopulation*($|P|$)
  while *not Stop-Condition()* **do**
    (s1,s2)=*ChooseParents*(P)
    s=*Crossover*(s1,s2)
    s=*LocalSearch*(s, L)
    P=*UpdatePopulation*(P,s)
end

Algorithm : The hybrid coloring algorithm

The algorithm first builds an initial population of configurations (*InitPopulation*) and then performs a series of cycles called *generations*. At each generation, two configurations s_1 and s_2 are chosen in the population (*ChooseParents*). A crossover is then used to produce an offspring s from s_1 and s_2 (*Crossover*). The LS operator is applied to improve s for a fixed number L of iterations (*LocalSearch*). Finally, the improved configuration s is inserted in the population

by replacing another configuration (*UpdatePopulation*). This process repeats until a stop condition is verified, usually when a pre-fixed number Max_{Iter} of iterations is reached. Note however that the algorithm may stop before reaching Max_{Iter}, if the population diversity becomes too small (see Section 6.2).

Let us notice that this hybrid algorithm differs from a genetic algorithm by some features. The fundamental difference is naturally that the random mutation operator of a GA is replaced with a LS operator. Another difference concerns the selection operator of a GA, which encourages the survival and reproduction of the best individuals in the population. In the hybrid algorithm, the selection is ensured jointly by *ChooseParents* and *UpdatePopulation*.

The initialization operator

The operator *InitPopulation(|P|)* initiates the population P with $|P|$ configurations. To create a configuration, we use the greedy saturation algorithm of [1] slightly adapted in order to produce a partition of k classes. The algorithm works as follows. We start with k empty color classes $V_1 = \cdots = V_k = \phi$. At each step, we chose a vertex $v \in V$ such that v has the minimal number of allowed classes (*i.e.* a class that does not contain any vertex adjacent to v). To put v in a color class, we chose among all the allowed classes of v the one V_i that has the minimal index i. In general, this process cannot assign all the vertices. Each of unassigned vertex is then put into one color class randomly chosen. Once a configuration is created, it is immediately improved by the LS operator for L iterations.

Due to the randomness of the greedy algorithm and the LS improvement, the configurations in the initial population are quite different. This point is important for population based algorithms because a homogeneous population cannot efficiently evolve.

The crossover operator

The crossover used here is the GPX crossover presented in Section 3. Let us show now how this crossover works. Given two parent configurations $s_1 = \{V_1^1, \cdots, V_k^1\}$ and $s_2 = \{V_1^2, \cdots, V_k^2\}$ chosen randomly by the *ChooseParent* operator from the population, the algorithm *Crossover(s1,s2)* builds an offspring $s = \{V_1, \cdots, V_k\}$ as follows.

Data : *configurations* $s_1 = \{V_1^1, \cdots, V_k^1\}$ *and* $s_2 = \{V_1^2, \cdots, V_k^2\}$
Result : *configuration* $s = \{V_1, \cdots, V_k\}$
begin
  for *l* $(1 \leq l \leq k)$ **do**
    if l is odd, then $A := 1$, else $A := 2$
    choose i such that V_i^A has a maximum cardinality
    $V_l := V_i^A$
    remove the vertices of V_l from s_1 and s_2
  Assign randomly the vertices of $V - (V_1 \cup \cdots \cup V_k)$
end
The GPX crossover algorithm

The algorithm builds step by step the k classes $V_1, \cdots, V_k$ of the offspring: at step l ($1 \leq l \leq k$), the algorithm builds the class V_l in the following way. Consider parent s_1 (A=1) or parent s_2 (A=2) according to whether l is odd or even. In the considered parent, choose the class having the maximum number of vertices to become class V_l and remove these vertices from parents s_1 and s_2. At the end of these k steps, some vertices may remain unassigned. These vertices are then assigned to a class randomly chosen.

In the shown example, there are 3 color classes ($k = 3$) and 10 vertices represented by capital letters A,B,$\cdots$,J. At step 1, class $\{D, E, F, G\}$ in parent 1 is chosen to become the first class V_1 of the offspring. Because vertices D,E,F and G are now assigned, they are removed from both s_1 and s_2: in s_1, we remove the complete class $\{D, E, F, G\}$, in s_2 we remove the vertices D,E,F and G from their respective classes. Similarly, we build classes V_2 and V_3 from parents 2 and 1 respectively. At the end of these 3 steps, vertex I is the only one to be randomly assigned to a class.

parent s_1 →	A B C	D E F G	H I J	$V_1 := \{D, E, F, G\}$	A B C		H I J
parent s_2	C D E G	A F I	B H J	remove D,E,F and G	C	A I	B H J
offspring s					D E F G		
parent s_1	A B C		H I J	$V_2 := \{B, H, J\}$	A C		I
parent s_2 →	C	A I	B H J	remove B,H and J	C	A I	
offspring s	D E F G				D E F G	B H J	
parent s_1 →	A C		I	$V_3 := \{A, C\}$			I
parent s_2	C	A I		remove A and C		I	
offspring s	D E F G	B H J			D E F G	B H J	A C

Table 2: The crossover algorithm: an example

The LS operator

The purpose of the LS operator *LocalSearch(s, L)* is to improve a configuration s produced by the crossover for a maximum of L iterations before inserting s into the population. In general, any local search method may be used. In our case, we use tabu search (TS), an advanced local search meta-heuristic [13].

TS performs guided search with the help of short and eventually long term memories. Like any LS method, TS needs a neighborhood function defined over the search space $N : S \mapsto 2^S$. Starting with a configuration, a typical TS procedure proceeds iteratively to visit a series of locally best configurations following the neighborhood. At each iteration, a *best* neighbor is chosen to replace the current configuration, even if the former does not improve the current one. This iterative process may suffer from cycling and get trapped in local optima. To avoid the problem, TS introduces the notion of *Tabu lists*. The basic idea is to record each visited configuration, or generally its attributes and to forbid to re-visit this configuration during next tl iterations (tl is called the tabu tenure).

The TS algorithm used in this work is an improved version of the TS algorithm proposed by Hertz and de Werra [16]. Here a *neighbor* of a given configuration s is obtained by moving a single vertex v from a color class to another color class

V_i. To make the search more efficient, the algorithm uses a simple heuristic: the vertex v to be moved must be conflicting with at least another vertex in its original class. Thus a neighboring configuration of s is characterized by a *move* defined by the couple $< v, i > \in V \times \{1 \cdots k\}$. When such a move $< v, i >$ is performed, the couple $< v, s(v) >$ is classified tabu for the next tl iterations, where $s(v)$ represents the color class of vertex v in s. Therefore, v cannot be reassigned to the class $s(v)$ during this period. Nevertheless, a tabu move leading to a configuration better than the best configuration found so far is always accepted (*aspiration criterion*). The tabu tenure tl for a move is variable and depends on the number nb_{CFL} of conflicting vertices in the current configuration: $tl = Random(A) + \alpha * nb_{CFL}$ where A and α are two parameters and the function $Random(A)$ returns randomly a number in $\{0, \cdots, A-1\}$. To implement the tabu list, it is sufficient to use a $V \times \{1 \cdots k\}$ table.

The algorithm memorizes and returns the *most recent* configuration s_* among the best configurations found: after each iteration, the current configuration s replaces s_* if $f(s) \leq f(s_*)$ (and not only if $f(s) < f(s_*)$). The rational to return the last best configuration is that we want to produce a solution which is as far away as possible from the initial solution in order to better preserve the diversity in the population (see Section 6.3 for more discussion on this topic). The skeleton of the TS algorithm is given below.

Data : *graph* $G = (V, E)$, *configuration* s_0
Result : the best configuration found
begin
  $s := s_0$
  while *not Stop-Condition()* **do**
    choose a best authorized move $< v, i >$
    introduce the couple $< v, s(v) >$ in the Tabu list for tl iterations
    perform the move $< v, i >$ in s
end
The TS operator

The configuration created by a crossover and improved by TS is now to be inserted in the population. To do this, the worst of the two parents is replaced.

5 Experimental results

In this section, we present experimental results obtained by HCA and make comparisons with other algorithms. In particular, detailed comparisons are presented between HCA and the TS algorithm.

5.1 Experimental settings

Test instances

The following graphs from the well-known second DIMACS challenge benchmarks are used [19][1].

- Three *random graphs*: DSJC250.5, DSJC500.5 and DSJC1000.5. They have 250, 500 and 1000 vertices respectively and a density of 0.5 with unknown chromatic number.
- Two *Leighton graphs*: le450_15c and le450_25c. They are structured graph with known chromatic number (respectively 15 and 25).
- two *flat graphs*: flat300_28 and flat1000_76. They are also structured graph with known chromatic number (respectively 38 and 76).

We are interested in these graphs because they were largely studied in the literature and constitute thus a good reference for comparisons. Moreover these graphs are difficult and represent a real challenge for graph coloring algorithms. Note that according to the optimization approach used (Section 4), each graph defines in reality a set of k-coloring instances for different values of k.

Evaluation criteria

To evaluate the performances of HCA and compare with other algorithms, we consider several criteria. Given a particular graph, the first criterion used is the *quality* of the best solution found, *i.e.* the smallest value of k for which the algorithm is able to find a k-coloring.

We are also interested in evaluating HCA for different values of k for a graph. Given a particular k-coloring instance (a graph and a value of k) and a particular stop criterion, we consider two criteria: *robustness* and *computational effort* or *solving speed*. Robustness is evaluated by the success rate "succ_runs/total_runs" (a successful run is one which finds a k-coloring for the fixed k). To measure computational effort, we use the (average) number of iterations and of crossovers needed for successful runs.

These criteria are also valid for TS, making it possible to compare the performance of HCA and TS.

Parameters

The two main parameters for HCA are the population size $|P|$ and the LS length L after a crossover. To fix $|P|$, we tested several different sizes and chose the size 10 for most of our experiments ($|P|$=5 is chosen for easy instances). Let us notice that this choice remains consistent with other existing HEAs which use also small populations.

Compared with $|P|$, the length L of LS chain is more critical on the performance of the algorithm. We study in detail the role of this parameter later (Section 6). Here we show simply its influence with an example. We take the graph DSJC500.5 and fix $k = 49$. We run the HCA with different values L: L=250,500,1000,2000,4000. For each value of L we perform 30 runs, count the number of successful runs and compute the average number of iterations needed for a successful run. Table 3 gives the results obtained with these values of L.

[1] Available via ftp from ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/.

graph	k	param	succ	iter	cross
DSJC500.5	49	(10,250)	0	-	-
		(10,500)	2	214,000	418
		(10,1000)	16	505,000	495
		(10,2000)	29	854,000	417
		(10,4000)	30	1,475,000	358

Table 3: Influence of the length of LS on the results of HCA

We observe from Table 3 that increasing L increases the success rate but also the number of iterations used for successful runs. Therefore larger L makes the algorithm more robust but also slower. This implies that there is no absolute best value for this parameter and that we must find a best compromise between robustness and speed for a given instance.

In addition to $|P|$ and L, the HCA requires two other parameters related to the TS algorithm. These parameters are A and α which are necessary to determine the tabu tenure tl. Experiments of various combinations suggested that ($A = 10$, $\alpha = 0.6$) is a robust combination for the chosen graphs.

5.2 Comparing HCA with TS

HCA uses TS as one of its main components. It is therefore interesting to know if HCA is able to improve on the results of the TS algorithm. To do this, we take each of the above presented graphs and fix different values for k, leading to different k-coloring instances. The chosen values for k begin from an initial value greater than the smallest one ever found for the graph and finishes at the best known value. For instance, the best k-colorings for DSJC500.5 require 48 colors. Then we take $k = 52$ to 48 (greater values gives too easy instances). In this way, we get a set of increasingly difficult k-coloring instances for each graph.

To solve a k-coloring instance, we run both the HCA and the TS algorithms several times (from 5 to 10), each run being given the same number of iterations (10 millions in general, more for the most difficult instances). Note that while unsuccessful TS runs stop always when the allowed number of iterations is reached, HCA may stop before reaching this number when the diversity in the population becomes too low ($D < 20$). Tables 4 and 5 show comparative results of HCA and TS for the chosen graphs.

In these tables, each line corresponds to an instance. For both algorithms, we indicate the number of successful runs (the number of fails appears in parenthesis) and the average number of iterations for successful runs. For HCA, the tables indicate in addition the average number of crossovers and the parameters $|P|$ and L used.

Table 4 presents the results of 18 k-coloring instances of the 3 random graphs. We notice first that TS and HCA can reach the same minimal value of k for the graph of 250 vertices. For the graph of 500 and 1000 vertices, HCA finds better solutions than TS and gains 2 colors and 6 colors respectively. Notice that our TS algorithm is very powerful compared with other TS coloring algorithms [16, 9, 6]. Taken this fact into account, it is remarkable that HCA is able to outperform largely the TS algorithm. Now if we consider the values of k for which both

graph	k	TS		HCA			
		succ	iter	succ	iter	cross	param
DSJC250.5	28	10	2,500,000	9(1)	490,000	235	(10,2000)
	29	10	587,000	10	96,000	86	(10,1000)
	30	10	97,000	10	18,000	62	(10,250)
DSJC500.5	48	-	-	5(5)	4,900,000	865	(10,5600)
	49	(10)	-	10	871,000	425	(10,2000)
	50	10	1,495,000	10	185,000	254	(10,700)
	51	10	160,000	10	62,000	119	(5,500)
	52	10	43,000	10	34,000	63	(5,500)
DSJC1000.5	83	-	-	1*	28,400,000	2015	(10,16000)
	84	-	-	3(2)	20,700,000	1283	(10,16000)
	85	-	-	4(1)	4,600,000	565	(10,8000)
	86	-	-	5	3,500,000	615	(10,5600)
	87	-	-	5	1,900,000	668	(10,2800)
	88	(5)	-	5	613,000	427	(10,1400)
	89	3(2)	4,922,000	5	350,000	490	(10,700)
	90	5	3,160,000	5	220,000	430	(10,500)
	91	5	524,000	5	114,000	157	(5,700)
	92	5	194,000	5	73,000	141	(5,500)

Table 4: Comparative results of HCA and TS on random graphs

algorithms find a k-coloring and compare the number of iterations necessary to find a solution, we observe that HCA is faster for all the considered instances.

Notice also that for a particular graph, solving a more difficult instance (a smaller k) requires a larger L. For instance, for the graph DSJC250.5, $L = 250, 1000$ and 2000 for $k = 30, 29$ and 28 respectively. This complements the information presented in Table 3.

graph	k	TS		HCA			
		succ	iter	succ	iter	cross	param
le450_15c	15	(10)	-	6 (4)	194,000	24	(10,5600)
	16	8 (2)	319,000	10	45,000	54	(10,700)
	17	10	18,000	10	29,000	72	(10,350)
le450_25c	25	(10)	-	-	(10)	-	-
	26	10	107,000	10	800,000	790	(10,1000)
	27	10	7,300	10	94,000	13	(10,4000)
flat300_28	31	(10)	-	6(4)	637,000	308	(10,2000)
	32	10	149,000	10	84,000	230	(10,350)
flat1000_76	83	-	-	4(1)	17,500,000	1008	(10,16000)
	84	-	-	5	5,300,000	652	(10,8000)
	85	-	-	5	2,000,000	490	(10,4000)
	86	(5)	-	5	1,100,000	540	(10,2000)
	87	1(4)	7,400,000	5	473,000	463	(10,1000)
	88	2(3)	4,000,000	5	288,000	566	(10,500)

Table 5: Comparative results of HCA and TS algorithms on structured graphs

Table 5 presents the results of the 14 k-coloring instances of the 4 structured graphs. We can observe similar results as for random graphs. Notice however an exception for the graph le450_25d for which HCA needs more iterations than TS to find 26 and 27 colorings. For this graph, we observe a different behavior as decreasing L does not make the search faster. On other graphs, HCA finds better solutions and is faster than TS.

To summarize, we see that quite often HCA improves strongly on the results of a long tabu search. This points out that the crossover is able to produce new

starting points which are very useful to make the search more efficient. Moreover, contrary to the believe that hybrid algorithms are computationally expensive, we see that the HCA algorithm is not only powerful, but also fast compared with TS. Next section will allows us to further appreciate the search power of the HCA.

5.3 Comparisons with the best known results

Now we compare the results of HCA with the best ones published in the literature [19]. To do this, we are interested in the quality criterion, *i.e.* the lowest value of k for which a k-coloring can be found. Table 6 presents comparative results on the 7 graphs.

graph	χ	best-known	TS	HCA
DSJC250.5	-	28	28	28
DSJC500.5	-	48	49	48
DSJC1000.5	-	84	89	83
le450_15c	15	15	16	15
le450_25c	25	25	26	26
flat300_28	28	31	32	31
flat1000_76_0.col	76	84	87	83

Table 6: Comparison between HCA and the best known results

Each line in Table 6 corresponds to a particular graph. When the chromatic number is known, it is indicated in column 2. Column 3 (best-known) indicates the smallest values of k for which a solution (k-coloring) has ever been found by an algorithm. Most of these results are produced by a population based LS algorithm combining two specific neighborhoods and using the strategy of successive building of color classes (Section 2) [22]. Column 4 (TS) indicates the results of our TS algorithm. Finally, column 5 (HCA) indicates the results of our hybrid coloring algorithm.

From the table, we observe that HCA finds the best known result for each graph except for le450_25c. What is more impressive is that HCA improves on the best known results for two graphs (DSJC1000.5, flat1000_76_0.col).

Note that for large random graphs (more than 300 vertices), the best algorithms use the strategy of successive building of color classes. No algorithm without this strategy is really efficient. HCA is the only algorithm able to compete with this strategy and even to improve on the best known results.

6 Analysis of HCA

As mentioned before, both the performance and the behavior of HCA is largely dependent on the value of the length L of TS chain after a crossover. Note first that the parameter L reflects the relative proportion of crossovers and LS in the algorithm, as L corresponds to the number of iterations performed after each crossover. In this section, we analyze the influence of the parameter L on two particularly interesting points: a) running profile of the cost function f and b) diversity of the population. For this purpose, experiments were performed on

various graphs. We present below in detail the results on a single graph, but these results are valid for other tested graphs.

The considered graph is DSJC500.5 with $k = 49$. We recall that this instance is very difficult for TS since TS alone is unable to find 49-coloring (50-coloring is the best result for TS). To solve this instance, we consider 4 different values of the parameter L: $L = 500, 1000, 2000$ and 4000. For each of these values, we perform 30 runs, each run being given a maximum of 3 millions iterations.

6.1 Running profile

A running profile is defined by the function $i \mapsto f_*(i)$ where i is the number of iterations and $f_*(i)$ the best cost value known at iteration i. Running profile is a natural way to observe the evolution of the best values of the cost function during a search. Figure 6.1 (left) shows the running profiles of HCA on the graph DSJC500.5 with $k = 49$ obtained with different values for L. The figure shows also the running profile for TS (right) for comparative purpose.

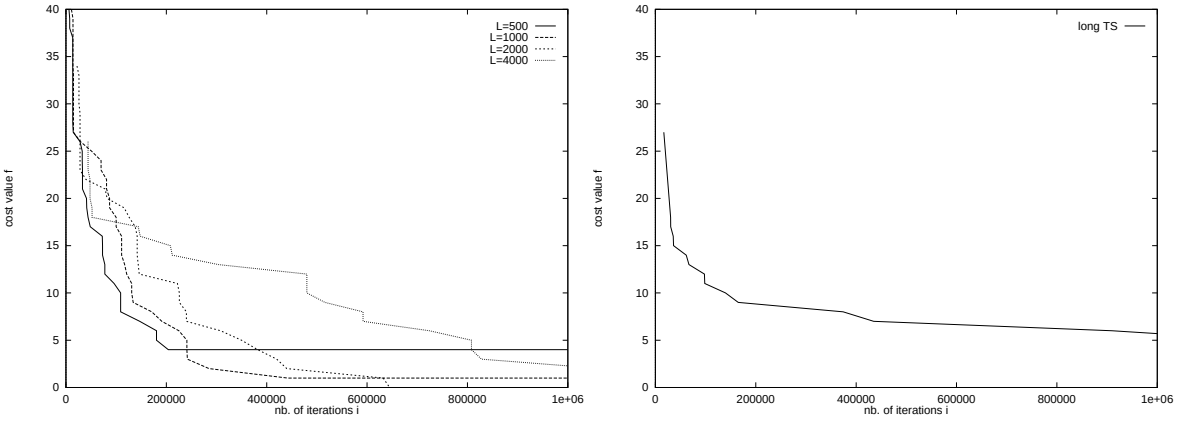

Figure 1: Running profile of HCA (left) and TS (right)

From the left figure, we first notice that HCA fails to find a solution with $L = 500$ and $L = 1000$, but is successful with $L = 2000$. For $L = 4000$, the algorithm finds also a solution, but needs more iterations (more than 1 millions, but smaller than 3 millions, not observable from the figure). We observe also that with $L = 500$ and $L = 1000$, the value of f_* decreases more quickly at the beginning than with $L = 2000$ and $L = 4000$. However, $L = 500$ and $L = 1000$ make the search blocked (at $f_* = 4$ for $L = 500$ and $f_* = 1$ for $L = 1000$) before f_* reaches a solution (f_*=0). On the contrary, $L = 2000$ and $L = 4000$ make the search progress more slowly, but for a longer time.

Concerning the running profile of TS (right), we see that f_* decreases quickly at the beginning of the run and then much more slowly, reaching a value of about 6 after 1 millions iterations. Comparing the running profiles of HCA and TS shows a clear interest of the crossover operator.

6.2 Evolution of diversity in HCA

For genetic algorithms, it is well-known that the population diversity has an important influence on the performance. A fast lost of the diversity in the population leads to a premature convergence. It is therefore crucial to be able to control correctly the evolution of the diversity.

For hybrid evolutionary algorithms, however, little is known yet on this topic. This section aims at studying the evolution of diversity during the search, by varying the parameter L. To do this, we need first a meaningful distance to measure the diversity. For many problems, hamming distance may be used. But for the coloring problem, this distance is no more meaningful. So we introduce the following specific distance applicable to partitions.

Given two configurations (partitions) s_1 and s_2, the distance between s_1 and s_2 is defined as the minimum number of elementary transformations necessary to transform s_1 into s_2. An elementary transformation corresponds to the operation of moving one element from a class to another class. The diversity of the population is then defined as the average distance D between all the configurations in the population. In the following, we use $D(i)$ to denote the diversity of the population after i iterations.

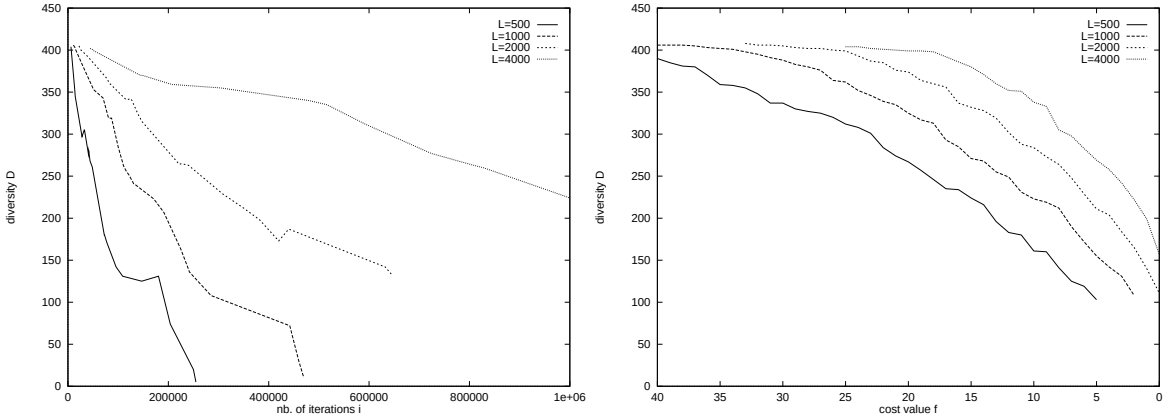

Figure 2: Diversity in function of i (left) and f_* (right)

Figure 6.2 (left) presents the evolution of the function $i \mapsto D(i)$ for $L = 500, 100, 2000$ and 4000. We first observe that the diversity decreases quite regularly for all the tested values of L. Moreover, we observe that the diversity is better preserved for $L = 2000$ and $L = 4000$ than for the two smaller values. Indeed, the diversity with smaller values of L decreases down to a value close to 0 while this is not the case for the two larger values. We remark also that the moment when the diversity reaches about 0 corresponds to the moment when the algorithm is blocked (see Figure 6.1). Note that this is not surprising since low diversity means high similarity among the configurations of the population. When this happens, crossovers will have no more effect because offspring created by a crossover will resemble its parents. Consequently, the population can no more evolve.

We see above that a too small L makes the algorithm blocked because the diversity decreases to 0 before f_* reaches 0. So it is interesting to study the

relationship between quality f_* and diversity D, depending on the value of L. Figure 6.2 (right) shows this relationship for the 4 studied values of L. From the figure, we observe that for a fixed value of f_*, the diversity is higher for a larger value of L, indicating that a longer TS chain preserves better the diversity in the population.

6.3 Why a longer LS chain preserves the diversity

Now we try to understand why a longer LS helps preserve the diversity. Note first that Figure 6.1 does not allow us to draw any conclusion about the relationship between f_* and D, though it shows a larger L makes both functions $i \mapsto f_*(i)$ and $i \mapsto D(i)$ decrease more slowly (Figure 6.1).

We present now the evolution of quality f_* and diversity D in function of the number of generations x instead of the number of iterations i (recall that a generation is a complete cycle of HCA, see Section 4.2).

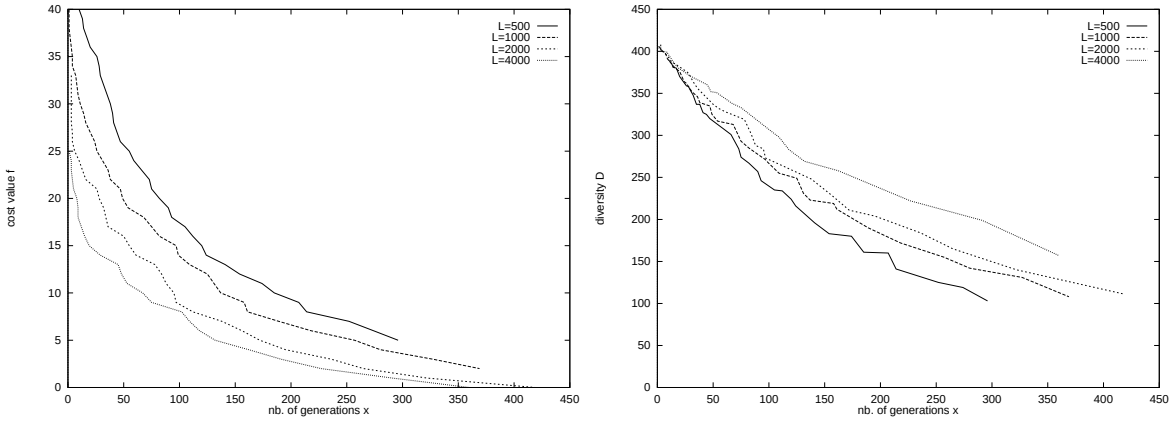

Figure 3: Quality f_* (left) and diversity D (right) in function of x

A first point shown in Figure 6.3 (left) is that a larger L makes the function $x \mapsto f_*(x)$ decrease more quickly than a smaller L does. This is not surprising as a longer LS helps to find better configurations at each generation. A second point observed in Figure 6.3 (right) is that a larger L makes the function $x \mapsto D(x)$ decrease more slowly. An intuitive explanation is that a longer LS chain makes the offspring generated by a crossover far from its parents, in particular from the parent remaining in the population. Consequently, for a same number of generations x, D must be larger for a same f_* with a larger L.

Now let us summarize the main results of this analysis. A long LS chain (less crossover) has two implications. The first one is that the search becomes slower. The second one is that the diversity of the population is better preserved. Reversely, a short LS chain (more crossover) makes the search faster, but reduces rapidly the diversity. Consequently, if one wants to find better results in terms of quality, one must use sufficiently large value for L. This is especially true if the problem to be solved is difficult. On the contrary, small values of L help find solutions more quickly for easy problems.

7 Discussion

7.1 Other partition crossovers

In previous sections, we presented and experimented a particular partition crossover. Now we present other partition crossovers that we have developed and experimented. These crossovers are based on the idea of renaming the color classes of one of the two parents. More precisely, to rename the color classes of the chosen parent, say s_2, we use a mapping $\sigma : \{1 \cdots k\} \rightarrow \{1 \cdots k\}$ such that $\Sigma_i |V_i^1 \cap V_{\sigma(i)}^2|$ is maximized. Once the renaming is done, the two parents may be recombined in several manners to generate offspring. Three crossovers were tested. The first one realizes simply a uniform recombination. In the second one, each V_i receives the intersection $V_i^1 \cap V_{\sigma(i)}^2$ and the configuration is completed with a greedy algorithm. In the third one, the V_i are built successively as follows: To build V_i, a) choose j such that $|V_j^1 \cap V_{\sigma(j)}^2|$ is maximum, b) build V_i from the union $V_j^1 \cup V_{\sigma(j)}^2$ and c) remove the vertices of V_i from their respective classes in the two parents.

Experiments with these crossovers show that they are generally able to produce interesting results. Taking into account the results reported in [7], we conclude that the basic principles behind the family of partition crossovers are really useful to design powerful crossovers for the GCP.

For comparative purpose, we also experimented some assignment crossovers within our HCA, including the uniform assignment crossover (Section 3) and the conflict-based crossover [9]. Results show that these crossovers sometimes improve on the results of TS, but cannot compete with the partition operators. All these results and those of [9] support the above conclusion concerning the importance of partition crossovers.

7.2 How to improve HCA

As seen in Section 6, the population diversity plays an important role on the performance of the algorithm. As shown before, one way to preserve the diversity is to apply a longer LS after each crossover. However, a longer LS makes the algorithm slower. At the same time, we have shown that short LS makes the algorithm fast. Therefore, the algorithm may use short LS (to be fast) and at the same time use other mechanisms to preserve the population diversity. To achieve this, there are several possibilities.

First, one may use a larger population. Although existing HEAs use often small populations, it should be interesting to see what happens with large ones. Second, one may design crossovers able to create offspring sufficiently different from its two parents. Third, the LS may be biased in order to prevent the offspring from becoming too close to its parents. Finally, selected solutions can be memorized when they are deleted from the population and re-inserted into the population when the diversity becomes to low.

8 Conclusions

In this paper, we introduced a new class of crossover operators for graph coloring and studied a particular member of this class. These crossovers are based on two initially simple ideas: a) a coloring is a partition of vertices and not an assignment of colors to vertices, b) a crossover should transmit subsets of color classes from parents to offspring. These new crossover operators are integrated into an hybrid algorithm using a well-known tabu search algorithm. The hybrid algorithm is evaluated on a set of difficult and large DIMACS challenge graphs.

Experimental results show clearly that the HCA improves strongly on the results of its TS operator making evident the importance of crossovers in the search process. Comparisons with other well-known coloring algorithms show that HCA is one of the most efficient algorithm and that hybrid algorithms are very powerful for this problem. Indeed, our HCA is able to find the best known results for most of the tested graphs and even able to improve on the best results for some largest graphs. Moreover, a hybrid evolutionary algorithm like the HCA is not computationally expensive to obtain high quality solutions, contrary to previous results reported in the literature.

Experiments are also realized to study the behavior of the HCA, especially the evolution of the cost function and of the diversity in the population. The experiments first confirm the necessity to preserve enough diversity for the search to be efficient. We analyze why more crossover, *i.e.* short LS, makes the search faster and at the same time tends to reduce the diversity while less crossover leads to opposite effects.

To conclude, this work confirms the potential power and interest of hybrid evolutionary algorithms for tackling hard combinatorial problems.

References

[1] D. Brélaz, "New methods to color vertices of a graph", Communications of ACM, 22: 251-256, 1979.

[2] G.J. Chaitin, "Register allocation and spilling via graph coloring", Proc. of ACM SIGPLAN 82 Symposium on Compiler Construction, 98-105, New York, NY, 1982.

[3] M. Chams, M. Hertz and D. de Werra, "Some experiments with simulated annealing for coloring graphs", EJOR 32: 260-266, 1987.

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

[5] L. Davis, "Handbook of genetic algorithms", Van Nostrand Reinhold, New York, 1991.

[6] R. Dorne and J.K. Hao, "Tabu Search for graph coloring, T-coloring and Set T-colorings", Chapter 3 of "Metaheuristics 98: Theory and Applications", I.H. Osman et al. (Eds.), Kluver Academic Publishers, 1998.

[7] R. Dorne and J.K. Hao, "A new genetic local search algorithm for graph coloring", Lecture Notes in Computer Science 1498: 745-754, Proc. of PPSN'98, Amsterdam, 1998.

[8] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing", Journal of Heuristics, 2(1): 5-30, 1996.

[9] C. Fleurent and J.A. Ferland, "Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability", in [19], 619-652, 1996.

[10] B. Freisleben and P. Merz, "New genetic local search operators for the travelling salesman problem", Lecture Notes in Computer Science 1141, 890-899, Springer-Verlag, 1996.

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

[12] M.R. Garey and D.S. Johnson, "Computer and intractability", Freeman, San Francisco, 1979.

[13] F. Glover and M. Laguna, "Tabu Search", Kluwer Academic Publishers, 1997.

[14] D.E. Goldberg, "Genetic algorithms in search, optimization and machine learning", Addison-Wesley, 1989.

[15] J.J. Greffenstette, "Incorporating problem specific knowledge into a genetic algorithm", Genetic Algorithms and Simulated Annealing (L. Davis, Ed.), 42-60, Morgan Kaufmann Publishers, 1987.

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

[17] J.H. Holland, "Adaptation and artificial systems", Ann Arbor, University of Michigan Press, 1975.

[18] 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.

[19] D.S. Johnson and M.A. Trick, "Proceedings of the 2nd DIMACS implementation challenge", Volume 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.

[20] F.T. Leighton, "A graph coloring algorithm for large scheduling problems", Journal of Research of the National Bureau Standard, 84: 489-505, 1979.

[21] P. Merz and B. Freisleben, "A genetic local search approach to the quadratic assignment problem", in Proc. of the 7th International Conference of Genetic Algorithms, 465-472, Morgan Kauffman Publishers, 1997.

[22] C. Morgenstern, "Distributed coloration neighborhood search", in [19], 335-358, 1996.

[23] H. Muehlenbein, M. Gorges-Schleuter and O. Kraemer, "Evolution algorithms in combinatorial optimization", Parallel Computing, 7: 65-88, 1988.

View publication stats