The sum coloring problem: a memetic algorithm based on two individuals

Laurent Moalic
IRIMAS, University of Haute-Alsace
Mulhouse, France
laurent.moalic@uha.fr

Alexandre Gondran
ENAC, French Civil Aviation University
Toulouse, France
alexandre.gondran@enac.fr

***Abstract*—Let G be a graph, for which each vertex is assigned one of k colors in $\{1, ..., k\}$. A legal coloring requires that two adjacent vertices have two different colors. The minimum sum coloring problem (MSCP) consists of finding such a legal coloring with the smallest sum value, knowing that to each color is assigned an integer value from 1 to k. This paper presents a new memetic approach for this NP-hard problem. The proposed hybrid algorithm is based on a very small population, composed of only two individuals. Thanks to this small population, no selection operator needs to be defined, nor any replacement strategy. In order to prevent a premature convergence of the algorithm, alternative mechanisms are introduced based on an elitist approach. The local search introduced in the population based algorithm is split in two phases. The first one is based on the well known *TabuCol* algorithm and aims to reduce the number of conflicting vertices. The second one, which is used to improve the total sum value, implements an efficient 2-move local operator. The proposed approach was successfully applied on many graphs from the reference COLORS02 and DIMACS benchmarks. It allowed to achieve most of the best known results, and to overpass the best known results for 15 challenging graphs.**

***Index Terms*—Minimum sum coloring, Metaheuristics, Memetic Algorithm, Tabu search, Hybridization**

I. Introduction

Given an undirected graph $G = (V, E)$, with $V = \{v_1, ..., v_n\}$ the set of vertices and $E \subset V \times V$ the set of edges. Let $\{1, ..., k\}$ a set of k colors. A legal coloring (or proper) is the assignment of one of k colors to each vertex, such that $\forall e = (v_1, v_2) \in E$, v_1 and v_2 have different colors. Notice that this problem is equivalent to assign each vertex to one among k color sets V_i (i.e. k-partition of G), verifying that $\forall i \in \{1, ..., k\}, \forall v_1, v_2 \in V_i, e = (v_1, v_2) \notin E$.

The classical graph coloring problem (GCP) consists of finding a legal coloring, with the smallest k value known as the chromatic number $\chi(G)$.

One of the variant of GCP is the minimum sum coloring problem (MSCP). It is an old problem introduced in [1] which aims to find a legal k-coloring with the smallest sum of the colors assigned to the vertices. The minimum sum of colors for G, denoted $\Sigma(G)$, is known as the chromatic sum. It may be obtained with a number of colors greater than the chromatic number. Determining this smallest sum for arbitrary graphs is shown to be NP-hard [1]. Many heuristic approaches have been developed over the last decades to tackle the MSCP. It is a particularly interesting problem because of its many applications, especially in scheduling.

Suppose that the integer corresponding to the color assigned to vertex v is noted $\phi(v)$. The sum value of a coloring c is obtained as follows:

$$\sigma(c) = \sum_{i=1}^{n} \phi(v_i) \tag{1}$$

It may be noted that two solutions c_1 and c_2 differing only from a color swap can be considered identical while $\sigma(c_1) \neq \sigma(c_2)$. This is why, in most of the studies, the symmetries are broken by not considering the color assigned to the vertices but by considering only the vertices having the same color, that is to say being in the same color class. This aspect is important for MSCP because color classes can be sorted from the biggest to the smallest one. In this context, equation (1) becomes:

$$\sigma(c) = \sum_{i=1}^{k} i|V_i| \tag{2}$$

with $|V_i|$ the cardinality of the color class V_i and $|V_1| \geq |V_2| ... \geq |V_k|$.

A simplistic reasoning might suggest that a good solution for the GCP is also good for the MSCP. Indeed, a coloring with few colors means that no vertex has a high value. Fig. 1 shows an example from Jin and Hao [2] which illustrates that it is not necessary true, even for a small graph.

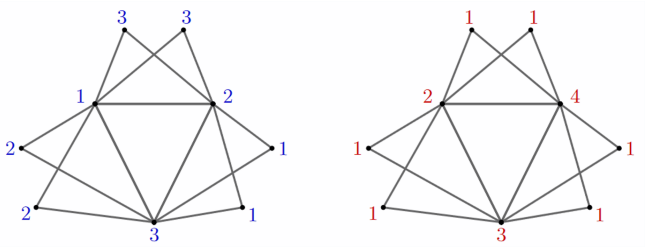

Fig. 1. Example of MSCP vs GCP from Jin and Hao [2]

On this example the chromatic number is clearly 3 as shown on the left graph of Fig. 1, but with a sum value $\sigma(c) = 18$. Adding one more color allows to reach the chromatic sum $\Sigma(G) = 15$ as illustrated on the right graph of Fig. 1.

978-1-7281-2153-6/19/$31.00

As stated above, MSCP is proved to be NP-hard [1] in the general case. Several works propose lower bounds for $\Sigma(G)$. Alavi et al. [3] showed that for any graph G, $\lceil\sqrt{8m}\rceil \leq \Sigma(G) \leq \lfloor\frac{3}{2}(m+1)\rfloor$. Although a lower bound does not provide a solution, it is an interesting indicator, especially to determine if a solution provided by a heuristic approach can be close to optimality.

Because of the scientific and applicative issues on the MSCP, many algorithms have been developed over the years for finding a good upper bound. It may be worth noting that, in some works, good mechanisms coming from the classical GCP are adapted to the special situation of MSCP. Li et al. [4] proposed greedy algorithms based on DSATUR and RLF, well known greedy algorithm for GCP.

Several local search algorithms have been developed based on different strategies. Wu and Hao [5], [6] proposed mechanisms particularly well suited to large graphs. They use a tabu strategy combined with a greedy approach.

The most effective algorithms are based on an hybridization of evolutionary strategies with powerful local searches [7]. Up to now, the two most effective algorithms are HESA [2] for most of the graphs and more recently EBES [8] which is specially designed for very big graphs.

A very complete survey [9] provides a good starting point for interested readers.

Notice that only few works address MSCP with exact approaches. Only small graphs or with special structures can be exactly solved. Furini et al. [10] proposed ILP models and column generation with good results on some instances.

The remainder of the paper is organized as follows. In Section II we present the proposed hybrid algorithm. In section III we report on the obtained results, on graphs coming from two references benchmarks. Section IV provides some analysis on the running of our algorithm. Finally, in section V a conclusion is drawn on the work presented.

II. A MEMETIC ALGORITHM WITH TWO INDIVIDUALS

Memetic Algorithms (MA), also known as Genetic Local Search (GLS), are based on the hybridization of a local search (LS) and a population based approach such as genetic algorithm (GA). The classical mutation, which can be see as a diversification operator in a GA, is replaced in MA by a LS, which is an intensification operator. Thus, the role played by the components in MA and GA is totally different. In classical GA, the optimal solution is generally reached after a crossover which can be seen as an intensification operator. In MA the crossover introduces the diversity, while LS provides the intensity. With both families of algorithms, we can notice that the larger the population, the greater the diversity.

In a recent work on GCP, the algorithm $HEAD$ [11], [12] introduced the principle of a population of only two individuals and a new way for managing the diversity. It is up to now among the most effective algorithms for solving GCP [1]. Instead of having a complete population of solutions, there is only a couple of solutions (a population of size two). The idea is that with few individuals, less time is lost to improve the individuals. The main drawback with only two individuals is the risk of a premature convergence of the algorithm, reached when the individuals become the same. Indeed, once the individuals are identical, the crossover has no effect and the algorithm becomes comparable to a simple local search. To prevent such a convergence of the two individuals, an elitist mechanism is introduced.

The goal of this work is to identify how the general principle developed in the $HEAD$ heuristic can be reused on another problem, the MSCP, and lead to a metaheuristic.

A. Main scheme of the $sumHEAD$ algorithm

The proposed memetic algorithm named $sumHEAD$ (sum Hybrid Evolutionary Algorithm in Duet) is characterized by a population of two individuals. The main steps of $sumHEAD$ are described in Algorithm 1. They can be summarized as follows. Lines 1-3 correspond to the initialization of the algorithm. At the beginning the number of colors k is equal to the number of vertices. That means that we are sure we can find a legal coloring, without conflict.

Each $while$ loop starting at line 4 corresponds to one generation. Lines 5-12 provide the central part of $sumHEAD$. Two children are generated from the two only possible parents in the population (lines 5-6). At this step the children can have a lot of conflicted edges. That is why before improving the sum coloring, lines 7-8 are for decreasing the number of conflicts, using the well known *TabuCol* algorithm initially developed for GCP. Then, lines 9-10 allow to remove the remaining conflicts, by adding new colors. Now c'_1 and c'_2 have no conflict and can be processed by the 2-move tabu search we have specially designed for improving the sum value (line 11 and 12). This algorithm is described in detail Section II-B2.

From line 13 to 34, this is the management of the newly improved p_1 and p_2 individuals, who are the parents for the next generation. In lines 13 to 17, current solutions are tested to determine if a new best solution is found, the best since the beginning of the algorithm ($best$) and the best since the beginning of the current cycle ($elite_1$ the best k-coloring and $elite_{\sigma 1}$ the coloring with the smallest sum value).

Since the algorithm started with as many colors as vertices, it is very likely that a legal coloring is found. When this happens, the local search *TabuCol* has no more effect. This situation is detected at line 18. If the remaining number of conflicts is 0, the number of colors is reduced by one. This operation causes an increasing number of conflicts. The color chosen to be deleted corresponds to the smallest class. This is obviously the most interesting, because a small class means that few vertices will have a new color, chosen at random in $1, ..., k-1$, which limits the number of new conflicts.

Now lines 20-33 deal with diversity management. Line 20 tests if the end of the current cycle is reached, or if the both parents are identical. If so, the best sum coloring from the previous cycle $elite_{\sigma 2}$ replaces parent p_1 and a new cycle begins. If despite the new cycle p_1 and p_2 remain

[1]C++ code available at: github.com/graphcoloring/HEAD

the same (line 25), the best solution in terms of conflicts from the previous cycle replaces p_1. This operation is called *perturbation*. And finally, if p_1 and p_2 remain the same after the *perturbation*, we consider that the algorithm is trapped in a local optimum. In such a situation, p_1 starts with a new random coloring. We call this diversification operator a *restart*. But only one parent starts from scratch, in order to help the other parent to escape from the local optimum.

We can notice that the *restart* mechanism does not allow the algorithm to reach a stable state, which could have been a stopping criterion. The end of $sumHEAD$ is determined by a user-supplied runtime value.

1) Search space: The algorithm starts with $|V|$ colors, and accepts proper and improper solutions. That is to say that, when the algorithm is running, it is possible to have solutions with two vertices u and v in the same color class, even if $(u, v) \in E$. Thus, the search space Ω is composed of all partitions of V in at most $|V|$ classes. However in practice, the search space evolves dynamically. After a while, two search spaces can be identified. The first one for the crossover and *TabuCol* accepts proper and improper colorings of size $k^* - 1$, with k^* the smallest number of colors found so far. The second search space, for *TabuSum*, only accepts legal colorings, that is, no conflict is allowed during this exploration. In order to have a legal coloring, the number of colors is not fixed. It is always possible to add new color classes, or conversely, to remove the last vertex of a set leading to a solution with one less color.

2) Objective function: For a coloring c, the fitness value is defined as the sum $\sigma(c)$, given by the equation Eq. (2). We are in a minimization context: given two colorings c_1 and c_2, we will say that c_1 is better than c_2 if and only if $\sigma(c_1) < \sigma(c_2)$. We can notice that solutions are evaluated after the *TabuSum* step. That is to say we are certain that the best solution recorded is always a legal coloring, when evaluating the sum value. Note that for the *TabuCol* local search, the objective function is not the same. Indeed, *TabuCol* works with k-fixed colorings and the fitness value is given by the number of edges remaining in conflict. In *TabuCol*, we consider that the coloring c_1 is better than c_2 if and only if c_1 has less conflicting edges than c_2.

3) Initial population: Any initialization process could be used for the proposed algorithm. Here we simply start with two random colorings, where each vertex is colored with a random color between 1 and $|V|$.

Algorithm 1: *sumHEAD* - main scheme

Input: $Iter_{TC}$, the number of *TabuCol* iterations; $Iter_{TS} = 10$, the number of *TabuSum* iterations without improvement; $Iter_{cycle} = 10$, the number of generations into one cycle.
Output: the best sum coloring found: $best$

1 $k \leftarrow nbVertices$ /* start with as many colors as vertices */
2 $p_1, p_2, elite_1, elite_2, elite_{\sigma 1}, elite_{\sigma 2}, best \leftarrow$ init() /* initialize with random k-colorings */
3 $generation,\ cycle \leftarrow 0$
4 **while** *not finished* **do**
/* c_1, c_2 (non-)legal k-colorings */
5 $c_1 \leftarrow GPX(p_1, p_2)$
6 $c_2 \leftarrow GPX(p_2, p_1)$
7 $c_1 \leftarrow TabuCol(c_1, Iter_{TC})$
8 $c_2 \leftarrow TabuCol(c_2, Iter_{TC})$
/* c'_1, c'_2 legal colorings with more than k colors or k colors */
9 $c'_1 \leftarrow removingConflits(c_1)$
10 $c'_2 \leftarrow removingConflits(c_2)$
11 $c'_1 \leftarrow TabuSum(c'_1, Iter_{TS})$
12 $c'_2 \leftarrow TabuSum(c'_2, Iter_{TS})$
13 $elite_{\sigma 1} \leftarrow$ saveBestSum(c'_1, c'_2, $elite_{\sigma 1}$) /* best legal σ-coloring of the current cycle */
14 $best \leftarrow$ saveBestSum($elite_{\sigma 1}, best$)
/* p_1, p_2 (non-)legal k-colorings */
15 $p_1 \leftarrow removingColor(c'_1, k)$
16 $p_2 \leftarrow removingColor(c'_2, k)$
17 $elite_1 \leftarrow$ saveBest(p_1, p_2, $elite_1$) /* best (non-)legal k-coloring of the current cycle */
18 **if** $remaining\_conflit(elite_1) = 0$ **then**
19 $k \leftarrow k - 1$ /* remove one color, randomly chosen among the smallest color classes */
20 **if** $generation \% Iter_{cycle} = 0$ *or* $p_1 = p_2$ **then**
/* elitist swap */
21 $p_1 \leftarrow removingColor(elite_{\sigma 2}, k)$
/* best σ-coloring of the previous cycle */
22 $elite_{\sigma 2} \leftarrow elite_{\sigma 1}$
23 $elite_{\sigma 1} \leftarrow$ init()
24 $cycle++$
25 **if** $p_1 = p_2$ **then**
/* despite the previous swap, p_1 and p_2 remain the same */
26 $p_1 \leftarrow elite_2$ /* best k-coloring of the previous cycle */
27 $elite_2 \leftarrow elite_1$
28 $elite_1 \leftarrow$ init()
29 $perturb++$
30 **if** $p_1 = p_2$ **then**
/* despite the two previous swaps, p_1 and p_2 remain the same */
31 $p_1 \leftarrow$ init() /* new random solution */
32 $p_1 \leftarrow$ *TabuCol*(p_1,$Iter_{TC}$)
33 $restart++$
34 $generation++$
35 **return** $best$

B. A double tabu search approach

The GCP and the MSCP clearly share some common features. Although a good solution for GCP (small number of colors) is not necessary good for MSCP (see Section I), it seems interesting to notice that it is a good starting point. Indeed, few colors means no vertex with a high sum contribution which is promising in terms of sum value. That is why we propose in our algorithm to use the effective *TabuCol* algorithm to remove the existing conflicts between vertices. Then, we have developed a second tabu LS, named *TabuSum*, to improve the global sum of the coloring.

1) TabuCol for removing conflicts: Finding the chromatic number $\chi(G)$ in GCP is NP-hard [13]. But here we only need an effective algorithm able to remove conflicts between vertices. An improvement of the *TabuCol* of [14] is used in the proposed approach. This is a powerful tabu local search that provide good results, sufficient for the role played in the

MSCP. It is important to see that *TabuCol* is not used to achieve a coloring without conflict, but only minimize conflicting vertices. In other words, it provides a good candidate for the step of improving the sum value.

2) TabuSum: a 2-move tabu improvement of the sum coloring: This tabu search is done at the end of each call to *TabuCol*. This step works with legal colorings. That is why, at first, conflicts are removed by adding new colors. In a second time, some vertices are cleverly rearranged from one color class to another, in order to improve the sum value of the solution.

a) Removing conflicts: The first step here consists of removing the conflicts existing between vertices. This is done in a greedy way:

1) starting with a k-colors solution with conflicts
2) for each conflicting vertex, we check if a color class can contain this vertex without conflict
3) if no existing class can get this vertex, a new color class is created and this vertex is assigned to the new class
4) ending with a $(k+j)$-coloring without conflict

b) The neighborhood principle: From a solution, the neighboring solutions are those that can be reached by a 2-move, and being not tabu. This 2-move mechanism is an important point of the proposed approach. Most often, LS algorithms take into account only one move. That is, a vertex is chosen and assigned to another color class if this move leads to a better solution. Here we consider one more step. A vertex is chosen and assigned to another color class regarding the move itself and the next best move, enabled by the released place.

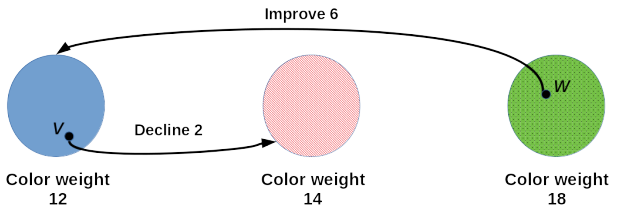

Fig. 2. Example of a two move leading to an improvement of 4

Figure 2 shows an example of a 2-move. The move #1 which consists of changing the vertex v from blue to red does not seem interesting because it increases the overall weight of two. But here, removing v from the blue class allows the move #2 of w from green to blue, with an improvement of 6. So, the overall impact of this 2-move noted $\Delta(v, w, c = red) = 2 - 6 = -4$ represents a significant improvement.

c) Neighbor choice: The strategy used to select the neighboring solution is a *non tabu best improve*. At each local step, we select the best triple (v, w, c), with v the vertex changing to color c, and w the vertex changing to the old color of vertex v, noted c'. Of course, these moves must satisfy the conflict constraints: $\forall x \in V_c, (v, x) \notin E$ and $\forall x \in V_{c'}, (w, x) \notin E$, with V_c and $V_{c'}$ are the set of vertices colored in c and respectively c'. Moreover, these movements must satisfy the following tabu constraints: $c \notin \text{tabu}(v)$ and $c' \notin \text{tabu}(w)$.

These steps are described in Algorithm 2.

Algorithm 2: *TabuSum* - a 2 move tabu algorithm

Input: s, the solution to improve ; $maxIter = 10$, the number of *TabuSum* iterations without improvement.
Output: the best sum-coloring found: $best$

1 $iter \leftarrow 0$ /* current iteration */
2 $iter_{last} \leftarrow 0$ /* record of the last improvement */
3 $best \leftarrow s$
4 $sumCost \leftarrow \sigma(s)$
5 $bestCost \leftarrow sumCost$
6 **while** $iter - iter_{last} <= maxIter$ **do**
7 $iter \leftarrow iter + 1$
8 $(v, w, c) \leftarrow$ getBestImproveNotTabu()
9 $sumCost \leftarrow sumCost + \Delta(v, w, c)$
10 $c' \leftarrow s(v)$ /* old color of vertex v */
11 $s(v) \leftarrow c$
12 $s(w) \leftarrow c'$
13 **if** $sumCost < bestCost$ **then**
14 $bestSol \leftarrow s$
15 $bestCost \leftarrow sumCost$
16 $iter_{last} \leftarrow iter$
17 **return** $best$

3) An incremental approach for a better speedup: Nearly all of the computation time is dedicated to the tabu searches. *TabuCol* contains incremental mechanisms not to compute the total number of conflicts from scratch. To achieve this, tables contain information about the impact in terms of conflicts for all the transitions, for every vertices and every colors. For more information about that please refer to [15]. The same principle is developed for *TabuSum*. Indeed, computing the sum value of a solution is significantly time consuming. It can be sum up as follow:

1) sorting color classes by decreasing size, from the largest to the smallest
2) assigning a weight ω_i to each node i, accordingly to the position of its color
3) determining the total sum of the solution s as $\sigma(s) = \sum_{i=1}^{n} \omega_i$

Of course, the main time consumption comes from step 1, to sort all classes.

We developed an incremental approach without any sorting mechanism during the *TabuSum* iterations. The only complete computation of the sum is performed at the beginning, at the line 4 in the Algorithm 2.

C. A partition based crossover

In the MSCP it is particularly interesting to have big color classes. Indeed, a small value is assigned to a big class. The more vertices in the big classes, the less the global sum value. A crossover developed for the GCP tends to favor the biggest classes. It is known as the Greedy Partition Crossover (GPX) from [15]. It takes two solutions as the parents, and transmits to the child the biggest color classes of each parent.

Figure 3 illustrates the main principle of this crossover. Starting with the first parent, the largest class is transmitted to

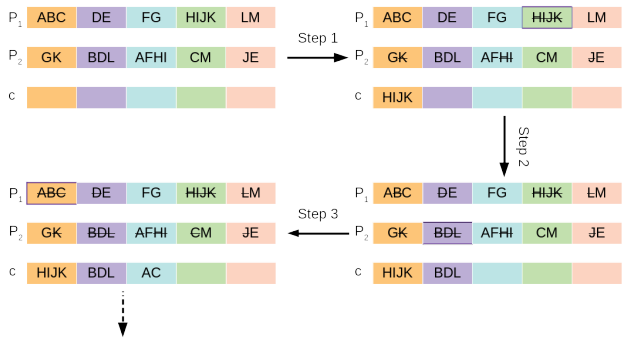

Fig. 3. Example of the GPX crossover from [15]

the child solution (step 1). Then the transmitted vertices are removed from both parents. Now the second parent transmits the biggest remaining class to the child (step 2) and so on. In the case of ties (see step 3 for example), one of the largest classes is selected randomly. In the example, it is the first color class with vertices A and C.

D. Premature convergence management

This is a recurrent issue with metaheuristic algorithms. The algorithm can be trapped into a local optima, with no possibility to escape for exploring other promising search areas. On the other hand, if we introduce too much diversity, it is possible to miss an interesting solution. A population of only two individuals allows a "quick" convergence of the algorithm. Diversity is introduced at different levels.

1) The tabu tenure used in both *TabuCol* and *TabuSum* allows, for the local searches, to find better solutions than a simple descent. It can be seen as a local diversification.
2) The crossover is a second level diversification. After local searches it allows to go farther.
3) And finally, three more levels of diversification are provided in Algorithm 1, at lines 16 to 29, with the introduction of the elite solution.

This multilevel diversity management is for sure a key point of the proposed algorithm. It allows a powerful balance between intensification and diversification.

III. Experimental results

A. Instances and benchmarks

The proposed algorithm was applied on many graphs coming from the second DIMACS challenge of 1992-1993 [16] and from the COLORS02[2] competition website. These benchmarks are among the most used due to the high variety of graphs and their high challenging level. They are commonly used for the graph coloring problem and the variants such as the sum coloring problem.

Several types of graphs are treated in this work.

[2]https://mat.gsia.cmu.edu/COLOR02/

DSJC are random graphs, characterized by a number of vertices and a density: DSJC$n.d$ means n vertices and each vertex is randomly linked to $(d * 100)\%$ of the other vertices.

DSJR are geometric graphs, defined by a number of vertices and a density. Notice that a graph named DSJR$n.d$c is the complement of the geometric graph.

Leighton graphs have a known chromatic number. Nevertheless, there is no guarantee on the best sum value.

Queen graphs have a specific structure. It is inspired by the chess: given an n by n chessboard, a queen graph is composed of n^2 vertices, each corresponding to a square of the board. An edge connects two vertices if and only if the corresponding squares are in the same row, column, or diagonal.

Flat graphs are structured to be particularly difficult. For these graphs, the chromatic number is known and given in the graph name: flatn_χ means a graph with n vertices an a chromatic number of χ.

B. Experimentation protocol

The proposed $sumHEAD$ algorithm was coded in C++. In order to perform all the experimentation we used a cluster with Intel Xeon CPU at 2.60GHz. Notice that more than 99% of the running time is for the local searches, *TabuCol* and *TabuSum*. At each generation, the two children are improved with the local search in parallel. Indeed, no exchange of information is required for this step and two processes can do the job simultaneously. The OpenMP library is used for this speedup implementation.

The parameters used for this experimentation are as follows:

- $\boldsymbol{L_1}$ **and** $\boldsymbol{\lambda}$ **:** parameters for the tabu tenure in *TabuCol*. At each step, the tenure d_1 is given by $d_1 = rnd(L_1) + \lambda F(s)$ with $F(s)$ the number of conflicting vertices in the solution s, $rnd(L_1)$ a random number in $[0; 9]$ and $\lambda = 0.6$.
- $\boldsymbol{L_2}$ **:** parameter for the tabu tenure in *TabuSum*. The tenure d_2 is given by $d_2 = rnd(L_2)$ with $rnd(L_2)$ a random number in $[0; 9]$.
- $\boldsymbol{Iter_{TS}}$ **:** is the max number of iterations without improvement in *TabuSum*. This parameter provides the stopping criteria for *TabuSum*. It is a fixed value for all the graphs, set at $Iter_{TS} = 10$.
- $\boldsymbol{Iter_{TC}}$ **:** is the only parameter tuned for each graph. It provides the fixed iteration number for *TabuCol* and is provided in Table I column 7.

Notice that the parameter $Iter_{TC}$ is experimentally determined. Only some values have been tried and new improvement could be found with a more in depth study of this point.

The proposed algorithm is clearly stochastic. That is to say that two runs on the same data leads to two different solutions. In order to get statistically significant results, $sumHEAD$ was run 30 times for each graph.

C. Computational results

Table I gives an overview of the results obtained with the $sumHEAD$ algorithm. The four first columns provide information on the graphs, with $|V|$ the number of vertices,

$|E|$ the number of edges, and f_{UB} the best known sum value to date. The following columns provide information on the results get with our algorithm. Notice that numerous papers on MSCP provide the results with a running time limit of two hours. Therefore, to easily compare the algorithms, we set the time limit to 2 hours.

For each graph, Table I provides the following information:

- $\boldsymbol{f^*}$**:** The best sum value obtained on the 30 runs
- $\boldsymbol{Avg.}$**:** The average sum value of the 30 runs
- $\boldsymbol{Iter_{TC}}$**:** The number of local iterations for *TabuCol*. This parameter allows to obtain a well suited search according to the characteristics of the graph
- **Cross:** The average number of generations for finding the best result of each run
- **Time:** The average running time for finding the best result of each run

As shown in Table I, $sumHEAD$ can reach the best known results for most of the graphs. Moreover, 15 new best upper bounds are found thanks to $sumHEAD$. Notice that the proposed approach fails to get the best known results on 11 graphs, especially for very big graphs with 1000 vertices. This is not a surprise since for such large graphs, algorithms such as [8] have been specifically designed and include specific mechanisms based on the extraction of large classes.

Among the graphs for which the proposed approach provides significant results, we can focus on the DSJC$n.d$ random graphs. Up to a size of 500 vertices, $sumHEAD$ always reaches the best known results and overpass for 4 challenging graphs, with different size and density.

IV. Some analysis of $SumHEAD$

We propose here to analyze the sensitivity of $sumHEAD$ to parameters and the choice of the operators. The goal is to better understand why and when the proposed algorithm works well. In the remainder of this section, we will first study the relationship between the number of local iterations in $TabuSum$ and the global results. Second, we will compare the proposed 2-move operator with a simple 1-move in *TabuSum*.

A. Which TabuSum duration?

As stated Section III-B the *TabuSum* duration is set to 10 for all the graphs. That means that after 10 local iterations without improvement, *TabuSum* stops searching. This value may seem very small, especially when compared to the number of local iterations for *TabuCol*. This means that it is more interesting to take time to reduce the number of conflicts than to directly improve the sum value.

The plots in Figure 4 show the results for 4 graphs, DSJC500.1, DSJC500.5, DSJC500.9 and le450_15b. The first three graphs are random graphs where $sumHEAD$ gets new best results, while for the last graph we fail to reach the best upper bound. For each graph, we experiment different local search duration for $TabuSum$, from 0 to 800 iterations. As a reminder, this duration defines the maximum number of local iterations without improvement. That is to say that, for LS=0, we don't accept any iteration without improvement. Thus, LS=0 means that we apply a simple descent until a local optima. For each configuration of each graph we run our algorithm 30 times, each for 2 hours. For an easy reading, solutions are ranked from the best sum value (the smallest one) to the worst.

The first interesting information is that, for the three DSJC graphs, a big number of LS is really bad in terms of sum value. Indeed, the yellow and orange curves (800 and 200 LS respectively) are for these three graphs at the top, which means a high sum value. This observation is not intuitive. In fact, if we greatly improve the sum value, at the end it is not good. A likely explanation is that with a long local search, the two individuals of the population become too different, that is, there is too much diversity. And finally, after the crossover step, $TabuCol$ can no longer eliminate so many conflicts.

Second, the plot of le450_15b reveals that a LS at 0 (purple) is significantly bad. A simple descent is obviously not enough and the tabu mechanism brings a very interesting behavior.

Finally, attentive readers may notice that for a very dense graph (DSJC500.9), many runs provide the same sum value. Except for LS=800, half of the sum values are at 29870. A very dense graph is so constrained that it seems difficult to get many different solutions.

From this analysis, we can clearly remove LS values 200 and of course 800. We can also remove LS=0, which gives bad results for the last graph. Finally we chose LS=10, which provides a good balance for these graphs.

B. Local 1-move vs 2-move operators

As described in Section II-B2 the neighborhood used in $TabuSum$ is based on a 2-move operator. The simplest idea is of course a 1-move local operator. That is, for a coloring c, a neighbor solution for c has one and only one vertex with a different color. But with a 1-move, we do not take into account the potential improvements allowed by leaving a color class.

In Table II we are interested in such a comparison between the two operators, 1-move and 2-move. The graphs are the same four as previously. And the results are provided after 30 runs, with the same parameters used in Table I.

The most interesting information is that the 1-move operator is clearly not competitive at all. The 2-move local operator proposed in this work plays an important role in the quality of the results we obtained.

V. Conclusion

This paper presents a new hybrid algorithm named $sumHEAD$ to solve the well-known Minimum Sum Coloring Problem (MSCP). It is based on a population reduced to the smallest possible size, composed of 2 individuals. In this memetic algorithm, the classical mutation is replaced by two successive local searches. The first one, *TabuCol*, works with a k-fixed number of colors and tends to decrease the number of conflicting vertices. The second local search, *TabuSum*, works with legal colorings and tends to decrease the sum value of the solution.

Graph				$sumHEAD$				
Name	$\|V\|$	$\|E\|$	f_{UB}	f^*	$Avg.$	$Iter_{TC}$	Cross	Time (min)
anna.col	138	986	276	276	276.00	100	7078.31	0.26
david.col	87	812	237	237	237.00	10	811.53	0
homer.col	561	3258	1150	1151	1153.60	100	36 433.23	45.34
huck.col	74	602	243	243	243.00	10	126.11	0
jean.col	80	508	217	217	217.00	20	1200.00	0
DSJC125.1.col	125	736	326	326	326.00	20	1296.96	0
DSJC125.5.col	125	3891	1012	1012	1012.00	50	63 488.35	2.39
DSJC125.9.col	125	6961	2503	2503	2503.00	200	65.33	0
DSJC250.1.col	250	3218	970	**967**	970.93	50	257 755.60	65.30
DSJC250.5.col	250	15 668	3210	3210	3222.92	400	200 440.11	36.18
DSJC250.9.col	250	27 897	8277	8277	8277.00	8000	16 129.01	9.61
DSJC500.1.col	500	12 458	2835	**2829**	2856.00	200	82 261.06	74.26
DSJC500.5.col	500	62 624	10 881	**10 871**	10 905.42	8000	50 576.48	70.93
DSJC500.9.col	500	224 874	29 856	**29 848**	29 866.90	25 000	15 349.00	43.12
DSJC1000.1.col	1000	49 629	8978	9108	9252.90	4000	11 121.78	71.83
DSJC1000.5.col	1000	249 826	37 567	37 729	38 065.11	140 000	3228.26	73.14
DSJC1000.9.col	1000	449 449	103 429	103 620	103 863.24	180 000	2422.63	74.94
DSJR500.1.col	500	3555	2155	2157	2162.47	200	137 964.50	66.97
DSJR500.1c	500	121 275	16 286	16 286	16 305.00	2000	47 091.37	55.42
DSJR500.5	500	58 862	25 440	**24 749**	24 829.60	4000	104 288.63	95.13
flat300_28_0	300	21 695	4238	**4230**	4263.40	20 000	37 682.46	46.22
flat1000_50_0.col	1000	245 000	25 500	25 500	25 500.00	20 000	179.48	1.92
flat1000_60_0.col	1000	245 830	30 100	30 100	30 100.00	20 000	1751.64	13.78
flat1000_76_0	1000	246 708	37 164	37 298	37 581.51	80 000	5653.27	89.81
le450_15a.col	450	8168	2626	**2621**	2658.79	50	86 881.16	68.51
le450_15b.col	450	8169	2632	2648	2673.80	50	95 675.94	76.16
le450_15c.col	450	16 680	3487	3491	3495.23	1200	52 116.45	84.88
le450_15d.col	450	16 750	3505	3512	3516.04	1200	38 503.94	69.54
le450_25a.col	450	8260	3150	3202	3216.05	20	73 055.41	53.88
le450_25b.col	450	8263	3356	3446	3459.53	20	56 736.93	51.30
le450_25c.col	450	17 343	4501	**4499**	4539.37	200	92 845.51	73.65
le450_25d.col	450	17 425	4532	**4530**	4600.46	50	100 657.10	77.90
miles250.col	128	774	325	325	325.00	10	7585.25	0
miles500.col	128	2340	705	705	705.00	50	51 814.50	1.16
miles750.col	128	4226	1173	1173	1173.00	10	60 257.00	2.20
miles1000.col	128	6432	1666	1666	1666.00	50	312 917.83	13.66
miles1500.col	128	10 396	3354	3354	3354.00	10	475.50	0
mug88_1.col	88	146	178	178	178.00	10	4.90	0
mug88_25.col	88	146	178	178	178.00	20	4.30	0
mug100_1.col	100	166	202	202	202.00	10	6.54	0
mug100_25.col	100	166	202	202	202.00	10	10.54	0
myciel3.col	11	20	21	21	21.00	20	1.75	0
myciel4.col	23	71	45	45	45.00	50	3.53	0
myciel5.col	47	236	93	93	93.00	20	618.75	0
myciel6.col	95	755	189	189	189.00	100	1 136 465.50	19.50
myciel7.col	191	2360	381	381	385.20	50	24 920.98	2.00
queen10_10.col	100	2940	553	553	553.00	50	6464.46	0.03
queen11_11.col	121	3960	733	**726**	729.53	50	964 829.50	42.26
queen12_12.col	144	5192	943	**937**	941.67	50	829 837.32	52.82
queen13_13.col	169	6656	1191	**1188**	1189.82	50	527 650.03	49.60
queen14_14.col	196	8372	1482	**1479**	1480.00	50	358 329.76	56.60
queen15_15.col	225	10 360	1814	**1811**	1812.16	50	271 019.87	47.79
queen16_16.col	256	12 640	2193	**2188**	2190.65	50	224 167.46	55.38
school1.col	385	19 095	2674	2674	2674.00	100	1047.57	1.86
school1_nsh.col	352	14 612	2392	2392	2392.00	1000	92.76	0.02

TABLE I
RESULTS OF *sumHEAD*

Graph	1-move		2-move	
	f^*	$Avg.$	f^*	$Avg.$
DSJC500.1.col	2844	2868.01	2829	2856.00
DSJC500.5.col	10896	10919.46	10 871	10 905.42
DSJC500.9.col	29 857	29 867.78	29 848	29 866.90
le450_15b.col	2717	2741.03	2648	2673.80

TABLE II
2-MOVE VS 1-MOVE LOCAL OPERATORS IN *TabuSum*

Compared to classical population-based approaches, using only 2 individuals allows to remove the selection operator, and the replacement strategy. Moreover, it does not spend too much time to improve many individuals. The risk of a premature convergence with this small population is balanced by an elitist approach. It reintroduces a previously obtained good solution, which has the property of being good in terms of

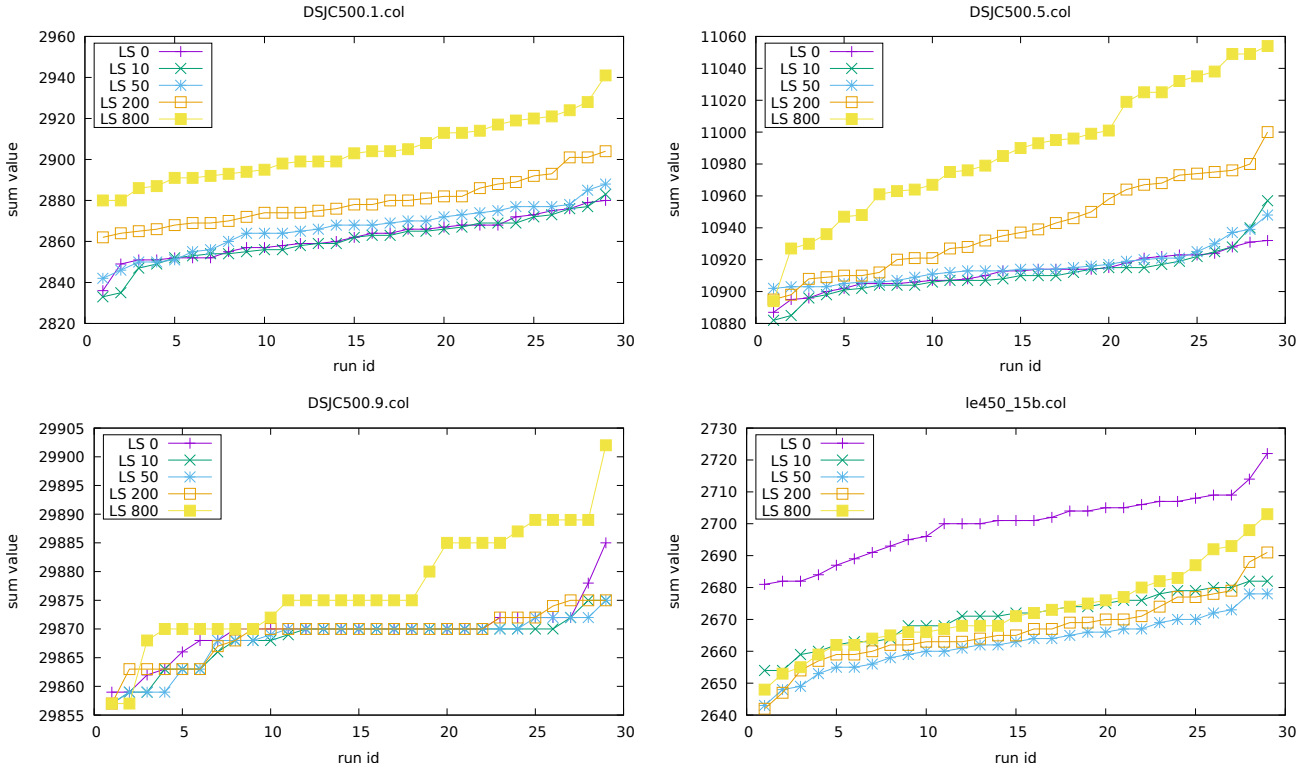

Fig. 4. Analysis of the $TabuSum$ duration impact on the sum values for 4 graphs

intensification, with a good fitness value, and good in terms of diversity, being potentially different because it has not evolved with the population for a number of generations.

The experiments were performed on 55 reference benchmark instances. The proposed $sumHEAD$ algorithm proved to be very competitive compared to the best algorithm known to date. It is able to reach most of the current best upper bounds, and has been able to improve the upper bound of 15 graphs known to be particularly challenging.

Acknowledgment

The experimentation of this work was performed using HPC resources from the Strasbourg Mesocenter.

References

[1] E. Kubicka and A. J. Schwenk, "An introduction to chromatic sums," in *Proceedings of the 17th Conference on ACM Annual Computer Science Conference*, ser. CSC '89. New York, NY, USA: ACM, 1989, pp. 39–45.

[2] Y. Jin and J.-K. Hao, "Hybrid evolutionary search for the minimum sum coloring problem of graphs," *Information Sciences*, vol. 352-353, pp. 15 – 34, 2016.

[3] C. Thomassen, P. Erds, Y. Alavi, P. J. Malde, and A. J. Schwenk, "Tight bounds on the chromatic sum of a connected graph," *Journal of Graph Theory*, vol. 13, no. 3, pp. 353–357, 1989.

[4] Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer, "Greedy Algorithms for the Minimum Sum Coloring Problem," in *Logistique et transports*, Sousse, Tunisia, Mar. 2009, pp. LT–027.

[5] Q. Wu and J.-K. Hao, "An effective heuristic algorithm for sum coloring of graphs," *Computers & Operations Research*, vol. 39, no. 7, pp. 1593 – 1600, 2012.

[6] Q. Wu and J. Hao, "Improved lower bounds for sum coloring via clique decomposition," *CoRR*, vol. abs/1303.6761, 2013.

[7] Y. Jin, J.-K. Hao, and J.-P. Hamiez, "A memetic algorithm for the minimum sum coloring problem," *Computers & Operations Research*, vol. 43, pp. 318 – 327, 2014.

[8] Q. Wu, Q. Zhou, Y. Jin, and J.-K. Hao, "Minimum sum coloring for large graphs with extraction and backward expansion search," *Applied Soft Computing*, vol. 62, pp. 1056 – 1065, 2018.

[9] Y. Jin, J.-P. Hamiez, and J.-K. Hao, "Algorithms for the minimum sum coloring problem: a review," *Artificial Intelligence Review*, vol. 47, no. 3, pp. 367–394, Mar 2017.

[10] F. Furini, E. Malaguti, S. Martin, and I.-C. Ternier, "Ilp models and column generation for the minimum sum coloring problem," *Electronic Notes in Discrete Mathematics*, vol. 64, pp. 215 – 224, 2018, 8th International Network Optimization Conference - INOC 2017.

[11] L. Moalic and A. Gondran, "Variations on memetic algorithms for graph coloring problems," *Journal of Heuristics*, vol. 24, no. 1, pp. 1–24, Feb 2018.

[12] ——, "The new memetic algorithm head for graph coloring: An easy way for managing diversity," in *Evolutionary Computation in Combinatorial Optimization*, ser. Lecture Notes in Computer Science, G. Ochoa and F. Chicano, Eds. Springer International Publishing, 2015, vol. 9026, pp. 173–183.

[13] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York, USA: Plenum Press, 1972, pp. 85–103.

[14] A. Hertz and D. de Werra, "Using Tabu Search Techniques for Graph Coloring," *Computing*, vol. 39, no. 4, pp. 345–351, 1987.

[15] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999.

[16] D. S. Johnson and M. Trick, Eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, RI, USA, 1996, vol. 26.