

Greedy Algorithms for the Minimum Sum Coloring Problem

Yu Li

Corinne Lucet

Aziz Moukrim

Kaoutar Sghiouer

Abstract—In this paper we present our study of greedy algorithms for solving the minimum sum coloring problem (MSCP). We propose two families of greedy algorithms for solving MSCP, and suggest improvements to the two greedy algorithms most often referred to in the literature for solving the graph coloring problem (GCP): DSATUR [1] and RLF [2].

I. INTRODUCTION

Greedy algorithms play an important role in the practical resolution of NP-hard problems. A greedy algorithm is a basic heuristic that builds a solution by iteratively adding the locally best element into the solution according to certain criteria. A greedy algorithm can either be used on its own to obtain a “good” solution, or it can be integrated into global optimization methods, for example, to limit the search space in branch and bound algorithms, or to generate initial solutions in metaheuristics.

In this paper we are interested in greedy algorithms for the Minimum Sum Coloring Problem (MSCP). Since MSCP is closely related to the basic Graph Coloring Problem (GCP), we start our study with GCP and then turn to MSCP. Concerning GCP, although a lot of work has been reported in the literature, little of it concerns greedy algorithms. The most widely used greedy algorithms remain DSATUR and RLF, proposed respectively by Brélaz [1] and Leighton [2]. We provide a comprehensive overview of greedy algorithms for MSCP as well as GCP, and propose two new families of greedy algorithms to solve these problems. The proposed algorithms not only produce good solutions for MSCP, but also improve the results obtained by DSATUR and RLF for GCP. We hope this study will help to design more efficient global optimization algorithms for MSCP in the future.

The paper is organized as follows. In the next section we give a formal definition for GCP and MSCP. In Section 3 we study the existing heuristics DSATUR and RLF, we propose two families of greedy algorithms and analyze their behavior and complexity. In Section 4 we present and analyze experimental results. In Section 5 we conclude the paper.

II. MINIMUM SUM COLORING PROBLEM (MSCP)

In this study we address two related graph problems, the Graph Coloring Problem (GCP) and the Minimum Sum Coloring Problem (MSCP). We consider an undirected graph

$G = (V, E)$, where V is the set of $|V| = n$ vertices and E the set of edges defined on $V \times V$.

A *coloring* of G is a function $c : v \mapsto c(v)$ that assigns to each vertex v a color $c(v)$. A coloring is said to be *feasible* if for any pair of vertices $u, v \in V$ such that $[u, v] \in E$ we have $c(u) \neq c(v)$. When $[u, v] \in E$ and $c(u) = c(v)$ we say that u and v are in *conflict*. If k colors are used in a feasible coloring, then this coloring of G is called a k -*coloring*. The minimum value of k over all the feasible colorings is called the *chromatic number* of the graph, denoted $\chi(G)$. GCP is finding this minimum number of colors. In our study the k colors are represented by the k consecutive integers $1, 2, \dots, k$.

Equivalently, a coloring can be seen as a *partition* X of the set of vertices into k subsets, called *color classes*: X_1, \dots, X_k , where the vertices in the X_i are colored with color i . We denote x_i the number of vertices in X_i .

The assignment perspective and the partition perspective are two alternative viewpoints in the coloring of a graph.

GCP has been the subject of a number of studies in the literature, and many good results have been published. There are different approaches to solving this problem. Exact methods such as branching methods [7] or decomposition methods [9], give an optimal solution, but only for very small graphs or specific graph families, because generally their runtime complexity is exponential with respect to the graph size. The second approach involves metaheuristic methods: tabu methods [3], evolutionary methods [6], hybrids of different methods, and so on. The fastest methods are the greedy heuristics that provide a feasible solution for very large graphs in a very short time, but the solution obtained is not necessarily optimal.

In MSCP the goal is to compute a feasible coloring of the graph as described above, such that the sum of the colors is minimum. This problem was introduced by Kubicka and Schwenk [4], who proved that it is NP-hard. If a feasible coloring of G is denoted X , with respective color classes X_1, \dots, X_k , the associated sum can be written as follows:

$$\begin{aligned} \text{Sum}(X) &= 1.x_1 + 2.x_2 + \dots + k.x_k \\ &\text{with } x_1 \geq \dots \geq x_i \geq \dots \geq x_k. \end{aligned}$$

The optimal value for MSCP is called the *chromatic sum* of G and is denoted $\Sigma(G)$.

$$\Sigma(G) = \min\{\text{Sum}(X) \mid X \text{ is a feasible coloring of } G\}$$

For any such optimal solution the associated number of required colors is termed the *strength* of the graph, and denoted $s(G)$. The strength of G can be greater than its chromatic number. For example, graph T in Fig. 1 is a tree,

Y. Li and C. Lucet, Laboratoire MIS, Université de Picardie Jules Verne, 33 rue St Leu, 80000 Amiens, France Yu.Li@u-picardie.fr, Corinne.Lucet@u-picardie.fr

A. Moukrim and K. Sghiouer, Université de Technologie de Compiègne, Laboratoire Heudiasyc UMR CNRS 6599, BP 20529, 60205 Compiègne, France Aziz.Moukrim@utc.fr, Kaoutar.Sghiouer@utc.fr

so $\chi(T) = 2$, but the optimal solution for MSCP requires 3 colors. We have $\Sigma(T) = 11$ and $s(T) = 3$.

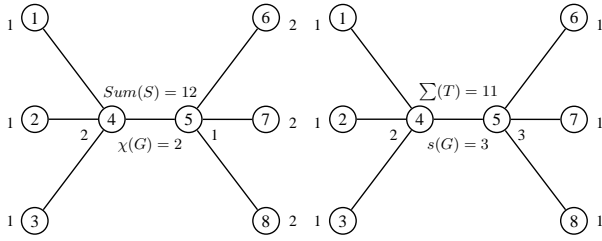


Fig. 1. Tree MSCP

MSCP has been less intensively studied in the literature than GCP. Some theoretical studies exist, but very few numerical results have been presented [10]. The main results are related to particular graph families such as trees for which there exist efficient algorithms for computing an optimal sum coloring. Kroon [5] developed a linear algorithm that computes the optimal solution for tree MSCP. Some works deal with theoretical upper or/and lower bounds of the optimal sum or the strength for some specific graphs. A split-graph is a graph in which the vertices can be partitioned into a clique and an independent set. If we consider the split-graph family, each graph G is composed of two specific vertex sets C and I , such that C is a clique and I is an independent vertex set. If $n_c = |C|$ and $n_i = |I|$ then we have the following relations [8]:

- $n_c \leq s(G) \leq n_c + 1$
- $n_i + \frac{n_c(n_c+1)}{2} \leq \Sigma(G) \leq n_c + n_i + \frac{n_c(n_c+1)}{2}$

Although finding a maximal independent set appears to be a good strategy for the tree in Fig. 1, we can see that this is not true as regards optimal coloring for the split-graph (Fig. 2). Finding good features in order to develop a good heuristic algorithm is thus not simple.

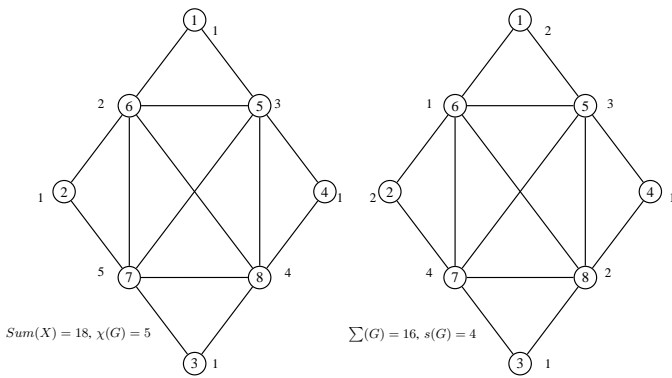


Fig. 2. Split-graph MSCP

III. GREEDY ALGORITHMS FOR MSCP

In this section we first give a general presentation of a greedy algorithm for coloring arbitrary graphs. Then, using this framework, we analyze DSATUR and RLF, before

proposing two families of greedy algorithms for MSCP as well as GCP. Finally, we discuss the similarities and differences between the two families of algorithms.

A. A general presentation of a greedy algorithm

In general, a greedy algorithm comprises four basic elements: an *objective function* f , a *configuration* C , a *candidate set* L_c , and a *greedy rule* r . A formal presentation is outlined below.

Algorithm 1: Greedy algorithm

```

begin
  Generate the initial configuration  $C$ ;
  Generate the list of candidates  $L_c$ ;
  while there are candidates in  $L_c$  do
    select a candidate  $c$  from  $L_c$  according to some
    selection rule  $r$ ;
    modify  $C$  by adding  $c$  into  $C$ ;
    modify  $L_c$  by removing the candidate  $c$  and
    computing new candidates;
  return  $C$ ;
end

```

We define and discuss these elements in the context of coloring a graph.

1) *Objective function*: For GCP, the objective function is to minimize the number of colors, and for MSCP, the objective function is to minimize the sum of colors.

2) *Configuration*: From the assignment perspective, a *configuration* C represents a partial or complete coloring, where t vertices have been colored without any conflict and there remain $n - t$ uncolored vertices; from the partition perspective, a configuration can be also considered as a partial or complete *partition* $X: X_1, \dots, X_i$, where X_i is in construction and all vertices in X_i are colored with color i .

At the outset of a greedy algorithm, the *initial configuration* represents a partial coloring where zero or more vertices are colored. During the execution of the algorithm, the configuration changes and represents a partial coloring in which more and more vertices are colored; finally, at the termination of the algorithm, the *final configuration* represents a complete coloring.

The initial configuration will influence the result of the algorithm. In order to minimize this impact and focus on the algorithm's behavior, we start from an empty initial configuration and let the algorithm choose the first vertex to be colored.

3) *Candidate set*: A *candidate* is an uncolored vertex. In all the existing greedy algorithms for solving GCP, only the smallest available color, noted *cmín*, is considered as a candidate for coloring a vertex v . We follow this convention in order to simplify our analysis and comparison.

The viewpoint is significant: from the assignment perspective the candidate set may contain all uncolored vertices,

whereas from the partition perspective the candidate set is composed only of the uncolored vertices whose $cm\text{in}$ is equal to i , given the current subset X_i .

4) *Greedy rule*: A greedy rule is a rule for choosing the locally best candidate vertex according to certain criteria. A *criterion* is an information or property about candidate vertices. The effectiveness of the greedy rule greatly depends on the quality of the criteria. A good criterion comes from an insight into the problem.

A *tie* appears when there exist several candidates having the same criterion values, making it necessary to apply a *tie-breaking rule* to choose only one candidate. A tie can be broken in a random or deterministic way. The random way consists in choosing a candidate arbitrarily, while the deterministic way involves choosing a candidate according to some rule, for example, always choosing the last one. The random tie break is often used in practice, but it increases the difficulty of studying an algorithm, since it can produce different results each time with the same instance. So we adopt a deterministic tie break which chooses the last candidate encountered from among those that have the same criterion values.

Obviously, the key part of a greedy algorithm is to design a good greedy rule, which is our focus in the following discussion.

B. Greedy algorithms from the assignment perspective

1) *DSATUR*: DSATUR is the most widely-used greedy algorithm for solving GCP. The idea behind DSATUR is to color a vertex having the largest number of colors used by its adjacent vertices, with the aim of making the total number of colors used as small as possible. DSATUR iteratively colors vertices from the assignment perspective.

Given a configuration C and an uncolored vertex v , the adjacent vertices of v can be divided into two parts: the colored vertices, $\Gamma^+(v)$, and the uncolored vertices, $\Gamma^-(v)$. In DSATUR, the number of different colors assigned to the vertices in $\Gamma^+(v)$ is called the *saturation degree* of v , denoted $dsat(v)$.

DSATUR uses $dsat$ as the principal selection criterion. The corresponding greedy rule consists in choosing an uncolored vertex v with the greatest $dsat(v)$. Ties are broken when there exist several vertices having the same saturation degree, by preferring a vertex with the greatest number of uncolored adjacent vertices, denoted $nb\text{VoisinNC}(v)$; if ties still remain, they are broken by choosing the last vertex. The complexity of DSATUR is $O(n^2)$.

The greedy rule of DSATUR can be formally expressed by: $[max(dsat), max(nb\text{VoisinNC})]$

2) *New criteria nbDsatC and nbDsatNC*: DSATUR is a classical greedy algorithm in the sense that it makes decisions on the basis of information without worrying about the impact these decisions may have for the current configuration.

The new criteria consider the impact on configuration of coloring a candidate vertex v with its $cm\text{in}(v)$. After v is colored with its $cm\text{in}(v)$, the $dsat$ of some vertices in $\Gamma^-(v)$ will be modified and incremented by one, and the $dsat$ of the other vertices in $\Gamma^-(v)$ will remain the same. So we can divide the vertices in $\Gamma^-(v)$ into two sets: the vertices which $dsat$ would change, denoted $\Gamma_{dsatC}^-(v)$, and the vertices which $dsat$ would not change, denoted $\Gamma_{dsatNC}^-(v)$. The cardinality of $\Gamma_{dsatC}^-(v)$ is denoted as $nbDsatC(v)$, and the cardinality of $\Gamma_{dsatNC}^-(v)$ as $nbDsatNC(v)$. We take as an example the configuration shown in Fig.3, where v_5 and v_7 are colored with 1, v_4 and v_6 with 2, and v_9 with 3. Then, $dsat$ and $cm\text{in}$ for v_1, v_2, v_3 and v_8 are respectively: $dsat(v_1) = 2$, $cm\text{in}(v_1) = 3$; $dsat(v_2) = 0$, $cm\text{in}(v_2) = 1$; $dsat(v_3) = 3$, $cm\text{in}(v_3) = 4$; $dsat(v_8) = 1$, $cm\text{in}(v_8) = 2$. So if v_1 is colored with 3, $dsat(v_2)$ and $dsat(v_8)$ will be modified, so we have $\Gamma_{dsatC}^-(v_1) = \{v_2, v_8\}$ with $nbDsatC(v_1) = 2$; but there exist no vertices whose $dsat$ will not change, so we have, $\Gamma_{dsatNC}^-(v_1) = \phi$ with $nbDsatNC(v_1) = 0$. Similarly, if v_2, v_3 and v_8 are colored with 1, 4 and 2, we have respectively: $\Gamma_{dsatC}^-(v_2) = \phi$ with $nbDsatC(v_2) = 0$ and $\Gamma_{dsatNC}^-(v_2) = \{v_1, v_3, v_8\}$ with $nbDsatNC(v_2) = 3$; $\Gamma_{dsatC}^-(v_3) = \{2, 8\}$ with $nbDsatC(v_3) = 2$ and $\Gamma_{dsatNC}^-(v_3) = \phi$ with $nbDsatNC(v_3) = 0$; $\Gamma_{dsatC}^-(v_8) = \{2\}$ with $nbDsatC(v_8) = 1$ and $\Gamma_{dsatNC}^-(v_8) = \{1, 3\}$ with $nbDsatNC(v_8) = 2$.

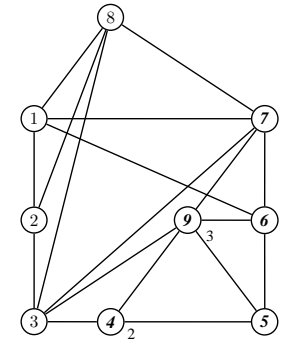


Fig. 3. Example of configuration

We see intuitively that the smaller the value of $nbDsatC(v)$, the smaller its impact on the configuration C . Likewise, the larger the value of $nbDsatNC(v)$, the smaller its impact on C . The results in Tables I and III show that $nbDsatC$ and $nbDsatNC$ contribute to MSCP and GCP in different

ways, $[min(nbDsatC)]$ being more correlated to MSCP and $[max(nbDsatNC)]$ more correlated to GCP.

3) *New greedy rules*: From a global point of view, a good decision should influence the current configuration as little as possible, and so we propose the following greedy rules with the aim of minimizing the impact on the current configuration:

- Examine sequentially $nbDsatNC$ and $nbDsatC$. There are two cases to be considered:
 - $[max(nbDsatNC), min(nbDsatC)]$: The algorithm consists in choosing an uncolored vertex v with maximum $nbDsatNC(v)$. If there exist several candidates having the same value, it selects the one with the smallest $nbDsatC$. We denote this algorithm MDSAT(1).
 - $[min(nbDsatC), max(nbDsatNC)]$: The algorithm consists in choosing an uncolored vertex v with minimum $nbDsatC(v)$. If there exist several candidates having the same value, it selects the one with the largest $nbDsatNC$. We denote this algorithm MDSAT(2).
- Examine in parallel $nbDsatC$ and $nbDsatNC$. We have:
 - $[min(nbDsatC/nbDsatNC)]$: The algorithm consists in choosing an uncolored vertex v with minimum $nbDsatC(v)/nbDsatNC(v)$. We denote this algorithm MDSAT(3).
 - $[min(nbDsatC/nbDsatNC), min(nbDsatC)]$: The algorithm consists in choosing an uncolored vertex v with minimum $nbDsatC(v)/nbDsatNC(v)$. If there exist several candidates having the same value, it selects the one with the smallest $nbDsatC$. We denote this algorithm MDSAT(4).
 - $[min(nbDsatC/nbDsatNC), max(nbDsatNC)]$: The algorithm consists in choosing an uncolored vertex v with minimum $nbDsatC(v)/nbDsatNC(v)$. If there exist several candidates having the same value, it selects the one with the largest $nbDsatNC$. We denote this algorithm MDSAT(5).

The tie-breaking rule is to select the vertex having the maximum label number. The complexity of each MDSAT(n) is $O(n^3)$.

C. Greedy algorithms from the partition perspective

1) *RLF*: RLF is the next best-known greedy algorithm for solving GCP. The idea of RLF is to color as many vertices as possible with one color before going to another color from the partition perspective.

Take a partition X_1, \dots, X_i where the color class X_i is under construction, and the candidates are the uncolored vertices whose $cmin$ is equal to i . For such a candidate v , its uncolored adjacent vertices in $\Gamma^-(v)$ can be divided into two sets: the vertices (denoted $\Gamma_{cminC}^-(v)$,) whose $cmin$ are equal to i , and the vertices (denoted $\Gamma_{cminNC}^-(v)$) whose $cmin$ are not equal to i . The $cmin$ for any vertex in $\Gamma_{cminC}^-(v)$, would change if v is colored with the color i , while the $cmin$ for any vertex in $\Gamma_{cminNC}^-(v)$ would remain the same. We denote the cardinality of $\Gamma_{cminC}^-(v)$ as $nbCminC(v)$ and the cardinality of $\Gamma_{cminNC}^-(v)$ as $nbCminNC(v)$.

RLF uses $nbCminNC$ as the principal criterion and $nbCminC$ as the second one. The corresponding greedy rule involves selecting the candidate vertex with the greatest $nbCminNC$, and if several such vertices exist, it chooses the vertex with the minimum $nbCminC$. The greedy rule of RLF can be formally expressed as: $[min(cmin), max(nbCminNC), min(nbCminC)]$

2) *MRLF*: It can be seen intuitively that coloring a candidate vertex v with the greatest $nbCminNC(v)$ means reducing to a minimum the probability of using a new color next. And coloring a candidate vertex v with the smallest $nbCminC(v)$ means keeping the size of the current class color X_i as large as possible. Thus, the two criteria should all be related to MSCP as well as GCP, which is confirmed by the experimental results in Tables II and IV, where we test the greedy rules $[max(nbCminNC)]$ and $[min(nbCminC)]$ on the instances in the literature.

So as to minimize the impact on the current configuration like we do for MDSAT(n), we extend RLF by combining two criteria in different ways and propose different greedy rules:

- Examine sequentially $nbCminC$ and $nbCminNC$. We have:
 - $[min(cmin), max(nbCminNC), min(nbCminC)]$, denoted MRLF(1), which corresponds to RLF.
 - $[min(cmin), min(nbCminC), max(nbCminNC)]$: The algorithm consists in choosing a candidate vertex v with minimum $nbCminC(v)$. If there exist several candidates having the same value, it selects the one with the largest $nbCminNC$. We denote this algorithm MRLF(2).
- Examine in parallel $nbCminC$ and $nbCminNC$. We have:
 - $[min(cmin), min(nbCminC/nbCminNC)]$: The algorithm consists in choosing a candidate vertex v with minimum $nbCminC(v)/nbCminNC(v)$. We denote this algorithm MRLF(3).
 - $[min(cmin), min(nbCminC/nbCminNC), min(nbCminC)]$: The algorithm consists in

choosing a candidate vertex v with minimum $nbCminC(v)/nbCminNC(v)$. If there exist several candidates having the same value, it selects the one with the smallest $nbCminC$. We denote this algorithm MRLF(4).

- $[min(cmin), min(nbCminC/nbCminNC), max(nbCminNC)]$: The algorithm consists in choosing an uncolored vertex v with minimum $nbCminC(v)/nbCminNC(v)$. If there exist several candidates having the same such value, it selects the one with the largest $nbCminNC$. We denote this algorithm MRLF(5).

The tie-breaking rule is to select the vertex with the largest label number. The complexity of MRLF(n) is $O(n^3)$.

D. Comparison of the two types of greedy algorithms

We compare MDSAT(n) and MRLF(n) with respect to three aspects: the computation time required to obtain a solution, the quality of the solution, and the way the solution is constructed. MDSAT(n) and MRLF(n) have the same complexity, $O(n^3)$.

Concerning the quality of the solution, MDSAT(n) and MRLF(n) have similar behaviors. They produce nearly the same solutions as GCP and MSCP for the uniform random graphs DSJC [11]. This similarity can be partially explained by their similar strategies for creating greedy rules, which are all based on the principle of minimizing the impact on the current configuration.

As regards the third aspect, they are quite different. MDSAT(n) colors vertex by vertex and, unlike MRLF(n), does not necessarily color class by color class. It is our intention to probe this difference further in future work.

IV. EXPERIMENTAL COMPARISON OF MDSAT(N) AND MRLF(N)

In this section, we give the experimental results for MDSAT(n) and MRLF(n) on some DIMACS instances [11]. The instances used include those that have been recognized as being among the most difficult to solve, including uniform DSJC random graphs.

Each of the algorithms MDSAT(n) and MRLF(n) was programmed in C and run on an Intel Dual Core T5450-1.66-1.67 (without parallelization of the program), with 2GB of RAM, running under Windows Vista Home Premium.

For GCP we compare the results obtained by MDSAT(n) and MRLF(n) and those obtained by DSATUR and RLF.

For MSCP, since there exist no numerical results, we look at the best result obtained by either MDSAT(n) or MRLF(n), which we denote $nbBest$.

The comparison between different criteria is then based on $nbBest$.

A. Experimental results for GCP

Tables I and II show the number of GCP colors obtained by each of the algorithms MDSAT(n) and MRLF(n). MDSAT(3) and MDSAT(4) give the best overall solutions, and these are equivalent respectively to MRLF(3) and MRLF(4), as we mentioned in Section III.

| Instances | DSATUR | MDSAT(1) | MDSAT(2) | MDSAT(3) | MDSAT(4) | MDSAT(5) |
|---------------|--------|----------|----------|----------|----------|----------|
| dsjc125.1 | 7 | 6 | 7 | 6 | 6 | 7 |
| dsjc125.5 | 21 | 21 | 22 | 21 | 21 | 21 |
| dsjc125.9 | 50 | 52 | 50 | 50 | 50 | 50 |
| dsjc250.1 | 10 | 10 | 12 | 10 | 10 | 10 |
| dsjc250.5 | 36 | 35 | 37 | 34 | 34 | 36 |
| dsjc250.9 | 90 | 86 | 84 | 83 | 83 | 84 |
| dsjc500.1 | 15 | 15 | 18 | 16 | 15 | 16 |
| dsjc500.5 | 66 | 61 | 61 | 59 | 59 | 60 |
| dsjc500.9 | 163 | 159 | 148 | 148 | 148 | 149 |
| dsjc1000.1 | 26 | 24 | 27 | 25 | 25 | 25 |
| dsjc1000.5 | 114 | 109 | 107 | 103 | 103 | 104 |
| dsjc1000.9 | 305 | 286 | 272 | 265 | 265 | 268 |
| <i>nbBest</i> | 4 | 5 | 2 | 10 | 11 | 3 |

TABLE I
RESULTS OF MDSAT(N) FOR GCP

| Instances | MRLF(1) | MRLF(2) | MRLF(3) | MRLF(4) | MRLF(5) |
|---------------|---------|---------|---------|---------|---------|
| dsjc125.1 | 6 | 8 | 6 | 6 | 7 |
| dsjc125.5 | 21 | 22 | 21 | 21 | 21 |
| dsjc125.9 | 52 | 50 | 50 | 50 | 50 |
| dsjc250.1 | 10 | 11 | 10 | 10 | 10 |
| dsjc250.5 | 35 | 37 | 34 | 34 | 36 |
| dsjc250.9 | 86 | 84 | 83 | 83 | 84 |
| dsjc500.1 | 15 | 18 | 16 | 15 | 16 |
| dsjc500.5 | 61 | 61 | 59 | 59 | 60 |
| dsjc500.9 | 159 | 148 | 148 | 148 | 149 |
| dsjc1000.1 | 24 | 27 | 25 | 25 | 25 |
| dsjc1000.5 | 109 | 107 | 103 | 103 | 104 |
| dsjc1000.9 | 286 | 272 | 265 | 265 | 268 |
| <i>nbBest</i> | 5 | 2 | 10 | 11 | 3 |

TABLE II
RESULTS OF MRLF(N) FOR GCP

B. Experimental results for MSCP

Tables III and IV show the minimum sum of colors for MSCP obtained by each of the algorithms MDSAT(n) and MRLF(n). MDSAT(4) and MDSAT(5) give the best overall solutions, and these are equivalent respectively to MRLF(4) and MRLF(5).

| Instances | DSATUR | MDSAT(1) | MDSAT(2) | MDSAT(3) | MDSAT(4) | MDSAT(5) |
|---------------|--------|----------|-------------|---------------|---------------|--------------|
| dsjc125.1 | 402 | 362 | 364 | 352 | 353 | 353 |
| dsjc125.5 | 1257 | 1210 | 1141 | 1147 | 1147 | 1151 |
| dsjc125.9 | 2782 | 2880 | 2653 | 2703 | 2703 | 2653 |
| dsjc250.1 | 1191 | 1145 | 1095 | 1107 | 1068 | 1080 |
| dsjc250.5 | 4284 | 4098 | 3829 | 3658 | 3658 | 3851 |
| dsjc250.9 | 10350 | 9792 | 9002 | 9077 | 9077 | 8942 |
| dsjc500.1 | 3668 | 3492 | 3328 | 3233 | 3234 | 3229 |
| dsjc500.5 | 15140 | 13939 | 12845 | 12742 | 12742 | 12717 |
| dsjc500.9 | 37973 | 35856 | 32967 | 32703 | 32703 | 33015 |
| dsjc1000.1 | 12223 | 11623 | 10581 | 10325 | 10276 | 10416 |
| dsjc1000.5 | 53139 | 50395 | 45702 | 45757 | 45757 | 45408 |
| dsjc1000.9 | 139209 | 130105 | 121103 | 119111 | 119111 | 119671 |
| <i>nbBest</i> | 0 | 0 | 2 | 4 | 5 | 5 |

TABLE III
RESULTS OF MDSAT(N) FOR MSCP

| Instances | MRLF(1) | MRLF(2) | MRLF(3) | MRLF(4) | MRLF(5) |
|---------------|---------|-------------|---------------|---------------|--------------|
| dsjc125.1 | 362 | 361 | 352 | 353 | 353 |
| dsjc125.5 | 1210 | 1141 | 1147 | 1147 | 1151 |
| dsjc125.9 | 2880 | 2653 | 2703 | 2703 | 2653 |
| dsjc250.1 | 1145 | 1090 | 1107 | 1068 | 1080 |
| dsjc250.5 | 4098 | 3829 | 3658 | 3658 | 3851 |
| dsjc250.9 | 9792 | 9002 | 9077 | 9077 | 8942 |
| dsjc500.1 | 3492 | 3328 | 3233 | 3234 | 3229 |
| dsjc500.5 | 13939 | 12845 | 12742 | 12742 | 12717 |
| dsjc500.9 | 33015 | 32967 | 32703 | 32703 | 33015 |
| dsjc1000.1 | 10416 | 10582 | 10325 | 10276 | 10416 |
| dsjc1000.5 | 50395 | 45702 | 45757 | 45757 | 45408 |
| dsjc1000.9 | 130105 | 121103 | 119111 | 119111 | 119671 |
| <i>nbBest</i> | 0 | 2 | 4 | 5 | 5 |

TABLE IV
RESULTS OF MRLF(N) FOR MSCP

V. CONCLUSION

In this paper we investigated greedy algorithms for MSCP as well as GCP. We began by studying the two greedy algorithms DSATUR and RLF for GCP, and proposed a family of greedy algorithms MDSAT(n) based on the criteria *nbDsatC* and *nbDsatNC*, which is the improvement and extension of DSATUR. The other family of greedy algorithms MRLF(n) is based on *nbCminC* and *nbCminNC*, which correspond to an extension of RLF.

The main idea behind the two families of greedy algorithms is to examine the impact on the current configuration produced by coloring a candidate vertex. They both choose a candidate that causes the minimum impact. MDSAT(n) and MRLF(n) not only produce good solutions for MSCP but also improve the results obtained by DSATUR and RLF for GCP.

REFERENCES

[1] D. Brélaz, "New methods to color the vertices of a graph", *ACM*, 1979, volume 22, number 4.

[2] FT, Leighton, "A graph Coloring Algorithm for Large Scheduling Problems", *Journal of research of the national institute of standards and technology*, 1979, volume 84, number 6.

[3] A. Hertz, D. De Werra, "Using Tabu search techniques for graph coloring", *Computing*, 1987, 39, pp 345-351.

[4] E. Kubicka, A. J. Schwenk, "An introduction to chromatic sum", *Proc. 17th Annual ACM Computer Science Conf.*, 1989.

[5] Leo G. Kroon, A. Sen, H. Deng, A. Roy, "The Optimal Cost Chromatic Partition Problem for Trees and Interval Graphs", *Graph-Theoretic Concepts in Computer Science*, 1997, pp 279-292.

[6] P. Galinier and J.K. Hao, "hybrid evolutionary algorithms for graph coloring", *Journal of Combinatorial Optimization*, 1999, pp 379-397.

[7] I. Mendez Diaz and P. Zabala, "A branch-and-cut algorithm for graphcoloring", *Computational Symposium on Graph Coloring and its Generalizations (COLOR02)*, 2002, Ithaca, N-Y.

[8] M. R. Salavatipour, "On sum coloring of graphs", *Discrete Appl. Math.*, 2003, volume 127, pp 200-203.

[9] C. Lucet, F. Mendes, A. Moukrim, "An Exact Method for Graph Coloring", *Computers and Operations Research*, 2006, volume 33, number 8, pp. 2189-2207.

[10] Z. Kokosinski and K. Kwarciany, "On Sum Coloring of Graphs with Parallel Genetic Algorithms", *ICANNGA*, 2007, part I, LNCS 4431, pp. 211-219.

[11] <http://mat.gsia.cmu.edu/COLOR/instances.html>.