A study of Breakout Local Search for the minimum sum coloring problem

Una Benlic and Jin-Kao Hao
email: {benlic,hao}@info.univ-angers.fr

LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers Cedex 01, France

Abstract. Given an undirected graph $G = (V, E)$, the minimum sum coloring problem (MSCP) is to find a legal assignment of colors (represented by natural numbers) to each vertex of G such that the total sum of the colors assigned to the vertices is minimized. In this paper, we present Breakout Local Search (BLS) for MSCP which combines some essential features of several well-established metaheuristics. BLS explores the search space by a joint use of local search and adaptive perturbation strategies. Tested on 27 commonly used benchmark instances, our algorithm shows competitive performance with respect to recently proposed heuristics and is able to find new record-breaking results for 4 instances.

Keywords: minimum sum coloring, adaptive perturbation strategy, heuristic, combinatorial optimization

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E, a legal k-coloring of G is a mapping $C : V \rightarrow \{1, ..., k\}$ such that $\forall\{v, u\} \in E, C(v) \neq C(u)$, i.e., no two adjacent vertices are assigned the same color label. A legal k-coloring C can also be represented as a partition of vertex set V into k mutually disjoint independent sets (also called color classes) $\{S_1, ..., S_k\}$ such that $\bigcup_i S_i = V$ and $v \in S_i$ if $C(v) = i$ (v receives color label i). The well-known NP-hard *graph coloring problem* (GCP) is to determine a legal k-coloring with the minimum value k. A related problem to the GCP is *the minimum sum coloring problem* (MSCP) which consists in finding a legal coloring $C = \{S_1, ..., S_k\}$ such that the following total sum of color labels is minimized:

$$Sum(C) = \sum_{i=1}^{k} i \cdot |S_i| \tag{1}$$

where $|S_1| \geq ... \geq |S_k|$ and $|S_i|$ is the size of S_i (i.e., number of vertices in S_i).

The MSCP is known to be NP-hard with several practical applications including VLSI design, scheduling, and distributed resource allocation (see [12] for a list of references). Over the past two decades, it has been studied mainly from a theoretical point of view. Only recently, several heuristics have been proposed for the practical solving of the general MSCP [3, 4, 6, 9, 10, 13].

In this work, we introduce the Breakout Local Search (BLS) for the MSCP. BLS follows the basic scheme of iterated local search [11] and combines features from other well-known methods including tabu search [5] and simulated annealing [8]. The basic idea of BLS is to use descent-based local search to discover local optima and employ adaptive perturbations to continually move from one attractor to another in the search space. Based on the information on the state of search, the perturbation strategy of BLS introduces a varying degree of diversification by dynamically determining the number of moves for perturbation and by adaptively selecting between several types of dedicated moves.

We evaluate the performance of BLS on 27 benchmark graphs which are commonly used in the literature to test MSCP algorithms. Despite its simplicity, BLS shows competitive results with respect to the most recent MSCP heuristics.

2 Breakout Local Search (BLS) for the MSCP

2.1 General BLS procedure

Recall that a local optimum with respect to a given neighborhood N is a solution s^* such that $\forall s \in N(s^*)$, $f(s^*) \leq f(s)$, where f is the objective function to be minimized. A basin of attraction of a local optimum l is the set B_l of solutions that lead the local search to the given local optimum l, i.e., $B_l = \{s \in S | LocalSearch(s) = l\}$. Since a local optimum l acts as an attractor with respect to the solutions B_l, the terms attractor and local optimum will be used interchangeably throughout this work.

Basically, our Breakout Local Search (BLS) approach moves from one basin of attraction formed by a local optimum to another basin by applying a smaller or larger number of perturbation moves whose type (e.g., random or directed moves) is determined adaptively. BLS starts from an initial solution C_0 and applies to it local search (descent) to reach a local optimum or an attractor C. Each iteration of the local search algorithm scans the whole neighborhood and selects the best improving neighboring solution to replace the current solution. If no improving neighbor exists, local optimality is reached. At this point, BLS tries to escape from the basin of attraction of the current local optimum and move into a neighboring basin of attraction. For this purpose, BLS applies a number of dedicated moves to the current optimum C (we say that C is perturbed). Each time an attractor is perturbed, the perturbed solution is used as the new starting point for the next round of the local search procedure.

If the search returns to the last attractor C, BLS perturbs C more strongly by increasing the number of moves to be applied for perturbation. After visiting a certain number of local optima without improving the best solution found so far, BLS applies a significantly stronger perturbation in order to drive definitively the search toward a new and more distant region in the search space.

The success of the described method depends crucially on two factors. First, it is important to determine the number L of perturbation moves (also called "jump magnitude") to be applied to change or perturb the solution. Second,

it is equally important to consider the type of perturbation moves to be applied. While conventional perturbations are often based on random moves, more focused perturbations using dedicated information could be more effective. The degree of diversification, introduced by a perturbation mechanism, depends both on the jump magnitude and the type of moves used for perturbation. A weak diversification induces a high probability for the local search procedure to cycle between two or more locally optimal solutions, leading to search stagnation. On the other hand, a too strong diversification will have the same effect as a random restart, which usually results in a low probability of finding better solutions in the following local search phases.

Algorithm 1 Breakout Local Search

Require: Initial jump magnitude L_0, jump magnitude L_s for strongest perturbation, max. number T of non-improving attractors visited before strong perturb.
Ensure: A solution C_{best}.
1: $GenerateInitialSolution(C)$
2: $C_{best} \leftarrow C$ /* C_{best} records the best solution found so far */
3: $C_p \leftarrow C$ /* C_p records the last local optimum */
4: $\omega \leftarrow 0$ /* Set counter for consecutive non-improving local optima */
5: **while** stopping condition not reached **do**
6: **if** Solution C is not legal **then**
7: $C \leftarrow LocalSearch2(C)$
8: **end if**
9: $C \leftarrow LocalSearch1(C)$
10: **if** $(Sum(C) < Sum(C_{best}))$ and (C is a legal solution) **then**
11: $C_{best} \leftarrow C$; /* Update the best solution found so far */
12: $\omega \leftarrow 0$ /* Reset the counter of consecutive non-improving local optima */
13: **else**
14: $\omega \leftarrow \omega + 1$
15: **end if**
16: /* Determine the perturbation strength L to be applied to C */
17: **if** $\omega > T$ **then**
18: /* Search is stagnating, strongest perturbation required */
19: $L \leftarrow L_s$
20: **else if** $C = C_p$ **then**
21: /* Search returned to previous local optimum, increment perturb. strength */
22: $L \leftarrow L + 1$
23: **else**
24: /* Search escaped from previous local optimum, reinitialize perturb. strength */
25: $L \leftarrow L_0$
26: **end if**
27: /* Perturb the current local optimum C with L perturbation moves */
28: $C_p \leftarrow C$
29: $C \leftarrow Perturbation(C, L)$
30: **end while**

Algorithm 1 presents the general BLS algorithm for the MSCP, whose ingredients are detailed in the following sections. BLS starts from an initial random solution C which may be a conflicting coloring (line 1). To reach a legal coloring, BLS employs two different local search procedures (lines 6-9, see next section), based on different move operators and evaluation functions. Once a local optimum is reached, the jump magnitude L is determined depending on whether the search escaped or returned to the previous local optimum, and whether the search is stagnating in a non-promising region (lines 17-26). BLS then applies L

perturbation moves to C in order to get a new starting point for the local search procedures (line 29).

2.2 Neighborhood relations and evaluation functions

As previously explained, a solution to sum coloring with k colors can be represented as a k-partition $C = \{S_1, S_2, ..., S_k\}$ of vertices into k disjoint subsets $S_1, ..., S_k$, where each subset $S_i, i \in \{1, ..., k\}$ is associated with a unique integer i (i.e., color). The objective of MSCP is to find a coloring C that minimizes the total sum of colors assigned to vertices, while its implicit constraint requires that any two adjacent vertices $\{u, v\} \in E$ belong to different subsets. If this constraint is violated, we say the coloring is illegal with conflicting vertices.

A common move for minimum sum coloring and graph coloring problems is the exchange move (v, j), which consists in moving a vertex (element) $v \in V$ from its current subset S_i to another subset S_j (i.e., changing the color of v from i to j). The proposed BLS algorithm distinguishes two types of move exchange operators. The first type of move operators only considers exchange moves that do not violate the implicit problem constraint. For the second type of move operators, the implicit constraint is not strictly imposed in order to permit more freedom during the search. These two move operators are used in BLS under specific conditions as explained below.

If the solution C is a legal coloring, BLS applies to C a local search procedure (call it $LocalSearch1$) which consists in identifying the exchange move that decreases the most the objective value of Eq. (1) such that the new coloring remains legal. This process is repeated until reaching a local optimum.

If the solution C is an illegal coloring (i.e., with conflicting vertices), before applying $LocalSearch1$, BLS first applies another local search procedure (call it $LocalSearch2$) which evaluates all the moves (i.e., $\forall v \in V$, and $\forall j \in \{1, ..., k\}$ and $C(v) \neq j$) by considering both the variation $\Delta_{conf}(v, j)$ in the number of conflicts and the variation $\Delta_{sc}(v, j)$ in the sum of colors when exchanging the color of vertex v to j. Both $\Delta_{conf}(v, j)$ and $\Delta_{sc}(v, j)$ are negative for an improving move. Each move is then evaluated with the following relation:

$$\Delta f(v, j) = \Delta_{conf}(v, j) * \gamma(v, j), \text{ where} \tag{2}$$

$$\gamma(v, j) = \begin{cases} abs(\Delta_{sc}(v, j)) + k + 1 & \text{if } \Delta_{sc}(v, j) < 0 \\ k - \Delta_{sc}(v, j) + 1 & \text{otherwise} \end{cases} \tag{3}$$

$LocalSearch2$ performs at each steep an exchange move with the smallest Δf, and stops after reaching a solution with no conflicting vertices (i.e., when $\Delta f = 0$). The evaluation function from Eq. 2 ensures that a conflicting vertex is eventually assigned a color $k + 1$ in case the conflict cannot be resolved by changing the color of v to a color less than or equal to k. However, a local optimum attained with $LocalSearch2$ is not necessarily a local optimum with respect to $LocalSearch1$. After reaching a local optimum with $LocalSearch2$, BLS thus applies $LocalSearch1$ to this local optimum which may improve the solution in terms of the objective value of Eq. (1).

Upon reaching a feasible locally optimal solution, BLS triggers its perturbation mechanism as described in the following section.

2.3 Adaptive perturbation mechanism

General idea The perturbation mechanism plays a crucial role within BLS since the descent-based local search alone cannot escape from a local optimum. BLS thus tries to move to the another basin of attraction by applying perturbations of different intensities depending on the search state. The idea of BLS is to first explore neighboring attractors. Therefore, after the local search phase, BLS performs most of the time a weak perturbation (by applying a small number L of moves) that is hopefully just strong enough to escape the current basin of attraction and to fall under the influence of a neighboring local optimum. If the jump was not sufficient to escape the current attractor, the perturbation strength L is incremented and the perturbation is applied again to the current attractor (lines 20–23 of Alg. 1). After visiting consecutively T legal local optima without any improvement of the best solution found, BLS sets the number of perturbation moves L to a significantly larger value L_s (see lines 17–19 of Alg. 1). In addition to the number of perturbation moves, we also determine which type of moves are to be applied. Instead of making random jumps all the time, BLS alternates between several types of dedicated perturbation moves, depending on the desired amount of diversification (i.e., the state of search).

The perturbation strategies The proposed BLS algorithm employs two directed perturbation strategies (Dp_1 and Dp_2), a recency-based perturbation (RB_p) and a random perturbation (R_p).

Directed perturbations are based on the idea of tabu list from tabu search [5]. These perturbations use a selection rule that favors the moves that minimize solution degradation, under the constraint that the moves are not prohibited by the tabu list. Move prohibition is determined in the following way. Each time vertex v is moved from subset S_i to S_j, it is forbidden to move v back to S_i for the next tt iterations (tt takes a random value from a given range). The information for move prohibition is maintained in a matrix H where the element $H_{v,j}$ is the iteration number when vertex v was last moved to subset S_j. The tabu status of a move is neglected only if the move leads to a new solution better than the best solution found so far. The *directed perturbations* rely thus both on history information and the quality of the moves to be applied for perturbation.

The first directed perturbation Dp_1 consists in making a legal non-tabu move which reduces the most the objective value, under constraint that the move does not violate an additional problem constraint. The second directed perturbation Dp_2 consists in performing a non-tabu exchange move (v, j) of the smallest $\Delta f(v, j)$, $\forall v \in V$, and $\forall j \in \{1, ..., k\}$ and $C(v) \neq j$ (see Eq. 2), such that the number of vertices in S_j is greater than or equal to the number of vertices in the current subset of v.

The recency-based perturbation RB_p consists in making the least recent legal exchange move (v, j) (i.e., a move that would not violate an additional constraint) provided that the subset S_j is not empty.

The move with random perturbation R_p consists first in selecting randomly two non-empty subsets S_i and S_j, such that $|S_i| \leq |S_j|$. A vertex v is then randomly chosen from S_i and moved to its new subset S_j.

Since moves with perturbations Dp_2 and R_p always lead to an illegal solution, we consider them to be stronger than perturbations Dp_1 and RB_p.

Our BLS approach takes turns probabilistically between a weaker perturbation (Dp_1 or RB_p) and a stronger perturbation (Dp_2 or R_p) depending on the search state, i.e., the current number of consecutive non-improving legal attractors visited ω. The idea is to apply weaker perturbation with a higher probability as the search progresses toward improved new solutions (when ω is small). With the increase of ω, the probability of using a stronger perturbation increases for the purpose of an important diversification.

Additionally, it has been observed from an experimental analysis that it is useful to guarantee a minimum of applications of a weaker perturbation. Therefore, we constraint the probability P of applying perturbations Dp_1 or RB_p to take values no smaller than a threshold P_0:

$$P = \begin{cases} e^{-\omega/T} & \text{if } e^{-\omega/T} > P_0 \\ P_0 & \text{otherwise} \end{cases} \tag{4}$$

where T is the maximum number of legal non-improving local optima visited before carrying out a stronger perturbation.

Given the probability P of applying perturbation Dp_1 or RB_p over Dp_2 or R_p, the probability of applying perturbation Dp_1 over RB_p is $P \cdot Q_1$ and $P \cdot (1 - Q_1)$ respectively, while the probability of applying Dp_2 over R_p is defined by $(1 - P) \cdot Q_2$ and $(1 - P) \cdot (1 - Q_2)$ respectively. Q_1 and Q_2 are two coefficients that take a value from $[0, 1]$.

2.4 Experimental results

Experimental protocol Our BLS algorithm is programmed in C++, and compiled with GNU gcc on a Xeon E5440 with 2.83 GHz and 2 GB. Two sets of benchmark graphs from the literature which are commonly used to test sum coloring algorithms are considered in the experiments. The first set is composed of 11 DIMACS[1] (DSJC125.1 to DSJC1000.5) instances. The second benchmark set is composed of 16 graphs from the COLOR02 competition website[2]. For all the instances, we run our BLS algorithm 20 times, with the maximum time limit set to 2 hours. However, BLS often attains its best result long before this limit. The parameter settings used to obtain the reported results are the following: initial jump magnitude $L_0 = 0.1 * |V|$, jump magnitude during strong perturbation $L_s = 0.25 * |V|$, maximum number of non-improving legal attractors

[1] ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/
[2] http://mat.gsia.cmu.edu/COLOR02/

visited before strong perturbation $T = 100$, smallest probability for applying weaker perturbation P_0 and probability coefficients Q_1 and Q_2 are set to 0.5.

We compare the performance of BLS with six recent algorithms from the literature. The comparison is solely based on the quality criterion according to Eq. (1) since information like the computing time are not always available for the reference approaches.

Table 1. Computational results of BLS on DIMACS and COLOR02 instances.

Name	Sum*	Sum	Avg(Std)	t(min)	*Name*	Sum*	Sum	Avg(Std)	t(min)
DSJC125.1	326	326	326.9 (0.6)	18.3	jean	217	217	217.0 (0.0)	0.1
DSJC125.5	1015	**1012**	1012.9 (0.3)	51.0	queen5.5	75	75	75.0 (0.0)	0.0
DSJC125.9	2511	**2503**	2503.0 (0.0)	1.1	queen6.6	138	138	138.0 (0.0)	0.0
DSJC250.1	977	**973**	982.5 (3.4)	111.7	queen7.7	196	196	196.0 (0.0)	0.0
DSJC250.5	3246	**3219**	3248.5 (14.5)	104.1	queen8.8	291	291	291.0 (0.0)	0.8
DSJC250.9	8286	8290	8316 (13.2)	41.6	games120	443	443	443.0 (0.0)	0.5
DSJC500.1	2850	2882	2942.9 (19.6)	112.4	miles250	325	327	328.8 (0.9)	31.1
DSJC500.5	10910	11187	11326.3 (75.3)	118.8	miles500	709	710	713.3 (1.2)	86.3
DSJC500.9	29912	30097	30259.2 (63.3)	37.1	myciel3	21	21	21 (0.0)	0.0
DSJC1000.1	9003	9520	9630.1 (65.7)	112.9	myciel4	45	45	45.0 (0.0)	0.0
DSJC1000.5	37598	40661	41002.6 (169.3)	113	myciel5	93	93	93.0 (0.0)	1.3
anna	276	276	276.0 (0.0)	14.8	myciel6	189	189	196.6 (4.3)	20.0
david	237	237	237.0 (0.0)	2.5	myciel7	381	381	393.8 (8.4)	38.3
huck	243	243	243.0 (0.0)	0.3					

Computational results and comparisons Table 1 presents computational results of our BLS algorithm on the sets of 27 DIMACS and COLOR02 instances. Column *Sum** shows the best-known results for these instances taken from references [6, 13]. Column *Sum* indicates the best result obtained with BLS after 20 independent runs. Columns *Avg(Std)* and *t(min)* provide respectively the average and standard deviation values over 20 executions and the average CPU time in minutes required by BLS to reach its best result. From Table 1, we observe that for the DIMACS instance, BLS improved the best-known result for four instances, but failed to reach the best reported result for six other (larger) instances. The results for instances of COLOR02 benchmark indicate that BLS attained the best-known result for 14 instances, and was unable to reach the current best result for two instances. The average computing times required by BLS to attain its best reported results from column *Sum* range from less then a minute up to 120 minutes for the largest instances.

To further evaluate the performance of BLS, we show a comparison with the following approaches from the literature: a very recent heuristic EXSCOL [13]; a hybrid local search (HLS) [4]; a greedy algorithm (MRLF) based on the popular RLF graph coloring heuristic [10]; a parallel genetic algorithm (PGA) [9]; a tabu search algorithm (TS) [3]; and a recent local search combining ideas of variable neighborhood search and iterated local search (MDSLS) [6]. EXSCOL [13] is the current best-performing approach in the literature and is particularly effective on large graphs. It is based on an iterative extraction of large independent sets, where each independent set defines a color class. Moreover, EXSCOL is highly

effective in terms of computing time. On the same computing platform as that we used to test BLS, it requires from 1 minute up to 30 minutes for large graphs (e.g., DIMACS1000.X) to reach the reported results. The time limit used by MDSLS [6] is 1 hour on a computer similar to ours. For other reference approaches [3, 4, 9, 10], details on testing conditions are not available.

Tables 2 and 3 report respectively the comparative results with these reference algorithms on the tested DIMACS and COLOR02 instances. From the results of Table 2, we observe that our BLS algorithm outperforms, for each of the tested DIMACS instance, the reference algorithms MRLF [10], TS [3], and MDSLS [6] (except on DSJC125.1 where BLS and MDSLS report the same result). However, compared to the highly effective EXSCOL algorithm [13], BLS shows a worse performance on large DIMACS instances, but is able to report a better result than EXSCOL for 4 DIMACS instances up to 250 vertices. Notice that two reference algorithms HLS and PGA do not report results for these instances.

Table 2. Comparative results between our BLS algorithm and four reference approaches on the set of DIMACS instances.

Name	Sum*	BLS	EXSCOL [13]	MRLF [10]	TS [3]	MDSLS [6]
DSJC125.1	326	**326**	**326**	352	344	**326**
DSJC125.5	1015	**1012**	1017	1141	1103	1015
DSJC125.9	2511	**2503**	2512	2653	2631	2511
DSJC250.1	977	**973**	985	1068	1046	977
DSJC250.5	3246	**3219**	3246	3658	3779	3281
DSJC250.9	8286	8290	**8286**	8942	9198	8412
DSJC500.1	2850	2882	**2850**	3229	3205	2951
DSJC500.5	10910	11187	**10910**	12717	–	11717
DSJC500.9	29912	30097	**29912**	32703	–	30872
DSJC1000.1	9003	9520	**9003**	10276	–	10123
DSJC1000.5	37598	40661	**37598**	45408	–	43614

Table 3. Comparative results between our BLS algorithm and five reference approaches on the set of COLOR02 instances.

Name	Sum*	BLS	EXSCOL [13]	HLS [4]	MRLF [10]	PGA [9]	MDSLS [6]
anna	276	**276**	283	–	277	281	**276**
david	237	**237**	**237**	–	241	243	**237**
huck	243	**243**	**243**	**243**	244	**243**	**243**
jean	217	**217**	**217**	–	**217**	218	**217**
queen5.5	75	**75**	**75**	–	**75**	**75**	**75**
queen6.6	138	**138**	150	**138**	**138**	**138**	**138**
queen7.7	196	**196**	**196**	–	**196**	**196**	**196**
queen8.8	291	**291**	**291**	–	303	302	**291**
games120	443	**443**	**443**	446	446	460	**443**
miles250	325	327	328	343	334	347	**325**
miles500	709	710	**709**	755	715	762	712
myciel3	21	**21**	**21**	**21**	**21**	**21**	**21**
myciel4	45	**45**	**45**	**45**	**45**	**45**	**45**
myciel5	93	**93**	**93**	**93**	**93**	**93**	**93**
myciel6	189	**189**	**189**	**189**	**189**	**189**	**189**
myciel7	381	**381**	**381**	**381**	**381**	382	**381**

For the COLOR02 instances, we observe in Table 3 that BLS shows a comparable performance to the recent MDSLS algorithm [6] and a slightly better performance than EXSCOL [13]. Moreover, the best result obtained with BLS is in each case either better or as good as that reported by HLS [4], MRLF [10] and PGA [9]. For TS, no results are reported on these instances in [3].

From the given comparison on both sets of benchmarks, we can conclude that our BLS algorithm for the MSCP is very competitive with the state of art approaches from the literature.

3 Discussions

One observes that among the different ingredients of the BLS algorithm (see Alg. 1), only the neighborhood relations, evaluation functions and perturbation moves (see sections 2.2 and 2.3) are specific to the MSCP. In fact, BLS is a generic search framework which inherits and combines features from iterated local search [11], tabu search [5] and simulated annealing [8]. We briefly discuss the similarities and differences between our BLS approach and these methods.

Following the general framework of ILS, BLS uses local search to discover local optima and perturbation to diversify the search. However, BLS distinguishes itself from most ILS algorithms by the combination of multiple perturbation strategies of different intensities, triggered according to the search status. In particular, a distinction of BLS is in the way a perturbation type is selected. As explained in Section 2.3, BLS applies a perturbation of weaker diversification with a higher probability P as the search progresses toward improved new local optima. This probability is progressively decreased as the number of consecutively visited non-improving local optima ω increases. The idea of an adaptive change of probability P is inspired by the popular acceptance criterion used in simulated annealing, which ensures that neighboring solutions (even those of bad quality) are accepted with higher probability when the temperature is high, and with lower probability as the temperature decreases.

In order to direct the search toward more promising regions of the search space, BLS employs directed perturbation strategies based on the notion of tabu list from tabu search. However, unlike tabu search, BLS does not consider the tabu list during its local search phases. As such, BLS and tabu search may explore quite different trajectories during their respective search, leading to different local optima. In fact, we believe that diversification during the descent-based improving phase is unnecessary. The compromise between search exploration and exploitation is critical only once a local optimum is reached. Other studies supporting this idea can be found in [1, 7].

To validate the generality of BLS, in addition to the MSCP presented in this paper, we have applied BLS to solve several other classical combinatorial optimization problems (quadratic assignment, maximum clique [2], and maximum cut) and observed very competitive performances on benchmark instances.

4 Conclusion

In this paper, we have presented the Break Local Search algorithm for solving the minimum sum coloring problem. The computational evaluation of the proposed algorithm on two sets of 27 DIMACS and COLOR02 benchmark instances has revealed that BLS is able to improve the best-known results for 4 DIMACS instances and attain the best-known results for 15 instances while failing to reach the best ones for 8 instances. These results are competitive compared with most of the state of art approaches for the MSCP.

Acknowledgment

We are grateful to the referees for their comments and questions which helped us to improve the paper. The work is partially supported by the Pays de la Loire Region (France) within the RaDaPop (2009-2013) and LigeRO (2010-2013) projects.

References

1. Battiti R., Protasi M.: Reactive search, a history-based heuristic for max-sat. ACM Journal of Experimental Algorithmics, 2 (1996)
2. Benlic U., Hao J.K.: Breakout local search for maximum clique problems. Accepted to Computers & Operations Research, DOI:10.1016/j.cor.2012.06.002 (2012)
3. Bouziri H., Jouini M.: A tabu search approach for the sum coloring problem. Electronic Notes in Discrete Mathematics, 36:915–922 (2010)
4. Douiri S.M., and Elbernoussi S.: New algorithm for the sum coloring problem. International Journal of Contemporary Mathematical Sciences, 6:453–463 (2011)
5. Glover F., Laguna M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
6. Helmar A., Chiarandini M.: A local search heuristic for chromatic sum. In MIC 2011, pp. 161–170 (2011)
7. Kelly J.P., Laguna M., Glover F.: A study of diversification strategies for the quadratic assignment problem. Computers and Operations Research, 21(8):885 – 893 (1994)
8. Kirkpatrick S., Gelett C.D., Vecchi M.P.: Optimization by simulated annealing. Science, 220:621–630 (1983)
9. Kokosiński Z., Kwarciany K.: On sum coloring of graphs with parallel genetic algorithms. Proceedings of the ICANNGA'07 (Part 1), LNCS 4431: 211–219 (2007)
10. Li Y., Lucet C., Moukrim A., Sghiouer K.: Greedy algorithms for minimum sum coloring algorithm. Proceedings of LT2009 (2009)
11. Lourenco H.R., Martin O., Stützle T.: Iterated local search, Handbook of Metaheuristics, Springer-Verlag, Berlin Heidelberg (2003)
12. Malafiejski M.: Sum coloring of graphs. In M. Kubale (Ed.), Graph colorings, AMS, pp 55–65 (2004)
13. Wu Q., Hao J.K.: An effective heuristic algorithm for sum coloring of graphs. Computers & Operations Research, 39(7):1593–1600 (2012)