# Using CP and ILP with tree decomposition to solve the sum colouring problem

Maël Minot, Samba Ndojh Ndiaye, Christine Solnon

# Using CP and ILP with tree decomposition to solve the sum colouring problem

M. Minot[1,3] (student), S. N. Ndiaye[1,2] (advisor), and C. Solnon[1,3] (advisor)

[1] Université de Lyon - LIRIS
[2] Université Lyon 1, LIRIS, UMR5205, F-69622 France
[3] INSA-Lyon, LIRIS, UMR5205, F-69621, France
{mael.minot,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

**Abstract.** The Minimum Sum Colouring Problem is an $\mathcal{NP}$-hard problem derived from the well-known graph colouring problem. It consists in finding a proper colouring which minimizes the *sum* of the assigned colours rather than the *number* of those colours. This problem often arises in scheduling and resource allocation. Mainly incomplete approaches were proposed, but Integer Linear Programming (ILP) and Constraint Programming (CP) have also been used. In this paper, we conduct a more in-depth evaluation of ILP and CP's capabilities to solve the sum colouring problem, with several improvements. Moreover, we propose to combine ILP and CP in a tree decomposition with a bounded height.

## 1 Introduction

The Minimum Sum Colouring Problem (MSCP) is an $\mathcal{NP}$-hard problem derived from the well-known graph colouring problem. The MSCP consists in finding a proper colouring which minimizes the *sum* of the assigned colours rather than the *number* of those colours. This problem arises in a variety of real-world problems, especially in scheduling and resource allocation. [14]

The MSCP has not been extensively studied yet. Mainly incomplete approaches were proposed. The few complete approaches include Integer Linear Programming (ILP) in [9] and Constraint Programming (CP) in [15]. However, the rather straightforward CP model only provided the authors with disappointing results.

In this paper, we propose to conduct a more in-depth evaluation of ILP and CP's capabilities to solve the sum colouring problem, with several improvements. Moreover, we use tree decomposition to improve the solution process by decomposing MSCPs into independent subproblems, as well as to combine ILP and CP, with promising results.

In Section 2, we recall necessary definitions. Section 3 lists existing approaches while Section 4 describes several improvements. Section 5 explains the basics of our decomposition, and Section 6 presents experimental results.

## 2 Definition of the sum colouring problem

**Definition 1.** *An undirected graph $G = (V, E)$ is defined by a set of nodes $V$ and a set $E \subseteq V \times V$ of edges. Each edge of $G$ is an undirected pair of nodes. We denote with* $\deg(v)$ *the degree of a vertex $v$,* i.e. $\deg(v) = |\{u \in V, \{u, v\} \in E\}|$, *and $\Delta(G)$ the largest degree found in the graph,* i.e. $\Delta(G) = \max\{\deg(v), v \in V\}$.

**Definition 2.** *A* clique *is a subset of nodes which are all pairwise linked. It is* maximal *if it is not strictly included in any other clique.*

**Definition 3.** *A legal or proper $k$-colouring of $G$ is a mapping $c : V \to \{1, \ldots, k\}$ such that $\forall \{x, y\} \in E, c(x) \neq c(y)$.*

The classic Graph Colouring Problem aims at finding a proper $k$-colouring that minimizes $k$, whereas the MSCP aims at finding a proper $k$-colouring minimizing the sum of the assigned colours ($\sum_{x \in V} c(x)$). The lowest achievable sum for a graph $G$ is called the *chromatic sum* of $G$ and is denoted $\Sigma(G)$.

In [18], theoretical bounds for the chromatic sum of a graph were proposed: $\lceil \sqrt{8|E|} \rceil \leq \Sigma(G) \leq \lfloor \frac{3(|E|+1)}{2} \rfloor$. In [14], it was demonstrated that an optimal sum colouring will never use strictly more than $\Delta(G) + 1$ colours. An often weaker bound can also prove useful in a few situations and states that $\Sigma(G) \leq |V| + |E|$.

The *domination* of some solutions by others was discussed in [15]: a colouring can be seen as an ordered partition on the vertices of the coloured graph, the $k$th set $S_k$ corresponding to the vertices using the colour $k$. A solution is dominated when the sum of the colours of all vertices can be lowered simply by swapping the indices (in other terms, by exchanging the respective colours) of these sets without actually changing the partition. Note that a proper colouring will remain legal when doing such exchanges. The optimal sum colouring for a given partition is obtained by colouring the vertices of the largest set with colour 1, those of the second largest with colour 2, and so on.

## 3 Existing approaches for the MSCP

**Incomplete approaches** Numerous incomplete approaches were used to find approximate solutions for the sum colouring problem. A review of most of these approaches can be found in [9]. It classifies main contributions for the MSCP in three classes: greedy algorithms [20,21], local search heuristics [2,5] and evolutionary algorithms [10,16,8]. It is important to notice that none of these algorithms is able to reach all best known upper and lower bounds. Yet, the percentage of best known upper bounds reached on tested graphs ranges from 46 % ([20,21]) to 90 % ([8]). These approaches actually prove the optimality of a solution whenever the lowest upper bound reaches the highest lower bound. However, such proofs were only made on about one third of tested instances, even when using simultaneously all bounds found by every methods in [9].

**CP** A basic CP model for the sum colouring problem was proposed in [15]. It associates a variable with each node, each with a domain equal to $\{1, \ldots, \Delta(G)+1\}$. There is a disequality constraint for each edge of the graph, and the objective is to minimize the sum of all variables. This model was evaluated using the solver Choco [11]. The results were rather disappointing, as solution times on rather easy instances were long.

**ILP** In [19], an ILP model was proposed for the MSCP. It associates a binary variable $x_{uk}$ with every pair $(u, k) \in V \times [1, \Delta(G) + 1]$, where $u$ is a vertex and $k$ a colour. The objective is to minimize $f(x) = \sum_{u=1}^{|V|} \sum_{k=1}^{\Delta(G)+1} k \cdot x_{uk}$ under the following constraints:

- $c_1$: $\sum_{k=1}^{\Delta(G)+1} x_{uk} = 1, \forall u \in \{1, \ldots, |V|\}$
- $c_2$: $x_{uk} + x_{vk} \leq 1, \forall (u, v) \in E, \forall k \in \{1, \ldots, \Delta(G) + 1\}$

## 4 Improving the CP and ILP models for the MSCP

We show that we may replace some binary disequality constraints by global *AllDifferent* constraints, then discuss the definition of initial domains. Finally, a filtering method for the CP model is depicted, along with a description of a colour swapping technique.

**Global *AllDifferent* constraints** CP and ILP models introduced previously use binary disequality constraints to prevent the neighbour vertices from being assigned the same colour. Slightly more elaborate models may be obtained by finding sets of binary disequality constraints corresponding to cliques and replacing them with global *AllDifferent* constraints. To find cliques, we built them in a simple greedy fashion, choosing incrementally vertices having a high degree.

Our representation of *AllDifferent* constraints for ILP can be seen as a generalisation of the constraints used in the model of [19] (see Section 3): for each clique $C$ and each colour $k$, we state that $\sum_{v \in C} x_{vk} \leq 1$.

**Initial domain reduction** In CP and ILP models, the colours that may be assigned to vertices are bounded by $\Delta(G) + 1$. We propose to lower this bound using the following property:

*Property 1.* For every optimal sum colouring $c$ of a graph $G = (V, E)$, we have $\forall v \in V, \mathrm{c}(v) \leq \deg(v) + 1$.

To prove this property, let us suppose that it is not true for a given optimal colouring $c$ of a graph $G$. It follows that there exists a vertex $v$ in $V$ such that $\mathrm{c}(v) > \deg(v) + 1$. In such a case, there exists an unused value $x$ in

$\{1, \ldots, \deg(v) + 1\}$, since $v$ only has $\deg(v)$ neighbours. As a consequence, a better colouring than $c$ can be obtained by using the colour $x$ for $v$ instead of $c(x)$. Therefore, $c$ is not optimal, which contradicts our initial claim.

**Filtering** We adapted the filtering algorithm used with branch and bound in [15] to the CP model. This filtering method builds a partition of cliques on the set of uncoloured vertices and derives a lower bound from it. In [15], such a partition is built at each node of the search tree, to obtain more accurate bounds. We noticed that this had an extensive computational cost and decided to compute a clique partition only once, at the root of the search tree. This partition is then used at each node of the tree to compute a new bound.

The lower bound obtained from our partition is the sum of bounds computed for each clique of the partition. For a clique $C$ of size $k$, the bound is the sum of the $k$ lowest values in the union of the domains of all variables of $C$. Any coloured vertex counts as a variable with only one value in its domain.

Using this strategy when only a few vertices are coloured usually yields loose bounds. To prevent unnecessary computations, we set a lower limit for the triggering of this filtering algorithm: we compute the sum of the values of currently assigned variables; if the distance between this sum and our current upper bound amounts to more than *gap* % of the current upper bound, we refrain from using this filter. In addition to this lower triggering limit, we added an upper one: we refrain from using this filter when *unc* or less vertices are uncoloured.

**Colour swapping** As pointed out in [15], the solutions built by solvers may be dominated: they can lead to an improved bound simply by rearranging the colours, following the rules used in [15] and recalled in Section 2. This method makes the upper bound go down faster. We will be referring to it as *colour swapping*. This has an exceptionally low computing cost, and the verifications are very easy to implement. Moreover, it breaks some symmetries by forbidding the recomputation of redundant solutions.

## 5 Combining solvers in a bounded-height tree decomposition

Early experiments with CPLEX showed us that it fails to solve many instances because of its high memory needs (see Section 6). We therefore propose to use a decomposition method: subproblems will be smaller and CPLEX will be more likely to be able to solve them.

We designed a method inspired from Backtracking bounded by Tree Decomposition (BTD) [6]. It splits the variables in several clusters, forming a tree decomposition. The main peculiarity of our tree is that it has a limited height of 1: every cluster is either the root or a leaf. As explained in [4], one of the main problems that arise when using BTD in optimization problem is that the solutions of non-leaf clusters must be enumerated, and that this might happen several times

for a same cluster. Using a decomposition of height 1, however, only the root is subject to this. The resulting decomposition will be called *Flower Decomposition* henceforward, and the resolution method using it will be accordingly referred to as Backtracking bounded by Flower Decomposition (BFD).

A flower decomposition may be obtained from a traditional tree decomposition. We firstly build the set of intersections between clusters. It has to be noted that these intersections are separators in the constraint graph. We then heuristically choose a subset of intersections, and regroup them into one cluster, which will become the root of the new decomposition. This subset is chosen by a heuristic, aiming to create a root with few nodes while observing a constraint on the maximal size of any leaf. If no subset of separators satisfies the constraints on the size of the leaves or of the separators themselves, we build a root cluster by extracting vertices of high degree, thus obtaining a single leaf cluster. After this step, we compute the connected components resulting from the removal of our new root from the constraint graph. These components form the leaf clusters. Each leaf is then extended by adding to it any variable linked by a constraint to a variable of this leaf. A similar process was employed in [7], where it was also demonstrated that it results in a correct tree decomposition.

The first tree decomposition is built using the *MinFill* algorithm, which is known to yield good decompositions [12]. By using this decomposition as the base for our new decomposition, we obtain separators that are properly spread out across the constraint graph. This allows us to obtain balanced leaf-clusters.

For each solution of the root cluster, BFD independently solves to optimality the subproblems induced by the leaf clusters. These optimal values are saved as valued goods on the intersections with the root cluster, to avoid useless recomputations. It has to be noticed that nothing prevents us from using different solving methods for the clusters of the decomposition. Choices of solvers are crucial here, because the task that must be accomplished in the root cluster (enumeration) is very different from solving the subproblems in the leaves (optimisation). Using CP, it is generally extremely easy to enumerate solutions, asking for one at a time, without any need to keep the whole of them in memory. On the other hand, ILP allows a fast optimal resolution of problems, as long as their size is reasonable. For these reasons, we decided to implement BFD using CP to enumerate the solutions of the root and ILP to solve the subproblems induced by the leaves for each assignment of the root. the CP model is the same as in Section 4, but restrained to the variables of the root cluster.

## 6 Experimental evaluation

### 6.1 Experimental setup and benchmark

Programs are executed on an Intel® Xeon® CPU E5-2670 0 at 2.60 GHz processor, with $20,480$ KB of cache memory and 4 GB of RAM. The domain reduction described in Section 4 was used for each model and each solver.

We consider 126 instances which are classicaly used for sum colouring, as in [19,9]. Some are from COLOR02/03/04[4], but most of them are DIMACS instances designed for the classical colouring problem[5]. Some of these instances have lower and upper bounds presented in [19,9]. We do not use them to solve the instances, but merely to compute relative distances to assess the quality of the solutions. We ran 24 hours (86,400 seconds) experiments.

### 6.2 Implementation

**Gec Base** We implemented the CP model in Gecode (version 4.2.1) [17]. The encoding of the model is straightforward. The Branch and Bound (BAB) search engine was selected, and *AllDifferent* constraints were represented by Gecode's "`distinct`" constraint. The consistency level was GAC, named "`ICL_DOM`" by Gecode. As the goal is to minimize the sum of the variables, the value ordering heuristic always choose the smallest available value. With regards to the variable choice heuristic, *minElim* was used. *minElim* chooses the variable that has the smallest value in its domain among all domains, and break ties by choosing the variable for which the chosen value would be removed from the fewest domains.

**Gec Improved** An improved, more parametrized version of Gecode was also used. A geometrical restart policy was used, with a scale of 100 and a base of 2. This policy offers quite an advantage to perform proofs while also allowing to quickly find good solutions. Colour swapping was activated, as well as the filter presented in Section 4. This version manages heuristics as such: for each instance *minElim* was used first, but was changed to *dynDeg* until the end of the resolution if the search endured ten restarts without having improved the global upper bound. *dynDeg* chooses the variable that has the highest number of uncoloured neighbours; ties are broken first by choosing the variable that has the smallest value in its domain, then by choosing the variable with the smallest current domain. The reason for this combination is that *minElim* is good at finding solutions and *dynDeg* at proving optimality. For the filtering algorithm, we set *gap* to 20 and *unc* to 5. These choices are based on tests that are not detailed here.

**CPLEX** We used ILOG CPLEX (version 12.6.2) to solve the ILP model [3]. To help CPLEX to avoid running out of memory, the two following parameters were added: Depth-First Search was forced as a node selection strategy, and the cuts factor was set to 1.5. Previous experiments, not shown here, proved us that these parameters did not significantly lessen CPLEX's ability to solve the instances we used.

[4] `http://mat.gsia.cmu.edu/COLOR02`
[5] `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/`

**Table 1.** A summary of results obtained using CPLEX, BFD 90 and 75, Gec Base, Gec Improved, Selector and the VBS. The first three lines give, respectively, the minimal, average, and maximal distance between the best solution found by the method and the best known solution (in percentages). "Proofs" (resp. "Mem. out") corresponds to the number of instances for which optimality has been proven (resp. a memory out occurred). "Found best" gives the count of instances for which the best known solution has been found, and "Times best" the number of times the method has been the best on an instance. The best method is determined primarily by whether the instance was solved, then by quality of solution, and then by the time needed to find the best solution.

| | | CPLEX | BFD 90 | BFD 75 | Gec Base | Gec Improved | Selector | VBS |
|---|---|---|---|---|---|---|---|---|
| Dist | Min | **−0.5** | 0 | 0 | 0 | 0 | **−0.5** | −0.5 |
| | Avg | 63.8 | 83.3 | 85 | 9.2 | 7.4 | **6.6** | 5.2 |
| | Max | 1,119.5 | 1,119.5 | 1,119.5 | 78.5 | **75.1** | **75.1** | 75.1 |
| Proofs | | 65 | 39 | 26 | 10 | 16 | **66** | 66 |
| Mem. out | | 23 | 18 | 16 | **0** | **0** | **0** | 0 |
| Found best | | 73 | 47 | 28 | 30 | 44 | **77** | 83 |
| Times best | | 67 | 10 | 9 | 3 | 40 | **91** | 126 |

**BFD $l$** For methods using BFD, a few new parameters are also to be considered, mostly to control the shape of the flower decomposition. The maximal size for the separators was set to 30. Actually, the effect of this limit is not that we merge clusters but that we do not record any valued good on any separator that exceeds this limit. Since Gecode is used for the root cluster in BFD, we did not use restarts, as it is not as useful when enumerating solutions as it is when solving a problem to optimality. The CPLEX part, to solve subproblems, was parametrized as we did for the method using CPLEX alone. The heuristic used to enumerate the solutions of the root was *minElim*, with no possible subsequent change. Finally, we introduced a parameter $l$ to set a limit on the size of the leaf clusters: BFD $l$ denotes a BFD where no leaf can contain more than $l$ % of the variables of the instance. We consider two values for $l$: 75 and 90.

**VBS and Selector** CPLEX, BFD 90, BFD 75 and Gec Improved have complementary advantages. This may be shown by considering a Virtual Best Solver (VBS) that selects the best approach for each instance. Hence, we propose to investigate the interest of portfolio approaches [13]. As this is just a preliminary paper, we will only use per-instance algorithm selection with a static rule based on the number of edges $|E_G|$. This simple approach, denoted Selector, proceeds as such: if $|E_G| \leq$ 440,000, use CPLEX; else if $|E_G| \leq$ 700,000, use BFD 90; else if $|E_G| \leq$ 1,200,000, use BFD 75; else use Gec Improved. These choices are based on the fact that the number of edges is heavily correlated with CPLEX's memory requirements.

### 6.3 Results

Table 1 summarizes the results. The issue that stands out the most is that CPLEX often runs out of memory. For instances for which there is no memory out, CPLEX appears to be very efficient, as it is able to solve the largest number of instances (65) and finds the best solutions for 73 instances. Only its memory requirements prevent it from yielding good results on the most difficult or large instances, sometimes even making it impossible to find any solution at all. This explains the fact that the solutions found by CPLEX are 64 % higher than best known solutions on average for all instances.

Gec Base is able to prove optimality on only ten instances, but it has reasonable memory needs, and thus never runs out of memory. This enables it to obtain a low average distance to the best known bounds: 9.2 %.

Gec Improved obtains 6 more proofs than Gec Base, with an average distance of 7.4. Filtering appears to be generally disadvantageous to find a good solution, but helps Gecode a lot when it comes to proving optimality. Restarts induce more diversity in the search, which translates to better bounds. The bounds deduced via colour swapping prevent us from exploring some dominated solutions and thus break symmetries.

BFD offers an interesting compromise between CPLEX's efficiency and Gecode's scalability. The leaf-size limit adds a way to slightly control the degree of the decomposition. Even though the global results of BFD are unsatisfying, both BFD 90 and BFD 75 are better than every other method on about ten instances. This highlights a certain complementarity.

We also present the results of the Selector approach and of the VBS. It appears quite clearly that using a very simple criterion such as the number of edges already yields satisfying results: every proof from CPLEX is preserved, and one proof is added. The average distance goes below what Gecode alone could achieve in its improved version, and no memory out occurred using this combination. Overall, the results of Selector are very close to those of the VBS.

## 7 Conclusion and future work

In this paper, we considered the minimum sum colouring problem, and demonstrated that even though CP seems uncompetitive at first sight, it holds a few advantages, like significantly lowered memory requirements compared to ILP solvers. This allows a solver such as Gecode to tackle instances that involve very large graphs. These abilities can be pushed further still with several improvements, such as carefully set restarts, a moderate filtering algorithm, and colour swapping on found solutions. We proposed a CP / ILP combination giving good results. The promising results obtained from the combination of methods presented here might direct our research towards more elaborated portfolio approaches. Furthermore, using conjunctions of constraints involving *AllDifferent* could lead to better bounds and filtering. [1]

**Acknowledgements** This work has been supported by the ANR project SoLStiCe (ANR-13-BS02-0002-01).

## References


1. Beldiceanu, N., Carlsson, M., Petit, T., Régin, J.C.: An o (nlog n) bound consistency algorithm for the conjunction of an alldifferent and an inequality between a sum of variables and a constant, and its generalization. In: ECAI. vol. 12, pp. 145–150 (2012)
2. Benlic, U., Hao, J.K.: A study of breakout local search for the minimum sum coloring problem. In: Simulated Evolution and Learning, pp. 128–137. Springer (2012)
3. CPLEX, I.: High-performance software for mathematical programming and optimization (2005)
4. De Givry, S., Schiex, T., Verfaillie, G.: Exploiting tree decomposition and soft local consistency in weighted csp. In: AAAI. vol. 6, pp. 1–6 (2006)
5. Helmar, A., Chiarandini, M.: A local search heuristic for chromatic sum. In: Proceedings of the 9th metaheuristics international conference. vol. 1101, pp. 161–170 (2011)
6. Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artificial Intelligence 146, 43–75 (2003)
7. Jégou, P., Kanso, H., Terrioux, C.: An algorithmic framework for decomposing constraint networks. In: Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on. pp. 1–8. IEEE (2015)
8. Jin, Y., Hao, J.K.: Hybrid evolutionary search for the minimum sum coloring problem of graphs, accepted to information sciences feb 2016 (2015)
9. Jin, Y., Hamiez, J.P., Hao, J.K.: Algorithms for the minimum sum coloring problem: a review. arXiv preprint arXiv:1505.00449 (2015)
10. Jin, Y., Hao, J.K., Hamiez, J.P.: A memetic algorithm for the minimum sum coloring problem. Computers & Operations Research 43, 318–327 (2014)
11. Jussien, N., Rochart, G., Lorca, X.: Choco: an open source java constraint programming library. In: CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08). pp. 1–10 (2008)
12. Kjaerulff, U.: Triangulation of graphs - algorithms giving small total state space. Tech. rep., Judex R.R. Aalborg., Denmark (1990)
13. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. arXiv preprint arXiv:1210.7959 (2012)
14. Kubale, M.: Graph colorings, vol. 352. American Mathematical Soc. (2004)
15. Lecat, C., Li, C.M., Lucet, C., Li, Y.: Comparaison de méthodes de résolution pour le probleme de somme coloration. In: JFPC'15: Journées Francophones de Programmation par Contraintes (2015)
16. Moukrim, A., Sghiouer, K., Lucet, C., Li, Y.: Upper and lower bounds for the minimum sum coloring problem, submitted for publication
17. Team, G.: Gecode: Generic constraint development environment, 2006 (2008)
18. Thomassen, C., Erdös, P., Alavi, Y., Malde, P.J., Schwenk, A.J.: Tight bounds on the chromatic sum of a connected graph. Journal of Graph Theory 13(3), 353–357 (1989)
19. Wang, Y., Hao, J.K., Glover, F., Lü, Z.: Solving the minimum sum coloring problem via binary quadratic programming. arXiv preprint arXiv:1304.5876 (2013)
20. Wu, Q., Hao, J.K.: An effective heuristic algorithm for sum coloring of graphs. Computers & Operations Research 39(7), 1593–1600 (2012)
21. Wu, Q., Hao, J.K.: Improved lower bounds for sum coloring via clique decomposition. arXiv preprint arXiv:1303.6761 (2013)