UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE COMPUTACIÓN

A combinatorial branch & bound algorithm for the minimum sum coloring problem on P_4-sparse graphs

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Francisco Julián Laborda

Director: Flavia Bonomo

Codirector: Javier Marenco

Buenos Aires, 2011

UN ALGORITMO BRANCH & BOUND COMBINATORIO PARA EL PROBLEMA DE COLOREO DE SUMA MÍNIMA EN GRAFOS P_4-SPARSE

El problema de Coloreo de Suma Mínima consiste en asignar números naturales a los vértices de un grafo de modo tal que pares de vértices adyacentes obtengan diferentes números, y la suma de los números asignados sea mínima. Un grafo es P_4-sparse si todo conjunto de 5 vértices contiene a lo sumo un P_4 inducido. Además, los grafos P_4-sparse tienen asociado un árbol de descomposición modular con raíz compuesto por operaciones básicas en los nodos y ciertos grafos primitivos en las hojas.

Actualmente, no se conoce un algoritmo polinomial que resuelva el problema de coloreo de suma mínima en grafos P_4-sparse, pero sí existe un algoritmo 2-aproximado, que retorna una solución cuya suma es a lo sumo el doble de la óptima.

En este trabajo, introducimos una familia de cotas inferiores para el problema y presentamos un algoritmo del estilo Branch & Bound combinatorio, el cual encuentra la solución óptima rápidamente para la mayoría de los casos.

Palabras claves: coloreo, coloreo de suma minima, grafos P_4-sparse, algoritmo aproximado.

A COMBINATORIAL BRANCH & BOUND ALGORITHM FOR THE MINIMUM SUM COLORING PROBLEM ON P_4-SPARSE GRAPHS

In the Minimum Sum Coloring problem (MSC), we aim to assign natural numbers to vertices of a graph so that adjacent vertices get different numbers, and the sum of the numbers assigned to the vertices is minimum. A graph is P_4-sparse if every set of five vertices contains at most one induced P_4. Moreover, P_4-sparse graphs have an associated modular decomposition tree consisting of basic operations as internal nodes, and certain primitive graphs as leaves.

Currently, there is no polynomial time algorithm to solve the minimum sum coloring problem on P_4-sparse graphs, but there is a polynomial time 2-approximation algorithm, that yields solutions with lower or equal sum than twice the optimal.

In this work, we introduce a family of lower bounds for the problem and present a combinatorial branch and bound implementation, which finds the optimal solution very quickly for most of the cases.

Keywords: graph coloring, minimum sum coloring, P_4-sparse graphs, approximation algorithm.

AGRADECIMIENTOS

Agradezco a mis directores Javier y Flavia por toda su atención y trabajo. A mi viejo por haber hecho posible todo esto con gigantezca paciencia. A todos los que colaboraron con este trabajo (en orden cuasi cronológico): Arti, Manix, el Pocho, Facu, Ger, Diego DD, Mai, Mati y Colby. A la Universidad de Buenos Aires y a la Universidad Nacional de General Sarmiento. Y por útimo, agradezco a todos y a cada uno de mis amigos, y a mi familia por bancarme.

A mi viejo.

CONTENTS

1. Introduction 1
   1.1 The Minimum Sum Coloring Problem 1
   1.2 Maximal sequences and optimal solutions of the MSC problem 2
   1.3 Maximal Sequences of P_4-sparse graphs 3
   1.4 A 2-approximation algorithm for the MSC problem on P_4-sparse graphs 4
   1.5 About this work 5

2. Lower bounds 7
   2.1 Our goal 7
   2.2 Alternatives to the 2-approximation algorithm 7
   2.3 Interesting bounds 8
   2.4 Looking for a constant approximation factor 9

3. A branch & bound algorithm 11
   3.1 Local search approach 11
   3.2 Overview 12
   3.3 Branching 12
   3.4 Finding feasible solutions and bounds 12
   3.5 Structures used 13
   3.6 Technical details 13

4. Results 15
   4.1 Instance generation 15
   4.2 Execution time 15

5. Conclusions and future work 21
   5.1 Conclusions 21
   5.2 Future work 21

Appendices 23

A. Proofs 23

1. INTRODUCTION

1.1 The Minimum Sum Coloring Problem

A *vertex coloring* of a graph $G = (V, E)$ is an assignment of colors to the vertices in V such that adjacent vertices receive different colors. We assume that the colors are positive integers. A vertex k-coloring of a graph G is a coloring such that the color of each vertex in V is taken from the set $\{1, 2, \ldots, k\}$. Given a vertex coloring of a graph G, the *sum* of the coloring is the sum of the colors assigned to the vertices. The *chromatic sum* $\Sigma(G)$ of G is the smallest sum that can be achieved by any proper coloring of G. In the *Minimum Sum Coloring* (MSC) problem we look for a coloring of G with sum $\Sigma(G)$. The minimum number of colors needed in a minimum sum coloring of G is called the *strength* of G and is denoted by $s(G)$. Clearly, for any graph G we have $s(G) \geq \chi(G)$, where $\chi(G)$ denotes the chromatic number of G.

The MSC problem was introduced by Kubicka [11]. The problem is motivated by applications in scheduling [1, 2, 6] and VLSI design [13, 15]. The computational complexity of determining the vertex chromatic sum of a simple graph has been extensively studied since then. In [12] it is shown that the problem is NP-hard in general, but solvable in polynomial time for trees. The dynamic programming algorithm for trees can be extended to partial k-trees and block graphs [10]. Furthermore, the MSC problem is NP-hard even when restricted to some classes of graphs for which finding the chromatic number is easy, such as bipartite or interval graphs [2, 15]. A number of approximability results for various classes of graphs were obtained in the last ten years [1, 5, 6, 4].

If G_1 and G_2 are two vertex disjoint graphs, then their *union* $G_1 \cup G_2$ is the graph with vertex set $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and edge set $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$. Similarly, their *join* $G_1 \vee G_2$ is the graph with $V(G_1 \vee G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \vee G_2) = E(G_1) \cup E(G_2) \cup \{(x, y) : x \in V(G_1),\ y \in V(G_2)\}$.

A *spider* is a graph whose vertex set can be partitioned into S, C and R, where $S = \{s_1, \ldots, s_k\}$ ($k \geq 2$) is an independent set; $C = \{c_1, \ldots, c_k\}$ is a complete set; s_i is adjacent to c_j if and only if $i = j$ (a *thin spider*), or s_i is adjacent to c_j if and only if $i \neq j$ (a *thick spider*); R is allowed to be empty and if it is not, then all the vertices in R are adjacent to all the vertices in C and non-adjacent to all the vertices in S. Clearly, the complement of a thin spider is a thick spider, and vice-versa. The triple (S, C, R) is called the *spider partition*, and can be found in linear time [8]. The sets S, C and R are called the *legs*, *body* and *head* of the spider, respectively. The *size* of the spider will be $|C|$. P_4-sparse graphs have a nice decomposition theorem as follows.

Theorem 1. [7, 9] *If G is a non-trivial P_4-sparse graph, then either G or $\overline{G}$ is not connected, or G is a spider.*

To each P_4-sparse graph G one can associate a corresponding decomposition rooted tree T in the following way. Each non-leaf node in the tree is labeled with either "$\cup$" (union-nodes), or "$\vee$" (join-nodes) or "SP" (spider-partition-nodes), and each leaf is labeled with a vertex of G. Each non-leaf node has two or more children. Let T_x be the subtree of T rooted at node x and let V_x be the set of vertices corresponding to the leaves in T_x. Then, each node x of the tree corresponds to the graph $G_x = (V_x, E_x)$. A union-node (join-node) corresponds to the disjoint union (join) of the P_4-sparse graphs associated with its children. A spider-partition-node corresponds to the spider with spider-partition (S, C, R) where S, C, and R are its children. Finally, the P_4-sparse graph associated with the root of the tree is just G, the P_4-sparse graph represented by this decomposition tree. The decomposition tree associated to a P_4-sparse graph can be computed in linear time [9].

1.2 Maximal sequences and optimal solutions of the MSC problem

The results in this section are an extract of previous work by F. Bonomo and M. Valencia-Pabon on the MSC Problem on P_4-sparse graphs [3].

A k-coloring of a graph $G = (V, E)$ can be regarded as a partition of the vertex set V into k independent sets $S_1, \dots, S_k$, where each vertex in S_i is colored with color i, for $1 \le i \le k$. So, for any such k-partition of V into independent sets, we can associate a non-negative *sequence* p such that $p[i] = |S_i|$ for $i = 1, \dots, k$ and $p[i] = 0$ for $i > k$. In the sequel, we deal with finite-support non-negative integer sequences only. Let $|p| = \max\{i : p[i] > 0\}$.

Definition 1. *Let p and q be two integer sequences. We say that p* dominates *q, denoted by $p \succeq q$, if for all $t \ge 1$ it holds that $\sum_{1 \le i \le t} p[i] \ge \sum_{1 \le i \le t} q[i]$.*

Definition 2. *Let p be a sequence. We denote by $\widetilde{p}$ the sequence that results from p when we order it in a non-decreasing way.*

The following two lemmas are direct consequences of Definition 1.

Lemma 1. *The dominance relation $\succeq$ is a partial order.*

Lemma 2. *Let p be a sequence. Then, $\widetilde{p} \succeq p$.*

The following lemma will be very useful in order to study the MSC problem on graphs.

Lemma 3. *Let p and q be two sequences and let $n = \max\{|p|, |q|\}$. If $p \succeq q$ and $\sum_{1 \le i \le n} p[i] = \sum_{1 \le i \le n} q[i]$, then it holds that $\sum_{1 \le i \le n} i \cdot p[i] \le \sum_{1 \le i \le n} i \cdot q[i]$.*

Proof. Let $N = \sum_{1 \le i \le n} p[i] = \sum_{1 \le i \le n} q[i]$. Let P and Q be two sequences obtained from p and q such that $|P| = |Q| = N$, and defined by $P[j] = \min\{k : \sum_{1 \le i \le k} p[i] \ge j\}$ (resp. $Q[j] = \min\{k : \sum_{1 \le i \le k} q[i] \ge j\}$) for $j = 1, \dots, N$. By hypothesis, $p \succeq q$, and so, $P[j] \le Q[j]$ for all $1 \le j \le N$. Therefore, $\sum_{1 \le i \le n} i \cdot p[i] = \sum_{1 \le j \le N} P[j] \le \sum_{1 \le j \le N} Q[j] = \sum_{1 \le i \le n} i \cdot q[i]$. □

Notice that if the sequences represent partitions of the vertex set of a graph into independent sets, where the value of the i-th element of the sequence represents the size of the i-th independent set in the partition, then for the sum-coloring problem on graphs we can restrict us to study maximal sequences w.r.t. the partial order $\succeq$. Notice also that maximal sequences are non-increasing sequences. We will call *maximal partition* to a partition of the vertex set of a graph into independent sets associated to a maximal sequence. In the following, we define some operations between sequences.

Definition 3. *Let p and q be two sequences. The join of p and q, denoted by $p \star q$, is the sequence that results by sorting the concatenation of the sequences p and q in non-increasing order.*

Definition 4. *Let p and q be two sequences. The sum of p and q, denoted by $p + q$, is the sequence whose i-th value is equal to $p[i] + q[i]$, for $i \ge 1$. Notice that $|p + q| = \max\{|p|, |q|\}$.*

Definition 5. *Let p and q be two sequences. We say that p and q are non-comparable, denoted by $p||q$, if $p \not\succeq q$ and $q \not\succeq p$.*

The following two lemmas will be useful in order to study the MSC problem on P_4-sparse graphs.

Lemma 4. *Let p, p' and q be sequences. If $\widetilde{p} \succeq \widetilde{p'}$ then $p \star q \succeq p' \star q$.*

Proof. Consider the sequence $p' \star q$. By definition of join, $\widetilde{p'}$ is a subsequence of $p' \star q$. Let s be the sequence that results from $p' \star q$ by replacing each element $\widetilde{p'}[i]$ by $\widetilde{p}[i]$. As by hypothesis, $\widetilde{p} \succeq \widetilde{p'}$, then we have that $s \succeq p' \star q$. But now, note that $p \star q = \widetilde{s}$ and thus, $p \star q \succeq s \succeq p' \star q$. $\square$

Lemma 5. *Let p, p', and q be sequences. Then, $p||p'$ if and only if $p+q||p'+q$.*

Proof. Note that $p||p'$ if and only if there exist two different positive integers j_1 and j_2 such that $\sum_{i=1}^{j_1} p[i] > \sum_{i=1}^{j_1} p'[i]$ and $\sum_{i=1}^{j_2} p[i] < \sum_{i=1}^{j_1} p'[i]$. This happens if and only if $\sum_{i=1}^{j_1}(p[i] + q[i]) > \sum_{i=1}^{j_1}(p'[i] + q[i])$ and $\sum_{i=1}^{j_2}(p[i] + q[i]) < \sum_{i=1}^{j_1}(p'[i] + q[i])$, which is equivalent to $p+q||p'+q$. $\square$

The following result can be proved similarly.

Lemma 6. *Let p, p', and q be sequences. Then, $p \succeq p'$ if and only if $p+q \succeq p'+q$.*

1.3 Maximal Sequences of P_4-sparse graphs

We include in this section results regarding maximal sequences and P_4-sparse graphs. All of these results were taken from previous work in [3] and they are not part of this work. All the proofs in this section have been omitted for the sake of clarity and are included in Appendix A.

In the sequel, sequences of a graph will represent partitions of its vertex set into independent sets. The following two lemmas show that if we are looking for maximal sequences of a graph that is either the union or the join of two vertex disjoint graphs G_1, G_2, then it is sufficient to consider maximal sequences of the graphs G_1 and G_2.

Lemma 7. *Let G_1, G_2 be two vertex disjoint graphs, and let $G = G_1 \cup G_2$. Then, every maximal sequence p of G can be expressed as $p = p_1 + p_2$, where p_1 (resp. p_2) is a maximal sequence of G_1 (resp. G_2).*

Lemma 8. *Let G_1, G_2 be two vertex disjoint graphs, and let $G = G_1 \vee G_2$. Then, every maximal sequence p of G can be expressed as $p = p_1 \star p_2$, where p_1 (resp. p_2) is a maximal sequence of G_1 (resp. G_2).*

A similar result holds in general for homogeneous sets. Let G be a graph. A set $H \subseteq V(G)$ of vertices is called *homogeneous* if, for each vertex $w \in V(G) \setminus H$, either w is adjacent to all the vertices in H or to none of them. For any subset of vertices $X \subseteq V(G)$, denote by $G[X]$ the subgraph of G induced by X.

Lemma 9. *Let G be a graph and H an homogeneous set of G. Let $S_1, \ldots, S_k$ be a maximal partition of G, and let $S_{i_1}, \ldots, S_{i_t}$ be the sets in the partition having nonempty intersection with H. Then, $S_{i_1} \cap H, \ldots, S_{i_t} \cap H$ is a maximal partition of $G[H]$.*

We describe next the maximal sequences of spiders.

Lemma 10. *Let $G = (S, C, R)$ be a spider such that $R \neq \emptyset$, and let p be a maximal sequence for G. Then there exists a partition $S_1, \ldots, S_{|p|}$ associated with p in which there are only three kinds of sets: sets entirely contained in R, sets entirely contained in C, and sets intersecting both R and S; moreover, sets entirely contained in C are the last $|C|$ sets, and only S_1 intersects both R and S, with $S \subseteq S_1$.*

Lemma 11. *Let $G = (S, C, R)$ be a spider such that $R \neq \emptyset$. Then, the number of maximal sequences of G is equal to the number of maximal sequences of $G[R]$. Moreover, for each maximal sequence q of $G[R]$ there exists only one maximal sequence q' of G with $|q'| = |q| + |C|$ and where $q'[1] = q[1] + |C|$, $q'[i] = q[i]$ for $2 \leq i \leq |q|$ (if $|q| \geq 2$), and $q'[i] = 1$ for $|q| + 1 \leq i \leq |q| + |C|$.*

Lemma 12. *Let $G = (S, C, R)$ be a thin spider such that $R = \emptyset$. Then, G has only one maximal sequence p, with $|p| = |C|$, where $p[1] = |C|$, $p[2] = 2$, and $p[i] = 1$ for $3 \leq i \leq |C|$.*

Lemma 13. *Let $G = (S, C, R)$ be a thick spider such that $|C| \geq 3$ and $R = \emptyset$. Then, G has only two maximal sequences p_1 and p_2, with $|p_1| = |C|$ and $|p_2| = |C| + 1$, where $p_1[i] = 2$ for $1 \leq i \leq |C|$, and $p_2[1] = |C|$ and $p_2[i] = 1$ for $2 \leq i \leq |C| + 1$.*

Notice also that the trivial graph has only one maximal sequence p, with $|p| = 1$, where $p[1] = 1$. Therefore, we have the following theorems.

Theorem 2. *Let G be a P_4-sparse graph such that in its modular decomposition there are no thick spiders (S, C, R) with $|C| \geq 3$ and $R = \emptyset$. Then,*

1. *$s(G) = \chi(G)$, G has a unique maximal sequence, and $\Sigma(G)$ and an optimal coloring of G can be computed from its modular decomposition in polynomial time.*
2. *In such an optimal coloring, each set S_i is a maximum independent set of $G \setminus \bigcup_{1 \leq j < i} S_j$ which verifies $\chi(G \setminus \bigcup_{1 \leq j \leq i} S_j) = \chi(G \setminus \bigcup_{1 \leq j < i} S_j) - 1$.*

Theorem 3. *Let G be a P_4-sparse graph on n vertices. Let t be the number of thick spiders (S, C, R) with $|C| \geq 3$ and $R = \emptyset$ in the modular decomposition of G. Then, $s(G) \leq \chi(G) + t$, the number of maximal sequences of G is at most 2^t, and an optimal coloring of G can be computed in $2^t P(n)$ time, where $P(n)$ is a polynomial in n.*

The algorithm described in Theorem 3 allows us to find also the minimum sum that can be attained by a coloring of G with at most r colors, for some given value $r \geq \chi(G)$, and the corresponding coloring.

1.4 A 2-approximation algorithm for the MSC problem on P_4-sparse graphs

The algorithm presented here is based on the study of maximal sequences of P_4-sparse graphs and was proposed by F. Bonomo and M. Valencia-Pabon in [3].

Let G be a P_4-sparse graph on n vertices. Let $\sigma^1, \sigma^2, \ldots, \sigma^t$ be the thick spiders in the decomposition tree of G, such that $\sigma^j = (S^j, C^j, \emptyset)$ and $|C^j| \geq 3$, for $j = 1, \ldots, t$. We assume that t is of order $\Omega(n)$, otherwise, by Theorem 3 (see Appendix A), an optimal solution for the MSC problem on G can be computed in polynomial time.

Consider the following algorithm to color G: first, for each thick spider σ^j, we choose as its maximal sequence p^j the sequence p_1 of Lemma 13 (see Appendix A), that is, $p^j[i] = 2$ for $1 \leq i \leq |C^j|$, and its corresponding maximal partition $S^j_1, \ldots, S^j_{|C^j|}$, where $S^j_i = \{s^j_i, c^j_i\}$, being

s_i^j and c_i^j non-adjacent vertices in S^j and C^j, respectively, for $i = 1, \dots, |C^j|$. Next, we apply the algorithm in the proof of item (1) of Theorem 2 (see Appendix A) in order to compute in $O(n^2)$ time a partition into independent sets for G. Let ϕ be the coloring of the vertices of G obtained by the previous algorithm.

Algorithm 1 shows a pseudocode of this algorithm. Given a decomposition tree T of graph G, it finds the associated sequence to coloring ϕ.

Algorithm 1 $\phi(T) : DecompositionTree \to Sequence < Integer >$:
($*$ stands for sequence concatenation, power operation stands for sequence repetition.)

```
1: $\phi(join(L,R)) \equiv \phi(L) \star \phi(R)$ {Lemma 7}
2: $\phi(union(L,R)) \equiv \phi(L) + \phi(R)$ {Lemma 8}
3: $\phi(thinSpider(S,C,R)) \equiv R \neq \emptyset$ ? $[|C| + first(\phi(R))] * tail(\phi(R)) * [1]^{|C|}$ {Lemma 11}
                                        : $[|C|, 2] * [1]^{|C|-2}$ {Lemma 12}
4: $\phi(thickSpider(S,C,R)) \equiv R \neq \emptyset$ ? $[|C| + first(\phi(R))] * tail(\phi(R)) * [1]^{|C|}$ {Lemma 11}
                                        : $[2]^{|C|}$ {Lemma 13}
5: $\phi(individualVertex) \equiv [1]$
```

Clearly, ϕ uses $\chi(G)$ colors. Let $\Sigma_\phi(G)$ be the sum of the colors of the vertices of G induced by the coloring ϕ. We claim the following.

Lemma 14. $\Sigma_\phi(G) \leq 2\Sigma(G)$.

Proof. Let Φ be an optimal coloring for G with sum $\Sigma(G)$. Let H be the induced subgraph of G obtained by removing from G all the vertices of each independent set S^j of the thick spider σ^j, for $1 \leq j \leq t$. Let $\widetilde{\Sigma}(H)$ be the sum of the colors of the vertices in H under the coloring Φ. On one hand, we have that $\widetilde{\Sigma}(H) \geq \Sigma(H)$. Moreover,

$$\Sigma(G) \geq \widetilde{\Sigma}(H) \geq \Sigma(H) \qquad (*)$$

On the other hand, let Φ' be an optimal coloring of H with sum $\Sigma(H)$. We extend the coloring Φ' of H to a coloring Φ'' of G, where each vertex $s_i^j \in S^j$ is assigned the color $\Phi'(c_i^j)$ of vertex $c_i^j \in C^j$, for $1 \leq i \leq |C^j|$, being $(S^j, C^j, \emptyset)$ the thick spider σ^j, for $1 \leq j \leq t$. Let $\Sigma_{\Phi''}(G)$ be the sum induced by the coloring Φ'' on G. Clearly, $\Sigma_\phi(G) \leq \Sigma_{\Phi''}(G)$, and $\Sigma_{\Phi''}(G) \leq 2\Sigma(H)$. Therefore, by $(*)$, we have that $\Sigma_\phi(G) \leq 2\Sigma(G)$. □

Therefore, we have the following result.

Theorem 4. *Algorithm 1 is a 2-approximation algorithm for the MSC problem on P_4-sparse graphs.*

1.5 About this work

With the 2-approximation algorithm from the previous section as the starting point, this work was developed as an attempt to find a better way to solve the MSC problem for the entire family of P_4-sparse graphs.

We began our work with the idea of implementing a combinatorial branch and bound algorithm for the MSC problem on P_4-sparse graphs by resorting to the 2-approximation algorithm as the bounding heuristic, instead of linear programming techniques.

We also believed that the factor 2 of the algorithm was not tight. In the first approach to this idea, we could not find an instance for which any of the candidate solutions (maximal sequences) would be greater than twice the optimal. That gave us the idea that a branch and bound algorithm with this kind of loose bounding would not be able to discard any branches. That is why we turned all our focus to the search of new lower bounds for the problem.

As we found new lower bounds and considered an alternative to Algorithm 1 for the upper bounds, we were ready to implement a combinatorial branch & bound algorithm for the the family of P_4-sparse graphs. So we did it and we also analyzed its performance.

The rest of this work is organized as follows: in Chapter 2 we show how we can use lower bounds to find a better k-approximation algorithm, we present a way of generalizing the 2-approximation algorithm (Algorithm 1) which introduces a family of lower bounds to the problem and we briefly discuss each of them. In Chapter 3 we present an exact combinatorial branch & bound algorithm for the MSC problem on P_4-sparse graphs, we explain its idea and implementation, and discuss its performance in Chapter 4. Finally, in Chapter 5 we comment our conclusions and ideas for future work, and we propose a conjecture regarding the approximation factor of a new algorithm for the MSC problem on P_4-sparse graphs.

2. LOWER BOUNDS

The main obstacle in the way to prove an algorithm is actually a k-approximation algorithm for a certain value of k is that we need to compare the approximate solution with the optimal solution, which is usually unknown. Typically, the way to get around this obstacle is finding lower bounds. In this chapter, we present a way of generalizing the 2-approximation algorithm shown in Section 1.4 and we introduce a family of lower bounds for the MSC problem on P_4-sparse graphs.

This chapter is organized as follows: in Section 2.1 we present a proof scheme for an approximation algorithm for the MSC problem on P_4-sparse graphs by using lower bounds, in Section 2.2 we present a possible generalization to the 2-approximation algorithm shown in 1.4 together with some particular instances of it and we discuss them, the actual lower bounds we found are shown in Section 2.3 and candidate upper bounds to those lower bounds are presented in Section 2.4.

2.1 Our goal

Let G be a P_4-sparse graph. Let Φ be an optimal coloring for G with sum $\Sigma(G)$. Let ϕ be the coloring obtained from an approximation algorithm with sum $\Sigma_\phi(G)$ and let $LB(G)$ be a lower bound to the problem, that is

$$LB(G) \leq \Sigma(G)$$

multiplying by $\frac{\Sigma_\phi(G)}{LB(G)}$ on both sides we get

$$\frac{\Sigma_\phi(G)}{LB(G)} LB(G) \leq \frac{\Sigma_\phi(G)}{LB(G)} \Sigma(G)$$

$$\Sigma_\phi(G) \leq \frac{\Sigma_\phi(G)}{LB(G)} \Sigma(G)$$

If we could prove that $\frac{\Sigma_\phi(G)}{LB(G)} \leq k$, then we would have proved that the approximation algorithm that calculated coloring ϕ is a k-approximation algorithm.

2.2 Alternatives to the 2-approximation algorithm

The 2-approximation algorithm in Section 1.4 chooses one out of two possible maximal sequences for each thick empty headed spider. It is possible to generalize this algorithm by making it parametric to the sequence used to color each thick empty headed spider. Following this idea, we can then consider each particular case of this possible generalization as an individual algorithm.

In the sequel, we will refer to these algorithms (and the coloring they calculate) as ϕ_{seq}, where $_{seq}$ represents the sequence they use for that kind of spider when their size is 3 (for bigger ones, they will follow the $_{seq}$ pattern.) For instance, using this notation, the 2-approximation algorithm from the previous section would be noted ϕ_{222}, and the $_{222}$ subindex comes from the sequence it chooses for every thick empty headed spider of size 3, which is a sequence of 2's of length equal to the size of the spider. This can bee seen in Algorithm 1 in the last part of Line 4.

For example, let $G = (S, C, R)$ be a thick spider such that $|C| \geq 3$ and $R = \emptyset$: algorithm ϕ_{3111} would yield the sequence p with $|p| = |C| + 1$ and $p_2[1] = |C|$ and $p_2[i] = 1$ for $2 \leq i \leq |C| + 1$. In particular, if $|C| = 3$, it yields the sequence $[3, 1, 1, 1]$.

Notice that, regarding all the possible algorithm alternatives,

- not all of them yield a proper coloring. (They could assign the color 0 to a vertex or assign the same color to adjacent vertices),
- even if they might be invalid coloring partitions, each ϕ_{seq} yields the minimum sum partition among those that use the $_{seq}$ pattern to color thick empty headed spiders,
- all of them run in polynomial time, as ϕ_{222} does (as long as $_{seq}$ is in non decreasing order), and they are calculated exactly as ϕ_{222} is, as shown in Algorithm 1, except for line 4, where every empty headed thick spider must be assigned the sequence that corresponds to the $_{seq}$ pattern.
- in fact, ϕ_{111} is the coloring used in the proof of Lemma 14 as Φ' (the optimal coloring of H with sum $\Sigma(H)$)[1],
- alternatively to ϕ_{222}, ϕ_{3111} is the one that chooses the alternate maximal sequence for every thick empty headed spider. Thus, ϕ_{3111} *does* achieve a proper coloring for a given P_4-sparse graph, and we will discuss it later as an heuristic and as a candidate k-approximation algorithm.

2.3 Interesting bounds

In Section 2.2, we presented the ϕ_{seq} family of algorithms, each of which yields a coloring (or pseudo-coloring, if invalid). We will now present some interesting lower bounds. These bounds (in particular, the best one we found) make possible the implementation of the exact branch & bound algorithm shown in the following chapter.

ϕ_{321} is the algorithm alternative that colors each thick spider (S, C, R) where $R = \emptyset$ with the sequence p, with $|p| = |C|$, where $p[1] = |C|$, $p[2] = 2$, and $p[i] = 1$ for $3 \leq i \leq |C|$. Let p_{222} and p_{3111} be the maximal sequences for thick empty headed spiders, we can see that p dominates both of them, therefore, its sum has to be a lower bound. In fact, p is equivalent to the coloring of a thin empty headed spider, which we already know has a unique maximal sequence, equal to the one just shown.

ϕ_{411} is the algorithm alternative that colors each thick spider (S, C, R) where $R = \emptyset$ with the sequence p, with $|p| = |C|$, where $p[1] = |C + 1|$ and $p[i] = 1$ for $2 \leq i \leq |C|$. We can see that p dominates both maximal sequences for this kind of spider, just like ϕ_{321} does, thus, ϕ_{411}'s sum has to be a lower bound.

ϕ_{311}'s sum is also a lower bound, since ϕ_{311} is the same coloring as ϕ_{321}, except for one vertex in every thick empty headed spider, for which its color is changed to 0. Therefore, $\Sigma_{\phi_{311}}(G) \leq \Sigma_{\phi_{321}}(G)$ for every P_4-sparse graph G and we can ensure ϕ_{311}'s sum is a lower bound as well.

ϕ_{312} is the coloring that assigns the second highest color twice for adjacent vertices in every empty headed thick spider. It is a special case, for it is not in non decreasing order. Because of this reason, we cannot work with it as we do with maximal sequences and we cannot ensure its sum is a lower bound. However, we believe it is worth mentioning because we could not find a counterexample for which it has a greater sum than the optimal solution.

[1] If ϕ_{211} was used instead, we can ensure $\Sigma_{\phi}(G) < 2\Sigma(G)$

2.4 Looking for a constant approximation factor

As we said in Section 2.1, we would like to prove $\frac{\Sigma_\phi(G)}{LB(G)} \leq k$ for every P_4-sparse graph G. A way of proving this would be to find what graph (or family of graphs) achieve the maximum value of $\frac{\Sigma_\phi(G)}{LB(G)}$ for some approximation algorithm ϕ and some lower bound LB.

For $\phi = \phi_{222}$, every lower bound found falls into the following situation. Let G be a thick empty headed spider of size n. ϕ_{222} would assign the sequence p to G with $|p| = n$ and $p_i = 2$ for $1 \leq i \leq n$. Suppose we pick ϕ_{311} as the lower bound (same thing happens with the rest of the lower bounds), then its sequence will be q with $|q| = n$, $q_1 = n$ and $q_i = 1$ for $2 \leq i \leq n$. Then,

$$\frac{\sum_{\phi_{222}}(G)}{\sum_{\phi_{311}}(G)} = \frac{2\sum_{i=1}^{n} i}{n - 1 + \sum_{i=1}^{n} i} = \frac{2n(n+1)}{n-1+n(n+1)} = \frac{2n^2+2n}{n^2+3n-2} \to 2$$

So we can see that ϕ_{222} will not yield an approximation factor lower than 2 if we compare it to the bounds we have.

For $\phi = \phi_{3111}$, the problem is that none of the lower bounds we found seem to be easily comparable. So we decided to take an experimental approach to understand better what was going on with the other lower bounds compared to ϕ_{3111}. This approach was an attempt to have an estimate of an approximation factor (better than 2) that could be used for the lower bounds of a branch & bound implementation, which we present in the next chapter.

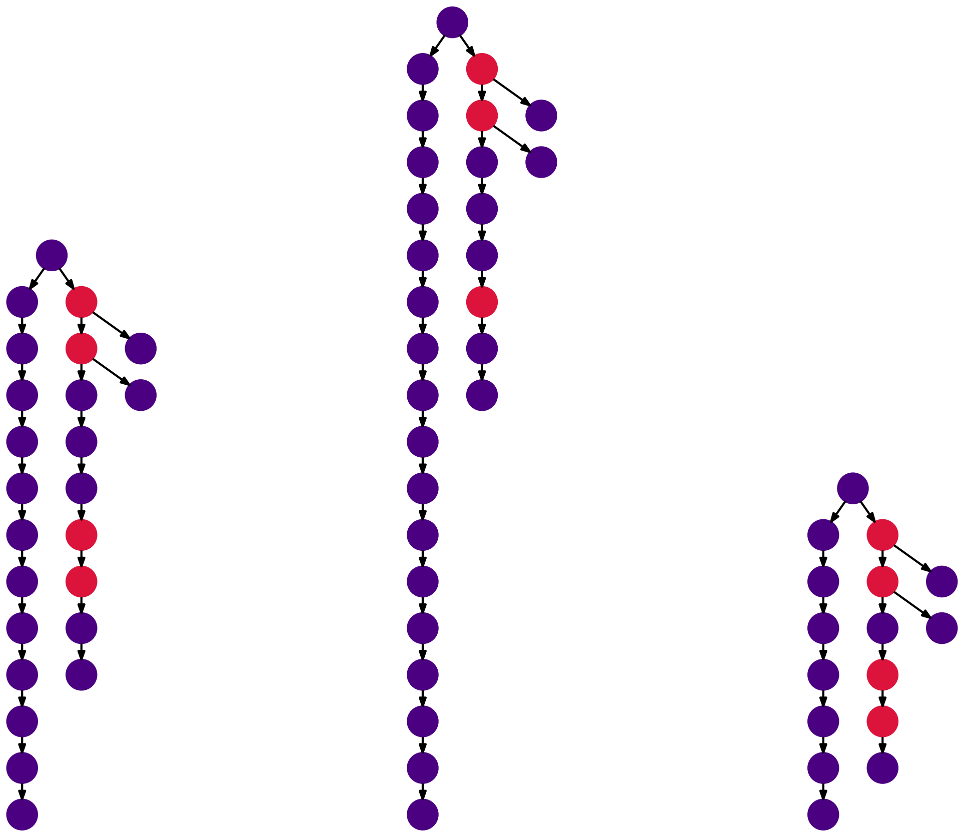
We calculated the value of the quotient $\frac{\Sigma_{\phi_{3111}}(G)}{LB(G)}$ together with several lower bounds for an exhaustive set of P_4-sparse graphs. Based on a binary tree generator (which generates all the possible binary trees of size n) we generated instances with internal nodes as join or union operations and leaves as individual vertices and thick empty headed spiders. We tried all possible combinations up to trees of size 10 and spiders of size 5. For larger sizes of trees and spiders, for which the execution would take too long, we interrupted it, but we still kept on searching among larger decomposition trees by setting the instance generation parameters manually. Highest values seem to be found around trees with all its leaves as thick empty headed spiders of size 3 and mostly join operations.

Figure 2.1 shows the highest value found for the quotient for each lower bound. Blue circles represent *join* nodes and red ones represent *union* nodes. All leaves (not shown in the figure) are thick empty headed spiders.

On the bright side, the lower bounds seem to be upper bounded and ϕ_{3111} seems to always yield a good quality solution. As we explore larger graphs, the value of the quotient seems to converge, and the values we are dealing with are lower than 2, which is the current best approximation factor.

On the other side, even if there seems to be a very clear graph pattern, the worst-case graphs found did not lead us to prove what kind of graph (or graph pattern) achieves the actual absolute maximum for the quotient. Moreover, if we keep on searching, we might find (slightly) higher values for larger decomposition trees that might not follow the observed pattern.

Finally, the best of the lower bounds found seems to be strong enough to be used in a branch & bound implementation, which we present in the following chapter.



(a) ϕ_{311}:1.65097 *(b)* ϕ_{411}:1.50876 *(c)* ϕ_{321}:1.32383

Fig. 2.1: Decomposition trees achieving the max value of $\frac{\Sigma_{\phi_{3111}}(G)}{LB(G)}$ attained in our experiments for different lower bounds.

3. A BRANCH & BOUND ALGORITHM

In this chapter we present an exact branch & bound algorithm that finds the optimal solution for the MSC problem on P_4-sparse graphs.

A branch & bound algorithm consists of a systematic enumeration of all candidate solutions, where large subsets of non optimal solutions are discarded all together by using upper and lower bounds of the quantity being optimized.

The key idea of any branch & bound algorithm is: if the lower bound for some tree node of the search tree (set of candidates) A is greater than the upper bound for some other node B, then A may be safely discarded from the search. This step is called pruning, and is usually implemented by maintaining a global variable m (shared among all nodes of the tree) that records the minimum upper bound seen among all subregions examined so far. Any node whose lower bound is greater than m can be discarded.

Our algorithm was developed from scratch, using the scheme of a regular branch & bound implementation, in which all the possible solutions are considered or discarded using some bounding criterion and relatively good solutions are found by the use of heuristics. Usually, bounds and feasible solutions are found using linear programming techniques, however, this algorithm does not use any of those techniques, but is based upon the bounds and approximation results shown in the previous sections instead.

The chapter is organized as follows: first, in Section 3.1 we show an example for which a local search approach does not yield the optimal solution. Then we briefly describe the branch & bound algorithm in Section 3.2, we focus on the branching scheme in Section 3.3 and we discuss the heuristics used for finding solutions and bounding in Section 3.4. Section 3.5 explains the structures used and finally some technical considerations are presented in Section 3.6.

3.1 Local search approach

Before we go on any further, let us present an example where a local search approach does not work to find the optimal solution. This is just a simplistic example, out of many others, that we found when considering a local search approach to the MSC problem on P_4-sparse graphs.

Consider a local search heuristic that uses ϕ_{3111} as the initial candidate solution and a neighborhood relation defined as all the solutions only differing in the way of coloring one of the thick empty headed spiders. Let $T = (((S_1 \cup S_2) \cup S_3) \vee (S_4 \cup S_5))$ be the decomposition tree of a P_4-sparse graph, where S_i is a thick empty headed spider of size 3 for $1 \leq i \leq 5$. Let ϕ^i_{3111} be the ith neighbor coloring of ϕ_{3111} in which the spider i is colored using the maximal sequence $[2, 2, 2]$, instead of $[3, 1, 1, 1]$. So, this would be the results for the first neighborhood exploration:

- $\phi_{3111} : [9, 6, 3, 3, 3, 2, 2, 2] = 99$
- $\phi^1_{3111} : [8, 6, 4, 4, 2, 2, 2, 2] = 100$
  $\phi^2_{3111} : [8, 6, 4, 4, 2, 2, 2, 2] = 100$
  $\phi^3_{3111} : [8, 6, 4, 4, 2, 2, 2, 2] = 100$
- $\phi^4_{3111} : [9, 5, 3, 3, 3, 3, 3, 1] = 102$
  $\phi^5_{3111} : [9, 5, 3, 3, 3, 3, 3, 1] = 102$

Due to the local symmetry in T, some of the colorings are equivalent to each other, that is why they are grouped together in the list.

We can see that the local minimum for this instance is ϕ_{3111} with a sum of 99, but an optimal coloring for T is achieved by coloring every S_i with sequence $[2, 2, 2]$, achieving coloring $[6, 6, 6, 4, 4, 4]$ with sum 96.

In this way we discarded a local search approach and we decided to face the branch & bound approach.

3.2 Overview

The algorithm's input is the decomposition tree of a P_4-sparse graph G. Its output is a partition into independent sets that represents the minimum sum coloring of G.

The algorithm explores the whole decision tree, calculating, on every node, a feasible solution and a lower bound for that branch. It globally stores the best solution found at each moment, and if the lower bound for a branch happens to be greater than or equal to the best solution, it discards the whole branch.

The branching is focused on thick empty headed spiders, as they hold the most "uncertainty" regarding the minimum sum coloring, the heuristic used is ϕ_{3111} (Section 2.2) and the bounding is achieved by ϕ_{321} (Section 2.2 as well).

3.3 Branching

Since the main uncertainty we currently have towards this problem lies on determining what sequence to use on each thick empty headed spider, we decided that branching should be based on whether each of them should be colored using either one of the maximal sequences for that case. So the algorithm fixes the first of these spiders to one of the options, it calls recursively, then it fixes it to the other possible option and calls recursively again to explore the other half of the solution space.

In other words, each node of the decision tree represents one thick empty headed spider being fixed to one out of two options. Thus, every branching is binary and each complete branch (from the root to a leaf) represents each of the valid solutions to the problem.

3.4 Finding feasible solutions and bounds

We used ϕ_{3111} as a heuristic to find feasible solutions. That is, the initial "best" solution is found efficiently with ϕ_{3111}, and then, as we go fixing thick empty headed spiders to certain options, we use ϕ_{3111} again to rapidly color the unfixed ones.

We decided to use the ϕ_{3111} heuristic because it achieves good quality solutions in practice, as shown in Figure 2.1, and also because it runs in polynomial time (See Section 2.2).

Regarding the bounds, the best lower bound we found is ϕ_{321}. It is the best because it yields the greatest sum among the bounds we have considered, and thus, it would work better as it bounds more tightly. Moreover, considering the dominance relation, we can see that $\Sigma_{\phi_{311}}(G) \leq \Sigma_{\phi_{411}}(G) \leq \Sigma_{\phi_{321}}(G)$ for any P_4-sparse graph G. Of course, it can also be computed in polynomial time.

For all of the above, we opted for ϕ_{321} as the only heuristic to find lower bounds in our branch & bound implementation.

It is important to note that the lower bound used in the algorithm has been proved to be an actual bound but has not been proved to be bounded itself. That means we cannot guarantee a certain quality of the bound, and it could possibly fail to be strong enough to discard a considerable amount of branches. We will present in the next section some graph instances for which the algorithm is forced to explore the entire solution space.

3.5 Structures used

We decided to focus on binary trees only, since they are easier to work with and still able to represent decomposition trees. So, on our decomposition trees representation, the only possible leaves are empty headed thick spiders and individual vertices, and the only possible internal nodes are *join* or *union* operations.

There is no loss in generality since, regarding the coloring sequences, all other types of spiders found on P_4-sparse graphs' decomposition trees can be "simulated" by other kind of elemental graphs appropriately set up together.

This "simulation" is possible because, regarding the decomposition trees, we can ignore the graphs they actually represent, but only care about the sequences that represent the colorings for those graphs. So if the sequences we are looking for are equal, then it does not matter what precise kind of graph we are actually going to use it for. This list describes how to "simulate" all kinds of spiders (except thick ones with empty heads):

- $spider(S, C, R)$ with $R \neq \emptyset$ can be thought of as $join(C, union(S, R))$ and their sequence is $[|C| + first(\phi(R))] * tail(\phi(R)) * [1]^{|C|}$, where $\phi(R)$ is the sequence corresponding to R,
- $thinSpider(S, C, R)$ with $R = \emptyset$ can be thought of as $join(S, union(C - K_1, K_1))$, where K_1 is an individual vertex, and their sequence is $[|C|, 2] * [1]^{|C|-2}$.

3.6 Technical details

To compare the results, we also implemented a brute force algorithm, which given a decomposition tree of a P_4-sparse graph with t thick empty headed spiders, lists all 2^t possible solutions and finds the best one.

Both the brute force and the branch and bound algorithms were implemented from scratch using Python v3.2 and all the scripts were ran on PyScripter over a Aspire Timeline X machine with an Intel Core i3 processor on Windows 7 and 4 GB of system memory.

We represented decomposition trees of P_4-sparse graphs using binary trees. In the brute force implementation, we recursively listed all the 2^t maximal sequences, for a tree of t thick empty headed spiders, and then found one that achieved the minimum sum.

The calculation of the lower bounds and the feasible solutions is done recursively over the tree structure combining sequences of the subtrees. Since finding both the feasible solutions and the bounds led to a large amount of recalculation, we implemented caching of partial solutions on each node of the decomposition tree. In this way, we tried to mitigate the algorithms' overhead. Caching would also be an advantage in a context where we have to make minor changes to the input graphs, preserving most of its structure, since it is possible to dynamically alter the decomposition tree of P_4-sparse graphs to reflect these changes in linear time[14]. In this context, the implemented cache would sensitively reduce the execution time of subsequent calls.

In the branch and bound implementation, we followed a typical recursive backtracking scheme. We codified solutions and partial solutions using strings in a similar way to bitsets. Each character in the string represents how a certain thick empty headed spider would be colored, where `'1'` means the option ϕ_{222} would pick, and `'0'` means the option ϕ_{3111} would. The character `'-'` is used for spiders that have not been fixed by the algorithm on any particular sequence yet. The order of the characters matches the order of appearance of the thick empty headed spiders in the tree from left to right.

Only partial solutions contain `'-'` characters. Partial solutions are used in two ways: first, to yield feasible solutions and second, to compute the lower bounds. In the former, `'-'` characters (which represent unfixed spiders) are colored using ϕ_{3111} style of sequences, since ϕ_{3111} is the chosen heuristic for this purpose. In the latter, this type of characters are colored using the ϕ_{321} option.

For example, if we get the partial solution `'01--'`, and we want a feasible solution, we will interpret each `'-'` as a `'0'` and we will color it as `'0100'`. That means every thick empty headed spider will be colored using the ϕ_{3111} sequence, except for the second one, which is set to `'1'` and will be colored with the ϕ_{222} sequence. On the other hand, if we want a lower bound for this partial solution, the first two characters will be colored in the same way, but the `'-'` characters will be colored using ϕ_{321}.

In this way, starting with an all `'-'` solution code, we are able to explore the solution space by fixing the characters in the solution from left to right and calculating the feasible solutions and the bounds.

Every evaluation of a solution code implied a computation over the entire decomposition tree. Each internal node stores the amount of thick empty headed spiders in its subtree, in this way, when a solution code is evaluated, we break it into two codes by splitting it in half, and call recursively on each subtree with the corresponding subcode.

As many of the evaluations of subcodes were calculated more than once, we decided to include a dictionary in every internal node of the tree to work as a cache memory for previously computed solution codes. We decided to use strings to represent these codes because, as an immutable data type, they can be used as keys in Python's dictionaries.

4. RESULTS

In this chapter we discuss the performance of the branch & bound algorithm for randomly generated decomposition trees.

4.1 Instance generation

Trying to cover most of the decomposition tree instance variety, we decided to work with trees consisting of thick empty headed spiders only and trees with individual nodes as well. The spider sizes used were 3, 6, 9 and random between 3 and 9. As said in Section 3.5, we did not focus on all the other different types of spiders since they can be "simulated" with individual vertices.

We named 5 groups of instances, *3s*, *6s*, *9s* and *rs* for trees consisting of thick empty headed spiders only of size 3, 6, 9 and random between 3 and 9 respectively, and *rs-300* for trees consisting of thick empty headed spiders and 300 individual vertices.

Instances were generated by random binary tree generation, using the following procedure for trees of n individual vertices, t thick empty headed spiders, and with $1 - p$ density of *join* nodes:

- if $n + m = 1$ (base case), then place an individual node or a thick spider node.
- else, pick randomly (with probability p) an internal node between *join* and *union* and randomly split amounts n and t to build recursively left and right subtrees of smaller size.

For every generated instance, the parameter p that determines the density of *join* nodes in the tree was picked randomly. In the charts, instances have been sorted by the amount of *join* nodes to help the visualization. Apparently, instances with higher density of this kind of nodes take proportionally longer computation time. This is reasonable because more colors are needed and thus, solutions and partial solutions are represented using longer sequences.

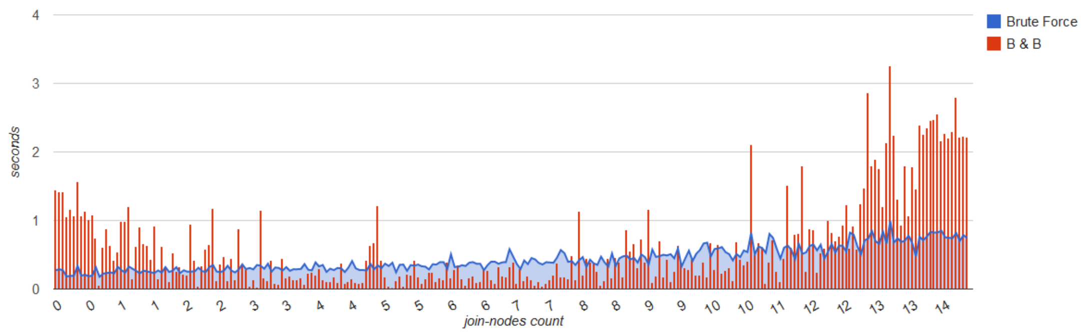
4.2 Execution time

Every chart shows execution time for 250 instances, both for the brute force (BF) and the branch & bound (B&B) implementations.

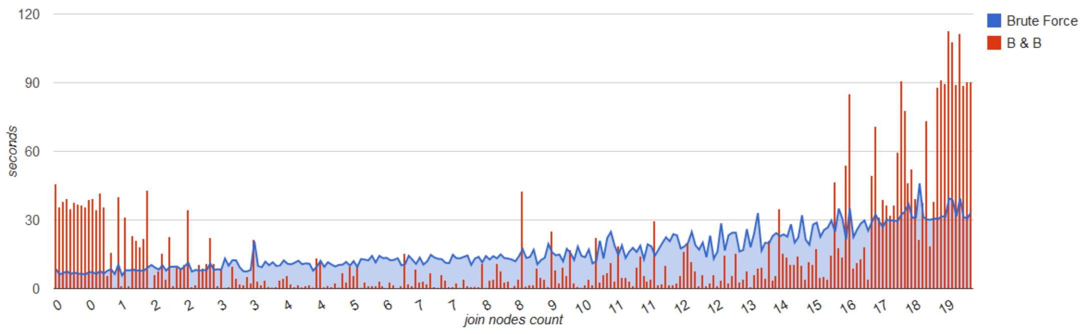
The size of the graphs represented by the decomposition tree instances is $2tk + i$, where t is the amount of thick empty headed spiders, k is their size (assuming all of them have equal size) and i is the amount of individual vertices. Therefore, the graphs represented by the decomposition trees in this chapter span between 90 and over 540 vertices.

We also tried to experiment with an integer programming model for the regular coloring problem on these kind of graphs for comparison purposes, but the graphs we were dealing with were too large to be handled.

Trees consisting of thick empty headed spiders of size 3 only seem to be the hardest case for the B&B implementation, specially for trees of few or many *join* nodes. We can see in Figure 4.1 how the B&B implementation runs slower for these cases, since it is forced to explore almost the entire decision tree without being able to discard any solutions. For the rest of the cases, the battle is more even, with B&B running faster in Figure 4.1b, having however some isolated cases where the brute force implementation wins.



(a) 15 leaves



(b) 20 leaves

Fig. 4.1: Decomposition trees consisting of thick empty headed spiders of size 3 only.

As we increment the spider size, we can see that the trend changes a bit and the B&B implementation only has trouble with instances of many *join* nodes. However, now B&B is the clear overall winner. We can see this in Figures 4.2 and 4.3.

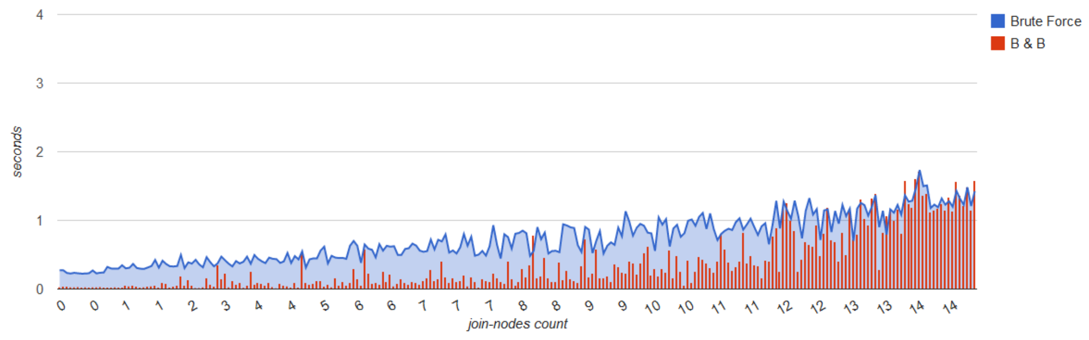
For all the cases, the performance of the B&B implementation is relatively better for larger trees. Something interesting to point out is that the B&B's performance is absolutely better for trees with larger empty headed spiders, which represent larger graphs. That means, the bigger the spiders (and the graphs), the faster the algorithm runs.

Randomness in the spider size seems to be well handled by the B&B implementation, as we can see in Figure 4.4.

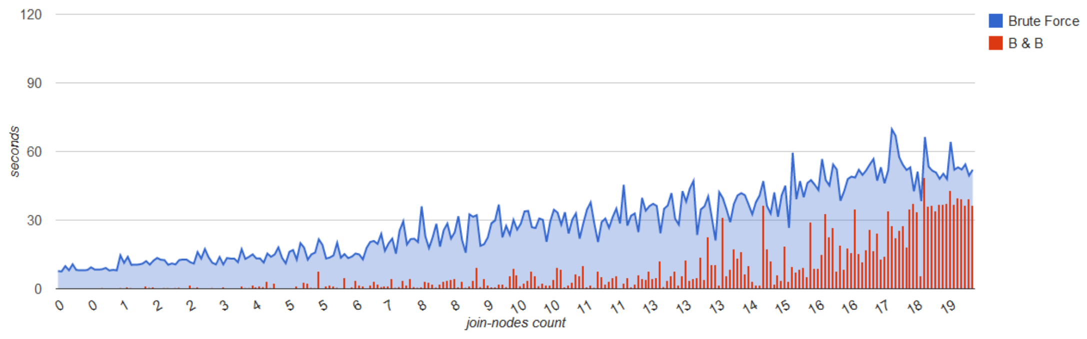
Regarding instances with individual vertices other than only thick empty headed spiders (Figure 4.5), the behavior seems to be more noisy, still showing an advantage in the B&B implementation's favor. Note that, for some of the instances, the brute force implementation could not yield a solution due to program memory shortage, while our algorithm did not have that kind of problem.

Table 4.1 is a summary of all the experiments we ran. It includes average percentage of visited nodes in the decision tree as well as average execution time grouped by instances with similar amount of *join* nodes. Columns in the left correspond to trees of 15 spiders and columns in the right, to trees of 20 spiders. The table is divided in 5 sections, according to the instance groups shown in Figures 4.1, 4.2 4.3, 4.4 and 4.5.

Looking at Table 4.1, we can see how the amount of visited nodes in the decision tree remains very low for all instances, except for the *3s* group. For this group, consisting of empty headed spiders of size 3 only, the algorithm is forced to visit more nodes, specially for instances with few or many *join* operations. This pattern is very similar to the one observed in the execution time charts, which is natural. In particular, for an instance consisting of empty headed spiders of size

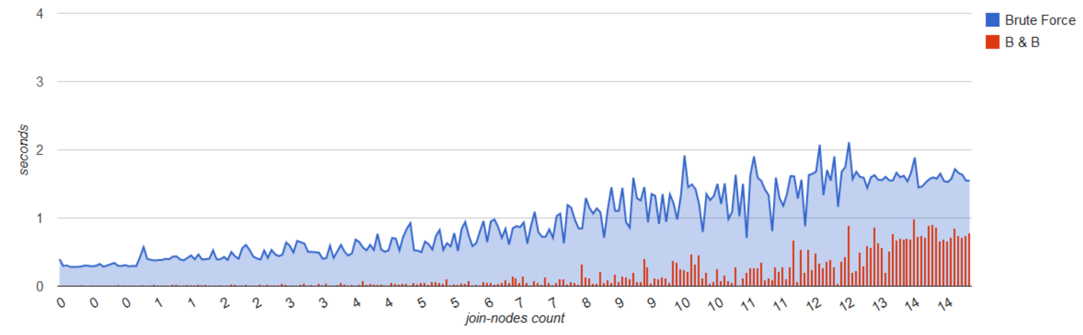


(a) 15 leaves

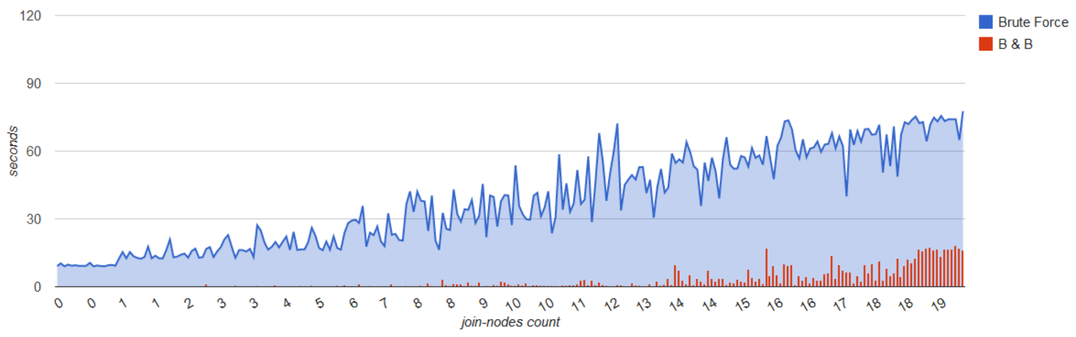


(b) 20 leaves

Fig. 4.2: Decomposition trees consisting of thick empty headed spiders of size 6 only.

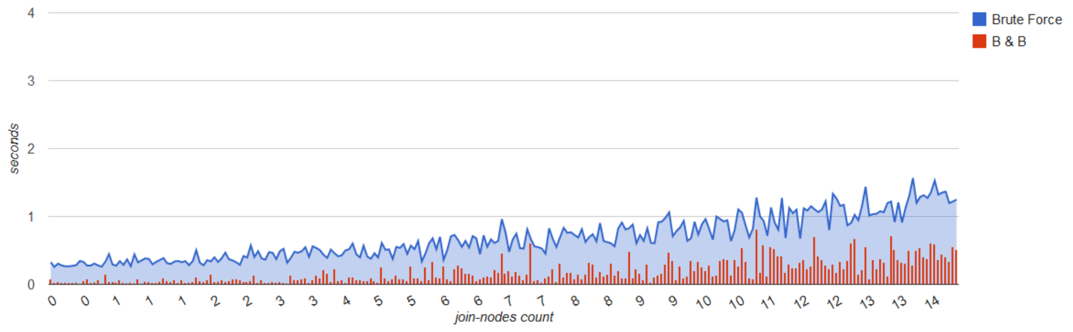


(a) 15 leaves

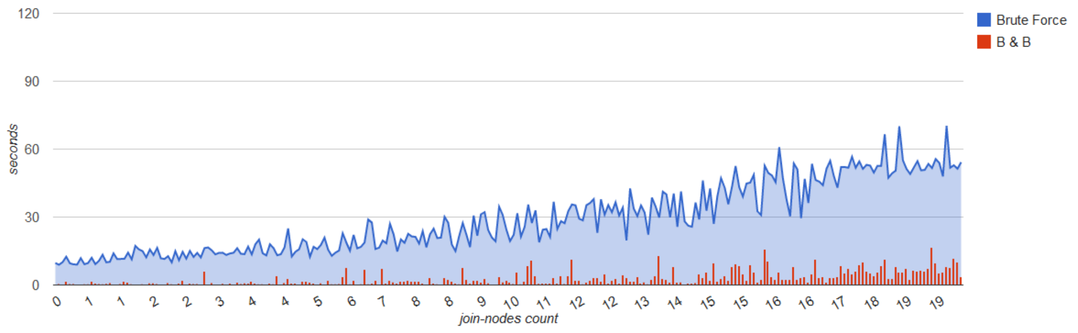


(b) 20 leaves

Fig. 4.3: Decomposition trees consisting of thick empty headed spiders of size 9 only.

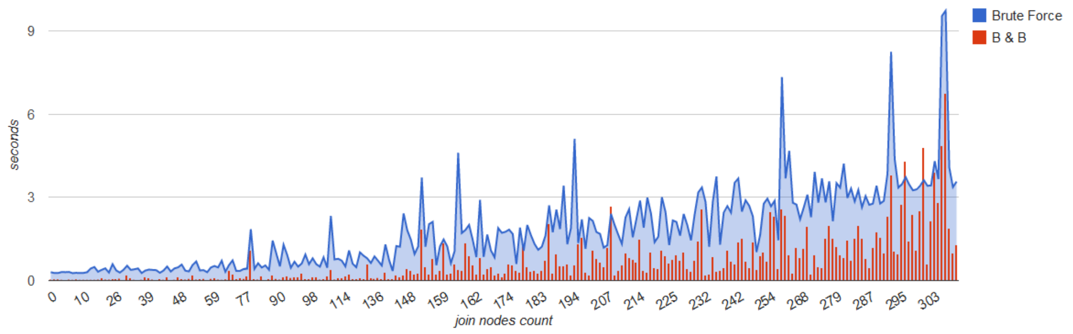


(a) 15 leaves

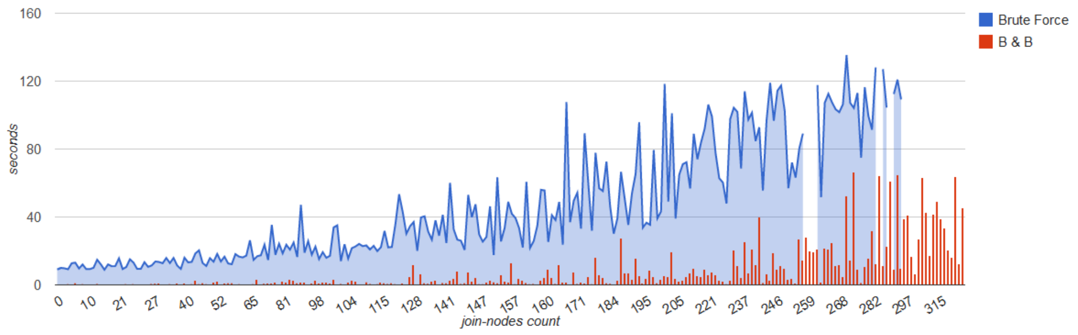


(b) 20 leaves

Fig. 4.4: Decomposition trees consisting of thick empty headed spiders of random size.



(a) 15 thick empty headed spiders and 300 individual vertices



(b) 20 thick empty headed spiders and 300 individual vertices

Fig. 4.5: Decomposition trees consisting of empty headed thick spiders and individual vertices.

Tab. 4.1: Summary of average execution time and visited nodes for different instances.

	Size: 15	**B&B**		**BF**	Size: 20	**B&B**		**BF**
Group	Join density	**Nodes**	**Time**	**Time**	Join density	**Nodes**	**Time**	**Time**
3s	0.08	51.77%	0.68	0.26	0.08	43.68%	18.14	8.61
	0.29	15.17%	0.24	0.31	0.29	5.45%	2.97	11.74
	0.50	12.10%	0.23	0.39	0.51	10.72%	6.86	15.38
	0.71	23.29%	0.51	0.53	0.69	9.81%	8.60	22.23
	0.93	62.80%	1.63	0.70	0.92	52.70%	54.85	31.23
6s	0.07	2.32%	0.05	0.32	0.08	0.51%	0.34	11.18
	0.30	4.84%	0.10	0.43	0.32	2.08%	1.70	18.53
	0.50	6.69%	0.18	0.64	0.53	3.29%	3.65	28.58
	0.72	11.16%	0.36	0.86	0.72	6.43%	8.67	37.41
	0.93	25.17%	1.03	1.18	0.92	15.95%	26.83	51.63
9s	0.07	0.59%	0.01	0.39	0.07	0.11%	0.09	13.33
	0.28	1.08%	0.03	0.60	0.28	0.22%	0.24	21.79
	0.50	2.04%	0.08	0.89	0.49	0.58%	0.82	34.53
	0.71	4.12%	0.20	1.31	0.71	1.29%	2.60	51.92
	0.94	10.18%	0.58	1.60	0.92	3.97%	8.94	66.20
rs	0.07	2.56%	0.05	0.33	0.09	1.08%	0.67	12.53
	0.28	3.90%	0.09	0.47	0.28	1.84%	1.39	17.61
	0.51	6.35%	0.17	0.67	0.49	2.16%	2.12	24.62
	0.72	8.12%	0.27	0.86	0.71	2.85%	3.73	35.97
	0.92	9.17%	0.38	1.15	0.93	3.18%	5.88	50.90
rs-300	0.10	2.50%	0.06	0.39	0.09	0.82%	0.53	12.92
	0.29	4.32%	0.15	0.73	0.31	1.62%	1.49	24.53
	0.50	8.12%	0.45	1.45	0.50	2.55%	3.61	42.21
	0.70	10.99%	0.80	2.26	0.70	3.47%	8.43	78.94
	0.91	14.81%	1.79	3.66	0.90	7.27%	27.42	106.47

3 only with no *join* operations the algorithm has to explore the entire decision tree. For the other groups, there is a slight increment in the amount of visited nodes as the *join* nodes density grows.

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

We have analyzed the current best approximation algorithm for the MSC problem over P_4-sparse graphs and, in the search for a better algorithm or a better approximation factor, we introduced new lower bounds to this problem.

These lower bounds open up a new way in the quest to prove that the proposed alternative to the current algorithm (ϕ_{3111}) is actually a k-approximation algorithm for some $k < 2$.

We performed exhaustive search looking for a graph or graph pattern that achieves the highest possible approximation factor for any of the bounds we have. This search led us to decomposition trees with similar structure, similarity that could be the reason for the high value of the approximability factor they achieve.

Even though the proof was not found, we successfully implemented a branch and bound algorithm for the problem that finds the optimal solution very quickly for relatively large graphs compared to the brute force implementation that we developed for comparison.

We have analyzed the execution time for the branch and bound implementation for different types and sizes of decomposition trees. We observed that for the vast majority of the instances the algorithm has a very good response and we also identified instances for which it is forced to explore almost the whole solution space.

We implemented caching of partial solutions inside the representation structure of the decomposition trees in order to avoid recalculation, since it was a major reason for the algorithms' overhead. The cached solutions remain in the structure after running, which means that they could be used for further queries, for example, in a context in which minor changes have to be made to the input graph, that can be dynamically reflected in the decomposition tree in linear time.

Moreover, it could be possible to easily alter the implementation to allow interruptions at any point of the execution and make it return a feasible solution together with a quality factor, in terms of the optimal solution. The quality factor could be theoretically lower than 2, by using the current approximation algorithm as a heuristic, and significantly lower than 2 in practice.

5.2 Future work

In the future, it would be interesting to continue this work in several directions:

- look for new bounds following the idea of considering invalid colorings. That is, for example, colorings that use same numbers for adjacent vertices or colorings that use color numbers other than positive integers,

- study the pattern seen in the decomposition trees that achieve the highest approximability factor found and try to characterize it in order to decide whether any of the lower bounds could yield a constant approximation factor.

- look for ways to detect and "break" symmetries in the decomposition trees in order to avoid this source of repeated calculations,

- look for different branching schemes, for example, try to identify thick spiders with empty head that are more relevant to the structure of the optimal solution, or more “penalizing” in order to explore those branches first,
- check if there might exist a link between the structure of the decomposition tree and the way it should be colored. For example, by trying to identify a way of estimating the “likeliness” of a certain thick spider with empty head to be colored in a certain way in the optimal solution,
- explore differences and similarities of different optimal solutions for the same instance graphs.

Finally, we propose the following conjecture: ϕ_{3111} is a 1.33-approximation algorithm.

Appendix A

PROOFS

We include now the proofs of the lemmas and theorems that have been omitted in Section 1.3. All of these proofs are part of previous work by F. Bonomo and M. Valencia-Pabon [3].

Proof. (Lemma 7) Let p be a maximal sequence of G and let $S_1, \ldots, S_k$ be a partition associated with p. Let $S_{i_1}, \ldots, S_{i_t}$ be the sets in the partition having nonempty intersection with $V(G_1)$. We claim that $\{i_1, \ldots, i_t\} = \{1, \ldots, t\}$. Otherwise, there is some value i such that $S_i \cap V(G_1) = \emptyset$ and $S_{i+1} \cap V(G_1) \neq \emptyset$. Since no vertex of G_1 is adjacent to a vertex of G_2, vertices in $S_{i+1} \cap V(G_1)$ can migrate to S_i, obtaining a sequence that strictly dominates p, a contradiction. The same happens for G_2, so p can be expressed as $p = p_1 + p_2$, where p_1 and p_2 are sequences of G_1 and G_2, respectively. By Lemma 6, they must be maximal for the corresponding graphs. $\square$

Proof. (Lemma 8) Let p be a maximal sequence of G and let $S_1, \ldots, S_k$ be a partition associated with p. Since every vertex of G_1 is adjacent to all the vertices of G_2 in G, each S_i, $1 \leq i \leq k$, is entirely contained either in G_1 or in G_2. Besides, as p is maximal, it is non-increasing. So p can be expressed as $p = p_1 \star p_2$, where p_1 and p_2 are sequences of G_1 and G_2, respectively. By Lemma 4, they must be maximal for the corresponding graphs. $\square$

Proof. (Lemma 9) Let p be the sequence of G associated to $S_1, \ldots, S_k$, that is, $p[i] = |S_i|$ for $i = 1, \ldots, k$, $p[i] = 0$ for $i > k$. Let q be the sequence of $G[H]$ associated to $S_{i_1} \cap H, \ldots, S_{i_t} \cap H$, that is, $q[j] = |S_{i_j} \cap H|$ for $j = 1, \ldots, t$, $q[j] = 0$ for $j > t$. Let q' be a maximal sequence for $G[H]$ such that $q' \succeq q$. Since $\sum_{j=1}^{|q'|} q'[j] = \sum_{j=1}^{|q|} q[j] = |H|$, we have $|q'| \leq |q| = t$. Let $S'_1, \ldots, S'_t$ be a partition of H associated with q', where maybe some of the sets are empty. Notice that every vertex in $(S_{i_1} \cup \ldots \cup S_{i_t}) \setminus H$ has at least a non-neighbor in H, since $S_{i_1}, \ldots, S_{i_t}$ are the sets having nonempty intersection with H and they are independent sets. Since H is an homogeneous set of G, every vertex in $(S_{i_1} \cup \ldots \cup S_{i_t}) \setminus H$ has no neighbors in H. So we can consider the partition of $V(G)$ obtained from $S_1, \ldots, S_k$ by replacing S_{i_j} by $(S_{i_j} \setminus H) \cup S'_j$, for $j = 1, \ldots, t$, that is a partition of $V(G)$ into independent sets. Let p' be the sequence associated to this new partition. Then $p'[i] = p[i]$ for $i \notin \{i_1, \ldots, i_t\}$, while $p'[i_j] = p[i_j] - q[j] + q'[j]$ for $j = 1, \ldots, t$. It is easy to see that $p' \succeq p$ because $q' \succeq q$, and that the domination is strict for p' and p if it is strict for q' and q. Since p is maximal for G, it follows that $q' = q$ and q is maximal for $G[H]$. $\square$

Proof. (Lemma 10) Let $S_1, \ldots, S_{|p|}$ be a partition associated with p. It is clear that no set of the partition can contain both vertices from C and R, since all the vertices in C are adjacent to all the vertices in R. If there is a set S_i containing vertices of S and no vertex of R, then either $S_i \subseteq S$ or S_i contains exactly one vertex of C, because C is a complete set. Since $R \neq \emptyset$, there is some set S_j containing vertices from R, thus $S_j \subseteq R \cup S$. Then the vertices in $S_i \cap S$ can migrate to S_j, possibly swapping S_i and S_j if $i < j$, thus obtaining a partition associated with a sequence p' such that $p' \succeq p$. Since p is maximal, $p' = p$. Therefore, there exists a partition associated with p in which every set is either entirely contained in R, or entirely contained in C, or intersects both R and S. Sets entirely contained in C have only one element, and since p is maximal, thus non-increasing, we may assume that these are the last $|C|$ sets. From now on, we will assume that $S_1, \ldots, S_{|p|}$ is such a partition. In particular, $S_1 \subseteq S \cup R$. Suppose that there is a set S_i, $i > 1$, containing vertices of S. Then the vertices in $S_i \cap S$ can migrate to S_1, obtaining a sequence that strictly dominates p, a contradiction. □

Proof. (Lemma 11) Let p be a maximal sequence of G, and $S_1, \ldots, S_{|p|}$ a maximal partition associated with p. Since $R \neq \emptyset$, by Lemma 10, we may assume that $S_1, \ldots, S_{|p|}$ is such that sets $S_{|p|-|C|+1}, \ldots, S_{|p|}$ are entirely contained in C, S_1 intersects both R and S and $S_2, \ldots, S_{|p|-|C|}$ (when $2 \leq |p| - |C|$) are entirely contained in R. By Lemma 9, $p[1] - |C|, p[2], \ldots, p[|p| - |C|]$ (or simply $p[1] - |C|$ when $|p| = |C| + 1$) is a maximal sequence for $G[R]$. Conversely, for each maximal sequence q of $G[R]$ associated with partition $T_1, \ldots, T_{|q|}$, define sequence q' of G with $|q'| = |q| + |C|$ and where $q'[1] = q[1] + |C|$, $q'[i] = q[i]$ for $2 \leq i \leq |q|$ (if $|q| \geq 2$), and $q'[i] = 1$ for $|q| + 1 \leq i \leq |q| + |C|$, associated with the partition $T_1 \cup S, \ldots, T_{|q|}, \{c_1\}, \ldots, \{c_k\}$ if $|q| \geq 2$, $T_1 \cup S, \{c_1\}, \ldots, \{c_k\}$ otherwise. Let q_1 and q_2 be maximal sequences of $G[R]$, and let q'_1 and q'_2 be their respective maximal sequences of G constructed as below. It is easy to see that if $q_1 || q_2$ then $q'_1 || q'_2$, so the lemma holds. □

Proof. (Lemma 12) Let $S = \{s_1, \ldots, s_k\}$ and $C = \{c_1, \ldots, c_k\}$, with $k \geq 2$. Let $S_1, \ldots, S_t$ be a partition of the vertex set of G into independent sets, with $t \geq 1$, such that its associated sequence p is maximal. By hypothesis, we have that $R = \emptyset$. Note first that each vertex $c_i \in C$ must belong to a different independent set S_j and so, $t \geq k$. Now, by definition of a thin spider, each vertex s_i is adjacent to vertex c_j if and only if $i = j$. We claim that there is S_i such that $S_i = S$ or there are S_i and S_j, with $i \neq j$, such that $S_i = (S \setminus \{s_n\}) \cup \{c_n\}$ and $S_j = \{s_n, c_m\}$, for some $n, m \in \{1, \ldots, k\}$, with $m \neq n$. Assume that it is not true. Suppose first that there are three sets S_i, S_j, S_l, with $i < j < l$, such that each one of them contains at least one vertex of S. Let $s_q \in S$ be a vertex in S_l. Then vertex $c_q \in C$ belongs to at most one of S_i or S_j but not to both. Thus, vertex s_q must migrate to one of S_i or S_j that contains no vertex c_q, which gives a sequence that strictly dominates p, a contradiction. Therefore, vertices in S belong either to only one set S_i or to two different sets S_i and S_j, with $i < j$. If S_i contains no vertex of C, all the vertices in $S \cap S_j$ must migrate to S_i, which gives a sequence that strictly dominates p, a contradiction. Else, S_i contains exactly one vertex c_n of C. In that case, by similar arguments, only vertex s_n could be in S_j. Since p is maximal, S_j contains also a vertex in C. (Otherwise we can merge two sets, obtaining a sequence that strictly dominates p, a contradiction.) As p is maximal, then p is such that: (i) $p[1] = k$ and $p[i] = 1$ for $2 \leq i \leq k + 1$, that is, $S_1 = S$ and $S_j = \{c_{j-1}\}$ for $2 \leq j \leq k + 1$; or (ii) $p[1] = k$, $p[2] = 2$, and $p[i] = 1$ for $3 \leq i \leq k$, that is, sequence p is associated with the partition $S_1 = (S \setminus \{s_1\}) \cup \{c_1\}$, $S_2 = \{s_1, c_2\}$, and $S_l = \{c_l\}$ for $3 \leq l \leq k$. Clearly, the sequence of Case (ii) dominates the one of Case (i), and it is the only maximal sequence for G. □

Proof. (Lemma 13) Let $S = \{s_1, \ldots, s_k\}$ and $C = \{c_1, \ldots, c_k\}$, with $k \geq 3$. By hypothesis, we have that $R = \emptyset$. By definition of a thick spider, each vertex s_i is adjacent to vertex c_j if and only if $i \neq j$.

The sequence p_1 with $|p_1| = k$, and such that $p_1[i] = 2$ for $1 \leq i \leq k$, can be obtained by defining $S_i = \{c_i, s_i\}$ for $1 \leq i \leq k$. The sequence p_2 with $|p_2| = k + 1$, and such that $p_2[1] = k$ and $p_2[i] = 1$ for $2 \leq i \leq k + 1$, can be obtained by defining $S_1 = S$ and $S_i = \{c_{i-1}\}$ for $2 \leq i \leq k + 1$. Moreover, we have that $p_1 || p_2$. In fact, let $j_1 = 1$ and $j_2 = k$. Then, $p_1[1] = 2 < k = p_2[1]$ and $\sum_{i=1}^{j_2} p_1[i] = 2k > 2k - 1 = \sum_{i=1}^{j_2} p_2[i]$.

We will show now that these are the only two maximal sequences for G. Let $S_1, \dots, S_t$ be a partition of the vertex set of G into independent sets, with $t \geq 1$, such that its associated sequence p is maximal. First notice that there is at most one set entirely contained in S, because two such sets could be merged obtaining a sequence that strictly dominates p, a contradiction.

Suppose first that there is some set S_i containing more than one vertex of S. Since no two vertices of S have a common non-neighbor in C, then $S_i \subseteq S$ and it is the only set entirely contained in S. Every other set is either contained in C or has one vertex of C and one of S. Since p is non-increasing, we may assume $i = 1$. If some set S_i with $i > 1$ contains a vertex of S, it can migrate to S_1, leading to a sequence that strictly dominates p, a contradiction. So $S_1 = S$ and $p = p_2$.

Suppose now that no set contains more than one vertex of S. Then each set is either composed by one vertex of C, or by one vertex of S, or by a vertex $s_i \in S$ and its only non-adjacent vertex $c_i \in C$. Clearly, p_1 dominates every such a sequence, so $p = p_1$. □

Proof. (Theorem 2) Let G be a P_4-sparse graph such that in its modular decomposition there are no thick spiders (S, C, R) with $|C| \geq 3$ and $R = \emptyset$.

1. We will prove by induction that G admits a unique maximal sequence p and that $|p| = \chi(G)$. This implies $s(G) = \chi(G)$. By Theorem 1, G is either trivial, or the union or join of two smaller P_4-sparse graphs G_1 and G_2, or G is a spider (S, C, R) and $G[R]$ is P_4-sparse. If G is trivial, the property holds. Suppose G is the union or join of G_1 and G_2. By inductive hypothesis, for $i = 1, 2$, G_i has a unique maximal sequence p_i, and $|p_i| = \chi(G_i)$. If $G = G_1 \cup G_2$ then, by Lemma 7, G has a unique maximal sequence $p = p_1 + p_2$. Therefore, $|p| = \max\{|p_1|, |p_2|\} = \max\{\chi(G_1), \chi(G_2)\} = \chi(G)$. If $G = G_1 \vee G_2$ then, by Lemma 8, G has a unique maximal sequence $p = p_1 \star p_2$. Therefore, $|p| = |p_1| + |p_2| = \chi(G_1) + \chi(G_2) = \chi(G)$. Finally, if G is a spider (S, C, R), then either G is a thin spider with $R = \emptyset$ or $R \neq \emptyset$. In the first case the property follows by Lemma 12. In the second case, by inductive hypothesis, $G[R]$ has a unique maximal sequence q, and $|q| = \chi(G[R])$. By Lemma 11, there exists only one maximal sequence p of G, and $|p| = |q| + |C| = \chi(G[R]) + |C| = \chi(G)$.

   Now, let n be the number of vertices in G. Let T be the decomposition rooted tree associated with G. It is well known that T can be computed in linear time [9]. We will show that the unique maximal sequence p of G and a partition associated with p can be computed from T in polynomial time. In order to compute an optimal coloring with $s(G)$ colors and sum $\Sigma(G)$ for this case, we proceed from the leaves to the root in T as follows. If x is a leaf in T then its associated partition is $S_1 = \{x\}$ having as maximal sequence p, with $|p| = 1$ and $p[1] = 1$. If node $x \in T$ is a union-node (resp. join-node) then, by Lemma 7 (resp. Lemma 8), the unique maximal sequence and its corresponding optimal partition of the vertex set of G_x into independent sets can be computed from the unique maximal sequences and their corresponding optimal partitions of the children of x. If node $x \in T$ is a spider-partition node representing the spider $\sigma = (S, C, R)$ then, the unique maximal sequence and its corresponding optimal partition of the vertex set of G_x into independent sets can be computed either as in Lemma 12 (if σ is a thin spider with $R = \emptyset$) or from the unique maximal sequence and their corresponding optimal partitions of the child $G_x[R]$ of x as shown in Lemma 11, if $R \neq \emptyset$. Finally, notice that each node $x \in T$ needs $O(n)$ time to compute its optimal partition. As there are at most $2n - 1$ nodes in T, then the complexity time of the algorithm is bounded by $O(n^2)$.

2. It follows by induction from Theorem 1, and using Lemma 7 (resp. Lemma 8) if G is a disjoint union (resp. join) of P_4-sparse graphs, and Lemma 12 (resp. Lemma 11) if G is a thin spider (S, C, R) with $R = \emptyset$ (resp. G is a spider (S, C, R) with $R \neq \emptyset$).

□

Proof. (Theorem 3) The statement holds for $t = 0$ by Theorem 2. Suppose $t \geq 1$, and let $\sigma^1, \ldots, \sigma^t$ be the thick spiders in the decomposition tree T of G such that $\sigma_j = (S^j, C^j, \emptyset)$ and $|C^j| \geq 3$, for $j = 1, \ldots, t$. By Lemma 13, each σ^j has exactly two maximal sequences. Clearly, there are 2^t ways of choosing maximal sequences (and their corresponding partitions) for the t thick spiders σ^j. Now, given a fixed choice for the thick spiders σ^j and by using the algorithm in the proof of item (1) of Theorem 2, we can compute in $O(n^2)$ time a maximal sequence and its corresponding partition into independent sets for G. This shows that G has at most 2^t maximal sequences and that an optimal coloring with $s(G)$ colors and sum $\Sigma(G)$ can be computed in $O(2^t n^2)$ time. Finally, note that for each thick spider σ^j, one of its maximal sequences has length $\chi(\sigma^j) + 1$ and thus, by induction, it can be proved that the number of colors used in an optimal solution for G is upper bounded by $\chi(G) + t$. □

BIBLIOGRAPHY

[1] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, *On chromatic sums and distributed resource allocation*, Information and Computation, 140 (1998), pp. 183–202.

[2] A. Bar-Noy and G. Kortsarz, *Minimum color sum of bipartite graphs*, Journal of Algorithms, 28 (1998), pp. 339–365.

[3] F. Bonomo and M. Valencia-Pabon, *Minimum sum coloring of P_4-sparse graphs*, in Proc. V Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS), vol. 35 of Electronic Notes in Discrete Mathematics, Elsevier, 2009, pp. 293–298.

[4] U. Feige, L. Lovász, and P. Tetali, *Approximating min sum set cover*, Algorithmica, 40 (2004), pp. 219–234.

[5] K. Giaro, R. Janczewski, M. Kubale, and M. Malafiejski, *A 27/26-approximation algorithm for the chromatic sum coloring of bipartite graphs*, in Proc. 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), vol. 2462 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 135–145.

[6] M. M. Halldórsson, G. Kortsarz, , and H. Shachnai, *Sum coloring interval and k-claw free graphs with application to scheduling dependent jobs*, Algorithmica, 37 (2003), pp. 187–209.

[7] C. T. Hoàng, *Perfect graphs*, PhD thesis, School of Computer Science, McGill University, 1985.

[8] B. Jamison and S. Olariu, *Recognizing P_4-sparse graphs in linear time*, SIAM Journal on Computing, 21 (1992), pp. 381–406.

[9] B. Jamison and S. Olariu, *A tree representation for P_4-sparse graphs*, Discrete Applied Mathematics, 35 (1992), pp. 115–129.

[10] K. Jansen, *Complexity results for the optimum cost chromatic partition problem*, in Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP), vol. 1256 of Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 727–737.

[11] E. Kubicka, *The Chromatic Sum of a Graph*, PhD thesis, Western Michigan University, 1989.

[12] E. Kubicka and A. J. Schwenk, *An introduction to chromatic sums*, in Proc. 17th ACM Annual Computer Science Conference, 1989, pp. 39–45.

[13] S. Nicoloso, M. Sarrafzadeh, and X. Song, *On the sum coloring problem on interval graphs*, Algorithmica, 23 (1999), pp. 109–126.

[14] S. Nikolopoulos, L. Palios, and C. Papadopoulos, *A Fully Dynamic Algorithm for the Recognition of P_4-Sparse Graphs*, Graph-Theoretic Concepts in Computer Science, Department of Computer Science, University of Ioannina, 2000.

[15] T. SZKALICZKI, *Routing with minimum wire length in the dogleg-free Manhattan model is NP-complete*, SIAM Journal on Computing, 29 (1999), pp. 274–287.