COMPUTING SOLUTIONS OF THE PAINTSHOP-NECKLACE PROBLEM

FRÉDÉRIC MEUNIER AND BERTRAND NEVEU

Abstract. How to assign colors to occurrences of cars in a car factory ? How to divide fairly a necklace between thieves who have stolen it ? These two questions are addressed in two combinatorial problems that have attracted attention from a theoretical point of view these last years, the first one more by people from the combinatorial optimization community, the second more from the topological combinatorics and computer science point of view.

The first problem is the paint shop problem, defined by Epping, Hochstättler and Oertel in 2004. Given a sequence of cars where repetition can occur, and for each car a multiset of colors where the sum of the multiplicities is equal to the number of repetitions of the car in the sequence, decide the color to be applied for each occurrence of each car so that each color occurs with the multiplicity that has been assigned. The goal is to minimize the number of color changes in the sequence.

The second problem, highly related to the first one, takes its origin in a famous theorem found by Alon in 1987 stating that a necklace with t types of beads and qa_u occurrences of each type u (a_u is a positive integer) can always be fairly split between q thieves with at most $t(q-1)$ cuts. An intriguing aspect of this theorem lies in the fact that its classical proof is completely non-constructive. Designing an algorithm that computes theses cuts is not an easy task, and remains mostly open.

The main purpose of the present paper is to make a step in a more operational direction for these two problems by discussing practical ways to compute solutions for instances of various sizes. Moreover, it starts with an exhaustive survey on the algorithmic aspects of them, and some new results are proved.

Introduction

The paintshop-necklace problem. There is a beautiful combinatorial problem, which, depending on the point of view, appears as an hard optimization problem or as one of the most beautiful applications of algebraic topology in combinatorics.

The optimization version of the problem is called the *paint shop problem*, and was defined by Epping, Hochstättler and Oertel [11]. The origins of the model lie in car manufacturing with individual demands, which is reported to occur often in Europe. Given a sequence of cars where repetition can occur, and for each car a multiset of colors where the sum of the multiplicities is equal to the number of repetitions of the car in the sequence, decide the color to be applied for each occurrence of each car so that each color occurs with the multiplicity that has been assigned. The goal is to minimize the number of color changes in the sequence. If cars are considered to be letters in an alphabet, the following is a formalisation.

Key words and phrases. combinatorial optimization; integer programming; local search; paintshop problem; path-following method; splitting necklace; TFNP problem.

Paint shop problem

INPUT: A finite alphabet Σ whose elements are called *letters*, a word $\boldsymbol{w} = (u_1, \ldots, u_n) \in \Sigma^*$, a finite color set C and a requirement $r : \Sigma \times C \to \mathbb{Z}_+$.

TASK: Find a *coloring* $\boldsymbol{c} = (c_1, \ldots, c_n) \in C^n$, such that for all $u \in \Sigma$ and $c \in C$, we have

$$|\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = c\}| = r(u, c),$$

and the number of color changes within $\boldsymbol{c}$ is minimized.

We say that we have a *color change* in $\boldsymbol{c}$ whenever $c_i \neq c_{i+1}$. The minimum of the number of color changes is denoted $\gamma = \gamma(\boldsymbol{w}; r)$.

The paint shop problem has a solution only if $\sum_{c \in C} r(u, c)$ is exactly the number of occurrences of u in $\boldsymbol{w}$.

A special case has received a special attention: the *binary paint shop problem*, which is the case when $|C| = 2$ and $r(u, c) = 1$ for the two elements c in C and for all $u \in \Sigma$. In this case, we have $n = 2|\Sigma|$.

The other version of this problem is the *necklace splitting problem*, which found its motivation in a theorem of Alon [1] (for a didactic proof, see [14]). q thieves have stolen a precious necklace, which has n beads. These beads belong to a set Σ of t different types. Assume that the number of beads of each type is a multiple of q, say qa_u beads of type $u \in \Sigma$, where a_u is a positive integer. Remark that we have $q \sum_{u \in \Sigma} a_u = n$.

As we do not know the exact value of each type of beads, a fair division of the necklace consists in giving the same number of beads of each type to each thief. The number of beads of each type is a multiple of q, hence such a division is always possible: cut the chain at the $n - 1$ possible positions. Isn't it possible to do the division with less cuts? The answer is yes, as shown by the following theorem proved by Alon:

Theorem 1. [1] *A fair division of the necklace with t types of beads between q thieves can be done with no more than $(q-1)t$ cuts.*

It was first proved by Goldberg and West in 1985 [13] when $q = 2$ (this last theorem got a simpler proof in 1986, found by Alon and West and using the Borsuk-Ulam theorem [4]).

Note that this theorem tells nothing about the way of finding such cuts. Given a necklace satisfying the condition of the theorem, the necklace splitting problem requires to find the $(q-1)t$ (or less) cuts giving a fair division.

To make the parallel with the paint shop problem, the necklace can be seen as a word $\boldsymbol{w}$ where the letters are the types of beads and the colors are the thieves. A formalisation is (only to respect the symmetry with the presentation of the paint shop problem)

Necklace splitting problem

INPUT: A finite alphabet Σ of size t whose elements are called *letters*, a word $\boldsymbol{w} = (u_1, \ldots, u_n) \in \Sigma^*$ in which each letter u is present qa_u times where a_u is a non negative integer.

TASK: Find a coloring $\boldsymbol{c} = (c_1, \ldots, c_n) \in \{1, \ldots, q\}^n$ such that for each $u \in \Sigma$, we have

$$|\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = 1\}| = \ldots = |\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = q\}|,$$

and such that the number of color changes is less than or equal to $(q-1)t$.

An instance of the necklace splitting problem is a special instance of the paint shop problem where each letter must occur a multiple of q times, where the set C of colors has cardinality q and

$r(u,1) = \ldots = r(u,q) = a_u$ for all $u \in \Sigma$. c_i indicates which thief (numbered $1, \ldots, q$) gets the ith bead. Theorem 1 states that in this case $\gamma(\boldsymbol{w}, r) \leq (q-1)t$.

Finding solutions for the necklace splitting theorem, in addition to the motivation from a complexity point of view explained below, has applications for instance in VLSI circuit design [8, 18] or for hashing problems [3].

Plan. The paper starts with a survey of known algorithmic approaches and complexity properties (Section 1); new results are also proposed in this section. In the second section (Section 2), we describe a linear programming formulation, and discuss some variants. The computation of solutions for the necklace splitting problem through a pivoting scheme on cubical complexes of [16] is described in Section 3. For large instances, local searches can be tried (Section 4). Finally, all these approaches are evaluated and compared in Section 5.

1. What is known from the algorithmic point of view

1.1. The paint shop problem.

The paint shop problem is hard.

Theorem 2. [9, 17] *The paint shop problem is APX-hard, even if $|C| = 2$ and $r(u,c) = 1$ for all $c \in C$ and $u \in \Sigma$ (the binary case).*

Theorem 3. [11] *The paint shop problem is NP-hard even if $|\Sigma| = 2$.*

An exact algorithm is proposed by Epping, Hochstättler and Oertel in [11] using a dynamic programming approach. The algorithm runs in $O\left(|C| n^{(|C|-1)|\Sigma|}\right)$ where n is the length of $\boldsymbol{w}$, hence it seems unlikely to be applicable in practice.

1.2. The binary paint shop problem.

1.2.1. *Polynomial time solvability.* For the binary case, which is APX-hard according to Theorem 2 a little more is known. In [9], the problem is translated into the matroid framework : the binary paint shop problem becomes the problem of finding a minimum size circuit containing a special element in a binary matroid. The authors note that finding the minimum number of changes for the binary paint shop problem is polynomial when the binary matroid satisfies the strong max-flow-min-cut property. In [17], it is explained how to find a minimum coloring itself in this case. Nevertheless, the strong max-flow-min-cut property for the binary matroid does not translate easily in terms of paint shop. In [17], some attempt is made in this direction and for instance, the following corollary is derived :

Proposition 1. [17] *The binary paint shop problem is polynomially solvable if the graph whose vertices are the letters and the edges are the pairs of* crossing letters *(that is, letters a, b, which occur in the order a, b, a, b) is bipartite.*

In the same paper, the following theorem is proved, roughly through an equivalence with the min-uncut problem in 4-regular graphs, leading to "good solutions" for the binary paint shop problem.

Theorem 4. [17] *Let $t = |\Sigma|$ be the size of the alphabet. A binary paint shop problem has a solution with at most $1/4p + 3/4t$ color changes, where p is a lower bound of the optimal solution, and both p and the solution can be computed in polynomial time.*

It improves the easy bound $\gamma \leq t$ which can be obtained with a simple greedy algorithm (it is also a consequence of Theorem 1 for $q = 2$).

These "good solutions" are the best known today. Whether there is a constant factor approximation algorithm for the binary paint shop problem (asked implicitly in [11] and explicitly in [9]) is an open question.

Open question. Is there a constant factor polynomial approximation algorithm for the binary paint shop problem ?

1.2.2. *Greedy approaches.* The greedy algorithm can be quite efficient in some cases, as the following theorems proved in [5] show. The *greedy algorithm* for an instance of the binary paint shop problem consists in the coloring of the letters in the given order so as to change the current color only at the second occurrences of letters, and only if necessary. A word $\boldsymbol{w}'$ is a *subword* of a word $\boldsymbol{w} = (u_1, \ldots, u_n)$ if $\boldsymbol{w}'$ is a subsequence of $\boldsymbol{w}$ (two consecutive occurrences of a letter in $\boldsymbol{w}'$ are not necessarily consecutive in $\boldsymbol{w}$).

Theorem 5. [5] *The instances of the binary paint shop problem having neither a subword of the form xyxzzy nor a subword of the form xyyzxz are solved optimally by the greedy algorithm.*

Note that the two excluded subwords are the same up to a mirror symmetry.

Theorem 6. [5] *Let Σ be a finite alphabet of size t. When the instances of the binary paint shop problem are chosen uniformly at random among all instances of fixed size $2t$ with alphabet Σ, then*

$$\mathbb{E}(g) \leq \frac{2}{3}t,$$

where g is the number of color changes when one applies the greedy algorithm.

Actually, in [6], it is proved that actually we have (conjectured in [5])

Theorem 7. *Let Σ be a finite alphabet and let t be its size. When the instances of the binary paint shop problem are chosen uniformly at random among all instances of fixed size $2t$ with alphabet Σ, then*

$$\lim_{t \to +\infty} \frac{1}{t}\mathbb{E}(g) = \frac{1}{2}$$

where g is the number of color changes when one applies the greedy algorithm.

In the same paper, another greedy algorithm, the "recursive greedy algorithm", is presented and it is proved that

Theorem 8. *Let Σ be a finite alphabet and let t be its size. When the instances of the binary paint shop problem are chosen uniformly at random among all instances of fixed size $2t$ with alphabet Σ, then*

$$\frac{2}{5}t + \frac{8}{15} \leq \mathbb{E}(rg) \leq \frac{2}{5}t + \frac{7}{10}$$

where rg is the number of color changes when one applies the recursive greedy algorithm.

1.2.3. *Lower bounds and quality of the optimal solution.* In [9], a lower bound – computable in polynomial time through dynamic programming – has been proposed for this problem. The same bound has been used in [5] to prove Theorem 5. This bound is given in Proposition 2. We give a new result, which is a necessary and sufficient condition for this bound to be optimal, and also a new one about instances for which all feasible requirement r have solutions with less than t colors changes. We start also a discussion about the expected value of the optimal value of the binary paint shop problem.

Define for each input $\boldsymbol{w} = (u_1, \ldots, u_{2t})$ of the binary paint shop problem an hypergraph on the set $\{1, \ldots, 2t-1\}$ defined as

$$\mathcal{I}(w) := \{\{i, i+1, \ldots, j-1\} : 1 \leq i < j \leq 2t,\, u_i = u_j\}.$$

The hyperedges are intervals. In terms of the paintshop problem one can think of the elements of $\{1, \ldots, 2t-1\}$ as possible moments for color change: if moment i ($i = 1, \ldots, 2t-1$) is chosen, that means changing the color right after the occurrence of u_i (before the occurrence of u_{i+1}). The

binary paint shop problem consists in *designing a minimum number of color changes so that each hyperedge – each interval – of $\mathcal{I}(\boldsymbol{w})$ contains an odd number of them.* Less formally, suppose given a finite set of intervals on the real line, the binary paint shop problem aims to find the minimum number of points such that each interval contains an odd number of them. Any finite set of intervals is not a $\mathcal{I}(\boldsymbol{w})$. Easily,

Claim 1. *All intervals in $\mathcal{I}(\boldsymbol{w})$ have distinct right endpoints. The same holds for left endpoints.*

Let $\mathcal{C}$ be a set of subsets of a finite set V. Two subsets $A, B \in \mathcal{C}$ are said to be *crossing* if

$$A \not\subseteq B \quad \text{and} \quad B \not\subseteq A \quad \text{and} \quad A \cap B \neq \emptyset.$$

If there are no crossing subsets in $\mathcal{C}$, then $\mathcal{C}$ is said to be *laminar*. The notion of laminar set is a very classical one in combinatorial optimization. In the case of binary paintshop problem, one will need a more specific notion. A set $\mathcal{C}$ is *evenly laminar* if

- it is laminar, and
- for all $A \in \mathcal{C}$, there is an even number of subsets of $\mathcal{C}$ contained in A and distinct from A: the set $\{B \in \mathcal{C} : B \subsetneq A\}$ has an even cardinality.

The following proposition is proved in [9].

Proposition 2. [9] *Let $\boldsymbol{w}$ be an input of the binary paint shop problem and let $\mathcal{B} \subseteq \mathcal{I}(\boldsymbol{w})$. If $\mathcal{B}$ is evenly laminar, then one has $\gamma(\boldsymbol{w}) \geq |\mathcal{B}|$. In other words, the cardinality of any evenly laminar subset of $\mathcal{I}(\boldsymbol{w})$ is a lower bound for the binary paint shop problem.*

As already noted, the greedy algorithm allows us to find a solution of the binary paint shop problem with at most t changes. In general, the optimum $\gamma(\boldsymbol{w})$ is smaller than t, but Proposition 2 shows that if the input $\boldsymbol{w}$ is such that $\mathcal{I}(\boldsymbol{w})$ is evenly laminar, then we cannot do better than t. In fact, the evenly laminar inputs are precisely the inputs for which we cannot do better than t.

Proposition 3. *Let $\boldsymbol{w}$ be an input of the binary paint shop problem and let $t = |\Sigma|$ be the size of the alphabet. Then $\gamma(\boldsymbol{w}) = t$ if and only if $\mathcal{I}(\boldsymbol{w})$ is evenly laminar.*

In terms of the necklace splitting problem, this proposition characterizes completely the case when we cannot do better than the number of cuts given by the necklace splitting theorem (Theorem 1) for two thieves and for $a_u = 1$ for all $u \in \Sigma$. For more than two thieves or for different values of the a_u, such a characterization is not known.

The evenly laminar instances have another interesting property.

Proposition 4. *Let $\boldsymbol{w}$ be an input of the binary paint shop problem and let $t = |\Sigma|$ be the size of the alphabet. For each letter, fix the repartition of the colors you want among the two occurrences ($r(u, c) = 0$, 1 or 2). If $\mathcal{I}(\boldsymbol{w})$ is evenly laminar, then it is always possible to get this repartition with at most t colors changes. t color changes are necessary only when $r(u, c) = 1$ for each color c and for each letter $u \in \Sigma$.*

This proposition is a consequence of Proposition 3 above and Corollary 3 of the paper "Necklace bisection with one cut less than needed" by Simonyi [24]. This latter states

> *If we can use at most $t - 1$ cuts and fair splitting is not possible then the thieves still have the following option. Whatever way they specify two disjoint sets D_1, D_2 of the types of beads with $D_1 \cup D_2 \neq \emptyset$, it will always be possible to cut the necklace (with $t - 1$ cuts) so that the first thief gets more of those types of beads that are in D_1 and the second gets more of those in D_2, while the rest is divided equally.*

It can also be proved directly, as we do it below. The proof shows even that the greedy algorithm finds a solution. Note also that conversely Proposition 3 and Proposition 4 together prove the result by Simonyi for the case when $a_u = 1$ for all $u \in \Sigma$.

To prove Proposition 3, we will need the following lemma.

Lemma 1. *Let $\boldsymbol{w}$ be an input of the binary paint shop problem such that $\mathcal{I}(\boldsymbol{w})$ is evenly laminar. Let $T \subseteq \{1, \ldots, 2t-1\}$ be such that for each $I \in \mathcal{I}(\boldsymbol{w})$, the cardinality of $T \cap I$ is odd. Then for each $I \in \mathcal{I}(\boldsymbol{w})$ there is $x_I \in I \cap T$ in T such that $x_I \notin K$ whenever $K \in \mathcal{I}(\boldsymbol{w})$ and $K \subsetneqq I$.*

Proof. By induction, on the cardinality of $\mathcal{I}(\boldsymbol{w})$. Delete an interval J of $\mathcal{I}(\boldsymbol{w})$ that is maximal for inclusion. Apply induction. We get an x_I for each interval $I \neq J$ such that $x_I \in I \cap T$ but $x_I \notin K$ whenever $K \in \mathcal{I}(\boldsymbol{w}) \setminus \{J\}$ and $K \subsetneqq I$. It is easy to see that, since $\mathcal{I}(\boldsymbol{w})$ is evenly laminar and since $|T \cap I|$ is odd for all $I \in \mathcal{I}(\boldsymbol{w})$ such that $I \subseteq J$, there is an element of T which is contained in J but in no other interval of $\mathcal{I}(\boldsymbol{w})$. □

Proof of Proposition 3. Assume first that $\mathcal{I}(\boldsymbol{w})$ is evenly laminar. A solution with at most $|\mathcal{I}(\boldsymbol{w})|$ changes can be found through the greedy algorithm. Proposition 2 allows us to conclude that it is optimal.

For the other direction, assume now that $\gamma(\boldsymbol{w}) = |\mathcal{I}(\boldsymbol{w})|$. We will prove by induction on t, which is $|\mathcal{I}(\boldsymbol{w})|$, that in this case $\mathcal{I}(\boldsymbol{w})$ is evenly laminar. For $t = 1$, it is obvious.

Assume $t \geq 2$.

Let T be a subset of $\{1, \ldots, 2t-1\}$ such that each interval of $\mathcal{I}(\boldsymbol{w})$ contains an odd number of elements of T and suppose that T is minimum. In other words, T is an optimal solution.

Denote by J the interval that has $2t-1$ as right endpoint. Denote by k the new highest right endpoint of an interval in $\mathcal{I}(\boldsymbol{w}) \setminus \{J\}$ (note that $k = 2t-2$ or $k = 2t-3$). By induction, $\mathcal{I}(\boldsymbol{w}) \setminus \{J\}$ is evenly laminar otherwise the greedy algorithm would extend an optimal solution on $\mathcal{I}(\boldsymbol{w}) \setminus \{J\}$ to an optimal solution for $\mathcal{I}(\boldsymbol{w})$ that is smaller than T.

Suppose now for a contradiction that there exists an interval I in $\mathcal{I}(\boldsymbol{w})$ such that I and J are crossing. Take T' an optimal solution for $\mathcal{I}(\boldsymbol{w}) \setminus \{J\}$. According to Lemma 1, there is an $x \in T'$ such that x is in I but in no other interval $K \subseteq I$. Deleting x from T' and replacing it by one of the endpoints of I (depending on whether $x \in J$ or not), we get a solution for $\mathcal{I}(\boldsymbol{w})$ with only $|T'| < |T|$ color changes. Contradiction.

Hence, $\mathcal{I}(\boldsymbol{w})$ is laminar, and necessarily evenly laminar otherwise any solution for $\mathcal{I}(\boldsymbol{w}) \setminus \{J\}$ would be a solution for $\mathcal{I}(\boldsymbol{w})$. □

Proof of Proposition 4. We prove the proposition by induction on the number of letters. It is trivially true if $t = 1$.

Assume that $t > 1$. Since $\mathcal{I}(\boldsymbol{w})$ is evenly laminar, there is somewhere in $\boldsymbol{w}$ a sequence of consecutive terms of the form $XYYZZT$, letters X and T being possibly the same or empty, but not necessarily. Let $\boldsymbol{w'}$ be the word obtained by deleting $YYZZ$ from $\boldsymbol{w}$. By induction, we can get any repartition in at most $t-2$ color changes. Extend greedily the coloring on $YYZZ$ starting from X. We get at most 3 color changes on $XYYZZT$: at most one for the Y's, at most one for the Z's and at most one for T. If we get 3 color changes, the color of X and the color of T are distinct, which means that we already have one color change between X and T at that place in $\boldsymbol{w'}$. In any case, the increase of color changes is at most 2.

The same inductive scheme allows us to prove the statement about the instances for which t color changes are necessary. □

In [6], it is asked whether it is possible to find the expected value of the optimal value when the instances are drawn uniformly at random. According to Theorem 8, we have $\mathbb{E}(\gamma) \leq \frac{2}{5}t + \frac{7}{10}$. We were not able to propose an interesting lower bound on $\mathbb{E}(\gamma)$, but we conjecture that $\mathbb{E}(\gamma) = o(t)$. Anyway, with the help of the formulation of Subsection 2.1 and with the help of CPLEX (see Section 5 for more details) we have computed the average value $\bar{\gamma}$ of γ for various values of $|\Sigma|$. Each value $\bar{\gamma}$ in the following array results from the computation of the optimal solution of 10 instances of the binary paint shop problem uniformly taken at random among instances for fixed $|\Sigma|$.

$\lvert\Sigma\rvert$	10	20	40	80	90	100	110	120
$\bar{\gamma}$	4.1	7.2	13	23.4	26.1	28.2	31.1	35.4

Remark. When looking at the lower bound provided by Proposition 2, it is tempting to use it in a branch-and-bound scheme. Partial solutions are given naturally in this context by subsets $S \subseteq \{1, \ldots, 2t-1\}$ and by fixing an element in $\{0,1\}^S$: if $i \in S$ is fixed to 1, it means that i must be a color change (the color of u_i and u_{i+1} must be different), and if i is fixed to 0, then u_i and u_{i+1} must have the same color. We see that, to get lower bounds for partial solutions, we shall generalize the notion of evenly laminar set. Indeed, when a partial solution is already fixed, the binary paint shop problem becomes: *let $\mathcal{I}$ be a set of intervals of $\{1, \ldots, 2t-1\}$; let $\alpha : \mathcal{I} \to \{0,1\}$; design a minimum number of color changes so that each interval I of $\mathcal{I}$ contains a number $= \alpha(I) \bmod 2$ of them.* We could call this problem *the generalized binary paint shop problem.*

So, the suitable generalization of evenly laminar is the *α-laminar set.* A set $\mathcal{C}$ is α-laminar, with $\alpha : \mathcal{C} \to \{0,1\}$, if

- it is laminar, and
- for all $A \in \mathcal{C}$, the set $\{B \in \mathcal{C} : B \subsetneqq A\}$ has a cardinality $= \alpha(I) + 1 \bmod 2$.

It is not too difficult to prove that the cardinality of α-laminar subsets of an input of a generalized binary paint shop problem is a lower bound for this latter problem. To compute the largest α-laminar subset, the dynamic programming approach described in [9] can be straightforwardly adapted. Having a way to encode partial solutions and to compute lower bounds can lead to a branch-and-bound scheme. We have implemented it but it is not at all competitive in comparison with the linear programming approach of the next section (some evenly laminar bounds are computed in Subsection 5.2 and are far from the optimum). Nevertheless, this can be used to derive valid inequalities for programs (1) or (2) specialized for the binary paintshop of the type

$$\sum_{i \in \bigcup \mathcal{J}} y_i \geq |\mathcal{J}|$$

for some α-laminar subsets $\mathcal{J}$. Violated inequalities of this type can be computed through a similar dynamic programming approach, but we have not tested this possibility of improvement.

1.3. **The necklace splitting problem.** From a complexity point of view, the question of finding the number of cuts stated in Theorem 1 is an intriguing open question. Problems of similar nature are BROUWER, BORSUK-ULAM, SECOND HAMILTONIAN CIRCUIT, NASH, etc. (see [21] for their definitions). They are problems for which there is no chance to prove their NP-hardness, unless NP=co-NP [15], since the objects that have to be found always exist. Motivated by the question of computing a Nash equilibrium, Megiddo and Papadimitriou [15] defined the complexity class TFNP, consisting exactly of all search problems in NP for which every instance is guaranteed to have a solution.

A subclass of the TFNP class is the PPAD class (see [22]) for which the existence is proved through the following argument :

> *In any directed graph with one unbalanced node (node with outdegree different from its indegree), there must be another unbalanced node.*

This class was defined by Papadimitriou in 1994 [21] in a paper in which he also shows that there exists PPAD-complete problem, that is a problem in PPAD for which the existence of a polynomial algorithm would imply the existence of a polynomial algorithm for any problem in PPAD. The necklace splitting problem for $q = 2$ belongs to the class PPAD, since it is a consequence of the

Borsuk-Ulam theorem that belongs to PPAD[1]. Nevertheless, even if the search problem associated to the Borsuk-Ulam theorem is PPAD-complete, it is not known whether the necklace splitting problem for $q = 2$ is PPAD-complete. This leads to the following question.

Open question. Is the necklace splitting problem for $q = 2$ PPAD-complete ?

Or, a counterpart (asked for instance in [2])

Open question. Is the necklace splitting problem polynomial ?

Even in the case with two thieves ($q = 2$), and even after some effort, the best running times known so far are $O(n^{t-2})$ for $t \geq 3$ and $O(n)$ for $t = 2$ [13]. Actually, when checking the details of the algorithm, the running time in this paper is $\Theta(n^{t-2})$. It means that there is constant α such that the algorithm requires at least αn^{t-2} evaluations of splittings: roughly speaking, the algorithm computes the value of a function on more or less all discrete points – each of them corresponding to a splitting – of a piecewise linear manifold of dimension $t-2$ in order to decide the relative position of this manifold (the "winding number" in the terminology of this paper) with respect to a special point ($\bar{a}$ still with the notations of the paper); it does it for many manifolds in order to find by a binary search the one containing the special point. This has seemed to be sufficiently prohibitive, in addition with other implementation difficulties (each point on which the function is evaluated requires a Gaussian elimination in $O(t^3)$, the algorithm requires degeneracy checks, ...) and we did not try to implement this method.

The GRAPH ISOMORPHISM problem is an NP problem which is not known to belong to either the P class or the NP-complete class, and for which all attempts to settle its complexity have failed. It is usually suspected to lie somewhere between the P class or the NP-complete class.

We cannot resist to make the supposition that we have with the NECKLACE SPLITTING problem a counterpart in the PPAD context. The NECKLACE SPLITTING problem (even for $q = 2$) is maybe somewhere between the P class and the PPAD-complete class.

An algorithmic proof of the case with two thieves exists [16]. It is a path-following method similar to the one used by Scarf [23] to prove Sperner's lemma. But, contrary to this latter proof, the path travels in a cubical complex, instead of a simplicial complex, and the labels are vertices of a cube, instead of being integers (see Section 3). Whether there is similar proof for more than two thieves remains open. This question is probably difficult since, although there are several proofs of this kind for the Borsuk-Ulam theorem, which hides behind the necklace splitting theorem for two thieves, there is no proof at all of this kind for the more general Dold theorem (see for instance the work by de Longueville and Zivaljevic [10]), which hides behind the necklace splitting theorem for an arbitrary number of thieves.

Open question. Is there a path-following proof of the necklace splitting theorem for an arbitrary number of thieves ?

2. Linear programming

2.1. **A first formulation.** Consider an instance $(\boldsymbol{w}, r)$ of the paint shop problem with alphabet Σ and color set C. Assume $\boldsymbol{w} = (u_1, \ldots, u_n)$ to be of length n. To ease the discussion, we introduce the notation $P(u)$ which indicates for $u \in \Sigma$ the positions of the letter u in $\boldsymbol{w}$.

[1]In [21], the necklace splitting problem is stated to belong to PPAD for any q but it seems unlikely to be true since in the general case, the necklace splitting theorem is a consequence of the more general Dold theorem, which is in PPA-q.

Any solution $\boldsymbol{c} = (c_1, \ldots, c_n)$ can be encoded through variables $x_{i,c}$ for all $i \in \{1, \ldots, n\}$ and $c \in C$ where

$$x_{i,c} = \begin{cases} 1 & \text{if } c_i = c \\ 0 & \text{if not.} \end{cases}$$

These variables must satisfy

$$\sum_{i \in P(u)} x_{i,c} = r(u,c) \quad \text{for all } c \in C \text{ and } u \in \Sigma$$

(the requirement is satisfied) and

$$\sum_{c \in C} x_{i,c} = 1 \quad \text{for all } i \in \{1, \ldots, n\}$$

(each occurrence of a letter receives exactly one color).

Adding a variable y_i encoding whether there is a color change between u_i and u_{i+1}, we can formulate the binary paint shop problem as an integer linear program. Define for all $i \in \{1, \ldots, n-1\}$

$$y_i = \begin{cases} 1 & \text{if } x_{i,c} \neq x_{i+1,c} \text{ for some } c \in C \\ 0 & \text{if not.} \end{cases}$$

The formulation is then

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{n-1} y_i & \\ \text{s.t.} & \displaystyle\sum_{i \in P(u)} x_{i,c} = r(u,c) & \text{for all } c \in C \text{ and } u \in \Sigma \quad (i) \\ & \displaystyle\sum_{c \in C} x_{i,c} = 1 & \text{for all } i \in \{1, \ldots, n\} \quad (ii) \\ & y_i \geq x_{i,c} - x_{i+1,c} & \text{for all } c \in C \text{ and } i \in \{1, \ldots, n-1\} \quad (iii) \\ & y_i \geq x_{i+1,c} - x_{i,c} & \text{for all } c \in C \text{ and } i \in \{1, \ldots, n-1\} \quad (iv) \\ & x_{i,c} \in \{0,1\} & \text{for all } c \in C \text{ and } i \in \{1, \ldots, n\} \quad (v) \\ & y_i \in \{0,1\} & \text{for all } i \in \{1, \ldots, n-1\}. \quad (vi) \end{array} \tag{1}$$

To solve the necklace splitting problem, we can use this formulation in a branch-and-bound and stop as soon as a solution with at most $(q-1)t$ color changes is found (as always in this paper $q = |C|$ and $t = |\Sigma|$). Moreover, we can use $(q-1)t + 1$ as an upper bound in order to cut a subtree in the branch-and-bound tree even before having found a feasible solution (consequence of Theorem 1).

In the special case of the necklace splitting problem, we face a problem with high symmetries, similar to the ones in the graph coloring or of the bin-packing problems. The following solution is classical in such a situation.

2.2. **Killing symmetries.** Let us consider the necklace splitting problem. Any solution can be encoded (up to symmetry) through $1 \leq j \leq i \leq n$

$$z_{j,i} = \begin{cases} 1 & \text{if } c_j = c_i \text{ and } c_k \neq c_j \text{ for all } k < j \\ 0 & \text{if not.} \end{cases}$$

These variables must satisfy

$$\sum_{i \in P(u) \text{ and } i \geq j} z_{j,i} = a_u z_{j,j} \quad \text{for all } u \in \Sigma \text{ and all } j \in \{1, \ldots, n\}$$

(the requirement is satisfied) and

$$\sum_{j \leq i} z_{j,i} = 1 \quad \text{for all } i \in \{1, \ldots, n\}$$

(each occurrence of a letter receives exactly one color).

An alternative formulation to (1) by an integer linear program for the necklace splitting problem is the following one.

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{n-1} y_i & \\ \text{s.t.} & \displaystyle\sum_{i \in P(u) \text{ and } i \geq j} z_{j,i} = a_u z_{j,j} & \text{for all } u \in \Sigma \text{ and all } j \in \{1, \ldots, n\} \\ & \displaystyle\sum_{j \leq i} z_{j,i} = 1 & \text{for all } i \in \{1, \ldots, n\} \\ & y_i \geq z_{j,i+1} - z_{j,i} & \text{for all } 1 \leq j \leq i+1 \leq n \\ & y_i \geq z_{j,i} - z_{j,i+1} & \text{for all } 1 \leq j \leq i+1 \leq n \\ & z_{j,i} \in \{0,1\} & \text{for } 1 \leq j \leq i \leq n \\ & y_i \in \{0,1\} & \text{for all } i \in \{1, \ldots, n-1\}. \end{array} \tag{2}$$

$z_{j,i}$ is equal to 1 if u_i gets the same color as u_j and if u_j is the first occurrence of a letter getting this color. It is clear that any solution of the necklace splitting problem can be encoded in such a way. More interesting, this formulation has not the drawback of the formulation (1): two solutions identical up to a permutation of C lead to the same solution of (2).

2.3. **Another formulation when there are only two colors.** If there are only two colors, another formulation is possible, namely the following one, which uses less variables than the preceding ones.

Fix one of the colors $c \in C$. The interpretation of the variables is now straightforward.

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i=1}^{n-1} y_i & \\
\text{s.t.} & \displaystyle\sum_{i\in P(u)} x_i = r(u,c) & \text{for all } u \in \Sigma \qquad (i) \\
& y_i \geq x_i - x_{i+1} & \text{for all } i \in \{1,\dots,n-1\} \qquad (ii) \\
& y_i \geq x_{i+1} - x_i & \text{for all } i \in \{1,\dots,n-1\} \qquad (iii) \\
& x_i \in \{0,1\} & \text{for all } i \in \{1,\dots,n\} \qquad (iv) \\
& y_i \in \{0,1\} & \text{for all } i \in \{1,\dots,n-1\}. \qquad (v)
\end{array}
\tag{3}
$$

2.4. **Lagrangian relaxation.** The first formulation (1) suits Lagrangian relaxation. We haven't implemented it. If we relax constraints (iii) and (iv), we get the following Lagrangian subproblem

$$
\sum_{u\in\Sigma} l_u(\boldsymbol{\mu},\boldsymbol{\mu}') + \min_{\boldsymbol{y}\in\{0,1\}^n} \sum_{i=1}^{n-1} \left(1 + \sum_{c\in C} \left(\mu_{i,c} + \mu'_{i,c}\right)\right) y_i,
$$

where $\boldsymbol{y} = (y_1,\dots,y_n)$, $\boldsymbol{\mu} = ((\mu_{i,c}))_{i\in\{1,\dots,n\},c\in C}$ and $\boldsymbol{\mu'} = ((\mu'_{i,c}))_{i\in\{1,\dots,n\},c\in C}$ and $l_u(\boldsymbol{\mu},\boldsymbol{\mu'})$ is the value of

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i\in P(u),c\in C} (\mu_{c,i-1} + \mu'_{c,i-1} - \mu_{c,i} - \mu'_{c,i}) x_{c,i} & \\
\text{s.t.} & \displaystyle\sum_{i\in P(u)} x_{i,c} = r(u,c) & \text{for all } c \in C \text{ and } u \in \Sigma \\
& \displaystyle\sum_{c\in C} x_{i,c} = 1 & \text{for all } i \in P(u) \\
& x_{i,c} \in \{0,1\} & \text{for all } c \in C \text{ and } i \in P(u)
\end{array}
,
$$

which is the value of an optimal $\boldsymbol{b}$-matching in a bipartite graph, computable in strongly polynomial time.

Since for any values of $\boldsymbol{\mu}$ and $\boldsymbol{\mu'}$, the Lagrangian subproblem has integer solutions, it has the so-called *integer property*, and the Lagrangian relaxation does not provide a better bound than the linear relaxation. Anyway, such a relaxation is interesting here since solving the Lagrangian multiplier problem is probably more efficient than solving the linear programming relaxation directly.

3. Necklace splitting through cubical complexes

We come now back to the necklace splitting problem. The problem is the following.

Necklace splitting problem for two thieves

INPUT: A finite alphabet Σ of size t whose elements are called *letters*, a word $\boldsymbol{w} = (u_1, \ldots, u_n) \in \Sigma^*$ in which each letter u is present an even number of times.

TASK: Find a coloring $c = (c_1, \ldots, c_n) \in \{1, 2\}^n$ such that for each $u \in \Sigma$, we have

$$|\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = 1\}| = |\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = 2\}|,$$

and such that the number of color changes is less than or equal to t.

Recall that more poetically the letters are the types of beads, the word $\boldsymbol{w}$ is the sequence of beads on the chain, and that the two colors correspond to the two thieves: for instance, $c_i = 1$ indicates that the ith bead goes to the thief number 1.

The only known algorithmic proof deals with the case $q = 2$ (two thieves) and is given in [16]. We describe in the sequel the algorithm but give first a short overview. The existence of a fair splitting is a consequence of a theorem by Ky Fan, which asserts the existence of a cube with certain labels (a "full" t-cube) in a cubical complex whose vertices get labels satisfying particular conditions (somewhat in the spirit of Sperner's lemma). We first explain this theorem (Subsection 3.1). Then we describe the algorithm finding the full t-cube whose existence is asserted by Ky Fan's theorem (Subsection 3.2). The correctness of the algorithm proves actually Ky Fan's theorem. Then, we explain how to encode the necklace splitting problem in the framework of cubical complexes, labels and Ky Fan's theorem (Subsection 3.3), in order to get an algorithm finding a fair splitting, which appears to be in this context a full t-cube. Finally, we give some details on how to exploit the particular features of the necklace to make quickly the checks and computations required by the algorithm (Subsection 3.4).

3.1. **Cubical complexes and Ky Fan's theorem.** Let C be a cube, ∂C the boundary of C, and $V(C)$ the set of vertices of C. Two facets of an m-cube are said to be *adjacent* if their intersection is a $(m-2)$-cube. For a given facet σ of C, there is only one non-adjacent facet. Two non-adjacent facets are *opposite*.

A *cubical complex* is a collection K of cubes embedded in a Euclidean space such that

- if $\tau \in \mathsf{K}$ and if σ is a face of τ then $\sigma \in \mathsf{K}$ and
- if σ and σ' are in K, then $\sigma \cap \sigma'$ is a face of both σ and σ'.

$V(\mathsf{K})$ is the set of vertices of the cubical complex K.

A cubical complex is said to be *pure* if all maximal cubes with respect to inclusion have same dimension.

A *m-dimensional cubical pseudo-manifold* K is a pure m-dimensional cubical complex in which each $(d-1)$-dimensional cube is contained in at most two m-dimensional cubes. The *boundary* of K, denoted by $\partial\mathsf{K}$, is the set of all $(m-1)$-cubes σ such that σ belongs to exactly one m-dimensional cube of K.

Denote by C^m the cubical complex which is the union of the cube $\square^m := [-1, 1]^m$ and all its faces. $\partial\mathsf{C}^m$, its boundary, is a $(m-1)$-dimensional cubical pseudo-manifold without boundary.

Let $k \geq 0$ be an integer. Given k distinct integers $i_1, i_2, \ldots, i_k$, each of them in $\{1, 2, \ldots, m\}$, and k numbers $\epsilon_1, \epsilon_2, \ldots, \epsilon_k$, each of them belonging to $\{-1, +1\}$, we denote

$$F\begin{pmatrix} i_1 & i_2 & \ldots & i_k \\ \epsilon_1 & \epsilon_2 & \ldots & \epsilon_k \end{pmatrix} := \{(x_1, x_2, \ldots, x_m) \in \{-1, +1\}^m : x_{i_j} = \epsilon_j \text{ for } j = 1, 2, \ldots, k\},$$

which is the set of vertices of an $(m-k)$-face of $\square^m$. For $k = 0$, we get all vertices of $\square^m$.

Let $n, t \geq 1$ be integers and M be the cubical pseudo-manifold without boundary obtained by subdividing $\partial\mathsf{C}^{t+1}$ by means of the hyperplanes $H_{i,j} = \{\boldsymbol{x} \in \mathbb{R}^{t+1} : x_i = \frac{j}{n} \text{ where } i \in \{1, 2, \ldots, t+1\}$

and $j \in \{-n+1, -n+2, \ldots, -1, 0, 1, \ldots, n-2, n-1\}$. Let $\lambda : V(\mathsf{M}) \to V(\mathsf{C}^m)$ be a labelling of its vertices with vertices of C^m such that the labels of two adjacent vertices are either identical or neighbors on the 1-skeleton of C^m. A d-cube of K (with $0 \leq d \leq m$) is said to be *full* if the image of its vertices under λ is of the form

$$F\begin{pmatrix} d+1 & d+2 & \ldots & m \\ \epsilon_1 & \epsilon_2 & \ldots & \epsilon_{m-d} \end{pmatrix}.$$

We have then the following theorem by Ky Fan (in a specialized version of Theorem 5 in [12]).

Theorem 9 (Ky Fan's theorem)**.** *If λ is antipodal (i.e. such that $\lambda(-v) = -\lambda(v)$ for all vertices $v \in \mathsf{M}$), there exists always at least one full t-cube.*

We explain in Subsection 3.3 how the necklace splitting problem can be embedded in this framework. The vertices of M will encode the different splittings. The labelling λ of a vertex will indicate the thief advantaged for each type of bead, for the splitting encoded by the vertex. The existence of a full t-cube will imply the existence of a fair splitting.

3.2. **An algorithm finding a full t-cube.** We endow M with *hemispheres*. These hemispheres are denoted $+H_0$, $-H_0$, $+H_1$, $-H_1$, ..., $+H_t$, $-H_t$. $+H_i$ (resp. $-H_i$) is the set of points $(x_1, x_2, \ldots, x_{t+1})$ of $\partial\square^{t+1}$ such that $x_{i+1} \geq 0$ (resp. $x_{i+1} \leq 0$), and such that, if $i \leq t-1$, then $x_j = 0$ for $j > i+1$. Note that the boundary of an hemisphere of dimension i is the union of the two hemispheres of dimension $i-1$.

The *sign* of the hemisphere $+H_i$ (resp. $-H_i$) is $+1$ (resp. -1) and we denote it by $\mathrm{sign}(+H_i) = +1$ and $\mathrm{sign}(-H_i) = -1$.

On Figure 1, the hemispheres of M obtained by subdividing $\partial\mathsf{C}^3$ for $n = 2$ are highlighted.

The *carrier hemisphere* $\mathrm{CarrHem}(\sigma)$ of a cube σ is the hemisphere of smallest possible dimension containing σ.

Now, assume that we have a labelling λ of M for some $m \geq t$ and assume moreover that λ is *antipodal*, that is that $-\lambda(v) = \lambda(-v)$ for any vertex $v \in V(\mathsf{M})$. We define the notion of *sign* for a full d-cube σ with $d < m$: let

$$F\begin{pmatrix} d+1 & d+2 & \ldots & m \\ \epsilon_1 & \epsilon_2 & \ldots & \epsilon_{m-d} \end{pmatrix}$$

be the image of its vertices. ϵ_1 is then its *sign*, denoted by $\mathrm{sign}(\sigma)$.

A d-cube is said to be *almost-full* if it is not full but one of its facets is full. Lemma 3.2 of [16] shows that all full facets of an almost-full cube have always the same sign, which allows to define the sign $\mathrm{sign}(\sigma)$ of an almost-full d-cube σ to be the common sign of its full facets.

In [16], the following method for finding a full t-cube is described. Consider the following graph $G = (N, E)$. The nodes of G are the cubes τ in one of these three situations.

Situation (1): τ is full, $\dim\tau = \dim(\mathrm{CarrHem}(\tau)) - 1$ and $\mathrm{sign}(\tau) = \mathrm{sign}(\mathrm{CarrHem}(\tau))$
Situation (2): τ is almost-full, $\dim\tau = \dim(\mathrm{CarrHem}(\tau))$ and $\mathrm{sign}(\tau) = \mathrm{sign}(\mathrm{CarrHem}(\tau))$
Situation (3): τ is full and $\dim\tau = \dim(\mathrm{CarrHem}(\tau))$

Two nodes τ and τ' are adjacent in G if all the following hold:

(a) τ' is a facet of τ,
(b) τ' is full,
(c) $\dim\tau = \dim(\mathrm{CarrHem}(\tau))$, and
(d) $\mathrm{sign}(\mathrm{CarrHem}(\tau)) = \mathrm{sign}(\tau')$.

It can be checked that all nodes have degree 1, 2 or 4. A cube has degree 1 only if its dimension is 0 (it is either the 0-cube $(1, 0, \ldots, 0)$ or $(-1, 0, \ldots, 0)$, no other 0-cube being contained in $+H_0$ or $-H_0$) or if it is a full t-cube. It can also be proved that in a connected component of G, there is

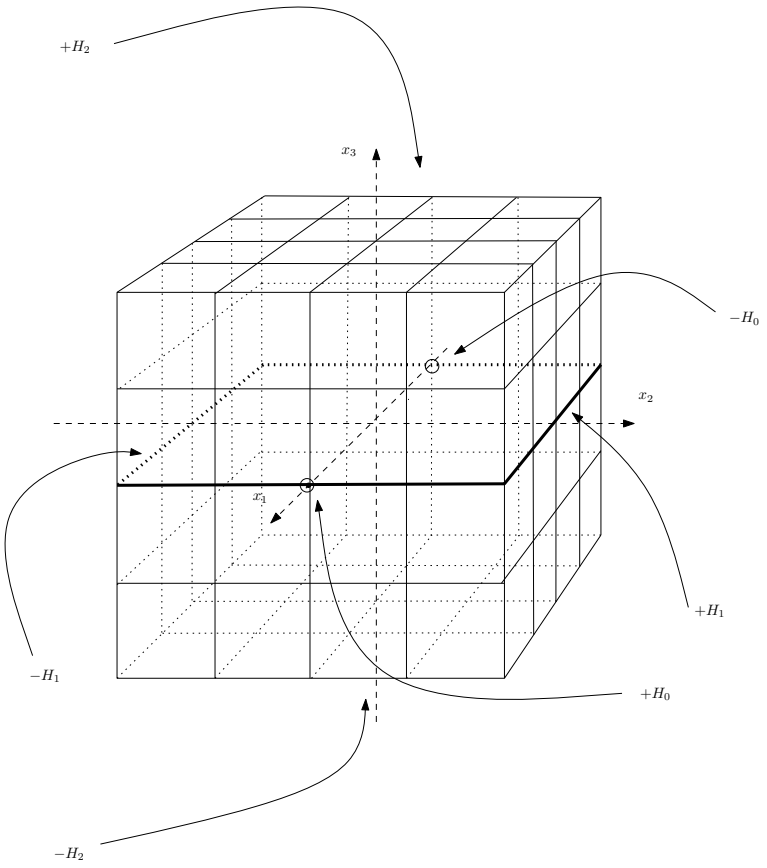

FIGURE 1. An illustration of M for $n = 2$ and $t = 2$.

at most one 0-cube node. If a cube τ is of degree 4, it can be proved that we are in Situation (2) and that it has exactly 4 full facets, two pairs of opposite facets. We make then two copies of τ, each of them linked to one of these pairs. G becomes a collection of cycles and paths. Following the path starting at $(1, 0, \ldots, 0)$, we eventually arrive at a full t-cube (and prove Theorem 9).

We get Algorithm 1. It is a path-following algorithm, similar to Scarf's algorithm computing fixed point for continuous mapping [23]. In the pseudocode Algorithm 1, τ is the current cube and σ the current cube of the previous iteration. Note that σ and τ are such that one is always the facet of the other and that the maximal one has always the dimension of its carrier hemisphere. The dimension of the current carrier hemisphere decreases when the next cube σ' is a facet of τ lying on the boundary of the carrier hemisphere of τ; it can happen in Situation (2) and also in Situation (3) when τ is a facet of σ. The dimension of the current hemisphere increases by one precisely when we are in Situation (3) with σ a facet of τ.

Since it is very far from the exhaustive enumeration of the t-cubes of M, we may hope that a similar phenomenon as for Scarf's algorithm happens, that provides a fast algorithm to find full t-cubes. The experimentation is made in Section 5.

3.3. **Application to the necklace.** The necklace is identified with the interval $]0, n[\subset \mathbb{R}$. The beads are numbered with the integers $1, 2, \ldots, n$, from left to right. The kth bead occupies uniformly the interval $]k-1, k[$.

For $S \subseteq]0, n[$, we denote by $\phi_i(S)$ the quantity of beads of type i in positions included in S. More precisely, we denote by $\mathbf{1}_i(u)$ the mapping which indicates if there is a bead of type i at point

Algorithm 1 The algorithm finding a full t-cube whose existence is claimed by Ky Fan's theorem.

$\sigma \leftarrow (1, 0, \ldots, 0)$
let τ be such that σ is a facet of τ and $\mathrm{CarrHem}(\tau) = \mathrm{sign}(\sigma)H_1$ // τ is an edge in $\pm H_1$
while τ is not full **or** $\dim \tau \neq t$ **do**
  if τ is full **and** $\dim \tau = \dim(\mathrm{CarrHem}(\tau)) - 1$ **then**
    // Situation (1)
    let $\tau' \in \mathrm{CarrHem}(\tau)$ be such that τ is a facet of τ' and $\tau' \neq \sigma$
  else if τ is almost-full **then**
    // Situation (2)
    let τ' be the facet of τ such that $\sigma \cap \tau' = \emptyset$ // σ and τ' are opposite
  else if τ is full **and** $\dim(\mathrm{CarrHem}(\tau)) = \dim \tau$ **then**
    // Situation (3)
    $d \leftarrow \dim \tau$
    if σ is a facet of τ **then**
      //the dimension of the carrier hemisphere is going to increase
      let τ' be such that τ is a facet of τ' and $\mathrm{CarrHem}(\tau') = \mathrm{sign}(\tau)H_{d+1}$
    else
      //the dimension of the carrier hemisphere is going to decrease
      let τ' be such that τ' is full, is a facet of τ and $\mathrm{sign}(\tau') = \mathrm{sign}(\mathrm{CarrHem}(\tau))$
    end if
  end if
  $\sigma \leftarrow \tau$
  $\tau \leftarrow \tau'$
end while
return τ

u of $]0, n[$:

$$B_i(u) = \begin{cases} 2 \text{ if the } \lceil u \rceil \text{th bead is of type } i \text{ and if it is the first bead of this type.} \\ 1 \text{ if the } \lceil u \rceil \text{th bead is of type } i \text{ and if it is not the first bead of this type.} \\ 0 \text{ if the } \lceil u \rceil \text{th bead is not of type } i. \end{cases}$$

We have then

$$\phi_i(S) := \int_S B_i(u)du.$$

The definition of B_i may be surprising since the first bead of each type is counted twice; this is only a trick to avoid tie between the two thieves.

Let $x := (x_1, x_2, \ldots, x_{t+1})$ be a vertex of M. We order the x_i in the increasing order of their absolute value. When $|x_i| = |x_{i'}|$, we put x_i before $x_{i'}$ if $i < i'$. This gives a permutation π such that

$$|x_{\pi(1)}| \leq |x_{\pi(2)}| \leq \ldots \leq |x_{\pi(t+1)}| = 1.$$

It encodes a cut as follows: the necklace is cut at points $n|x_i|$. Since among the $|x_i|$, at most t are different from 1 (we are in M, which is embedded in $\partial \mathsf{C}^{t+1}$), we get a splitting in at most t cuts, as required. From left to right, we rank the subnecklaces. Given a subnecklace, we look at the smallest k such that $n|x_k|$ is greater than or equal to the coordinate of any point of the subnecklace (i.e the cut with smallest index at the right of the subnecklace). The sign of x_k indicates the thief who gets this subnecklace: if it is $+$, it goes to the first thief; it is $-$, it goes to the second one.

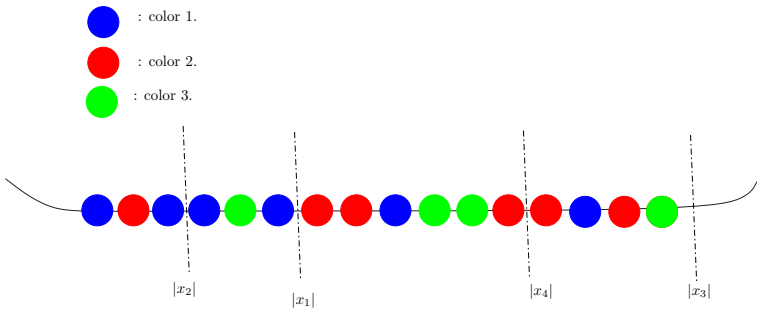

FIGURE 2. The cuts associated to the vertex $x_1 = \frac{6}{16}$, $x_2 = -\frac{3}{16}$, $x_3 = -1$, $x_4 = -\frac{12}{16}$.

According to this assignment rule, we define $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_t) : V(\mathsf{M}) \to \{-1, +1\}^t = V(\mathsf{C}^t)$ as follows: we cut the necklace at points $n|x_i|$. By convention, $x_{\pi(0)} := 0$; we define

$$\lambda_i(x) := \begin{cases} +1 \text{ if } \sum_{j=0}^{t} \operatorname{sign}(x_{\epsilon(j)})\phi_i\left(\left]n|x_{\pi(j)}|, n|x_{\pi(j+1)}|\right[\right) > 0 \\ -1 \text{ if not,} \end{cases}$$

for $i = 1, 2, \ldots, t$, where $\epsilon(j)$ is the smallest integer k such that $|x_k| \geq |x_{\pi(j+1)}|$, and with the convention $\operatorname{sign}(0) := 0$.

This formula can be understood as follows: $\lambda_i(x)$ equals $+1$ (resp. -1) if the first thief gets strictly more (resp. strictly less) beads of type i than the second one, when we cut the necklace at points $n|x_j|$, and assign the subnecklaces to the thieves according to the rule described in the previous paragraph.

Let us give an example. Figure 2 shows a necklace with $n = 16$, $t = 3$, and 4 cuts. These 4 cuts are compatible for instance with the vertex (x_1, x_2, x_3, x_4) where

$$x_1 = \frac{6}{16},\ x_2 = -\frac{3}{16},\ x_3 = -1,\ x_4 = -\frac{12}{16}.$$

There are 4 subnecklaces. The first subnecklace goes to the first thief since $\frac{6}{16} \geq \left|-\frac{3}{16}\right|$ (the minimal index k such that $16|x_k|$ is greater than all points of this subnecklace is 1) and $x_1 > 0$. The second subnecklace goes also to the first thief exactly for the same reasons. The third subnecklace goes to the second thief since the minimal index k such that $16|x_k|$ is greater than all points of both subnecklaces is 3 and $x_3 < 0$. The fourth one goes also to the second thief, for the same reasons. The image of this vertex through λ is $(+1, -1, -1)$ where the first component corresponds to the blue beads (color 1), the second one to the red beads (color 2), and the third one to the green beads (color 3).

It can be checked that λ maps adjacent vertices to the same vertex or to adjacent vertices, and that it is antipodal. We are thus in the framework of Subsection 3.1 and Theorem 9 can be applied. A full d-cube σ is obtained by d cuts sliding on d beads and since it is full, each of these cuts "slides" along distinct beads of distinct types, from type 1 to type d. More precisely, take the splitting encoded by the vertex of σ with the highest (resp. lowest) coordinates : the first thief (resp. the second thief) is advantaged for all the first d types of beads.

A full t-cube having necessarily a vertex encoding a fair splitting (changing the assignment of the type i bead changes the thief advantaged for type i), Algorithm 1 solves the necklace splitting problem. Nevertheless, we have *a priori* no guarantee on the time needed to solve a real problem. In Section 5, this algorithm is experimented.

3.4. **Some details of implementation.**

3.4.1. *Cubes.* A cube σ of $\partial\mathsf{M}$ is encoded through a couple (`LowerPoint`,`UpperPoint`) where `LowerPoint` (resp. `UpperPoint`) is the coordinate of the vertex having the lowest (resp. highest) values for each coordinate. Note that there are no ambiguity and that the dimension of σ is the number of coordinates that differ between `LowerPoint` and `UpperPoint`.

3.4.2. *Full cubes.* To check if a d-cube is full, it is enough to check if the first thief is advantaged for each of the first d types by the splitting encoded by `UpperPoint` and if the second thief is advantaged for each of the first d types by the splitting encoded by `LowerPoint`.

3.4.3. *Sign.* The sign of a d-cube is given by the thief advantaged for type $d+1$.

3.4.4. *Carrier hemisphere.* The carrier hemisphere of a vertex is given by the rank and the sign of its last nonzero coordinate.

4. Local searches

We have developed two local search models, one for the paintshop problem, and one for the necklace splitting.

4.1. **Paintshop moves.** For the general paint shop problem, the objective to minimize is the number of color changes. The initial configuration is an admissible coloring $(c_1, \ldots, c_n) \in C^n$ such that it satisfies the coloring constraints, i.e.

$$|\{i \in \{1, \ldots, n\} : u_i = u \text{ and } c_i = c\}| = r(u, c) \quad \text{for all } u \text{ and } c.$$

The local search algorithm performs swap moves in the sequence between two positions $i_1 < i_2$ such that

- $u_{i_1} = u_{i_2}$ and
- $c_{i_1} \neq c_{i_2}$.

In order to intensify the search, i_1 is chosen just before or after a color change. We have used the same moves for the binary paintshop problem.

4.2. **Necklace splitting moves.** The local search designed for the paint shop version can be applied to the necklace version – the search must be stopped as soon as the number of cuts becomes less than or equal to $(q-1)t$. It appears that in the special case of the necklace, we can design another local search.

For the necklace splitting problem, the objective is to find a fair division between q thieves, with the number of cuts $N = (q-1)t$ given by the theorem 1. Remark that this number of cuts is not always the minimum.

The model is made of $2*N$ variables, i.e for each cut we have 2 variables, the place (in $\{1, \ldots, n\}$) of the cut and the thief who receives the part just after the cut. For breaking symmetries, we give the first part to the first thief. The initial configuration is made of N random cuts and each part is randomly assigned to a thief. The objective function to minimize is the sum of the deviations for each type and thief couple between the current division and a fair one. The algorithm stops when a fair division is found, or when a timeout is reached.

The local search algorithm performs 2 types of moves :

- Displacement of a cut : a cut is chosen, and is displaced to a new place which is distant of less than D places from the current place.
- Change of thief : a part is chosen and given to another thief, distinct from the thieves that own the next left and right parts, in order to keep constant the number of cuts.

We wanted to perform small moves, that can be efficiently incrementally evaluated. We limited therefore each cut displacement by a maximum distance of $D = 20$ places to the left or to the right of the current place. We experimentally fixed the ratio between the 2 types of moves (displacements and changes of thief) to 4, i.e we try with a probability of 0.8 a displacement move and a probability of 0.2 a change of thief.

We have implemented a simplified variant for 2 thieves, where the only variables are the N cuts places and a single type of moves, the cut displacements (also limited with a maximum distance equal to 20).

4.3. **Metaheuristic.** For both problems, the metaheuristic used for escaping local minima is IDWalk [20]. It is a very simple metaheuristic, based on the size of the explored neighborhood. Like all local search algorithms, the algorithm changes a current configuration by performing at each step a local transformation, i.e. a local move as described above. At the end of the walk, it returns the best configuration found during the walk.

The pseudo-code described in Figure 2 shows in details the main step of the algorithm : how the neighbor that will become the next current configuration is selected. IDWalkStep randomly performs an incomplete exploration of the neighborhood of the current configuration. It selects the first improving or equal neighbor found if any, and in case all `MaxNeighbors` visited neighbors degrade the objective function, it selects the less degrading one. The parameter `MaxNeighbors` of the algorithm is the size of the randomly explored neighborhood. This parameter controls the behavior of the algorithm. If the `MaxNeighbors` potentially explored neighbors represent an important part of the neighborhood, the algorithm will perform more *intensification* than if `MaxNeighbors` is small w.r.t. the size of the neighborhood.

In addition, the *diversification* performed by the algorithm also mainly depends on this parameter. If `MaxNeighbors` is small, it is not likely to find a configuration better than or equal to the current one among these explored neighbors. The degradation of the objective obtained then by choosing the less degrading explored neighbor will be greater than if a larger number of neighbors would be tested. With a small part of the neighborhood being explored, it is also not likely that the algorithm returns in the following moves to an already visited configuration and the presence of a tabu list device becomes unnecessary.

For each instance to be solved, this `MaxNeighbors` parameter is automatically tuned. We alternate tuning and running phases. During the tuning phases, the algorithm is run on small walks with different parameter values. The parameter value that yields the best result is selected and used during the next running phase.

5. Computational results

The algorithms described in the previous sections were implemented in C++. The implementations were tested on a computer with a Intel Core 2 Duo CPU 2.26 GHz processor, with 1.96 GB RAM, using the GNU-C++ compiler (gcc 3.4.5). For the linear programming resolution, we have used CPLEX 12.2. The local search uses the version of IDWalk implemented in the `INCOP` [19] C++ local search library.

The following notations are used.

- LP linear programming.
- LSp local search designed for the paintshop version (Subsection 4.1).
- LSn local search designed for the necklace version (Subsection 4.2).
- n the length of the word $\boldsymbol{w}$,
- t the cardinality of Σ and
- q the number of colors

Algorithm 2 `IDWalkStep` (x: current configuration, *best*: best configuration, `MaxNeighbors`: maximum number of neighbors to be visited)

```
Candidate ← 1
RejectedCandidates ← ∅
Accepted? ← false
while (Candidate ≤ MaxNeighbors) and (not Accepted?) do
  Select randomly a neighbor x' of x
  if cost(x') ≤ cost(x) then
    Accepted? ← true
  else
    RejectedCandidates ← RejectedCandidates ∪ {x'}
  end if
  Candidate ← Candidate +1
end while
if Accepted? then
  x ← x'
else
  x ← the configuration in RejectedCandidates with the smallest cost
end if
if cost(x) < cost(best) then
  best ← x
end if
```

All instances were generated at random, since no benchmark is available (as noted in the Introduction, this problem had not been studied from a practical point of view). They can be downloaded from

http://cermics.enpc.fr/∼meuniefr/PaintShopNecklaces_Instances

5.1. **General paint shop problem.** The approaches tested for the (general) paint shop problem are linear programming (Subsection 2.1) and the local search (Subsection 4.1). We have generated 15 instances, as follows. The parameters n, q, t are fixed. The ith letter is an element drawn uniformly at random from Σ, independently from the other letters. Each time a letter is drawn, a color is also drawn, therefore at the end we have for each letter a pool of colors to be attributed to its occurrences.

For the local search method, as it is a stochastic incomplete method depending on the initial configuration, we performed 2 sequences of 10 tries per instance, each try being limited to 1 min CPU time for the first sequence, and to 5 min CPU time for the second sequence.

We report the best and the median solutions among these 10 tries.

The linear programming was experimented with a time limit equal to 30 min CPU time.

The results are given in Table 1.

Linear programming is able to solve optimally the general paint shop problem when the number of types and the number of colors are quite small (≤ 5). As soon as these parameters grow, the gap between the upper and lower bounds provided by the linear program becomes large. The local search behaves well, even if we were not able to give a reasonable lower bound to estimate the gap with the optima. The feasible solutions found by the local search are anyway largely better than the ones obtained by the linear program, and in a laps of time really smaller.

Instance	n	q	t	LP (1) 30 min Solution	Lower Bound	LSp 60 s best sol	LSp 60 s median sol	LSp 300 s best sol	LSp 300 s median sol
PS_200_1	200	3	4	7* (15 s)	7	7	7	7	7
PS_200_2	200	5	5	14	9	12	13	12	12
PS_200_3	200	5	10	24	17	21	21	21	21
PS_200_4	200	10	5	33	7	20	21	20	20
PS_200_5	200	10	10	43	18	32	33	32	32
PS_500_1	500	3	4	5* (3 min 42 s)	5	6	6	6	6
PS_500_2	500	5	5	16	9	17	18	15	17
PS_500_3	500	5	10	27	13	25	27	25	26
PS_500_4	500	10	5	68	7	32	33	29	32
PS_500_5	500	10	10	109	14	44	47	41	44
PS_1000_1	1000	3	4	5* (18 min 35 s)	5	8	10	7	8
PS_1000_2	1000	5	5	29	6	20	22	18	20
PS_1000_3	1000	5	10	58	12	37	38	33	35
PS_1000_4	1000	10	5	129	7	40	44	41	43
PS_1000_5	1000	10	10	211	15	65	67	58	62

TABLE 1. Results for the general paint shop problem.

Instance	n	q	t	LP (1) 30 min Sol.	LP (1) 30 min Lower Bound	LP (2) 30 min Sol.	LP (2) 30 min Lower Bound	LP (3) 30 min Sol.	LP (3) 30 min Lower Bound	LSp 60 s Best Sol.	LSp 60 s Median Sol.	LSp 300 s Best Sol.	LSp 300 s Median Sol.	Evenly Laminar Bound
BPS_250	250	2	125	35*	35	35*	28	35*	35	35	35	35	35	19
BPS_500	500	2	250	70	52	133	18	71	51	70	71	70	71	29
BPS_1000	1000	2	500	143	71	-	-	144	73	135	135	135	135	36
BPS_2000	2000	2	1000	365	105	-	-	292	105	270	277	270	272	59
BPS_5000	5000	2	2500	895	162	-	-	892	160	685	691	686	687	91

TABLE 2. Results for the binary paint shop problem.

5.2. **Binary paint shop problem.** The approaches tested are the three formulations of linear programming, (1), (2) and (3), as well as local search.

We have generated instances, where a sequence of n letters with 2 occurrences of each letter is randomly generated. The objective is to color this sequence with 2 colors, with a minimum of color changes.

The results are given in Table 2.

Note that LP (2) behaves very badly – the '-' means that CPLEX terminates because of an 'out of memory' error. The reason is probably that it has too many variables.

The binary paintshop problem also proves the superiority of the local search to find good feasible solutions, and again more quickly than linear programming. We can anyway note that the linear programming approach (LP (1) and (3)) is efficient up to $n \simeq 250$ and the instances of this size are solved to optimality in less than 30 minutes. The efficiency of the linear programming approach was already noted in Subsection 1.2, in which it was tested on instances of size $n \leq 240$.

The last column shows the size of the largest evenly laminar subset, which provides a lower bound according to Proposition 2. We see that there is a large gap with the optimal solution, as claimed at the end of Subsection 1.2. Even if it is better than the linear relaxation, which should provide a 0 bound at the root of the branch-and-bound, CPLEX adds sufficiently many cuts to be largely more efficient.

Instance	n	q	t	LP (1) Time	LSn Succ.	LSn Time (in s)	LSp Succ.	LSp Time (in s)
N20_3_5_1	300	3	5	7.42 s	10	0.15	10	0.10
N20_3_5_2	300	3	5	14.93 s	8	0.09	10	0.09
N20_3_5_3	300	3	5	3.14 s	9	0.03	10	0.05
N20_3_5_4	300	3	5	7.40 s	10	0.11	10	0.05
N20_3_5_5	300	3	5	15.07 s	9	0.02	10	0.03
N20_4_5_1	400	4	5	2 min 17 s	10	0.23	10	0.87
N20_4_5_2	400	4	5	4 min 52 s	10	0.18	10	0.60
N20_4_5_3	400	4	5	3 min 20 s	10	0.10	10	0.46
N20_4_5_4	400	4	5	35.79 s	10	0.04	10	0.32
N20_4_5_5	400	4	5	2 min 23 s	10	0.06	10	0.36
N20_5_5_1	500	5	5	–	10	0.69	10	2.37
N20_5_5_2	500	5	5	15 min 03 s	10	0.18	10	1.05
N20_5_5_3	500	5	5	–	10	0.41	10	1.38
N20_5_5_4	500	5	5	36 min 15 s	10	0.63	10	1.16
N20_5_5_5	500	5	5	–	10	0.87	10	1.88

TABLE 3. Results for the necklace splitting problem with $q \geq 3$ – small instances.

5.3. **Necklace splitting.** When there are strictly more than two thieves, we have tested the two formulations (1) and (3) of the linear programming and the two local search methods LSn and LSp. (2) was not able to solve any instance, whence we have omitted its results in the table. When there are exactly two thieves, we have added the path-following method of Section 3.

The instances were generated uniformly at random among the instances with the parameters n, q and t fixed, and with the same number of beads of each type. We have generated 15 "small" and 15 "large" instances with $q \geq 3$ and also 15 "small" and 25 "large" instances with $q = 2$.

For the linear programming approaches, we have initialized the branch-and-bound with an upper bound $(q-1)t + 0.1$ since by the splitting necklace theorem (Theorem 1), we know that there is a solution strictly better than this value, and we stop the branch-and-bound when the first integer solution is found (in CPLEX, there is a so-called "MIP integer solution limit").

For the LSn local search method, as described in previous section, we use a model where the number of cuts is fixed to $(q-1)t$ cuts. This method stops with success when a fair division is found. The LSp local search method works with fair divisions and tries to lower the number of cuts. It stops with success when a solution with $(q-1)t$ or less cuts is found.

The results are given in Tables 3, 4, 5 and 6. For the local search methods, we performed 10 tries per instance, each try being limited to 1 min . CPU time. We report for each instance, the number of successes and the average time for a successful try.

When there are 3 or more thieves (see Tables 3 and 4) the local search – especially LSn – is efficient, and the linear programming approach hardly finds a solution for relative small instances ($n \simeq 400$). For large instances (Table 4), linear programming and LSp were both unable to solve any of them in the time limits.

We can remark – see Table 6 – that for the problem with 2 thieves and 3 types of beads, the cubical path-following method is more efficient than the local search, in the sense that it is a complete method that always finds a solution in a time comparable with the time of a successful

Instance	n	q	t	LSn Success	LSn Time (in s)
N200_3_5_1	3000	3	5	5	2.4
N200_3_5_2	3000	3	5	7	1.6
N200_3_5_3	3000	3	5	8	8.9
N200_3_5_4	3000	3	5	6	3.6
N200_3_5_5	3000	3	5	6	5.5
N200_4_5_1	4000	4	5	5	14.3
N200_4_5_2	4000	4	5	8	11.9
N200_4_5_3	4000	4	5	4	6.9
N200_4_5_4	4000	4	5	5	12.7
N200_4_5_5	4000	4	5	3	33.5
N200_5_5_1	5000	5	5	2	34.3
N200_5_5_2	5000	5	5	1	35.2
N200_5_5_3	5000	5	5	3	15.2
N200_5_5_4	5000	5	5	1	11.5
N200_5_5_5	5000	5	5	3	34.9

TABLE 4. Results for the necklace splitting problem with $q \geq 3$ – large instances.

Instance	n	q	t	Cubical path-following Time	LP (1) Time	LP (3) Time	LSn Succ.	LSn Time (s)	LSp Succ.	LSp Time (s)
N1000_2_3_1	6000	2	3	0.28 s	13 min 54 s	10 min 10 s	9	0.18	0	–
N1000_2_3_2	6000	2	3	0.22 s	13 min 50 s	12 min 18 s	10	1.23	7	25.48
N1000_2_3_3	6000	2	3	0.63 s	5 min 09 s	13 min 19 s	10	0.87	2	19.74
N1000_2_3_4	6000	2	3	1.39 s	5 min 46 s	21 min 40 s	9	1.18	0	–
N1000_2_3_5	6000	2	3	1.31 s	2 min 47 s	10 min 53 s	10	5.30	0	–
N100_2_10_1	2000	2	10	4 min 31 s	3 min 03 s	2 min 02 s	10	0.11	0	–
N100_2_10_2	2000	2	10	25 s	2 min 33 s	2 min 13 s	10	0.10	1	15.80
N100_2_10_3	2000	2	10	14 s	2 min 00 s	2 min 15 s	10	0.14	0	–
N100_2_10_4	2000	2	10	1 min 11 s	1 min 08 s	1 min 27 s	10	0.17	1	3.03
N100_2_10_5	2000	2	10	1 min 45 s	2 min 00 s	2 min 35 s	10	0.08	0	–
N20_2_20_1	800	2	20	3 min 22 s	12 s	1 s	10	0.10	10	0.15
N20_2_20_2	800	2	20	3 min 07 s	25 s	6 s	10	0.12	10	0.09
N20_2_20_3	800	2	20	1 min 45 s	6 s	9 s	10	0.11	10	0.36
N20_2_20_4	800	2	20	1 min 59 s	1 s	6 s	10	0.05	10	0.78
N20_2_20_5	800	2	20	7 min 57 s	6 s	4 s	10	0.25	10	0.11

TABLE 5. Results for the necklace splitting problem with $q = 2$ – small instances.

execution of the local search method. This latter method sometimes fails, depending on the initial configuration. On the contrary, when the number of types grows, the cubical path-following method becomes less efficient than the local search method.

We can also note – see Table 5 – that the linear programming approach and LSp work better when the number of cuts $(q-1)t$ ensured by the necklace theorem becomes high. This can be explained by the strategy followed by these approaches, which consists in decreasing the number of cuts until reaching the target number.

Instance	n	q	t	Cubical path-following Time (in s)	LSn Success	LSn Time (in s)
N5000_2_3_1	30000	2	3	1.1	7	6.2
N5000_2_3_2	30000	2	3	4.8	7	1.9
N5000_2_3_3	30000	2	3	1.2	9	2.8
N5000_2_3_4	30000	2	3	2.7	7	2.5
N5000_2_3_5	30000	2	3	2.7	9	5.8
N10000_2_3_1	60000	2	3	5.4	5	7.0
N10000_2_3_2	60000	2	3	5.4	2	9.3
N10000_2_3_3	60000	2	3	4.9	1	27.1
N10000_2_3_4	60000	2	3	2.8	4	2.5
N10000_2_3_5	60000	2	3	3.4	8	1.4
N15000_2_3_1	90000	2	3	12.6	4	3.7
N15000_2_3_2	90000	2	3	3.0	0	–
N15000_2_3_3	90000	2	3	4.4	8	5.8
N15000_2_3_4	90000	2	3	4.3	2	3.4
N15000_2_3_5	90000	2	3	4.3	3	12.3
N1000_2_5_1	10000	2	5	7.36	10	1.9
N1000_2_5_2	10000	2	5	39.0	9	0.2
N1000_2_5_3	10000	2	5	9.72	10	9.4
N1000_2_5_4	10000	2	5	36.7	10	0.8
N1000_2_5_5	10000	2	5	620	10	1.5
N2000_2_5_1	20000	2	5	92.3	8	2.1
N2000_2_5_2	20000	2	5	259	8	6.8
N2000_2_5_3	20000	2	5	47	9	1.1
N2000_2_5_4	20000	2	5	143	6	1.5
N2000_2_5_5	20000	2	5	231	9	1.5

TABLE 6. Results for the necklace splitting problem with $q = 2$ – large instances.

5.4. **Conclusion.** The two local searches – when applied to the problems they have been designed for – are really efficient. Exact approaches exist when the size of the word (the total number of beads on the necklace) is small (linear programming) for both the necklace problem and the paintshop problem, and for the necklace problem with two thieves and not too many types, say ≤ 5 (the path-following method). It is worth noting that the path-following method is very efficient for the splitting of the necklace between two thieves, when there are only 3 types of beads, the solution being computed in few seconds even for 90000 beads.

6. Some concluding remarks

On the theoretical hand, many questions remain open both for the paint shop and the necklace versions – see Section 1. These questions come within complexity theory, discrete probability, combinatorial optimization and combinatorial topology. So many areas meet rarely in the same problem. On the practical hand, we know how to solve the paintshop-necklace problem for high instances with local searches, and we propose approaches through linear programming and a path-following method. An efficient exact method for the paintshop problem remains to be designed, as well for the necklace problem for three thieves and more.

Finally, let us emphasize that the path following method appears to be quite efficient. Actually, it was designed to solve the problem of finding a full t-cube (see Section 3), which is more general than the splitting of the necklace. The existence of a full t-cube is very close to Tucker's lemma, the combinatorial counterpart of the Borsuk-Ulam theorem. One may ask whether this method would be efficient for the computation of the zeroes of an antipodal map $f : \mathcal{S}^d \to \mathbb{R}^d$, whose existence is ensured by the Borsuk-Ulam theorem. However, a step is missing, namely the existence of a "cubical approximation theorem" which is a still open question: any continuous mapping can be approximated by a simplicial map; it is not known if a similar statement holds for a cubical map (see for instance Property 6.1 in [7]). Since cubical complexes are often more tractable than simplicial complexes a positive answer to this latter question would provide a promising approach to the computation of zeroes of antipodal maps.

References

1. N. Alon, *Splitting necklaces*, Adv. in Math. **63** (1987), 247–253.
2. ———, *Non-constructive proofs in combinatorics*, Proc. of the International Congress of Mathematicians, Kyoto 1990, Japan, Springer Verlag, Tokyo, 1991, pp. 1421–1429.
3. N. Alon, D. Moshkovitz, and S. Safra, *Algorithmic construction of sets for k-restrictions*, ACM Trans. Algorithms **2** (2006), 153–177.
4. N. Alon and D. West, *The Borsuk-Ulam theorem and bisection of necklaces*, Proc. Amer. Math. Soc. **98** (1986), 623–628.
5. H. Amini, F. Meunier, H. Michel, and A. Mohajeri, *Greedy colouring of the binary paintshop problem*, Journal of Discrete Algorithms **8** (2010), 8–14.
6. S. D. Andres and W. Hochstättler, *Some heuristics for the binary paintshop problem and their expected number of colour changes*, Journal of Discrete Algorithms (2010).
7. E. Babson, H. Barcelo, M. de Longueville, and R. Laubenbacher, *A homotopy theory for graphs*, Journal of Algebraic Combinatorics **24** (2006), 31–44.
8. S. N. Bhatt and C. E. Leiserson, *How to assemble tree machines*, Proc. of the 14th Symposium on the Theory of Computing, San Francisco, 1981, USA, 1981, pp. 99–104.
9. P. S. Bonsma, T. Epping, and W. Hochstättler, *Complexity results on restricted instances of a paint shop problem for words*, Discrete Appl. Math. **154** (2006), 1335–1343.
10. M. de Longueville and R. Zivaljevic, *The Borsuk-Ulam-property, Tucker-property and constructive proofs in combinatorics*, Journal of Combinatorial Theory, Ser. A **113** (2006), 839–850.
11. T. Epping, W. Hochstättler, and P. Oertel, *Complexity results on a paint shop problem*, Discrete Appl. Math. **136** (2004), 217–226.
12. K. Fan, *Combinatorial properties of certain simplicial and cubical vertex maps*, Arch. Math. **11** (1960), 368–377.
13. C. H. Goldberg and D. West, *Bisection of circle colorings*, SIAM J. Algebraic Discrete Methods **6** (1985), 93–106.
14. J. Matoušek, *Using the Borsuk-Ulam theorem*, Springer Verlag, Berlin–Heidelberg–New York, 2003.
15. N. Meggido and C. H. Papadimitriou, *A note on total functions, existence theorems, and computational complexity*, Tech. report, IBM, 1989.
16. F. Meunier, *Discrete splittings of the necklace*, Mathematics of Operations Research **33** (2008), 678–688.
17. F. Meunier and A. Sebő, *Paint shop, odd cycles and splitting necklace*, Discrete Appl. Math. **157** (2009), 780–793.
18. R. H. Möhring, D. Wagner, and F. Wagner, *VLSI Network Design*, vol. 8, ch. 8, pp. 625–712, 1995.
19. B. Neveu and G. Trombettoni, *Incop : An Open Library for INcomplete Combinatorial OPtimization*, Proc. International Conference on Principles Constraint Programming, CP'03 (Kinsale,Ireland), LNCS, vol. 2833, Springer, 2003, pp. 909–913.
20. B. Neveu, G. Trombettoni, and F. Glover, *ID-Walk : A Candidate List Strategy with a Simple Diversification Device*, Proc. 10th International Conference on Principles and Practice of Constraint Programming - CP 2004, vol. LNCS 3258, Springer, 2004, pp. 423–437.
21. C. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs of existence*, J. Comput System Sci. **48** (1994), 498–532.
22. ———, *The Complexity of Finding Nash Equilibria*, Alghorithmic Game Theory (N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, eds.), 2007, pp. 29–51.
23. H. Scarf, *The approximation of fixed points of a continuous mapping*, SIAM J. Appl. Math. **15** (1967), 1328–1343.
24. G. Simonyi, *Necklace bisection with one cut less than needed*, The electronic journal of combinatorics **15** (2008).

Université Paris Est, CERMICS, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France.

Université Paris Est, LIGM, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France.

E-mail address: frederic.meunier@enpc.fr, bertrand.neveu@enpc.fr