Computational aspects of the 2-dimension of partially ordered sets

M. Habib [b], L. Nourine [a], O. Raynaud [a], E. Thierry [c,*]

[a] *LIMOS, Complexe des Ceseaux, 63000 Clermont-Ferrand, France*

[b] *LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France.*

[c] *ENS Lyon, 46 allée d'Italie, 69007 Lyon, France.*

Abstract

A well-known method to represent a partially ordered set P (*order* for short) consists in associating to each element of P a subset of a fixed set $S = \{1, \ldots, k\}$ such that the order relation coincides with subset inclusion. Such an embedding of P into 2^S (the lattice of all subsets of S) is called a *bit-vector encoding* of P. These encodings provide an interesting way to store an order. They are economical with space and comparisons between elements can be performed efficiently via subset inclusion tests.

Given an order P, minimizing the *size* of the encoding, i.e. the cardinal of S, is however a difficult problem. The smallest size is called the *2-dimension* of P and denoted by $\mathbf{Dim_2}(P)$. It is known that the decision problem for the *2-dimension* is $\mathcal{NP}$-complete and generating small bit-vector encodings is a challenging issue.

Several works deal with bit-vector encodings from a theoretical point of view and from a practical point of view. In this article, we focus on computational complexity results. After a synthesis of known results, we come back on the $\mathcal{NP}$-completeness by detailing a proof and enforcing the conclusion with non-approximability ratios. Besides this general result, we investigate the complexity of the 2-dimension for the class of trees. We describe a 4-approximation algorithm for this class. It uses an optimal balancing strategy which solves a conjecture of Krall, Vitek and Horspool. Several interesting open problems are listed.

Key words: Partially ordered sets, trees, boolean lattices, embeddings, bit-vector encodings, 2-dimension, approximation algorithms.

* Corresponding author.
Email address: `Eric.Thierry@ens-lyon.fr` (E. Thierry).

1 Introduction

A well-known method to represent a partially ordered set (*order* for short) is to embed it into another order with good properties. More formally, let $P = (X, \leq_P)$ be an order where $\leq_P$ is an order relation (i.e. reflexive, anti-symmetric and transitive) on a ground set X, and $Q = (Y, \leq_Q)$ another order. An *embedding* from P into Q is a mapping ϕ from X into Y such that for all $x, y \in X$, $x \leq_P y$ if and only if $\phi(x) \leq_Q \phi(y)$. By requiring that Q should belong to a particular class of orders, different interesting classes of embeddings can be defined.

In this article, we study *bit-vector encodings* of orders which are embeddings into boolean lattices. In other words, let $P = (X, \leq_P)$ be an order, a *bit-vector encoding* of P is a mapping ϕ from X into 2^S (the set of all the subsets of a set S, ordered by inclusion) such that for all $x, y \in X$, $x \leq_P y$ if and only if $\phi(x) \subseteq \phi(y)$. We will only consider finite orders and the *size* of the encoding ϕ is the cardinal of S. It is well-known that there always exists a *canonical* bit-vector encoding embedding P into 2^X and defined for all $x \in X$ by $\phi(x) = \{y \in X | y \leq_P x\}$.

Figure 1 shows three different descriptions of a unique bit-vector encoding ϕ of size $|S| = 6$ for a given order P. Note that conventionally orders will be drawn in a botttom-up fashion. The first description (a) associates to each element x its *code* $\phi(x)$ which is a subset of $S = \{1, 2, 3, 4, 5, 6\}$, and checking whether $x \leq_P y$ is equivalent to check whether $\phi(x) \subseteq \phi(y)$. The second one (b) associates to each element x a vector V_x of $|S|$ bits where bit i is equal to 1 if $i \in \phi(x)$ and equal to 0 otherwise. In that case, checking whether $x \leq_P y$ is equivalent to check whether V_x `OR` $V_y = V_y$ on the vectors. The last one (c) associates with each element x its *reduced code* $\phi_r(x) = \phi(x) \setminus \bigcup_{y \leq_P x,\ y \neq x} \phi(y)$. These three representations are perfectly equivalent. The complete code of an element is obtained by propagating the reduced codes of its predecessors. The reduced codes do not provide a direct test to compare two elements, they are mainly introduced to study and generate bit-vectors encodings of an order. In order to distinguish the elements of S and the ones of X, we will call *colors* the elements of S.

Bit-vector encodings provide a compact way to store an order. The size of a bit-vector encoding can be really lower than the n bits per element required for instance by the binary matrix storage of the order relation (where n is the number of elements). Concerning the speed of the inclusion tests, we have seen that checking whether $x \leq_P y$ is equivalent to check whether V_x `OR` $V_y = V_y$ on the vectors of bits. These are elementary bitwise boolean operations, and the speed is proportional to the size of the encoding divided by the length of

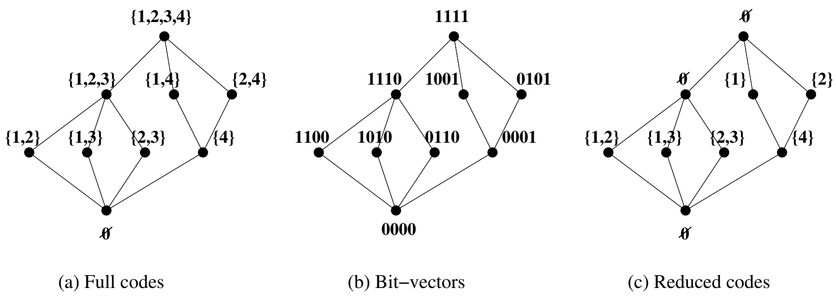

Fig. 1. Three descriptions of a unique bit-vector encoding

machine words. This is a reason why these encodings have been so used (see for instance [3,12,41]).

Thus, minimizing the size of a bit-vector encoding improves both space compression and comparison speed. However this optimization problem (referred as problem $\mathbf{Dim_2}$) is a challenging issue.

Problem 1 $\mathbf{Dim_2}$: embedding orders into boolean lattices
INSTANCE : An order P.
GOAL : minimize k such that there exists an embedding of P into a boolean lattice 2^S with $|S| = k$ (i.e. a bit-vector encoding of size k).

Given an order P, the smallest size of a bit-vector encoding of P is called the *2-dimension* of P and denoted by $\mathbf{Dim_2}(P)$. In [35], Stahl and Wille have shown that the associated decision problem, i.e. deciding whether $\mathbf{Dim_2}(P) \leq k$ for an order P and an integer k, is $\mathcal{NP}$-complete. As a consequence, generating small bit-vector encodings for orders is a challenging problem. A bit-vector encoding of size $\mathbf{Dim_2}(P)$ is called an *optimal bit-vector encoding* of P.

The 2-dimension was originally studied from a mathematical point of view. In 1963, extending the definition and characterization of the *dimension* of orders by Dushnik and Miller [11] and Ore [27], Novak introduced the *k-dimension* of an order P as the minimum number of chains of height $k-1$ such that there exists an embedding of P into their cartesian product [26]. The 2-dimension is a particular case as a cartesian product of chains of height 1 is isomorphic to a boolean lattice and also called hypercube. One of the first study of the 2-dimension was lead by Trotter [38] and some results about k-dimension can be found in [39]. Representations of orders, and more specifically lattices, by sets have also been studied by Markowski [23] and Bouchet [6]. In particular, they solve a more constrained version of the 2-dimension: they give characterizations of the smallest bit-vector encodings of lattices that preserve infimum or supremum. Some exact formulas or bounds were given for a few particular classes of orders such as chains, antichains, crowns, some suborders of boolean lattices, distributive and extremal lattices.

Some previous syntheses introducing algorithmic aspects were written by Habib and Nourine [18,19] and Fall [14]. The $\mathcal{NP}$-completeness arguments for the computation of the 2-dimension were provided by Stahl and Wille [35]. Several combinatorial optimization problems were linked with the computation of the 2-dimension, such as graph coloring [32]. From then, several heuristics have been proposed in order to generate small bit-vector encodings. One of the most famous is the *simple coloring* heuristic introduced by Caseau [7]: the idea is to restrict to the search of bit-vector encodings such that the reduced code of each element has cardinality at most 1. This heuristic takes its name from an equivalence with a graph coloring problem on the elements of the order. In [22], Vitek, Krall and Horspool revisited this approach mainly by adding a preprocessing step which modifies the initial order. Detailed studies of this heuristic can be found in [20] and [8]. Application fields of bit-vector encodings include databases [3], knowledge representation [12], object oriented programming [41].

This work focuses on the algorithmic point of view. First, we investigate the complexity of approximating the 2-dimension. We detail a proof of $\mathcal{NP}$-completeness and show that the reduction fullfills all conditions so that non-approximability results from graph coloring can be transferred to the computation of the 2-dimension. As a consequence, the 2-dimension can not be approximated within a constant factor unless $\mathcal{P} = \mathcal{NP}$.

Moreover, little is known about the complexity of the 2-dimension for specific classes of orders. The class of trees looks as a good candidate for further investigation: it belongs to many classical classes of orders and in many applications, the orders involved are tree-like. Such a study could lead to new ideas in order to design efficient heuristics. We study in details an algorithm which was first presented in [29] and which generates compact bit-vector encodings for trees. It is based on a "splitting and balancing" strategy. We show that we have obtained an optimal balancing strategy for trees, working in polynomial time and thus solving a conjecture of [22]. We also prove that the generated encodings approximate the 2-dimension of trees within a factor 4, but we conjecture that this factor may be improved down to 2.

Section 2 introduces useful definitions and propositions for orders and lattices. An overview of known results about the 2-dimension, such as equalities and bounds, is presented in Section 3. In Section 4, non-approximability results for the 2-dimension in the general case are presented. As a consequence of these complexity results, Section 5 shows that the recognition of orders with their 2-dimension equal to their height, is $\mathcal{NP}$-complete. Finally, Section 6 describes our "splitting and balancing" algorithm for the class of trees and evaluates the performances of this algorithm.

2 Definitions and notations

2.1 Partial order definitions

Let $P = (X, \leq_P)$ be a partial order (or *order*) on the ground set X. We only consider finite orders and we also denote by $|P|$ the cardinal of X. The same order relation $\leq_P$ restricted to a subset Y of X is called a *suborder* of P and also referred as the *order induced* by P on Y. Let $x, y \in X$, $x \neq y$, we say that x and y are *comparable* in P if either $x \leq_P y$ or $y \leq_P x$. Otherwise we say that x and y are *incomparable*. A partial order where every pair of elements is comparable is called a *chain*. A partial order where every pair of elements is incomparable is called an *antichain*. By extension, for the order $P = (X, \leq_P)$, a nonempty subset Y of X is called a *chain* (resp. *antichain*) of P if every pair of elements of Y is comparable (resp. incomparable) in P. The maximum cardinality of a chain of P minus 1 is called the *height* of P. The maximum cardinality of an antichain of P is called the *width* of P. An element $x \in X$ is called the *maximum* (resp. *minimum*) of P if for all $y \in X$, $y \leq_P x$ (resp. $x \leq_P y$).

The *comparability graph* of P, denoted by $G(P)$, is the undirected graph whose vertices are the elements of X and where two elements are adjacent if they are comparable in P.

The *strict order relation* for $P = (X, \leq_P)$ is denoted by $<_P$ and defined for all $x, y \in X$ as $x <_P y$ if $x \leq_P y$ and $x \neq y$. For each $x \in X$, we will consider the set of *predecessors* (resp. *successors*) of x in P defined by $Pred_P(x) = \{y \in X | y <_P x\}$ (resp. $Succ_P(x) = \{y \in X | x <_P y\}$).

Moreover we say that x is *covered* by y in P, denoted by $x \prec_P y$, if $x <_P y$ and there is no element $z \in X$ such that $x <_P z$ and $z <_P y$. To manipulate this *cover relation*, for each $x \in X$, we define the set of *immediate predecessors* (resp. *immediate successors*) of x in P which is $ImPred(x) = \{y \in X | y \prec_P x\}$ (resp. $ImSucc(x) = \{y \in X | x \prec_P y\}$).

An order $P = (X, \leq_P)$ is called a *bipartite order* if there exists a partition of X into two sets X_1 and X_2 such that for all $x, y \in X$, $x <_P y$ implies that $x \in X_1$ and $y \in X_2$. In that case, the order is also denoted by $P = (X_1, X_2, <_P)$. A *complete bipartite order* $P = (X_1, X_2, <_P)$ is a bipartite order such that for all $x \in X_1$ and $y \in X_2$, we have $x <_P y$.

An order $P = (X, \leq_P)$ is *isomorphic* to an order $Q = (Y, \leq_Q)$ is there exists a bijection ϕ from X into Y which is an embedding from P into Q (as defined in the introduction). Given two orders $P = (X, \leq_P)$ and $Q = (Y, \leq_Q)$ where X and Y are disjoint sets, we can also define their *parallel composition* $P \cup Q$

which is the order on the ground set $X \cup Y$ such that the order induced by $P \cup Q$ on X (resp. Y) is P (resp. Q) and for all pairs $x \in X$, $y \in Y$, x and y are incomparable.

An order $T = (X, \leq_T)$ is called a *tree* if it has a minimum m (called the *root* of T) and for all $x \in X$, $x \neq m$, x has an unique immediate predecessor (called the *parent* of x). In that case, the elements of T without any successor are called *leaves*. When they exist, the immediate successors of an element x are called the *children* of x. A *subtree* of T is a suborder of T and let $x \in X$ we denote by $T(x)$ the order induced by T on x and its successors.

2.2 Lattice definitions

A lattice $L = (X, \leq_L)$ is an order such that for all $x, y \in X$, the pair $\{x, y\}$ has an infimum $x \wedge_L y$ and a supremum $x \vee_L y$. For instance, the set of all the subsets of a fixed set S ordered by inclusion is a lattice. It is denoted by 2^S for short and called a *boolean lattice of dimension* $|S|$. All boolean lattices of dimension k are isomorphic.

Some elements of a lattice play an essential role : *join-irreducible* and *meet-irreducible* elements. Let $L = (X, \leq_P)$ be a lattice, an element $j \in X$ is called *join-irreducible* if j covers only one element. In the same way, an element $m \in X$ is called *meet-irreducible* if m is covered by only one element. The set of all *join-irreducible* (resp. *meet-irreducible*) elements of L is denoted $J(L)$ (resp. $M(L)$). We can also associate with L its *incidence bipartite order* defined as $Bip(L) = (J(L), M(L), \not\leq_L)$. The irreducible elements may be used to represent all other elements as shown in Proposition 2 (see [10] for a proof).

Proposition 2 ([10]) *Let $L = (X, \leq_L)$ be a lattice. For all $x \in X$, define $J_x(L) = \{j \in J(L) | j \leq_L x\}$ (resp. $M_x(L) = \{m \in M(L) | x \leq_L m\}$). Then for all $x, y \in X$, $x \leq_P y$ if and only if $J_x(L) \subseteq J_y(L)$ (resp. $M_x(L) \supseteq M_y(L)$).*

From an order $P = (X, \leq_P)$, several useful lattices can be constructed such as its *Dedekind-MacNeille completion* and its *lattice of maximal antichains.* They both enable to reconstruct a lattice from its irreducible elements.

The *Dedekind-MacNeille completion* of P is the unique lattice (up to an isomorphism) denoted by $DM(P)$ verifying the two properties: there exists an embedding of P into $DM(P)$ and for any lattice L such that there exists an embedding of P into L, then there exists an embedding of $DM(P)$ into L (see [10] for a proof of its existence). Theorem 3 shows the link with irreducible elements (see [10] for a proof).

Theorem 3 ([10]) *Let L be a lattice and $JM(L) = (J(L) \cup M(L), \leq_L)$ the or-*

der induced on its irreducible elements. Then L is isomorphic to $DM(JM(L))$.

Let $P = (X, \leq_P)$ be an order and let A_1, A_2 be two maximal antichains of P (with respect to inclusion). We say that $A_1 \leq_{MA(P)} A_2$ if for every element $x \in A_1$, there exists an element $y \in A_2$ such that $x \leq_P y$. We denote by $MA(P)$ the set of all maximal antichains (with respect to inclusion) of P ordered by the relation $\leq_{MA(P)}$. This order $MA(P)$ is a lattice called the *lattice of maximal antichains* of P.

Moreover it has been proved that any lattice L is the lattice of maximal antichains of some order [4] and more precisely of the bipartite order $Bip(L)$ [30]. This result has been expanded by Theorem 5 which makes use of the reduction given in Definition 4. This reduction process and the theorem are illustrated on Figure 2 where it can be seen that $MA(B)$ is isomorphic to $MA(Red(B))$.

Definition 4 (Morvan, Nourine [24]) *Let $B = (X, Y, <_B)$ be a bipartite order. This order is called* **reduced** *if there is neither $x \in X$ nor $y \in Y$ satisfying one of the following conditions:*

(1) $Succ_B(x) = \bigcup_{x' \neq x, Succ_B(x') \subseteq Succ_B(x)} Succ_B(x')$.
(2) $Pred_B(y) = \bigcup_{y' \neq y, Pred_B(y') \subseteq Pred_B(y)} Pred_B(y')$.

For any bipartite order B, repeatedly removing vertices satisfying condition (1) or (2) eventually leads to a unique (up to an isomorphism) reduced bipartite order denoted by $Red(B)$.

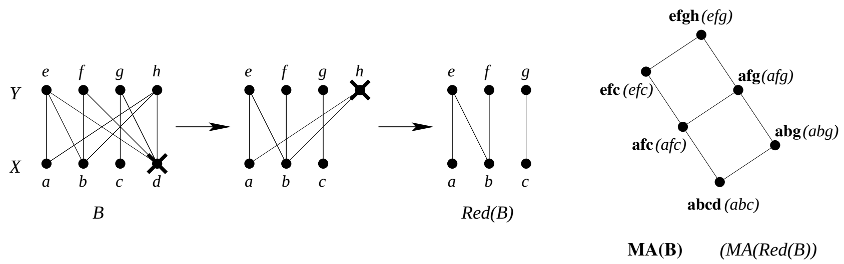

Fig. 2. A bipartite order B, its reduced bipartite $Red(B)$, their lattice of maximal antichains ($MA(L)$ in bold, $MA(Red(L))$ between brackets).

Theorem 5 (Morvan, Nourine [24]) *Let L be a lattice, then $Bip(L)$ is reduced and L is isomorphic to $MA(Bip(L))$. Moreover, let B be a bipartite order such that $L = MA(B)$ then $Red(B)$ is isomorphic to $Bip(L)$.*

Theorems 3 and 5 as well as Proposition 2 will play an important role in the study of the computational complexity of the 2-dimension.

3 Basic results

3.1 Formulas and bounds on the 2-dimension

Several bounds have been established concerning the 2-dimension of orders, and in a few particular cases, some exact formulas have been set. We give here an overview of these results. The following equalities and inequalities give some precisions on the general behaviour of the 2-dimension. At first, Proposition 6 concerns duality, monotony and continuity. For any order $P = (X, \leq_P)$, its dual $P^d = (X, \leq_{P^d})$ is defined by $x \leq_{P^d} y$ iff $y \leq_P x$. Let $x \in X$, then $P - x$ denotes the order induced by P on $X \backslash \{x\}$. Proofs for these first properties can be found in [39,36].

Proposition 6

(1) Let P be an order, then $\mathbf{Dim_2}(P^d) = \mathbf{Dim_2}(P)$ (Duality principle).
(2) Let P and Q be two orders, if there exists an embedding of P into Q then $\mathbf{Dim_2}(P) \leq \mathbf{Dim_2}(Q)$. As a consequence, if P is a suborder of Q, then $\mathbf{Dim_2}(P) \leq \mathbf{Dim_2}(Q)$ (Monotony).
(3) Let P be an order and x one of its elements. Then $\mathbf{Dim_2}(P) \leq \mathbf{Dim_2}(P - x) + 2$ (Continuity).

The next proposition presents the first natural bounds for the 2-dimension.

Proposition 7 *For any order $P = (X, \leq_P)$, $log_2(|P|) \leq \mathbf{Dim_2}(P) \leq |P|$.*

PROOF. On one hand let ϕ be an embedding of P into 2^S. As ϕ is injective and the cardinal of 2^S is $2^{|S|}$ we have $|P| \leq 2^{|S|}$ and thus $log_2(|P|) \leq dim_2(P)$. On the other hand as the canonical embedding which associates to each element of P the set $\{y \in X | y \leq_P x\}$, is a bit-vector encoding of size $|P|$ we have $\mathbf{Dim_2}(P) \leq |P|$.

These two bounds are tight. On one hand, any boolean lattice 2^S has $\mathbf{Dim_2}(2^S) = |S| = log_2(2^S)$. On the other hand, Trotter has given in [38] a complete characterization of the orders satisfying $\mathbf{Dim_2}(P) = |P|$. These orders are particular cases of series-parallel orders and thus they can be recognized with a linear algorithm (see for instance [40]).

We now give two important particular cases where the exact 2-dimension is known. The first case is trivial, but the other one corresponds to a theorem of Sperner, set in 1928, which has lead to a large amount of work known as *Sperner Theory* [1,13].

Proposition 8 *Given a total order P with n elements then $\mathbf{Dim_2}(P) = n-1$. Let $x_0, x_1, \ldots, x_{n-1}$ be the n elements of P ordered by $x_0 <_P x_1 <_P \ldots <_P x_{n-1}$, then an optimal bit-vector encoding ϕ using colors from $S = \{1, \ldots, n-1\}$ is given by $\phi(x_0) = \emptyset$ and $\phi(x_i) = \{1, \ldots, i\}$ for all $1 \leq i \leq n-1$.*

Proposition 9 (Sperner [33]) *Given an antichain P with n elements, then $\mathbf{Dim_2}(P) = sp(n)$ where $sp(n) = \min\{k | \binom{k}{\lfloor k/2 \rfloor} \geq n\}$. A optimal bit-vector of P is obtained by associating with each element a combinaison of $\lfloor sp(n)/2 \rfloor$ colors from $S = \{1, \ldots, sp(n)\}$.*

The next proposition gives a tight approximation of the numbers $sp(n)$ for $n \geq 2$ (see Appendix for a proof).

Proposition 10 *Let $n \geq 2$ and $sp(n) = \min\{k | \binom{k}{\lfloor k/2 \rfloor} \geq n\}$. Then*

$$\log_2(n) + \log_2 \log_2(n)/2 \leq sp(n) \leq \log_2(n) + \log_2 \log_2(n)/2 + 2.$$

These bounds enable to compute $sp(n)$ efficiently. Note however that no exact closed formula is known for $sp(n)$. As a corollary of these two previous propositions and because of monotony, we have the following lower bounds.

Figure 3 shows some optimal bit-vector encodings for a chain and for an antichain.

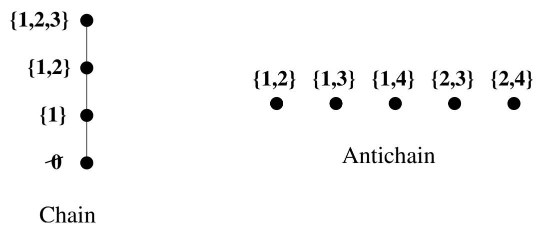

Fig. 3. Optimal encodings for a chain and for an antichain ($sp(5) = 4$).

Corollary 11 *Let $P = (X, \leq_P)$ be an order whose height is $h(P)$ and width $w(P)$ then $\mathbf{Dim_2}(P) \geq h(P)$ and $\mathbf{Dim_2}(P) \geq sp(w(P))$.*

Note that, contrary to the upper bound of Proposition 7, deciding whether $\mathbf{Dim_2}(P) = h(P)$ is $\mathcal{NP}$-complete as it will be shown in Section 5.

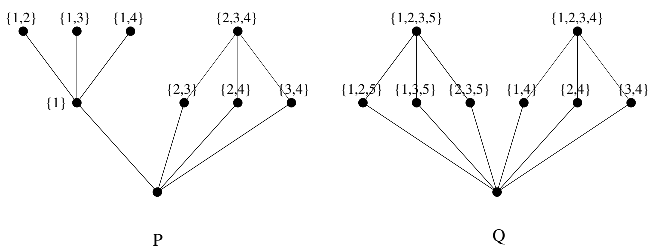

Fig. 4. Two orders with optimal bit-vector encodings.

A parameter of orders is called a **comparability invariant** if its value remains the same for all orders having isomorphic comparability graphs. Unlike the dimension of orders and in spite of the property (1) of duality, the 2-dimension is not a comparability invariant as shown on Figure 4.

Proposition 12 *The 2-dimension is not a comparability invariant.*

PROOF. The orders P and Q on Figure 4 have isomorphic comparability graph, however it can be checked that $\mathbf{Dim_2}(P) \neq \mathbf{Dim_2}(Q)$. Thanks to the encodings on the figure, we have $\mathbf{Dim_2}(P) \leq 4$ and $\mathbf{Dim_2}(Q) \leq 5$. In order to show that $\mathbf{Dim_2}(Q) = 5$, suppose that $\mathbf{Dim_2}(Q) = 4$, then the antichain of Q with six elements necessarily receives the codes $\{1,2\}$, $\{1,3\}$,$\{1,4\}$,$\{2,3\}$,$\{2,4\}$,$\{3,4\}$ (see Proposition 9). However it can be checked that there is no way to assign these codes to the six elements in order to have a bit-vector encoding (there is always one of the maximal elements receiving a code greater than the code of an incomparable element).

The next proposition concerns the Dedekind-MacNeille completion and is a simple consequence of the definition of the 2-dimension as bit-vector encodings are embeddings into lattices. The last statement follows from Theorem 3.

Proposition 13 *Let P be an order and $DM(P)$ its Dedekind-MacNeille completion, then $\mathbf{Dim_2}(DM(P)) = \mathbf{Dim_2}(P)$. In particular, let L be a lattice and $JM(L) = (J(L) \cup M(L), \leq_L)$, then $\mathbf{Dim_2}(L) = \mathbf{Dim_2}(JM(L))$.*

In the particular case of lattices, the upper bound of Proposition 7 may be improved.

Proposition 14 *Given a lattice L, then $\mathbf{Dim_2}(L) \leq \max(|J(L)|, |M(L)|)$.*

This is a consequence of Proposition 2 which involves that $\phi(x) = J_x(L)$ as well as $\phi(x) = M(L)\backslash M_x(L)$ are bit-vector encodings.

Lattices L such that $h(L) = |J(L)|$ or $h(L) = |M(L)|$ are called *extremal lattices*, and *distributive lattices* belong to this class. Thanks the previous propositions, given an extremal lattice L, we have $\mathbf{Dim_2}(L) = h(L)$.

A few other classes of orders have been studied for the 2-dimension and bounds have been given for crowns [34] and parts of boolean lattices [38,21]. These results are not developed here, an interested reader is also referred to [37].

It is known that the problem of computing the 2-dimension of an order or a lattice can be reduced to some other optimization problems such as covering bipartite orders by complete bipartite orders, graph coloring, minimal set basis or non-ambiguous rank of boolean matrices [19]. For our purposes, we will focus on the covering by complete bipartite orders (referred as problem **BipCover**) and Theorem 16 sets the reduction for lattices.

Problem 15 BipCover : covering by complete bipartite orders
INSTANCE : A bipartite order B.
GOAL : minimize k such that there exists a family of k complete bipartite orders covering B ? A bipartite order B is said to be covered by a family $B_1, B_2, ..., B_k$ of complete bipartite orders if and only if for all $1 \leq i \leq k$, B_i is a suborder of B and for all x, y, $x <_B y$ implies that there exists i such that $x <_{B_i} y$.

The minimal number of complete bipartite orders necessary to cover B is denoted by $\mathbf{BipCover}(B)$. The decision problem associated to Problem 15 is $\mathcal{NP}$-complete (GT18 in [17]). In particular, it has been studied by Fishburn and Hammer [16], by Muller [25] and it is also called s-dimension by Stahl and Wille [42,35].

Theorem 16 *Let $L = (X, \leq_L)$ be a lattice and an integer k, the three following propositions are equivalent:*

(1) there exists an embedding of L into a boolean lattice of dimension k.
(2) there exists an embedding of $JM(L) = (J(L) \cup M(L), \leq_L)$ into a boolean lattice of dimension k.
(3) there exists a covering of $Bip(L)$ by k complete bipartite orders.

As a consequence we have $\mathbf{Dim_2}(L) = \mathbf{Dim_2}(JM(L)) = \mathbf{BipCover}(Bip(L))$.

SKETCH OF PROOF. (1) $\Longrightarrow$ (2). This is just monotony: as $JM(L)$ is a suborder of L, it is sufficient to restrict the embedding of L to $JM(L)$.

(2) $\Longrightarrow$ (3). Let ϕ be an embedding of $JM(L)$ into $2^{\{1,\ldots,k\}}$. Then for each $1 \leq i \leq k$, consider all the couples of elements $j \in J(L)$ and $m \in M(L)$ such that $i \in \phi(j) \backslash \phi(m)$, and call B_i the order induced by $Bip(L)$ on these elements. It can be checked that $B_1, \ldots, B_k$ is a covering of $Bip(L)$ by complete bipartite orders. We just need to use the fact that ϕ is an embedding of $JM(L) = (J(L) \cup M(L), \leq_L)$ into $2^{\{1,\ldots,k\}}$.

(3) $\Longrightarrow$ (1). Now let $B_1, \ldots, B_k$ be a covering of $Bip(L)$ by k complete bipartite orders. Then the mapping ϕ from X into $2^{\{1,\ldots,k\}}$ defined for all

$x \in X$ by the formula below is an embedding of L into $2^{\{1,\dots,k\}}$:

$$\phi(x) = \bigcup_{j \leq_L x \ , \ j \not\leq_L m} \{i \mid j <_{B_i} m\}.$$

For a detailed proof, see [19]. Note that theses equivalences provide another way to prove the last statement of Proposition 13.

The next proposition describes the behaviour of the reduction process of Definition 4 in relationship with the **BipCover** problem.

Proposition 17 *Let B be a bipartite order, then* $\mathbf{BipCover}(Red(B)) = \mathbf{BipCover}(B)$.

SKETCH OF PROOF. It is easy to check that each element removal during the reduction process does not change **BipCover**. More precisely, let $B = (X, Y, <_B)$ be the bipartite order and $x \in X$ an element satisfying the condition (1) of Definition 4. By removing x, we obtain the bipartite order $B - x = (X \backslash \{x\}, Y, <_B)$. Let $B_1, B_2, ..., B_k$ be k complete bipartite orders covering B, by removing x from all B_i containing x, we get a family of k complete bipartite orders covering $B - x$. Conversely, let $B_1, B_2, ..., B_k$ be k complete bipartite orders covering $B - x$, by adding x to all $B_i = (X_i, Y_i, <_B)$ such that $B_i \subseteq Succ_B(x)$, we get a family of k complete bipartite orders and the condition (1) ensures that this family covers B. The dual constructions applies to an element satisfying the condition (2) of Definition 4.

4 Non-approximability of the 2-dimension

In light of all the previous results, Theorem 18 now describes a reduction of the problem **BipCover** to the problem $\mathbf{Dim_2}$ which does not only ensure that the decision problem associated to $\mathbf{Dim_2}$ is $\mathcal{NP}$-complete but also enables to transfer non-approximability results.

Theorem 18 *There exists a polynomial algorithm generating from any bipartite order B an order P with $|P| \leq |B|$ such that for any covering of B by k complete bipartite orders we can construct polynomially an embedding of P into $2^{\{1,\dots,k\}}$ and conversely.*

PROOF. This algorithm reducing the problem **BipCover** to the problem $\mathbf{Dim_2}$ works as follows:

- First, we compute the reduced bipartite order $B' = Red(B)$ from B as described in Definition 4. It can be done polynomially in $|B|$. Thanks to

Proposition 17, we know that from any covering of B' by complete bipartite orders we can construct in polynomial time a covering of B with the same number of complete bipartite orders, and inversely. Thus we can focus on the coverings of B'.

- Now consider its lattice of maximal antichains $L = MA(B')$. From Theorem 5, B' is isomorphic to $Bip(L)$. From Theorem 16, we know that $Bip(L)$ behaves for **BipCover** like $JM(L) = (J(L) \cup M(L), \leq_L)$ for $\mathbf{Dim_2}$. Thus we aim at generating an order isomorphic to $JM(L)$.
- The core of the reduction lies in the fact that from $Bip(L) = (J(L), M(L), \not\leq_L)$, we can rebuild $JM(L) = (J(L) \cup M(L), \leq_L)$ in polynomial time. From $Bip(L)$, construct the bipartite order $B(L) = (J(L)\dot{\cup}M(L), M(L), <_{B(L)})$ where $J(L)\dot{\cup}M(L)$ is the disjoint union of $J(L)$ and $M(L)$, and for all $j \in J(L)$, $m \in M(L)$, $j <_{B(L)} m$ if and only if $j <_{Bip(L)} m$, and for all $m \in M(L)$, $m' \in M(L)$, $m <_{B(L)} m'$ if and only if $Pred_{Bip(L)}(m) \not\supseteq Pred_{Bip(L)}(m')$. Then for all $x, y \in J(L)\dot{\cup}M(L)$, it can be proved that $x \leq_L y$ if and only if $Succ_{B(L)}(x) \subseteq Succ_{B(L)}(y)$ (use Proposition 2 to prove that for all $x \in J(L)\dot{\cup}M(L)$, $Succ_{B(L)}(x) = M(L) \backslash M_x(L)$ and conclude with the same proposition). This equivalence enables to rebuild $\leq_L$ from $B(L)$ in polynomial time. Note that $Succ_{B(L)}(x) = Succ_{B(L)}(y)$ may only occur when $x \in J(L)$ and $y \in M(L)$ and means that $x = y \in J(L) \cap M(L)$, which enables to rebuild $J(L) \cup M(L)$.
- This construction does not use the labels of the elements and thus by applying this construction to B' which is isomorphic to $Bip(L)$ we obtain an order P isomorphic to $JM(L)$ and whose elements have the labels of the elements of B'.
- The equivalence of Theorem 16 can be used then: from any covering of B' by k complete bipartite orders we can construct an embedding of P into $2^{\{1,\dots,k\}}$ and conversely. A careful study of these transformations in Theorem 16 shows that they can be done in polynomial time.

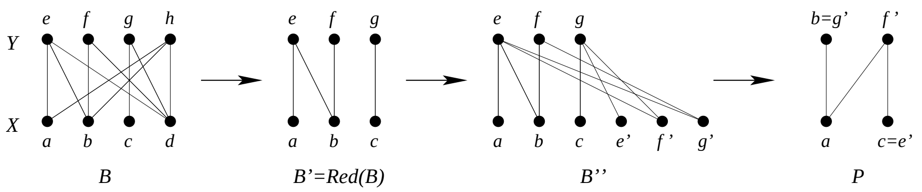

Fig. 5. A bipartite order B, its reduced bipartite $B' = Red(B)$ isomorphic to $Bip(L)$ where $L = MA(B')$, the bipartite order B'' built from B and isomorphic to $B(L)$, the order P built from B'' and isomorphic to $JM(L)$.

This reduction is illustrated on Figure 5. Note that from B', we could have simply built $L = MA(B')$ which also satisfies Theorem 18 expect that $|L|$ is not necessary lower than $|B|$. However its generation can not be done in polynomial time since $|MA(B')|$ may be exponential in $|B'|$. Thus considering $JM(L)$ which shares the same properties as L for Theorem 16 but also verifies $|JM(L)| \leq |B'|$ enables to have a polynomial reduction.

As proved in [28] the decision problem associated to **BipCover** is $\mathcal{NP}$-complete. Thanks to Theorem 18 we have the following corollary.

Corollary 19 *Deciding whether* $\mathbf{Dim_2}(P) \leq k$ *for any order* P *and integer* k *is* $\mathcal{NP}$*-complete.*

Moreover in [31], Simon presented a reduction showing that for any function f on integers, if there exists a polynomial algorithm approximating $\mathbf{BipCover}(B)$ within $f(n)$ for any bipartite order B with n elements, then there exists a polynomial algorithm approximating the chromatic number of any graph G within $2f(n^3)$ where n is the number of vertices of G. Approximating the chromatic number of graphs has been intensively studied and one of the main result is due to Bellare, Goldreich and Sudan [5]: for all $\epsilon > 0$, there is no polynomial algorithm approximating the chromatic number of graphs within $\mathcal{O}(n^{1/7-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$. Theorem 18 allows to transfer the non-approximability results of **BipCover** to $\mathbf{Dim_2}$ and the next corollary sums up the consequence of all these results.

Corollary 20 *For all* $\epsilon > 0$*, there is no polynomial algorithm approximating* $\mathbf{Dim_2}(P)$ *within* $\mathcal{O}(n^{1/21-\epsilon})$ *for any order* P *with* n *elements, unless* $\mathcal{P} = \mathcal{NP}$*.*

This corollary has consequences on the approximability of some other minimal embeddings: for all integers $k \geq 2$, denote by $\mathbf{Dim_k}(P)$ the k-dimension of the order P as it is defined in the introduction. It is known that for any $k \geq 2$ and any order P, it is easy to transform embeddings in order to prove that $\mathbf{Dim_k}(P) \leq \mathbf{Dim_2}(P) \leq (k-1)\mathbf{Dim_k}(P)$ (see Appendix). Thus for any fixed integer $k \geq 2$, $\mathbf{Dim_k}$ and $\mathbf{Dim_2}$ have the same behaviour for approximation (up to a constant factor), leading to the next corollary.

Corollary 21 *Given a fixed integer* $k \geq 2$*, then for all* $\epsilon > 0$*, there is no polynomial algorithm approximating* $\mathbf{Dim_k}(P)$ *within* $\mathcal{O}(n^{1/21-\epsilon})$ *for any order* P *with* n *elements, unless* $\mathcal{P} = \mathcal{NP}$*.*

5 Recognition of orders such that $\mathbf{Dim_2}(P) = h(P)$

It is easy to find an order P such that $\mathbf{Dim_2}(P) = h(P)$ where $h(P)$ is the height of P: consider for instance a chain. Extremal lattices also work (see Subsection 3.1). However recognizing such orders is difficult in general. In Section 3, we have stated that the problem of deciding whether $\mathbf{Dim_2}(P) = h(P)$ for any order P is $\mathcal{NP}$-complete. A proof of this result relies on the $\mathcal{NP}$-completeness of the decision problem associated with $\mathbf{Dim_2}$ and on the following lemma.

Lemma 22 *Let $P = (X, \leq_P)$ and $Q = (Y, \leq_Y)$ be two orders such that P has a minimum m_P and a maximum M_P, and Q has a minimum m_Q and a maximum M_Q. Then their parallel composition $P \cup Q$ satisfies $\mathbf{Dim_2}(P \cup Q) = \max(\mathbf{Dim_2}(P), \mathbf{Dim_2}(Q)) + 2$.*

PROOF. Without loss of generality, suppose that $\mathbf{Dim_2}(P) \leq \mathbf{Dim_2}(Q)$. Thus there exists an embedding ϕ_1 of P into 2^{S_1} and an embedding ϕ_2 of Q into 2^{S_2} such that $|S_1| = \mathbf{Dim_2}(P)$, $|S_2| = \mathbf{Dim_2}(Q)$ and $S_1 \subseteq S_2$. Add two new colors a and b to S_2. We can easily check that the mapping ϕ defined from $X \cup Y$ into $2^{S_2 \cup \{a,b\}}$ by $\phi(x) = \phi_1(x) \cup \{a\}$ if $x \in X$ and $\phi(x) = \phi_2(x) \cup \{b\}$ if $x \in Y$ is an embedding of $P \cup Q$. Thus $\mathbf{Dim_2}(P \cup Q) \leq \max(\mathbf{Dim_2}(P), \mathbf{Dim_2}(Q)) + 2$.

Consider now an embedding ϕ of $P \cup Q$ into 2^S. There exists a color $a \in S$ such that $a \in \phi(m_P) \backslash \phi(M_Q)$ since $m_P \not\leq_{P \cup Q} M_Q$. It means that for all $y \in Y$, $a \notin \phi(y)$. Dually, there exists $b \in S$ such that $b \in \phi(m_Q) \backslash \phi(M_P)$ implying that for all $x \in X$, $b \notin \phi(x)$. Then we can check that the mapping ϕ_1 (resp. ϕ_2) from X (resp. Y) into $2^{S \backslash \{a,b\}}$ defined by $\phi_1(x) = \phi(x) \backslash \phi(m_P)$ (resp. $\phi_2(x) = \phi(x) \backslash \phi(m_Q)$) is an embedding of P (resp. Q). Thus $|S \backslash \{a, b\}| \geq \max(\mathbf{Dim_2}(P), \mathbf{Dim_2}(Q)) + 2$, which enables to conclude that $\mathbf{Dim_2}(P \cup Q) \geq \max(\mathbf{Dim_2}(P), \mathbf{Dim_2}(Q)) + 2$.

Note that the condition on the existences of a minimum and a maximum for both orders is important. Otherwise we may have the equality $\mathbf{Dim_2}(P \cup Q) = \max(\mathbf{Dim_2}(P), \mathbf{Dim_2}(Q)) + 1$. Now we can prove the $\mathcal{NP}$-completeness.

Proposition 23 *Deciding whether $\mathbf{Dim_2}(P) = h(P)$ for any order P is $\mathcal{NP}$-complete.*

PROOF. Consider an instance of the decision problem associated to the problem $\mathbf{Dim_2}$, namely an order P with a maximum and a minimum, and an integer k such that $h(P) \leq k \leq |P|$. The restrictions on P and k are allowed since adding a maximum (resp. minimum) to an order which has no maximum (resp. minimum) does not change its 2-dimension, and since $\mathbf{Dim_2}(P) \geq h(P)$ (see Corollary 11). It is known that deciding whether $\mathbf{Dim_2}(P) \leq k$ is $\mathcal{NP}$-complete (see Corollary 19).

The reduction proving Proposition 23 transforms this instance as follows. Consider a chain C_{k+1} with $k + 1$ elements and build the parallel composition $P \cup C_{k+1}$. Add a minimum and a maximum to $P \cup C_{k+1}$, which gives the order Q. This order Q satisfies $h(Q) = \max(h(P), h(C_{k+1})) + 2 = k + 2$ since $h(P) \leq k = h(C_{k+1})$, and $\mathbf{Dim_2}(Q) = \max(\mathbf{Dim_2}(P), k) + 2$ thanks to Lemma 22. The construction of Q is polynomial (since $k \leq |P|$) and

$\mathbf{Dim_2}(Q) = h(Q)$ iff $\mathbf{Dim_2}(P) \leq k$, which completes the proof of $\mathcal{NP}$-completeness. The transformation is illustrated on Figure 6.

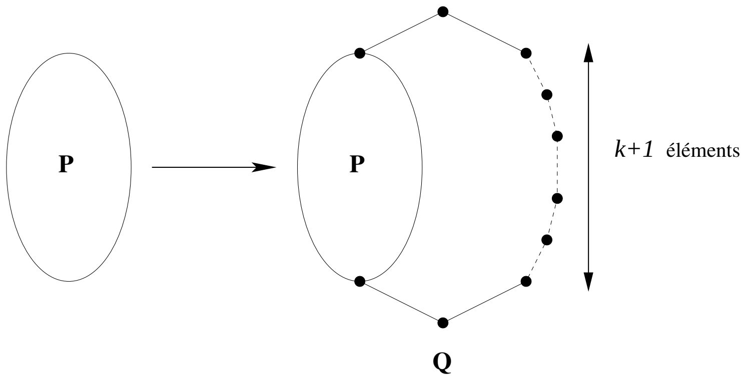

Fig. 6. Transformation of P into Q such that $\mathbf{Dim_2}(P) \leq k$ iff $\mathbf{Dim_2}(Q) = h(Q)$.

6 The class of trees

As mentionned before, there are really few classes of orders for which the computational complexity of the 2-dimension has been set. This section is dedicated to the class of trees which appears as an interesting case since trees belong to several classical classes of orders and are a common structure encountered in practice.

After presenting the previous works about this class of orders, we introduce a particular class of bit-vector encodings: *dichotomic encodings*. They formalize the "splitting and balancing" strategies used to compute bit-vectors of trees. Then we describe a new algorithm generating very compact bit-vector encodings for trees. We show that our algorithm produces dichotomic encodings of minimum size. And finally we demonstrate that the bit-vector encodings generated by our algorithm approximate the 2-dimension of a tree within a factor 4 (which we think may be improved).

6.1 Previous work on trees

Bit-vector encodings of trees have been studied by Caseau [7], Krall, Vitek and Horspool [22] and Caseau *et al* [8]. All these authors have proposed some heuristics in order to produce some bit-vector encodings of small size. We will briefly present two of them in this subsection. They use the following definitions.

Definition 24 *Let $T = (X, \leq_T)$ be a tree. The* **degree** *of an element $x \in X$ is $deg_T(x) = |ImSucc_T(x)|$. Let C be a chain of T, the* **chain-degree** *of C is defined as $deg_P(T) = \sum_{x \in C} deg_T(x)$. The* **maximum chain-degree** *of T is* $\max\{deg_T(C) | C \text{ chain of } T\}$.

Algorithm Cmax described below is the heuristic proposed by Caseau in [7]. This algorithm proceeds in a top-down way and associates to each element a reduced code which is a singleton. Then the full codes are obtained at step 2 by propagating the colors of the reduced codes.

Algorithm Cmax : Caseau encoding generation for a given tree T

Data : A tree T.
Result: A bit-vector encoding of T.
begin
  Each leaf v receives a weight: $weight(v) \leftarrow 0$;
  Other elements are not given any weight;
  while *There exists an element s without weight such that all its children have a weight* **do**
    Let $v_1, ..., v_k$ be the k children of s, and max the maximum of the weights of these children;
1   **for** *i from 1 to k* **do** Give to v_i the color : $c(v_i) \leftarrow max + i$;
    Give to s the weight : $weight(s) \leftarrow max + i$;
2 Associate with each node s of T the following encoding : $code(s) = \{c(s') | s' \in T \text{ and } s' \leq_{BT} s\}$;
end

Theorem 25 (Caseau [7]) *Let T be a tree. Then Algorithm* Cmax *generates in polynomial time bit-vector encodings of T with sizes equal to the maximum chain-degree.*

In [8], Caseau *et al* have modified the step 1 of Algorithm Cmax. We call their heuristic Algorithm CHNR. It replaces step 1 by associating to the elements $v_1, ..., v_k$ reduced codes which are the codes of an optimal bit-vector encoding of the antichain of width k using colors from $max + 1$ to $max + sp(k)$. It is easy to check that Algorithm CHNR still generates bit-vector encodings and Theorem 26 sums up the consequences.

Theorem 26 (Caseau,Habib,Nourine,Raynaud [8]) *Let T be a tree. Then Algorithm* CHNR *generates in polynomial time bit-vector encodings of T with sizes equal to* $\max\{\sum_{x \in C} sp(deg_T(x)) | C \text{ chain of } T\}$.

Figure 7 represents the reduced codes of a bit-vector encoding generated by Algorithm Caseau and of another one generated by Algorithm CHNR.

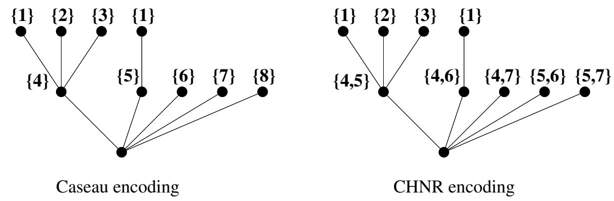

Fig. 7. The reduced codes of a Caseau encoding and a CNHR encoding.

Our algorithm for trees is presented in the next subsections. It is based on two ideas that were first suggested by Caseau [7] and Krall, Vitek and Horspool [22] : "splitting" and "balancing". We start by formalizing the "splitting" idea in the following subsection.

6.2 Dichotomic encodings

The idea consists in splitting in two parts the initial tree and giving a different color to the codes of each part. By repeating this operation on each part recursively until each subtree is empty, we are able to produce what we call a *dichotomic encoding.*

Let T be an order with a bit-vector encoding, let i be a color of this encoding, we will call T_i the set of the elements of T which contain i in their code. We give a recursive definition of a *dichotomic encoding.*

Definition 27 *Let T be an order, we will say that a bit-vector encoding of T is dichotomic if there exist two colors i and j such that:*

- *The code of each element of T contains either i or j (if it is nonempty).*
- *The codes of T_i minus i form a dichotomic encoding of T_i.*
- *The codes of T_j minus j form a dichotomic encoding of T_j.*
- *If T is composed of an unique element, the empty code is considered as a dichotomic encoding.*

Trees are the only orders admitting dichotomic encodings. We call this technique *dichotomic* because we split the tree in two differents parts which can be encoded independently. Let us first remark that a dichotomic encoding is equivalent to an embedding of the hierarchy into a *binary tree*, i.e. a tree where all elements have at most two children.

Proposition 28 *Let T be a tree. Then there exists a dichotomic encoding of T of size k if and only if there exists an embedding of T into a binary tree of maximum chain-degree k.*

PROOF. Consider a dichotomic encoding of T of size k. There exist two colors i and j satisfying the conditions of Definition 27. If T_i is nonempty and has a minimum, consider the tree $T'_i = T_i$ with the dichotomic encoding obtained by removing i from the codes of T_i. If T_i is nonempty and has no minimum, add a new element x and insert it as a child of the root of T and a predecessor of all the elements of T_i, and then consider the tree T'_i composed of $x$with an empty code as a root and T_i with the dichotomic encoding obtained by removing i from the codes of T_i. Do the same for T_j (which gives T'_j). Then if T'_i (resp. T'_j) is nonempty, repeat recursively this construction on T'_i (resp. T'_j). The additions of new elements lead to the construction of a binary tree BT of maximum chain-degree k in which T is embedded.

Conversely, if there exists an embedding of T into a binary tree BT of maximum chain-degree k, use Algorithm Cmax to construct a bit-vector encoding of BT of size k. This provides a bit-vector encoding of T of size k.

These constructions are illustrated on Figure 8 (on the right, the added elements are in white).

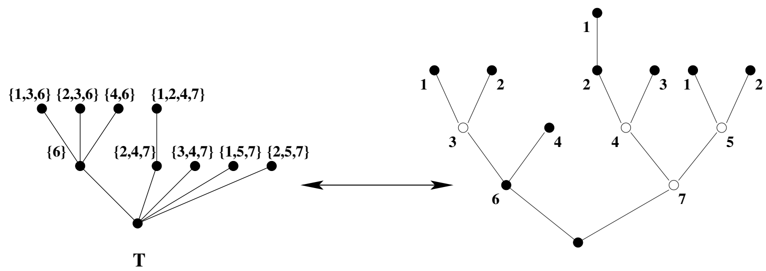

Fig. 8. A dichotomic encoding of the tree T and the associated binary tree with its reduced codes.

There exist many dichotomic encodings for a given tree. In the next subsection, we present an algorithm which produces dichotomic encodings of minimum size.

6.3 The algorithm

Our algorithm, called Algorithm Dicho, is a top-down greedy algorithm which basically constructs a binary tree BT from the input tree T by introducing intermediate elements (symbolising "splitting" operations) and it associates weights with the elements in order to "balance" this binary tree. The weight of a element x denoted by $weight(x)$ corresponds to the number of colors necessary to encode the subtree composed of x and all its successors (this subtree is denoted by $T(x)$). The computation of the weights follows a top-down topological order of the tree: we need to calculate the weights of all the children of a element before calculating its own weight.

At step 2, the algorithm has constructed the binary tree BT in which the input tree T is embedded and every element s of BT (the elements of T and the new intermediate elements) has a unique color stored in $c(s)$. It corresponds to the reduced encoding of our bit-vector encoding and the last step consists in propagating these colors through the tree to obtain the final encoding.

Algorithm Dicho : Dichotomic encoding generation for a given tree T

Data : A tree T.
Result: A dichotomic encoding of T.
begin
  Each leaf v receives a weight: $weight(v) \leftarrow 0$;
  Other elements are not given any weight;
  while *There exists an element s without weight such that all its children have a weight* **do**
    if *s has a unique child v* **then**
      Give to v the color : $c(v) \leftarrow weight(v) + 1$;
1   Give to s the weight : $weight(s) \leftarrow weight(v) + 1$;
    if *s has exactly 2 children u and v* **then**
      Give to u the color : $c(u) \leftarrow \max(weight(u), weight(v)) + 1$;
      Give to v the color : $c(v) \leftarrow \max(weight(u), weight(v)) + 2$;
      Give to s the weight :
      $weight(s) \leftarrow \max(weight(u), weight(v)) + 2$;
    if *s has more than 3 children* **then**
      Choose u and v two children whose weights are minimal;
      Introduce a new node w whose children are u and v, and which is the child of s;
      Give to u the color: $c(u) \leftarrow \max(weight(u), weight(v)) + 1$;
      Give to v the color : $c(v) \leftarrow \max(weight(u), weight(v)) + 2$;
      Give to w the weight :
      $weight(w) \leftarrow \max(weight(u), weight(v)) + 2$;
2 Let BT be the tree generated from T, associate with each node s of T the following encoding : $code(s) = \{c(s') | s' \in BT \text{ and } s' \leq_{BT} s\}$;
end

Figure 9 shows an execution of Algorithm Dicho which generates the tree BT from T. Intermediate elements are in white, the color $c(s)$ of a node s is given by an integer, the integer given between parenthesis corresponds to the weight carried by the node. The last drawing represents the initial tree with its final bit-vector encoding (after step 2).

Computing the weight of a node from the weight of its k children can be done in $O(k * log_2 k)$ time by sorting the weights of the children. Algorithm Dicho can be implemented in a top-down way running in $O(n * log_2 \Delta)$ time with

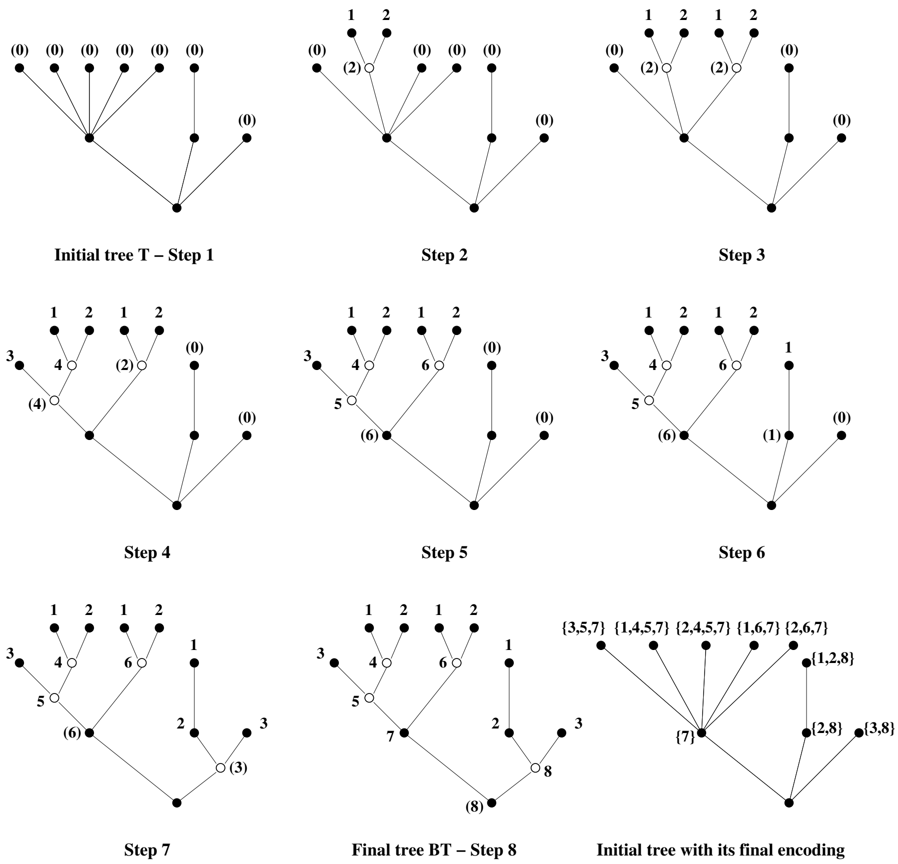

Fig. 9. The initial tree T, the binary tree BT and the final *dichotomic encoding* computed by Algorithm Dicho

$O(n)$ space up to step 2 and $O(n * s)$ time and space for step 2, where n is the number of elements of the tree, Δ is the maximum degree of an element of the tree and s is the size of the output encoding.

6.4 Optimality result

The next theorem garanties that in the framework of "splitting and balancing" strategies, our algorithm provides a bit-vector encoding of optimal size. In [22], Krall, Vitek and Horspool conjectured it was feasible in polynomial time. Algorithm Dicho solves this conjecture.

Theorem 29 *Algorithm* Dicho *generates dichotomic encodings of minimum size, in* $O(n * s)$ *time where* s *is the size of the output encoding,* $s \leq n$*.*

PROOF.

Thanks to Proposition 28, we know that producing a dichotomic encoding of a tree T is equivalent to embed T into a binary tree BT and the size of the dichotomic encoding is the maximum chain-degree of BT.

Let T be the initial tree. Consider an embedding of T into a binary tree BT. We can associate with each element x of BT a weight $w(x)$ which is the maximum chain-degree of the subtree $BT(x)$ having x as a root (in terms of dichotomic encoding this is the number of colors used to encode this subtree).

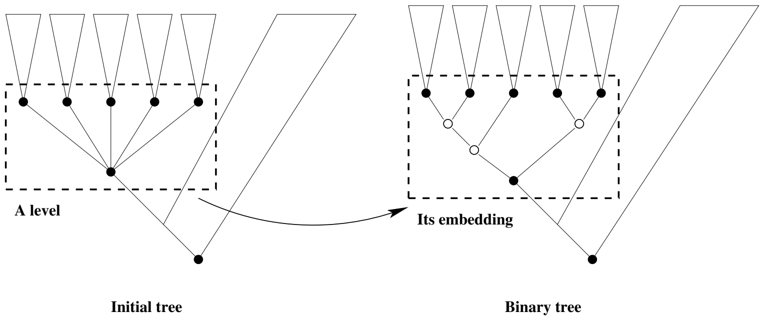

Fig. 10. A ***level*** of the initial tree and its embedding.

To prove that our algorithm computes a binary tree embedding T with the smallest maximum chain-degree, we just have to prove locally that it makes the best embedding for a ***level***, namely a subtree consisting in an element s of T without weight with all its children carrying a weight (see Figure 10). We are going to prove that the construction of Algorithm Dicho minimizes the weight $w(s)$ that will be propagated to the element s. We only consider the case when there are at least two children, otherwise with one child, Algorithm Dicho has clearly the best behaviour.

(1) Consider an embedding of this level into a binary tree BL. We associate with each element x its height $h(x)$ in BL and each leaf x of BL carries a weight $w(x)$. The weight $w(s)$ propagated to s the root of BL depends on the structure of BL. It is denoted by $weight(BL)$ and it is the maximum of $w(x) + 2 * h(x)$ for all the leaves of BL.
(2) Suppose now that two leaves α and β of BL with the smallest weights $w(\alpha)$ and $w(\beta)$ are not "married" in BL (the children of the same element). Then we can construct another binary tree BL' embedding the level such that α and β are married and $weight(BL') \leq weight(BL)$. Figure 11 describes this situation. Initially the elements α and β have a first common ancestor which is s', α is "married" with the element γ and β is "married" with the element δ. The height of the parent of α and γ (resp. β and δ) from s' is denoted by h_γ (resp. h_δ). Now suppose that $h_\gamma \leq h_\delta$, by inverting δ and α we have a new binary tree BL' where α and β are "married" and such that $weight(BL') \leq weight(BL)$ (if we had $h_\gamma \geq h_\delta$, we would have inverted γ and β).
(3) Suppose now that we suppress two "married" leaves of BL with weights

α and β and we associate with the new leaf (their common parent) the weight $\max(w(\alpha), w(\beta)) + 2$. Then it is clear that this new binary tree BL' verifies $weight(BL') = weight(BL)$.

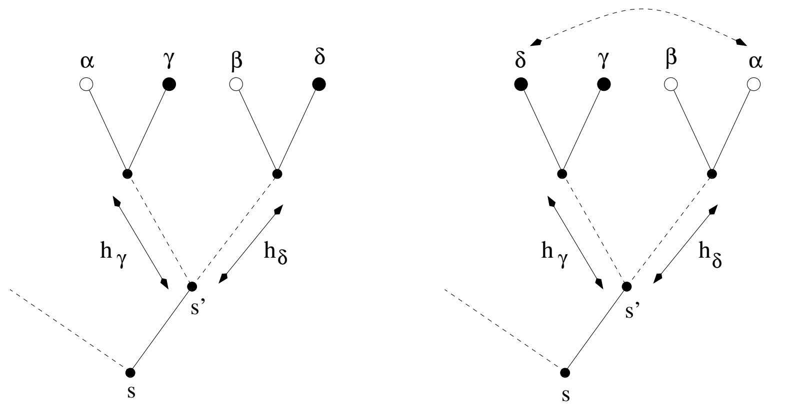

Fig. 11. Transformation of the embedding when $h_\gamma \leq h_\delta$.

Given a level with weights on the leaves, consider an embedding BL of minimum weight $weight(BL)$. By alternatively using the transformations 2 and 3, we transform BL into another binary tree BL' with the same optimal weight and which is an embedding of the level that can be constructed by Algorithm Dicho. Since all the embeddings of a level that can be constructed by Algorithm Dicho propagate the same weight to its root, they are all optimal.

Another formulation of this result is that for any tree T, Algorithm Dicho computes a binary tree of smallest maximum chain-degree in which T is embedded. If all the elements of T have zero or at least two children, then Algorithm Dicho computes a binary tree of smallest height in which T embeds (since chain-weight is twice the height in that case).

Note the analogy between Algorithm Dicho and the usual algorithm used to compute Huffman encodings in compression theory. They are both greedy algorithms constructing binary trees and guided by the propagation of weights. The proof of optimality above is clearly similar to proofs of optimality for Huffman encodings (such as the one in [9]).

6.5 4-approximation result

Algorithm Dicho does not provide the exact value of the 2-dimension for trees, as it can be seen for a tree simply composed of a root with n children: Algorithm Dicho generates encodings of size $2\lceil \log_2(n) \rceil$ whereas the 2-dimension is the one of the antichain, namely $sp(n)$. However we are going to prove that our algorithm approximates the 2-dimension of trees within a factor 4.

For sake of simplicity, we will consider a **variant** of our algorithm, called DichoEven, such that all the propagated weights are even: at step 1, change "$weight(s) \leftarrow weight(v) + 1$;" by "$weight(s) \leftarrow weight(v) + 2$;" (it does not change anything if initially all the elements have zero or at least two children). For an element x and the associated subtree $T(x)$, we will denote by $algo(T(x))$ half the weight propagated to x during the running of the variant of the algorithm (it is possible since all computed weights will be even).

Proposition 30 *Let T be a tree and let s be an element without weight having k children respectively carrying the weights $2p_1, 2p_2, ..., 2p_k$, then Algorithm* DichoEven *will always associate to s the weight $2p$ such that $p = \min\{n | 2^{p_1} + 2^{p_2} + ... + 2^{p_k} \leq 2^n\}$ if $k \geq 2$, and $2p_1 + 2$ if $k = 1$.*

PROOF. By induction on the number k of children of s.

- If s has only one child whose weight is $2p_1$, Algorithm DichoEven gives to s the weight $2p_1 + 2$.
- Suppose now that the proposition is true for elements having at most $k-1$ children. Let s be an element with k children with weights $2p_1 \leq 2p_2 \leq ... \leq 2p_k$. Algorithm DichoEven replaces the two children of weight $2p_1$ and $2p_2$ by a unique new children whose weight is $2p_2 + 2$ and goes on with the $k-1$ children. Since Proposition 30 applies for $k-1$ children, in order to prove the formula for k children, we just have to prove the equivalence: for all $n \leq 0$, $2^{p_1} + 2^{p_2} + ... + 2^{p_k} \leq 2^n$ if and only if $2^{p_2+1} + 2^{p_2} + ... + 2^{p_k} \leq 2^n$. The equivalence involves that the minimal value n is not changed after the replacement of the two children. The direction $\Leftarrow$ is clear since $2^{p_1} + 2^{p_2} \leq 2^{p_2+1}$. To prove the other implication, let us assume that $2^{p_1} + 2^{p_2} + ... + 2^{p_k} \leq 2^n$. By deleting the first term, we get the strict inequality $2^{p_2} + ... + 2^{p_k} < 2^n$. A division by 2^{p_2} gives $1 + 2^{p_3-p_2} + ... + 2^{p_k-p_2} < 2^{n-p_2}$ and then $2 + 2^{p_3-p_2} + ... + 2^{p_k-p_2} \leq 2^{n-p2}$. A final multiplication by 2^{p_2} provides the expected result.

Proposition 31 *Let T be a tree of height h and with l leaves, then $algo(T) \leq h + log_2(l)$.*

PROOF. By induction on the height of the tree.

- If T is reduced to one node, $algo(T)$ is equal to zero, so the property is true.
- Let us assume that the root of T has only one child. Let $T_1 = T(x_1)$ be the subtree rooted at x_1. We have $algo(T) = algo(T_1) + 1$. Let h_1 be the height of T_1 and l_1 its number of leaves, then $h = h_1 + 1$ and $l = l_1$. Since $h_1 < h$, by applying the induction hypothesis, we have $algo(T_1) \leq h_1 + log_2(l_1)$ and thus $algo(T) = algo(T_1) + 1 \leq h_1 + 1 + log_2(l_1) = h + log_2(l)$.

- If the root of T has k children $x_1, ..., x_k$, $k \geq 2$. Let $T_i = T(x_i)$ be the subtree rooted at x_i with height h_i and l_i leaves, $1 \leq i \leq k$. We have $h = \max_{1 \leq i \leq k}(h_i) + 1$ and $f = \sum_{1 \leq i \leq k}(f_i)$ and the induction hypothesis ensures that for all i, $1 \leq i \leq k$, $algo(T_i) \leq h_i + log_2(l_i)$. From Proposition 30, $algo(T) = \min\{n | 2^{algo(T_1)} + 2^{algo(T_2)} + ... + 2^{algo(T_k)} \leq 2^n\}$. Since $2^{algo(T_1)} + 2^{algo(T_2)} + ... + 2^{algo(T_k)} \leq 2^{max(h_i)}(f_1 + f_2 + ... + f_k) = 2^{h-1} f$, we get $algo(T) \leq \lceil h - 1 + log_2(f) \rceil \leq h + log_2(f)$.

Theorem 32 *Let T be a tree, then Algorithm* Dicho *generates bit-vector encodings approximating $\mathbf{Dim_2}(T)$ within a factor 4.*

PROOF. We show this ratio for Algorithm DichoEven. Let T be a tree with height h and l leaves, we know from Proposition 11 that $\mathbf{Dim_2}(T) \geq max(h, \lceil log_2(l) \rceil)$. The size of a *dichotomic encoding* generated by Algorithm DichoEven is equal to $2 * algo(T)$. From Proposition 31, we have $algo(T) \leq h + \lceil log_2(l) \rceil$ thus $algo(T) \leq 2 * \mathbf{Dim_2}(T)$. As a consequence the size of the *dichotomic encoding* generated by Algorithm DichoEven is at most $4 * \mathbf{Dim_2}(T)$.

Theorem 32 shows that the computational complexity of the 2-dimension in the case of trees is different from the general case where there is no polynomial algorithm approximating the 2-dimension of any order within a constant factor unless $\mathcal{P} = \mathcal{NP}$. Experimentation on benchmarks [29] and small examples have shown that Algorithm Dicho performs very well and it seems that its approximation ratio is better than 4. It has lead us to state the following conjecture.

Conjecture 33 *Let T be a tree, then Algorithm* Dicho *generates bit-vector encodings approximating $\mathbf{Dim_2}(T)$ within a factor 2.*

The approximation ratio of Algorithm Dicho can not be lower since it is reached for instance when T is a comb tree where all the elements different from leaves have a degree equal to 2 and when its height tends to infinity (see Figure 12). For such a comb of height h, Algorithm Dicho generates a dichotomic encoding of size $2h$ whereas $\mathbf{Dim_2}(T) = h + 1$.

For now, we have only been able to prove that Conjecture 33 is true for trees where all elements have a degree at most 4.

We can prove that for any tree where all elements have a degree at most 4 there always exists an dichotomic encoding whose size approximates the 2-dimension within a factor 2. Thanks to Theorem 29, this involves that Conjecture 33 is true for these particular trees. The proof uses the next lemma.

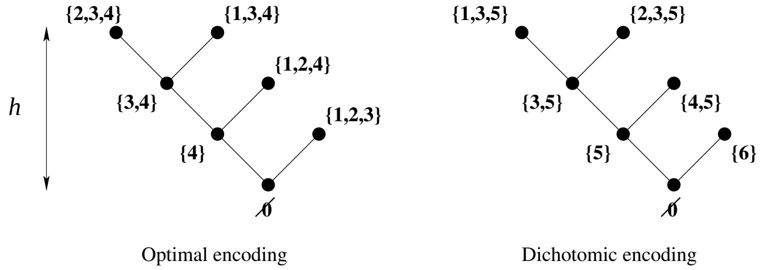

Fig. 12. A comb tree of height 3, an optimal bit-vector encoding and an encoding generated by Algorithm Dicho.

Lemma 34 *Let T be a tree such that its root has two children x_1 and x_2. Let $T_1 = T(x_1)$ (resp. $T_2 = T(x_2)$) be the subtree of T rooted at x_1 (resp. x_2). Let $d = \mathbf{Dim_2}(T)$, then $\mathbf{Dim_2}(T_1) = d - 1$ implies that $\mathbf{Dim_2}(T_1) \leq d - 3$.*

PROOF. Let ϕ be an optimal embedding of T into 2^S, $|S| = d$, and suppose that $\mathbf{Dim_2}(T_1) = d - 1$. Then the mapping ϕ_1 which associates with each element x of T_1 the code $\phi_1(x) = \phi(x) \backslash \phi(x_1)$ is a bit-vector encoding of size $|S| - |\phi(x_1)|$. Since $\phi(x_1) \neq \emptyset$ and $\mathbf{Dim_2}(T_1) = d - 1$, it implies that $|\phi(x_1)| = 1$. Thus $\phi(x_1) = \{a\}$ with $a \in S$.

Let's study the code of x_2. It is clear that $\phi(x_2) \neq \emptyset$. Suppose that $\phi(x_2) = \{b\}$, $b \in S$. As x_2 is incomparable to all the elements of T_1, we would have for all x element of T_1, $b \notin \phi(x)$. This would imply that ϕ_1 is in fact an embedding of T_1 into $2^{\{a,b\}}$, and thus $\mathbf{Dim_2}(T_1) \leq d-2$ which contradicts the hypothesis. Consequently we have $|\phi(x_2)| \geq 2$.

Now consider the mapping ϕ_2 defined for any element x of T_2 by $\phi_2(x) = \phi(x) \backslash \phi(x_2)$, this is a bit-vector encoding of T_2. Since x_1 is incomparable to all the elements of T_2, we have $a \notin \phi(x)$ for all x element of T_2. Thus ϕ_2 is in fact an embedding of T_2 into $2^{\{a\} \cup \phi(x_2)}$, of size $\leq d - 3$. The lemma is proved.

Figure 13 sums up the two kinds of distribution of the 2-dimension which are permitted when $\mathbf{Dim_2}(T) = d$ and its root has two children.

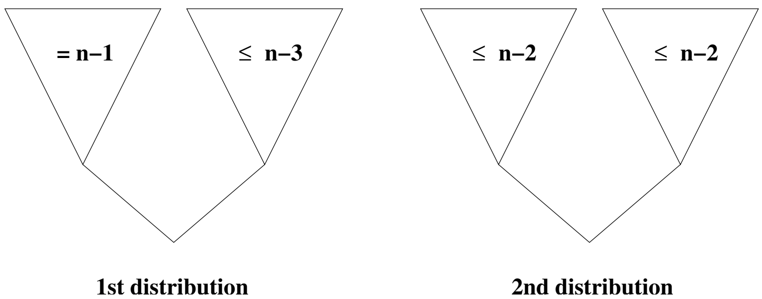

Fig. 13. The two possible distributions of the 2-dimension when $\mathbf{Dim_2}(T) = d$ and the root of T has two children.

Proposition 35 *Let T be a tree where all elements have a degree at most 4. Then there exists an dichotomic encoding whose size approximates the 2-dimension within a factor 2.*

PROOF. By induction on the height of the tree. If T is reduced to a single element, the empty code works. Otherwise we consider the worst case, namely when the root has four children x_1,x_2,x_3,x_4 (the others cases when there are fewer children can be easily deduced from this case). Let $T_1 = T(x_1)$, $T_2 = T(x_2)$, $T_3 = T(x_3)$, $T_4 = T(x_4)$ the four corresponding subtrees of T, with respective 2-dimensions equal to d_1,d_2,d_3,d_4. Let d be the 2-dimension of T.

Suppose that $d_1 \geq d_2 \geq d_3 \geq d_4$. By induction, there exist a dichotomic encoding for each tree T_i with size $\leq 2d_i$ ($1 \leq i \leq 4$). For greater convenience, we will consider these dichotomic encodings as embeddings into binary trees (see Proposition 28). Two cases occur:

- Suppose that $d_1 = d - 1$. Then the subtree consisting in T_1, T_2 and the root of T is a tree of 2-dimension at most d. Thus thanks to Lemma 34, $\mathbf{Dim_2}(T_2) \leq d - 3$. In the same way, $\mathbf{Dim_2}(T_3) \leq d - 3$ and $\mathbf{Dim_2}(T_4) \leq d - 3$. In that case, by using the embeddings of T_1,T_2,T_3,T_4 into binary trees and ending by the construction presented on Figure 14, we can build an embedding of T into a binary tree such that the corresponding dichotomic encoding is clearly of size at most $2d$. On the left of Figure 14 the 2-dimensions of the subtrees are indicated in the triangles representing them and on the right the label of an element indicates the number of colors used to encode the subtree rooted at this element.

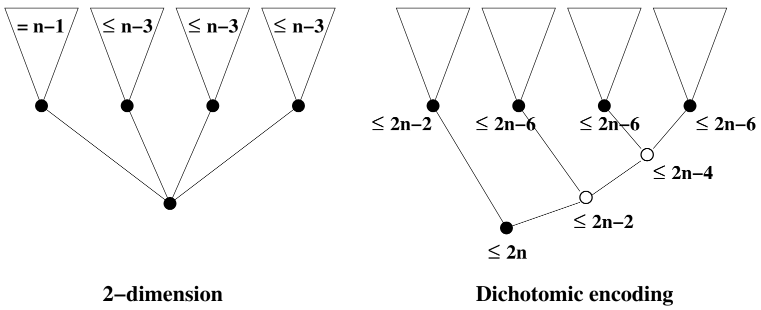

Fig. 14. First distribution of the 2-dimension and the associated embedding into a binary tree.

- Otherwise $d_1 \leq d-2$, which implies that $d_2 \leq d-2$, $d_3 \leq d-2$ and $d_4 \leq d-2$. In the same way as the previous case, we can build an embedding of T in a binary tree by using the embeddings of T_1,T_2,T_3,T_4 into binary trees and ending with a different construction which is presented on Figure 15. The notations on the figure are the same as the previous figure, and we can check that the construction leads to a dichotomic encoding of T of size at most $2d$.

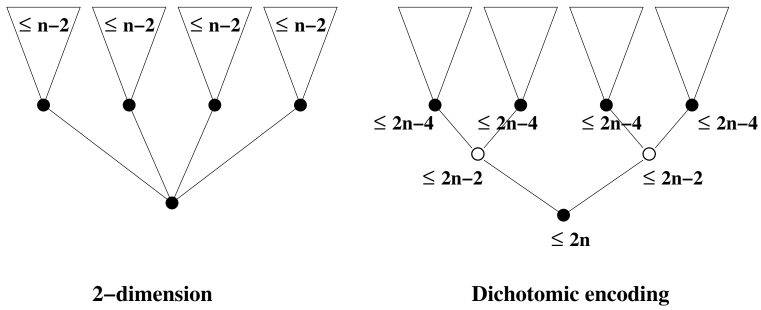

Fig. 15. Second distribution of the 2-dimension and the associated embedding into a binary tree.

Unfortunately this reasoning does not apply when some elements have a degree greater that 4. In order to ensure the existence of dichotomic encodings approximating the 2-dimension of any tree, we probably need a more general lemma concerning the distribution of the 2-dimension among the subtrees of the initial tree: we think that the approximation within a factor 2 is a consequence of a structural property of optimal embeddings into boolean lattices which is formulated in Conjecture 36.

Conjecture 36 *Let T be a tree and $x_1, ..., x_k$ the children of its root. Let $T_i = T(x_i)$ the subtree rooted at x_i, $1 \leq i \leq k$. Then* $2^{\mathbf{Dim_2}(T_1)} + \ldots + 2^{\mathbf{Dim_2}(T_k)} \leq 2^{\mathbf{Dim_2}(T)}$.

By using Proposition 30 and reasoning by induction, it easy to check that Conjecture 36 implies that with Algorithm DichoEven we have $algo(T) \leq \mathbf{Dim_2}(T)$ for any tree T, which leads to a proof of Conjecture 33.

7 Conclusion and open problems

This study of the computational complexity of the 2-dimension in the general case and for trees has provided some approximability results. We think it is possible to give worse non-approximability ratios for the general case by finding a more economical transformation between graph coloring and covering by complete bipartite orders. Moreover any improvement of the non-approximability ratios for graph coloring will also improve the non-approximability ratios for the 2-dimension.

Some recent improvements of Algorithm Dicho have been published in [15], but they do not compute the exact 2-dimension of trees. The computational complexity of the 2-dimension for trees remains an open problem, as well as for lattices. In light of the approximation results, we state the following conjecture.

Conjecture 37 *Computing the 2-dimension of trees can be done in polynomial time.*

We hope that such theoretical results will lead to the design of new heuristics to generate efficiently small bit-vector encodings of orders.

References

[1] I. Anderson. *Combinatorics of Finite Sets.* Clarendon Press, Oxford University Press, 1987.

[2] E. Artin. *The Gamma Function.* Holt, Rinehart and Winston, 1964.

[3] H. At-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, jan 1989.

[4] G. Behrendt. Maximal antichains in partially ordered sets. *Ars Combin.*, C(25):149–157, 1988.

[5] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability - towards tight results. *SIAM J. Comp.*, pages 804–915, 1998.

[6] A. Bouchet. *Etude combinatoire des ordonnés finis, Applications.* PhD thesis, Université scientifique et médicale de Grenoble, 1971.

[7] Y. Caseau. Efficient handling of multiple inheritance hierarchies. In *Proceedings of OOPSLA'93*, pages 271–287, 1993.

[8] Y. Caseau, M. Habib, L. Nourine, and O. Raynaud. Encoding of multiple inheritance hierarchies and partial orders. *Computational Intelligence*, 15:50–62, 1999.

[9] T. Cormen, C. Leiserson, and R. Rivest. *Introduction à l'algorithmique.* Dunod, 1994.

[10] B. A. Davey and H. A. Priestley. *Introduction to lattices and orders.* Cambridge University Press, 1991.

[11] B. Dushnik and E. W. Miller. Partially ordered sets. *Amer. J. Math.*, 63:600–610, 1941.

[12] G. Ellis. Efficient retrieval from hierarchies of objects using lattice operations. In *Conceptual graphs for knowledge representation (International Conference on Conceptual Structures)*, volume 699 of *Lecture Notes in Artificial Intelligence*, 1993.

[13] Engel. *Sperner Theory.* Cambridge University Press, 1997.

[14] A. Fall. The foundations of taxonomic encodings. *Computational Intelligence*, 14:598–642, 1998. Simon Fraser University Technical Report 94-20, 1994.

[15] R. E. Filman. Polychotomic encoding: A better quasi-optimal bit-vector encoding of tree hierarchies. In *Proceedings of ECOOP'2002*, pages 545–561, 2002.

[16] P. Fishburn and P. Hammer. Bipartite dimension and bipartite degrees of graphs. Technical Report 76, DIMACS, 1993. Présent à l'adresse http://dimacs.rutgers.edu/TechnicalReports.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[18] M. Habib and L. Nourine. Bit-vector encoding for partially ordered sets. In *Proceedings of ORDAL'94*, LNCS 831, pages 1–12, 1994.

[19] M. Habib and L. Nourine. A new dimensional parameter and a removal theorem for boolean dimension. In *Proceedings of the 4th Twente Workshop on Graphs and Combinatorial Optimization*, pages 127–132, 1995.

[20] M. Habib, L. Nourine, and O. Raynaud. A new lattice-based heuristic for taxonomy encoding. In *Proceedings of KRUSE'97*, pages 60–71, 1997.

[21] H. A. Kierstead. On the order dimension of 1-sets versus k-sets. *J. of Comb. Theory*, 73 Series A:219–228, 1996.

[22] A. Krall, J. Vitek, and R.N. Horspool. Near optimal hierarchical encoding of types. In *Proceedings of ECOOP'97*, pages 128–145, 1997.

[23] G. Markowski. The representation of posets and lattices by sets. *Algebra Universalis*, 11:173–192, 1980.

[24] M. Morvan and L. Nourine. Simplicial elimination schemes, extremal lattices and maximal antichain lattices. *Order*, 13(2):159–173, 1996.

[25] H. Müller. On edge perfectness and classes of bipartite graphs. *Discrete Mathematics*, 149:159–187, 1996.

[26] V. Novak. On the pseudo-dimension of ordered sets. *Czechoslovak Math. Journal*, 13:587–598, 1963.

[27] O. Ore. Theory of graphs. *AMS Colloq. Publ.*, 38, 1962.

[28] J. Orlin. Contentment in graph theory: covering graphs with cliques. unpublished manuscript, 1976.

[29] O. Raynaud and E. Thierry. A quasi optimal bit-vector encoding of tree hierarchies. application to efficient type inclusion tests. In *Proceedings of ECOOP'2001*, LNCS 2072, pages 165–180, 2001.

[30] K. Reuter. The jump number and the lattice of maximal antichains. *Discrete Mathematics*, 88:289–307, 1991.

[31] H. U. Simon. On approximate solutions for combinatorial optimization problems. *SIAM J. Disc. Math.*, 3:294–310, 1990.

[32] M. Skorsky. *Endliche Verbände Diagramme und Eigenschaften.* PhD thesis, Darmstad, Germany, 1992. (écrit en anglais).

[33] E. Sperner. Ein satz uber untermengen einer endlichen menge. *Math. Z.*, 27:544–548, 1928.

[34] J. Stahl. On the 2-dimension of the crown s_k^n. Technical Report 800, TH Darmstad, 1984.

[35] J. Stahl and R. Wille. Preconcepts and set representation of contexts. In *Proc. Universal Algebra*, Sienna, 1984.

[36] E. Thierry. *Sur quelques interactions entre structures de données et algorithmes efficaces pour les ordres et les graphes.* PhD thesis, LIRMM, Université Montpellier II, 2001.

[37] T. Trotter. Graphs and partially ordered sets: recent results and new directions. Technical report, Arizona state university, 1995.

[38] W. T. Trotter. Embedding finite posets in cubes. *Discrete Mathematics*, 12:165–172, 1975.

[39] W. T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory.* John Hopkins University Press, Baltimore, 1991.

[40] J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.

[41] J. Vitek, R.N. Horspool, and A. Krall. Efficient type inclusion tests. In *OOPSLA'97*, pages 142–157, 1997.

[42] R. Wille. Restructuring lattice theory: an approach based on hierarchies of contexts. In I. Rival, editor, *Ordered Sets*, number 83 in NATO ASI, pages 445–470. 1982.

Appendix

A. Bounds for Sperner numbers $sp(n)$.

Sperner numbers $sp(n)$ are defined for any integer $n \geq 1$ by the formula $sp(n) = \min\{k | \binom{k}{\lfloor k/2 \rfloor} \geq n\}$. They come up in the study of the 2-dimension since the 2-dimension of an antichain with n elements is exactly equal to $sp(n)$ (see Proposition 9) and they also appear in the conception of some heuristics (see Theorem 26). Thus a natural problem is to evaluate these numbers in order to compute them efficiently.

In order to obtain an asymtotic approximation of $sp(n)$, we can use a Stirling formula such as $k! = \sqrt{2\pi k}\ 2^{-k} k^k (1 + O(1/k))$. This formula enables to approximate the binomial coefficient $\binom{k}{\lfloor k/2 \rfloor}$ and eventually leads to a first evaluation: $sp(n) = log_2(n) + log_2 log_2(n)/2 + O(1)$. This evaluation may be improved with the following bounds.

Proposition 10. Let $n \geq 2$ and $sp(n) = \min\{k | \binom{k}{\lfloor k/2 \rfloor} \geq n\}$. Then

$$\log_2(n) + \log_2 \log_2(n)/2 \leq sp(n) \leq \log_2(n) + \log_2 \log_2(n)/2 + 2.$$

PROOF. We use the next inequality that can be found in [2]. For all real $x > 0$ and integer $k \geq 1$,

$$\Gamma(x) \frac{k}{x+k} \leq \frac{k^x k!}{x(x+1)\dots(x+k)} \leq \Gamma(x),$$

where Γ is the *Gamma function*. We apply this inequality for $x = 1/2$, knowing that $\Gamma(1/2) = \sqrt{\pi}$. It gives

$$\sqrt{\pi} \frac{k}{\frac{1}{2}+k} \leq \frac{k^{1/2} k! 2^{k+1}}{1.3.\dots.(2k+1)} \leq \sqrt{\pi}.$$

This inequality can be reformulated as

$$\sqrt{\pi} \frac{k}{\frac{1}{2}+k} \leq k^{1/2} 2^{2k+1} \frac{k!k!}{(2k+1)!} \leq \sqrt{\pi} \tag{1}$$

This inequality can be rewritten in two versions in order to introduce binomial coefficients:

$$\sqrt{\pi} \frac{k}{\frac{1}{2}+k} \leq k^{1/2} \frac{2^{2k+1}}{k+1} \frac{1}{\binom{2k+1}{k}} \leq \sqrt{\pi} \tag{2}$$

$$\sqrt{\pi}\frac{k}{\frac{1}{2}+k} \le k^{1/2}\frac{2^{2k+1}}{2k+1}\frac{1}{\binom{2k}{k}} \le \sqrt{\pi} \tag{3}$$

First we prove the lower bound of Proposition 10.

Claim 1 For all $k \ge 1$ and $n \ge 2$, $\binom{2k}{k} \ge n \implies 2k \ge \log_2(n)+\log_2\log_2(n)/2$.

It is sufficient to prove for all $k \ge 1$ that $2k \ge \log_2\binom{2k}{k} + \frac{1}{2}\log_2\log_2\binom{2k}{k}$. Thanks to inequality (3), we have $\binom{2k}{k} \le \frac{2^{2k}}{\sqrt{\pi k}}$. Thus

$$\begin{aligned}
&2k - (\log_2\binom{2k}{k} + \frac{1}{2}\log_2\log_2\binom{2k}{k})\\
&\ge 2k - \log_2(\frac{2^{2k}}{\sqrt{\pi k}}) - \frac{1}{2}\log_2\log_2(\frac{2^{2k}}{\sqrt{\pi k}})\\
&\ge 2k - (2k - \frac{1}{2}\log_2 k - \log_2\sqrt{\pi}) - \frac{1}{2}\log_2(2k - \frac{1}{2}\log_2 k - \log_2\sqrt{\pi})\\
&\ge \frac{1}{2}\log_2 k + \log_2\sqrt{\pi} - \frac{1}{2}\log_2(2k)\\
&\ge \log_2\sqrt{\pi} - \frac{1}{2} > 0
\end{aligned}$$

Claim 2 For all $k \ge 1$, $n \ge 2$, $\binom{2k+1}{k} \ge n \implies 2k+1 \ge \log_2(n) + \log_2\log_2(n)/2$.

If $\binom{2k+1}{k} \ge n$, then $\binom{2k+2}{k+1} = 2\binom{2k+1}{k} \ge 2n$. Thanks to Claim 1, it implies that $2k+2 \ge \log_2(2n) + \log_2\log_2(2n)/2$. And finally $2k+1 \ge \log_2(n) + \log_2\log_2(n)/2$.

Claim 1 and Claim 2 can be used to deal with $sp(n)$ even or odd, which achieves to prove the lower bound for $sp(n)$. Now we prove the upper bound of Proposition 10.

Claim 3 For all $k \ge 1$ and $n \ge 2$, $\binom{2k+1}{k} < n \implies 2k+2 \le \log_2(n) + \log_2\log_2(n)/2 + 2$.

It is sufficient to prove for all $k \ge 1$ that $2k+2 \le \log_2(\binom{2k+1}{k}+1) + \frac{1}{2}\log_2\log_2(\binom{2k+1}{k}+1)+2$. Thanks to inequality (2), we have $\binom{2k+1}{k} \le \frac{k^{1/2}2^{2k+1}}{\sqrt{\pi}(k+1)}$. Thus

$$\begin{aligned}
&\log_2\binom{2k+1}{k}+\frac{1}{2}\log_2\log_2\binom{2k+1}{k}+2-(2k+2)\\
&\geq \log_2(\frac{k^{1/2}2^{2k+1}}{\sqrt{\pi}(k+1)})+\frac{1}{2}\log_2\log_2(\frac{k^{1/2}2^{2k+1}}{\sqrt{\pi}(k+1)})-2k\\
&\geq 1-\log_2\sqrt{\pi}+\frac{1}{2}\log_2 k-\log_2(k+1)+\frac{1}{2}\log_2(2k+1-\log_2\sqrt{\pi}+\frac{1}{2}\log_2 k-\log_2(k+1))\\
&\geq \frac{3}{2}-\log_2\sqrt{\pi}+\frac{1}{2}\log_2 k-\log_2(k+1)+\frac{1}{2}\log_2(k+\frac{1}{2}\log_2\sqrt{\pi}+\frac{1}{4}\log_2 k-\frac{1}{2}\log_2(k+1))
\end{aligned}$$

Consider this last line as a function of k that we denote by $f(k)$. A careful study (using derivation) shows that this function f is non-decreasing for all $k>0$ and that it converges to $\frac{3}{2}-\log_2\sqrt{\pi}$ when k tends to infinity. It can be checked that $f(3)>0$ and therefore for all $k\geq 3$, $f(k)>0$ which implies that $2k+2\leq \log_2(\binom{2k+1}{k}+1)+\frac{1}{2}\log_2\log_2(\binom{2k+1}{k}+1)+2$. The cases when $k=1$ and $k=2$ can be checked independently.

Claim 4 For all $k\geq 2$ and $n\geq 2$, $\binom{2k}{k}<n \implies 2k+1\leq \log_2(n)+\log_2\log_2(n)/2+2$.

If $\binom{2k}{k}<n$, then $\binom{2k-1}{k-1}=\binom{2k}{k}/2<n/2$. Thanks to Claim 3, it implies that $2k\leq \log_2(n/2)+\log_2\log_2(n/2)/2$. And finally $2k+1\geq \log_2(n)+\log_2\log_2(n)/2$.

Claim 3 and Claim 4 can be used to deal with $sp(n)$ even or odd, which achieves to prove the upper bound for $sp(n)$.

Note that the proofs have also given the asymptotic approximation of $sp(n)$:

$$\log_2(n)+\frac{\log_2\log_2(n)}{2}+\log_2\sqrt{\pi}-\frac{1}{2}\leq sp(n)\leq \log_2(n)+\frac{\log_2\log_2(n)}{2}+\log_2\sqrt{\pi}+\frac{1}{2}+o(1)$$

These bounds are really close together. And we may hope to find a simple closed formula to compute exactly $sp(n)$ from n.

B. Relations between $\mathbf{Dim_2}$ *and* $\mathbf{Dim_k}$*.*

Some ratios between $\mathbf{Dim_2}$ and $\mathbf{Dim_k}$ have been presented in Section 4. We prove them briefly.

Proposition 38 *Let $k\geq 2$ be an integer and P an order. Then* $\mathbf{Dim_k}(P)\leq \mathbf{Dim_2}(P)\leq k*\mathbf{Dim_k}(P)$.

PROOF. We consider here embeddings of P into products of chains. An k-embedding of dimension d of $P = (X, \leq_P)$ is a mapping ϕ from X into $[1, k]^d$ (where $[1, k]$ is the set of all integers between 1 and k) defined by $\phi(x) = (x_1, \ldots, x_d)$ and such that $x \leq_P y$ iff $x_i \leq y_i$ for all $1 \leq i \leq d$. A bit-vector encoding of P of size d is exactly a 2-embeddding of dimension d. As any 2-embedding may be viewed as a k-embedding, we have $\mathbf{Dim_k}(P) \leq \mathbf{Dim_2}(P)$. And if we consider a k-embedding ϕ of dimension d, we can construct a 2-embedding ϕ' of dimension $k * d$ by replacing $\phi(x) = (x_1, ..., x_i, ..., x_d)$ by $\phi'(x) = (x_{1,1}, ..., x_{1,k}, ..., x_{i,1}, ..., x_{i,k}, ..., x_{d,1}, ..., x_{d,k})$ where $(x_{i,1}, ..., x_{i,k}) = (0, ..., 0, \underbrace{1, ..., 1}_{x_i \text{times}})$. This implies that $\mathbf{Dim_2}(P) \leq k * \mathbf{Dim_k}(P)$.