Encoding Partial Orders Through Modular Decomposition

Laurent Beaudou[1], Kaoutar Ghazi[2], Giacomo Kahn[3], Olivier Raynaud[4] and Eric Thierry[5]
{laurent.beaudou@univ-bpclermont.fr[1], ghazi@isima.fr[2], giacomo.kahn@univ-bpclermont.fr [3]
raynaud@isima.fr [4], eric.thierry@ens-lyon.fr[5]}

Limos laboratory, Blaise Pascal University, 63000 Clermont-Ferrand, France[1, 2, 3, 4]
LIP laboratory, University of Lyon, ENS Lyon, France[5]

Abstract. A well-known method to represent a partially ordered set P consists in associating to each element of P a subset of a fixed set S ={1,...,k} such that the order relation coincides with subset inclusion. Such an embedding is called a bit-vector encoding of P and is economical with space. As a consequence, they have found applications in knowledge representation, distributed computing or object-oriented programming. The smallest size of such an encoding is called the 2-dimension of P and its computation is known to be NP-hard in the general case [12]. Finding heuristics which provide compact encodings is challenging and it has yielded many works. Our paper presents a new heuristic through modular decomposition. This unified process is a 4-approximation for rooted trees 2-dimension and provides reduced encoding by 40% for series-parallel posets. It reaches or improves the best results for general posets.

Keywords: Bit-vector encodings; 2-dimension; Modular decomposition, Series-parallel partial orders; Embeddings; Heuristics.

1 Introduction

Partially ordered sets (posets for short) occur in numerous fields of computer science, like distributed computing, programming languages, databases or knowledge representation. Such applications have raised the need for storing and handling them efficiently. Many ways of encoding partially ordered sets have been proposed in the literature. Depending on the purpose, several criteria are commonly considered to guide the choice of the most appropriate encoding. One may cite the compromise between speeding up operations and saving space, the choice between dynamic or static data structures with regard to possible modifications of the order, the complexity of generating the encoding from usual data structures (like matrices or lists of successors), the restrictions on the data structures imposed by hardware and software (e.g. storing the order in a database which can be then accessed only by means of SQL requests). Performing fast comparisons between elements while saving space is the most usual issue.

Here is a non-exhaustive list of approaches that have been studied: numbering the elements in order to compress their lists of successors [1], partitioning the order into nice subsets like antichains [7, 22, 16] or chains [2, 7, 13], mixing numbering and partitioning [10, 23], describing the order as the union of nice orders on the same set of elements [3], focusing on lattice operations [20], embedding the order into one which is known to have a nice representation [18].

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 a poset, denoted by P if there is no ambiguity. 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) such that for all x and y in X, x $\leq_P$ y if and only if $\varphi(x) \subseteq \varphi(y)$. The size of the encoding φ is the cardinal of S and we will refer to the elements of S as colors of the encoding. A classical implementation of bit-vector encodings associates to each element x a vector V_x of |S| bits where the ith bit is equal to 1 for i in $\varphi(x)$ 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. Figure 1 illustrates different representations of such embeddings.

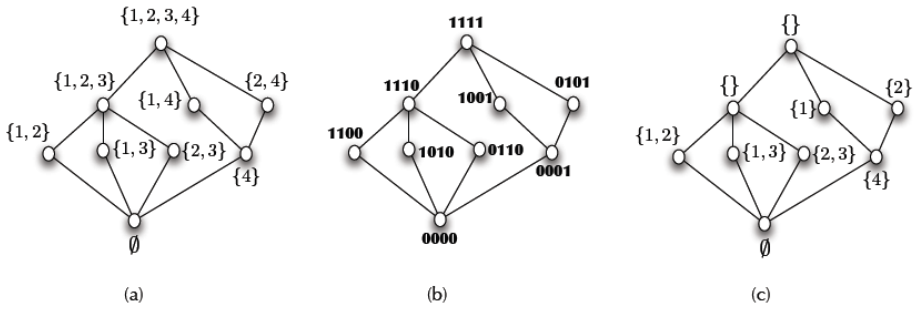

Fig. 1. Three descriptions of the same bit-vector encoding of P. On the left the embedding $X \rightarrow 2^{\{1,2,3,4\}}$, in the center $X \rightarrow \{0,1\}^4$, on the right a reduced encoding of P.

Given an order P, the smallest size of a bit-vector encoding of P is called the 2-dimension of P and denoted $Dim_2(P)$. Originally defined by V. Novak in 1963 ([15]), this parameter has yielded many studies in computer science. Its computation is known to be NP-hard in the general case (see [12] for a survey) and the assets of bit-vector encodings have urged to design good heuristics for applications. Beyond algorithms for the general case [5, 11, 22], the class of rooted trees has been specifically studied by several authors ([6, 8, 17]).

Our contribution is organized as follows: in section 2 we give a description of previous approaches and definitions about the modular decomposition. In Section 3 we describe series-parallel posets and our heuristic. We compare its performance on synthetic datasets and rooted trees. Then in Section 4, we describe an original blow up operator used to extend our heuristic to the whole class of posets and we analysis its performance on benchmarks traditionally used in this context.

2 Preliminaries

2.1 Historical approaches

Today, the best approach to compute the encoding of a given poset can be done in two steps. A first preprocessing step, called *Splitting & Balancing* (SB), that splits and balances the poset ([4], [14]) and a second effective step, called *Simple Coloring* (SC), that computes the encoding of the resulted order through a graph coloring strategy ([5]). We denote this approach SBSC.

Splitting & balancing In [14], authors proposed a preprocessing step to compute the size of a bit vector encoding of a poset. By referring to the work of Caseau in [4], they propose to split the set of children of a node if this set is too large. Indeed, Caseau showed that this splitting can easily be done by adding some additional nodes in the hierarchy so that the encoding is smaller. Then, the balancing consists in adding splitting nodes so that the hierarchy remains as balanced as possible (see Figure 2).

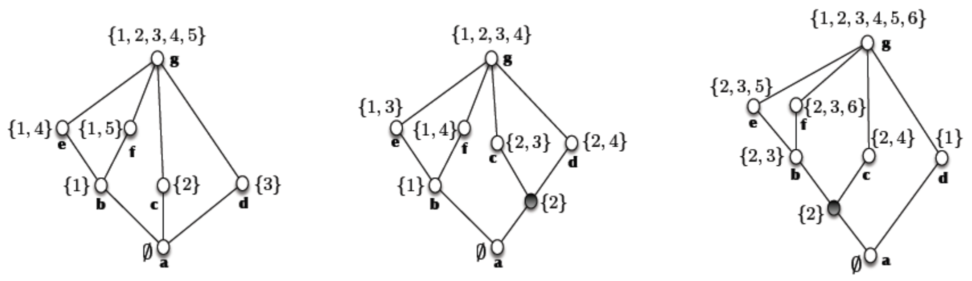

Fig. 2. On the left, an encoding of P of size 5. One new color is assigned to each internal node. In the middle, a new node is added to split children of the node a, P is balanced and the size of the encoding is 4. By introducing a new node in the right spot, the number of needed colors is reduced. On the right, a new node is added to split children of the node a, P is not balanced and the size of the encoding is 6.

Simple coloring In [5], authors show how to reduce the bit-vector encoding problem in a graph coloring problem. We need to introduce few definitions.

Definition 1 (Join/meet irreducible elements) Given a poset P and an element x in P, the set $\uparrow x$ (resp. $\downarrow x$) denotes all elements of P which are greater (resp. smaller) than or equal to x. An element x in P is join-irreducible if there exists y such that y is in $P \setminus \uparrow x$ and for all z in P, if $z \leq x$ then $z = x$ or $z \leq y$. Meet irreducible elements are defined dually. The set of join irreducible (resp. meet-irreducible) elements of P is denoted J(P) (resp. M(P)).

Definition 2 (Critical pairs) Given two elements x and y in P, we say that (x,y) is a critical pair if y is not in $\uparrow x$ and $\downarrow x \setminus \{x\} \subseteq \downarrow y$ and $\uparrow y \setminus \{y\} \subseteq \uparrow x$.

Two join irreducible elements j and j\` are said to be in conflict if there exists m in M(P) such that (j,m) is a critical pair and j\` $\leq$ m or (j\`,m) is a critical pair and j $\leq$ m. The conflict graph of P denoted $G_{conflict}(P)$ has J(P) as vertex set and an edge jj\` if j and j\` are in conflict.

Proposition 1 (Simple coloring and the 2-dimension [5]) Given P, $Dim_2(P) \leq \chi(G_{conflict}(P))$.

Actually, join-irreducible elements are the only elements which need their own color. Other elements will be assigned some colors by inheritance in the initial poset. Moreover, Proposition 1 states which join-irreducibles can share the same own color. See Figure 3 for an example.

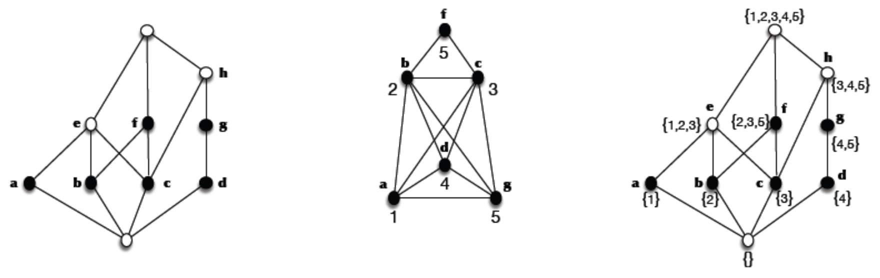

Fig. 3. On the left a given poset whose join-irreducible elements are in black. In the middle its conflict graphs $G_{conflict}$ and an associated coloration. On the right, an encoding of the poset using the given coloration.

2.2 Modular decomposition

We present the modular decomposition process used in our heuristic.

Definition 3 (Modules) Given a poset P, a module of P is a subset M of P such that any two elements of M have the same comparison relation with elements of P \ M. Formally,

$$\forall\, x, x` \in M, \forall\, y \in P \ \backslash M, (x \leq y \ \Leftrightarrow x` \leq y)\ and\ (y \leq x \ \Leftrightarrow y \leq x`)$$

A non empty module M is strong if every other module is a subset of M or a superset of M or does not intersect M. Gallai [9] proved that maximal strong modules (not equal to P) form a partition of P called the modular partition. See Figure 4 for an illustration of the last definitions.

Definition 4 (Quotient) Let P = (X,P) be a poset and M = $\{M_1,\ldots M_\ell\}$ be the modular partition of P. The quotient poset $P_{/M}$ is defined on the ground set M and $M_i \leq_{P/M} M_j$ if there are x in M_i and y in M_j, such that $x \leq_{\boldsymbol{P}} y$ (i.e. each strong module is shrinked into a single element).

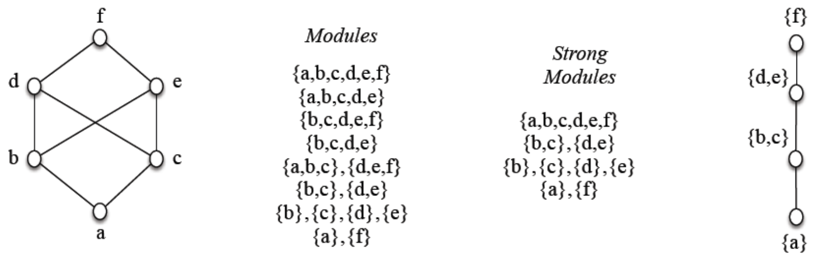

Fig. 4. On the left, a poset P = (X,$\leq_P$) defined on the ground set X = {a,b,c,d,e,f}. In the middle, the lists of its modules and its strong modules. On the right its quotient graph $P_{/M}$ with M= {{a},{b,c},{d,e},{f}} its corresponding modular partition.

Definition 5 (Chain/Antichain) A poset P is a chain (resp. antichain) if for all x and y in P, x and y are (resp. are not) comparable, i.e. $x \leq_P y$ or $y \leq_P x$.

Definition 6 (Modular decomposition) Let P be a poset, the modular decomposition of P is the rooted tree denoted T_P of its strong modules with inclusion as ancestor relation. The leaves are exactly the elements of P. Each inner node corresponds with a strong module whose induced poset can be further decomposed. If its quotient by its modular partition is a chain (resp. an antichain, resp. another type of poset), the node is called a series (resp. parallel, resp. prime) node.

Definition 7 (Series-parallel posets) A poset is called series-parallel if all the inner nodes of its modular decomposition are either series or parallel. See Figure 5 for an illustration.

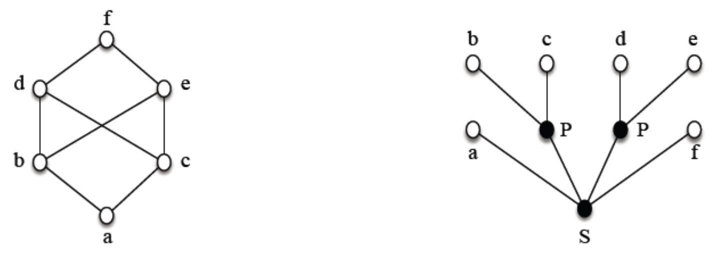

Fig. 5. On the left, a poset P. On the right its modular decomposition tree. S stands for series node and P for parallel node. Since there is no prime node, P is series-parallel.

3 Series-parallel posets

3.1 The series operation and the 2-dimension

The 2-dimension of chains is easy to compute.

Proposition 2 (Folklore) Given a chain P = (X,$\leq_P$) of n elements, then Dim_2(P) is n-1. Let x_0, x_1 ,…, x_{n-1} be the n elements of P ordered by $x_0 <_P x_1 <_P \dots <_P x_{n-1}$, then an optimal bit-vector encoding φ using colors from S = {1,…, n-1} is given by φ (x_0) = 0 and φ (x_i) = {1,…, i} for all $1 \leq i \leq$ n-1.

This proposition can be generalized to posets whose quotients by their modular decomposition are chains.

Proposition 3 (Series node) Let (P,$\leq_P$) be a poset whose modular partition is {M_1 ,.., M_ℓ} and the quotient poset is the chain {M1 < … < M_ℓ}. Then

$$Dim_2(P) = \sum_{1 \leq i \leq \ell \mid |M_i \geq 2} Dim_2(M_i) + |\{1 \leq i \leq \ell | |M_i| = |M_{i-1}| = 1|$$

3.2 The parallel operation and the 2-dimension

Proposition 4 (Sperner [19]) Given an antichain P = (X, $\leq_P$) with n elements, then $Dim_2(P)$ = sp(n) where sp(n) = min{k | $\binom{k}{\frac{k}{2}} \geq n$}. An optimal bit-vector of P is obtained by associating with each x in P a combination of $\frac{sp(n)}{2}$ elements from S = {1,…,sp(n)}.

In [6], authors introduced the generalized polychotomic algorithm, denoted GP, which is a 4-approximation of the 2-dimension of rooted trees. Thanks to Proposition 4, one can design an algorithm to approximate the 2-dimension of posets whose quotients by their modular decompsition are antichains.

Proposition 5 (Parallel node [6]) Let P = (X, $\leq_P$) be a poset whose modular partition is {$M_1,\ldots,M_\ell$} and its quotient is the antichain <$M_1,\ldots,M_\ell$>. Then, from the sequence <weight(M_1),…,weight(M_ℓ)>, where weight(M_i), for $1 \leq i \leq \ell$, is the size of the encoding associated with each induced poset M_i, the GP algorithm computes a bit-vector encoding of P.

3.3 Heuristic description

Thanks to Propositions 3 and 5 we are able to design an algorithm to compute the size of a bit-vector encoding of a given poset P. Each module which is a leaf of T_P is assigned a weight equal to 0. Then, by a recursive process in T_P, when all sub-modules $M_1,\ldots,M_\ell$ of a given module M are assigned a weight, depending on its type (S or P), we compute the own weight of M (see Figure 6):

- If type(M) = S then weight(M) = $\sum_{1\leq i\leq \ell \mid |M_i|\geq 2} Dim_2(M_i) + |\{1 \leq i \leq \ell||M_i| = |M_{i-1}| = 1|$
- If type(M) = P then weight(M) = GP(<weight(M_1),…,weight(M_ℓ)>)

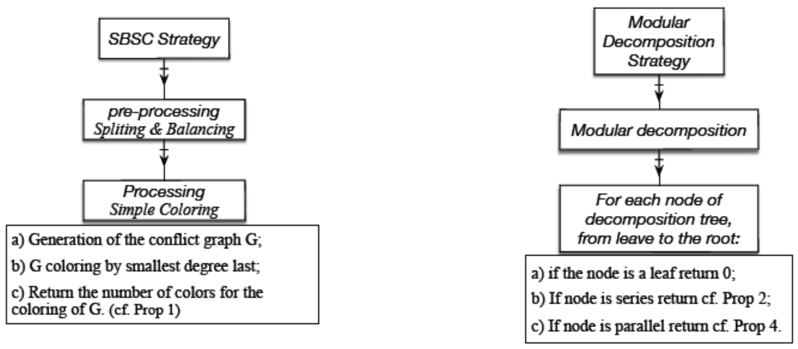
SBSC Strategy
pre-processing Spliting & Balancing
Processing Simple Coloring
a) Generation of the conflict graph G;
b) G coloring by smallest degree last;
c) Return the number of colors for the coloring of G. (cf. Prop 1)
Modular Decomposition Strategy
Modular decomposition
For each node of decomposition tree, from leave to the root:
a) if the node is a leaf return 0;
b) If node is series return cf. Prop 2;
c) If node is parallel return cf. Prop 4.

Fig. 6. On the left the referral two steps process SBSC. On the right the modular decomposition process.

3.4 Theoretical result

Proposition 6 states that the modular decomposition strategy (MD) maintains the 4-approximation result of the generalized polychotomic algorithm for the class of rooted trees which are series-parallel posets.

Proposition 6 Let T be a tree. MD(T) is a 4-approximation of $Dim_2(T)$.

3.5 Experimental results

Regrettably, no natural hierarchy corresponds to formal series-parallel posets. Thus, some evaluations are done on synthetic datasets that have been generated with parameters such as the number of elements, depth and maximal degree of the poset. Some other evaluations are done on known rooted tree hierarchies (see Table 1).

Table 1. First columns give descriptors of posets, column SBSC gives the sizes of the encoding with the referral SBSC process. Last columns give the size with the modular decomposition strategy and with GP Algorithm on tree hierarchies.

Dataset	Size	Depth	Max parents	SBSC	MD	
Synthetic data						
	200	13	64	49	34	
	200	29	4	49	47	
	1000	22	140	90	56	
	1000	19	32	53	39	
	10000	6	6222	32	19	
	10000	4	2	27	17	
Rooted trees						GP
VisualWorks	1956	15	1	50	19	19
Digitalk3	1357	14	1	36	26	26
NextStep	311	8	1	23	17	17
ET++	371	9	1	30	19	19

Results show the efficiency of our heuristic on series-parallel posets with an improvement rate around 40% for posets with a significant width. Moreover, in accordance with Proposition 6, our heuristic find the same values than GP Algorithm on rooted tree hierarchies.

4 General posets

4.1 The prime operation and the 2-dimension

A prime node of a modular decomposition tree corresponds to a suborder P, let M be its modular decomposition. Intuitively, Dim_2 (P) can't be higher than $Dim_2(P_{/M})$ plus the sum of its own suborders' 2-dimension [24]. Actually, Proposition 7 provides a finer upper bound by stating that most of these suborders can share the same colors.

Proposition 7 (Bounds on the 2-dimension) Let (P,$\leq_P$) be a poset and M its modular partition. Then

$$Dim_2(P) \leq Dim_2(P_{/M}) + \max_{\text{C chain of } P_{/M}} \sum_{M \in C} Dim_2(M)$$

4.2 Blow up operator

We define below a new operator called Blow up to apply to the quotient order. Thanks to this operator we will be able to provide a tighter upper bound of the 2-dimension of posets.

Definition 8 (Blow up) Let (P,$\leq_P$) be a poset, M be its modular partition and $P_{/M}$ the associated quotient. The blow up of P, denoted B(P), is the poset obtained from $P_{/M}$ by substituting each module M in M with a chain of $Dim_2(M) + 1$. See Figure 7.

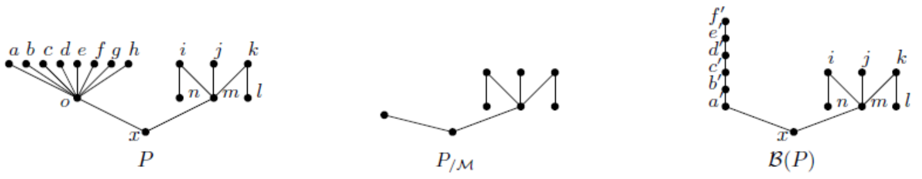

Fig. 7. On the left, a poset P whose modular partition is M = {{o, a, b, c, d, e, f, g; h}, {x}, {i}, {j}, {k}, {l},{m}, {n}}. On the right $P_{/M}$ and B(P).

Proposition 8 (Blow up bounds) Let (P,$\leq_P$) be a poset, M be its modular partition and B(P) its blow up. Then

$$\mathrm{Dim}_2(\mathrm{P}) \leq \mathrm{Dim}_2(\mathrm{B(P)}) \leq \mathrm{Dim}_2(\mathrm{P}_{/\mathrm{M}}) + \max_{\text{C chain of } P_{/M}} \sum_{M \in C} Dim_2(M)$$

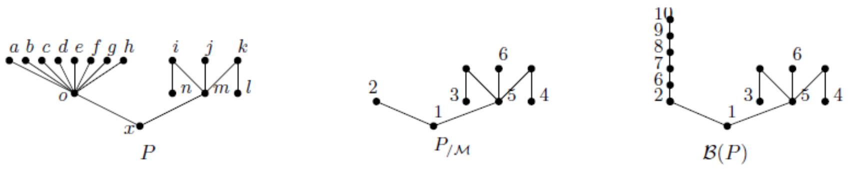

Fig. 8. On the left, a poset P with a non trivial module {o,a,b,c,d,e,f,g,h} requiring 5 bits for its encoding. On the center its quotient whose encoding requires 6 bits. On the right its blow up whose encoding requires 10 bits (Prop. 8) instead of 11 (Prop. 7).

4.3 Heuristic description

The principle of our heuristic is the same as described in Section 3.3 with additional rules to deal with prime nodes. For such a node P, we apply the Simple coloring to B(P) and then return the chromatic number of the corresponding conflict graph $G_{conflict}(B(P))$. However, first results showed that applying the blow up operator to prime nodes is not sufficient to reach the smallest encoding providing by the referral approach SBSC. Nevertheless, by introducing the Splitting & Balancing pre-processing in our own process, we are able to produce the shortest bit vector encodings. We have two options to introduce this pre-processing. We can execute it on the initial poset and then launch the modular decomposition (Global Split & Balance). Or we can launch the modular decomposition and execute the pre-processing to deal with each inner prime node (Local Split & Balance), (see Figure 9).

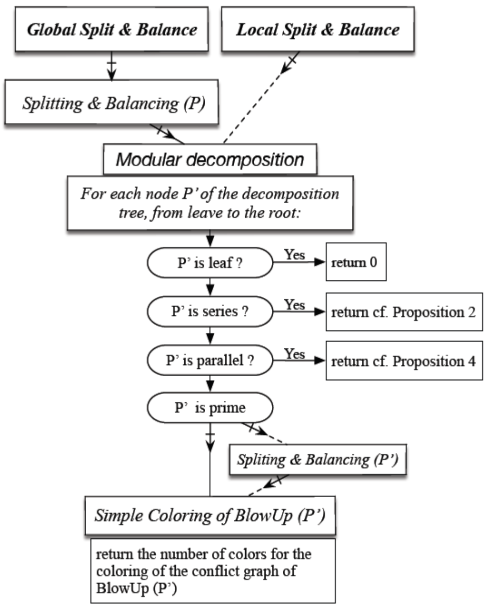
Global Split & Balance
Local Split & Balance
Splitting & Balancing (P)
Modular decomposition
For each node P' of the decomposition tree, from leave to the root:
P' is leaf ?
Yes
return 0
P' is series ?
Yes
return cf. Proposition 2
P' is parallel ?
Yes
return cf. Proposition 4
P' is prime
Spliting & Balancing (P')
Simple Coloring of BlowUp (P')
return the number of colors for the coloring of the conflict graph of BlowUp (P')

Fig. 9. Test protocols

4.3 Experimental results

Our heuristic has been applied on class libraries having poset structures ([14]). We have introduced in this list the recent Java8 hierarchy. Characteristics of benchmarks are given in Table 2 and representation of the Java8 hierarchy in Figure 10. From results given in Table 3 we can see that by applying the blow up processing with the one or the other option we reach the smallest known encoding or we improve it.

Note that these benchmarks have been studied for a long time and thus results from referral approach SBSC should be very closed to the optimum. For this reason, it is our understanding that to win few bits for each benchmark stay a challenge. Moreover, our heuristic was first designed for series-parallel partial orders and it is a nice result that this approach is also the most efficient for the whole class of posets and improves by 20% some of the benchmarks.

Table 2. Benchmarks characteristics.

Data	Size	Depth	Max parents	Max Children
Unidraw	613	10	2	147
Self	1801	18	9	232
Love-ed	436	10	10	78
Laure	295	12	3	8
Geode	1318	14	16	323
Ed	434	11	7	78
Java	225	7	3	112
Java8	17086	11	17	4621

Table 3. First column states the name of hierarchies. Second column gives results of the referral approach and the two last columns give results of our heuristic with the Global Split & Balance and the Local Split & Balance options.

Data	SBSC	Global S&B	Local S&B
Unidraw	30	30	24
Self	53	53	52
Love-ed	54	54	58
Laure	23	23	23
Geode	89	89	97
Ed	50	50	53
Java	19	19	16
Java8	84	81	78

5 Conclusion

Computing optimal bit-vector encodings was proved non-approximable and NP-hard in [12], and the best heuristics rely on another hard problem, that is graph coloring [14, 5]. Nonetheless the class of trees, which includes chains and antichains, was shown to be approximable [12, 6]. In this paper, we show how to combine these heuristics for trees and the reduction to graph coloring to compute bit-vector encodings for general posets. Based on the modular decomposition, this heuristic uses tree ideas for series and parallel inner nodes, our original blow up operator and colorings for prime inner nodes. With two options depending on the preprocessing step, it outperforms previously known algorithms, especially for series-parallel posets.

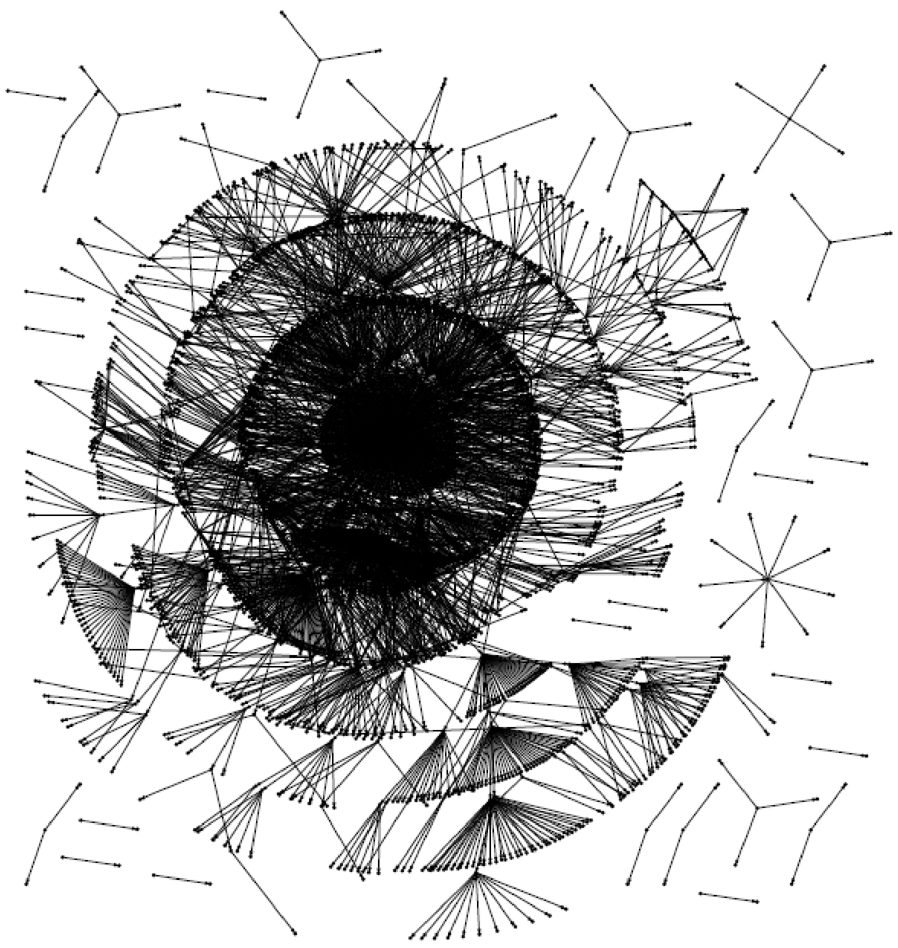

Fig. 10. Graphical representation of Java8 hierarchy

References

[1]R. Agrawal, A. Borgida, and J. V. Jagadish.: Efficient management of transitive relationships in large data and knowledge bases. ACM SIGMOD International Conference on Management of Data. pp. 115-146 (1989).
[2]A. Bouchet.: Etude combinatoire des ordonnes finis, Applications. PhD thesis, Universite scientifique et medicale de Grenoble (1971).
[3]C. Capelle.: Representation of an order as union of interval orders. Proceedings of ORDAL'94, LNCS 831. pp. 143-161 (1994).
[4]Y. Caseau.: Efficient handling of multiple inheritance hierarchies. Proceedings of OOPSLA'93. pp. 271-287 (1993).

[5]Y. Caseau, M. Habib, L. Nourine, and O. Raynaud.: Encoding of multiple inheritance hierarchies and partial orders. Computational Intelligence. pp. 50-62 (1999).
[6]P. Colomb, O. Raynaud, and E. Thierry.: Generalized polychotomic encoding. Proceedings of MCO'08. pp. 77-86 (2008).
[7]A. Fall. The foundations of taxonomic encodings. Computational Intelligence. pp. 598-642 (1998).
[8]R. E. Filman.: Polychotomic encoding: A better quasi-optimal bit-vector encoding of tree hierarchies. Proceedings of ECOOP'2002. pp. 545-561 (2002).
[9]T. Gallai.: Transitiv orientierbare graphen. Acta Mathematica Academiae Scientiarum Hungaricae. pp. 25-66 (1967).
[10]J. Gil and Y. Zibin.: Efficient subtyping tests with pq-encoding. ACM Trans. Program. pp.819-856 (2005).
[11]M. Habib and L. Nourine.: Bit-vector encoding for partially ordered sets. Proceedings of ORDAL'94, LNCS 831. pp.1-12 (1994).
[12]M. Habib, L. Nourine, O. Raynaud, and E. Thierry.: Computational aspects of the 2-dimension of partially ordered sets. Theor. Comput. Sci. pp. 401-431 (2004).
[13]H. V. Jagadish.: A compression technique to materialize transitive closure. ACM Transactions on Database Systems. pp. 558-598 (1990).
[14]A. Krall, J. Vitek, and R.N. Horspool.: Near optimal hierarchical encoding of types. Proceedings of ECOOP'97. pp. 128-145 (1997).
[15]V. Novak.: On the pseudo-dimension of ordered sets. Czechoslovak Math. Journal. pp. 587-598 (1963).
[16]Krzysztof Palacz and Jan Vitek.: Java subtype tests in real-time. ECOOP'03 Conference Proceedings. pp. 378-404 (2003).
[17]O. Raynaud and E. Thierry.: A quasi optimal bit-vector encoding of tree hierarchies. Proceedings of ECOOP'2001, LNCS 2072. pp. 165-180 (2001).
[18]O. Raynaud and E. Thierry.: The complexity of embedding orders into small products of chains. pp. 365-381 (2010).
[19]E. Sperner.: Ein satz • uber untermengen einer endlichen menge. Math. Z. pp. 544-548 (1928).
[20]M. Talamo and P. Vocca.: An efficient data structure for lattice operations. SIAM J. Comput. pp. 1783-1805 (1999).
[21]M. Tedder, D. Corneil, M. Habib, and C. Paul.: Simple, linear-time modular decomposition. ICALP. N 5125 in LNCS. pp. 634-645 (2008).
[22]J. Vitek, R.N. Horspool, and A. Krall.: Efficient type inclusion tests. OOP- SLA'97. pp. 142-157 (1997).
[23]Y. Zibin and Y. Gil.: Efficient subtyping tests with pq-encoding. Proceedings of OOPSLA'2001 (2001).
[24]E. Thierry.: Sur quelques interactions entre structures de donnees et algorithms efficaces pour les ordres et les graphes. PhD thesis, LIRMM, Universite Montpellier II, (2001).