Computing a Hierarchical Static order for Decision Diagram-Based Representation from P/T Nets*

Silien Hong[1], Fabrice Kordon[1], Emmanuel Paviot-Adet[2], and Sami Evangelista[3]

[1] Univ. P. & M. Curie, CNRS UMR 7606 – LIP6/MoVe, 4 place Jussieu, F-75005 Paris
Slien.Hong@lip6.fr, Fabrice.Kordon@lip6.fr
[2] Univ. Paris Descartes and LIP6/MoVe, 143 Avenue de Versailles, F-75016
Emmanuel.Paviot-Adet@lip6.fr
[3] LIPN, CNRS UMR 7030, Univ. Paris XIII, 99 avenue J-B. Clément, F-93430 Villetaneuse
Sami.Evangelista@lipn.univ-paris13.fr

Abstract. State space generation suffers from the typical combinatorial explosion problem when dealing with industrial specifications. In particular, memory consumption while storing the state space must be tackled to verify safety properties. Decision Diagrams are a way to tackle this problem. However, their performance strongly rely on the way variables encode a system. Another way to fight combinatorial explosion is to hierarchically encode the state space of a system. This paper presents how we mix the two techniques via the hierarchization of a precomputed variable order. This way we obtain a *hierarchical static order* for the variables encoding a system. This heuristic was implemented and exhibits good performance.

Keywords: State space generation, Decision Diagrams, Hierarchy.

1 Introduction

Context Model Checking is getting more and more accepted as a verification technique in the design of critical software such as transportation systems. However, the associated state space generation suffers from the typical combinatorial explosion problem when dealing with industrial specifications. In particular, memory consumption when computing the state space must be tackled to verify safety properties.

Decision Diagrams, such as Binary Decision Diagrams [6], are now widely used as an extremely compact representation technique of state spaces [8]: a state is seen as a vector of values and a state space represented by decision diagrams is a set of such vectors where identical extremities are shared. Performances of decision diagram based techniques are thus strongly related to the way values are ordered to encode a system. Bad performance are observed when the encoding does not exhibit a good sharing factor.

To overcome the problem of shared parts limited to extremities, a hierarchical class of decision diagrams has recently been introduced: Set Decision Diagrams (SDD) [16]. Here, values on the arcs may also be sets of vectors of values represented by decision

* This work was supported by FEDER Île-de-France/System@tic—free software thanks to the NEOPPOD project.

diagrams (and recursively we can have vectors of vectors of... of values), defining a *hierarchical structure*. This way, sub-structures are also shared, the same way the extremities are, allowing more compression. An application to the philosopher problem shows an impressive compression ratio [18].

Problem Since finding the variable order with the best sharing factor is an NP-complete problem [3], many heuristics have been proposed (see [27] for a survey).

Variable ordering problem can be *static* (the order is computed before the state space enumeration) or *dynamic* (the order is adapted during the state space enumeration). This work focus on static variable ordering only.

When dealing with hierarchy, the problem is to find a hierarchical structure where sub-structures can be shared. No work has been done, to the best of our knowledge, concerning the definition of heuristics in such a context.

Contribution This paper proposes ways to order variables in a hierarchical way. To do so, we reuse heuristics for non-hierarchical decision diagrams to hierarchical ones. Our heuristic deals with P/T net because they provide easy ways to exploit their structure.

Experiments and measures show that the hierarchical version of the order performs better than the flat one in most cases.

Structure of the paper Section 2 details our objective and refers to some related works. Then, Section 3 introduces the main definitions we use in the paper. Section 4 describes our approach and its performances before some concluding remarks in Section 5.

2 Objectives and Related Work

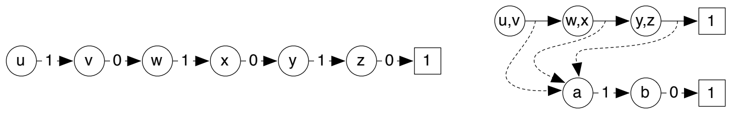
Context Historically, Decision Diagrams were first used to encode models for boolean formulæ via Binary Decision Diagrams [6] (BDDs). Therefore, variables have boolean values in this structure.

In BDDs, each variable in a vector is represented by a node and its value labels an arc connecting the node to its successor in the vector as shown in Figure 1(a). A terminal node (terminal for short) is added to the end of the structure. All root nodes are shared, all identical terminal structures are shared and all nodes have distinct labels on their outgoing arcs. These features ensure a canonical representation of the vectors: i.e. no vector can be represented more than once.

Since BDDs has been successfully used in model checking (SMV [15] and VIS [5] model checkers), numerous kinds of decision diagrams have been developed and employed in this domain. Let us cite Interval Decision Diagram – IDDs [29] – where variables values are intervals (used in DSSZ-MC [20], a model checker for P/T nets[4]), Multi-valued Decision Diagram – MDDs [25] – where variables values are integers (used in SMART [13]), Set Decision Diagrams – SDDs [16] – that use integers as

[4] see definition 3.1.

variables values and introduce hierarchy (used in the ITS [30] model checker) and sigmaDD [7], an instantiation of SDDs to represent terms in term-writing area. Decision diagrams have also been extended to associate values to each vector (MDDs and Algebraic Decision Diagrams [2]). We do not consider such extensions in this work.



(a) BDD encoding (b) SDD encoding

Fig. 1. Encoding the initial state of a system with BDD and SDD

Let us now illustrate why hierarchy is a potentially powerful feature. Figure 1 depicts two Decision Diagrams based representation of the boolean formula $u \wedge \bar{v} \wedge w \wedge \bar{x} \wedge y \wedge \bar{z}$. Figure 1(a) is BDD based, while Figure 1(b) is SDD based. Since values of pairs of variables $(u,v),(w,x),(y,z)$ are the same, they can be shared. This is done in Figure 1(b) by adding the sub-structure with variables a and b. This second structure is more compact that the first one. Let us note that the two terminal nodes are not merged to ease readability of the figure, but should be, as it is in memory.

Contribution To the best of our knowledge, no work has been done to automatically define a hierarchical structure. Such structures are now either defined manually, or directly encoded in higher level formalism such as ITS [30].

Since sub-structures are shared in SDD, we aim at finding parts of the model (in our case P/T nets) that are partially identical thanks to structural analysis. A hierarchical structure is then defined to hierarchically organize these parts.

In this paper, we propose an heuristic reusing an existing variable order to build hierarchical clusters to be encoded using SDD (see Section 4). The existing variable order we use in input of our heuristic can be computed by state of the art algorithms. We observe that, in most cases, hierarchization provides good results.

State of the Art Heuristics to compute variable order to optimize decision diagrams encoding have been studied in several works [27]. Among them, let us report the two following ones, for which an implementation is available.

FORCE [1] computes the forces acting upon each variable and move them in the corresponding direction. In the context of P/T nets, this corresponds to minimizing the distance between places linked to the same transition. The ”average position” of places (e.g. their center of gravity) is recursively computed until stabilization of the algorithm.

The DSSZ-MC model checker for ordinary P/T net, based on Interval Decision Diagrams proposes another heuristic: NAO99 [20]. It exploits the net structure to associate a weight to places and then uses it to compute the variable order.

So far, no heuristic exploiting hierarchical order has been proposed.

3 Preliminary Definitions

This section first recalls the definition of Petri Nets and SDD. Then, it introduces the notions we use to describe our hierarchization algorithm.

3.1 P/T nets

P/T nets stands for Place/Transition nets (also known as Petri nets). They are one of the numerous models used to describe parallelism.

Definition 1 (P/T nets). *A* P/T net *is a 4-tuple $N = \langle P, T, Pre, Post \rangle$ where:*

- *P is a finite set of places,*
- *T is a finite set of transitions (with $P \cap T = \emptyset$),*
- *$Pre : P \times T \to \mathbb{N}$ (resp. $Post : P \times T \to \mathbb{N}$) is the precondition (resp. postcondition) function.*

A marking *M of N is a function associating an integer to each place: $M : P \to \mathbb{N}$. The* initial marking *is denoted M_0.*

The firing *of a transition t changes the marking M_1 into a new marking M_2 (denoted $M_1[t > M_2)$:*

- *t can be fired iff $\forall p \in P, Pre(p,t) \leq M_1(p)$,*
- *$\forall p \in P, M_2(p) = M_1(p) + Post(p,t) - Pre(p,t)$.*

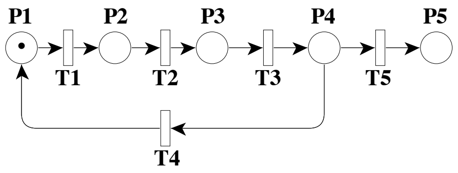


Fig. 2. A simple P/T net

A P/T net is thus a bipartite graph where vertices are places and transitions. Places are depicted by circles (the one of Figure 2 has five places P_1, P_2, P_3, P_4 and P_5), transitions by rectangles (the one of Figure 2 has five transitions T_1, T_2, T_3, T_4 and T_5), and an arc is drawn between a place (resp. transition) and a transition (resp. place) iff the precondition (resp. postcondition) function associates a non null value to the couple. 1 is assumed when no value is associated to arcs in Figure 2). A marking is depicted by black dots in places (in Figure 2 the initial marking of place P_1 is 1).

Note 1 (Successor and predecessor sets). These notations are used later in the paper. The successor set of a place p (resp. transition t) is denoted $p^\bullet = \{t \mid Pre(p,t) \geq 1\}$ (resp. $t^\bullet = \{t \mid Post(p,t) \geq 1\}$). The predecessor function is defined accordingly: ${}^\bullet p = \{t \mid Post(p,t) \geq 1\}$ and ${}^\bullet t = \{t \mid Pre(p,t) \geq 1\}$. Those notations are extended to sets of nodes $S \subseteq P \cup T$: $S^\bullet = \bigcup_{s \in S} s^\bullet$ and ${}^\bullet S = \bigcup_{s \in S} {}^\bullet s$.

3.2 Hierarchical Set Decision Diagram [16]

SDDs are data structures representing sets of assignments sequences of the form $\omega_1 \in s_1; \omega_2 \in s_2; \ldots; \omega_n \in s_n$ where ω_i are variables and s_i are sets of values. In [16] no variable ordering is assumed, and the same variable can occur several times in an assignment sequence. The terminal (labelled by 1) represents the empty assignment sequence, that terminates any valid sequence. Another terminal labelled by 0 is also used to represent the empty set of assignment sequences that terminates invalid sequences[5]. In the following, *Var* denotes a set of variables, and for any $\omega \in Var$, $Dom(\omega)$ represents the domain of ω which may be infinite.

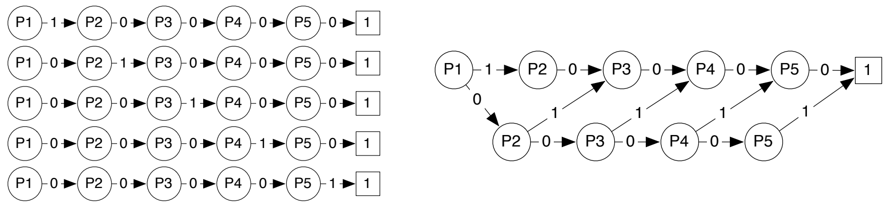
Definition 2 (Set Decision Diagrams). *Let Σ be the set of SDDs. $\delta \in \Sigma$ is inductively defined by:*

- $\delta \in \{0,1\}$ *or*
- $\delta = \langle \omega, \pi, \alpha \rangle$ *with:*
  - $\omega \in Var$.
  - *A partition $\pi = s_0 \cup \cdots \cup s_n$ is a finite partition of $Dom(\omega)$, i.e. $\forall i \neq j, s_i \cap s_j = \emptyset, s_i \neq \emptyset$, with n finite.*
  - $\alpha : \pi \to \Sigma$*, such that* $\forall i \neq j, \alpha(s_i) \neq \alpha(s_j)$.

Elements of $Dom(\omega)$ can be integers, real numbers, vectors, etc. These elements can also be sets represented by SDDs, in which case a hierarchical decision diagram structure is obtained.

SDDs are used here to efficiently store P/T nets reachable markings. We need to define a hierarchical structure (see heuristic described in section 4) and automatically encode the *Pre* and *Post* operations.

SDDs also introduce homomorphisms, a "tool box" to manipulate diagrams. Three kinds of homomorphisms are used here: one to access the desired variable in the hierarchy, one to test and decrement the value (*Pre* operation) and the last one to increment the value (*Post* operation). Further description is out of scope. See [16] for more details on homomorphisms.



(a) All paths for the state space (b) The corresponding SDD without hierarchy

Fig. 3. SDD-based representation of the state space for the P/T net of Figure 2

[5] the 0 terminal and paths leading to the 0 terminal are usually not represented in the structure to save memory.

Let us illustrate the encoding of a P/T net on the example presented in Figure 2. The five states in the reachability graph can be described in the SDD paths shown in Figure 3(a). This leads to the corresponding SDD structure presented in Figure 3(b). This structure does not take advantage of hierarchy.

3.3 Hierarchical Static Order

Definition 3 describes the structure used in this work to efficiently encode a P/T net. We compose two techniques: the computation of a static order and the use of hierarchical decision diagrams.

Definition 3 (Hierarchical Static Order). *A* hierarchical static order *is represented as a list of elements. Each element can be either :*

– *a list of elements $e_1,\ldots,e_n$ denoted $[e_1,\ldots e_n]$,*
– *a place of the encoded P/T net (each place of the net is encoded only once).*

Figure 4 shows the use of a hierarchical static order, $[[P_3,P_4,P_5],[P_1,P_2]]$, to encode the state space for the P/T net of Figure 2. We observe that the sub-structure P_1,P_2 is not shared with the P_4,P_5 one because of the different labelling of the nodes. This problem is tackled in section 3.4.

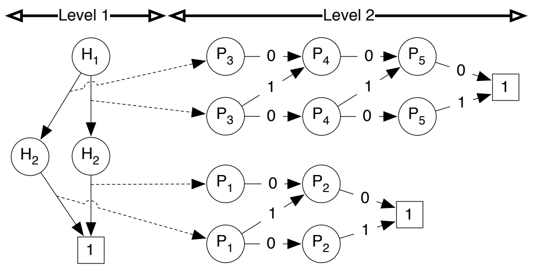


Fig. 4. Final SDD structure with hierarchy

It is also possible to flatten the hierarchy order. This is useful to preserve the precedence relation for each variable and compare the hierarchical static order to its related static order.

Definition 4 (Flattened Hierarchical Static Order). *Let us define a flattened hierarchical static order $f = p_1, p_2, ..., p_n$ with $p_i \in P$. $f = F(h)$ where h is a hierarchical static order. We define F as follows:*

– *$F(h = [h_1, ...h_n]) = [F(h_1), ..., F(h_n)]$, where h_i are the sub-elements of h,*
– *$F(h = [p]) = [p]$ when the element is reduced to a place.*

As an example, the flatten order of $[[P_3,P_4,P_5],[P_1,P_2]]$ is $[P_3,P_4,P_5,P_1,P_2]$.

3.4 Anonymization

Since no hierarchical static order was originally defined for SDDs, variable names were initially associated to SDD nodes. This way, homomorphisms could be defined in such a general structure. In Figure 4, nodes P_4 and P_5 are not shared with P_1 and P_2 because of their different labeling. We thus propose to remove it. Thus the encoding without labelling is said to be *anonymized*.

This labeling information is redundant when a hierarchical static order is defined because the path followed in the structure is sufficient to retrieve the variable name associated to each node.

Definition 5 (Anonymous SDD). *An* anonymous SDD *is defined over a set Var reduced to a unique variable:* $\mid Var \mid = 1$.

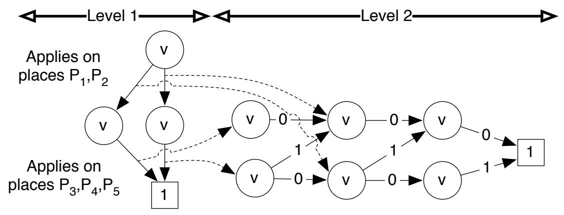


Fig. 5. Final SDD structure with hierarchy and anonymization

Figure 5 illustrates the advantages of anonymization on the SDD structure presented in Figure 4. Since nodes in the second level have similar structures, homomorphisms must be aware of the path followed at the first level: e.g. if the second level is considered as a value of the root node of the first level, then nodes are implicitly labelled $P1$ and $P2$, otherwise, they are labelled $P3$, $P4$ and $P5$. The final hierarchical SDD contains 9 nodes instead of 13 in Figure 4.

Remark in this specific example, let us notice that the flattened SDD encoding in Figure 3(b) holds as many nodes as in the hierarchical one of Figure 5. This is a side effect due to the small size of an understandable example. Benchmarks provided later in this paper show the benefits of hierarchy for large models.

4 Hierarchization of Computed Flat Static Orders

As presented in Section 3.3, we must increase the number of identical modules to increase the sharing via hierarchy and anonymization. A first approach is to define the hierarchy when modeling the system. In [30], a new type of formalism (ITS for Instantiable Transition Systems) allows such a hierarchical modeling with an appropriate relation to SDD. This relation allows an efficient encoding when the model has symmetries. As an example, the state space for the philosopher model with $2^{20,000}$ philosophers could be encoded. However, this technique assumes that the system designer is able to build this ”well formed” hierarchy.

To automate the building of a hierarchical structure, we could have used symmetries as defined in [9] for colored net and [28] for P/T net. But this would lead to a restrictive application of the technique. For instance the model in Figure 2 does not exhibit any symmetry for set of places $[P_4, P_5]$ and $[P_1, P_2]$ but Figure 5 shows that the projection of the state spaces of these sets are the same.

So, we propose to split an already computed “flat” variable static order into modules. Then, a hierarchical structure is defined on top of these modules. This approach allows to adapt all the previously proposed heuristics to compute static variable orders. It also allows us to validate the gain we can obtain from the hierarchy. It is, however, a bit rough since no clue is given on modules similarity. We thus call this heuristic: *N-Cut-Naive Hierarchy*.

4.1 Algorithm

The main idea of this heuristic is to create a hierarchy of decision diagrams representing portions of markings that are similar ones to the others, thus increasing the sharing of elements. We exploit the hierarchical static order together with the anonymization mechanism (introduced in section 3.4).

```
N-Cut-Naive($X, N, Order$)
begin
       Input: $X$ an integer: Hierarchy depth
       Input: $N \geq 1$ : Number of elements in a level of hierarchy
       Input: $Order \neq \emptyset$ : The order to split
1      if $X > 0$ then
2          $splitOrder = \emptyset$
3          $counter = 0$
4          $subStructure = \emptyset$
5          for $element \in Order$ do
6              if $counter <= N$ then
7                  $subStructure.addQueue(element)$
8              end
9              else
10                 $splitOrder.addQueue(subStructure)$
11                 $subStructure = \emptyset$
12                 $counter = 0$
13             end
14             $counter = counter + 1$
15         end
16         $Order =$ N-Cut-Naive($X - 1, N, splitOrder$)
17     end
18     return $Order$
end
```

Fig. 6. Algorithm of the N-Cut-Naive Hierarchy heuristic

Our heuristic is presented in Figure 6. It requires in input: *i)* a precomputed flat order ($Order$), *ii)* the maximum number of elements in a module (N) and *iii)* the height of the hierarchical structure (X).

To give an intuition of the way our algorithm behaves, let us assume the following input order:

$$[P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12}]$$

where P_i are variables encoding places of a P/T Net. We split the initial flat order into modules of size n specified by the user. When the number of modules is large enough, the list of modules is recursilvely split to add an extra-level to the hierarchical structure. Figure 7 depicts the obtained hierarchical structure when $N = 2$. Here, three levels are needed $H_{1,x}$, $H_{2,x}$ and $H_{3,x}$.

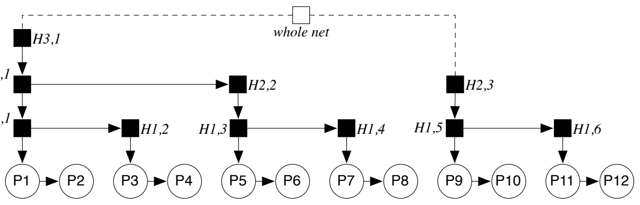


Fig. 7. Computed Static Hierarchical Order with $X = 3$ and $N = 2$

The size of the module is a critical parameter: it is obvious that if we construct two modules by randomly choosing an identical number of places, the sharing factor between the two modules usually decreases when the parameter N grows. We have the same kind of effect with parameter X.

In the representation of Figure 7, sub-structures $H_{1,1}$ and $H_{1,2}$ can be represented by a single SDD if P_1, P_2 valuations are *identical* to P_3, P_4 ones. At the next level, the condition is more difficult to satisfy since a single representation for $H_{2,1}$ and $H_{2,2}$ requires that $H_{1,1}, H_{1,2}$ are identical to $H_{1,3}, H_{1,4}$ (*e.g.* the valuations of P_1, P_2, P_3, P_4 is identical to P_5, P_6, P_7, P_8).

Let us note that this computed static order hierarchy represents a sharing potential that is only possible when sub-structures are identical. This is where the structure of the P/T net introduces constraints. All hierarchical structures do not systematically exhibit a good sharing factor but we statistically observe that, for small values of X, the algorithm provides good results.

We have empirically computed (see experiment 1 in section 4.2) that, for $N = 2$ and $X = \lceil log_N(\mid P \mid) \rceil$, the sharing factor is close to the optimal one.

4.2 Performance Analysis

This section evaluates the performance of the N-Cut-Naive Hierarchy heuristic compared to given flat orders. Evaluation is done for state space generation. Experiments are performed using a tool developed within the NEOPPOD project to verify safety properties: PNXDD [21]. Performances are computed on an 2.80GHz Intel Hyperthreaded Xeon computer with 14Gbyte of memory.

Models We have selected three types of models to assess our contribution:

- *"Colored" model* that are P/T nets derived from colored models by unfolding[6]. In this category, we can use the size of color domains as a parameter to increase the size of the obtained models.
- *K-Bounded models*[7] for which the number of initial tokens is the scaling parameter.
- *Cases studies models* that are extracted from projects involving industrial problems and are thus larger. Neoppod and PolyORB are not fully colored since they contain some uncolored places that have an effect on the unfolding into P/T. MAPK is purely K-bounded. These models have a more complex structure from which extraction of patterns is more difficult.

[6] Here, unfolding means the expansion of a colored net into its corresponding P/T places as presented in [23]. It suppresses all 0-bounded places and the transitions pre and post-conditions of such places.

[7] Let us note K-Bounded models where $K > 1$ and K is the maximal number of tokens for each place in any reachable marking

Model	scaling parameter	Places	Transitions	Description
Colored Models				
Philosopher [17]	300 500 800 2,000 4,000	1,200 2,000 3,200 8,000 16,000	1,500 2,500 4,000 10,000 20,000	Classical multi-process synchronization problem. The scaling parameter corresponds to the number of philosophers around the table.
Peterson [26]	2 3 4 5	30 102 244 480	30 126 332 690	Concurrent programming algorithm for mutual exclusion. The scaling parameter corresponds to the number of processes to be synchronized.
Token Ring [11]	5 10 100 200 500	40 80 800 1,600 4,000	40 80 800 1,600 4,000	Classical local network protocol communication. The scaling parameter corresponds to the number of processes involved in the ring.
K-bounded Models				
FMS [14]	10 to 30	22	20	Model of a Flexible Manufactoring System. The scaling parameter changes the initial number of tokens in several places of the model.
Kanban [13]	50 to 3,000	16	16	Kanban system modeling. The scaling parameter changes the number of tokens in several places of the model.
Cases Studies Models				
Neoppod [10]	2 3 4 5 6	49 120 229 382 585	30 118 330 753 1,498	Broadcast Consensus Protocol in a distributed database system when the set of masters nodes in the system are electing the primary master. The scaling parameter corresponds to the number of master nodes.
PolyOrb [22]	2 3 4 5 6	222 297 372 447 522	962 1,444 1,925 2,414 2,902	Models the core of a middleware that manages parallelisms between a set of thread to be operated. We use a configuration with 4 event sources, and an query FIFO with 4 slots. The scaling parameter corresponds to the number of threads that are really activated in the middleware.
MAPK [19]	8 to 80	22	30	Bio-chimical modeling. The scaling parameter changes the number of tokens in several places of the model.

Table 1. Presentation of the models selected for our benchmark

The two first types of benchmarks correspond to typical combinatorial explosion situations to be handled in state space generation. The third one shows more "realistic" situations from an industrial point of view.

Table 1 summarizes for each model: its origin, what it is modeling, the meaning of its scaling parameter, and its size (number of places and transitions) according to the scaling parameters (the number of instance corresponds either to the cardinality of a color domain or to the number of tokens in the place modeling the initial state of an actor).

Evaluation Procedure Evaluation is performed on each model and for various scaling parameters using the following procedure:

1. produce a flat order using external ways (e.g. from the state of the art),
2. apply our heuristic to generate a hierarchical static order,
3. generate the state space using the flat order as well as the hierarchical static one (PNXDD is able to import a given precomputed order, flat or hierarchical) and compare them,

To compare the generated state spaces, we consider the following parameters:

– the number of nodes in the final SDD diagram and the total number of nodes used during the computation of the state space (peak[8]), for the studied hierarchical order and its associated flattened version,
– CPU time in seconds used for computation,
– memory used for computation in Mbytes. This involves the nodes used while computing the state space, as well as the operation caches used to speed up SDD operations [4].

Experiments Three experiments have been elaborated.

Experiment 1 aims at studying the impact of the hierarchical structure height (parameter X in function `N-Cut-Naive`). To do so, we use an existing flat static order *Order* computed using NOA99. We apply function `N-Cut-Naive`$(X, N, Order)$ to build a hierarchical order before evaluating the memory required to store the state space. This experience shows that if we use $N = 2$ and $X = \lceil log_N(\mid P \mid) \rceil$ we are usually close to the optimal values for X and N by studying the obtained performance with different values of N (in $\{2,5\}$) and X (in $\{0,5\}$ [9] except for Philosopher (300) where a wider range is used because of the large number of places). Performances strongly depend on the initial flat order.

Experiment 2 uses the values of N and X validated in experiment 1. Here the objective is to show that hierarchy allows us to study more complex models than the flat version does. We compare here the performance obtained using NOA99 and FORCE

[8] Once a node is created, it is never destroyed in this implementation. So the term *peak* refers here to the total number of nodes used for the computation.

[9] When $X = 0$, the hierarchical order becomes a flat one.

heuristics (in a flat and hierarchical version) for models with larger instances than in experiment 1.

In experiment 3, we check the general behavior of hierarchization to be sure that there is no side-effect due to the selected heuristics (FORCE and NOA99). To do so, we select two models from the “case study” group: Neoppod (mostly colored) and MAPK (strictly K-bounded) to check the behavior of our heuristic on 500 randomly generated flat static orders.

4.3 Experiment 1: Effects of Parameters X and N

This experiment aims at studying the impact of the hierarchical structure height in order to find an appropriate value to be used in our heuristic.

Results They are displayed in Figure 8 that shows the evolution of memory consumption according to values of X and N (each value of N is a curve, and X values are shown in abscissa). The table in Figure 8(i) summarizes memory consumption in the best case, the worst case, and for the pair $\langle X,N\rangle$ when $X = \lceil log_N(\mid P\mid)\rceil$ and $N = 2$ (later referred to as the ”computed solution”).

Conclusions Almost all curves for a fixed N (variation of X) have the same shape: for a given value of N, performance start to be better while X increases. However, after a while, performance decreases when X continues to increase (performance loss is important in most examples). An explanation could be that adding a hierarchical level is only of interest when many modules are shared in its immediate sub-level. This is unlikely the case when the number of modules in this sub-level is not great enough (when splitting the flat order, we are not sure to obtain identical modules). This is why we decided to use the following formula $X = \lceil log_N(\mid P\mid)\rceil$.

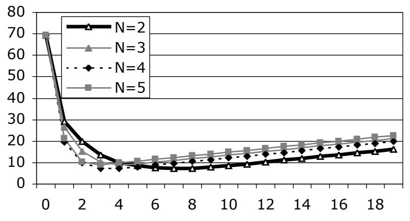
Table in Figure 8(i) shows that in most cases, our automatic computation of X for $N = 2$ (value selected in our heuristic) provides performances that are reasonably close to the best solution.

4.4 Experiment 2: Gain with Respect to Existing Flat Orders Heuristics

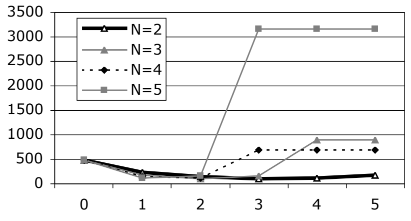
The aim is to compare our N-Cut-Naive Hierarchy heuristic performance to the ones of the two flat static orders we introduced in section 2: FORCE [1] and NOA99 [20] (respectively noted F and N in Tables 2 to 4). We compare the state space generated for the benchmark models. For each scaling parameter, we measure:

- The total SDD nodes once the state space is produced,
- The peak SDD nodes when producing the state space,
- Computation time in seconds,
- The peak of memory consumption (in Mbytes).

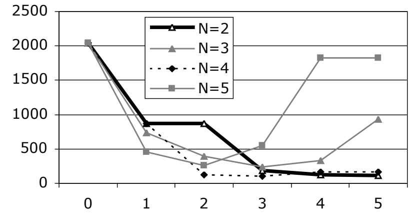
In the tables, MOF means that the experiment could not be computed due to the lack of Memory ($>$ 10 Go). SOF means that execution reached a Stack OverFlow (we used



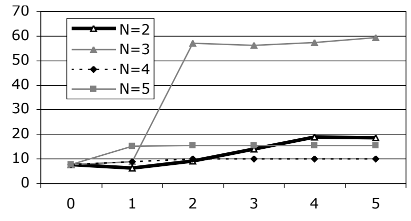
(a) Philosopher (300)



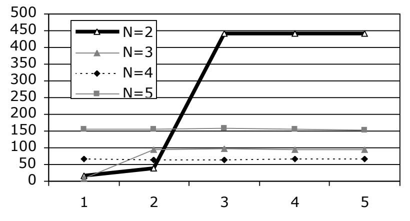
(b) Peterson (4)



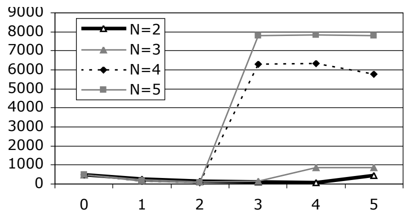
(c) Token ring (100)



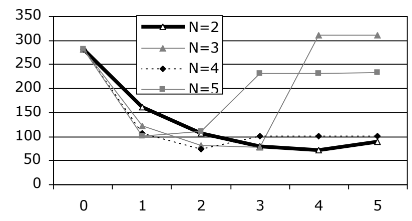
(d) FMS (10)



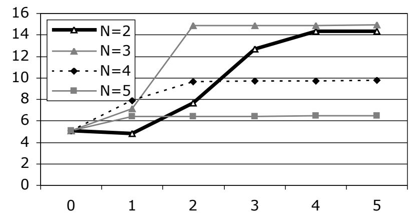
(e) Kanban (20)



(f) Neoppod (4)



(g) PolyORB (2)



(h) MAPK (8)

Model	**Best Solution**			**Computed Solution**			**Worst Solution**			**Loss with respect to Best Solution**	**Gain with respect to Worst Solution**
	N	X	memory (MB)	N	X	memory (MB)	N	X	memory (MB)		
Philosopher 300	4	4	7.17	2	4	10.04	–	0	69.3	-40%	85%
Peterson 4	2	3	98.31	2	3	98.10	4	5	3167	0%	97%
Token Ring 100	4	3	105.88	2	4	121.30	–	0	2048.5	-14.5%	94%
FMS 10	2	1	6.43	2	1	6.43	3	3	56.3	0%	67.8%
Kanban 20	3	1	7.90	2	1	18.00	2	5	443.00	-56%	98%
Neoppod 4	2	4	77.03	2	3	114.50	4	5	7 821	-48.5%	99%
PolyORB 2	2	4	72.00	2	5	89.00	5	3	311.00	-19%	71%
MAPK 8	2	1	4.81	2	1	4.81	3	5	15	0%	89.1%

(i) Comparison of best and worst solutions to one computed by our heuristic

Fig. 8. Analysis of the impact of the X and N parameters on the N-Cut-Naive Hierarchy heuristic (vertical scale shows memory usage in Mbyte). Abscissa of figure 8(e) starts with $X = 1$ because execution was too fast to capture memory for $X = 0$.

the default Unix stack size on our machine) and TOF that we reached a Time OverFlow (3 days).

There are three sets of data: measures on flat orders, measures on the hierarchization of the associated flat order and finally, the gain we measure from the flat order to the hierarchy one.

Results Measures are shown in Table 2 for colored models, in Table 3 for K-bounded ones, and in Table 4 for case studies models. It is of interest to see that, over the types of model, our heuristic behaves differently.

We can observe on Table 2 that for colored models, we get good results. Our heuristic save memory up to 80% for the Neoppod and Peterson models. It even allows the computation of the state space when the flat orders fail for the philosopher and token ring models. We also observe a good scalability parameter: in all cases, increasing of the color cardinalities means an increase of the gain we observe.

Number of Instances	State Space Size	Flat	Flat order performance				Hierarchical order performance				Gain (in %)			
			Final	Peak	Time	MB	Final	Peak	Time	MB	Final	Peak	Time	Mem
Philosopher's diner														
300	1.4×10^{143}	F	7,216	33,449	7.9	39.22	813	9,787	0,5	13,1	89	71	94	67
		N	7,769	42,106	8.9	69.30	430	4,350	0.5	13.11	94	90	93	81
500	3.6×10^{238}	F	12,061	55,992	20.5	80.12	1,219	14,802	0,9	18,2	90	74	96	77
		N	12,969	70,306	22.2	151.61	390	3,810	1.0	12.29	97	95	95	92
800	4.0×10^{381}	F	19,275	89,722	77.3	260.39	1,492	18,432	1,9	27,6	92	79	98	89
		N	20,769	112,606	79.4	260.52	577	5,883	2.6	20.03	97	95	97	92
2000	1.7×10^{954}	F	SOF	SOF	SOF	SOF	2,459	30,786	8,9	63,6	∞	∞	∞	∞
		N	SOF	SOF	SOF	SOF	724	7,415	25.6	45.42	∞	∞	∞	∞
4000	3.0×10^{1908}	F	SOF	SOF	SOF	SOF	SOF	SOF	SOF	SOF	–	–	–	–
		N	SOF	SOF	SOF	SOF	1,349	14,287	96,5	125,3	∞	∞	∞	∞
Peterson algorithm														
2	158	F	188	1,085	0.06	4.25	78	415	0.01	3.98	59	62	72	6
		N	135	777	0.04	4.10	66	357	0.01	3.93	51	54	62	4
3	20,754	F	3,767	41,395	2.8	24.54	769	10,857	0.4	11.26	80	74	84	54
		N	4,441	62,317	3.9	28.19	756	10,837	0.2	9.54	83	83	93	66
4	3.4×10^{6}	F	81,095	1.3×10^{6}	482.1	897.81	10,719	344,170	37.7	241.5	87	74	92	73
		N	72,578	996,414	265.8	483.53	9,291	262,144	13.8	98.31	87	74	95	80
Token Ring protocol														
5	53,856	F	246	3,424	0.1	5.08	96	1,889	0.03	3.95	61	45	83	22
		N	230	2,882	0.1	4.86	75	1,549	0.02	3.86	67	46	82	21
10	8.3×10^{9}	F	997	17,263	0.9	10.19	281	10,198	0.1	6.67	72	41	82	35
		N	809	16,708	0.9	9.62	200	9,523	0.1	6.45	75	43	84	33
100	2.6×10^{105}	F	93,534	8.4×10^{6}	2,712	2,367	5,954	3.5×10^{6}	178.1	636.8	94	59	93	73
		N	71099	5.9×10^{6}	1671.62	2048	2,613	538,470	23.7	123.4	96	91	99	94
200	8.3×10^{211}	F	MOF	MOF	MOF	MOF	14,623	2.4×10^{7}	2,114	4,961	∞	∞	∞	∞
		N	MOF	MOF	MOF	MOF	5,150	4.1×10^{6}	206.2	657	∞	∞	∞	∞
500	5.0×10^{531}	F	MOF	MOF	MOF	MOF	MOF	MOF	MOF	MOF	-	-	-	-
		N	MOF	MOF	MOF	MOF	31,473	5.7×10^{7}	3,140	8,860	∞	∞	∞	∞

Table 2. Results of experiment 2 on colored models. Time is expressed in seconds.

As shown in Table 3, results are not that good for K-bounded models: the peak size remains reasonable, but the memory consumption increases faster with the hierarchical version than with the flat one. This is due to the cache size that, in our experiments is

Number of Ins-tances	State Space Size	Flat	Flat order performance				Hierarchical order performance				Gain (in %)			
			Final	Peak	Time	MB	Final	Peak	Time	MB	Final	Peak	Time	Mem
Flexible Modeling System (FMS)														
10	2.5×10^{9}	F	338	4,325	0.21	5.14	257	5,075	0.14	5.42	24	-17	33	-6
		N	1,256	13,578	0.61	7.85	684	9,144	0.17	6.45	46	33	72	18
20	6.0×10^{12}	F	848	16,480	0.7	8.88	697	23,455	1.1	11.98	18	-42	-41	-35
		N	4,391	64,728	2.7	22.43	2,354	47,784	1.4	19.13	46	26	47	15
50	4.2×10^{17}	F	3,578	129,745	5.7	45.75	3,217	238,347	39.1	103.47	10	-84	-575	-126
		N	25,196	488,059	27	134.93	13,364	432,297	105	266.55	47	11	-292	-98
Kanban														
50	1.0×10^{16}	F	3,216	30,770	0.86	0.0	4,235	144,373	148	423	-31	-369	-17109	$-\infty$
		N	53,791	1.0×10^{7}	1,170	3,009	8,060	4.3×10^{6}	397	955	85	57	34	68
100	1.7×10^{19}	F	11,416	111,570	2.84	30.91	15,960	873,517	26,484	2,246	-39	-682	-9×10^{5}	-7166
		N	MOF	MOF	MOF	MOF	MOF	MOF	MOF	MOF	-	-	-	-
2,000	2.9×10^{33}	F	4.0×10^{6}	3.3×10^{7}	48,243	6,971	MOF	MOF	MOF	MOF	-	-	-	-
		N	MOF	MOF	MOF	MOF	MOF	MOF	MOF	MOF	-	-	-	-
3,000	-	F	MOF	MOF	MOF	MOF	MOF	MOF	MOF	MOF	-	-	-	-
		N	MOF	MOF	MOF	MOF	MOF	MOF	MOF	MOF	-	-	-	-

Table 3. Results of experiment 2 on K-bounded models. Time is expressed in seconds.

number of Ins-tances	State Space Size	Flat	Flat order performance				Hierarchical order performance				Gain (in %)			
			Final	Peak	Time	MB	Final	Peak	Time	MB	Final	Peak	Time	Mem
Mitogen Activated Protein Kinase (MAPK)														
8	6.1×10^{6}	F	452	4,018	0.28	5.1	256	4,819	0.14	5.2	43	-20	51	-3
		N	498	5,705	0.28	5.1	336	5,628	0.35	5.4	33	1	-23	-7
20	8.8×10^{10}	F	2,812	29,332	1.9	13.1	1,177	43,856	0.4	20.6	58	-50	78	-57
		N	3,082	52,264	2.8	18.5	1,992	69,661	3.6	23.9	35	-33	-27	-30
80	5.6×10^{18}	F	94,582	962,902	112.5	310.0	10,719	344,170	37.7	241.5	89	64	67	22
		N	96,462	2.4×10^{6}	290	651.4	-	-	-	-	-	-	-	-
NEOPPOD Consensus protocol														
2	194	F	202	679	0.024	4.1	80	217	0.007	3.42	60	68	71	17
		N	463	1,688	0.024	4.5	154	558	0.011	3.57	67	67	55	21
3	90,861	F	5,956	48,269	0.86	19.0	1,126	12,491	0.15	8.2	81	74	82	57
		N	3,820	23,974	0.76	12.2	709	4,838	0.10	6.7	81	80	87	45
4	9.7×10^{8}	F	84,398	1.0×10^{6}	62.6	338.8	11,728	178,921	5.0	70.83	86	82	92	79
		N	155,759	1.3×10^{6}	186.1	490.3	19,875	217,536	0.007	114.2	87	83	99	77
PolyOrb														
2	1.6×10^{6}	F	223,243	3.1×10^{6}	580.8	1,316	27,548	491,803	29.3	352	88	84	95	73
		N	78,785	451,494	98.7	273	10,050	127,791	6.2	81	87	72	94	70
3	2.8×10^{7}	F	593,363	1.2×10^{7}	4,708	2,851	78,067	2.1×10^{6}	188.7	692	87	83	96	76
		N	280,068	2.5×10^{6}	948.8	1,513	33,526	524,288	64.3	358	88	79	93	76
4	2.1×10^{8}	F	1.2×10^{6}	3.1×10^{7}	8,310	5,765	146,589	4.2×10^{6}	528.7	1,780	87	86	94	69
		N	666,886	8.4×10^{6}	217,173	2,457	84,126	2.1×10^{6}	326.0	1,451	87	75	99	41
5	1.4×10^{9}	F	TOF	TOF	TOF	TOF	143,903	1.1×10^{7}	2,361.6	3,045	∞	∞	∞	∞
		N	TOF	TOF	TOF	TOF	87,875	4.2×10^{6}	1,045.1	2,216	∞	∞	∞	∞
6	9.2×10^{9}	F	TOF	TOF	TOF	TOF	288,649	2.2×10^{7}	15,757	6,144	∞	∞	∞	∞
		N	TOF	TOF	TOF	TOF	140,565	1.5×10^{7}	19,474	4,865	∞	∞	∞	∞
7	-	F	TOF	TOF	TOF	TOF	MOF	MOF	MOF	MOF	-	-	-	-
		N	TOF	TOF	TOF	TOF	TOF	TOF	TOF	TOF	-	-	-	-

Table 4. Results of experiment 2 on case studies models. Time is expressed in seconds.

never cleaned. Other experimentations with different cleaning policies did not lead to better results.

In the FMS model, marking complexity increases slower than in MAPK. However, we can observe that increasing the initial marking decreases performance.

Table 4 shows that our heuristic scales up quite well for large models but still remains less efficient for purely K-bounded models (MAPK). It is of interest to observe that, for PolyORB that contains some K-bounded places, performances of our algorithms are still good: the flat orders we used fail for PolyORB 5 while the hierarchical one fails for 7

The MAPK model also reveals very different behavior of our algorithm when we elaborate the hierarchy from NOA99 or FORCE flat orders.

Conclusions It is obvious from this experiment that the N-Cut-Naive Hierarchy heuristic is of interest for large P/T net models, like the ones obtained by unfolding of colored nets (or mostly colored nets). The increase of color classes cardinality generates numerous P/T places, thus increasing the probability to find identical parts that can be shared. This is why this approach scales up well.

However, our heuristic is less adapted for K-bounded models such as FMS, Kanban and MAPK. Our diagnostic is threefold:

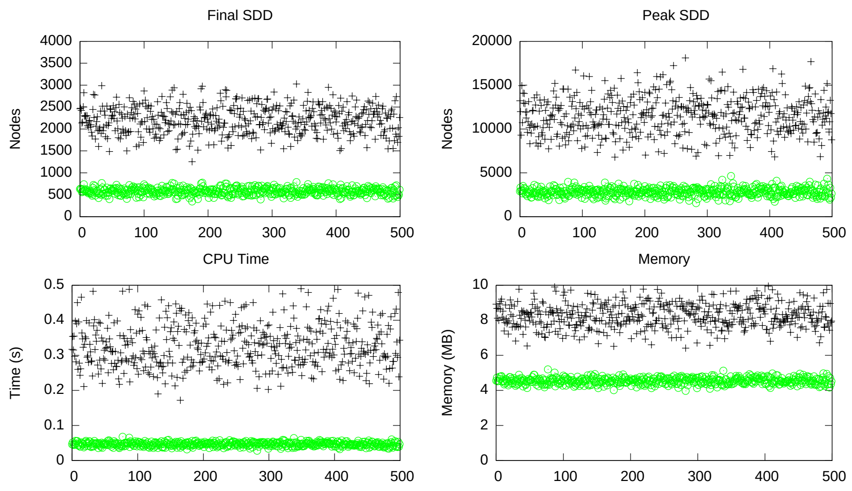
– First, they are small (22 places or less) and thus, the probability to have identical part of SDD is practically null.
– Second, since the models do not grow in number of places but only involve more tokens, the SDD structure remains the same (with no more sharing probability). The gain from the structure remains fixed while, when initial marking increases, the complexity due to the generated markings still increases (this effect is contradictory to the previous one).
– Third, a deep analysis of the SDD structure shows that, if the SDD size grows in depth (longer sequences and hierarchies) for colored models, complexity increases in width for K-bounded models. In this configuration, exploration of values during the union and intersection operations grows quadratically with the width of the SDD[10].

4.5 Experiment 3: Interest for Hierarchical Encoding

The objective is to check if no side-effect comes from the selected flat order heuristics. To do so, we proceed to an analysis of performances against random flat orders. We apply these orders on two cases studies models: Neoppod 2 (mostly colored) and MAPK 8 (K-bounded).

Results they are displayed on Figures 9 (for Neoppod 2) and 10 (for MAPK 8). Each figure contains four charts showing, for each of the 500 executions: *i)* as "+" dots, the value for the selected random flat static order and *ii)* as "◦" dots, the value we get when

[10] This ruins the saturation [12] mechanism we aim to activate with our encoding.



(a) Measures for the 500 experiments

Measure	Flat			Hierarchicalized			Success	Lowest	Highest
	Min	Avg	Max	Min	Avg	Max	Rate	Gain	Gain
Final SDD	1 254	2 211	3 027	348	581	790	100%	72%	75%
Peak SDD	6 790	11 556	18 113	1 552	2 830	4 640	100%	64%	85%
Time (s)	0.17	0.32	0.56	0,03	0.05	0.07	100%	78%	90%
Memory (Mbytes)	6.4	8.28	10.33	3.97	4.54	5.19	100%	36%	54%

(b) Summary of data

Fig. 9. Experiment on random orders and associated hierarchization for Neoppod (2)

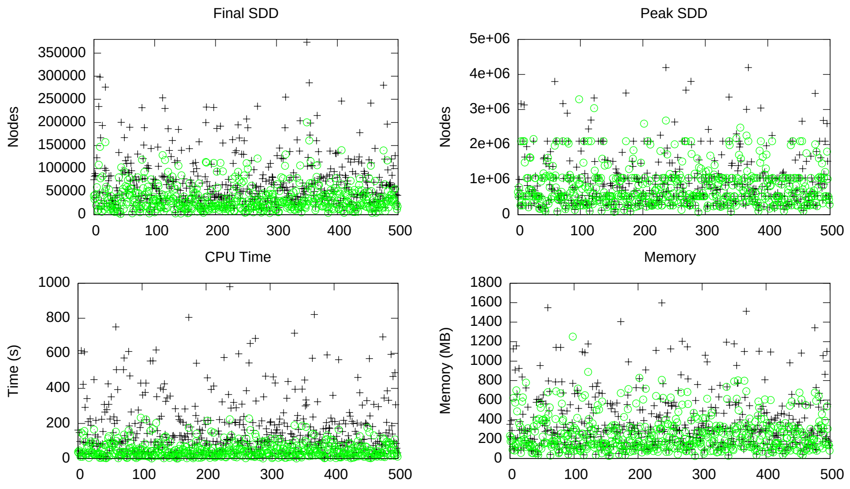
the N-Cut-Naive Hierarchy heuristics is processed on this order for $N = 2$. Four values are measured:

- The number of SDD nodes,
- The peak of SDD nodes,
- Time computation in seconds,
- The peak of memory consumption in Mbytes.

For each figure, a table summarizes results and shows the success rate of the N-Cut-Naive Hierarchy heuristic (*i.e.* the hierarchical order is better than the flat one). The lowest and highest gains between the flat static order and the hierarchical one are also shown on this table.

Conclusions Once again, we observe different results for the Colored and K-bounded models. For Neoppod, there is always a clear separation between the two measures, showing that, in all cases, our heuristic provides better results (see summary in Table 9(b)):

- more than 72% gain for the final number of SDD,



(a) Measures for the 500 experiments

Measure	Flat			Hierarchicalized			Success	Lowest	Highest
	Min	Avg	Max	Min	Avg	Max	Rate	Result	result
Final SDD	2 845	71 255	373 639	1 513	36 182	200 176	100%	29%	71%
Peak SDD	84 240	932 657	4 194 300	72 305	795 300	3 292 390	65%	-173%	71%
Time (s)	4.40	164.63	980.54	1.42	49.73	233.83	99%	-38%	93%
Memory (Mbytes)	34.89	362.69	1598.36	26.12	250.31	1251.59	91%	-122%	79%

(b) Summary of data

Fig. 10. Experiment on random orders and associated hierarchization for MAPK (8)

– more than 64% gain for the SDD peak,
– more than 78% gain for CPU,
– more than 36% gain for memory.

These are good results since the proposed flat orders are usually not efficient.

For MAPK, results are not as good as for the other model. The hierarchical static order is not always better than the flat ones. However, except for the peak, hierarchization often generates a better results (91% hits for memory consumption). This is consistent with the second experience. However, we note in Table 10(b) that we sometimes get good results (for example, up to 79% memory gain).

4.6 Discussion

From these experiments we can conclude that our heuristic is good for large P/T nets and of less interest when the size of the state space depends on the number of tokens only (*e.g.* K-bounded models).

The last experiment also showed that our heuristics does not rely on any flat static order. It can thus be plugged to any existing heuristics providing a flat order.

However, this heuristics suffers from its blindness: it does not consider the structure of the P/T net and thus may discard parts of the structure that could have been shared efficiently.

5 Conclusion

In this paper, we propose a hierarchical way to encode a state space using decision diagrams (Hierarchical Set Decision Diagrams – SDD [16]) to cope with combinatorial explosion of state space generation. To the best of our knowledge no work has been done to automatically define a hierarchical static order with decision diagrams.

We present the N-Cut-Naive heuristics. It starts from an existing flat order and builds a hierarchical order suitable for large P/T net models.

Benefits of the proposed algorithm are:

- State space generation consumes less memory than the one needed when using flat orders, thus allowing to process larger specifications.
- the CPU time required to hierarchize a flat order is negligible compared to the state space generation.

Experiments show excellent gains when the complexity comes from the structure of the specification (*e.g.* large models). However, this is not the case for smaller specifications with a large state space due to numerous tokens hold in places. This is illustrated with two characteristics of our benchmark models: P/T nets unfolded from Colored nets (large models) and K-bounded P/T nets (large markings).

Nevertheless, one can think that when modeling complex systems, both types of complexity are present in the specification: thus, our approach should provide benefits for model checking of most P/T nets. This is confirmed by our results on some "case studies" models that do not exclusively belong to one class (*i.e.* Neoppod and PolyORB models).

Future Work The problems detected for K-bounded models comes from the presence of strongly communicating places (this is very true in the MAKP specification). A future study would be to mix IDD and SDD (allowing intervals to label some arcs of hierarchical diagram). However, such a study needs a more detailed static analysis of the structure of the model. In that domain, more flexible classes of decision diagrams such as the poly-DD [24] might offer better solutions.

Another extension of this work concerns the elaboration of an algorithm that directly computes a hierarchical order. Once again, the use of structural analysis such as P-invariants, deadlocks, or traps in the model is needed to provide the computation modules leading to a better sharing in the state space and thus, to a better compression ratio.

Thanks The authors thank the anonymous reviewers for their useful comments on the paper.

References

1. F. A. Aloul, I. L. Markov, and K. A. Sakallah. FORCE: a fast and easy-to-implement variable-ordering heuristic. In *ACM Great Lakes Symposium on VLSI*, pages 116–119. ACM, 2003.
2. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal Methods in System Design*, 10:171–206, 1997.
3. B. Bollig and I. Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Transactions on Computers*, 45(9):993–1002, September 1996.
4. K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *27th Annual ACM IEEE Design Automation Conference*, pages 40–45. ACM, 1991.
5. R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Staple, G. Swamy, and T. Villa. Vis: A system for verification and synthesis. In *Computer Aided Verification*, volume 1102 of *LNCS*, pages 428–432. Springer Verlag, 1996.
6. R. Bryant. Graph-based algorithms for boolean functionmanipulation. C-35(8):677–691, August 1986.
7. D. Buchs and S. Hostettler. Sigma decision diagrams. In A. Corradini, editor, *TERMGRAPH 2009 : Premiliminary proceedings of the 5th International Workshop on Computing with Terms and Graphs*, number TR-09-05, pages 18–32. Università di Pisa, 2009.
8. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *5th Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, 1990.
9. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, 1993.
10. C. Choppy, A. Dedova, S. Evangelista, S. Hong, K. Klai, and L. Petrucci. The neo protocol for large-scale distributed database systems: Modelling and initial verification. In *31st International Conference on Applications and Theory of Petri Nets (PETRI NETS 2010)*, volume 6128 of *LNCS*, pages 145–164. Springer, 2010.
11. G. Ciardo, G. Lüttgen, and A. S. Miner. Exploiting interleaving semantics in symbolic state-space generation. *Formal Methods in System Design*, 31(1):63–100, 2007.
12. G. Ciardo, G. Lüttgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In *TACAS, LNCS 2031*, April 2001.
13. G. Ciardo and A. S. Miner. Smart: Simulation and markovian analyzer for reliability and timing. volume 0, page 60, Los Alamitos, CA, USA, 1996. IEEE Computer Society.
14. G. Ciardo and K. Trivedi. A decomposition approach for stochastic reward net models. *Perf. Eval*, 18:37–59, 1993.
15. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model verifier. In *Computer Aided Verification*, volume 1633 of *LNCS*, pages 682–682. Springer Verlag, 1999.
16. J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical Decision Diagrams to Exploit Model Structure. In *25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems - FORTE 2005*, volume 3731 of *LNCS*, pages 443–457. Springer, 2005.
17. E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informaticae*, 1:115–138, 1971.
18. A. Hamez, Y. Thierry-Mieg, and F. Kordon. Building efficient model checkers using hierarchical set decision diagrams and automatic saturation. *Fundamenta Informaticae*, 94(3-4):413–437, September 2009.

19. M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In *8th International School on Formal Methods for the Design of Computer - SFM 2008*, volume 5016 of *LNCS*, pages 215–264. Springer, 2008.
20. M. Heiner, M. Schwarick, and A. Tovchigrechko. DSSZ-MC - A Tool for Symbolic Analysis of Extended Petri Nets. In *Applications and Theory of Petri Nets*, volume 5606 of *LNCS*, pages 323–332. Springer Verlag, 2009.
21. S. Hong, E. Paviot-Adet, and F. Kordon. PNXDD model checkers, `https://srcdev.lip6.fr/trac/research/NEOPPOD/wiki`, 2010.
22. J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, pages 139–157. Elsevier, September 2004.
23. F. Kordon, A. Linard, and E. Paviot-Adet. Optimized Colored Nets Unfolding. In *26th International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, volume 4229 of *LNCS*, pages 339–355, Paris, France, September 2006. Springer.
24. A. Linard, E. Paviot-Adet, F. Kordon, D. Buchs, and S. Charron. polyDD: Towards a Framework Generalizing Decision Diagrams. In *10^{th} International Conference on Application of Concurrency to System Design (ACSD'2010)*, pages 124–133, Braga, Portugal, June 2010. IEEE Computer Society.
25. A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In *20st International Conference on Applications and Theory of Petri Nets (ICATPN 1999)*, volume 1639 of *LNCS*, pages 6–25. Springer, 1999.
26. G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, June 1981.
27. M. Rice and S. Kulhari. A survey of static variable ordering heuristics for efficient BDD/MDD construction. Technical report, UC Riverside, 2008.
28. K. Schmidt. How to Calculate Symmetries of Petri Nets. *Acta Informaticae*, 36(7):545–590, 2000.
29. K. Strehl and L. Thiele. Symbolic model checking of process networks using interval diagram techniques. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 686–692, New York, NY, USA, 1998. ACM.
30. Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical Set Decision Diagrams and Regular Models. In *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, volume 5505 of *LNCS*, pages 1–15. Springer, 2009.