Gradient-based Variable Ordering of Decision Diagrams for Systems with Structural Units

Elvio Gilberto Amparore[1], Marco Beccuti[1], Susanna Donatelli[1]

[1] Università di Torino, Dipartimento di Informatica
{amparore,susi,beccuti}@di.unito.it

Abstract. This paper presents Gradient-Π, a novel heuristics for finding the variable ordering of Decision Diagrams encoding the state space of Petri net systems. Gradient-Π combines the structural informations of the Petri net (either the set of minimal P-semiflows or, when available, the structure of the net in terms of "Nested Units") with a gradient-based greedy strategy inspired by methods for matrix bandwidth reduction. The value of the proposed heuristics is assessed on a public benchmark of Petri net models, showing that Gradient-Π can successfully exploit the structural information to produce good variable orderings.

Keywords: Decision Diagrams, Variable Ordering, Petri Nets

1 Introduction

The use of binary decision diagrams [8] and their variants is at the base of the so-called symbolic model-checking techniques. These techniques have given an incredible boost to state-based verification of systems over the last 30 years and have given rise to a number of successful tools for automatic verification of systems, in particular for discrete event dynamic systems (DEDS) [9]: systems in which the state is composed by a set of variables, taking values on a set of finite and discrete values, and events that change the state of the system.

It is well-known that the size of the decision diagram representation of a state space heavily depends on the chosen order of the variables that represent the state of the system, and that the problem of finding such an ordering is NP-complete. Many heuristics for finding "good" variable orderings have been proposed in the past (see for example the surveys in [23,26]) and they are often crucial for the performances of model-checking tools and of decision diagram libraries [19].

Variable ordering can be *static* or *dynamic*, or, more precisely, can be used statically or dynamically. In the first case an ordering is computed before state space generation and it is kept fixed through the whole generation procedure, while in the second case a new order can be computed and applied at run-time if the decision diagram size grows too large. In this paper we concentrate on static ordering and we aim at understanding *if and how the structure of the system can be exploited for devising a good variable ordering.*

To answer this question we need to place our research in a specific context, as it is difficult to draw conclusions on the efficacy of a heuristics without referring

to a specific state-space generation technique and to an appropriate benchmark of systems on which to exercise our findings. As system specification language we consider Petri Nets, that have proven useful in modelling a large variety of DEDS, from hardware [24] to business process models [1]. The evaluation of the proposed heuristics is based on a benchmark extracted from the public model set of the Model-Checking Contest (MCC2016) [14], which includes 664 model instances. We use symbolic model-checking based on *Multi-terminal Dcision Diagrams* (MDDs) as implemented in the GreatSPN tool [4][5] based on the MDD library Meddly [7]. We assume that MDD levels encode Petri net places, i.e. we do not consider merging multiple places into a single level like in [11]. The state space generation algorithms employs *saturation* [10].

Another reason for choosing Petri nets is the large amount of literature on structural analysis techniques [12], that is to say techniques that allow to check properties of the state space (like invariant properties of the variables, finiteness of the state space and liveness of events) without building the state space itself.

Structural informations are at the base of many of the heuristics designed for the analysis of (Petri net models of) circuits, which exploit the locality of the inter-dependent variables, or the input-output dependencies, as the widely used heuristics based on fan-in [20]. However, only a subset of these heuristics can be used effectively on general Petri net models, which usually have cyclic behaviour and no clear input/output dependencies. Popular static variable ordering heuristics applied to Petri nets include, among others: The Noack and Tovchigrechko [16] methods; The Force and the Mince heuristics [3]; Bandwidth-reduction techniques like the Sloan method [27].

In a previous paper [6] we evaluated 14 different heuristics on 386 model instances (belonging to 62 parametric models) taken from the MCC2016 model set. In that paper we also tested a heuristics called *P-chain* based on P-semiflows (which are subsets of places with constant weighted sum of tokens in all reachable states) to improve variable ordering heuristics. *P-chain* showed rather poor performances, suggesting that the concatenation of P-semiflows which is at the base of the technique is not enough. In this paper we propose Gradient-Π, a new heuristics that modifies the bandwidth-reduction method of Sloan to order places on a subset-basis, using either P-semiflows or Nested Units [15] to provide such subsets. An extensive benchmark enlights that Gradient-Π has better performance than the Sloan method and better than other structural-based methods like P-chain, as well as various other state-of-the-art variable ordering methods.

Note that Gradient-Π assumes that in the MDD we assign one place per level. Other techniques have instead used P-semiflows for identifying places to be grouped in a single level [11] or for state compression for binary decision diagrams [25]. These technique can be seen as orthogonal to the proposed one.

The paper is organized as follows: Section 2 introduces the notation used in the paper; Section 3 describes the Gradient-Π heuristics; Estimation of the heuristics parameters is done in Section 3.1; The effectiveness of the new method is assessed in Section 4. Finally, Section 5 concludes the paper.

2 Background

This section first reviews the definition of Petri nets [2] and the related notions of P-semiflows and Nested Units. The section then presents a description of the two variable ordering algorithms (the Sloan method [27] and the P-chain method [6]) which influenced the definition of the proposed static variable ordering heuristics.

The Petri net (PN) is a graphical mathematical formalism that has been widely used to model and study real systems in different fields (e.g. communication systems, biological systems, power systems, work-flow management, ...). A Petri net is a bipartite directed graph with nodes partitioned into *places* or *transitions*. An example of PN is depicted in Figure 4. Places, graphically represented as circles, correspond to the state variables of the system, while transitions, graphically represented as boxes, correspond to the events that determine the state changes. The arcs connecting places to transitions (and vice versa) express the relations between states and event occurrences. Each arc is labeled with a non null natural number representing its "weight". The state of a PN is usually called a *marking* $\mathbf{m}$, a multiset on the set P of places.

Definition 1 (Petri Net). *A PN system is a tuple* $\mathcal{N} = (P, T, \mathbf{I}^-, \mathbf{I}^+, \mathbf{m_0})$*, where:* P *is a finite and non empty set of* places*;* T *is a finite and non empty set of* transitions *with* $P \cap T = \emptyset$*;* $\mathbf{I}^-, \mathbf{I}^+ : T \times P \to \mathbb{N}$ *are the* input *and* output *matrices, that define the arcs of the net and that specify their weight; and* $\mathbf{m_0} : P \to \mathbb{N}$ *is a multiset on* P *representing the* initial marking *of the net.*

A *marking* $\mathbf{m}$ (or state) of a PN is a multiset on P. A transition t is *enabled* in marking $\mathbf{m}$ iff $\mathbf{I}^-(t,p) \leq \mathbf{m}(p)$, $\forall p \in P$, where $\mathbf{m}(p)$ is the number of tokens in place p in marking $\mathbf{m}$. Enabled transitions may *fire*, and the firing of transition t in marking $\mathbf{m}$ yields a marking $\mathbf{m'} = \mathbf{m} + \mathbf{I}^+(t) - \mathbf{I}^-(t)$. Marking $\mathbf{m'}$ is said to be reachable from $\mathbf{m}$ because of the firing of t and is denoted by $\mathbf{m}[t\rangle\mathbf{m'}$.

The markings which are reachable from a given initial marking $\mathbf{m}_0$ form the *Reachability Set* ($\mathrm{RS}(\mathbf{m}_0)$). The *incidence matrix* is the matrix $\mathbf{C} = \mathbf{I}^+ - \mathbf{I}^-$, so that $\mathbf{C}_{t,p} = \mathbf{I}^+(t,p) - \mathbf{I}^-(t,p)$ describes the effect of the firing of transition t on the number of tokens in place p. Any left annuller of matrix $\mathbf{C}$, a vector $x \in \mathcal{Z}^{|P|}$ solution of the matrix equation $x\mathbf{C} = 0$ is called a *P-semiflow*.

The set of P-semiflows are at the base of the first notion of "structure" used in this paper. Indeed if x is a P-semiflow and $\mathbf{m}$ any state reachable from the initial state $\mathbf{m_0}$, we can write that $x \cdot \mathbf{m} = K$, where K is a constant value that can be computed from the initial state, $K = x \cdot \mathbf{m_0}$. This suggests that all the places that are in the same P-semiflow (that is to say the set π_x of all places p with a non-null value of $x(p)$) have a form of "circular" dependency and it is advisable to put them close together in the decision diagram [11]. The intuition is that each P-semiflow usually represents a local sub-component of a more complex model. The set π_x consists of connected places that can be used to identify a PN subnet, the Petri Net structure that will be at the basis of the algorithm proposed in the next section. All the P-semiflows of a PN can be expressed as linear combinations of the set Π_{MPS} of *minimal P-semiflows*

Algorithm 1 Pseudocode of the Sloan algorithm.

Function `Sloan`:
  Select a vertex u of the graph.
  Select v as the most-distant vertex to u with a graph visit.
  Assign to each vertex v' a gradient $grad(v') = dist(v, v')$.
  Initialize visit frontier $Q = \{v\}$
  repeat until Q is empty:
    Remove from the frontier Q the vertex v' that minimizes $P(v')$.
    Add v' to the variable ordering l.
    Add the unexplored adjacent vertices of v' to Q.

(MPS). Since the number of MPS can be exponential in the number of places, their computation is in EXPSPACE. However, on many practical cases, the set of MPS is much smaller and can be computed in almost linear time [12]. In the experimental section we shall detail on which models the computation is feasible.

The second notion of "structure" used in this paper is that of *nested unit* (NU) which is at the base of Nested-Unit Petri Nets (NUPN) [15]. NUPN are ordinary Petri nets (all arcs have weight one) with an additional structure of "nested units". Units constitute a partition of the set P of places and are characterized by having at most one place of the unit marked in any reachable state. NUPN have been developed as the target formalism for the translation from process algebra to Petri nets. Let Π_{NU} be the set of nested units of a NUPN.

Note that while the structure based on MPS can be computed on (almost) any net, the structure of NU applies only to NUPN, and is part of the model definition. This topic will be discussed with more detail in the experimental section. Both NU and MPS express an important property of mutual dependency of the involved variables in PN system. In a NU, at most one place can have a token in any marking. In a MPS, a weighted sum of tokens is constant in any marking. Therefore, we expect that a variable ordering that groups together MDD variables corresponding to places in the same NU/MPS should reduce the MDD size, due to the inter-dependence between the variables [11,25].

The Sloan algorithm for static variable ordering

The Sloan algorithm [27] is an algorithm to reorder the entries of a sparse symmetric matrix $\mathbf{A}$ around its diagonal. It computes a permutation of the rows/columns of $\mathbf{A}$ such that most of the non-zero entries are maintained as close as possible to the diagonal. Recently, the work in [21] has shown that Sloan is a promising algorithm for variable ordering of Petri net models. The idea is to express variable-variable interactions in $\mathbf{A}$. Compacting $\mathbf{A}$ around the diagonal results in an improved transition locality in the MDD. Since the Sloan method requires $\mathbf{A}$ to be *symmetric*, some form of symmetrization is needed. In our context, we define $\mathbf{A}_{i,j}$ to be non-zero iff there is a transition that connects place i with place j, regardless of the direction of the arcs.

The pseudocode of Sloan is given in Algorithm 1. The method can be divided into two phases. In the first phase it searches a pseudo-diameter of the $\mathbf{A}$ matrix graph, i.e. two vertices v, u that have an (almost) maximal distance. Usually, a

Algorithm 2 Pseudocode of the P-semiflows chaining algorithm.

Function `P-chain(`Π_{MPS}`)`:
  $l = \varnothing$ is the ordered list of places.
  $S = \varnothing$ is the set of currently discovered common places.
  Select a MPS $\pi_i \in \Pi_{\text{MPS}}$ s.t. $\max_{\{i,j\} \in |\Pi_{\text{MPS}}|} \pi_i \cap \pi_j$ with $i \neq j$
  $\Pi_{\text{MPS}} = \Pi_{\text{MPS}} \setminus \{\pi_i\}$
  $\pi_{curr} = \pi_i$
  Append $V(\pi_{curr})$ to l
  repeat until Π_{MPS} is empty:
    Select a MPS $\pi_j \in \Pi_{\text{MPS}}$ s.t. $\max_{j \in |\Pi_{\text{MPS}}|} \pi_{curr} \cap \pi_j$
    Remove $(l \cap V(\pi_j)) \setminus S$ to l
    Append $V(\pi_{curr} \cap \pi_j) \setminus S$ to l
    Append $V(\pi_j) \setminus (S \cap V(\pi_{curr}))$ to l
    Add $V(\pi_{curr} \cap \pi_j)$ to S
    $\pi_{curr} = \pi_j$
    $\Pi_{\text{MPS}} = \Pi_{\text{MPS}} \setminus \{\pi_j\}$
  return l

heuristic approach based on the construction of the *root level structure* of the graph is employed. The method then performs a visit, starting from v, exploring in sequence all vertices in the visit frontier Q that maximize a priority function $P(v')$. The function $P(v')$ guides the variable selection in the greedy strategy. It is defined as $P(v') = -W_1 \cdot \mathit{incr}(v') + W_2 \cdot \mathit{dist}(v, v')$ where v, v' are vertices in graph $\mathbf{A}$, $\mathit{incr}(v')$ is the number of unexplored vertices adjacent to v', $\mathit{dist}(v, v')$ is the distance of the shortest path between v and v', and W_1 and W_2 are two integer weights. The weights control how Sloan prioritizes the visit of the local cluster (W_1) and how much the selection should follow the gradient (W_2).

The P-chain algorithm for static variable ordering

The *P-semiflows chaining algorithm*(P-chain) is based on the idea of keeping together the places belonging to the same P-semiflow for the MDD variable ordering. The idea behind this algorithm is to maintain the places shared by two P-semiflows as close as possible in the final DD variable ordering, since their markings cannot vary arbitrarily. The pseudo-code is reported in Algorithm 2.

The algorithm takes as input the Π_{MPS} set and returns as output a variable ordering l. Initially, the MPS π_i sharing the highest number of places with any unit is removed from Π_{MPS} and saved in π_{curr}. All its places are added to l.

Then the main loop runs until Π_{MPS} becomes empty. The loop comprises the following operations. The MPS π_j sharing the highest number of places with π_{curr} is selected. All the places of π_j in l, which are not currently S (the list of currently discovered common places) are removed. The common places between π_i and π_j not present in S are appended to l, followed by the places present only in π_j. After these steps, S is updated with the common places in π_i and π_j, and π_j is removed from Π_{MPS}. Finally π_{curr} becomes π_j, completing the iteration.

Evaluation methodology

When testing multiple methods on a set of models, we need one or more score

functions to determine the efficacy of the methods. We mainly consider the peak node size of the constructed MDD as a basis for the score functions, since it is the upper bound of the MDD computation both in terms of memory and time. Let $\mathcal{A}$ be a set of considered methods, let $\mathcal{I}$ be the set of model instances, and let i be a model instance solved by algorithms $\mathcal{A} = \{a_1, \ldots, a_m\}$ with peak nodes $P_i = \{p_{a_1}(i), \ldots, p_{a_m}(i)\}$. We consider three score functions for each instance i:

- $Solved_a(i)$: 1 if the RS generation using the variable ordering provided by algorithm a finishes in the time/memory constraints, 0 otherwise;
- $Optimal_a(i)$: 1 if algorithm a found the variable ordering among the tested method which leads to the smallest MDD peak size, 0 otherwise;
- *Normalized Score* (NS): is a value between 0 and 1 which weights the "quality" of the variable ordering, and it is defined as:

$$\text{NS}_a(i) = 1 - \frac{\min\{p \in P_i\}}{p_a(i)} \tag{1}$$

  The optimal algorithm for an instance i receives a NS score of 0. If an algorithm does not terminate in the given time/space limits, we arbitrarily assign a score of 1 to that algorithm.

3 The Gradient-Π Algorithm

This section presents the Gradient-Π heuristics, the main contribution of the paper. It is an hybrid algorithm that combines the features of both the Sloan method (which is *gradient-based*) and the P-Chain method (which is based on the idea of ordering the *structural units*). As observed in [6] Sloan performs rather well, especially in terms of the number of "solved" models (models on which the state space is generated within the given time and space constraints), but it is not always the best one. Methods like Tovchigrechko, that takes into consideration some aspect of the net graph (like number of input and output arcs of a place) often exhibit better performances. That analysis also showed that a heuristics like Force, that usually reaches intermediate performances, when modified to include information on the NU (method called Force-NU in [6]) can go beyond the performance of the best performers (like Tovchigrechko and Sloan) on the set of NUPN models.

The idea behind the Gradient-Π heuristics is indeed to combine the generality of Sloan, that results in a large number of solved models, with the exploitation of structural informations that could result in better performances. The exploitation of structural info is similar to the idea behind the P-chain algorithm. According to the benchmark results reported in [6], P-chain does not perform well, and our hypothesis is that its poor performances have two motivations: 1) there is no clever choice of the order in which the MPSs are considered and 2) there is no indication on how to order the variables of the same MPS (which corresponds to linearize the places inside the MPS which is basically a cyclic structure). In Gradient-Π the choices associated to the two points above are resolved using a gradient-based approach mutuated from Sloan algorithm

Algorithm 3 Pseudocode of the Gradient-Π heuristics.

Function `Gradient-`$\Pi(v_0, \Pi)$:
  // Phase 1: establish a gradient based on v_{first} and v_{last}.
  Start a graph visit from v_0. Let v_{last} the variable that maximizes $dist(v_0, v_{\text{last}})$.
  Start a visit from v_{last}. Let v_{first} be the variable that maximizes $dist(v_{\text{last}}, v_{\text{first}})$.
  for each variable $v \in V$:
    Compute $grad(v)$.
  $S \leftarrow \varnothing$
  $l \leftarrow [\,]$
  // Phase 2: Linearize the elements of Π along the gradient.
  while exists at least one $\pi \in \Pi$ with $\pi \setminus S \neq \varnothing$:
    for each element $\pi \in \Pi$ with $\pi \setminus S \neq \varnothing$:
      Compute $score(\pi)$.
    Let $\pi_{\max}$ be the element with maximum $score(\pi)$ value.
    // Phase 3: Linearize the variables in the selected element π_{max}.
    Append variables in $(\pi_{\max} \setminus S)$ to l in ascending gradient order.
    $S \leftarrow S \cup \pi_{\max}$.
  Append all variables in $(V \setminus S)$ to l in ascending gradient order.
  return l.

The method takes as input an initial vertex and a structure Π of places. We consider two different applications of this method: Gradient-P uses the set of minimal P-semiflows Π_{MPS} as input, and Gradient-NU uses the set of Nested Units Π_{NU} as input. A pseudo-code is given in Algorithm 3.

The algorithm is subdivided into three main phases. In the first phase, the algorithm identifies a *pseudo-diameter* of the system graph, whose vertices v_{first} and v_{last} are the opposite ends. Identification is done using two graph visits. Alternatively, a *root level structure* [18] can be used for this task. The identified diameter is a *pseudo-diameter*, since there is no guarantee that $(v_{\text{first}}, v_{\text{last}})$ forms the maximum diameter of the graph. However, this method usually finds a reasonable approximate of the pseudo-diameter, and it is very fast. Once the pseudo-diameter is established, a scalar *gradient* is assigned to each vertex v. The definition of $grad(v)$ is the subject of study in the next section.

The second phase of the algorithm takes one element of Π at a time, according to the gradient order. To do so, it assigns a *score* value to each element, with the goal of taking the one with the maximum score. The algorithm tracks the set of variables S that have already been inserted in the variable ordering l. Again we shall discuss the considered score function in the next section. Once the element with the maximum score $\pi_{\max}$ has been identified, the algorithm performs the third phase, which consists in a greedy local optimization of the variable ordering. Variables in $(\pi_{\max} \setminus S)$ are ranked according to the $grad(v)$ value, and appended to l. The method continues selecting elements $\pi \in \Pi$ until all variables have been inserted in l. For completeness of the method, if some variable is not covered by any element of Π, it is appended at the end of l. The implemented method also considers the case where the graph is not fully connected. In that case, the algorithm is run separately for each connected component.

3.1 Parameter Estimation of the Score and Gradient Functions

The Gradient-Π method depends on a set of different parameters and functions. We defined a set of experiments to estimate empirically these parameters. The targeted questions are the following:

Q1. What is the best strategy for the selection of the initial vertex v_0?
Q2. What $score(\pi)$ function should be used?
Q3. What $grad(v)$ function should be used?

To answer these questions, we run a prototype implementation of the Gradient-Π method on a benchmark composed of 51 PN model instances, made by a seection of the smallest instances of the MCC2016 benchmark, used for the Gradient-Π assessment in Section 4. We consider Π_{MPS} as the input set Π.

Q1. We consider three different criterias for the selection of v_0: 1) Take v_0 as the first place P_0 in the net; 2) Take v_0 as a random place in the net; and 3) Take v_0 as the place with the maximum number of input/output arcs.
We run Gradient-Π on the test set, collecting the peak MDD size for each run. Figure 1 shows the comparative results obtained by the three criterias.

Each plot in Figure 1 shows the MDD peak values of the compared runs, on a log-log scale. Each dot represent a model run, where the x and y coordinates are the MDD peaks obtained in two of the three tested configurations. The data shows that the method is not very sensible to the selection of v_0, with the third configuration only marginally better than the other two. However, since the third configuration is also the typical strategy for the initial vertex selection of Sloan, we adopt it for our implementation of Gradient-Π.

Q2. The score function for an element $\pi \in \Pi$ should balance these quantities:

1. An element π that has many variables already in S should be preferred;
2. The score should be proportional to the gradient;
3. The element cardinality $|\pi|$ can be used as a weight parameter.

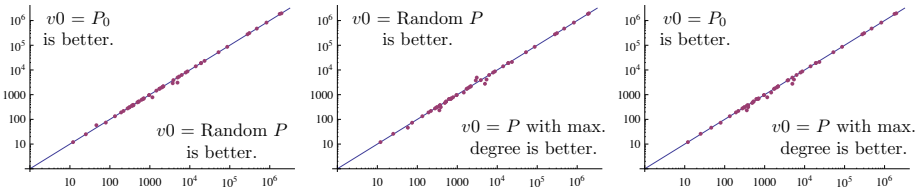


Fig. 1: MDD peaks obtained for different choices of v_0.

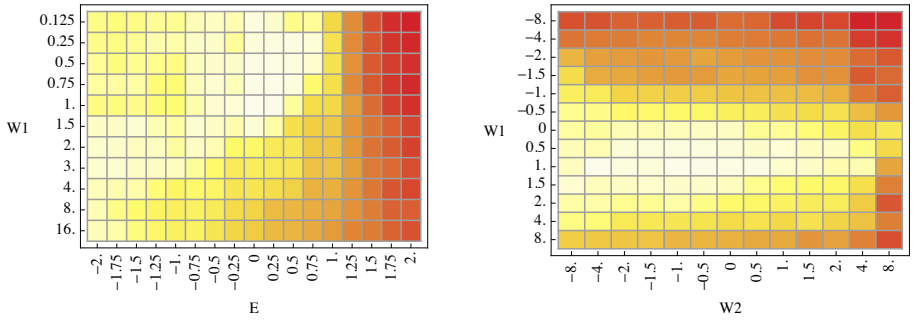


Fig. 2: Normalized scores on the parameter space of $W_1 \times E$ and $W_1 \times W_2$.

To encode these three desiderata into a single score function, we defined (empirically) two parametric score functions:

$$score_{\text{mult}}(\pi) = \overbrace{\left(W_1 \cdot \sum_{v \in \pi \cap S} grad(v) - \sum_{v \in \pi \setminus S} grad(v)\right)}^{d} \cdot \left|\pi \setminus S\right|^{E \cdot sign(d)} \quad (2)$$

$$score_{\text{add}}(\pi) = W_1 \cdot \sum_{v \in \pi \cap S} grad(v) - \sum_{v \in \pi \setminus S} grad(v) + W_2 \cdot \left|\pi \setminus S\right| \quad (3)$$

These functions are inspired by both the weight function of Sloan and the weight function of the Noack method [16]. In the function $score_{\text{mult}}(\pi)$ the weight of the element size is a multiplicative factor, while for the function $score_{\text{add}}(\pi)$ it is an additive factor. The power in Eq. (2) consideres the sign of d to ensure that when E is positive the score increases for increasing values of $|\pi \setminus S|$ regardless of the sign of d. Vice versa, it should decrease when E is negative. Both functions are controlled by a set of parameters (W_1 and E for the first, W_1 and W_2 for the second). We run a set of experiments on the parameter space of $W_1 \times E$ and $W_1 \times W_2$, in order to determine the best values for both.

Figure 2 shows the normalized scores of Eq. (1) of the runs on the 51 model instances considered. The left plot shows the NS results when the $score_{\text{mult}}(\pi)$ is used, for varying (W_1, E). Similarly, the right plot shows the NS results when using the $score_{\text{add}}(\pi)$, for varying (W_1, W_2). Lighter blocks have a smaller NS, which means that the algorithm running on that pair of parameter's values computes better variable orderings. The resulting plots are remarkably smooth. Interestingly, the left one has a local minimum in W_1=1, E=0, while the right one has a local minimum in W_1=1, W_2=0. This analysis suggests that the element size does not bring any advantage to the score function, neither in multiplicative nor in additive form. Therefore, the final score function that we adopt is:

$$score(\pi) = \sum_{v \in \pi \cap S} grad(v) - \sum_{v \in \pi \setminus S} grad(v) \quad (4)$$

Q3. Since the Gradient-Π method works using a pseudo-diameter, a critical element is the gradient function $grad(v)$. Let $b = \max_{v \in V} dist(v_{\text{last}}, v)$. We consider

two gradient functions: 1) An *integer* function: $grad(v) = dist(v_{\text{first}}, v)$, based solely on v_{first}; 2) A *fractional* function: $grad(v) = dist(v_{\text{first}}, v) + \frac{1}{b}\big(b + 1 - dist(v_{\text{last}}, v)\big)$. The first function is the same used in the Sloan method. The second function also considers the distance to v_{last} as a fractional part. We investigate if this addition brings benefits to the method.

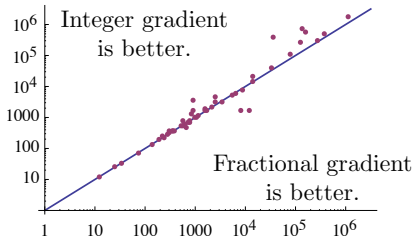


Fig. 3: Effect of the two tested gradient functions.

Figure 3 shows the comparison of the two gradient functions. The plot compares the MDD peaks obtained running the same test using the two different functions. The results show that the integer function is slightly better than the fractional one in some cases, but the advantage cannot be determined clearly.

Example 1 (Gradient-Π example on a Petri net using P-semiflows).

We now illustrate the Gradient-Π algorithm on a small Petri net, taken from the MCC2016 model set. The model is called "SwimmingPool" and describes a sort of protocol to use a pool[1]. We use the set of P-semiflows Π_{MPS} for the input elements Π. Figure 4 shows in the upper-left frame the Petri net model. Numbers written in the places are the ordering computed by the Gradient-P algorithm. Numbers written aside of each place are the computed integer gradient. The three replicas of the model on the right show the three MPS π_1, π_2 and π_3.

The algorithm computes three iterations of the second phase loop. In the first iteration, π_1 is selected, since it has the highest score. Since no variable has been selected yet, all variables in π_1 are taken, in gradient order. In the second iteration, π_2 is selected, which has already two variables in S. Finally, π_3 is selected. Intuitively, the P-semiflows of this model represents closed loops where a constant token quantity circulates. The algorithm attaches these "loops" one after the other, following the gradient order.

4 Comparison of Gradient-Π with Other Heuristics

The goal of this section is to test the effectiveness of the proposed heuristics against other commonly used variable ordering algorithms for Petri net models. We use the evaluation methodology of [6]. We only consider static variable ordering methods. The set $\mathcal{A}$ of considered methods is:

[1] Model details can be found in `http://mcc.lip6.fr/pdf/SwimmingPool-form.pdf`.

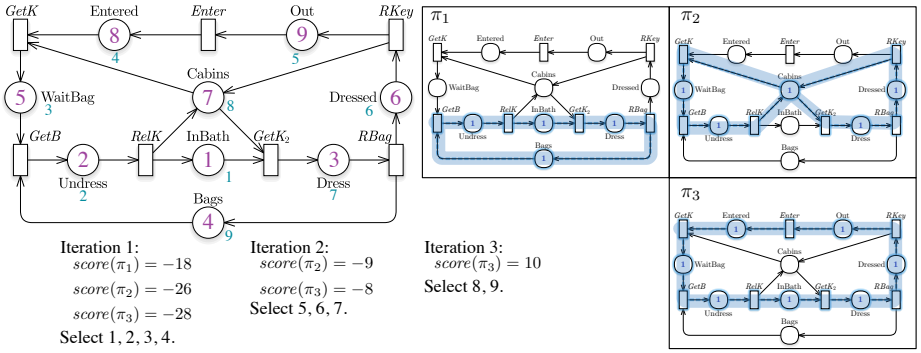
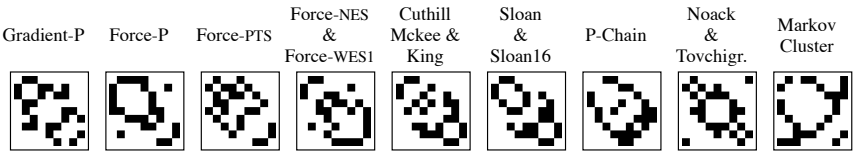


Fig. 4: Gradient-P run on the Swimming pool model.

- **Force-{PTS, NES, WES}**: variants of the Force heuristics [3] where 200 orderings are generated, and the one with the smallest score is selected. The considered score functions are: *point-transition span* (PTS), *normalized event span* (NES) and *weighted event span* (WES), respectively [26].
- **Force-{P, NU}**: variant of Force where structural elements are also centers of gravity, along with the events. Elements can be either Π_{MPS} or Π_{NU}.
- **Cuthill-Mckee** heuristics, defined in [13].
- **King** heuristics, defined in [17].
- **Sloan/Sloan16** heuristics, defined in [27] and recalled in Section 2. We consider two variations of this method: Sloan uses $\frac{W_1}{W_2} = \frac{1}{2}$, while Sloan16 uses $\frac{W_1}{W_2} = \frac{1}{16}$ (the parameters used in [21] and in [6], respectively).
- **P-Chain** heuristics: defined in [6] and recalled in Section 2
- **Noack** heuristics, defined in [22], is a greedy heuristics for Petri net models that tries to minimize the locality of the events in the ordering.
- **Tovchigrechko** heuristics, defined in [16], is a variation of the Noack heuristics with a different selection criteria.
- **Markov Cluster** uses the Markov cluster algorithm [28] to identify variable clusters and group them together.

Example 2 (Tested algorithms on the running example model). Figure 5 shows the variable orderings obtained with the tested methods on the SwimmingPool model. The model has a very clear structure of partially overlapped P-semiflows, as shown in Figure 4, and we expect Gradient-P to perform well as it is designed to exploit this type of structure.

Each block shows the algorithm name, the symmetric adjacency matrix of the ordered model, and the performances of the state space generation using that ordering. Algorithms are ordered according to peak nodes (smallest to highest). Some algorithms obtained the same ordering, and have been grouped together in a single column. The matrix has a black square in (i, j) if there exists an event that links the variable at level i with the variable at level j. In this visualization we do not consider the event directionality, and we make the matrix symmetric.



	Gradient-P	Force-P	Force-PTS	Force-NES & Force-WES1	Cuthill Mckee & King	Sloan & Sloan16	P-Chain	Noack & Tovchigr.	Markov Cluster
Nodes:	2629	13154	14144	13364	2684	2684	22179	40134	13124
Peak:	6802	17531	17838	20029	21196	25088	61362	230483	430907
Time:	0.607	1.234	1.217	1.582	1.937	2.259	3.653	27.325	28.869
NS:	0.000	0.613	0.619	0.661	0.680	0.729	0.890	0.971	0.985

Fig. 5: Comparison of Gradient-P with the other tested methods.

The rows below each matrix report the number of MDD nodes, the peak nodes, the time needed to construct the MDD, and the normalized score assigned to that variable order. In this example, Gradient-P performs better and thus receives an NS score of 0. All the other algorithms receive a score that is proportional to how much their peak node size departs from the minimum peak node size found for that instance. Note that Gradient-P has a peak size which is almost one third of the second best ordering algorithm (Force-P, which is also an algorithm that exploits the net structure), one fourth of Sloan (which is a generic algorithm that does not exploit P-semiflows). Note that algorithms that have been specifically performed for Petri nets, like Tovchigrechko and Noack perform rather poorly on this example, with peak sizes more than 30 times bigger than Gradient-P.

4.1 Empirical assessment on the benchmark

The model of the previous section is just an example of a structure on which Gradient-P performs very well. In this section we test Gradient-P and Gradient-NU on a broader set of models, to evaluate their average performance. The evaluation is based on a benchmark with two subsets of models, taken from the 664 instances of the Model Checking Contest (MCC) 2016 [14] model set:

- The set $\mathcal{I}_{\text{MPS}}$ where P-semiflows are computable in less than 30 seconds. The set is made by 408 model instances, belonging to 45 models. In 294 of these instances at least one algorithm finishes in the time/memory limits.
- The set $\mathcal{I}_{\text{NU}}$ of NUPN instances, with well identified nested units that correspond to process algebra terms. The set is made by 80 model instances belonging to 12 different models. For 67 of these instances, at least one algorithm finishes in the time/memory limits.

The excluded instances either are not NUPN, or the MPS set is not computable. All computations have been done using the GreatSPN tool [5], with a maximum of 4GB of memory and 60 minutes of time.

Table 1(left) reports the results for the $\mathcal{I}_{\text{MPS}}$ set. Since in the MCC model set the number of instances-per-model vary largely (some have just one, others have up to forty instance), the table reports the results on a per-model and per-instance basis. For each algorithm $a \in \mathcal{A}$, the table indicates the number of

Table 1: Benchmark results on the $\mathcal{I}_{\text{MPS}}$ and $\mathcal{I}_{\text{NU}}$ model sets.

	Models (45)			Instances (294)		
Method	solv.	opt	NS	solv.	opt	NS
Gradient-P	**28.61**	**12.42**	**12.73**	240	**76**	**137.22**
Sloan16	**28.61**	4.60	21.46	**244**	53	181.29
Sloan	28.51	4.84	22.07	242	43	188.83
Tovchigr.	28.29	5.62	18.80	240	52	169.27
P-Chain	26.81	2.61	24.88	229	21	223.33
Noack	26.31	4.34	19.21	228	36	176.99
Force-NES	22.69	2.90	23.98	192	23	209.52
Force-P	22.40	4.67	20.75	199	30	188.09
Force-PTS	22.33	2.53	24.32	189	19	213.19
Force-WES	22.04	3.40	23.27	190	23	206.30
Cuthill-M.	21.32	2.74	24.63	214	52	180.11
King	20.80	1.77	24.99	206	21	199.89
MarkovCl.	18.61	1.30	27.29	174	15	235.13

	Models (12)			Instances (67)		
Method	solv.	opt	NS	solv.	opt	NS
Gradient-NU	**9.82**	**7.03**	**2.64**	50	**34**	**23.72**
Sloan16	9.47	0.45	8.10	**55**	9	45.40
Sloan	9.37	1.31	8.56	53	7	49.07
Tovchigr.	8.82	0.60	8.51	44	12	47.71
Noack	7.83	0.15	8.34	42	3	50.66
Force-NES	6.30	0	10.22	35	0	61.85
Force-PTS	5.74	0	10.43	27	0	64.24
Force-NU	5.71	1.30	8.78	38	5	54.59
Force-WES	5.41	0	10.11	34	0	61.72
MarkovCl.	4.91	0	9.19	24	0	60.94
Cuthill-M.	2.53	0	10.58	21	0	65.59
King	2.49	0	10.59	21	0	65.71

models for which a terminates (solv.), the number of models where the a found the best variable ordering (opt) and the total NS score for that algorithm. The last three columns replicate the same data for the model instances.

The computation of the NS score for the per-instance analysis of algorithm a just sums the value of $\text{NS}_a(i)$ of equation (1) over all instances i. In the per-model analysis the sum is over the $\text{NS}_a(m)$ values for each model m, where the NS score of a model m is computed as $\text{NS}_a(m) = \sum_{i \in m} \frac{1}{|m|} \text{NS}_a(i)$, to balance models that have many instances. Analogous rescaling is done for the number of solved and optimally solved models (which results in fractional numbers).

From the data it emerges that Gradient-P and the variation of Sloan that we propose (Sloan16) are the best performers. In particular if we observe the per-model results, Gradient-P has a significant margin in finding the optimal ordering (thus reducing the MDD peak size) on both Sloan16 and Sloan (and even on their sum) and a much better NS score both with respect to Sloan/Sloan16 and with respect to the second one on the NS column (Tovchigrechko). Surprisingly, neither P-chain nor Force-P methods, which are both P-semiflow based algorithm, reaches similar performances to that of Gradient-P. From these positive results we conjecture that the combination of a Sloan-like gradient order with the structural information of P-semiflows produces variable orderings that are better that those generated by algorithms that use just one of the two elements (gradient or P-semiflows).

In the per-instance analysis, Gradient-P has the best NS score, actually significantly better than the second one in the column (which is, again, Tovchigrechko), a number of solved instances that is only 1.7% less than the best performer on solved instances (Sloan16) and the best number of optimal solved instances. This last results is nevertheless not particularly relevant, since the splitting of Sloan on two variations (Sloan and Sloan16) may have lead to an

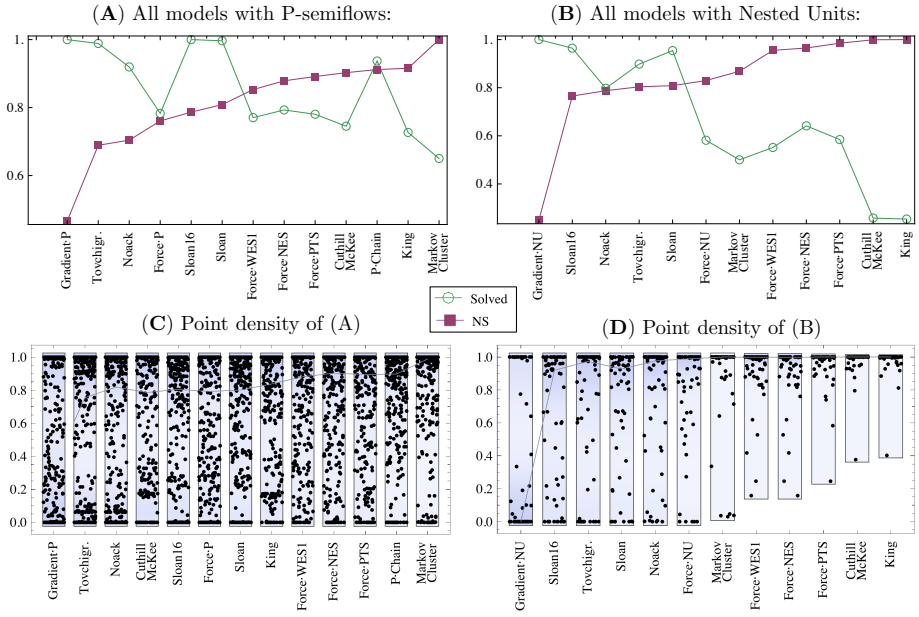


Fig. 6: Performance obtained using Gradient-Π on the benchmark.

underestimation of the value with respect to a benchmark in which only one of the two is present.

Table 1(right) reports the results for the $\mathcal{I}_{\text{NU}}$ set. The results are similar to that of the previous case but even more striking for what concerns the number of optimally solved models/instances: Sloan, Gradient-NU and Tovchigrechko methods occupy the top positions in terms of solved models/instances, but Gradient-NU finds the optimal variable ordering among the tested methods more often than the others, with a significantly lower NS score. The number of solved instances is now worse than the best one by 10%, but, as explained before, the results per instance are less stable, since it is enough to have a single model with many instances, on which an algorithm does not perform well, to badly influence the results.

From the data analysis it seems that Gradient-P and Gradient-NU finds better variable ordering compared to state-of-the-art methods like Sloan and Tovchigrechko. To confirm this observation, we look at the point density of the NS scores of each instance. Figure 6 reports on the top row the plots of the NS score and the count of solved models (normalized from 0 to 1), for each algorithm. Plots (C) and (D) shows the NS point density of each run in the benchmark. In the bar of algorithm a there is a single black point for each $\text{NS}_a(m)$, which allows to understand the distribution that makes up the NS value reported on Tables 1. From the plot (C) emerges that Sloan and Tovchigrechko methods are more polarised with a higher concentration of lower scores in the upper part of the diagrams (higher NS scores), while the behaviour of Gradient-P is

more distributed. Note that also the algorithms of the Force type show a more distributed values of NS than Sloan and Tovchigrechko. The small number of $\mathcal{I}_{\mathrm{NU}}$ instances do not allow to draw very definite conclusions, but apparently the trend of plot (D) is similar to that observed in plot (C), were Sloan/Tovchigrechko have many more instances with high NS scores.

5 Conclusions

Motivated by the aim of understating if and how the structure of the system can be exploited to improve the performance of *gradient-based* algorithms for devising a good variable ordering, in this paper we proposed a new algorithm for statically computing a variable ordering of DDs that exploits the net structure. The algorithm combines the features of the Sloan method, with the ideas of the P-Chain method. It finds the variable ordering by sorting a set of structural units of the system along a Sloan-like gradient. The structural units should be available or computable, which may limit the applicability of the method on a subset of models. In practice, these units are computable on most models in a reasonable time. For the set of tested models with structural information (294 instances), the efficacy of the proposed algorithm emerges, producing better results over the set of considered state-of-the-art variable ordering methods (13 were tested). Our tests show that the combination of a gradient-based order with the structural units is effective in reducing the MDD peak size. Parameter estimation techniques were used to tune the internals of the proposed heuristics. This allowed the identification of the range of the internal score function coefficients that have been used for the 294 instances test.

We would like to extend the method to formalisms other than Petri nets (like process algebra models or workflows models), to see if the observed performance is consistent. In addition, other kinds of structural informations could be exploited, as for instance the min-cut partitioning.

References

1. Van der Aalst, W.M.: The application of petri nets to workflow management. Journal of circuits, systems, and computers 8(01), 21–66 (1998)
2. Ajmone-Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley & Sons (1995)
3. Aloul, F.A., Markov, I.L., Sakallah, K.A.: FORCE: A fast and easy-to-implement variable-ordering heuristic. In: Proc. of GLSVLSI. pp. 116–119. ACM, NY (2003)
4. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 Years of GreatSPN, chap. In: Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi, pp. 227–254. Springer, Cham (2016)
5. Amparore, E.G., Beccuti, M., Donatelli, S.: (Stochastic) model checking in GreatSPN. In: Ciardo, G., Kindler, E. (eds.) 35th Int. Conf. Application and Theory of Petri Nets and Concurrency, Tunis. pp. 354–363. Springer, Cham (2014)
6. Amparore, E.G., Donatelli, S., Beccuti, M., Garbi, G., Miner, A.: Decision diagrams for Petri nets: which variable ordering? In: Petri Net Performance Engineering conference (PNSE). pp. 31–50. CEUR-WS (2017)

7. Babar, J., Miner, A.: Meddly: Multi-terminal and edge-valued decision diagram library. In: Quantitative Evaluation of Systems, International Conference on. pp. 195–196. IEEE Computer Society, Los Alamitos, CA, USA (2010)
8. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35, 677–691 (August 1986)
9. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
10. Ciardo, G., L uttgen, G., Siminiceanu, R.: Saturation: An efficient iteration strategy for symbolic state-space generation. In: TACAS'01. pp. 328–342 (2001)
11. Ciardo, G., L uttgen, G., Yu, A.J.: Improving static variable orders via invariants. In: ICATPN 2007: 28th Int. Conf, Poland. pp. 83–103. Springer, Berlin (2007)
12. Colom, J.M., Silva, M.: Convex geometry and semiflows in P/T nets. a comparative study of algorithms for computation of minimal P-semiflows. In: Advances in Petri Nets 1990. pp. 79–112. Springer, Berlin (1991)
13. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proc. of the 1969 24th National Conference. pp. 157–172. ACM, New York (1969)
14. F. Kordon et all: Complete Results for the 2016 Edition of the Model Checking Contest. `http://mcc.lip6.fr/2016/results.php` (June 2016)
15. Garavel, H.: Nested-Unit Petri Nets: A structural means to increase efficiency and scalability of verification on elementary nets. In: 36th Int. Conf. Application and Theory of Petri Nets, Brussels. pp. 179–199. Springer, Cham (2015)
16. Heiner, M., Rohr, C., Schwarick, M., Tovchigrechko, A.A.: MARCIE's secrets of efficient model checking. In: Transactions on Petri Nets and Other Models of Concurrency XI. pp. 286–296. Springer, Heidelberg (2016)
17. King, I.P.: An automatic reordering scheme for simultaneous equations derived from network systems. Journal of Numerical Methods in Eng. 2(4), 523–533 (1970)
18. Kumfert, G., Pothen, A.: Two improved algorithms for envelope and wavefront reduction. BIT Numerical Mathematics 37(3), 559–590 (1997)
19. Lu, Y., Jain, J., Clarke, E., Fujita, M.: Efficient variable ordering using a BDD based sampling. In: Proceedings of the 37th Annual Design Automation Conference. pp. 687–692. DAC '00, ACM, New York, NY, USA (2000)
20. Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A.: Logic verification using binary decision diagrams in a logic synthesis environment. In: IEEE Int. Conf. on Computer-Aided Design (ICCAD). pp. 6–9 (Nov 1988)
21. Meijer, J., van de Pol, J.: Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In: NASA Formal Methods, 2016. pp. 255–271. Springer, Cham (2016)
22. Noack, A.: A ZBDD package for efficient model checking of Petri nets (in German). Ph.D. thesis, BTU Cottbus, Department of CS (1999)
23. Rice, M., Kulhari, S.: A survey of static variable ordering heuristics for efficient BDD/MDD construction. Tech. rep., University of California (2008)
24. Roig, O., Cortadella, J., Pastor, E.: Verification of asynchronous circuits by BDD-based model checking of Petri nets, pp. 374–391. Springer, Berlin (1995)
25. Schmidt, K.: Using Petri net invariants in state space construction. In: TACAS 2003, 9th Int. Conf. pp. 473–488. Springer, Berlin (April 2003)
26. Siminiceanu, R.I., Ciardo, G.: New metrics for static variable ordering in decision diagrams. In: 12th Int. Conf. TACAS 2006. pp. 90–104. Springer, Heidelberg (2006)
27. Sloan, S.W.: An algorithm for profile and wavefront reduction of sparse matrices. International Journal for Numerical Methods in Engineering 23(2), 239–251 (1986)
28. Van Dongen, S.: A cluster algorithm for graphs. Inform. systems 10, 1–40 (2000)