First difference principle applied to modular/questionable-width, clique-width, and rank-width of binary structures

Laurent Lyaudet*

October 25, 2020

Abstract

In this article, we study various widths/decompositions of binary structures under the light of first difference principle. We show the equality of modular-width and questionable-width, although questionable representation is more primitive/crude than modular/clan decomposition. Using first difference principle, we show that clique-decomposition of binary structures has a sequential equivalent, that we call clique-questionable representation. Last but not least, we give remarks on the true validity of rank-width of binary structures.

Current version : 2020/10/24

Keywords : first difference principle, binary structures, 2-structures, labelled 2-structures or labelled directed graphs or edge-coloured directed graphs, graphs, labelled property graph model, graph databases, orders, lexicographic, questionable representation, questionable-width, hierarchical decompositions, clique-width, clique-questionable-width, clique-questionable representation, rank-width, maximum rank-width under injective mapping, minimum rank-width under injective mapping

1 Introduction

Abstract of the introduction: Be functional (pun intended). To obtain a matrix apply Zermelo's axiom two times. To obtain a reversible matrix apply Zermelo's axiom one more time.

Apologies: We do science as a hobby, it is not our daily job and there is an impact on the quality of the bibliography. For an unpublished work we did in 2015, we started doing bibliographic search during 9 months, but all the gathered references were lost when a hacker erased all our files on our laptop. Since then, we chose to publish our ideas on arXiv and correct the bibliography afterwards. For preparing this article, we have read entirely the book "The Theory of 2-structures" by Ehrenfeucht et al. (1999), the thesis of Bui-Xuan (2008), and a few articles.

*https://lyaudet.eu/laurent/, laurent.lyaudet@gmail.com

In this article, we study various widths/decompositions of binary structures under the light of first difference principle. First difference principle is the principle at the heart of many mathematical objects:

- lexicographic order, where it is used before any length consideration,
- hierarchic order, positional notation or positional numeral system, where it is used after length consideration, etc.

A simple example is to compare the integers 111 and 143. We will look at the first difference between 111 and 143 since they have the same length/number of digits. It occurs on position 2 where we have to compare the digit 1 with the digit 4. Thus it is at least 5000 years old. To our knowledge, it was named "Principe de première différence" by Sierpiński (1932). It was previously used by Hausdorff (1907) for universal orders. The idea is simply to define a mathematical object with a sequence of "smaller" (in most cases, finite) objects/items/digits. Then two objects/sequences can be compared, linked by a relation resp., according to the order, relation resp., between the first different elements in the sequences. These objects (for example, integers) defined by sequences (for example, of decimal digits) are in turn elements of (a set) another mathematical object (for example, $\mathbb{N}$), and at least one mathematical property (for example, order) of these subobjects (for example, integers) in these bigger object (for example, $\mathbb{N}$) is faithfully preserved by these sequences of items (for example, of decimal digits) under the first difference principle. Thus first difference principle is used for binary structures : orders, graphs, (labelled) 2-structures. To our knowledge, a "3-way" first difference principle (like we have 3-way merge for source code) is yet to be defined, if it is possible (it should be first differences principle).

After studying first difference principle for orders in Lyaudet (2018) and Lyaudet (2019), it was natural for us to consider that this principle could be used for binary structures by applying Zermelo's axiom three times. The third application of Zermelo's axiom yielding a *reversible* binary structure. We illustrate this by showing how to convert a labelled properties graph model used in graph databases into an edge-coloured directed simple graph. These graphs are studied under different names like labelled directed graphs or labelled 2-structure (see Ehrenfeucht et al. (1999)).

We first present our logical framework that is slightly unconventional: (graph theory?) logic uses more frequently relations than functions; for example, a graph using coloured edges where there are at most k distinct colours will be represented by k distinct binary relations; we adopt a "functional" point of view on graphs that is closer to matrices; thus a k-edge-coloured graph will use a single binary function with k distinct values, a relation being a function with true, false values. Usually in logic, almost everything goes in the universe/domain of the logical structure to be studied, except for the logical values true, false, the logical quantifiers and connectives, and the relations/functions of the structure. The functional point of view makes appear more clearly that, for many use cases, the universe of the structure can be split between:

- an *internal domain* made of the "true/real" elements of the structure, like vertices and edges for graphs/binary structures,
- an *external domain* made of auxiliary elements like the k distinct values of the adjacency function used for k-edge-coloured graphs.

Our goal will be to use the class of the ordinals as a standard external domain (further restricted to the set of integers when possible, for example if the number of vertices of the binary structure is finite).

We stress that we do not want to solve the whole theory of the ordinals or of the integers, when we are studying some particular binary structure. It is an external problem.

Consider a binary signature $\mathcal{S}$ of unary or binary relations and functions. A logical structure with an internal domain and an external domain using this signature, such that the domain of relations and functions is in the internal domain and the image set of functions is in the external domain is what we call a binary structure. With the functional point of view in mind, we consider that relations are functions with values 0 for false or 1 for true. Thus, without loss of generality we have only functions. We will use the term properties for unary functions as is standard for unary relations, and we will use the term adjacencies for binary functions. Hence, $\mathcal{S} = ((P_i)_{i\in\alpha}, (A_j)_{j\in\beta})$, where $(P_i)_{i\in\alpha}$ are the properties indexed by the ordinal α, and $(A_j)_{j\in\beta}$ are the adjacencies indexed by the ordinal β. Because of the functional point of view, the atomic formulas must use an equality, an inequality, or an external binary relation like an order relation, between the result of a function and a constant of the image set of the function: for example, $P(x)$ becomes $P(x) = 1$, $\neg P(x)$ becomes $P(x) = 0$. For example, we can have for edge-weighted graphs atomic formulas like $A(x, y) = 3$, $A(x, y) \leq 12$, etc. It is assumed for practical purposes that evaluating such "atomic formulas" will always be very efficient, the equality or other provided relations on the external domain being trivial to evaluate. In this respect, practical cases may use more complex atomic formulas: comparing two functions values like $A(x, y) \leq A(x, z)$, or using simple arithmetic over the integers, for example. We present limited results but it should be clear that the allowed atomic formulas could/should be extended with further work.

Having a set of arcs with label and properties, as in labelled properties graph model used in graph databases is an example of binary structures. We explain further this point: Consider a graph database using labelled properties graph model, where you have vertices representing persons, cars, and addresses, and the following directed relations with properties:

- "is_mother_of" links a mother to its child with a property "age_at_birth" giving the age of the mother when she gave birth to this child,
- "is_father_of" links a father to its child with a property "age_at_birth" giving the age of the father when his wife gave birth to this child,
- "play_tennis_with" links two persons,
- "has_home_at" links a person to the address vertex corresponding to its home,
- "owns" links a person to its car with a property "buying_date" and a property "buying_location".

The corresponding internal domain will be made of the vertices and the arcs of this labelled properties graph. (For this example, the adjacencies functions will be "incidencies" functions but later on what will remain will be adjacencies.) We have 3 properties

that apply to vertices : "is_a_person", "is_a_car", "is_an_address". We may have other properties that apply to vertices like : "birth_date" (the domain of "birth_date" should be a subset of the domain of "is_a_person"), etc. We have two adjacency functions (but they are semantically incidency functions in the labelled properties graph, we started with): In, and Out, linking an arc with its in-vertex, resp. out-vertex. The name of the directed relations is transformed into a special arc property "relation_name". Thus, if Isabelle is the mother of Charles, we have a vertex Isabelle, a vertex Charles, an arc a, the property "is_a_person" is true on Isabelle and Charles, the adjacencies of $\mathrm{In}(a, Isabelle)$ and $\mathrm{Out}(a, Charles)$ are true, $relation\_name(a) = is\_mother\_of$ and we may have $age\_at\_birth(a) = 31$. Other arc properties in the labelled properties graph are expressed by properties defined on a subset of the arcs. We have the "null" constant when we want to express that the property is not stored in the graph database for some vertex or arc. For example, we may have $buying\_location(a) = \mathrm{null}$ on the arc between Cécile and her car, because she inherited her car from her passed away grand-mother.

Monadic second order logic with (internal) domain restricted to vertices and a single adjacency relation Adj is usually denoted by MS_1. The Adj relation is symmetric when the simple graph is symmetric, arbitrary when the simple graph is directed. We extend MS_1 into PMS_1 by allowing (external) properties (unary functions) on vertices. Monadic second order logic with (internal) domain restricted to vertices and edges and a single incidency relation Inc is usually denoted by MS_2. The Inc relation is replaced by the couple of incidency relations In and Out when the (multi) graph is directed. We extend MS_2 into PMS_2 by allowing (external) properties (unary functions) on vertices and edges. Thus, PMS_2 is suited for labelled properties graph model of graph databases. A matrix or arcs-coloured simple graph or labelled 2-structure can be studied, for example, with monadic second order logic with (internal) domain restricted to vertices and a single adjacency external function Adj with values over the ordinals, we will denote it by FMS_1, or FPMS_1 when we have additional properties on the vertices. We will define an intermediate logic $\mathrm{PMS}_{1.1}$ between PMS_1 and PMS_2; it uses an internal domain made of vertices and edges; it captures many queries on graph databases and has the nice property that any problem expressed in $\mathrm{PMS}_{1.1}$ over a binary structure S can be solved on a equivalent reversible matrix S' with internal domain equal to the vertices in the internal domain of S using an equivalent formula in FMS_1 or FPMS_1 when there was additional (external) properties on the vertices.

Note that PMS_2 is equivalent to FPMS_2, since external functional values over the incindencies functions In and Out can be replaced by two external edge properties $in\_functional\_value$ and $out\_functional\_value$.

Clearly, any problem expressed on a binary structure S using monadic second order logic with quantification of elements and subsets of its internal domain can be expressed using PMS_2 on the incidencies structure $I(S)$ obtained from S (if S already had vertices and edges of a graph as its internal domain, these vertices and edges become the new vertices, and the incidency relationships between them become the new edges). Thus we can always restrict ourselves to two relational adjacencies/incidencies and a set of functional properties.

Definition 1.1 (Adjacency selector)**.** *An* adjacency selector $AS(x, y)$ *is a* PMS_2 *for-*

mula with two free variables x and y that must be vertices, such that:

- *it starts with a single existential quantifier over an arc a ($\exists a$),*
- *then it fixes that x and y are the in-vertex and the out-vertex of a ($\mathrm{In}(a, x) \wedge \mathrm{Out}(a, y)$),*
- *then it gives constraints on the properties of a using a (possibly empty) subformula using only atomic formulas (like $P_i(a) = c$, $P_i(a) \neq c$, $P_i(a) < c$, or $R(P_i(a), c)$, where c is an external constant and R is a binary relation over the external domain only, for some $i \in \alpha$, etc.), $\neg, \wedge, \vee$.*

$AS(x, y) = \exists a, \mathrm{In}(a, x) \wedge \mathrm{Out}(a, y) \wedge (age\_at\_birth(a) = 31 \vee age\_at\_birth(a) > 33)$

is an example of an adjacency selector.

Definition 1.2 ($\mathrm{PMS}_{1.1}$)**.** *A PMS_2 formula F is a $\mathrm{PMS}_{1.1}$ formula, if:*

- *whenever there is an existential quantification over an arc in F, then the subformula rooted on this existential quantification is an adjacency selector,*
- *whenever there is an universal quantification over an arc in F, then the subformula rooted on this universal quantification is the negation of an adjacency selector.*

A $\mathrm{PMS}_{1.1}$ formula F is in existential form *if there is no universal quantification over an arc in it. (Using negation it is trivial to have $\mathrm{PMS}_{1.1}$ formula in existential form.)*

Theorem 1.3. *Testing if a $\mathrm{PMS}_{1.1}$ formula is true on a labelled properties graph G is equivalent to testing if a FPMS_1 formula is true on a reversible labelled 2-structure S with internal domain equal to the vertices of G and external domain equal to some ordinal. G may be infinite in which case the internal domain of S is also infinite, but it may still be the case that the external domain is finite. The logic is reduced to FMS_1 if there was no external property on vertices in G.*

Proof:

This proof is written with examples and lacks rigour. It is intentional: The result is easy, and this is the introduction.

Let F be a $\mathrm{PMS}_{1.1}$ formula. Without loss of generality, we assume that F is in existential form. Assume that you have enumerated all possible directed relations you have in some labelled properties graph; for example, you have vertices representing persons, cars, and addresses, and the following directed relations with properties:

- "is_mother_of" links a mother to its child with a property "age_at_birth" giving the age of the mother when she gave birth to this child,
- "is_father_of" links a father to its child with a property "age_at_birth" giving the age of the father when his wife gave birth to this child,
- "play_tennis_with" links two persons,

- "has_home_at" links a person to the address vertex corresponding to its home,
- "owns" links a person to its car with a property "buying_date" and a property "buying_location".

As a first step, you use Zermelo's axiom (in the infinite case) a first time to list all the combinations of a relation plus its properties that appear in the graph, like first combination is {("relation_name", "is_mother_of"), ("age_at_birth", 31)}, {("relation_name", "is_mother_of"), ("age_at_birth", 24)}, third combination is ("has_home_at", {}), etc. Each combination describes the corresponding set of arcs. A person familiar with graph databases would probably be more familiar with a notation close to Python like ("is_mother_of", {"age_at_birth": 24}), or {"relation_name":'is_mother_of", "age_at_birth": 24}, instead of {("relation_name", "is_mother_of"), ("age_at_birth", 24)}. We use a notation respecting the mathematical meaning of parentheses for tuples/sequences and brackets for sets, instead of introducing "dictionaries" or "associative arrays".

For the sake of efficiency, the first step of the transformation should depend on the formula. For example, if you have a formula "there exists a person p such that, (there exists a car c such that, p owns c) and (for all car c', p owns c' implies p bought c' before 2020/01/01)", then you will first forget all edges corresponding to the relations "is_mother_of", "is_father_of", "has_home_at". Moreover you should also forget all properties of "owns" except "buying_date" ("buying_location" has no usefulness for the problem at hand). Because these relations and properties have no impact on the adjacency selectors of the formula at hand. Thus you will enumerate only the combinations {("relation_name", "owns"), ("buying_date", 2011/01/01)}, {("relation_name", "owns"), ("buying_date": 2012/01/01)}, for example. You have a ternary relation LinkedBy(x,y,comb) where x and y are vertices (internal domain) and comb is the integer/ordinal (external domain) corresponding to a combination of a relation plus its properties. Assume that the combinations 1, 3, 4 are all the combinations of "owns" relation, and that the combinations 1, 3 are all the combinations of "owns" relation with buying date before 2020/01/01. So far, the formula becomes "there exists a person p such that, (there exists a car c such that, LinkedBy(p,c,1) or LinkedBy(p,c,3) or LinkedBy(p,c,4)) and (for all car c', LinkedBy(p,c',1) or LinkedBy(p,c',3) or LinkedBy(p,c',4) implies LinkedBy(p,c',1) or LinkedBy(p,c',3))".

Note that in this step and the two other steps coming next, we may use a disjonction of an infinite number of atomic subformulas, when the binary structure is infinite. We consider that the OR nodes/gates have unbounded fan-in. Thus the depth of the formula cannot increase since any adjacency selector has depth at least 3, and we replaced the adjacency selectors with subformulas of depth 2.

As a second step, you use Zermelo's axiom (in the infinite case) a second time to list all the set of combinations of a relation plus its properties that appear between two vertices in the graph with the same in-vertex and out-vertex, like first set of combinations is {{("relation_name", "play_tennis_with")}}, second set of combinations is {{("relation_name", "play_tennis_with")}, {("relation_name", "is_mother_of"), ("age_at_birth", 31)}}, third set of combinations is {{("relation_name", "is_mother_of"),

("age_at_birth", 24)}}, fourth set of combinations is {} (no relations), etc. Any couple (ordered pair) of vertices has a unique corresponding set of combinations, and each set of combinations has at least one corresponding couple of vertices.

We kept the description of the combinations for clarity but at this step, combinations are already integers/ordinal numbers, like first set of combinations is {1}, second set of combinations is {1,5}, third set of combinations is {3}, fourth set of combinations is {} (no relations), etc. Thus, it is easy to find duplicate sets using lexicographic order and ordering combinations inside each set. You have a ternary relation AdjacencySemiType(x,y,s) where x and y are vertices (internal domain) and s is the integer/ordinal (external domain) corresponding to a set of combinations of a relation plus its properties.

Back at our example, assume that the set of combinations 2, 5, 11 are all the set of combinations containing "owns" relation, and that the set of combinations 2, 11 are all the set of combinations containing "owns" relation with buying date before 2020/01/01. So far, the formula becomes "there exists a person p such that, (there exists a car c such that, AdjacencySemiType(p,c,2) or AdjacencySemiType(p,c,5) or AdjacencySemiType(p,c,11)) and (for all car c', AdjacencySemiType(p,c',2) or AdjacencySemiType(p,c',5) or AdjacencySemiType(p,c',11) implies AdjacencySemiType(p,c',2) or AdjacencySemiType(p,c',11))".

After using two times Zermelo's axiom, we already have a matrix/edge-coloured simple graph/labelled 2-structure and a wealth of results starts to apply. Our example is not realistic because there should be probably much more AdjacencySemiTypes than combinations.

Now, we need to apply Zermelo's axiom a third time in order to use first difference principle. Indeed, the adjacency type of the couple (x,y) is the couple of the two adjacency semi-types from x to y and from y to x. Again, we may fear a quadratic increase in the number of adjacency types compared to the number of adjacency semi-types, when enumerating them. Our formula may become "there exists a person p such that, (there exists a car c such that, AdjacencyType(p,c,3) or AdjacencyType(p,c,14) or AdjacencyType(p,c,111)) and (for all car c', AdjacencyType(p,c',3) or AdjacencyType(p,c',14) or AdjacencyType(p,c',111) implies AdjacencyType(p,c',14) or AdjacencyType(p,c',111))". We now have what is called a reversible labelled 2-structure. It is reversible because now the function that gives the adjacency type from x to y has the property that whenever two couples of vertices (x,y) and (x',y') have the same adjacency type, then the two couples of vertices (y,x) and (y',x') have the same adjacency type.

Clearly, the formula has only been modified by substituting a disjunction of a (probably high) number of atomic subformulas (AdjacencyType(x,y,t)) to a single adjacency selector (e.g. p owns c') in the original formula. Its main structure remained the same.

We note that LinkedBy must be a ternary relation, but AdjacencyType and AdjacencySemiTypes are equivalently ternary relations or binary external functions. All three are directed.

Last but no least, it is easy to see how to deal with multiple edges/arcs. On this respect, if we have k arcs from x to y like "has_won_a_tennis_match_against", we keep only one arc "has_won_a_tennis_match_against" and we had a special property "multiplicity" of value k on this arc. If we had arcs with properties values like {("relation_name", "has_won_a_tennis_match_against"), ("date", 2020/01/01)}, {("relation_name", "has_won_a_tennis_match_against"), ("date", 2020/01/01)}, and {("relation_name", "has_won_a_tennis_match_against"), ("date", 2020/01/03)}, we cannot merge the last with the two previous; we have {("relation_name", "has_won_a_tennis_match_against"), ("date", 2020/01/01), ("multiplicity", 2)} and {("relation_name", "has_won_a_tennis_match_against"), ("date", 2020/01/03), ("multiplicity", 1)}. It does not change the three steps since we start from the combinations of a relation plus its properties, instead of bare relations. ■

If you think of labelled 2-structures as matrices, reversible labelled 2-structures are somehow "weakly symmetric" matrices (but not necessarily skew symmetric since we may have many symmetric adjacency types). We bought the book by Ehrenfeucht et al. (1999) wanting to know if it was already known that "weakly symmetric" matrices/2-structures are sufficient to study. In this book, they justify reversibility relatively to clans but do not say a word of logical formulas that are marginally modified. The result for clans unchanged under reversibility can be seen as a corollary of our theorem, since clans can be defined using a formula using quantification over vertices only. Thus, it is clear that for many formulas and binary structures, we can use the first difference principle and reversible labelled 2-structures. We wrote this theorem thinking it was the most missing point in the introduction of Ehrenfeucht et al. (1999), in order to justify the study subject.

We note that the previous proof can be done considering a set of labelled properties graph instead of a single graph. From a mathematical point of view, we will still use Zermelo's axiom three times, each step with probably much more combinations, adjacency semi-types, or adjacency types since we must gather all such possibilities on all graphs in the considered set of graphs. In that case, we may have an infinite graph yielding a finite formula and an infinite family of graphs yielding an infinite formula, for example.

It is not hard to find formulas with quantification over arcs that are not adjacency selectors for which the adjacency type trick yields the same theorem. (Consider a formula that decides if there are at least two distinct arcs from the vertex x to the vertex y. Clearly the adjacency type between x and y determines if it is true or false.) Thus, with some work $\mathrm{PMS}_{1.1}$ logic may be replaced by $\mathrm{PMS}_{1.2}$, $\mathrm{PMS}_{1.3}$, etc. logics to be defined in the future (each step closer to PMS_2 logic).

We note also that the proof of our theorem uses only the restriction of quantification over the edges and the external domain (adjacency selectors for edges, no quantification for the external domain). Thus, it is also true for first order quantification over the vertices or second order quantification over the vertices, etc. Quantification over the vertices is left unmodified between the original formula and the final formula.

Whenever our theorem yields some finite labelled 2-structure(s) with a finitely bounded number of edge colours, we can further use Theorem 3.9 in Courcelle (2006), to obtain a simpler structure like an undirected graph.

Section 2 contains the proof of equivalence between modular-width and questionable-width. In section 3, we show how clique-width belongs to the scope of first difference principle. Section 4 concludes with remarks on rank-width of (reversible) (labelled) 2-structures.

2 Modular-width and questionable-width

We want to apologize to our readers of our previous articles on first difference principle. Probably, many of them already knew modular and clan decompositions, and they immediately have seen that questionable-width was equivalent to modular/clan-width. We did not work on modular decomposition previously and what we remembered of talks about it was erroneously that you would decompose a graph/2-structure when it was either disconnected (parallel node) or its complement was disconnected (series node), stopping when both are connected. Our further readings corrected this mistake and we have seen that the modular/clan decomposition goes on under primitive nodes. At this point, it was immediate to see that questionable-width was equivalent. See Ehrenfeucht et al. (1999), and Habib and Paul (2010) for a survey on algorithmic modular/clan-decomposition.

Consider a binary signature $\mathcal{S}$ of unary and binary relations and functions. Given a set S, an $(\mathcal{S}, S, k, l)$-*mapping-run* is an (ordinal-indexed) sequence $(S_i)_{i\in l}$ of length l of $\mathcal{S}$-structures of cardinality at most k, k being the lowest such cardinal, together with a sequence of mappings f_i from S to the domains of S_i structures. Each S_i is an $\mathcal{S}$-*structure-item* (of the mapping-run/the $(S_i)_{i\in l}$ sequence). Each element/vertex x of S is thus associated to a sequence $(x_i = f_i(x))_{i\in l}$; this sequence is an *"element"* of the mapping-run/the $(S_i)_{i\in l}$ sequence; it corresponds to an element/vertex of S; each $x_i = f_i(x)$ is an *element/vertex-item*.

Definition 2.1 (Question)**.** *Given two elements X, Y of a mapping-run, we say that (k, x_k, y_k, S_k) is the* question *of X, Y, if k is the smallest ordinal such that $x_k \neq y_k$.*

Definition 2.2 (Questionable representation)**.** *A* questionable representation *of a reversible labelled 2-structure S is a $(\mathcal{A}, S, k, l)$-mapping-run, for some k and l, where $\mathcal{A}$ is the binary signature made of a single function that maps each couple of vertices to its adjacency type, such that first difference principle applied to this mapping-run yields the adjacencies of S. k is its width and l is its length. The* questionable-width *of S is the minimum of the widths of its questionable representations.*

The modular/clan-width is the maximum of the cardinals of the domains of quotient graphs/2-structures that appear in primitive nodes of the modular/clan decomposition of a given graph/2-structure. We fix the following convention that the modular/clan-width of a graph/2-structure without primitive nodes in his modular/clan decomposition is exactly two.

Under this convention, we have the following result.

Theorem 2.3. *The questionable-width of a finite reversible labelled 2-structure is equal to its clan-width.*

Proposition 2.4. *The questionable-width of a finite reversible labelled 2-structure S is at most its clan-width.*

Proof:

We process the nodes of the clan-decomposition from the root to its leaves, the order of the nodes is unimportant as long as the set of treated nodes is connected at any step. At any step, we lengthen a sequence of hopefully small 2-structures(-items) S_i, such that any vertex of the 2-structure S is associated to exactly one vertex(-item) in each S_i. Our induction hypothesis is that if a node has been treated, then all vertices of the 2-structure that belongs to distinct branches below this node have their adjacency types that are already set by first difference principle. Thus we obtain a questionable representation at the final step.

If the current node to be treated is an α-complete node where α is a symmetric adjacency type, or an α-linear node where α is an asymmetric adjacency type, we add at the end of our current partial questionable representation a sequence of 2-structures(-items) of size 2, their two vertices(-items) x_j^0 and x_j^1 having α adjacency type on the couple (x_j^0, x_j^1). If the number of nodes under the current node is k, we need $\lceil \lg(k) \rceil$ such small 2-structures(-items), and we associate to each vertex y of S under the current node to x_j^0 or x_j^1 using the binary notation of the ordinal finite number/integer corresponding to the subnode containing y. (The binary notations must be padded on the left with 0s to have the same length.) All other vertices of S are associated to $x_j^0, 1 \leq j \leq \lceil \lg(k) \rceil$.

If the current node to be treated is a primitive node, we add at the end of our current partial questionable representation a 2-structure(-item) isomorphic to the quotient 2-structure at the primitive node. We associate to each vertex y of S under the current node the vertex(-item) of its maximal prime clan in the quotient 2-structure. All other vertices of S are associated to the same arbitrary vertex(-item) in the quotient 2-structure.

■

Proposition 2.5. *The questionable-width of a finite reversible labelled 2-structure is at least its clan-width.*

Proof:

Assume for a contradiction that there is a primitive node separating vertices x and y, and that the 2-structure(-item) in the questionable representation, where the first difference between (/the question of) x and y occurs, has size less than the cardinal of the quotient 2-structure at the primitive node. We take such a pair of vertices x and y with minimum question index. Clearly, without loss of generality, there must be some vertex z that is in a subbranch below the primitive node; this subbranch is distinct of the subbranches of x and y; and nevertheless x and z have the same image in the question 2-structure. But then since the preimage of the image of x (and z) restricted to the vertices below the primitive node is a clan of the substructure induced by these vertices, it contradicts the fact that the quotient 2-structure was primitive.

■

The previous results solve the problem of optimizing the width of a questionable representation. But there remains some open-problems :

Open problem 2.6. *Optimize the length of a questionable representation under optimality of the width or a relaxed assumption on the width. (Without width constraint this is obviously one.) Optimize the area (sum of the cardinals of the domains of the labelled 2-structures(-items) in the questionable representation) or the approximated area (length times width) of a questionable representation, with or without width constraint.*

Open problem 2.7. *Some graphs admits questionable representations with logarithmic length, area, and approximated area, under optimal width constraint. Find equivalent characterizations of such graphs. (All labelled 2-structures admits questionable representations with linear length, area, and approximated area, under optimal width constraint.)*

Acccording to Ehrenfeucht et al. (1999), and Habib and Paul (2010), the origin of modular/clan decomposition can be traced back to Gallai (1967). Putting it in the scope of first difference principle, we go a few milleniums before but graphs and 2-structures were not studied yet. It would be wrong to say that modular/clan decomposition can be traced back to a few milleniums before, but we can say that a simple principle was already known that could lead to it with some work.

3 Clique-width and first difference principle

A few years ago, we did not know much about 2-structures, and we generalized clique-width with weighted graphs, the weights being over some field (see Flarup and Lyaudet (2008)). When we read the book by Courcelle and Engelfriet (2012), it was a surprise to see their generalization to labelled edge graphs using separate edge families. However the two generalizations are equivalent, may it be using ordinal weights for infinite graphs, or rational numbers weights for finite binary structures. This is a consequence of the theorem in introduction and of the following fact: with clique-width, you can add all the edges between two vertices as soon as the two paths from the leaves corresponding to two vertices to the root of the clique-decomposition joins the same node in the decomposition; hence vertices coming from the same branch with distinct adjacency types to a vertex of the other branch must have distinct clique-labels.

In order to find a sequential/first difference principle equivalent to clique-width and clique decomposition, we need to introduce nyf-extended labelled 2-structures. nyf is a constant special adjacency type meaning "not yet fixed". Only reversible labelled 2-structures appearing in decompositions may use this special adjacency type. Its meaning is obvious, when the first difference between two vertices, that occurs in a nyf-extended labelled 2-structures, has the nyf-adjacency type between the images of the two vertices then their adjacency type must be given by another difference later. Thus, nyf-extended first difference principle is that the adjacency type is given by first difference where adjacency type is not nyf.

Definition 3.1 (Question (nyf-extended))**.** *Given two elements X, Y of a mapping-run, we say that (k, x_k, y_k, S_k) is the* question *of X, Y, if k is the smallest ordinal such that $x_k \neq y_k$ and the adjacency type between x_k and y_k is not* nyf.

There is a dual constant to nyf-extended labelled 2-structures. This is alf-extended labelled 2-structures. alf is a constant special adjacency type meaning "already fixed". Only reversible labelled 2-structures to be decomposed may use this special adjacency type. This is what is used to further decompose some reversible labelled 2-structure in tree-questionable-width. (We introduced tree-questionable-width in Lyaudet (2019).) It is somewhat linked to bi-modular decomposition in this setting, see Fouquet et al. (2004). We shall use it in a future article on classes of bounded balanced tree-questionable-width, but not in this article.

A nyf-connected-component, resp. nyf-clique, in a nyf-extended labelled 2-structures is simply a connected component, resp. a clique, in the underlying undirected simple graph where nyf-adjacency corresponds to edges and everything else correspond to a non-edge.

Definition 3.2 (Clique-questionable representation)**.** *A* clique-questionable representation *of a reversible labelled 2-structure S is a $\mathrm{nyf}-(\mathcal{A}, S, k, l)$-mapping-run, for some k and l, where $\mathcal{A}$ is the binary signature made of a single function that maps each couple of vertices to its adjacency type, such that:*

(i) *nyf-extended first difference principle applied to this mapping-run yields the adjacencies of S;*

(ii) *the nyf-extended reversible labelled 2-structures(-items) must have the property that there are exactly two nyf-connected-components, and that these nyf-connected-components are nyf-cliques;*

(iii) *two vertices of S that have a nyf-difference must have only nyf-differences until they have a question/non-nyf-difference; they cannot be mapped to the same vertex(-item) again until some question/non-nyf-difference occurs.*

k is its width and l is its length. The clique-questionable-width *of S is the minimum of the widths of its clique-questionable representations.*

Theorem 3.3. *The clique-questionable-width of a countable reversible labelled 2-structure is between its clique-width and two times its clique-width, under the constraint that clique-questionable representations must have length at most ω.*

Proposition 3.4. *The clique-questionable-width of a countable reversible labelled 2-structure S is at most two times its clique-width, under the constraint that clique-questionable representations must have length at most ω.*

Proof:

Without loss of generality, we may assume that all edges are added in the clique-decomposition as soon is possible, using the compact clique-algebra (Courcelle and Engelfriet (2012)). Moreover, at the cost of doubled clique-width, we assume that when two branches joins in the clique decomposition, their port labels are disjoint.

We process the (compact-)nodes of the clique-decomposition from the root to its leaves, the order of the nodes is unimportant as long as the set of treated nodes is connected at any step. At each step, we lengthen a sequence of hopefully small nyf-extended labelled 2-structures(-items) S_i, such that any vertex of the 2-structure S is associated to exactly one vertex(-item) in each S_i, respecting condition (iii) above. Moreover, each S_i is the union of two nyf-cliques. Our induction hypothesis is that if a node has been treated, then all vertices of the 2-structure that belongs to distinct branches below this node have their adjacency types that are already set by nyf-extended first difference principle (condition (i')), and conditions (ii) and (iii) are respected. Thus we obtain a clique-questionable representation at the final step, since all vertices have been pairwise separated in distinct branches at some point.

Since the compact clique-algebra yields a single type of nodes (no need to handle the leaves corresponding to isolated vertices), the induction is straight-forward. For each compact node, we add a nyf-extended labelled 2-structure(-item) S_i made of two nyf-cliques, the first nyf-clique having as much vertices as port labels in the left branch below the current node, the second nyf-clique having as much vertices as port labels in all the clique-decomposition (condition (ii) is satisfied). The mapping of vertices from S to S_i associates all vertices below the current node to the vertex(-item) corresponding to its port label (under the current node) in the nyf-clique corresponding to its branch; other vertices are associated to the vertex(-item) corresponding to its port label (under the last corresponding node treated) in the nyf-clique of S_i having all port labels. The adjacency types in S_i between the two nyf-cliques are the ones corresponding to the node of the clique-decomposition with the additional no-edge default adjacency type explicitly set, when the compact-node adds nothing between two port labels sets of vertices. Clearly condition (i') is satisfied. To see that condition (iii) is satisfied, we observe that:

- for all vertices below the current node, condition (iii) is true between two vertices in the same branch below the current node, because it was true previously, and no renaming operation may give distinct port labels if they had the same port label below the node; two vertices in distinct branches have now distinct port labels and a fixed adjacency type;
- for all other vertices, condition (iii) was satisfied previously, and nothing changed with the previous step with respect to being mapped to the same vertex or not;
- for a vertex below the current node and a vertex not below, condition (iii) was satisfied previously, and they have already a question/non-nyf difference, hence it will never be unsatisfied in the future.

Q.E.D. ∎

Proposition 3.5. *The clique-questionable-width of a countable reversible labelled 2-structure is at least its clique-width, under the constraint that clique-questionable representations must have length at most ω.*

Proof:

Given a clique-questionable representation, we construct a (compact)-clique decomposition of same width in a top-down manner. First, using a number of port labels equal to the width of the clique-questionable representation, we assign a distinct port label to each vertex(-item) of S_i, for all i; this choice is completely arbitrary as long as two distinct vertices of the same S_i have distinct port labels (and we do not use more port labels than necessary for that). At each step, we maintain a set of subsets of S, $\{X_j\}$, that have already been splitted, all adjacency types between distinct $X_j, X_{j'}$ being already fixed both by the beginning of clique-questionable representation and the in-construction (compact-)clique-decomposition. Each X_j corresponds to a leaf of the in-construction (compact-)clique-decomposition. Before first step, $\{X_j\} = \{S\}$. For each (step) S_i item in the clique-questionable representation, consider the substructure (substep) $S_{i,j}$ of S_i restricted to the image of X_j. If $S_{i,j}$ is included in one of the two nyf-cliques, then this substep is finished. Otherwise $S_{i,j}$ is made of two parts $S_{i,j,0}$ and $S_{i,j,1}$ corresponding to the two nyf-cliques. We replace the leaf corresponding to X_j with a node that renames the port labels of the vertices in $S_{i,j}$ into the port labels of X_j vertices at the substep X_j was created (we can do so with respect to distinct vertices in the same $S_{i,j,b}$ thanks to condition (iii)); under this node we add adding adjacency type operations/nodes to reflect $S_{i,j}$ (this is easy since all port labels are distinct); last we had a disjoint union node separating two leaves corresponding to $X_{j,0}, X_{j,1}$ the domains of $S_{i,j,0}, S_{i,j,1}$. (The nodes created at a substep can trivially be grouped into a compact-node.) It is trivial to see that the clique-decomposition obtained constructs the same labelled 2-structure as the clique-questionable representation, and does not use more port labels than its width. ∎

Proposition 3.4 can not be improved unless Proposition 2.105 (4) in Courcelle and Engelfriet (2012) can be improved. (It cannot be improved for trivial graphs such as the discrete graphs and the complete graphs. We do not know a non-trivial family of counter-examples.) First difference principle, as we use it, is totally "adjacency type impartial", it does not make any difference between some default adjacency type such as no-edge and other adjacency types. It has the advantage that all results apply under any injective mapping from a set of adjacency types to another. Hence, the two previous definitions also define the questionable-width and the clique-questionable-width of a 2-structure instead of a labelled 2-structure. It would be possible to shrink (clique-)questionable representations in some cases, with the convention that, when no two differences occurs, a default adjacency type is given. Non-edges gives an example of such a symmetric default adjacency type. Lexicographic order gives an example of such an asymmetric default adjacency type, using the length to determine the orientation. We do not study further this topic yet because we think that as the saying goes "Explicit is better than implicit", and "Premature optimization is the root of all evil" (Knuth). Further research may prove us wrong. Courcelle generalized clique-decompositions to countable graphs/labelled 2-structures. nyf-extended first difference principle and clique-questionable representation give us an alternative for reversible (labelled) 2-structures of any cardinality. Equivalently, we can substitute to the nyf-clique requirement the fact that nyf-adjacency defines an equivalence relation on vertices(-items) with exactly two equivalence classes in each 2-structure-item.

In light of the introductory theorem and the "good behaviour" of clique-width and monadic second order logic with vertex quantification, it was an easy guess that first difference principle may have a link with clique-width.

Déjà vu:

Open problem 3.6. *Optimize the length of a clique-questionable representation under optimality of the width or a relaxed assumption on the width. Optimize the area (sum of the cardinals of the domains of the labelled 2-structures in the clique-questionable representation) or the approximated area (length times width) of a clique-questionable representation, with or without width constraint.*

Open problem 3.7. *Some graphs admits clique-questionable representations with logarithmic length, area, and approximated area, under optimal width constraint. Find equivalent characterizations of such graphs.*

4 Rank-width of (labelled) 2-structures

In 2008, we were working to extend the results of Barvinok (1996) and Flarup and Lyaudet (2008). Barvinok showed that if the underlying matrix has bounded rank, both the permanent and the hamiltonian polynomials can be evaluated in polynomial time; we showed a similar result for bounded weighted clique-width matrices. We hoped to show that the permanent of a bounded rank-width matrix can be computed in polynomial time. Our opinion at that time was that rank-width was a kind of algebraic trick to "compress" clique-width. We did not know whether rank-width and clique-width are still equivalent over any arbitrary field. We talked about this subject with Frédéric Mazoit, who talked about it to Michaël Rao, who gave us the following example in 2010:

"Consider a finite set of vertices, each associated to a distinct strictly positive integer, giving the matrix over $\mathbb{Q}$ where the edge weight between vertices i and j is $i \times j$. This matrix has rank 1, hence has rank-width 1. But it should have clique-width around $n/2$, where n is the number of vertices."

We can modify this example to guaranty that all edges are distinct: Consider a finite set of vertices, each associated to a distinct prime number p_i (We have a symmetric 2-structure). Or even that all arcs are distinct: cell i, j has weight $p_i^2 \times p_j$ (We have an asymmetric reversible 2-structure).

After that, our opinion on rank-width was that we should not be talking about equivalence between rank-width and other widths with too much confidence because rank-width lives in his own realm. It is able to capture "too predictable chaos" when all adjacencies are distinct.

At the light of first difference principle and "adjacency type impartiality", we think we should not focus on the algebraic meaning of rank-width. Instead we should feel free to compute the rank-width of any (labelled) 2-structure with an arbitrary injective mapping from its adjacency types to values in some arbitrary field. It yields two new widths of (labelled) 2-structures: maximum rank-width under injective mapping, and minimum rank-width under injective mapping. We have the corresponding open-problems:

Open problem 4.1. *For (finite) (labelled) 2-structures, what are the fields on which the maximum rank(-width) under injective mapping is obtained. Is it true for finite (labelled) 2-structures that an injective mapping maximizing rank(-width) always exists over $\mathbb{Q}$? Find algorithms in the finite case.*

Open problem 4.2. *For (finite) (labelled) 2-structures, what are the fields on which the minimum rank(-width) under injective mapping is obtained. Is it true for finite (labelled) 2-structures that an injective mapping minimizing rank(-width) always exists over the smallest prime field of cardinality at least the number of adjacency types (plus 1 when there are only 2 adjacency types ?) of the 2-structure? Find algorithms in the finite case.*

Open problem 4.3. *Is it true that clique-width of labelled 2-structures is closely equivalent to their maximum rank-width under injective mapping?*

Open problem 4.4. *Rank-width is able to capture "too predictable chaos", but minimum rank-width under injective mapping does capture "too predictable chaos". What kind of logic is suited for that? For example, it is clear that first order model checking is trivial on a labelled 2-structure where all adjacency types are distinct, at least if the formula has no disjunction between possible adjacency types for some couple of quantified vertices in it.*

We gave a try at these open problems for a few weeks, with meager success.

Lemma 4.5. *Let e and e' be the values in some field $\mathbb{F}$ corresponding to an edge and a non-edge. We can assume without loss of generality that one of these two values is $1_{\mathbb{F}}$ without modifying the rank(-width).*

The following matrix has rank 2 over $\mathbb{F}_2$ but rank 3 over $\mathbb{Q}$:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

It is the smallest such matrix (unique matrix of dimension 3) and the adjacency matrix of a triangle. It yields the following matrix that has rank 3 in all fields except $\mathbb{F}_2$:

$$\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix}$$

But the matrix $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ has rank 2 and the matrix $\left(\begin{smallmatrix} -1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$ has rank 1, over any field except $\mathbb{F}_2$.

The non-trivial linear combinations of 2 rows of -1, 1 matrices cannot be a -1, 1 row.

The following matrix has rank 3 over $\mathbb{F}_3$ but rank 4 over $\mathbb{F}_2$ (easy to generalize) :

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

The following matrix has rank 3 over $\mathbb{F}_2$ and $\mathbb{Q}$:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

but the 3 linearly independent rows are not the same. It proves that a simple proof like "any linear combination of 0,1 rows over $\mathbb{Q}$ that yields a 0,1 row has an equivalent linear combination over $\mathbb{F}_2$" does not exist. Thus it is not trivial, if true, that the rank over $\mathbb{Q}$ of 0, 1 matrices is the maximum rank under injective mapping of such matrices.

Trying to find a 0, 1 matrix of higher rank over $\mathbb{F}_2$ than over $\mathbb{Q}$, we did an algorithm that checked that no such matrix exists for dimension at most 6. Dimension 7 was out of reach for our laptop. The source code in PHP is in the appendix.

Acknowledgements

We thank God: Father, Son, and Holy Spirit. We thank Maria. They help us through our difficulties in life.

We thank Bruno Courcelle for some useful remarks.

References

A. Barvinok. Two algorithmic results for the traveling salesman problem. *Mathematics of Operations Research*, 21:65–84, 1996.

B. Bui-Xuan, M. Habib, V. Limouzy, and F. de Montgolfier. Homogeneity vs. adjacency: Generalizing some graph decomposition algorithms. In F. V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG 2006, Bergen, Norway, June 22-24, 2006, Revised Papers*, volume 4271 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2006. doi: 10.1007/11917496_25. URL https://doi.org/10.1007/11917496_25.

B.-M. Bui-Xuan. *Tree-Representation of Set Families in Graph Decompositions and Efficient Algorithms*. PhD thesis, Université Montpellier II, september 2008.

G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre. *Math. Ann.*, 46: 481–512, 1895.

B. Courcelle. The monadic second-order logic of graphs XV: on a conjecture by D. Seese. *J. Appl. Log.*, 4(1):79–114, 2006. doi: 10.1016/j.jal.2005.08.004. URL https://doi.org/10.1016/j.jal.2005.08.004.

B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. ISBN 978-0-521-89833-1. URL http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

A. Ehrenfeucht, T. Harju, and G. Rozenberg. *The Theory of 2-Structures - A Framework for Decomposition and Transformation of Graphs*. World Scientific Publishing Co. Pte. Ltd., 1999. ISBN 981-02-4042-2.

U. Flarup and L. Lyaudet. On the expressive power of permanents and perfect matchings of matrices of bounded pathwidth/cliquewidth (extended abstract). In *CSR 2008, Computer Science in Russia*, volume 5010 of *Lecture Notes in Computer Science*, pages 180–193. Springer Verlag, 2008.

J. Fouquet, M. Habib, F. de Montgolfier, and J. Vanherpe. Bimodular decomposition of bipartite graphs. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science, 30th International Workshop,WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, volume 3353 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2004. doi: 10.1007/978-3-540-30559-0_10. URL https://doi.org/10.1007/978-3-540-30559-0_10.

T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18:25–66, 1967.

M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.*, 4(1):41–59, 2010. doi: 10.1016/j.cosrev.2010.01.001. URL https://doi.org/10.1016/j.cosrev.2010.01.001.

F. Hausdorff. Untersuchungen über Ordnungstypen V. *Ber. über die Verhandlungen der Königl. Sächs. Ges. der Wiss. zu Leipzig. Math.-phys. Klasse*, 59:105–159, 1907.

L. Lyaudet. A class of orders with linear? time sorting algorithm. *CoRR*, abs/1809.00954, 2018. URL http://arxiv.org/abs/1809.00954.

L. Lyaudet. On finite width questionable representations of orders. *CoRR*, abs/1903.02028, 2019. URL http://arxiv.org/abs/1903.02028.

W. Sierpiński. Généralisation d'un théorème de Cantor concernant les ensembles ordonnés dénombrables. *Fundamenta Mathematicae*, 18:280–284, 1932.

Appendix

We give here the PHP source code of our experiments on rank of 0,1 matrices. It serves two purposes :

- We may have a bug in our code that would invalidate what we said previously; any reader can check and reproduce our experiment and correct us in case we were wrong.
- We found no relevant article on generating all 0,1 matrices of some dimension, up to transposition, row permutation, and/or column permutation; and we think that the algorithm we designed for this task using lexicographic order of rows is quite good. However, our algorithm for rank computation is very crude.

```
<?php

function printMatrix($matrix){
  static $iNumberOfMatricesPrinted = 0;
  foreach($matrix as $row){
    echo '( '.join(' ', $row).' )'."\n";
  }
  //echo "\n";
  ++$iNumberOfMatricesPrinted;
  echo $iNumberOfMatricesPrinted." matrices printed\n";
  //echo "\n";
}

function generateMatrices($iDimension, $funcCallback){
  $matrixGerm = array();
  for($i = 0; $i < $iDimension; ++$i){
    $row = array();
    for($j = 0; $j < $iDimension; ++$j){
      $row []= 0;
    }
    $matrixGerm []= $row;
  }
  //printMatrix($matrixGerm);
  for($i = 1; $i <= $iDimension; ++$i){
    generateMatricesWith($iDimension, $funcCallback, $i, $matrixGerm);
  }
}


function generateMatricesWith(
  $iDimension,
  $funcCallback,
  $iAtMostOnesOnEachColumnRow,
  $matrixGerm
){

  $tabINumberOfOnesOnColumn = array();
  for($i = 0; $i < $iAtMostOnesOnEachColumnRow; ++$i){//setting first row
    $matrixGerm[0][$i] = 1;
    $tabINumberOfOnesOnColumn[$i] = 1;
  }
  for(;$i < $iDimension; ++$i){
    $tabINumberOfOnesOnColumn[$i] = 0;
  }
  for($i = 1; $i <=  $iAtMostOnesOnEachColumnRow; ++$i){
    generateMatricesWith2(
```

```
      $iDimension,
      $funcCallback,
      $iAtMostOnesOnEachColumnRow,
      $matrixGerm,
      $i,
      $tabINumberOfOnesOnColumn
    );
  }
}


function generateMatricesWith2(
  $iDimension,
  $funcCallback,
  $iAtMostOnesOnEachColumnRow,
  $matrixGerm,
  $iNumberOfOneOnFirstColumn,
  $tabINumberOfOnesOnColumn
){
  for($i = 1; $i < $iNumberOfOneOnFirstColumn; ++$i){//setting first column
    $matrixGerm [$i][0] = 1;
  }
  $tabINumberOfOnesOnColumn[0] = $iNumberOfOneOnFirstColumn;
  //printMatrix($matrixGerm); return;
  $iCurrentNumberOfOnesOnThisRow = 1;
  $isCurrentRowEqualToPrevious = true;
  if($iNumberOfOneOnFirstColumn === 1){
    $iCurrentNumberOfOnesOnThisRow = 0;
    $isCurrentRowEqualToPrevious = false;
  }
  generateMatricesWith3(
    $iDimension,
    $funcCallback,
    $iAtMostOnesOnEachColumnRow,
    $matrixGerm,
    $tabINumberOfOnesOnColumn,
    1, 1,
    $isCurrentRowEqualToPrevious,
    $iCurrentNumberOfOnesOnThisRow
  );
}


function generateMatricesWith3(
  $iDimension,
  $funcCallback,
```

```
  $iAtMostOnesOnEachColumnRow,
  $matrixGerm,
  $tabINumberOfOnesOnColumn,
  $iCR,
  $iCC,
  $isCurrentRowEqualToPrevious,
  $iCurrentNumberOfOnesOnThisRow
){
  //We generate the new matrices with 0 or 1
  //Choice 0
  $isCurrentRowEqualToPreviousNew =
    ($isCurrentRowEqualToPrevious && $matrixGerm[$iCR-1][$iCC] === 0);
  generateMatricesWith3Bis(
      $iDimension,
      $funcCallback,
      $iAtMostOnesOnEachColumnRow,
      $matrixGerm,
      $tabINumberOfOnesOnColumn,
      $iCR,
      $iCC,
      $isCurrentRowEqualToPreviousNew,
      $iCurrentNumberOfOnesOnThisRow
  );

  //Choice 1
  if($iCurrentNumberOfOnesOnThisRow >= $iAtMostOnesOnEachColumnRow
    || $tabINumberOfOnesOnColumn[$iCC] >= $iAtMostOnesOnEachColumnRow
  ){
    return;
  }
  $isCurrentRowEqualToPreviousNew =
    ($isCurrentRowEqualToPrevious && $matrixGerm[$iCR-1][$iCC] === 1);
  if($isCurrentRowEqualToPrevious && !$isCurrentRowEqualToPreviousNew){
    //les lignes ne sont pas dans l'ordre lexicographique
    return;
  }
  $matrixGerm[$iCR][$iCC] = 1;
  $iCurrentNumberOfOnesOnThisRow += 1;
  $tabINumberOfOnesOnColumn[$iCC] += 1;
  generateMatricesWith3Bis(
      $iDimension,
      $funcCallback,
      $iAtMostOnesOnEachColumnRow,
      $matrixGerm,
      $tabINumberOfOnesOnColumn,
      $iCR,
```

```
      $iCC,
      $isCurrentRowEqualToPreviousNew,
      $iCurrentNumberOfOnesOnThisRow
  );
}

function generateMatricesWith3Bis(
  $iDimension,
  $funcCallback,
  $iAtMostOnesOnEachColumnRow,
  $matrixGerm,
  $tabINumberOfOnesOnColumn,
  $iCR,
  $iCC,
  $isCurrentRowEqualToPrevious,
  $iCurrentNumberOfOnesOnThisRow
){
  if($iCC === $iDimension -1){
    if($iCR === $iDimension -1){
      $funcCallback($matrixGerm, $iDimension);
      return;
    }
    $isCurrentRowEqualToPrevious =
        ($matrixGerm[$iCR][0] === $matrixGerm[$iCR + 1][0]);
    $iCurrentNumberOfOnesOnThisRow = $matrixGerm[$iCR + 1][0];
    generateMatricesWith3(
      $iDimension,
      $funcCallback,
      $iAtMostOnesOnEachColumnRow,
      $matrixGerm,
      $tabINumberOfOnesOnColumn,
      $iCR + 1,
      1,
      $isCurrentRowEqualToPrevious,
      $iCurrentNumberOfOnesOnThisRow
    );
    return;
  }
  generateMatricesWith3(
    $iDimension,
    $funcCallback,
    $iAtMostOnesOnEachColumnRow,
    $matrixGerm,
    $tabINumberOfOnesOnColumn,
    $iCR,
    $iCC + 1,
```

```
    $isCurrentRowEqualToPrevious,
    $iCurrentNumberOfOnesOnThisRow
  );
}

function computeRankOver(
  $matrix,
  $iDimension,
  $iModulo = null,//works for modulo 2 only at the moment,
  $bPrint = false
){
  $tabZeroRows = array();
  $tabExcludedRows = array();
  for($i = 0; $i < $iDimension; ++$i){
    for($j = 0; $j < $iDimension; ++$j){
      if($matrix[$i][$j] != 0){
        $tabZeroRows[$i] = false;
        $tabExcludedRows[$i] = false;
        continue 2;
      }
      $tabZeroRows[$i] = true;
      $tabExcludedRows[$i] = true;
    }
  }
  //for each column
  for($j = 0; $j < $iDimension; ++$j){
    //find first non-excluded row with non-zero cell
    $rowChosed = null;
    for($i = 0; $i < $iDimension; ++$i){
      if($tabExcludedRows[$i]){
        continue;
      }
      if(round($matrix[$i][$j], 7) != 0){
        $rowChosed = $i;
        $tabExcludedRows[$i] = true;
        break;
      }
    }
    if($rowChosed === null){
      continue;
    }
    for($i = 0; $i < $iDimension; ++$i){
      if($tabExcludedRows[$i]){
        continue;
      }
      if(round($matrix[$i][$j], 7) == 0){
```

```
        continue;
      }
      $coef = -($matrix[$i][$j] / $matrix[$rowChosed][$j]);
      $isZeroRow = true;
      for($k = $j; $k < $iDimension; ++$k){
        $matrix[$i][$k] = $matrix[$i][$k] + $coef * $matrix[$rowChosed][$k];
        if(round($matrix[$i][$k], 7) != 0){
          $isZeroRow = false;
        }
        if($iModulo !== null){
          while($matrix[$i][$k] < 0){
            $matrix[$i][$k] += $iModulo;
          }
          $matrix[$i][$k] = (int)round($matrix[$i][$k]);
          $matrix[$i][$k] = $matrix[$i][$k] % $iModulo;
        }
      }
      if($isZeroRow){
        $tabZeroRows[$i] = true;
        $tabExcludedRows[$i] = true;
      }
    }
  }
  $iRank = $iDimension;
  for($i = 0; $i < $iDimension; ++$i){
    if($tabZeroRows[$i]){
      --$iRank;
    }
  }
  if($bPrint){
    printMatrix($matrix);
  }
  return $iRank;
}


function hasSmallerRankOverQThanZ2(
  $matrix,
  $iDimension
){
  $iRankOverZ2 = computeRankOver(
    $matrix,
    $iDimension,
    2
  );
  $iRankOverQ = computeRankOver(
```

```
    $matrix,
    $iDimension
  );

  if($iRankOverZ2 > $iRankOverQ){
    printMatrix($matrix);
    echo(
        "Cette matrice est de rang $iRankOverZ2 sur Z2"
       ." mais de rang $iRankOverQ sur Q\n\n"
    );
  }
}


//generateMatrices(3, 'printMatrix');
/*
generateMatrices(3, 'hasSmallerRankOverQThanZ2');
generateMatrices(4, 'hasSmallerRankOverQThanZ2');
generateMatrices(5, 'hasSmallerRankOverQThanZ2');
//*/
//generateMatrices(6, 'hasSmallerRankOverQThanZ2'); //10 minutes d'attente
//generateMatrices(7, 'hasSmallerRankOverQThanZ2');

function compareRanksOverQAndZ2(
  $matrix,
  $iDimension
){
  $iRankOverZ2 = computeRankOver(
    $matrix,
    $iDimension,
    2,
    true
  );
  $iRankOverQ = computeRankOver(
    $matrix,
    $iDimension,
    null,
    true
  );

  printMatrix($matrix);
  echo(
      "Cette matrice est de rang $iRankOverZ2 sur Z2"
     ." et de rang $iRankOverQ sur Q\n\n"
  );
}
```

```
$iDimension = 9;
$matrix = array(
  array(1, 0, 0, 0, 0, 1, 1, 1, 1),
  array(1, 1, 0, 0, 0, 0, 1, 1, 1),
  array(1, 1, 1, 0, 0, 0, 0, 1, 1),
  array(1, 1, 1, 1, 0, 0, 0, 0, 1),
  array(0, 1, 1, 1, 1, 0, 0, 0, 0),
  array(0, 0, 1, 1, 1, 1, 0, 0, 0),
  array(0, 0, 0, 1, 1, 1, 1, 0, 0),
  array(0, 0, 0, 0, 1, 1, 1, 1, 0),
  array(1, 1, 1, 1, 1, 1, 1, 1, 1),
);

compareRanksOverQAndZ2(
  $matrix,
  $iDimension
);


$iDimension = 9;
$matrix = array(
  array(1, 0, 0, 0, 0, 0, 0, 0, 0),
  array(1, 1, 0, 0, 0, 0, 0, 0, 0),
  array(1, 1, 1, 0, 0, 0, 0, 0, 0),
  array(1, 1, 1, 1, 0, 0, 0, 0, 0),
  array(0, 1, 1, 1, 1, 0, 0, 0, 0),
  array(0, 0, 1, 1, 1, 1, 0, 0, 0),
  array(0, 0, 0, 1, 1, 1, 1, 0, 0),
  array(0, 0, 0, 0, 1, 1, 1, 1, 0),
  array(1, 1, 1, 1, 1, 1, 1, 1, 1),
);

function generateInterestingMatrices(
  $iDimension,
  $matrix,
  $funcCallback
){
  $iMinRow = 0;
  $tabModificationsToDo = array();
  for($j = 5; $j < $iDimension; ++$j){
    $iNumberOfOnesToSet = $j - 4;
    $iMaxRow = $j - 1;
    $tabModificationsToDo []= array(
      'column' => $j,
      'minr' => $iMinRow,
```

```
      'maxr' => $iMaxRow,
      'nb1' => $iNumberOfOnesToSet,
    );
  }
  modifyMatrix(
    $iDimension,
    $matrix,
    $funcCallback,
    $tabModificationsToDo
  );
}

function modifyMatrix(
  $iDimension,
  $matrix,
  $funcCallback,
  $tabModificationsToDo
){
  if(empty($tabModificationsToDo)){
    $funcCallback($matrix, $iDimension);
    return;
  }
  $modification = array_pop($tabModificationsToDo);
  performModification(
    $iDimension,
    $matrix,
    $funcCallback,
    $tabModificationsToDo,
    $modification
  );
}

function performModification(
  $iDimension,
  $matrix,
  $funcCallback,
  $tabModificationsToDo,
  $modification
){
  if($modification['nb1'] === 0){
    modifyMatrix(
      $iDimension,
      $matrix,
      $funcCallback,
      $tabModificationsToDo
    );
```

```
    return;
  }
  $modification['nb1'] -= 1;
  for($i = $modification['minr']; $i <= $modification['maxr']; ++$i){
    $matrix[$i][$modification['column']] = 1;
    $modification['minr'] += 1;
    performModification(
      $iDimension,
      $matrix,
      $funcCallback,
      $tabModificationsToDo,
      $modification
    );
    $matrix[$i][$modification['column']] = 0;
  }
}

generateInterestingMatrices(
  $iDimension,
  $matrix,
  'hasSmallerRankOverQThanZ2'
);

?>
```