MDDs boost equation solving on discrete dynamical systems

Enrico Formenti, Jean-Charles Régin, Sara Riva

MDDs boost equation solving on discrete dynamical systems

Enrico Formenti[1][0000−0002−1007−7912], Jean-Charles Régin[1][0000−0001−6204−5894], and Sara Riva[1][0000−0003−2133−8089]

Université Côte d'Azur, CNRS, I3S, France
{enrico.formenti,jean-charles.regin,sara.riva}@univ-cotedazur.fr

Abstract. Discrete dynamical systems (DDS) are a model to represent complex phenomena appearing in many different domains. In the finite case, they can be identified with a particular class of graphs called dynamics graphs. In [9] polynomial equations over dynamics graphs have been introduced. A polynomial equation represents a hypothesis on the fine structure of the system. Finding the solutions of such equations validate or invalidate the hypothesis.
This paper proposes new algorithms that enumerate all the solutions of polynomial equations with constant right-hand term outperforming the current state-of-art methods [10]. The boost in performance of our algorithms comes essentially from a clever usage of Multi-valued decision diagrams.
These results are an important step forward in the analysis of complex dynamics graphs as those appearing, for instance, in biological regulatory networks or in systems biology.

Keywords: Multi-valued decision diagrams · Discrete Dynamical Systems · Graphs semiring .

1 Introduction

Multi-valued Decision Diagrams (MDD) are a generalization of Binary Decision Diagrams (BDD) [1,5] used to obtain efficient representations of functions (with finite domains) or (finite) sets of tuples. An MDD is a Directed Acyclic Graph (DAG) created from a finite set of variables with specific (finite) domains. Associating each variable with a level of the structure, the MDD represents a set of feasible assignments as a path from the *root* to the *final* node. A crucial aspect of MDDs is the exponential compression power of the reduction operation and the fact that many classical operations (intersection, union, *etc.*) can be performed without decompression. In the last years, MDDs have been applied in many disparate research domains proving the potential of this structure. MDDs are used, for instance, to improve random forest algorithms replacing the classic binary decision trees [12], to represent and analyze automotive product data of valid/invalid product configurations [6], and to perform trust analysis in social networks [16]. There are also applications related to mathematical models like

Multi-State Systems (MSS) [15]. In this case, MDDs represent the MSS structure in terms of Multi-Valued Logic to compute some measures. MDDs find applications also in the analysis of the discrete dynamical systems. Indeed, in [13], MDD represents logic rules to analyze some properties and dynamics aspects in the case of regulatory networks (for example, to perform a stable states identification).

In this work, we propose to apply MDDs to equations over Discrete Dynamical Systems (DDS). DDS are a model to represent complex phenomena which evolve in (discrete) time coming from different domains such as genetic regulatory networks, boolean automata networks, population dynamics, and many others. DDS consist in a finite set of states and a next-state function. In the finite case, DDS correspond to a particular class of graphs, called dynamic graphs (*i.e.* graphs with outgoing degree one). Therefore, DDS are simple structures that can be applied to any phenomena that evolve according to a function. Often, when representing phenomena with DDS, one needs to validate "macro" dynamics coming from experimental results, or to identify the set of "micro" dynamics which generate a given "macro" behavior observed.

In [9], operations of sum and multiplication have been introduced to explain how DDS can be combined to generate new dynamical behaviors. Equipping the set of DDS with these operations provides a commutative semiring, and this naturally leads to write polynomial equations over DDS. This is interesting because these equations (with a constant right-hand term) can model hypotheses over the dynamics, and if one can prove properties for DDS then they can be automatically lifted to the application models. The idea is to solve these equations to validate the corresponding hypotheses. This paper introduces a pipeline to solve equations over the cycles of dynamics graphs (*i.e.* equations/hypotheses over the long term behavior of a phenomenon), formally introduced in [10]. The first part of this paper proposes a new algorithm which outperforms the state-of-art technique (Colored-Tree method) using MDDs to enumerate the solutions set of simple equations. The second part introduces a pipeline to solve general equations using MDDs to solve basic cases and to limit the exploration of the solutions space. The overall purpose is to introduce the first complete pipeline, based on MDD, to solve the equations introduced in [9,10] to study the hypothesis over the long term behavior of a phenomenon modeled by DDS.

2 Preliminaries

2.1 Multi-valued decision diagrams (MDDs)

Multi-valued decision diagrams are the extension of Binary decision diagrams (BDD) in which the diagram consists of a rooted acyclic graph able to represent a multi-valued function $\mathrm{f} : \{0...d-1\}^r \to \{true, false\}$. Considering a generic MDD, each level represents a variable and a final layer contains the true terminal node (tt). Therefore, the data-structure contains $r+1$ layers and a path from the *root* to the tt node represents a valid set of assignments. Each node is characterised by the variable represented in the layer and at most d outgoing

edges. In general, the edges are directed from an upper layer to a lower one. An edge corresponds to an assignment of the variable to a certain value specified over the edge. This data-structure presents only one root and one leaf. The false node, and the corresponding paths to reach this node, are omitted.

According to [2,8] , an MDD is **deterministic** if all the nodes have pairwise distinct labels on outgoing edges. Moreover, an MDD is **ordered** if given two nodes A and B such that A is the parent of B, we have that the variable represented into the node A is the smallest (*w.r.t.* a given total order over the set of variables).

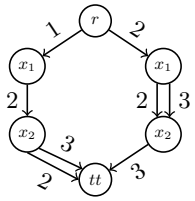

Fig. 1: The MDD corresponding to the valid assignments $\{(1,2,3),(2,2,3),(2,3,3),(1,2,2)\}$ for the variables x_0, x_1 and x_2. The structure presents 4 layers. The *root* and its edges represent the variable x_0 and its possible assignments.

One of the most interesting aspects of MDDs is their **reduction**. According to this procedure, they can gain an exponential factor in representation space. The reduction of MDDs aims at merging equivalent nodes *i.e.* that have equivalent outgoing paths. In particular, two nodes are equivalent if they have the same label and destination for each edge. The idea is to identify and merge equivalent nodes from the bottom to the top of the MDD, see Figure 2 for an example.

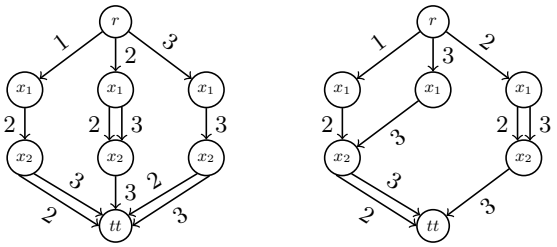

Fig. 2: An MDD before reduction (left) and its reduced version (right).

Several reduction algorithms for MDDs have been introduced [3, 7]. In our software we use the **pReduce algorithm** [14] since the time complexity per node is bounded by its number of outgoing arcs and the complexity is linear on the size of the MDD.

Another big advantage of using MDDs is that classical set operations such as Cartesian product, complement, intersection, union, difference, and many others can be performed without decompressing the structure. For instance, the cartesian product over MDDs can be performed just by transforming the *root* of an MDD into the *tt* node of another one (see Figure 3 for an example).

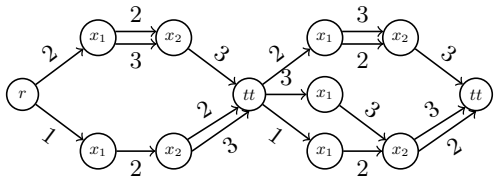

Fig. 3: The MDD representing the Cartesian product of the MDD in Figure 1 and the MDD in Figure 2 (right). The diagram is drawn horizontally for lack of room.

Given two MDDs, the intersection algorithm creates a new MDD to represent the solutions contained in both the original ones (see Figure 4). To achieve this goal, the process starts with the creation of a new root, and only the outgoing edges that are common between the two structures are recreated. In this way, each new node corresponds to two original nodes and the process is iterated. It is important to remember that in the end, it is necessary to verify if there are nodes without children; if any, they must be deleted. The drawback of this algorithm is that the resulting MDD can be larger than the original ones. For more details, we refer the reader to [14] and [4].

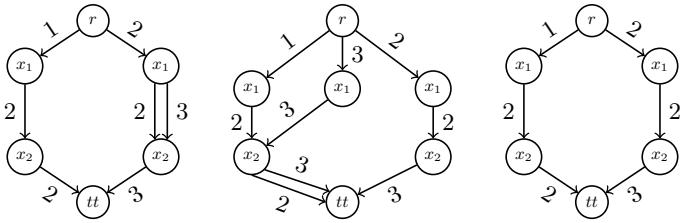

Fig. 4: Two MDDs (left and center) and their intersection (right).

2.2 Discrete Dynamical Systems and dynamics graphs

A Discrete Dynamical System (DDS) is a structure $\langle \chi, f \rangle$ where χ is a finite **set of states** and $f : \chi \to \chi$ is a function called the **next state map**. When modelling a phenomenon by a DDS $\langle \chi, f \rangle$, χ is its set of states and f is the law which brings from state $\alpha \in \chi$ at time t to the state $f(\alpha)$ at time $t + 1$.

When χ is finite, any DDS $\langle \chi, f \rangle$ can be identified with its **dynamics graph** $G \equiv \langle V, E \rangle$ where $V = \chi$ and $E = \{(\alpha, \beta) \in V \times V,\ f(\alpha) = \beta\}$. Therefore, all the properties of the DDS can be deduced from the properties of its dynamics graph. As a first property, one can remark that they are graphs with outgoing degree one and hence each strongly connected components of such graphs is made by a single cycle (or loop). From now on, we will turn all the discussion about DDS in terms of dynamics graphs. Call $\mathcal{DG}$ the set of all dynamics graphs up to (graph) isomorphism. One can define on $\mathcal{DG}$ two operations: sum and product as follows. Given two graphs, $G_1 = \langle V_1, E_1 \rangle \in \mathcal{DG}$ and $G_2 = \langle V_2, E_2 \rangle \in \mathcal{DG}$, the **sum** $G_1 + G_2$ is the graph $G = \langle V_1 \sqcup V_2, E_1 \sqcup E_2 \rangle \in \mathcal{DG}$, where $\sqcup$ is the disjoint union operator. The **product** $G_1 \cdot G_2$ is the graph $\langle V', E' \rangle \in \mathcal{DG}$ with $V' = V_1 \times V_2$ and $E' = \{((\alpha_1, \alpha_2), (\beta_1, \beta_2)) \in V' \times V', (\alpha_1, \beta_1) \in E_1 \text{ and } (\alpha_2, \beta_2) \in E_2\}$. The

product defined above consists in the parallel synchronous execution of the two dynamics graphs. The sum is the mutually exclusive alternative between the two behaviours.

$R := \langle \mathcal{DG}, +, \cdot \rangle$ is a commutative semiring in which $\langle \emptyset, \emptyset \rangle$ is the neutral element *w.r.t.* $+$ and $\langle \{\alpha\}, \{(\alpha, \alpha)\} \rangle$ is the neutral element *w.r.t.* multiplication.

Now, consider the semiring $R[x_1, x_2, \ldots, x_s]$ of polynomials over R in the variables x_i, naturally induced by R. Polynomial equations of the form (1) model hypotheses about a certain dynamics deduced from experimental data.

$$a_1 \cdot x_1^{w_1} + a_2 \cdot x_2^{w_2} + \ldots + a_k \cdot x_s^{w_s} = C \tag{1}$$

Equation (1) can be interpreted as follows. The constant term C on the right-hand side is the dynamical system deduced from experimental data. The point is that C comes just from experimental data and hence it might be the "macro" result of many cooperating hidden variables at a "micro" level. On the left hand side of (1), we have a hypothesis on the "micro" structure based on partial information (the coefficients) and unknown information (the variables). In other words, the coefficients a_i are hypothetical sub-dynamical systems that should cooperate to produce the observed dynamics C. Finding valid values for the unknown variables provides a finer structure for C which can bring further knowledge about the observed phenomenon.

More generally, **one can interpret Equation** (1) **as a question over dynamics graphs (*i.e.* directed graphs with outgoing degree 1)**. The constant right-hand side represents the current graph and the left-hand side is a question (hypothesis) about a possible decomposition (according to the semi-ring operations).

In [9], it has been proved that finding solutions to generic polynomial equations (*i.e.* in which both left and right-hand side of the equation are made by polynomials) over DDS is undecidable, while the problem is decidable when the right-hand side of the equation is constant. However, even in the decidable case, the complexity of the problem is beyond NP, except for very particular cases.

Some abstractions are introduced to progressively filter the solutions space according to features of the real solutions. Therefore, the general solutions are found in the intersection of the solutions of these abstractions. For this reason, the solutions enumeration of each abstraction is fundamental.

At least three abstractions can be devised on dynamics graphs, namely, abstraction on cycles, on cardinality (of the vertex set) and on paths. Studying each abstraction separately allows to study different aspects of a dynamics. In particular, solving equations over cycles leads to the validation of hypotheses over the long term behavior of a phenomenon.

3 The abstraction on cycles

In this paper we analyze equations over the cyclic part of the dynamics. Considering a generic Equation (1), we introduce a pipeline to solve the abstraction over the long term behavior but before we need to recall some notation and some concepts that will be useful in the sequel.

For a dynamics graph G, let $\mathring{G}$ be the subgraph of G which contains only the cycles and the loops. Denote $\mathcal{R}$ the restriction of R such that for any $G \in R$, $\mathring{G} \in \mathcal{R}$. It is not difficult to see that $\langle \mathcal{R}, +, \cdot \rangle$ is a sub-semiring of $\langle R, +, \cdot \rangle$. Thus, solving Equation (1) over $\mathcal{R}$ is a necessary step for solving it over R. This also implies that we have to enumerate all solutions in $\mathcal{R}$ first and then filter them out using the other abstractions to find the general solutions.

Notation 1 *A cycle* $\{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ *of length* p *can be conveniently denoted by* C_p^1. *A subgraph, with* K *different lengths of cycles is denoted* $\bigoplus_{i=1}^{K} C_{p_i}^{n_i}$.

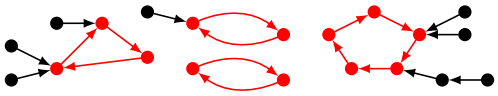

Fig. 5: A dynamics graph G, in which the subgraph $\mathring{G}$ is the red part and it is denoted $C_2^2 \oplus C_3^1 \oplus C_5^1$ according to our notation.

The operations of sum and product over $\mathcal{R}$ can be conveniently applied to the new notation. Graphically, we can consider the result of a sum as a new system composed by all the cycles of the input systems, and the product one as a new system generated with the Cartesian product of the cycles of the input systems.

Definition 1 (Sum). *Consider two dynamics graphs* $\mathring{A} \equiv \bigoplus_{i=1}^{K_A} C_{p_{Ai}}^{n_{Ai}}$ *and* $\mathring{B} \equiv \bigoplus_{j=1}^{K_B} C_{p_{Bj}}^{n_{Bj}}$, $\mathring{A} \oplus \mathring{B}$ *is* $\bigoplus_{i=1}^{K_A} C_{p_{Ai}}^{n_{Ai}} \oplus \bigoplus_{j=1}^{K_B} C_{p_{Bj}}^{n_{Bj}} = C_{p_{A1}}^{n_{A1}} \oplus \ldots \oplus C_{p_{AK_A}}^{n_{AK_A}} \oplus C_{p_{B1}}^{n_{B1}} \oplus \ldots \oplus C_{p_{BK_B}}^{n_{BK_B}}$.

Considering two sets of cycles $C_{p_{Ai}}^{n_{Ai}}$ and $C_{p_{Bj}}^{n_{Bj}}$, if they have the same cycles length ($p_{Ai} = p_{Bj}$), then they can be rewritten like $C_{p_{Ai}}^{n_{Ai}+n_{Bj}}$.

Definition 2 (Product). *Consider* $\mathring{A} \equiv \bigoplus_{i=1}^{K_A} C_{p_{Ai}}^{n_{Ai}}$ *and* $\mathring{B} \equiv \bigoplus_{j=1}^{K_B} C_{p_{Bj}}^{n_{Bj}}$, $\mathring{A} \odot \mathring{B}$ *is* $\bigoplus_{i=1}^{K_A} C_{p_{Ai}}^{n_{Ai}} \odot \bigoplus_{j=1}^{K_B} C_{p_{Bj}}^{n_{Bj}} = \bigoplus_{i=1}^{K_A} \bigoplus_{j=1}^{K_B} C_{p_{Ai}}^{n_{Ai}} \odot C_{p_{Bj}}^{n_{Bj}} = \bigoplus_{i=1}^{K_A} \bigoplus_{j=1}^{K_B} C_{\mathrm{lcm}(p_{Ai}, p_{Bj})}^{n_{Ai} \cdot n_{Bj} \cdot \gcd(p_{Ai}, p_{Bj})}$ *where* gcd *is the greatest common divisor and* lcm *is the least common multiple.*

With the help of the previous notation, Equation (1) can be rewritten as

$$\left(\bigoplus_{i=1}^{K_1} C_{p_{1i}}^{n_{1i}} \odot \mathring{x_1}^{w_1}\right) \oplus \left(\bigoplus_{i=1}^{K_2} C_{p_{2i}}^{n_{2i}} \odot \mathring{x_2}^{w_2}\right) \oplus \ldots \oplus \left(\bigoplus_{i=1}^{K_s} C_{p_{si}}^{n_{si}} \odot \mathring{x_s}^{w_s}\right) = \bigoplus_{j=1}^{m} C_{p_j}^{n_j} \quad (2)$$

where K_z is the number of distinct cycles size in the system a_z with $z \in \{1, \ldots, s\}$ (*cf.* Equation (1)) and n_{zi} is the number of cycles of length p_{zi} of a_z. In the right term C, there are m different periods, where for the j^{th} different period there are n_j cycles of period p_j. However, Equation (2) is still hard to solve in the present form. We can simplify it further by performing a **contraction step** which consists in cutting Equation (2) into two simpler equations

$$\begin{cases} C_{p_{11}}^{n_{11}} \odot \mathring{x_1}^{w_1} = \bigoplus_{j=1}^{m} C_{p_j}^{u_j} & \text{(3a)} \\ C_1^1 \odot \mathring{y} = \bigoplus_{j=1}^{m} C_{p_j}^{v_j} & \text{(3b)} \end{cases}$$

with $\mathring{y} = (\bigoplus_{i=2}^{K_1} C_{p_{1i}}^{n_{1i}} \odot \mathring{x_1}^{w_1}) \oplus (\bigoplus_{i=1}^{K_2} C_{p_{2i}}^{n_{2i}} \odot \mathring{x_2}^{w_2}) \oplus \ldots \oplus (\bigoplus_{i=1}^{K_s} C_{p_{si}}^{n_{si}} \odot \mathring{x_s}^{w_s})$ and $n_j = u_j + v_j$ for $j \in \{1, \ldots, m\}$. By recursively applying contraction steps, and for all possible u_j, v_j values in Equations (3a) and (3b), solving Equation (2) boils down to solve multiple times **simple equations** *i.e.* equations of the form

$$C_p^1 \odot X = C_q^n \tag{4}$$

where $p \in \{p_{11}, p_{12}, \ldots, p_{sK_s}\}$, $q \in \{p_1, p_2, \ldots, p_m\}$, and n is smaller than the number of cycles of length q in the right part.

Assume that X is $\mathring{x_z}^{w_z}$ such that $z \in \{1, \ldots, s\}$. It is necessary to find $\mathring{x_z}$ from X *i.e.* we need to compute the w_z root of X. Given $2M$ integers $p_i, k_i \in \mathbb{N}$, with $p_i > 0$ for all $i \in \{1, \ldots, M\}$, let $l(p_1, p_2, ..., p_t, k_1, k_2, ..., k_t)$ be the lcm between the p_i for which $k_i \neq 0$ and $t \leq M$ (with $l(p_1, p_2, ..., p_t, k_1,\ k_2, ..., k_t) = 1$ iff $\forall 1 \leq i \leq t$, $k_i = 0$). Assume $\mathring{A} \equiv C_{p_1}^1 \oplus C_{p_2}^1 \oplus ... \oplus C_{p_M}^1$. Then,

$$\left(\mathring{A}\right)^w \equiv \bigoplus_{i=1}^{M} C_{p_i}^{p_i^{w-1}} \oplus \bigoplus_{\substack{k_1+k_2+\ldots+k_M=w \\ 0\leq k_1,k_2,\ldots,k_M<w}} \binom{w}{k_1,k_2,\ldots,k_M} C_{l(p_1,p_2,\ldots,p_M,k_1,k_2,\ldots,k_M)}^{\prod_{t=1}^{M} p_t^{k_t-1} \cdot \prod_{\substack{t=2 \\ k_t\neq 0}}^{M} gcd(l(p_1,\ldots,p_{t-1},k_1,\ldots,k_{t-1}),p_t)} . \tag{5}$$

3.1 The state-of-art method

In [10], two computational problems are devised concerning simple equations. The SOBFID (*SOlve equation on BIjective Finite DDS*) problem is a decision problem which takes in input $p, n, q \in \mathbb{N} \setminus \{0\}$ and returns true iff $C_p^1 \odot X = C_q^n$ admits a solution. EnumSOBFID is the problem which takes the same input as SOBFID and outputs the list of all solutions of $C_p^1 \odot X = C_q^n$.

To the best of our knowledge, the Colored-Tree Method (CTM) proposed in [10] is the best current technique to solve this problem. The method exploits a connection between EnumSOBFID and the well-known Change-making problem [11] coupled with a completeness-check (running in exponential time) to explore the feasible solutions space. Essentially, the right term n is decomposed in every possible way and these possibilities are represented in a tree. The method comprises two main phases: tree building and solutions aggregation. In the first one, the algorithm uses a tree to decompose the right part of the equation in a certain number of subgraphs, searching for each node of the tree (each subgraph) the minimum number of product operations (minimum number of cycles in the variable or minimum number of children) which are necessary to produce it. The subgraphs found are then divided into subsets of children. Iterating this idea on each subset produced, the method arrives to enumerate all the possible ways to generate the cycles involved in the right part. During the second phase, the method computes (bottom-up) the real solutions of the equation represented in the tree.

4 Boosting everything up with MDDs

The enumeration of solutions for Equation (2) is one of the main objective of this paper. In order to achieve this, we solve the EnumSOBFID problem and

show that MDDs boost up the enumeration. In other words, we are interested to find all the possible ways to generate C_q^n cycles starting with one cycle of length p. This last problem is similar to the well-known Change-making problem (in its enumeration version) in which one aims at finding all the possible ways to change a total amount with a given set of coins. In our case, the total amounts are the C_q^n cycles but to complete the similarity we need to find the coins which can be part of a solution. Remark that a cycle C_u^1 generates C_q^r cycles in the right part iff r divides q, $u = \frac{q}{p} \cdot r$, $\gcd(p, \frac{q}{p} \cdot r) = r$ and $\mathrm{lcm}(p, \frac{q}{p} \cdot r) = q$. A cycle C_u^1 with the previous properties is called **feasible** and r is a **feasible divisor** of q.

Let $D_{p,q} = \{d_1, \ldots, d_l\}$ be the set of feasible divisors (*w.r.t.* Equation (4) of course). There is a solution to (4) iff there exist $\overline{x_1}, \ldots, \overline{x_l}$ such that $\sum_{i=1}^{l} d_i \overline{x_i} = n$ (*i.e.* there is a solution to the Change-making problem for a total amount n and a coins system $D_{p,q}$). To solve EnumSOBFID we need to enumerate all solutions of the previous Change-making problem. At this point MDDs come into play. We are going to use them to have a compact and handful representation of the set of all possible solution to Equation (4).

SB-MDD The MDD $M_{p,q,n}$ containing all solutions to Equation (4) is a labelled digraph $\langle V, E, \ell \rangle$ where $V = \bigcup_{i=1}^{Z} V_i$ with $Z = \lfloor \frac{n}{\min D_{p,q}} \rfloor + 1$ and $V_1 = \{root\}$, V_i is a multiset of $\{1, \ldots, n-1\}$, and, finally, $V_Z = \{tt\}$. For any node $\alpha \in V$, let $val(\alpha) = \alpha$ if $\alpha \neq root$ and $\alpha \neq tt$, $val(root) = 0$ and $val(tt) = n$. For any $i \in \{1, \ldots, Z-1\}$ and for any $\alpha \in V_i$ and $\beta \in V_{i+1} \cup \{tt\}$, $(\alpha, \beta) \in E$ iff $val(\beta) - val(\alpha) \in D_{p,q}$ and $val(\beta) \leq val(tt)$. The labelling function $\ell \colon E \to D_{p,q}$ is such that for any $(\alpha, \beta) \in E$, $\ell((\alpha, \beta)) = val(\beta) - val(\alpha) \in D_{p,q}$.

Graphically, $M_{p,q,n}$ can be represented by layers as usual, interpreting each layer as the usage of the i-th coin. Each node represents the sum of coins from the *root* to the node. Moreover, the longest path in $M_{p,q,n}$ is $Z = \lfloor \frac{n}{\min D_{p,q}} \rfloor + 1$. Remark that $M_{p,q,n}$ contains duplicated solutions, indeed, it contains all the permutations of a solution but according to Equation (4) different permutations lead to the same solution. For this reason, we impose a **symmetry breaking constraint**: for any node α (different from tt), let e be the label of the incoming edge; the only allowed outgoing edges of α are those with label $\ell \leq e$. In this way all the paths of the MDD will be ordered and the size of the MDD will be smaller. An **SB-MDD** is an MDD which satisfies the symmetry breaking constraint. The building of $M_{p,q,n}$ ends with a pReduction which merges equivalent nodes and deletes all nodes (and the corresponding edges) which are not on a path from *root* to tt. Let us illustrate the construction with an example.

Example 1. Consider the simple equation $C_2^1 \odot X = C_6^6$. The set of divisors of q (smaller or equal to n) is $\{6, 3, 2, 1\}$. However, $D_{p,q} = \{1, 2\}$. Indeed, we have

$$\begin{aligned} r = 6 \wedge u = 18 &\rightarrow gcd(2, 18) \neq 6 \wedge lcm(2, 18) \neq 6 \\ r = 3 \wedge u = 9 &\rightarrow gcd(2, 9) \neq 3 \wedge lcm(2, 9) \neq 6 \\ r = 2 \wedge u = 6 &\rightarrow gcd(2, 6) = 2 \wedge lcm(2, 6) = 6 \\ r = 1 \wedge u = 3 &\rightarrow gcd(2, 3) = 1 \wedge lcm(2, 3) = 6 \end{aligned}$$

Figure 6 shows $M_{2,6,6}$ in its classic form (left) and in its SB-MDD form (right). Remark that in Figure 6 (left) many solutions are duplicated. For example, the solution $[2, 2, 1, 1]$ (in red) is represented (more than) twice. Once $M_{2,6,6}$ is built and reduced, reading solutions correspond to paths labels from *root* to *tt*: $\{[2, 2, 2], [2, 2, 1, 1], [2, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]\}$.

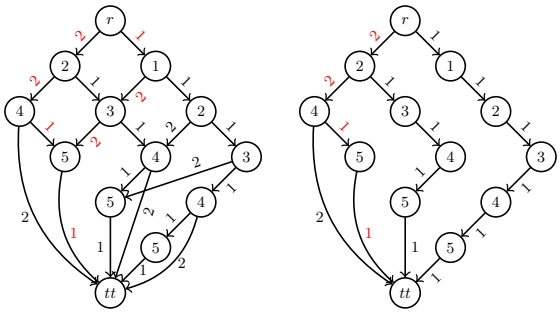

Fig. 6: The reduced MDD representing all the solutions of $C_2^1 \odot X = C_6^6$ in its classic form (left) and in its SB-MDD form (right).

The solution $[2, 2, 2]$ says that 6 is changed with 3 coins of value $r = 2$. Recalling that $u = \frac{q}{p} \cdot r$ we can express the solution in term of dynamics graphs as C_6^3. Operating similarly for all the other solutions we find $\left\{C_6^3, C_6^2 \oplus C_3^2, C_6^1 \oplus C_3^4, C_3^6\right\}$.

4.1 Equations over dynamics graphs

We have shown how MDDs can be used to compute the solutions set of simple equations. In this section, we introduce a pipeline to solve Equations of form (2) to validate hypotheses over discrete systems represented by DDS. The goal is the enumeration of solutions. The pipeline consists in the following steps:

- identification and resolution of the necessary equations (each necessary equation corresponds to an SB-MDD);
- enumeration of the contractions steps by an MDD structure;
- computation of the solutions of each contraction step with a particular technique to compute the intersection between SB-MDDs.

Necessary equations. As we have already seen, to solve a generic equation over dynamics graphs we need to find the solutions of a certain number of contraction steps (Equations (3a) and (3b)). Each of these contraction steps ends up with a system of equations of the form

$$C_{p_{zi}}^{n_{zi}} \odot X_z = \bigoplus_{j=1}^{m} C_{p_j}^{v_j} \tag{6}$$

where $z \in \{1, \ldots, s\}$, $i \in \{1, \ldots, K_z\}$, and $v_j \leq n_j$. This means that each monomial is responsible for the generation of a subgraph of the right term. In other words, the solution of (6) is the Cartesian product of the solutions of a certain number of simple equations. Remark that a single simple equation might occur several times while searching for a solution of a single contraction step or in multiple contraction steps.

According to the product rules, each equation $C_{p_{zi}}^{n_{zi}} \odot X_z = C_{p_j}^{v_j}$ is equivalent to $C_{p_{zi}}^{1} \odot X_z = C_{p_j}^{\frac{v_j}{n_{zi}}}$. Remark that if v_j/n_{zi} is not an integer, then the equation has no solutions.

Since there are many contraction steps that need to be explored, we aim at limiting the exploration by ignoring those which involve simple equations without solutions. Therefore, the method starts with the computation of the set of simple equations that can be involved in the result of a contractions step. This amounts to compute all the SB-MDDs $M_{p_{zi},p_j,\frac{n}{n_{zi}}}$ with $p_{zi} \in \{p_{11},\ldots,p_{sK_s}\}$, $p_j \in \{p_1,\ldots,p_m\}$, for all $n \in \{1,\ldots,n_j\}$. In this way, even if a simple equation is involved in many contraction steps, it is solved only once. A simple equation with at least one solution is called **necessary equation**. It is important to notice that it is not necessary to explore an SB-MDD to decide if a solution exists. Indeed, by construction, an SB-MDD is generated iff it contains at least a solution.

Contractions steps. Once the set of necessary equations has been computed, we create an MDD CS to enumerate all the contractions steps that must be taken into account to enumerate the solutions of Equation (2). This MDD will be a Cartesian product of other MDDs, one for each p_j of the right term, *i.e.* $CS = \bigtimes_{j=1}^{m} CS_j$.

Consider a cycle length p_j in the constant term. The MDD CS_j accounts for all the feasible ways (according to the set of necessary equations) to generate n_j cycles of length p_j using the monomials of the equation. Therefore, CS_j is a labelled digraph $\langle V_j, E_j, \ell_j\rangle$ where $V_j = \bigcup_{z=1}^{s+1}\bigcup_{i=1}^{K_z} V_{j,zi}$ with $K_{s+1} = 1$ (see Equation (2) for the meaning of K_i) and $V_{j,11} = \{root\}$, $V_{j,zi} \subseteq \{0,...,n_j\}$, and, finally, $V_{j,(s+1)1} = \{tt\}$. For any node $\alpha \in V_j$, let $val(\alpha) = \alpha$ if $\alpha \neq root$ and $\alpha \neq tt$, $val(root) = 0$ and $val(tt) = n_j$. The set of possible outgoing edges of level $V_{j,zi}$ is $D_{p_{zi},p_j} = \{g \in \mathbb{N} \mid 1 \leq g \leq n_j \text{ and } M_{p_{zi},p_j,\frac{g}{n_{zi}}} \in \text{necessary equations}\} \cup \{0\}$. For any $\alpha, \beta \in V_j$, $(\alpha,\beta) \in E_j$ iff

1. $\alpha \in V_{j,zi}$ and either $\beta \in V_{j,z(i+1)}$ for some $i < K_z$ or $\beta \in V_{j,(z+1)1}$;
2. $val(\beta) - val(\alpha) \in D_{p_{zi},p_j}$ and $val(\beta) \leq val(tt)$.

In this MDD the outgoing edges of a level $V_{j,zi}$ represent the cycles of length p_j generated by the monomial $C_{p_{zi}}^{n_{zi}} \odot X_z$ of the left part. The labelling function $\ell_j \colon E_j \to \bigcup_{z=1}^{s}\bigcup_{i=1}^{K_z} D_{p_{zi},p_j}$ is such that for any $(\alpha,\beta) \in E_j$, $\ell_j((\alpha,\beta)) = val(\beta) - val(\alpha) \in D_{p_{zi},p_j}$ with $\alpha \in V_{j,zi}$. Starting from each node, a label 0 means that the monomial is not involved in the generation of the cycles of length p_j. The sum of the labels of each path from the *root* to the *tt* node will be equal to n_j, because n_j cycles of length p_j must be generated.

For each path of CS, it is necessary to perform some some additional steps to understand if it leads to feasible solutions of the equation as explained in the next section.

Solve a system. Each contraction step corresponds to a system of Equations of type (6). To compute the solutions set of these equations one needs to compute the Cartesian product between the different solutions of the corresponding

simple equations. In its turn, a solution to a simple equation is represented by a SB-MDD. Therefore, their Cartesian product is computed in linear time by placing the SB-MDDs one on top of the other to form a new MDD. We call **SB-Cartesian MDD** an MDD build in such a way. Remark that an SB-Cartesian MDD is not a SB-MDD.

Recall that X_z in Equation (6) represents a variable ${\mathring{x}_z}^{w_z}$ and if $\mathring{x}_z$ is involved in different monomials, then, an intersection operation is required. To compute the solutions set of the variable, we start considering equations involving $\mathring{x}_z$ with the same power w_z by computing the intersection over the corresponding MDDs. Moreover, remark that each ${\mathring{x}_z}^{w_z}$ corresponds to a SB-Cartesian MDD (except for the case in which a monomial is responsible for the generation of only one cycle length). However, notice that the classic algorithm to perform the intersection over MDDs cannot be used if the goal is the intersection between MDDs issued by a Cartesian product (*i.e.* SB-Cartesian MDDs) because the result depends on the order of the MDDs. In the next section, we propose a new algorithm to perform this task independently from the order.

Once ${\mathring{x}_z}^{w_z}$ is assigned with a set of solutions, we need to compute the w_z-th root for each value of ${\mathring{x}_z}^{w_z}$ and finally the intersection between all the roots found so far.

Finally, we stress that the root procedure is not a trivial step. Indeed, the inverse operations of sum and product are not definable in the commutative semiring of DDS. Therefore, we need an algorithmic technique to compute the result of the w-th root of $\mathring{x}$. Considering Formula (5), the root can be computed combinatorially or through a finite number of polynomials equations over real numbers of increasing degree. We combine both approaches in order to speedup the computations. Consider a generic system $\mathring{x} = C_{p_1'}^{n_1'} \oplus C_{p_2'}^{n_2'} \oplus \ldots \oplus C_{p_l'}^{n_l'}$ such that $\mathring{x}^w = C_{p_1}^{n_1} \oplus C_{p_2}^{n_2} \oplus \ldots \oplus C_{p_h}^{n_h}$, the number of cycles n_1', of the minimum length p_1, involved into the root's solution is computed with the polynomial equation $(p_1)^{w-1} \cdot (n_1')^w = n_1$ with $p_1 = p_1'$. The remaining part of the solution is combinatorially computed knowing that a length of cycle p_i may be involved in the root solution iff $n_i \geq (p_i)^{w-1}$, and we will have a maximum number of cycles of this length equals to $\frac{n_i}{(p_i)^{w-1}}$.

SB-Cartesian Intersection. Consider a set $\overline{M}$ of MDDs in which some are SB-Cartesian MDDs and others are not. We propose an algorithm (Algorithm 1) which computes the intersection of all the elements in $\overline{M}$. Our algorithm needs to be started with an initial set of candidate solutions $\overline{S}$ (*initial guess*). If $\overline{M}$ contains at least a simple SB-MDD (*i.e.* one which is not a SB-Cartesian MDD), then $\overline{S}$ is the set of solutions read in the MDD resulting from classical intersection of the SB-MDDs in $\overline{M}$; otherwise $\overline{S}$ is the set of solutions read in an arbitrarily chosen SB-Cartesian of $\overline{M}$. Using $\overline{S}$, we will compute the intersection between the remaining SB-Cartesians. The idea is to search the solutions into each SB-Cartesian MDD and update each time the remaining solutions.

Each candidate solution is ordered and recursively searched in a SB-Cartesian MDD in $\overline{M}$ (Algorithms 2 and 3). If it is not found, then it is removed from the

set of candidate solutions. Given a SB-Cartesian element M of $\overline{M}$, a candidate solution s is validated if it possible to visit each SB-MDD involved in M using a subset of the elements of the solution. Starting from the biggest elements of s, we try to find a path from the *root* to the first *tt* node (recall that we are visiting a SB-Cartesian MDD). We proceed in this way because the paths of each SB-MDD are ordered. After having gone through the first SB-MDD, we need to recursively repeat the procedure with the remaining elements of s.
Two special cases need our attention:

- a generic node in which it is not possible to find a common element between the remaining elements of a solution and the outgoing edges;
- a node (different from the final *tt* node) in which there are no more elements of s to compare with the outgoing edges.

In both cases, it is necessary to go back in the visited nodes until there is another possible outgoing edge that can be taken into consideration. Once the validation procedure arrives at the last SB-MDD of M, a linear search is performed with the remaining elements. Finally, $s \in M$ if there exists a path from the last *root* to the final *tt* node following the remaining elements (Algorithm 4). If the linear search fails, then we return to the first node with a different feasible outgoing edge. If no different feasible edges exist, then s is not a solution. The previous procedure is performed over each candidate solution of the initial guess.
We stress that in the worst case, for a given candidate solution, our algorithm explores only the subgraph of a SB-Cartesian MDD made of feasible edges.

Algorithm 1: SB-Cartesian Intersection

```
Input  : M̄, set of MDDs
Output: S̄, solutions of the intersection in M̄
Cartesian← ∅;
Traditional← ∅;
forall m ∈ M̄ do
    if m is a SB-Cartesian MDD then Cartesian.add(m) ;
    else Traditional.add(m) ;
S̄ ← ∅;
if |Traditional|> 0 then
    if |Traditional| = 1 then
        S̄ ←Traditional[0].readSolutions();
    else
        MddIntersected←ClassicIntersection(Traditional[0],Traditional[1]);
        forall m ∈Traditional \ {Traditional[0], Traditional[1]} do
            MddIntersected←ClassicIntersection(MddIntersected,m);
        S̄ ←MddIntersected.readSolutions();
else
    S̄ ←Cartesian[0].readSolutions();
    Cartesian.remove(0);
if |S̄|≠ 0 then
    forall m ∈Cartesian do
        CartesianSearch(S̄,m);
return S̄
```

The approach introduced to compute the intersection of SB-Cartesian MDDs is suitable for our needs, its improvement constitutes an interesting future research direction. We need to precise an additional aspect of our application. As explained above, an SB-MDD corresponding to a simple equation has la-

bels based on the feasible divisors set (or feasible coins) and each divisor/coin corresponds to a certain cycle C_u^1. However, two different coins of two different simple equations can correspond to the same C_u^1 as well as the same coin may correspond to different cycles lengths for different simple equations. Therefore, when searching for a solution in a SB-Cartesian MDD, we must take into account these cases. In our application, two divisors/coins are considered equivalent if they correspond to the same C_u^1.

Algorithm 2: SB-Cartesian Search

```
Input  : S set of solutions, M a SB-Cartesian MDD
Output: S̄ solutions involved in M
S̄ ← ∅;
forall s ∈ S do
    s.order();
    find←FindSolution(s,0,M);
    if find then S̄.add(s) ;
return S̄
```

Algorithm 3: FindSolution

```
Input  : s solution, 0 ≤ i < |M|, M a SB-Cartesian MDD
Output: true if s ∈ M, false otherwise
return FindSolutionNode(s,M[i].root,i)
```

Algorithm 4: FindSolutionNode

```
Data: S sub-solution, N node, 0 ≤ i < |M|, M a SB-Cartesian MDD
Result: true if S ∈ M, false otherwise
find←FALSE;
if N is not a tt then
    forall e ∈ S.removeDuplicates() ∧ find=FALSE do
        if N.edge.contains(e) then
            newSubSolution← S \ {e} ;
            find←findSolutionNode(newSubSolution,N.children(e),i);
else
    i ← i + 1;
    if i= |M|−1 then
        valid←LinearSearch(S,M[i].root);
        if valid then return TRUE ;
    else
        return FindSolution(S,i)
return find
```

5 Experiments

The experimental evaluation is divided into two parts: one concerns simple equations and the other one is devoted to the complete method which solves generic equations.

Concerning equations of the form $C_p^1 \odot X = C_q^n$, in our experiments, we set $p = q$ since this grants the existence of at least a solution. Using the MDDs it is possible to outperform the Colored-Tree method (CTM) *w.r.t.* both memory and time. If we compare the dimension (in terms of nodes) of a colored-tree with the corresponding SB-MDD for a given equation, the second one is smaller. CTM presents some out of memory cases even for equations with n, q, and p smaller than 30 and memory limit of 30GB (see Figure 7 (left)). Using MDDs, we solved equations with n, q, and p up to 100 without any out of memory case and only 6GB RAM limit (see Figure 7 (right)).

Analysing the time to solve equations with parameters up to 30, it turns out that the new technique is faster than the previous one (see Table 1). The reason is that CTM requires a time consuming check procedure to ensure the completeness of the solutions which is not necessary in the MDD case. Due to too high memory and time costs, CTM is unsuitable to solve simple equations coming from contractions steps. The new method fixes these issues allowing to solve generic polynomial equations.

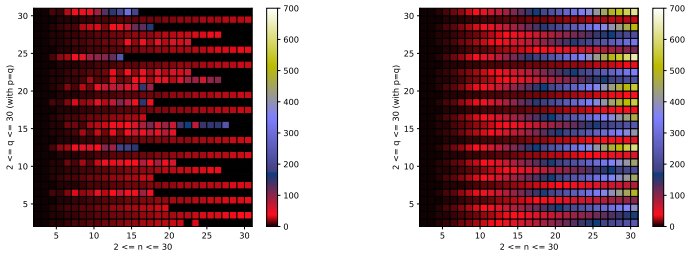

Fig. 7: The number of nodes for the colored-tree with memory limit of 30GB (left) and for the SB-MDD with memory limit of 4GB (right) in the case of equation for type $C_q^1 \odot X = C_q^n$. Remark that the black square in the right part of the left diagram are out of memory cases.

Turning to generic equations, we use the proposed pipeline to find the solutions (if any). If roots computation is not performed then everything depends on the number of monomials, the number of distinct sizes of cycles in the right side of the equations. If these quantities grow also the number of contraction steps to consider grows. Solutions spaces are limited by considering only contraction steps that are feasible according to the necessary equations. For example, consider the equation $C_5^1 \odot \mathring{y} \oplus C_4^1 \odot \mathring{y}^2 \oplus C_3^1 \odot \mathring{x}^2 \oplus C_2^1 \odot \mathring{x}^3 = C_{10}^3 \oplus C_4^{68} \oplus C_3^9 \oplus C_6^9 \oplus C_{12}^{136}$, the number of contraction steps is $\approx 2,42 \cdot 10^{16}$, but only 6665400 are considered.

The computation of roots is the most time consuming operation. Indeed, consider the equation $C_3^2 \odot \mathring{x} \oplus C_3^4 \odot \mathring{y} = C_3^{162} \oplus C_6^{20} \oplus C_{12}^{104}$, we can find 49329000 solutions considering the 6642 feasible contraction steps in only 58 seconds. If we consider the same equation but with $\mathring{x}^2$ in place of $\mathring{x}$, then the number of contraction steps explored is the same, but we found 4510 solutions in 5.6 hours (therefore the average computational time per contraction is 3.04 seconds).

Roots computation is expensive and this is reasonable in a sense. Since the division and subtraction operations cannot be performed directly, one needs sophisticated techniques. The pipeline introduced above leads to the first complete technique to solve equations over the asymptotic behavior of DDS. In the end, the pipeline is already proving itself suitable for applications over real experimental data.

6 Conclusions and perspectives

Equations on DDS are useful to analyze dynamics of phenomena. They allow to model hypotheses on the dynamical behavior and solving them leads to their

validation or invalidation. In particular, Equation (2) allow studying the long-term aspects of the dynamics.

This paper introduces new algorithms to solve these equations over cycles (*i.e.* long term behavior). In the case of simple equations, our technique outperforms the CTM. This is an important breakthrough because CTM was used to solve simple equations (generated by contractions steps) and it was practically unusable due to huge memory consumption and time-consuming check procedures. Moreover, this paper proposes a pipeline to solve general equations. The pipeline computes the necessary equations to limit the exploration of the contractions steps and, in the end, it solves each system of equations corresponding to a contractions step. This allows the treatment of much larger dynamics graphs and much more complicated hypotheses.

Future perspectives include the improvement of the pipeline by parallelising the identification of solutions of different feasible contractions steps. Another research direction would try to speedup roots computation by increasing the number of coefficients computed through polynomial equations to reduce the combinatorics. In the end, this work aims to call for further research in MDDs. In fact, studying different ways to perform the intersection between SB-Cartesian MDDs is surely another subject that is worth exploring.

This research work leads to a complete and performing pipeline to validate hypotheses over the long term behavior of dynamics graphs adding one more item to the growing list of successful applications of MDDs.

q,P \ n	2	3	4	5	6	7	8	9	10
2	56|0	56|0	54|0	54|0	53|0	54|0	54|0	54|0	53|0
3	54|1	54|0	55|1	54|0	55|1	54|0	55|1	54|0	55|1
4	55|2	54|1	56|2	53|0	57|2	54|0	55|2	55|1	55|2
5	55|2	56|2	59|2	54|0	61|30	54|0	58|2	55|3	56|2
6	58|2	56|2	60|2	58|1	63|9	55|0	60|2	56|2	62|2
7	60|3	58|2	63|19	56|1	78|21	54|0	63|29	60|2	61|2
8	63|2	59|2	96|10	60|2	107|20	56|1	97|9	61|2	65|2
9	66|3	60|3	106|21	57|2	153|22	57|1	168|21	60|3	76|20
10	84|3	62|2	140|11	58|2	185|21	57|2	369|11	70|2	120|17

q,P \ n	11	12	13	14	15	16	17	18	19	20
11	55|0	824|25	55|0	116|3	74|21	406|22	55|0	1071|22	57|0	1334|23
12	58|1	17678|26	57|0	132|4	88|21	4105|12	56|0	6022|22	56|0	3672|22
13	61|2	177894|27	56|0	246|21	92|21	4163|27	55|0	5967|24	56|0	3332|24
14	59|2	1277979|26	61|1	900|11	116|22	19895|24	56|0	27381|27	56|0	96997|26
15	60|2	-|28	60|2	3721|22	169|22	19711|25	56|0	637457|26	59|0	419000|25
16	62|2	-|29	61|2	19900|12	502|23	-|13	57|0	1185947|26	60|0	759365|26
17	62|2	-|30	62|2	25908|24	554|23	-|26	57|0	-|27	57|0	-|27
18	64|2	-|46	62|2	164167|13	1102|22	-|26	61|2	-|28	61|0	-|30
19	66|2	-|32	63|2	226315|25	950|24	-|27	62|2	-|34	57|0	-|39
20	68|2	-|32	65|2	1707299|25	2749|24	-|16	63|2	-|31	62|2	-|29

q,P \ n	21	22	23	24	25	26	27	28	29	30
21	2712|23	343542|26	60|0	-|35	95|4	389971|7	2034|12	-|28	58|0	-|37
22	23399|24	-|14	62|0	-|36	103|4	381929|7	2711|24	-|30	59|0	-|35
23	27430|24	-|27	59|0	-|38	134|4	-|7	2712|24	-|31	59|0	-|37
24	149296|25	-|16	64|2	-|39	149|4	-|8	20641|14	-|42	59|0	-|41
25	162413|25	-|28	65|2	-|40	160|4	-|26	24632|24	-|34	59|0	-|39
26	212277|25	-|16	66|2	-|42	403|5	-|15	24177|25	-|33	60|0	-|40
27	-|25	-|27	69|2	-|45	454|4	-|33	-|15	-|34	59|0	-|45
28	-|26	-|17	70|2	-|47	488|5	-|16	-|25	-|35	60|0	-|44
29	-|28	-|28	69|2	-|66	506|5	-|28	-|27	-|36	61|1	-|46
30	-|26	-|27	69|3	-|51	838|6	-|17	-|17	-|38	65|2	-|48

Table 1: Computation times (millisec) of CTM (left) and MDD (right) over different input parameters. Symbols '-' represent out of memory cases.

Acknowledgments This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on computers pp. 509–516 (1978)
2. Amilhastre, J., Fargier, H., Niveau, A., Pralet, C.: Compiling csps: A complexity map of (non-deterministic) multivalued decision diagrams. International Journal on Artificial Intelligence Tools **23**(04), 1460015 (2014)
3. Andersen, H.R.: An introduction to binary decision diagrams. Lecture notes, available online, IT University of Copenhagen p. 5 (1997)
4. Bergman, D., Cire, A.A., van Hoeve, W.: Mdd propagation for sequence constraints. Journal of Artificial Intelligence Research **50**, 697–722 (2014)
5. Bergman, D., Cire, A.A., Van Hoeve, W.J., Hooker, J.: Decision diagrams for optimization, vol. 1. Springer (2016)
6. Berndt, R., Bazan, P., Hielscher, K.S., German, R., Lukasiewycz, M.: Multi-valued decision diagrams for the verification of consistency in automotive product data. In: 2012 12th International Conference on Quality Software. pp. 189–192. IEEE (2012)
7. Cheng, K.C., Yap, R.H.: An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints **15**(2), 265–304 (2010)
8. Darwiche, A., Marquis, P.: A knowledge compilation map. Journal of Artificial Intelligence Research **17**, 229–264 (2002)
9. Dennunzio, A., Dorigatti, V., Formenti, E., Manzoni, L., Porreca, A.E.: Polynomial equations over finite, discrete-time dynamical systems. In: Proc. of ACRI'18. pp. 298–306 (2018)
10. Dennunzio, A., Formenti, E., Margara, L., Montmirail, V., Riva, S.: Solving equations on discrete dynamical systems. In: Cazzaniga, P., Besozzi, D., Merelli, I., Manzoni, L. (eds.) Computational Intelligence Methods for Bioinformatics and Biostatistics. pp. 119–132. Springer International Publishing, Cham (2020)
11. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., New York, NY, USA (1990)
12. Nakahara, H., Jinguji, A., Sato, S., Sasao, T.: A random forest using a multi-valued decision diagram on an fpga. In: 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL). pp. 266–271. IEEE (2017)
13. Naldi, A., Thieffry, D., Chaouiya, C.: Decision diagrams for the representation and analysis of logical models of genetic networks. In: International Conference on Computational Methods in Systems Biology. pp. 233–247. Springer (2007)
14. Perez, G., Régin, J.C.: Efficient operations on mdds for building constraint programming models. In: IJCAI 2015 (2015)
15. Zaitseva, E., Levashenko, V., Kostolny, J., Kvassay, M.: A multi-valued decision diagram for estimation of multi-state system. In: Eurocon 2013. pp. 645–650. IEEE (2013)
16. Zhang, L., Xing, L., Liu, A., Mao, K.: Multivalued decision diagrams-based trust level analysis for social networks. IEEE Access **7**, 180620–180629 (2019)