MDDs are Efficient Modeling Tools: An Application to some Statistical Constraints

Guillaume Perez and Jean-Charles Régin

Université Nice-Sophia Antipolis, I3S, CNRS, Sophia Antipolis, France
guillaume.perez06@gmail.com, jcregin@gmail.com

Abstract. We show that from well-known MDDs like the one modeling a sum, and operations between MDDs we can define efficient propagators of some complex constraints, like a weighted sum whose values satisfy a normal law. In this way, we avoid defining ad-hoc filtering algorithms. We apply this idea to different dispersion constraints and on a new statistical constraint we introduce: the Probability Mass Function constraint. We experiment out approach on a real world application. The conjunction of MDDs clearly outperforms all previous methods.

1 Introduction

Several constraints, like `spread` [16], `deviation` [21,22,24,23], `balance` [3,5] and `dispersion` [17], have mainly been defined to balance certain features of a solution. For example, the balanced academic curriculum problem [1] involves courses that have to be assigned to periods so as to balance the academic load between periods. Most of the time the mean of the variables is fixed and the goal is to minimize the standard deviation, the distance or the norm.

The `dispersion` constraint is a generalization of the `deviation` and `spread` constraints. It ensures that X, a set of variables, has a mean (i.e. $\mu = \sum_{x \in X} x$) belonging to a given interval and $\varDelta$ a norm (i.e. $\sum_{x \in X}(x-\mu)^p$) belonging to another given interval. If $p = 1$ then it is a `deviation` constraint and $p = 2$ defines a `spread` constraint. Usually, the goal is to minimize the value of $\varDelta$ or find a value below a given threshold.

In some problems, variables are independent from a probabilistic point of view and are associated with a distribution (e.g. a normal law) that specifies probabilities for their values. Thus, globally the values taken by the variables have to respect that law and we can define a constraint ensuring this property, either by using a `spread`, a `dispersion`, a `KolmogorovSmirnov` or a `Student's t-test` constraint[19]. However, if only a subset of variables is involved in a constraint, then the values taken by these variables should be compatible with the distribution (e.g. the normal law), but we cannot impose the distribution for a subset of values because this is a too strong constraint. Therefore, we need to consider interval of values for μ and $\varDelta$. The definition of an interval for μ can be done intuitively. For instance we can consider an error rate of 10%. Unfortunately, this is not the case for $\varDelta$. It is hard to control the relation between two values of $\varDelta$, because data are coming from measures and there are some errors and because it is difficult to apply a continuous law on finite set of values. Since we use constraint programming solvers we have to make sure that we do not forbid tuples that could be acceptable. This is why, in practice, the problem is not defined in term of μ and Δ but by the probability mass function (PMF). The probability mass function gives the probability that X_r, a discrete random variable, is exactly equal to some value. In other words, if X_r takes its values in V, then the PMF gives the probability of each value of V. The PMF is usually obtained from the histogram of the values. From f_P, a PMF, we can compute the probability of any tuple by multiplying the probability of the values it contains, because variables are independent. Then, we can avoid outliers of the statistical law but imposing that the probability of a tuple belongs to a given interval $[P_{min}, P_{max}]$. With such a constraint we can select a subset of values from a large set having a mean in a given interval while avoiding outliers of the statistical law. Roughly, the minimum probability avoids having tuples with values having only very little chance to be selected and the maximum probability avoids having tuples whose values have only the strongest probability to be selected. Thus, we propose to define the `PMF` constraint, a new statistical constraint from μ, f_P, P_{min} and P_{max}.

Since we define a new constraint we need to define a propagator for it. Instead of designing an ad-hoc propagator we propose to represent the constraint by an MDD and to use MDD propagators, like MDD4R [13], for establishing arc consistency of the constraint. MDDs of constraints can also be intersected in order to represent the combination of the constraints and MDD operators applied on them. Recent studies show that such combinations give excellent results in practice [20].

Pesant has proposed a specific algorithm for filtering the dispersion constraint. His propagator establishes domain consistency on the X variables. Unfortunately, it is ad-hoc and so it cannot be easily combined with other constraints. Thus, we propose to also use MDD propagators for the classical version of the dispersion constraint.

The advantage of using MDDs representing the constraints in these cases, is that these MDDs are defined on sum constraints which are well-known MDDs.

We tested all propagators on a part of a real world application mainly involving convolutions which is expressed by a knapsack constraint (i.e. $\sum \alpha_i x_i$). The results show the advantage of our generic approach.

The paper is organized as follows. First, we recall some basics about MDDs, MDD propagators and the dispersion constraint. Then, we introduce simple models using MDDs for modelling the `dispersion` constraint with a fixed or a variable mean, and we show how we can combine them in order to obtain only one MDD. Next, we present the `PMF` constraint and show how it can be represented by an MDD and filtered by MDD propagators. We give some experiments supporting our approach. At last, we conclude.

2 Preliminaries

Multi-valued decision diagram (MDD) An MDD is a data-structure representing discrete functions. It is a multiple-valued extension of BDDs [6]. An MDD, as used in CP [7,13,2,10,11,4,9], is a rooted directed acyclic graph (DAG) used to represent some multi-valued function $f : \{0...d-1\}^r \rightarrow \{true, false\}$, based on a given integer d. Given the r input variables, the DAG representation is designed to contain $r+1$ layers of nodes, such that each variable is represented at a specific layer of the graph. Each node on a given layer has at most d outgoing arcs to nodes in the next layer of the graph. Each arc is labeled by its corresponding integer. The arc (u, v, a) is from node u to node v and labeled by a. All outgoing arcs of the layer r reach the true terminal node (the false terminal node is typically omitted). There is an equivalence between $f(a_1, ..., a_r) = true$ and the existence of a path from the root node to the true terminal node whose arcs are labeled $a_1, ..., a_r$.

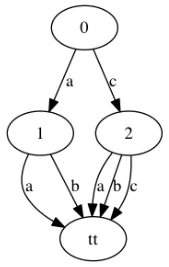

Fig. 1. An MDD of the tuple set {(a,a),(a,b),(c,a),(c,b),(c,c)}. For each tuple, there is a path from the root node (node 0) to the terminal node (node tt) whose arcs are labeled by the tuple values.

MDD of a constraint. Let C be a constraint defined on $X(C)$. The MDD associated with C, denoted by MDD(C), is an MDD which models the set of tuples satisfying C. More precisely, MDD(C) is defined on $X(C)$, such that layer i corresponds to the variable x_i and the labels of arcs of the layer i correspond to values of x_i, and a path of MDD(C) where a_i is the label of layer i corresponds to a tuple $(a_1, ..., a_r)$ on $X(C)$.
Consistency with MDD(C)***.*** A value a of the variable x is valid iff $a \in D(x)$, where $D(x)$ is the possible values of the variable x. An arc (u, v, a) at layer i is valid iff $a \in D(x_i)$. A path is valid iff all its arcs are valid.
Let $path^s_{tt}$(MDD(C)) be the set of paths from s, the root node, to tt in MDD(C). The value $a \in D(x_i)$ is consistent with MDD(C) iff there is a valid path in $path^s_{tt}$(MDD(C)) which contains an arc at layer i labeled by a.
MDD propagator. An MDD propagator associated with a constraint C is an algorithm which removes some inconsistent values of $X(C)$. The MDD propagator establishes arc consistency of C if and only if it removes all inconsistent values with MDD(C). This means that it ensures that there is a valid path from the root to the true terminal node in MDD(C) if and only if the corresponding tuple is allowed by C and valid.
Cost-MDD. A cost-MDD is an MDD whose arcs have an additional information: the cost c of the arc. That is, an arc is a 4-uplet $e = (u, v, a, c)$, where u is the head, v the tail, a the label and c the cost. Let M be a cost-MDD and p be a path of M. The cost of p is denoted by $\gamma(p)$ and is equal to the sum of the costs of the arcs it contains.
Cost-MDD of a constraint [9]. Let C be a constraint and f_C be a function associating a cost with each value of each variable of $X(C)$. The cost-MDD of C and f_C is denoted by cost-MDD(C, f_C) and is MDD(C) whose the cost of an arc labeled by a at layer i

is $f_C(x_i, a)$.
Cost-MDD propagator [8,15]. A cost-MDD propagator associated with C, f_C, a value H, and a symbol $\prec$ (which can be $\leq$ or $\geq$) is an MDD propagator on MDD(C) which ensures that for each path p of cost-MDD(C, f_C) we have $\gamma(p) \prec H$. A cost-MDD propagator establishes arc consistency of C iff each arc of cost-MDD(C) belongs to p a valid path of $path^s_{tt}$(cost-MDD(C)) with $\gamma(p) \prec H$.
MDD of a Generic Sum Constraint [25]. We define the generic sum constraint $\Sigma_{f,[a,b]}(X)$ which is equivalent to $a \leq \sum_{x_i \in X} f(x_i) \leq b$, where f is a non negative function. The MDD of the constraint $\sum_{x_i \in X} f(x_i)$ is defined as follows. For the layer i, there are as many nodes as there are values of $\sum_{k=1}^{i} f(x_k)$. Each node is associated with such a value. A node n_p at layer i associated with value v_p is linked to a node n_q at layer $i+1$ associated with value v_q if and only if $v_q = v_p + f(a_i)$ with $a_i \in D(x_i)$. Then, only values v of the layer $|X|$ with $a \leq v \leq b$ are linked to tt. The reduction operation is applied after the definition and delete invalid nodes [14]. The construction can be accelerated by removing states that are greater than b or that will not permit to reach a. For convenience, $\Sigma_{id,[\alpha,\alpha]}(X)$ is denoted by $\Sigma_\alpha(X)$.

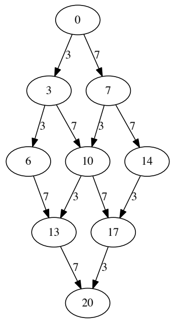

Fig. 2. MDD of the $\sum x_i = n\mu$ constraint

Figure 2 is an example of MDD$(\Sigma_{20}(X))$ with $\{3, 7\}$ as domains. Since f_C in non negative, the number of nodes at each layer of MDD$(\Sigma_{f,[a,b]}(X))$ is bounded by b.

Dispersion Constraint [17]. Given $X = \{x_1, ..., x_n\}$, a set of finite-domain integer variables, μ and Δ, bounded-domain variables and p a natural number. The constraint DISPERSION(X, μ, Δ, p) states that the collection of values taken by the variables of X exhibits an arithmetic mean $\mu = \sum_{i=1}^{n} x_i$ and a deviation $\Delta = \sum_{i=1}^{n} |x_i - \mu|^p$.

The `deviation` constraint is a `dispersion` constraint with $p = 1$ and a `spread` constraint is a `dispersion` constraint with $p = 2$.

The main complexity of the `dispersion` constraint is the relation between μ and Δ variables, because μ is defined from the X variables, and Δ is defined from X and from μ. So, some information is lost when these two definitions are considered separately. However, when μ is assigned, the problem becomes simpler because we can independently consider the definitions of μ and $\varDelta$. Therefore, we propose to study some models depending on the fact that μ is fixed or not.

3 Dispersion Constraint with fixed mean

Arc consistency for the X variables has been established by Pesant [17], who proposed an ad-hoc dynamic programming propagator for this constraint. However, it exists a simpler method avoiding such problems of ad-hoc algorithms: we define a cost-MDD from μ and $\varDelta$ and obtain a propagator having the same complexity.

3.1 MDD on μ and $\varDelta$ as cost

The mean μ is defined as a sum constraint. Since μ is fixed, we propose to use the cost-MDD of the constraint $\sum x_i = n\mu$ and the cost function defined by $\varDelta$.

The constraint $\sum x_i = n\mu$ can be represented by MDD($\varSigma_{n\mu}(X)$).

$\varDelta$ as cost. We represent the dispersion constraint by cost-MDD($\varSigma_{\mu}(X), \varDelta$). There are two possible ways to deal with the boundaries of $\varDelta$. Either we define two cost-MDD propagators on cost-MDD($\varSigma_{\mu}(X), \varDelta$), one with a and $\geq$, and one with b and $\leq$; or we define only one cost-MDD propagator on cost-MDD($\varSigma_{\mu}(X), \varDelta$) which integrates the costs at the same time as proposed by Hoda et al [11].

These methods are simpler than Pesant's algorithm because they do not require to develop any new algorithm. If we use an efficient algorithm [15] for maintaining arc consistency for cost-MDDs then we obtain the the same worst case complexity as Pesant's algorithm but better result in practice.

3.2 MDD on μ intersected with MDD on $\varDelta$

Since μ is fixed, then the definition of $\varDelta$ corresponds to a generic sum as previously defined. Thus, the dispersion constraint can be model by defining the MDD of $\varSigma_{\mu}(X)$ and the MDD of $\varSigma_{\varDelta,[\underline{\varDelta},\overline{\varDelta}]}(X)$ and then by intersecting them. Replacing a cost-MDD by the intersection of two MDDs may strongly improve the computational results [15]. In addition, we can intersect the resulting MDD with some other MDDs in order to combine more the constraints. This method is the first method establishing arc consistency for both μ and $\varDelta$. The drawback is the possible size of the intersection.

With similar models we can also give an efficient implementation of the `Student's t-test` constraint and close the open question of Rossi et al.[19].

4 Dispersion Constraint with variable mean

In order to deal with a variable mean, we can consider all acceptable values for $n\mu$, that is the integers in $[n\lfloor\underline{\mu}\rfloor, n\lceil\overline{\mu}\rceil]$, and for each value we separately apply the previous models for the fixed mean. Unfortunately, this often leads to a large number of constraints. Therefore it is difficult to use this approach in practice. In addition, note that there is no advantage in making the union of these constraints because they are independent.

Thus, we propose another model using the probability mass function.

5 Probability Mass Function (PMF) constraint

In this section we define the `PMF` constraint which aims at respecting a variable mean and avoiding outliers according to a statistical law given by a probability mass function.

Given a discrete random variable X taking values in $X = \{v_1, ...v_m\}$ its probability mass function P: $X \rightarrow [0, 1]$ is defined as $P(v_i) = Pr[X = v_i]$ and satisfies the following condition: $P(v_i) \geq 0$ and $\sum_{i=1}^{m} P(v_i) = 1$

The PMF gives for each value v, $P(v)$ the probability that v is taken. Let f_P be a PMF and consider a set of variables independent from a probabilistic point of view and associated with f_P that specifies probabilities for their values. Since the variables are independent, we can define the probability of an assignment of all the variables (i.e. a tuple) as the product of the probabilities of the assigned values. Then, in order to avoid outliers we can constrain this probability to be in a given interval.

Definition 1 *Given a set of finite-domain integer variables $X = \{x_1, x_2, ..., x_n\}$ that are independent from a probabilistic point of view, a probability mass function f_P, a bounded variable μ (not necessarily fixed), a minimum probability P_{min} and a maximum probability P_{max}. The constraint* $\mathrm{PMF}(X, f_P, \mu, P_{min}, P_{max})$ *states that the probabilities of the values taken by the variables of X is specified by f_P, the collection of values taken by the variables of X exhibits an arithmetic mean μ and that $\Pi_{x_i \in X} x_i$ the probability of any allowed tuple satisfies $P_{min} \leq \Pi_{x_i \in X} f_P(x_i) \leq P_{max}$.*

This constraint can be represented by cost-MDD$(\Sigma_{id,[\underline{\mu},\overline{\mu}]}(X), logP)$ where $logP$ is the logarithm of the PMF that is $logP(x) = \log(f_P(x))$. We take the logarithm because in this way we have a sum function instead of a product function: $\log(\Pi f_P(x_i)) = \sum \log(f_P(x)) = \sum logP(x)$. Then, we define a cost-MDD propagator on cost-MDD$(\Sigma_{id,[\underline{\mu},\overline{\mu}](X),logP}$ with $\log(P_{min})$ and $\geq$ and with $\log(P_{max})$ and $\leq$.

6 Experiments

The experiments were run on a macbook pro (2013) Intel core i7 2.3GHz with 8 Go. The constraint solver used is or-tools. MDD4R [13] is used as MDD propagator and cost-MDD4R as cost-MDD propagator [15].

The data come from a real life application: the geomodeling of a petroleum reservoir [12]. The problem is quite complex and we consider here only a subpart. Given a seismic image we want to find the velocities. Velocities values are represented by a probability mass function (PMF) on the model space. Velocities are discrete values of variables. For each cell c_{ij} of the reservoir, the seismic image gives a value s_{ij} and the from the given seismic wavelet (α_k) we define a sum constraint $\sum_{k=1}^{22} \alpha_k log(x_{i-11+k-1}j) = s_{ij}$. Locally, that is for each sum, we have to avoid outliers w.r.t. the PMF for the velocities. Globally we can use the classical dispersion constraint. The problem is huge (millions of variables) so we consider here only a very small part.

The first experiment involves 22 variables and a constraint C_α: $\sum_{i=1}^{n} \alpha_i x_i = I$, where I is an tight interval (i.e. a value with an error variation). C_α is represented by $mdd_\alpha = \text{MDD}(\Sigma_{a_i,I}(X))$ where $a_i(x_i) = \alpha_i x_i$.

First, we impose that the variables have to be distributed with respect to a normal distribution with μ, a fixed mean.

$M_{\sigma<,\sigma>}$ represents the model of Section 3.2: one cost-MDD propagator on $mdd_\mu = \text{cost-MDD}(\Sigma_{n\mu}(X), \sigma)$ with σ and $\leq$ and one with σ and $\geq$. This model is similar to Pesant's model.

M_{GCC} involves a GCC constraint [18] where the cardinalities are extracted from the probability mass function.

$M_{\mu\cap\sigma}$ represents the mean constraint by $mdd_\sigma = \text{MDD}(\Sigma_{n\mu}(X))$. It represents the sigma constraint by the $\text{MDD}(\Sigma_\sigma(X))$. Then the two MDDs are intersected. An MDD propagator is used on this MDD, named $mdd_{\mu\sigma}$. See Section 3.3.

$M_{\mu\cap\sigma\cap\alpha}$ intersects mdd_α, the MDD of the constraint C_α, with $mdd_{\mu\sigma}$ the previous MDD to obtain mdd_{sol}. In this case, all constraints are combined.

Then, we consider a `PMF` constraint and that μ is variable:

M_{log}. We define a cost-MDD propagator on $mdd_{I_\mu} = \text{cost-MDD}(\Sigma_{id,[\underline{\mu},\overline{\mu}]}(X), logP)$ with $\log(P_{min})$ and $\geq$ and with $\log(P_{max})$ and $\leq$. See Section 5.

$M_{log\cap\alpha}$. We define $mdd_{I_{log}} = \text{MDD}(\Sigma_{logP,I_{log}}(X))$ and we intersect it with mdd_{I_μ}. Then, we intersect it with mdd_α, the MDD of C_α, to obtain $mdd_{log\alpha}$.

Table 3 shows the result of these experiments. As we can see when the problem involves many solutions, all the methods perform well (excepted M_{GCC}). We can see that an advantage of the intersection methods is that they contain all the solutions of problem. Table 4 shows the different sizes of the MDDs.

		Fixed μ				Variable μ	
Sat?	#sol	$M_{\sigma<,\sigma>}$	M_{GCC}	$M_{\mu\cap\sigma}$	$M_{\mu\cap\sigma\cap\alpha}$	M_{log}	$M_{log\cap\alpha}$
Sat	Build	50	31	138	2,203	34	317,921
	10 sol	14	T-O	16	0	14	0
	All sol	T-O	T-O	T-O	0	T-O	0
UnSat	Build	55	28	121	151	37	133,752
	10 sol	T-O	T-O	T-O	0	T-O	0
	All sol	T-O	T-O	T-O	0	T-O	0

Fig. 3. Comparison solving times (in ms) of models. 0 means that this is immediate. T-O indicates a time-out of 500s.

Random instances. The intersection methods $M_{\mu\cap\sigma}$ and $M_{\mu\cap\sigma\cap\alpha}$ have been tested on random bigger instances. Table 5 and 6 gives some results showing how this method scales with the number of variables. In the first line, the couple is #var/#val. Times are in ms. Experiments of Table 5 set $0 < \sigma < 4n$ for having a delta depending on the number of variables like in [17], whereas experiments of Table 6 impose $100 < \sigma < 400$, these numbers come from our real world problem.

These experiments show that the $M_{\mu\cap\sigma}$ model can often be a good trade-off between space and time. Using the lower bound of the expected size of the MDD [15], we

		Fixed μ					Variable μ		
Sat?	N/A	mdd_α	mdd_μ	mdd_σ	$mdd_{\mu\sigma}$	mdd_{sol}	mdd_{I_μ}	$mdd_{I_{log}}$	$mdd_{log\alpha}$
Sat	#nodes	3	3	5	67	521	2	18	24,062
Sat	#arcs	44	27	55	660	4,364	30	268	341,555
UnSat	#nodes	3	2	5	67	0	2	18	0
UnSat	#arcs	46	27	55	660	0	30	268	0

Fig. 4. Comparison of MDD sizes (in thousands) of different models. 0 means that the MDD is empty.

can estimate and decide if it is possible to process $M_{\mu\cap\sigma\cap\alpha}$. The last two columns of Table 5 show that it is not always possible to build such an intersection.

$0 < \sigma < 4n$									
Method	n/d	20/20	30/20	40/30	40/40	50/40	50/50	100/40	100/100
$M_{\mu\cap\sigma}$	T(ms)	26	132	391	401	848	875	12,780	14,582
	#nodes	18	63	153	1578	306	311	2,285	2,532
	#arcs	198	808	2,308	2,427	5,196	5,354	53,757	62,057
$M_{\mu\cap\sigma\cap\alpha}$	T(ms)	561	3,084	11,864	10,789	58,092	60,513	M-O	M-O
	#nodes	163	764	0	0	0	0	M-O	M-O
	#arcs	1,788	8,416	0	0	0	0	M-O	M-O

Fig. 5. Time (in ms) and size (in thousands) of the MDDs of models $M_{\mu\cap\sigma}$ and $M_{\mu\cap\sigma\cap\alpha}$. 0 means that the MDD is empty. M-O means memory-out.

$100 < \sigma < 400$					
Method	n/d	20/20	30/20	40/30	40/40
$M_{\mu\cap\sigma}$	T(ms)	162	333	586	602
	#nodes	81	184	326	338
	#arcs	823	1,865	3,329	3,479
$M_{\mu\cap\sigma\cap\alpha}$	T(ms)	2,663	10,379	21,063	26,393
	#nodes	1,098	2,555	35	0
	#arcs	11,166	23,764	151	0

Fig. 6. Time (in ms) and size (in thousands) of the MDDs of models $M_{\mu\cap\sigma}$ and $M_{\mu\cap\sigma\cap\alpha}$. $M_{\mu\cap\sigma\cap\alpha}$ is empty because there is no solution.

7 Conclusion

We have shown that modeling constraints by MDDs has several advantages in practice. It avoids to develop ad-hoc algorithms, gives competitive results and leads to efficient combination of constraints outperforming the other approaches. We have emphasized our approach on statistical constrains including the new PMF constraint we proposed.

References

1. Problem 30 of CSPLIB. (`www.csplib.org`).
2. Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.
3. N. Beldiceanu, M. Carlsson, S. Demassey, and T. Petit. Global constraint catalog: Past, present and future. *Constraints*, 12(1):21–62, 2007.
4. David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, pages 20–35, 2011.
5. Christian Bessiere, Emmanuel Hebrard, George Katsirelos, Zeynep Kiziltan, Émilie Picard-Cantin, Claude-Guy Quimper, and Toby Walsh. The balance constraint family. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2014.
6. Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C35(8):677–691, 1986.
7. K. Cheng and R. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15:265–304, 2010.
8. Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.
9. G. Gange, P. Stuckey, and Radoslaw Szymanek. Mdd propagators with explanation. *Constraints*, 16:407–429, 2011.
10. Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.
11. Samid Hoda, Willem Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *CP*, pages 266–280, 2010.
12. Wayne D. Pennington. Reservoir geophysics. 66(1), 2001.
13. G. Perez and J-C. Régin. Improving GAC-4 for table and MDD constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 606–621, 2014.
14. G. Perez and J-C. Régin. Efficient operations on mdds for building constraint programming models. In *International Joint Conference on Artificial Intelligence, IJCAI-15*, pages 374–380, Argentina, 2015.
15. G. Perez and J-C. Régin. Soft and cost mdd propagators. In *Proc. AAAI'17*, 2017.
16. G. Pesant and J-C. Régin. Spread: A balancing constraint based on statistics. In *CP'05*, pages 460–474, 2005.
17. Gilles Pesant. Achieving domain consistency and counting solutions for dispersion constraints. *INFORMS Journal on Computing*, 27(4):690–703, 2015.
18. J-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI-96*, pages 209–215, Portland, Oregon, 1996.
19. Roberto Rossi, Steven David Prestwich, and S. Armagan Tarim. Statistical constraints. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 777–782, 2014.
20. Pierre Roy, Guillaume Perez, Jean-Charles Régin, Alexandre Papadopoulos, François Pachet, and Marco Marchini. Enforcing structure on temporal sequences: The allen constraint. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 786–801, 2016.
21. P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. In *CPAIOR'07*, pages 260–274, 2007.

22. P. Schaus, Y. Deville, P. Dupont, and J-C. RÃ©gin. *Future and Trends of Constraint Programming*, chapter Simplification and extension of the SPREAD Constraint, pages 95–99. ISTE, 2007.
23. P. Schaus and J-C. Régin. Bound-consistent spread constraint. 2(3), 2014.
24. Pierre Schaus, Yves Deville, and Pierre Dupont. Bound-consistent deviation constraint. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, pages 620–634, 2007.
25. M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. In *CPAIOR'01*, 2001.