Proceedings of the Federated Conference on Computer Science and Information Systems pp. 31–34
DOI: 10.15439/2020F169
ISSN 2300-5963 ACSIS, Vol. 21

Computing Duals of Finite Gödel Algebras

Pietro Codara, Gabriele Maurina and Diego Valota
Dipartimento di Informatica, Università degli Studi di Milano, Italy
codara@di.unimi.it, gabriele.maurina@studenti.unimi.it, valota@di.unimi.it

***Abstract*—We introduce an algorithm that computes and counts the duals of finite Gödel-Dummett algebras of $k \geq 1$ elements. The computational cost of our algorithm depends on the factorization of k, nevertheless a Python implementation is sufficiently fast to compute the results for very large values of k.**

I. Introduction

Mathematical Fuzzy Logics (MFL) interpret predicates in truth-degrees ranging over the unitary real interval $[0, 1]$. It has been argued that this is a valid approach to deal with the inherent *vagueness* of terms in human languages [1]. In his seminal book [2], Hájek introduced a family of many-valued fuzzy logical systems where conjunction and implication connectives are modeled by continuous *t-norms* [1] and their residua, respectively. One of the three main fuzzy logics in Hájek's framework is Gödel-Dummett logic $\mathcal{G}$, whose conjunction is modeled by the *minimum*. Gödel-Dummett logic is a non-classical logic whose studies date back to Gödel [3] and Dummett [4]. Note that $\mathcal{G}$ can be obtained by adding the prelinearity axiom to *Intuitionistic logic*.

The algebraic counterpart of Gödel-Dummett logic is the variety of Gödel algebras $\mathbb{G}$. In the study of the algebraic semantics of non-classical logics, the notion of free algebra is of particular importance. This is due to the well-known isomorphism between free algebras and *Lindenbaum algebras* of logically equivalent formulas in a given logic. One can find in the literature several methods to obtain the order structures and cardinalities of free Gödel algebras. In 1969, Horn [5] obtained a recurrence formula to compute the cardinalities of free k-generated Gödel algebras, for any $k \in \mathbb{N}^+$. Another solution to this problem can be achieved by restating the Horn's recurrence in terms of finite forests [6].

A related counting problem is the *fine spectrum* problem [7], which aim to find the number of non-isomorphic algebras of cardinality k in a given variety. In [8], the author introduces a method to generate duals of finite Gödel algebras of a given cardinality and a recurrence relation to count the number of such structures, solving in this way the fine spectrum problem for $\mathbb{G}$. Such a result is obtained by exploiting the relation between finite forests and finite Gödel algebras.

In this paper, we build on [8] to obtain an algorithm that accepts a positive integer k as input and returns the number of non-isomorphic k-elements Gödel algebras. Moreover, we propose a Python implementation of such algorithm which is also able to generate the dual structures of the algebras. Finally, we compare the execution times of our algorithm with those obtained using Mace4 [9], a general purpose computer algebra system used to generate finite models. Besides being an interesting theoretical problem, the generation of finite algebras has also important applications in *automated reasoning* [10]. Indeed, such procedures can be used to find countermodels of logical formulas.

[1] A t-norm is an associative, commutative and monotone function $\odot\colon [0,1]^2 \to [0,1]$, where 1 is the neutral element.

II. Gödel Algebras, Forests and Recurrence

We assume that the reader is acquainted with many-valued logics in Hájek's sense and with their algebraic semantics. We refer the reader to [2] for any unexplained notion. Throughout the paper we use the same symbols for logic's connectives and their algebraic interpretations.

Hájek's logic $\mathcal{BL}$ is the logic of all continuous t-norms and their residua, built over the language $\{\odot, \wedge, \vee, \to, \neg, \bot, \top\}$. The algebraic semantics of $\mathcal{BL}$ is given by the variety $\mathbb{BL}$ of BL algebras, that is, prelinear, divisible, commutative, bounded, integral, residuated lattices [2]. A *BL algebra* is an algebra $\mathbf{A} = \langle A, \wedge, \vee, \odot, \to, \bot, \top\rangle$ of type $(2, 2, 2, 2, 0, 0)$ such that $\langle A, \wedge, \vee, \bot, \top\rangle$ is a bounded lattice, with top $\top$ and bottom $\bot$, $\langle A, \odot, \top\rangle$ is a commutative monoid, satisfying the *residuation* equivalence, $x \odot y \leq z$ if and only if $x \leq y \to z$, the *prelinearity* equation $(x \to y) \vee (y \to x) = \top$, and *divisibility* $x \odot (x \to y) = x \wedge y$. Notice that divisibility implies that the lattice structure is distributive. A BL algebra satisfying $x \wedge y = x \odot y$ is called *Gödel algebra*. Hence, the variety of Gödel algebras $\mathbb{G}$ is a subvariety of $\mathbb{BL}$ [2].

Let $\mathbf{A}$ be a Gödel algebra, a *filter* of $\mathbf{A}$ is a non-empty subset p of A such that for all $y \in A$, if there is x in p such that $x \leq y$ then $y \in p$, and $x \wedge y \in p$ for all $x, y \in p$. We call *propers* the filters p such that $p \neq A$. A proper filter p of $\mathbf{A}$ is said to be *prime* when for all $x, y \in A$, either $x \to y \in p$ or $y \to x \in p$. The set of all prime filters $\mathsf{Spec}(\mathbf{A})$ of $\mathbf{A}$ ordered by reverse inclusion is called the *prime spectrum* of $\mathbf{A}$. When $\mathbf{A}$ is finite, each prime filter p of $\mathbf{A}$ is generated by a join-irreducible element a as $p = \{b \in \mathbf{A} \mid a \leq b\}$. On the other hand, each join-irreducible element of $\mathbf{A}$ singly generates a prime filter of $\mathbf{A}$. Hence, $\mathsf{Spec}(\mathbf{A})$ is isomorphic with the poset of the join-irreducible elements of $\mathbf{A}$. See Fig. 1 as an example where $\mathbf{A}$ is the free 1-generated Gödel algebra. A *forest* F is a poset such that the downset $\downarrow q$ of every $q \in F$ is a *chain*, that is $\downarrow q$ is totally ordered. A forest with a bottom element is called a *tree*. Such bottom element is called the *root* of the tree. A *subforest* of a forest F is the downset of some $Q \subseteq F$. Finite forests and open maps form a category FF. Given two forests F, F' we denote $F \sqcup F'$ the disjoint union

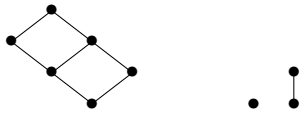

Fig. 1. The free Gödel Algebra on one generator and its prime spectrum.

of F and F'. Since FF is a category, disjoint unions are in fact coproducts in FF. For our purposes, we need to introduce an additional operation, called the *lifting* of a forest F, denoted by $F_\perp$, and obtained by adding a common root to all trees in F. Clearly, for every forest F, $F_\perp$ is a tree. A complete account with proofs on the operations in FF can be found in [11]. The prime spectrum of a finite Gödel algebra forms a forest, as shown by Horn in [12]. We can also obtain a Gödel algebra from a finite forest F in the following way. Let $\mathsf{Sub}(F)$ be the finite set of subforests of F. We equip $\mathsf{Sub}(F)$ with the structure of a Gödel algebra $\langle \mathsf{Sub}(F), \cap, \cup, \to, \emptyset, F\rangle$, where $F' \to F'' = F \setminus \uparrow (F' \setminus F'')$, for all $F', F'' \in \mathsf{Sub}(F)$. In this way, we can obtain the following isomorphisms $\mathsf{Spec}(\mathsf{Sub}(F)) \cong F$ and $\mathsf{Sub}(\mathsf{Spec}(\mathbf{A})) \cong \mathbf{A}$, for a given finite Gödel algebra $\mathbf{A}$. When $F \cong \mathsf{Spec}(\mathbf{A})$ and $\mathbf{A} \cong \mathsf{Sub}(F)$, we call such F the *dual* of $\mathbf{A}$, while $\mathbf{A}$ is the *primal* of F. It is also possible to define Sub and Spec over maps, making them functors acting on the category of finite forests and open maps FF and the category of finite Gödel algebras and their homomorphisms, extending in this way the above equivalence to a full categorical duality. These constructions go beyond the scope of the present paper, and we refer the interested reader to [13] for further details. With this machinery in mind we are ready to formally define the fine spectrum problem for $\mathbb{G}$.

Problem: $\text{Fine}_{\mathbb{G}}(k)$
Input: $k \in \mathbb{N}^+$
Output: $m \in \mathbb{N}^+$ such that $m = |\{[\mathbf{A}] \in \mathbb{G} \mid k = |A|\}|$, where $[\mathbf{A}]$ is the class of finite Gödel algebras isomorphic with $\mathbf{A}$.

To solve this problem, we summarize the recurrence relation introduced in [8] to generate duals of finite Gödel algebras. This procedure is based upon the concept of *multiplicative partition* of a positive integer k [14], that is

$$\mathsf{MP}(k) := \{(n_1, \ldots, n_t) \mid k = n_1 \times \cdots \times n_t, n_1 \leq \cdots \leq n_t, t > 1\}.$$

Each t-uple in $\mathsf{MP}(k)$ is composed of natural numbers whose product is equal to k. Note that the usual definition of multiplicative partition of k includes (k), while our definition does not. We define recursively the following sets of forests, which are fundamental for our work:

$$H_1 = \{\emptyset\} \tag{H_1}$$
$$H_k = P_k \cup Z_k \tag{H_k}$$
$$P_k = \{F_\perp | F \in H_{k-1}\} \tag{P_k}$$
$$Z_k = \{F_1 \sqcup \cdots \sqcup F_t \mid F_1 \in P_{n_1}, \ldots, F_t \in P_{n_t}, (n_1, \ldots, n_t) \in \mathsf{MP}(k)\} \tag{Z_k}$$

Theorem 1 ([8]): $F \in H_k$ if and only if $|\mathsf{Sub}(F)| = k$. Moreover, $\langle \mathsf{Sub}(F), \cap, \cup, \to, \emptyset, F\rangle$ is isomorphic to a k-elements Gödel algebra $\mathbf{A}$ such that $\mathsf{Spec}(\mathbf{A}) \cong F$.

Thanks to this recursive definition of the set of duals of k-elements Gödel algebras H_k, we are also able to compute the cardinality of H_k, that is the fine spectrum of $\mathbb{G}$.

Corollary 1 ([8]): $\text{Fine}_{\mathbb{G}}(k) = f(k) + pr(k) \times g(k)$ with,

$$pr(k) = \begin{cases} 0 & \text{if } k \text{ is prime;} \\ 1 & \text{otherwise.} \end{cases} \tag{pr}$$
$$f(1) = 1 \tag{f_1}$$
$$f(k) = \text{Fine}_{\mathbb{G}}(k-1) \tag{f_k}$$
$$g(k) = \sum_{(n_1,\ldots,n_t)\in \mathsf{MP}(k)} f(n_1) \times \cdots \times f(n_t) \tag{g_k}$$

The sequence of numbers generated by $\text{Fine}_{\mathbb{G}}(k)$ was already contained in the *On-Line Encyclopedia of Integer Sequences* as sequence A130841, that counts the number of ways to express an integer as a sum of so-called *oterms*. In [8], the author shows that oterms are a syntactic description of finite forests. In the next section we show how to obtain a fast algorithm implementing $\text{Fine}_{\mathbb{G}}(k)$, able to compute such values for very large k.

III. Algorithm

A naive implementation of the recurrences of the previous Section leads to recursive procedure that runs at exponential costs by recursively calling $\text{Fine}_{\mathbb{G}}(m)$ for every instance of m occurring in $\mathsf{ML}(i)$ for $1 \leq i \leq k$. We rewrite the recurrence in Corollary 1 in a more compact way:

$$\text{Fine}(k) = \text{Fine}(k-1) + \\ + pr(k) \times \sum_{(n_1,\ldots,n_t)\in \mathsf{MP}(k)} \text{Fine}(n_1 - 1) \times \cdots \times \text{Fine}(n_t - 1).$$

We can now obtain a more efficient algorithm just by applying *dynamic programming* [15] to this recurrence. Indeed, for computing $\text{Fine}(k)$ we need (potentially all) the values $\text{Fine}(i)$ for $1 \leq i \leq k$. So, we compute $\text{Fine}(i)$ for every $i \in (1 \leq 2 \leq 3 \leq \cdots \leq k)$ following the natural integers order and storing the computed values in a k-element vector $Fine$. The algorithm is outlined in Algorithm 1, and it assumes that there exists a function `multpart` that receives a $m \in \mathbb{N}^+$ and returns the set of multiplicative partitions $\mathsf{ML}(m)$ when m is composite, otherwise when m is prime `multpart` returns the empty set. We show in Section IV how to implement such a function using Python libraries.

The number of multiplicative partitions $\mathsf{MP}(k)$ is less than or equal to $\frac{k}{log\ k}$ for every $k \in \mathbb{N}^+$ such that $k \neq 144$ [16]. Hence, it is straightforward to see that for every $k \in \mathbb{N}^+$ such that $k \neq 144$, the inner `for` cycle (Line 8 in Algorithm 1) makes $O(\frac{k}{log\ k})$ steps to compute G. Then, the computation of $Fine[k]$ need necessarily $(k \times \frac{k}{log\ k})$ steps, but this is not sufficient. This bound cannot be fruitfully used to study the cost of the full algorithm. Indeed to compute the multiplicative partitions $\mathsf{MP}(n)$ of each n in $\{1, \ldots, k\}$, we need to factorize

Algorithm 1 A function calculating $\mathrm{Fine}_{\mathbb{G}}(k)$

```
    Fine[1] ← 1;
 2: if k == 1 then
        return Fine[1];
 4: end if
    for n = 2; n ≤ k; n = n + 1 do
 6:     M ← multpart(n)
        G ← 0
 8:     for each (n_1, ..., n_t) ∈ M do
            G ← G + (Fine[n_1 − 1] × ··· × Fine[n_t − 1])
10:     end for
        Fine[n] ← Fine[n − 1] + G;
12: end for
    return Fine[k];
```

n in the function `multpart`(n) (Line 6 in Algorithm 1). By now, no efficient integer factorization algorithm can be found in literature. In the next Section we see that our strategy relies on the *prime factorization* of n, and this is essentially an exponential procedure.

IV. Implementation

Algorithm 1 has been implemented in Python using SymPy library [17]. The main issue in the implementation is to find an efficient way to obtain the set of t-uples $\mathsf{MP}(k)$ for a given $k \in \mathbb{N}^+$. We have used the `sympy.factorint` method to obtain the list of prime factors $fact(k)$ of k. Such method is particularly useful to our purpose because it uses different algorithms in the library, selecting the most efficient one according to the size of k. Now it is easy to realize that each t-uple $\mathsf{MP}(k)$ can be obtained from the multiset partitions of $fact(k)$. Hence, we have used the method `multiset_partitions` in `sympy.utilities.iterables` to create exactly the list of t-uples corresponding to elements of $\mathsf{MP}(k) \cup (k)$. As mentioned above we don't need the one-block partition (k), so our code just ignore it.

By slightly modifying Algorithm 1, we have also implemented functions to generate the full set of forests H_k by building trees using parenthesis representation, and to produce images of such forests using GraphViz library [18].

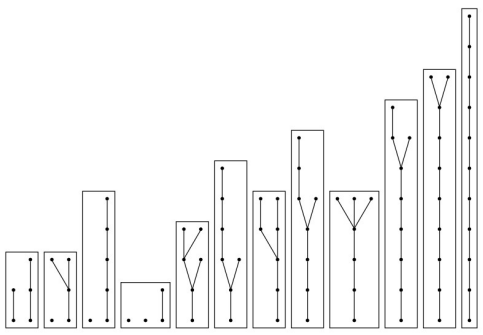

Fig. 2. The set of forests H_{12} generated by our Python code.

Example 4.1: Let $k = 12$. Then, $fact(k) = \{2, 3\}$ and applying `multiset_partitions` to $fact(12)$ we obtain $[[[2, 2, 3]], [[2, 2], [3]], [[2, 3], [2]], [[2], [2], [3]]]$. From this list it is easy to obtain the t-uples in $\mathsf{MP}(12) \cup (12)$ by multiplying elements in the same blocks, that is $[[12], [4, 3], [6, 2], [2, 2, 3]]$. Since by definition $(12) \notin \mathsf{MP}(12)$, the code skip the first partition $[12]$. Fig. 2 is the depiction of H_{12} produced by our program. As an instance, the parenthesis representation of the first and second forest on the left in Fig. 2 are $[[]]$, $[[[]]]$ and $[]$, $[[[], []]]$ respectively.

The following results have been obtained on a GNU/Linux Debian 4.9.130-2 system with an Intel Core i7-5500U CPU and 8GB of RAM. The Python implementation, together with the Mace4 input and output files, can be downloaded from https://homes.di.unimi.it/~valota/code/finegodel.zip.

To study the effectiveness of our implementation we have used Mace4 [9] to compute the number of finite Gödel algebras. Mace4 produces the algebraic structures on output files, then we need to run two tools: `interpformat` to convert the outputs in a readable format, and `isofilter` to get rid of isomorphic copies of our algebraic structures. Running times, calculated with the Debian GNU/Linux command-line tool `time` are summarized in Table I. The $\cdot$ symbol indicates that the instance is not meaningful. In fact, Algorithm 1 computes only the structure of forests in H_i for every $i \in \{2, \ldots, k\}$.

TABLE I
Running times to compute finite Gödel algebras (or their duals) of cardinality 2 to 11.

k	Running Times	Mace4	`interpformat` +`isofilter`	Algorithm 1
2 to 9	`real`	0m49.623s	0m5.427s	0m0.024s
	`user`	0m49.272s	0m5.384s	0m0.024s
10	`real`	10m58.814s	0m44.284s	$\cdot$
	`user`	10m56.092s	0m44.148s	$\cdot$
11	`real`	191m27.975s	8m28.547s	$\cdot$
	`user`	190m48.804s	8m26.960s	$\cdot$
2 to 11	`real`	$\cdot$	$\cdot$	0m0.024s
	`user`	$\cdot$	$\cdot$	0m0.020s

The huge increase in computing time when passing from cardinality 10 to cardinality 11 in Mace4 runnings, shows the unsuitability of brute-force approaches. In fact, inspecting (H_k) one realizes that to obtain duals of finite Gödel algebras of cardinality 11, it is sufficient to lift all the forest in H_{10}. To compare, our Python implementation is able to generate the parenthesis representation of forests in H_i from $i = 2$ to $i = 11$, our script takes time `real: 0m0.024s` and `user: 0m0.020s` (last line in Table I). However, for counting purposes the script runs very fast, as testified by the performances summarized in Table II.

TABLE II
RUNNING TIMES OF ALGORITHM 1 FOR LARGE VALUES OF k.

k	Running Times	$\text{Fine}_{\mathbb{G}}(k)$
2 to 1000	`real`: 0m0.558s `user`: 0m0.556s	$\text{Fine}_{\mathbb{G}}(1000) =$ 3316527416
2 to 5000	`real`: 0m24.486s `user`: 0m24.476s	$\text{Fine}_{\mathbb{G}}(5000) =$ 772140728313177
2 to 10000	`real`: 2m2.602s `user`: 2m2.580s	$\text{Fine}_{\mathbb{G}}(10000) =$ 416184590541943029

V. CONCLUSIONS AND FURTHER WORKS

We should point out that Mace4 generates full models with tables for each algebraic operation, while our software only generates the order structure of the duals of finite algebras. To obtain the algebraic structures, one can appeal to Theorem 1 and obtain from each forest F produced by the Python script, the structure of the corresponding Gödel algebra by considering every subforest in $\mathsf{Sub}(F)$. Then for instance, the construction of the conjunction operation's table amount to consider set-inclusion among the subforests. Such functionality is not present in our software and is left as future work. Mace4 is a general-purpose Computer Algebra System. Another interesting approach to compare with our work is contained in [19], where authors introduce a brute-force algorithm with a heuristic test to detect isomorphic lattices, that computes tables operation for classes of residuated lattices. The counting of such structures is a byproduct of their work. They count several finite algebras related to many-valued logics until cardinality 12, including Gödel algebras. Our algorithm is essentially based on the duality between finite forests and open maps, and finite Gödel algebras and their homomorphisms. So, it makes sense to generalize this duality-based approach to other classes of algebras for which combinatorial dualities exist. In particular, we need subforest representations for (at least) finite algebras. Such type of representations can be found in the literature for locally finite subvarieties of MTL algebras [20], such as nilpotent minimum algebras [21], and revised drastic product algebras [22] and their subvarieties: drastic product algebras and EMTL algebras [23]. Another interesting duality for finite $\text{Gödel}_{\triangle}$ algebras is introduced in [24], and in [25] is shown that the dual category of this variety is also dual to the variety of drastic product algebras (studied in [26]). All these varieties are subvarieties of an interesting algebraic variety related to weak negation functions over $[0, 1]$, the variety of WNM algebras. In [27] one can find an extensive study of WNM chains that leads to a combinatorial representation of finitely generated free WNM algebras. Free and finite algebras are closely related. Indeed, every k-generated algebra in a variety $\mathbb{V}$ can be obtained as a quotient of the free k-generated algebra in $\mathbb{V}$. However, different congruences may generate isomorphic quotients. Hence, studies on free and fine spectra can be also used to investigate congruences in varieties. Finally, our algorithm can be used to obtain additional insight on the structure of finite Gödel algebras, helping researchers to find structural properties of such algebraic structures or a bound for the fine spectrum of $\mathbb{G}$.

REFERENCES

[1] N. Smith, "Fuzzy logics in theories of vagueness," in *Handbook of Mathematical Fuzzy Logic. Vol. 3*, P. Cintula, C. Fermüller, and C. Noguera, Eds. College Publications, 2016, vol. 58, pp. 1237–1281.

[2] P. Hájek, *Metamathematics of Fuzzy Logic*, ser. Trends in Logic. Kluwer Academic Publishers, 1998, vol. 4.

[3] K. Gödel, "Zum intuitionistischen Aussagenkalkul," *Anzeiger Akademie der Wissenschaften Wien*, vol. 69, pp. 65–66, 1932.

[4] M. Dummett, "A propositional calculus with denumerable matrix," *J. Symb. Log.*, vol. 24, no. 2, pp. 97–106, 1959.

[5] A. Horn, "Free L-Algebras," *J. Symb. Log.*, vol. 34, no. 3, pp. 475–480, 1969.

[6] O. M. D'Antona and V. Marra, "Computing coproducts of finitely presented Gödel algebras," *Ann. Pure Appl. Logic*, vol. 142, no. 1, pp. 202–211, 2006.

[7] W. Taylor, "The fine spectrum of a variety," *Algebra Universalis*, vol. 5, no. 1, pp. 263–303, 1975.

[8] D. Valota, "Spectra of Gödel Algebras," in *Language, Logic, and Computation. TbiLLC 2017*, ser. Lecture Notes in Computer Science, A. Silva, S. Staton, P. Sutton, and C. Umbach, Eds., 2019, vol. 11456.

[9] W. McCune, "Prover9 and mace4," 2005–2010, `http://www.cs.unm.edu/~mccune/prover9/`.

[10] J. A. Robinson and A. Voronkov, Eds., *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[11] S. Aguzzoli and P. Codara, "Recursive formulas to compute coproducts of finite Gödel algebras and related structures," in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 201–208.

[12] A. Horn, "Logic with Truth Values in a Linearly Ordered Heyting Algebra," *J. Symb. Log.*, vol. 34, no. 3, pp. pp. 395–408, 1969.

[13] S. Aguzzoli, S. Bova, and B. Gerla, "Free Algebras and Functional Representation for Fuzzy Logics," in *Handbook of Mathematical Fuzzy Logic*, P. Cintula, P. Hájek, and C. Noguera, Eds. College Publications, 2011, vol. 2, pp. 713–791.

[14] A. Knopfmacher and M. E. Mays, "A survey of factorization counting functions," *International Journal of Number Theory*, vol. 01, no. 04, pp. 563–581, 2005.

[15] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction To Algorithms*. McGraw-Hill Publishing Company, 2001.

[16] F. Dodd and L. Mattics, "Estimating the number of multiplicative partitions," *Rocky Mountain Journal of Mathematics*, vol. 17, no. 4, pp. 797–814, 12 1987.

[17] A. Meurer and et al., "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[18] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraph - static and dynamic graph drawing tools," in *GRAPH DRAWING SOFTWARE*. Springer-Verlag, 2003, pp. 127–148.

[19] R. Belohlavek and V. Vychodil, "Residuated lattices of size $\leq$ 12," *Order*, vol. 27, no. 2, pp. 147–161, 2010.

[20] F. Esteva and L. Godo, "Monoidal t-norm based logic: Towards a logic for left-continuous t-norms," *Fuzzy Sets and Systems*, vol. 124, no. 3, pp. 271–288, 2001.

[21] S. Aguzzoli, M. Busaniche, and V. Marra, "Spectral Duality for Finitely Generated Nilpotent Minimum Algebras, with Applications," *J. Log. Comput.*, vol. 17, no. 4, pp. 749–765, 2007.

[22] S. Bova and D. Valota, "Finite RDP-algebras: Duality, Coproducts and Logic," *J. Log. Comput.*, vol. 22, no. 3, pp. 417–450, 2012.

[23] D. Valota, "Representations for logics and algebras related to revised drastic product t-norm," *Soft Computing*, vol. 23, pp. 2331–2342, 2019.

[24] S. Aguzzoli, M. Bianchi, B. Gerla, and D. Valota, "Probability Measures in $\text{Gödel}_{\triangle}$ Logic," in *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, A. Antonucci, L. Cholvy, and O. Papini, Eds. Springer, 2017, pp. 353–363.

[25] ——, "Free algebras, states and duality for the propositional $\text{Gödel}_{\triangle}$ and Drastic Product logics," *International Journal of Approximate Reasoning*, vol. 104, pp. 57–74, 2019.

[26] S. Aguzzoli, M. Bianchi, and D. Valota, "A note on Drastic Product logic," in *Information Processing and Management of Uncertainty*, ser. Communications in Computer and Information Science, vol. 443. Springer, 2014, pp. 365–374.

[27] S. Aguzzoli, S. Bova, and D. Valota, "Free weak nilpotent minimum algebras," *Soft Computing*, vol. 21, no. 1, pp. 79–95, 2017.