Computing Generating Sets of Minimal Size in Finite Algebras[⋆]

Mikoláš Janota[a,*], António Morgado[b], Petr Vojtěchovský[c]

[a]*Czech Technical University, Jugoslávských partyzánů 1580/3, Prague, 160 00, Czechia*
[b]*INESC-ID, Portugal*
[c]*University of Denver, USA*

Abstract

We present an algorithm for calculating a minimal generating set of a finite algebra. Despite the fact that the problem is in NP, a single call to a SAT solver is impractical since the encoding is cubic. Instead, the proposed algorithm solves a series of smaller subproblems. The individual subproblems are formulated as integer linear programs (ILP) that are solved by an off-the-shelf solver. Our implementation shows that the proposed algorithm is highly efficient and is able to compute minimal generators for algebras of orders approximately 2000.

In our experiments we focus on Moufang loops, a variety of loops with properties close to groups. For Moufang loops of prime power order, we are able to calculate a minimal generating set by another method, using theoretical results on the Frattini subloop and algorithms for permutation groups, of which some are reported here for the first time. This second method does not cover all cases, but in the covered cases it serves as a check of correctness of the ILP-based algorithm.

[⋆]The results were supported by the MEYS within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902, This article is part of the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306. This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020, the project INFOCOS with reference PTDC/CCI-COM/32378/2017. P. Vojtěchovský supported by the Simons Foundation Mathematics and Physical Sciences Collaboration Grant for Mathematicians no. 855097 and by the PROF grant of the University of Denver.

[*]Corresponding author

Email address: `mikolas.janota@cvut.cz` (Mikoláš Janota)

Keywords: computational algebra, generators, rank, integer linear programming

1. Introduction

For a subset S of an algebra A, let $\langle S\rangle$ be the subalgebra of A generated by S, that is, the smallest subalgebra of A containing S. Note that $\langle S\rangle$ is the subset of A obtained iteratively from S by applying the operations of A. We say that S is a *generating set* of A if $\langle S\rangle = A$. A generating set of A of the smallest possible cardinality is called a *minimal generating set*. The *rank* $r(A)$ of A is the cardinality of a minimal generating set of A.

Finding small and minimal generating sets is of importance in algebra, both theoretically and for the purposes of computations. For instance, a vector space over a fixed underlying field is completely characterized by its rank (that is, dimension) and computational complexity of linear algebraic algorithms depends on the rank. Groups with a single generator (aka cyclic groups) are easy to understand, while groups with two generators can already be arbitrarily complicated in some sense [1]. The efficiency of algorithms in computational group theory depends heavily on the number of generators given [2, Thm 2.1.1]. As a final example, alternative algebras with two generators are associative [3, Thm 3.1] and hence relatively easy to understand compared to general alternative algebras.

In this paper we present an algorithm for calculating minimal generating sets in *magmas* (sets with a single binary operation) by means of SAT solvers and integer linear programs. Our method cannot compete with specialized algorithms in highly structured magmas, such as groups, but it performs well in less structured magmas where efficient rank algorithms are not available.

According to our best knowledge, there are no existing algorithms for the calculation of the smallest generating sets in the general case. Papadimitriou and Yannakakis introduce a complexity class allowing to classify the complexity of the task for quasigroups [4] (this is possible because quasigroups always have a rank in $O(\log n)$). Generating sets have been heavily studied for groups, cf. [5, 6, 7, 8, 9]. In particular, Lucchini and Mengazzo introduce dedicated algorithms for calculating a set of generators of minimal cardinality for finite *solvable* groups [10].

As a case study, we focus on Moufang loops, a generalization of groups related to alternative algebras [3] and octonions [11]. Moufang loops are a good test candidate for the algorithm for the following reasons:

- The minimal generating set problem is not interesting for random magmas which are typically of rank one. Moufang loops are far from being random magmas.

- The problem is hopelessly difficult for magmas A in which the rank $r(A)$ is comparable to the size of A, such as for some semigroups. In Moufang loops, like in groups, the rank $r(A)$ is bounded above by $\log_2(|A|)$.

- Unlike in groups, no efficient minimal generating set algorithm is known for Moufang loops.

- On the other hand, there exist classes of Moufang loops where the rank and a minimal generating set can be calculated by group-theoretical algorithms running on a (much) larger permutation group associated with the Moufang loop. This allows us to check our method for correctness.

- The rank of a Moufang loop can be increased and controlled within a certain range by simple constructions, such as the direct product. This allows us to construct test examples of desired size and (approximately controlled) rank.

- Moufang loops of certain orders, such as 2^6 or 3^5, have been classified up to isomorphism. By using such an ensemble of algebras for testing, it is demonstrated that the algorithm performs well in most cases, not just in carefully constructed examples.

2. Background on Quasigroups and Loops

An algebra is a set A with a collection of We start with some background on quasigroups and loops, cf. [12]. Readers interested only in the algorithm can skip ahead to Section 4.

A set A with three binary operations $\cdot$, $\backslash$, $/$ is a *quasigroup* if the following axioms hold: $x \cdot (x\backslash y) = y$, $x\backslash(x \cdot y) = y$, $(x \cdot y)/y = x$ and $(x/y) \cdot y = x$. The three binary operations in quasigroups are referred to as *multiplication*, *left division* and *right division*, respectively.

Equivalently, a quasigroup is a set A with a single binary operation $\cdot$ (that is, a magma) such that all left translations $L_x : A \to A$, $y \mapsto x \cdot y$ and all right translations $R_x : A \to A$, $y \mapsto y \cdot x$ are permutations of A.

A *loop* is a quasigroup A with identity element 1 satisfying $x \cdot 1 = x = 1 \cdot x$ for every $x \in A$. A *Moufang loop* is a loop satisfying additionally the identity $x \cdot (y \cdot (x \cdot z)) = ((x \cdot y) \cdot x) \cdot z$, a weakening of the associative law.

A nonempty subset S of a quasigroup (loop) A is a *subquasigroup* (*subloop*) if it is a quasigroup (loop) in its own right. The following result shows that divisions play no role in *finite* quasigroups and loops as far as subalgebras are concerned.

Lemma 2.1. *Let S be a nonempty subset of a finite quasigroup (loop) A. Then S is a subquasigroup (subloop) if and only if it is closed under multiplication.*

Proof. It suffices to show that if S is closed under multiplication then it is also closed under divisions and, in the case of loops, it contains the identity element.

Let $x, y \in S$. The left quotient $x\backslash y$ can be expressed as $L_x^{-1}(y)$. Since A is finite, the permutation L_x has finite order, say n. Then $L_x^{n-1} = L_x^{-1}$ and $x\backslash y = L_x^{-1}(y) = L_x^{n-1}(y)$. We conclude that $x\backslash y \in S$ since S is closed under multiplication. Dually, $x/y \in S$.

If A is a loop with identity element 1 and $x \in S$, then $1 = x\backslash x \in S$. □

Proper subquasigroups cannot be too large, cf. Proposition 2.2. This imposes a logarithmic upper bound on the rank of a quasigroup in terms of its order, cf. Corollary 2.3, which allows us to test Algorithm 1 on relatively large algebras. On the other hand, the complements of proper subalgebras are then large, which potentially adds to the running time of the algorithm.

Proposition 2.2. *Let A be a finite quasigroup and let S be a proper subquasigroup of A. Then $|S| \leq |A|/2$.*

Proof. Let $x \in A \setminus S$. We show that the right coset $Sx = \{sx : s \in S\}$ is disjoint from S and of the same size as S. The fact that $|Sx| = |S|$ follows from the fact that $Sx = R_x(S)$ and R_x is a permutation. Suppose, for a contradiction, that $y \in S \cap Sx$. Then $y = s_1 = s_2 x$ for some $s_1, s_2 \in S$ and therefore $x = s_2\backslash s_1 \in S$, a contradiction. □

Corollary 2.3. *Let A be a finite quasigroup. Then $r(A) \leq \log_2(|A|)$.*

For general quasigroups and loops, Proposition 2.2 is best possible.

We can use the direct product construction to bring the order of the magma to the desired size and the expected rank to a predictable range, cf. Lemma 2.4. The output of Algorithm 1 demonstrates that the entire range of possible ranks $r(A \times B)$ from Lemma 2.4 is attained in examples.

Lemma 2.4. *Let A and B be finite magmas with identity elements. Then*

$$\max(r(A), r(B)) \leq r(A \times B) \leq r(A) + r(B).$$

Proof. The first inequality is clear: If S is a generating set of $A \times B$ then $\{a : (a, b) \in S\}$ is a generating set of A, and similarly for B.

For the second inequality, consider $X \in \{A, B\}$, let 1_X be the identity element of X and let S_X be a minimal generating set of X. Then

$$S = (\{1_A\} \times S_B) \cup (S_A \times \{1_B\})$$

is a generating set of $A \times B$, since the identity elements allow us to independently obtain any element of A from S_A in the first coordinate and any element of B from S_B in the second coordinate. This shows that $r(A \times B) \leq |S| = |S_A| + |S_B| = r(A) + r(B)$. □

In some situations the rank $r(A \times B)$ can be predicted from $r(A)$ and $r(B)$. For instance, if A, B are loops in which powers of elements are well-defined, the order of every element divides the order of the loop and $\gcd(|A|, |B|) = 1$, then $r(A \times B) = \max(r(A), r(B))$, attaining the lower bound of Lemma 2.4. But in general, it is difficult to predict the rank of $r(A \times B)$ from $r(A)$ and $r(B)$ without explicitly calculating it. Our algorithm does not take advantage of any theoretical results concerning the rank $r(A \times B)$.

3. Results on Frattini Subalgebras and Moufang Loops

In this section we gather results on the Frattini subalgebra that is highly relevant to the problem of minimal generating sets, and also on Moufang loops, our case study. Theorem 3.4 due to Gábor P. Nagy appears in the literature for the first time here; its proof will be presented elsewhere [13].

An element x of an algebra A is said to be a *nongenerator* if for every subset S of A satisfying $\langle S, x\rangle = A$ we already have $\langle S\rangle = A$. In other words, an element $x \in A$ is a nongenerator if it can be removed from any generating set of A without impacting its generating property. The *Frattini subalgebra* $\Phi(A)$ of A is the set of all nongenerators of A, and it is indeed a subalgebra of A by Lemma 3.1.

The following two results are an easy generalization of the results from [12, Section VI.2].

Lemma 3.1. *Let A be an algebra. Then $\Phi(A)$ is a subalgebra of A.*

Proof. Let f be any n-ary operation of A and suppose that $x_1, \dots, x_n \in \Phi(A)$. We will show that $f(x_1, \dots, x_n)$ is a nongenerator. Let $S \subseteq A$ be such that $\langle S, f(x_1, \dots, x_n)\rangle = A$. Since $f(x_1, \dots, x_n) \in \langle S, x_1, \dots, x_n\rangle$, we have

$$\langle S, x_1, \dots, x_n\rangle \geq \langle S, f(x_1, \dots, x_n)\rangle = A.$$

Since x_n is a nongenerator, we conclude that $\langle S, x_1, \dots, x_{n-1}\rangle = A$. Proceeding similarly for the nongenerators $x_1, \dots, x_{n-1}$, we reach $\langle S\rangle = A$. □

A subalgebra M of A is said to be *maximal* if $M < A$ and whenever $M \leq B \leq A$ for some subalgebra B then either $M = B$ or $B = A$.

Proposition 3.2. *If A is an algebra that contains at least one maximal subalgebra then $\Phi(A)$ is the intersection of all maximal subalgebras of A, else $\Phi(A) = A$.*

Proof. First suppose that A has at least one maximal subalgebra and let $F < A$ be the intersection of all maximal subalgebras of A. If $x \in A \setminus F$ then there is a maximal subalgebra M of A such that $x \notin M$, but then $\langle M, x\rangle = A > M = \langle M\rangle$ by maximality, proving that $x \notin \Phi(A)$.

Conversely, if $x \notin \Phi(A)$, let $S \subseteq A$ be such that $\langle S\rangle < \langle S, x\rangle = A$. By Zorn's Lemma, there exists a subalgebra $M \leq A$ that is maximal with respect to the property that $S \subseteq M$ and $x \notin M$. Clearly, $M < A$. Consider any $B \leq A$ such that $M < B$. Since $S \subseteq B$ and $M < B$ we must also have $x \in B$ and thus $B \geq \langle S, x\rangle = A$. Hence M is in fact a maximal subalgebra of A. Then $x \notin F \leq M$.

Now suppose that A has no maximal subalgebras. If $x \notin \Phi(A)$ then the above paragraph shows that A has a maximal subalgebra M, a contradiction. Hence $\Phi(A) = A$. □

Note that the assumption of Proposition 3.2 is satisfied in nontrivial finite loops.

The following result of Bruck describes the rank and all minimal generating sets in nilpotent loops of prime power order.

The *center* $Z(Q)$ of a loop is the set of all elements of Q that commute *and* associate with all elements of Q. The *iterated centers* are then defined by $Z_0(Q) = 1$, $Z_{i+1}(Q) = \pi_i^{-1}(Z(Q/Z_i(Q)))$, where $\pi_i : Q \to Q/Z_i(Q)$ is the canonical projection $x \mapsto xZ_i(Q)$. A loop Q is *nilpotent* if $Q = Z_m(Q)$ for some m. For a prime p, a group G is an *elementary abelian p-group* if it is a commutative group such that $x^p = 1$ for every $x \in G$. In additive notation, an elementary abelian p-group $(G, +)$ of size p^d can be seen as a vector space of dimension d over the field of order p.

Theorem 3.3 ([12, Theorem VI.2.3]). *Let p be a prime and Q a nilpotent loop of order $p^n > 1$. Then the Frattini subloop $\Phi(Q)$ is normal in Q and $Q/\Phi(Q)$ is an elementary abelian group of order p^d for some $d > 0$. The rank of Q is then equal to d and $\{x_1, \dots, x_d\} \subseteq Q$ generates Q if and only if $\{x_1\Phi(Q), \dots, x_d\Phi(Q)\}$ is a basis of the vector space $Q/\Phi(Q)$.*

To calculate the rank and a minimal generating set in a nilpotent loop Q of prime power order by Theorem 3.3, we therefore need to calculate the Frattini subloop $\Phi(Q)$. The following recent result shows that this can be done by transferring the calculation to the *multiplication group*

$$\mathrm{Mlt}(Q) = \langle L_x, R_x : x \in Q \rangle$$

of Q, as long as $\mathrm{Mlt}(Q)$ is itself nilpotent. This might seem counterproductive since $\mathrm{Mlt}(Q)$ is typically much larger than Q, but it is a group, and standard algorithms from computational group theory apply.

Theorem 3.4 (Gábor P. Nagy). *Let Q be a finite loop such that* $\mathrm{Mlt}(Q)$ *is nilpotent. Consider the natural permutation action of* $\mathrm{Mlt}(Q)$ *on Q. Then* $\Phi(Q)$ *is equal to the orbit of the group* $\Phi(\mathrm{Mlt}(Q))$ *containing* 1.

Let us now turn attention to Moufang loops, one of the most investigated varieties of loops.

Glauberman and Wright proved that every Moufang loop of prime power order is nilpotent [14, 15]. Bruck showed that a nilpotent loop of prime power order has a nilpotent multiplication group, cf. [12, Lemma VI.2.2]. Theorems 3.3 and 3.4 therefore apply to Moufang loops of prime power order. (However, we stress that they do not apply to general Moufang loops, much less to general magmas, and hence the orbit computation of Theorem 3.4 does not supersede Algorithm 1.)

Moufang loops of order $n < 64$ were classified by Goodaire, May and Raman [16], of order $n \in \{64, 81\}$ by Nagy and Vojtěchovský [17], and of order $n = 243$ by Slattery and Zenisek [18]. Libraries of Moufang loops are available in the `GAP` [19] package `LOOPS` [20].

Example 3.5. Let Q be the 100th Moufang loop of order 64 in the `LOOPS` package, i.e., `MoufangLoop(64,100)`. Then $G = \mathrm{Mlt}(Q)$ is a nilpotent group of order 4096. The Frattini subgroup $\Phi(G)$ of G has order 64. The Frattini subloop $\Phi(Q)$ of Q is the orbit of $\Phi(G)$ containing 1, a set with 8 elements. Thus $Q/\Phi(Q)$ is an elementary abelian group of size $64/8 = 8 = 2^3$, Q has rank 3, and any 3-element subset $\{x_1, x_2, x_3\}$ of Q such that $\{x_1\Phi(Q), x_2\Phi(Q), x_3\Phi(Q)\}$ forms a basis of the vector space $Q/\Phi(Q)$ is a minimal generating set of Q.

We conclude with another observation for Moufang loops that impacts Algorithm 1.

As in the case of groups, the Lagrange Theorem holds for Moufang loops [21, 22], but unlike in the case of groups, no elementary proof of Lagrange Theorem is known for Moufang loops; all known proofs rely on the classification of finite simple groups.

Theorem 3.6 (Gagola-Hall, Grishkov-Zavarnitsine)**.** *Let A be a finite Moufang loop and let S be a subloop of A. Then $|S|$ divides $|A|$.*

Corollary 3.7. *Let A be a finite Moufang loop and let p be the smallest prime dividing $|A|$. If S is a proper subloop of A then $|S| \leq |A|/p$.*

4. Algorithm

In this section we present an algorithm for finding a minimal generating set of a general finite algebra A. The decision version of the problem is in NP because verifying that a set of elements is generating can be done in polynomial time.[1] One could therefore find a rank of a finite algebra by a series of SAT calls. Encoding the problem into SAT is essentially a reachability problem, where we introduce Boolean variables s_x^i representing that an element x is generated by applying i-times the operations of A. For finite magmas, this results in a polynomial encoding because we only need to consider at most $|A|$ applications of the binary operation of multiplication. Unfortunately, since the whole multiplication table needs to be considered in each step, the resulting encoding is cubic, which renders it impractical.

Rather than solving the problem by a single SAT call, we propose to apply the counterexample guided abstraction refinement paradigm (CEGAR) [23].

[1] We conjecture that it is also NP-hard but this is out of the scope of the paper.

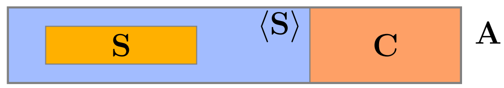


Figure 1: Illustration of the concept of a subcomplement $C = A \smallsetminus \langle S \rangle$

This enables us to translate the problem into a sequence of sub-problems, which are substantially easier than the whole problem.

If S is a subset of the algebra A, then $C(S) = A \smallsetminus \langle S \rangle$ will be called the *subcomplement* of S.[2] The concept is illustrated by Figure 1. The terminology is supposed to indicate that $C(S)$ is a complement of the *sub*algebra $\langle S \rangle$, as well as that $C(S)$ is a *sub*set of the complement $A \smallsetminus S$.

Subcomplements are crucial to the presented algorithm for two reasons: (i) the subcomplement $C(S)$ is empty if and only if the set S is a generating set, (ii) if a subcomplement is nonempty, every generating set must intersect with it, cf. Proposition 4.1.

Proposition 4.1. *Let A be an algebra, S a subset of A and $C(S) = A \smallsetminus \langle S \rangle$. If $C(S)$ is nonempty (that is, S does not generate A), then every generating set of A has a nonempty intersection with $C(S)$.*

Proof. Suppose that $T \subseteq A$ satisfies $\langle T \rangle = A$ and $T \cap C(S) = \emptyset$. Since $T \cap (A \smallsetminus \langle S \rangle) = \emptyset$ and $T \subseteq A$, it follows that $T \subseteq \langle S \rangle$. As $\langle T \rangle$ is the smallest subalgebra of A containing T, we deduce that $A = \langle T \rangle \subseteq \langle S \rangle \neq A$, a contradiction. □

Given a collection Γ of subsets of A, a subset $H \subseteq A$ is a *hitting set* of Γ if for every $C \in \Gamma$ we have $H \cap C \neq \emptyset$. By Proposition 4.1, any generating set of A must be a hitting set of any collection Γ of nonempty subcomplements.

This enables us to invoke the implicit hitting set paradigm where new sets C are gradually generated; interested reader is referred to relevant literature [24, 25, 26, 27, 28]. Here we focus on the specific algorithm for generating sets inspired in this paradigm.

Algorithm 1 shows the overall structure of the computation. The algorithm maintains a set of subcomplements Γ and in each iteration it calculates a smallest hitting set S of Γ. If this hitting set is a generating set of A, the

[2] Note that in the case of finite quasigroups, the closure $\langle S \rangle$ in Algorithm 1 can be calculated by closing the set S under multiplications only, cf. Lemma 2.1.

Algorithm 1: Minimal Generating Set

input : finite algebra A
output: minimal subset S of A s.t. $\langle S \rangle = A$

$\Gamma \leftarrow \emptyset$ `// collected subcomplements`
while *true* **do**
  $S \leftarrow$ minimal hitting set of Γ
  if $\langle S \rangle = A$ **then**
    return S `//` S `is generating`
  $\Gamma \leftarrow \Gamma \cup \{\mathsf{extractSubcomplement}(A, S)\}$

algorithm stops and we are done. Otherwise, we construct a new subcomplement (see below) and add it to the set Γ.

The minimum hitting set problem for a set of sets Γ is readily translated into an integer linear program (ILP) as follows.

$$\begin{gathered}\min \sum_{x \in A} x \textit{ subject to} \\ x \in \{0, 1\} \textit{ for every } x \in A, \\ \sum_{x \in C} x \geq 1 \textit{ for every } C \in \Gamma.\end{gathered}$$

We remark that calculating the smallest hitting set is a well-known NP-complete problem [29], which warrants the use of ILP. Also observe that the ILP program is always feasible since setting all $x \in A$ to 1 trivially satisfies all the constraints. Due to Proposition 4.1, any generating set of A must be a hitting set of Γ and the fact that it is a smallest hitting set guarantees that it is a minimal generating set, once found.

An important ingredient of Algorithm 1 is how to enlarge the set Γ of subcomplements while $S \subseteq A$ is under consideration. In the pseudocode, this is encapsulated by the routine extractSubcomplement.

The most direct approach is to add the subcomplement $A \setminus \langle S \rangle$ to Γ. However, to speed up the algorithm, we attempt to calculate a smaller subcomplement instead. Adding a smaller subcomplement to Γ restricts more strongly future hitting sets S and therefore increases our chance of finding a generating set or may speed up the demonstration of non-existence thereof.

Algorithm 2: Subcomplement minimization

input : finite algebra A and $S \subseteq A$
output: subcomplement $C \subseteq A \setminus \langle S \rangle$

$E \leftarrow A \setminus \langle S \rangle$ `// possible extensions`
while $E \neq \emptyset$ **do**
  $x \leftarrow$ arbitrary element of E
  if $\langle S \cup \{x\} \rangle = A$ **then**
    $E \leftarrow E \setminus \{x\}$ `//` $S \cup \{x\}$ `already generating`
  else
    $S \leftarrow S \cup \{x\}$
    $E \leftarrow A \setminus \langle S \rangle$
return $A \setminus \langle S \rangle$

A smaller subcomplement can be found by trying to extend the set S by some element $x \in A \setminus \langle S \rangle$ and check that it still does *not* generate the whole of A. And if it does not, we use the subcomplement of this extended S instead. This process is repeated until there are no more elements to try. This procedure is summarized in Algorithm 2.

In essence, Algorithm 2 is greedy and does not guarantee to produce the smallest possible subcomplement but it does guarantee that it is irreducible in the sense that removing any element from it makes the extended S already generating.

Note that Algorithm 2 requires $O(|A \setminus \langle S \rangle|)$ tests $\langle S \cup \{x\} \rangle \neq A$. Since $\langle S \rangle \neq A$ is anti-monotone with respect to S, more advanced procedures can be considered [30].

Example 4.2. As a toy example consider the Klein four-group $A = \{e, a, b, c\}$, which is abelian and has rank 2. The product is defined by the following rules: $ex = xe = x$, $xx = e$, and in other cases $yz = w$ with $w \neq y$, $w \neq z$, $w \neq e$.

Calling the ILP solver on the initial empty Γ yields empty S as the minimal hitting set, i.e. its subcomplement is the whole of A. Algorithm 2 reduces the subcomplement's size as follows. Suppose Algorithm 2 attempts to extend S with elements of A in the order e, a, b, c. Since $\langle S \cup \{e\} \rangle = \langle \{e\} \rangle = \{e\}$, the element e is kept. Analogously, a is also kept. Extending with either of b, c already gives a generating set and therefore these will not be inserted into S. Hence, we are left with the subcomplement $A \setminus \langle \{e, a\} \rangle = \{b, c\}$. This means that after the 1^{st} iteration of Algorithm 1, the set Γ is $\{\{b, c\}\}$.

Suppose that in the 2^{nd} iteration the ILP solver returns $S = \{b\}$ with $\langle S\rangle = \{b, e\}$. In this case, Algorithm 2 is unable to extend S because adding any element already generates A. Hence, after the 2^{st} iteration of Algorithm 1, the set Γ is $\{\{b, c\}, \{a, c\}\}$.

In the 3^{nd} iteration, the ILP solver necessarily returns $S = \{c\}$. Similarly as before, Algorithm 2 does not extend S. The set Γ increases to $\{\{b, c\}, \{a, c\}, \{a, b\}\}$. At this point, at least 2 elements are necessary to construct a hitting set of Γ, i.e., the lower bound on the rank of A is 2.

Suppose that the solver picks the hitting set $S = \{a, b\}$, which is generating and we have an answer. In this example, any hitting set will be generating but this might not be true in general.

5. Experimental Evaluation

We consider the problem of computing the generating set of smallest cardinality for a finite magma. As such, Algorithm 1 was implemented in the tool *mgens*. Two versions of mgens were considered. The first version computes the minimal hitting set on line 3 of Algorithm 1 iteratively using a SAT solver together with encodings of cardinality constraints into CNF. The SAT solver used was the CaDiCaL [31] solver using the library PBLIB [32] to encode the cardinality constraints into CNF. The second version of mgens takes advantage of the capabilities of an ILP solver to natively optimize a cost function to compute the minimal hitting set. The ILP solver used was the Gurobi [33] solver. In the experiments, we refer to the version of mgens using the SAT solver with cardinality constraints as *mgens-iter*, while the version using the ILP solver as *mgens-opt*.

Additionally, we have implemented a brute-force search that exhaustively tests for each subset of the elements whether it is generating or not. The search goes systematically from smallest to largest subsets and terminates once a generating set is found. In the experiments, we refer to the brute-force search as *mgens-bf*.

The experiments were run on a set of Moufang loops (see Section 3) and products of Moufang loops with the groups of order 8 and 9. We considered all Moufang loops from the package `LOOPS` [20] of `GAP`. The package contains all nonassociative Moufang loops of order $n \leq 64 = 2^6$ and of orders $n = 81 = 3^4$ and $n = 243 = 3^5$. Note that no efficient methods for calculating the rank of (general) Moufang loops are known. A total of 4497 Moufang loops were used. There are 5 groups of order 8 and 2 groups of order 9. The total number of products between Moufang loops and groups, order 8 and 9, amounts to 31479. We divided the benchmarks into four sets as follows:

- **Moufangs** — all the considered Moufang loops (4497 instances),
- **Moufangs** $\times$ **G8.1** — the product between the Moufang loops and the cyclic group of order 8 denoted as 8.1 (4497 instances),
- **Moufangs** $\times$ **G8** — the product between the Moufang loops and the groups of order 8 (22485 instances),
- **Moufangs** $\times$ **G9** — the product between the Moufang loops and the groups of order 9 (8994 instances).

Note that the set Moufangs $\times$ G8 includes the instances Moufangs $\times$ G8.1. Nevertheless we present the results for the set Moufangs $\times$ G8.1 in order to be able to compare the different implementations. Given the large number of benchmarks in Moufangs $\times$ G8, we obtain results for this set of benchmarks using only our best implementation, that is *mgens-opt*.

All the experiments were performed on servers with Intel(R) Xeon(R) CPU at 2.60GHz, 24 cores, 64GB RAM with a timeout of 600 seconds.

Table 1 presents the number of solved instances by each of the versions of *mgens* grouped by the reported rank. The table suggests several interesting observations. When multiplying the Moufang loops by the cyclic group of order 8, the rank of the loops almost always goes up (by 1). For instance, there are 780 Moufang loops of rank 3 but only 100 of the products have rank 3. Notably, *all* of the loops of rank 5 lead to a product of rank 6.

Since the rank of the considered Moufang loops is at most 5, the maximum possible rank is 8, which is obtained in products with the elementary abelian group of order 8 (rank 3). Table 1 shows that this happens in all 80 cases.

As can be seen from the table, *mgens-opt* solves the largest number of benchmarks, being able to solve instances with bigger ranks. On the other hand, the brute-force search algorithm *mgens-bf* solves the least number of instances, having difficulties solving instances with rank 4 or higher. Additionally, we can also see from the table that all the versions are able to solve all the Moufang loop benchmarks, while for product benchmarks, only *mgens-opt* is able to solve the majority of the benchmarks followed by *mgens-iter*.

Benchmark set	**Rank**	**Total**	**mgens-opt**	**mgens-iter**	**mgens-bf**
Moufangs	3	780	780	780	780
	4	3637	3637	3637	3637
	5	80	80	80	80
Moufangs × G8.1	3	100	100	100	100
	4	698	698	646	45
	5	3619	3619	4	0
	6	80	80	0	0
Moufangs × G8	3	361	361	n/a	n/a
	4	875	875	n/a	n/a
	5	5693	5634	n/a	n/a
	6	11616	11441	n/a	n/a
	7	3860	3832	n/a	n/a
	8	80	79	n/a	n/a
Moufangs × G9	3	1440	1440	1440	1440
	4	7300	7300	6927	34
	5	237	237	0	0
	6	17	17	0	0

Table 1: Number of instances solved per solver with the corresponding rank.

Figure 2 offers a more detailed presentation of the results, where the results are presented as *cactus plots*. A cactus plot shows how many instances are solved within a specific timeout. The plot is obtained by ordering problem instances by CPU-time needed to solve the instance (in increasing order). Then, each point in the plot corresponds to a problem instance where the horizontal coordinate is its position in this sequence and the vertical coordinate is CPU-time. Additionally, each of the points in the cactus plots is colored according to the rank of the instance.

Figure 2 (a) confirms that all the versions of *mgens* are able to solve all the Moufang loop benchmarks. Nevertheless, *mgens-opt* is able to solve all the benchmarks in less than 10 seconds while, *mgens-iter* takes less than 40 seconds, and *mgens-bf* requires slightly more than 100 seconds.

From Figure 2 (b) and (d), we can see that *mgens-opt* is the fastest version with the highest number of benchmarks solved, solving the majority of these instances in less than 100 seconds, followed by the version *mgens-iter*. The slowest version is *mgens-bf* that is able to solve less than a quarter of the benchmarks within the timeout.

Finally, from Figure 2 (c) we can see that *mgens-opt* is able to solve the majority of the benchmarks with less than 100 seconds each, many of them with rank 7. The plots confirm that loops with higher rank are more difficult to calculate, which is only natural since for a loop A of order n, the algorithm needs to rule out $\sum_{k\in 1..r(A)-1}\binom{A}{k}$ possible candidates for a generating set.

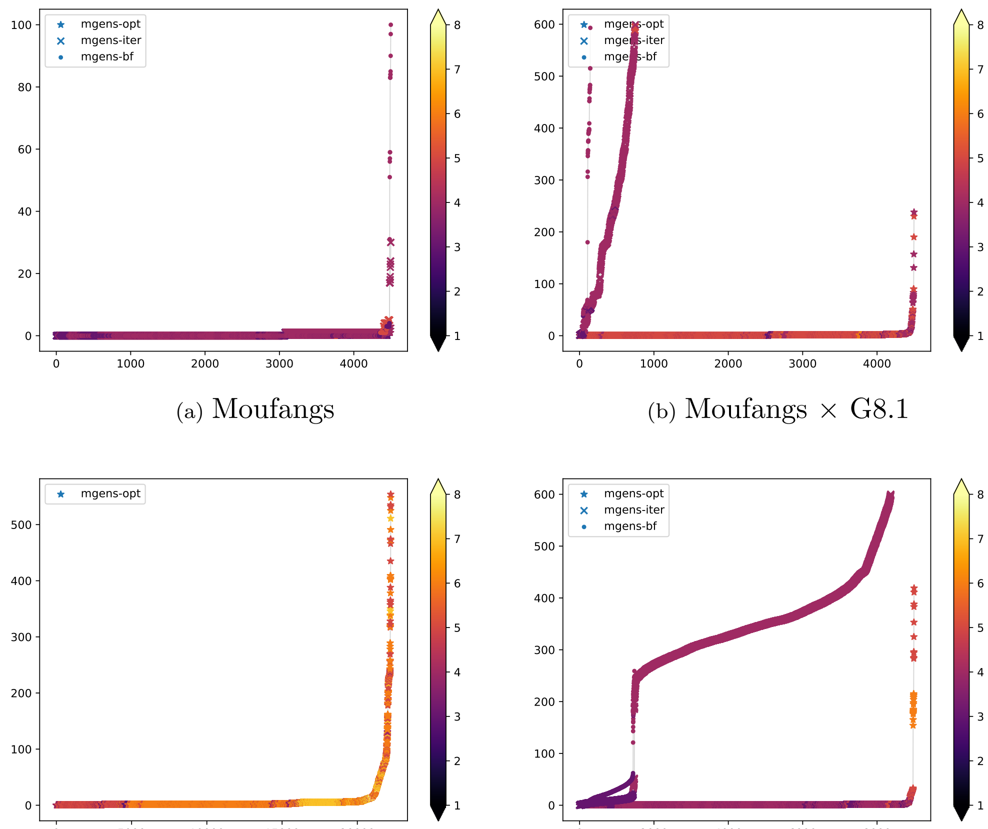
In our experiments, ILP clearly outperforms SAT. Possible justification for this might be that modern SAT solvers are anchored in propositional resolution [34] and proving the optimality of a hitting set in propositional resolution may lead to exponential refutations [35].

5.1. Difficult Instances

Out of the 35976 instances considered in the evaluation we were left with 263 unsolved. We tackled these instances as follows. Recall that Algorithm 2 is used to minimize a subcomplement in every iteration of the principal algorithm (Algorithm 1). Algorithm 2 has a random component because extension elements can be tried in an arbitrary order. We reran the algorithm with randomly shuffled order of elements, which have solved new 193 instances, leaving 66 unsolved. This observation gives us a simple way of tackling a hard instance by trying different random orders—controlled by a seed for the pseudorandom generator.

For the 66 unsolved instances we tried 10 different seeds with a 60 second timeout, resolving further 46 instances. So in the end, we were left with 20 instances not solved by our algorithm. However, since they were all of order $512 = 2^9$ we were able to apply Theorems 3.3 and 3.4 (applicable to Moufang loops of order that is a power of some prime number).

All the instances that required solving by hand had rank 7 except in one case where the rank was 6 and one case of 8. Further, all these instances resulted from a product of a Moufang loop of order 64 with the elementary Abelian group of order 8. This is not surprising because this group has rank 3 and therefore has the potential of substantially increasing the rank of the product. We have also tested running our algorithm on these instances with an enforced lower bound 7 on the rank and then the algorithm always terminated, i.e., providing a witness for the generating set whose size we have determined theoretically.



(a) Moufangs

(b) Moufangs × G8.1

(c) Moufangs × G8

(d) Moufangs × G9

Figure 2: Cactus plots for different sets of instances using the solvers *mgens-iter*, *mgens-opt* and *mgens-bf*

6. Conclusions

The article tackles the problem of calculating the rank of a given magma, which translates to calculating the smallest generating set of the magma. To the best of our knowledge, no other algorithm for this problem is known.

It is easy to see that the problem is in NP but converting it to SAT is impractical because we are unaware of an encoding better than cubic. Rather than converting to a single NP-hard problem, we propose an algorithm that solves a series of (easier) NP-hard problems.

As a case study, we have considered a set of Moufang loops as well as their products with groups. In this way, we have obtained algebraically interesting magmas. Random magmas typically have rank 1 and therefore would not be interesting for our study.

The experimental results show that this approach is surprisingly effective. We are able to solve instances with magmas of order 512 and rank 8. Since $\binom{512}{8} \approx 10^{17}$, any brute-force approaches are ruled out for such cases. On the theoretical side, we have also shown that for Moufang loops of prime power order it is possible to convert the problem to a calculation of the Frattini subgroup of a larger permutation group.

It is an open problem why the algorithm performs so well on the considered instances. In particular, despite the problem being in NP, it is practically efficient to solve it by Algorithm 1, which may require exponential number of calls to an NP oracle. One may therefore ask, if there is a theoretical justification why our algorithm works so well. Such justification cannot be entirely trivial because the subcomplements are necessarily large in our case (see Corollary 3.7).

We believe that the results themselves will be interesting for algebraists. For instance, the rank of a product of two magmas is upper bounded by the sum of the ranks of the operands. However, in some cases the rank of the product is lower than the sum. This behavior is not characterized and our results provide data to help forming new hypotheses in that direction.

References

[1] A. Shalev, Asymptotic group theory, Notices Amer. Math. Soc. 48 (4) (2001) 383–389.

[2] A. Seress, Permutation group algorithms. Cambridge Tracts in Mathematics, Vol. 152, Cambridge University Press, Cambridge, 2003.

[3] R. D. Schafer, An introduction to nonassociative algebras. Pure and Applied Mathematics, Vol. 22, Academic Press, New York-London, 1966.

[4] C. H. Papadimitriou, M. Yannakakis, On limited nondeterminism and the complexity of the V-C dimension, Journal of Computer and System Sciences 53 (2) (1996) 161–170. doi:https://doi.org/10.1006/jcss.1996.0058.

[5] F. D. Volta, A. Lucchini, Finite groups that need more generators than any proper quotient, Journal of the Australian Mathematical Society. Series A. Pure Mathematics and Statistics 64 (1) (1998) 82–91. doi:10.1017/s1446788700001312.

[6] V. Arvind, J. Torán, The complexity of quasigroup isomorphism and the minimum generating set problem, in: T. Asano (Ed.), Algorithms and Computation, 17th International Symposium, ISAAC 2006, Kolkata, India, December 18-20, 2006, Proceedings, Vol. 4288 of Lecture Notes in Computer Science, Springer, 2006, pp. 233–242. doi:10.1007/11940128_25.

[7] L. Halbeisen, M. Hamilton, P. Růžička, Minimal generating sets of groups, rings, and fields, Quaestiones Mathematicae 30 (3) (2007) 355–363. doi:10.2989/16073600709486205.

[8] F. Mengazzo, The number of generators of a finite group, Irish Math. Soc. Bulletin 50, http://www.irishmathsoc.org//bull50/ (2003).

[9] A. Lucchini, The largest size of a minimal generating set of a finite group, Archiv der Mathematik 101 (1) (2013) 1–8. doi:10.1007/s00013-013-0527-y.

[10] A. Lucchini, F. Menegazzo, Computing a set of generators of minimal cardinality in a solvable group, Journal of Symbolic Computation 17 (5) (1994) 409–420. doi:10.1006/jsco.1994.1027.

[11] J. C. Baez, The octonions, Bull. Amer. Math. Soc. (N.S.) 39 (2) (2002) 145–205. doi:10.1090/S0273-0979-01-00934-X.

[12] R. H. Bruck, A survey of binary systems. Gruppentheorie Ergebnisse der Mathematik und ihrer Grenzgebiete, Vol. 20, Springer Verlag, Berlin-Göttingen-Heidelberg, 1958.

[13] A. Drápal, M. Kinyon, P. Vojtěchovský, Loop theory, in preparation.

[14] G. Glauberman, On loops of odd order. II, J. Algebra 8 (1968) 393–414. doi:10.1016/0021-8693(68)90050-1.

[15] G. Glauberman, C. R. B. Wright, Nilpotence of finite Moufang 2-loops, J. Algebra 8 (1968) 415–417. doi:10.1016/0021-8693(68)90051-3.

[16] E. G. Goodaire, S. May, M. Raman, The Moufang loops of order less than 64, Nova Science Publishers, Inc., Commack, NY, 1999.

[17] G. P. Nagy, P. Vojtěchovský, The Moufang loops of order 64 and 81, J. Symbolic Comput. 42 (9) (2007) 871–883. doi:10.1016/j.jsc.2007.06.004.

[18] M. C. Slattery, A. L. Zenisek, Moufang loops of order 243, Comment. Math. Univ. Carolin. 53 (3) (2012) 423–428.

[19] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.11.1 (2021).
URL https://www.gap-system.org

[20] G. Nagy, P. Vojtěchovský, LOOPS, a package for GAP 4.3, download GAP at https://www.gap-system.org/. (2006).

[21] S. G. III, J. Hall, Lagrange's theorem for Moufang loops, Acta Sci. Math. (Szeged) 71 (1–2) (2005) 45–64.

[22] A. N. Grishkov, A. V. Zavarnitsine, Lagrange's theorem for Moufang loops, Mathematical Proceedings of the Cambridge Philosophical Society 139 (1) (2005) 41–57. doi:10.1017/s0305004105008388.

[23] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, J. ACM 50 (5) (2003).

[24] J. Davies, F. Bacchus, Solving MAXSAT by solving a sequence of simpler SAT instances, in: J. H. Lee (Ed.), Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings, Vol. 6876 of Lecture Notes in Computer Science, Springer, 2011, pp. 225–239. doi:10.1007/978-3-642-23786-7_19.

[25] J. Davies, F. Bacchus, Exploiting the power of MIP solvers in MaxSAT, in: M. Järvisalo, A. V. Gelder (Eds.), Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings, Vol. 7962 of Lecture Notes in Computer Science, Springer, 2013, pp. 166–181. doi:10.1007/978-3-642-39071-5_13.

[26] A. Ignatiev, M. Janota, J. Marques-Silva, Quantified maximum satisfiability: A core-guided approach, in: Theory and Applications of Satisfiability Testing - SAT, Vol. 7962 of Lecture Notes in Computer Science, Springer, 2013, pp. 250–266. doi:10.1007/978-3-642-39071-5_19.

[27] E. Moreno-Centeno, R. M. Karp, The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment, Oper. Res. 61 (2) (2013) 453–468. doi:10.1287/opre.1120.1139.

[28] P. Saikko, J. P. Wallner, M. Järvisalo, Implicit hitting set algorithms for reasoning beyond NP, in: C. Baral, J. P. Delgrande, F. Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR, AAAI Press, 2016, pp. 104–113.
URL http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12812

[29] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[30] M. Janota, J. Marques-Silva, On the query complexity of selecting minimal sets for monotone predicates, Artif. Intell. 233 (2016) 73–83.
URL https://doi.org/10.1016/j.artint.2016.01.002

[31] A. Biere, CaDiCaL, Lingeling, PLingeling, Treengeling and YalSAT entering the SAT competition 2017 (2017).

[32] T. Philipp, P. Steinke, PBLib – a library for encoding pseudo-boolean constraints into CNF, in: M. Heule, S. Weaver (Eds.), Theory and Applications of Satisfiability Testing – SAT 2015, Vol. 9340 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 9–16. doi:10.1007/978-3-319-24318-4_2.

[33] L. Gurobi Optimization, Gurobi optimizer reference manual (2021). URL http://www.gurobi.com

[34] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, Vol. 336 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2021, pp. 133–182. doi:10.3233/FAIA200987.

[35] S. Jukna, Exponential lower bounds for semantic resolution, in: P. Beam, S. R. Buss (Eds.), Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, April 21-24, 1996, Vol. 39 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS/AMS, 1996, pp. 163–172. doi:10.1090/dimacs/039/10.