Enumerating Projective Planes of Order Nine with Proof Verification

Daniel Dallaire and Curtis Bright

University of Windsor, Canada

Abstract. In this paper we describe a method of enumerating projective planes of order nine. The enumeration was previously completed by Lam, Kolesova, and Thiel using highly optimized and customized search code. Despite the importance of this result in the classification of projective geometries, the previous search relied on unverified code and has never been independently verified. Our enumeration procedure uses a hybrid satisfiability (SAT) solving and symbolic computation approach. SAT solving performs an enumerative search, while symbolic computation removes symmetries from the search space. Certificates are produced which demonstrate the enumeration completed successfully.

Keywords: Satisfiability Checking · Symbolic Computation · Combinatorial Enumeration · Computer-assisted Proof

1 Introduction

Projective geometry is a form of geometry developed by Renaissance artists in order to describe how to represent a three dimensional scene onto a canvas. Projective geometry differs from the more familiar Euclidean geometry because there are no parallel lines in a projective geometry. For example, a pair of train tracks—parallel lines in three dimensions—are no longer parallel when projected onto a canvas because the tracks will meet on the horizon.

The classification of projective geometries is an important and long-standing mathematical problem which despite intense study is still incomplete. Even in the finite case—when the geometry consists of a finite number of points—a classification is missing in the two-dimensional case. Two dimensional projective geometries are known as *projective planes* and in this work we consider one of the few cases for which a classification is known (albeit one relying on custom-written and unverified search code). We develop a computer-assisted method that we plan to use to complete the classification of projective planes that have exactly ten points on each line—known as projective planes of order nine.

The method fits into the "SC-square" paradigm of relying on both *satisfiability checking* and *symbolic computation*—two fields of computer science which developed with little interaction [1] but recently have been combined and applied to a variety of problems like circuit verification [8], finding new algorithms for matrix multiplication [6], and improving cylindrical algebraic decomposition algorithms [2]. Our method uses satisfiability (SAT) solvers to search for projective

planes and symbolic computation to detect when two partial projective planes are isomorphic to each other. During the search many isomorphic subplanes are detected and pruned from the search, thereby dramatically improving the efficiency of the solver.

Crucically, our work does not rely on trusting the output of either the SAT solver or the computer algebra system (CAS). Both tools produce certificates that can be used to *verify* the output without taking their claims on faith. Although this is work in progress we are confident that our approach will successfully complete a classification of projective planes of order nine without requiring the trust of any search code. This is particularly important in computational classification which requires the generation of a complete list of *all* instances of a given object—it is in general quite difficult to prove that the list is complete. Moreover, it is not feasible to prove that a complicated algorithm (like a search procedure) generates correct results in all cases [11]. A similar SAT+CAS approach has also successfully been used [3] to generate proof certificates verifying Lam et al.'s experimental proof of the nonexistence of projective planes of order ten [13].

2 Background

The objects we will be interested in this paper are primarily finite projective planes, which we introduce here. These projective planes have a few different representations, the first of which is perhaps the easiest to understand. After giving this as the definition, we discuss another representation which will be more useful for the purposes of doing an exhaustive computer search for these objects. Also playing a role in this work is the notion of a latin square which we define here. Lastly, we describe the notion of a Boolean satisfiability problem or SAT problem.

To start, we define a projective plane of a given order n:

Definition 1. *A* projective plane of order n *is a collection of* $n^2 + n + 1$ *lines and* $n^2 + n + 1$ *points such that:*

(1) every line contains $n + 1$ *points,*
(2) every point is on $n + 1$ *lines,*
(3) any two distinct lines intersect at exactly one point, and
(4) any two distinct points lie on exactly one line.

From this definition, we can see that projective planes are objects which have a natural interpretation as an incidence structure—that is, two disjoint sets equipped with a relation between them which we call the incidence relation [5]. One of the simplest examples of this is the *Fano plane*, which is a projective plane of order 2 (i.e., it has $2^2 + 2 + 1 = 7$ points and lines) as seen in Figure 1. With such structures, we may associate a bipartite graph, whose parts in our case are the lines and points of the projective plane respectively, and whose edge set is determined by the incidence relation. Specifically, in this bipartite graph, we make an edge between a point and a line if and only if that point lies

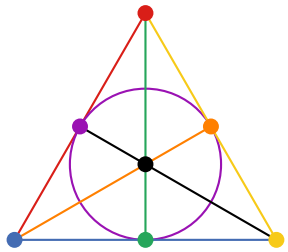

Fig. 1. The Fano Plane. The points are represented by dots and the lines of the plane are represented by straight or curved lines between the points.

on the line. Suppose now that we order the points and lines of the projective plane as: $p_1, p_2, \ldots, p_{n^2+n+1}$ and $\ell_1, \ell_2, \ldots, \ell_{n^2+n+1}$. Then we may represent the associated bipartite graph as an $(n^2+n+1) \times (n^2+n+1)$ binary *incidence matrix*, in which each row represents a point and each column represents a line. The (i, j)th entry of this matrix will be 1 if the point p_i lies on the line ℓ_j, and it will be 0 otherwise. This definition is more workable for the purposes of using a SAT solver as we can encode the incidence matrix as $(n^2+n+1)^2$ Boolean variables. To be able to utilize this representation, we must also translate the axioms of the projective plane (1)–(4) given above in terms of the incidence matrix. Two binary vectors are said to *intersect* if they both contain 1s in the same location. One can check that an $(n^2+n+1) \times (n^2+n+1)$ incidence matrix is one which arises from a projective plane of order n if the following properties are satisfied:

(1) each column sum of the matrix is $n+1$,
(2) each row sum of the matrix is $n+1$,
(3) two distinct columns of the matrix intersect exactly once, and
(4) two distinct rows of the matrix intersect exactly once.

Next, we also introduce latin squares.

Definition 2. *A* $k \times k$ latin square *is a* $k \times k$ *array consisting of the integers* $1, 2, \ldots, k$ *such that:*

1. *each row contains each of the numbers* $1, 2, \ldots, k$ *exactly once, and*
2. *each column contains each of the numbers* $1, 2, \ldots, k$ *exactly once.*

For example, the following is a latin square of order 4:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{bmatrix}$$

The role that latin squares play in our search for projective planes will become more clear after our discussion in section 2.2.

Lastly, we describe SAT problems—which are useful because heuristic algorithms exist that often solve SAT instances extremely efficiently. A *literal* is a Boolean variable or a negated Boolean variable, and a *clause* is a disjunction of literals. Given a list of clauses, we say the list is *satisfiable* if we can assign true and false values to its variables such that every clause evaluates to be true. A *SAT problem* is the problem of determining if a given list of clauses is satisfiable or not. This problem is known to be NP-complete and there is no known polynomial time algorithm solving it. However, there are good heuristic SAT solvers like MapleSAT [14] which can often solve SAT instances.

2.1 Problem Overview

With this background established, we now give more details about the problem we're interested in. As we discussed, we can represent a hypothetical projective plane of order n as an $(n^2+n+1)\times(n^2+n+1)$ incidence matrix satisfying certain properties. For the order 9 case, we're interested in enumerating the 91×91 incidence matrices (note: $91 = 9^2+9+1$) satisfying certain properties up to a certain symmetry. We now comment on what this symmetry is.

Notice that given a fixed projective plane of order 9, one can obtain a distinct, but similar, projective plane by relabelling some of the points, and relabelling some of the lines. Realistically however, these "new" planes are the same as the original one, thus we wish to ignore these extra planes in our search if possible. These relabellings correspond to column and row permutations of the incidence matrix. Consequently, we can give a group theoretic interpretation of this: Let G be the group $S_{91}\times S_{91}$. We may view the first component of this group as the row permutations, and the second as the column permutations. In this way, we have a group action of G on the set of incidence matrices corresponding to projective planes. Then we can say that two planes are equivalent if they are in the same *orbit* under this action. Two planes are in the same orbit, if one can be obtained from the other via action of the group G, or rather, by applying column and row permutations.

Searching for only a representative of each orbit makes our search drastically more feasible. This computational search was first done by Lam, Kolesova, and Thiel [12]. Before this search was done, there were four distinct projective planes of order 9 known to exist. Lam et al.'s search showed that these were the only four planes up to isomorphism. We repeat this search using a SAT solver and by exploiting the symmetry group mentioned above to reduce the search space. By applying the elements of the symmetry group to a hypothetical projective plane, we can fix certain structure for the plane. This is described in Section 2.2.

2.2 Structure of Planes of Order 9

Here we detail the structure we impose on the incidence matrix of our hypothetical projective plane for the purposes of reducing the search space of our problem. The first important thing to note is that a hypothetical projective plane must contain a *triangle*—that is, a set of three non-collinear points. We may suppose

that the first three points p_1, p_2, and p_3 form this triangle, and furthermore that ℓ_1 is the unique line joining p_2 and p_3, ℓ_2 is the unique line joining p_1 and p_2, and lastly that ℓ_3 is the unique line joining p_1 and p_3. We may impose this order simply by applying the needed row and column permutations to the incidence matrix. With this ordering of the points and lines, the upper left 3×3 sub-matrix of the incidence matrix will look like:

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

Once this is done, we know by the first two axioms of a projective plane, that there should be 8 more 1's in each of the first three rows and columns. By applying further row and column permutations, we may impose a staircase like structure on these entries. For example, we can make the first three rows in columns 4 to 27 will look like:

$$\begin{bmatrix} 1\,1\cdots 1\,0\,0\cdots 0\,0\,0\cdots 0 \\ 0\,0\cdots 0\,1\,1\cdots 1\,0\,0\cdots 0 \\ 0\,0\cdots 0\,0\,0\cdots 0\,1\,1\cdots 1 \end{bmatrix}$$

with the entries in columns 28 and above being 0 in those rows.

Then, using further column and row permutations, we may fix many of the entries in the first 27 columns of the incidence matrix shown in Figure 2. The entries in the submatrix formed by columns 20–27 and rows 28–91 will not be fixed but they do have a nice structure.

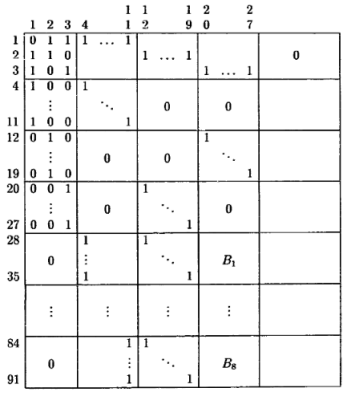

Fig. 2. Form of the normalized incidence matrix from Lam's 1991 paper.

One can check that the 8×8 blocks B_1, B_2, ..., B_8 given in Figure 2 will be 8×8 permutation matrices. Consequently, these blocks correspond to permutations π_1, π_2, ..., $\pi_8 \in S_8$. Furthermore, we know that we can't have $\pi_i(k) = \pi_j(k)$

where $i \neq j$ since otherwise we would have two distinct rows intersecting twice. This property ensures that we may encode this set of 8 permutations as an 8×8 latin square. A more detailed explanation of this normalized form for the partial plane can be found in Kolesova's thesis [10, Prop. 4.1].

Thus, for a given 8×8 latin square, we may obtain the 27 columns of a partial projective plane, and in this way, all such partial planes may be obtained if all 8×8 latin squares can be enumerated. Thus, our approach for generating the projective planes of order 9 will begin by generating the latin squares of order 8. However, two distinct latin squares can give equivalent partial planes. Thus we will be interested in equivalence classes of latin squares as detailed in the next section.

3 Enumeration Method

Our enumeration method proceeds in two steps. First, all 8×8 latin squares are enumerated up to isomorphism using a SAT+CAS method. Second, for every 8×8 latin square generated in the first step the initial 27 columns of a projective plane of order nine are defined in terms of the structure revealed in Section 2.2. For each possible way of completing the first 27 columns a SAT instance is created which is satisfiable exactly when the 27 columns can be extended to 40 columns of a projective plane.

3.1 Step 1: Latin Squares

As the structure we identified in the last section suggests, a good first step to generate the projective planes of order 9 will be to first generate all 8×8 latin squares up to a certain equivalence. More specifically, we will generate a representative of each *main class* of latin squares. Two 8×8 latin squares are said to belong to the same main class if they differ only by a permutation of the rows, columns, or symbols, or also possibly an exchange of the roles of these three things. Thus we can view the main class of a latin square as its orbit under the natural action of the group $(S_8 \times S_8 \times S_8) \rtimes S_3$ on the set of latin squares. To generate all such representatives, we reduce the problem to SAT, and then utilize a SAT solver to find all solutions of the instance. The exhaustive search is performed by adjoining a "solution blocking clause" to the instance whenever a solution is found. To reduce the computation, we also provide a mechanism for doing isomorphism checking along the way; if two partial squares belong to the same main class, we throw away the duplicates to reduce the computation time.

The reduction of the latin square problem into SAT is well known, and can be found in a variety of sources [4, 7]. Before utilizing the SAT solver, we first normalize the latin squares by insisting that the numbers 1, 2, ..., 8 appear in order in the first column and row. This is done by adding unit clauses specifying that the entries in the first row have such values. This is justified since we may apply column and row permutations to impose this structure. With this done, we use MapleSAT to extend one row at a time up to row 4 of the latin square,

and then from row 4 to row 8. After each row extension, isomorphism removal is also done.

Isomorphism removal is done by translating our latin squares to graphs and then checking if the corresponding graphs are isomorphic to each other using pynauty [15]. We do this in such a way that the corresponding graphs will be isomorphic if and only if the original latin squares belonged to the same main class. Our translation of a latin square $(A_{i,j})$ to a graph is as follows: The vertex set is $\{ V_{i,j} : 1 \leq i, j \leq n \}$ and we draw an edge between $V_{i,j}$ and $V_{i',j'}$ if $i = i'$, $j = j'$, or $A_{i,j} = A_{i',j'}$. More details of this construction and its validity can found in Miller's paper [16].

Before performing the final isomorphism removal after generating all 8 rows of the latin squares, there were about 44 million solutions found by MapleSAT. Once the isomorphism removal is applied, there were 283,657 representatives of the main classes of latin squares which remained, which serve as the starting point for the next step in our computation. The main classes of latin squares of order 8 have been enumerated by others including Lam et al. [9] and the number of latin squares produced by our computation matches the number previously reported. Our computation took about 147 hours (6.125 days) on a single CPU core, with the majority of this time taken up by the final isomorphism removal step (4.84 days).

3.2 Step 2: Column 40 Extension

The next major step in our enumeration of the projective planes to extend each of the 283,657 partial planes (each with 27 columns to start) to partial planes of 40 columns. We accomplish this by once again using a SAT solver. However, we take a slightly different approach than we did with the latin squares since the axioms of a projective plane don't translate as nicely into SAT. A partial plane $(A_{i,j})$ has a natural encoding as a set Boolean variables by defining a variable $a_{i,j}$ for each entry. We encode in SAT only axioms (3) and (4)—namely, that two distinct rows or columns intersect exactly once. First, the requirement that they intersect at most once is given by the *quadfree* clauses:

$$\neg a_{i,j} \vee \neg a_{i',j} \vee \neg a_{i,j'} \vee \neg a_{i',j'}$$

for $i < i'$ and $j < j'$. If one of these clauses is false this means there is a rectangle in the incidence matrix whose corners are 1s—which is exactly what happens if two rows or columns intersect more than once.

Then, we check that a given column intersects the first 27 columns of the partial plane each at least once. Note that because the first 19 columns are the only ones which are fixed among all partial planes, we will need a different list of clauses for each starting point of the extension. If column $j \leq 27$ has 1s in the entries (i_1, j), (i_2, j), $\ldots$, (i_{10}, j), then for $j' > 27$, we include the clause

$$a_{i_1,j'} \vee a_{i_2,j'} \vee \cdots \vee a_{i_{10},j'}$$

which ensures that column j' intersects column j at least once. In addition to these clauses, we also use unit clauses to impose some structure on the first 19

rows (an exact transpose of the first 19 columns in fact) in order to remove symmetry.

This is currently work in progress but in practice the extension takes about 0.8 seconds for each 27-column partial plane, resulting in an estimated total CPU time of under 78 hours. The few 40-column partial planes that are generated can typically be extended to a full 91 columns (or shown not to complete at all) almost instantly. The final step will be to perform isomorphism checking on each of the complete projective planes found by the solver.

4 Verification

The SAT solver MapleSAT returns DRAT proofs which can be checked using a proof verifier such as DRAT-trim [17]. This way only the proof verifier—which is much simpler than the SAT solver—needs to be trusted.

In step 1, we do several row extensions with MapleSAT to produce the final list of latin squares. For each of these, will generate and verify the proofs for these steps. This involves adding a blocking clause for each solution of the SAT instance and then showing a conflict can be derived from the original list of clauses in conjunction with the blocking clauses. In step 2, we have a separate list of clauses for each of the 27 column partial planes, thus we end up with a separate proof for each one, all of which need to be verified independently just as in step 1. Lastly, we must also do this for the final extension from column 40 to the full 91 columns.

In addition to verifying the proofs above, we will verify the correctness of the isomorphism removal which is done at several stages. To do so, when an isomorphism removal is done, we will store a relabelling (provided by pynauty) of the graph which shows that an isomorphic copy of that graph remains in the reduced list.

5 Conclusion

This method of enumerating of the projective planes of order nine relies on two main components: the generation of solutions with our SAT solver (MapleSAT), and the isomorphism removal of partial solutions with our CAS (pynauty). Both components are crucial—if either component were removed this work would not be feasible to complete in a reasonable amount of time. To the best of our knowledge, the resulting enumeration will be the first independent verification of the search of Lam, Kolesova, and Thiel [12]. Moreover, an enumeration using a SAT+CAS system can be trusted to a higher degree of certainty than an enumeration via custom-written search code. Our enumeration has been designed to require trusting a *certificate verifier* rather than trusting a *search procedure*.

References

1. Abraham, E.: Building bridges between symbolic computation and satisfiability checking. In: Proceedings of the 2015 ACM on International

Symposium on Symbolic and Algebraic Computation. ACM (Jun 2015). https://doi.org/10.1145/2755996.2756636
2. Bradford, R., Davenport, J.H., England, M., Sadeghimanesh, A., Uncu, A.: The DEWCAD project: pushing back the doubly exponential wall of cylindrical algebraic decomposition. ACM Communications in Computer Algebra **55**(3), 107–111 (Sep 2021). https://doi.org/10.1145/3511528.3511538
3. Bright, C., Cheung, K.K.H., Stevens, B., Kotsireas, I., Ganesh, V.: A SAT-based resolution of Lam's problem. In: Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence. pp. 3669–3676 (2021), `https://ojs.aaai.org/index.php/AAAI/article/view/16483`
4. Bright, C., Gerhard, J., Kotsireas, I., Ganesh, V.: Effective problem solving using SAT solvers. In: Communications in Computer and Information Science, pp. 205–219. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-41258-6_15
5. Godsil, C., Royle, G.F.: Algebraic Graph Theory. Springer Science & Business Media (2001)
6. Heule, M.J.H., Kauers, M., Seidl, M.: New ways to multiply 3×3-matrices. Journal of Symbolic Computation **104**, 899–916 (May 2021). https://doi.org/10.1016/j.jsc.2020.10.003
7. Jin, J., Lv, Y., Ge, C., Ma, F., Zhang, J.: Investigating the existence of costas latin squares via satisfiability testing. In: Theory and Applications of Satisfiability Testing – SAT 2021, pp. 270–279. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-80223-3_19
8. Kaufmann, D., Biere, A., Kauers, M.: SAT, computer algebra, multipliers. In: EPiC Series in Computing. EasyChair. https://doi.org/10.29007/j8cm
9. Kolesova, G., Lam, C.W.H., Thiel, L.: On the number of 8×8 Latin squares. Journal of Combinatorial Theory, Series A **54**(1), 143–148 (May 1990). https://doi.org/10.1016/0097-3165(90)90015-o
10. Kolesova, G.I.: Enumeration of the Finite Projective Planes of Order Nine. Master's thesis, Concordia University (1989)
11. Lam, C.W.H.: Opinion. The Mathematical Intelligencer **12**(1), 8–12 (Dec 1990). https://doi.org/10.1007/bf03023977
12. Lam, C.W.H., Kolesova, G., Thiel, L.: A computer search for finite projective planes of order 9. Discrete Mathematics **92**(1-3), 187–195 (Nov 1991). https://doi.org/10.1016/0012-365x(91)90280-f
13. Lam, C.W.H., Thiel, L., Swiercz, S.: The non-existence of finite projective planes of order 10. Canadian Journal of Mathematics **41**(6), 1117–1123 (Dec 1989). https://doi.org/10.4153/cjm-1989-049-4
14. Liang, J.H., Ganesh, V., Poupart, P., Czarnecki, K.: Learning rate based branching heuristic for SAT solvers. In: Theory and Applications of Satisfiability Testing – SAT 2016, pp. 123–140. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-40970-2_9
15. McKay, B.D., Piperno, A.: Practical graph isomorphism, II. Journal of Symbolic Computation **60**, 94–112 (Jan 2014). https://doi.org/10.1016/j.jsc.2013.09.003
16. Miller, G.L.: On the $n^{\log n}$ isomorphism technique (a preliminary report). In: Proceedings of the tenth annual ACM symposium on Theory of computing - STOC '78. ACM Press (1978). https://doi.org/10.1145/800133.804331
17. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In: Lecture Notes in Computer Science, pp. 422–429. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-09284-3_31