Classification of some strongly regular subgraphs of the McLaughlin graph

J. Degraer [*], K. Coolsaet

Department of Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281–S9, B–9000 Ghent, Belgium

Abstract

By means of an exhaustive computer search we have proved that the strongly regular graphs with parameters $(v, k, \lambda, \mu) = (105, 32, 4, 12)$, $(120, 42, 8, 18)$ and $(176, 70, 18, 34)$ are unique upto isomorphism. Each of these graphs occurs as an induced subgraph in the strongly regular McLaughlin graph. We have used an orderly backtracking algorithm with look-ahead and look-back strategies, applying constraints based on several combinatorial and algebraic properties of graphs with the given parameters.

Key words: Computer classification, Strongly regular graph, Orderly generation
1991 MSC: 05E30, 05–04

1 Introduction

Let Γ be a simple undirected graph with vertex set $V = \{1, \ldots, v\}$. We shall write $p \sim q$ to indicate that two vertices $p, q \in V$ are adjacent (and distinct) and $p \not\sim q$ to indicate that they are not adjacent (and distinct). If p is a vertex of Γ then the *neighbourhood graph* $\Gamma(p)$ with respect to p is the subgraph of Γ induced by all the vertices that are adjacent to p.

A graph Γ is called *regular* of degree k when every vertex of Γ has exactly k common neighbours. Γ is called *strongly regular* with parameters (v, k, λ, μ) when it has v vertices, is regular of degree k and moreover each pair of adjacent

[*] Corresponding author.
Email addresses: `Jan.Degraer@UGent.be` (J. Degraer), `Kris.Coolsaet@UGent.be` (K. Coolsaet).

vertices has the same number λ of common neighbours and each pair of non-adjacent vertices has the same number μ of common neighbours. We shall call a strongly regular graph with given parameters *unique* if and only if all strongly regular graphs with these parameters are isomorphic.

The neighbourhood graph of a vertex p of a strongly regular graph Γ is also called a *first subconstituent* of Γ. The subgraph of Γ induced on all vertices of Γ which are not adjacent to (and different from) p, is called a *second subconstituent*.

The *McLaughlin* graph [6] is the well known unique strongly regular graph with $(v, k, \lambda, \mu) = (275, 112, 30, 56)$. This graph contains many induced subgraphs which are again strongly regular (for example, the first and second subconstituents of the graph). For four of the corresponding parameters sets, i.e., $(105, 32, 4, 12)$, $(120, 42, 8, 18)$, $(176, 70, 18, 34)$ and $(253, 112, 36, 60)$, the strongly regular graph is not known to be unique. In this paper we report on an exhaustive computer search which settles the uniqueness question in the first three cases.

2 Backtrack search with isomorphism rejection

2.1 General strategy

For a simple graph Γ we use the following computer representation : vertices are represented by the consecutive integers $1, \ldots, v$ and adjacency information is stored in a square symmetric $v \times v$ matrix M, where rows an columns are indexed by the vertices of the graph and the entries $(M)_{xy}$ are defined as follows :

$$(M)_{xy} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x \sim y, \\ 2 & \text{if } x \not\sim y. \end{cases}$$

The exhaustive search algorithm basically generates all matrices of this kind in a recursive way, trying each of the two possible values from the *domain* $\{1, 2\}$ for each of the (upper diagonal) matrix entries, and filters out those that do not satisfy the definition of a strongly regular graph with a given parameter set.

Different techniques are used to do this in an intelligent way and make the program complete its search within reasonable time. Most of them amount to 'pruning' the search tree at points where M is only partially instantiated, either because we know that it will never be possible to complete it to a strongly regular graph with the requested properties, or because the partially instantiated matrix can be proved isomorphic to an instance we have already considered earlier in the search.

Also the order in which the various matrix entries are filled in may influence the speed of the algorithm dramatically. We use the *column-by-column* order

$$(x, y) = (1, 2), (1, 3), (2, 3), (1, 4), (2, 4), (3, 4), \dots, (v-2, v), (v-1, v)$$

but deviate from this when certain domain values are 'forced' (see §2.5 below).

2.2 Combinatorial constraints

Several properties of strongly regular graphs can be used to prune the search tree. As a strongly regular graph is regular of degree k, we may prune the search as soon as a row or column of M contains more than k entries equal to 1 (denoting adjacency). Similarly, no row or column may contain more than $v - k - 1$ entries equal to 2 (denoting non-adjacency) as the complement of the graph is regular of degree $v - k - 1$.

As a direct consequence of the definition of a strongly regular graph we may also prune the search whenever a partially filled matrix M has two adjacent vertices that have more than λ vertices in common or two non-adjacent vertices with more than μ common neighbours. Moreover, similar restrictions can be deduced for the cardinality of the set of vertices z that are adjacent to p but not to q, adjacent to q but not to p and adjacent to neither p nor q. We list the corresponding numbers in the tables below:

When $p \sim q$:

	$z \sim q$	$z \not\sim q$
$z \sim p$	λ	$k - \lambda - 1$
$z \not\sim p$	$k - \lambda - 1$	$v - 2k + \lambda$

When $p \not\sim q$:

	$z \sim q$	$z \not\sim q$
$z \sim p$	μ	$k - \mu$
$z \not\sim p$	$k - \mu$	$v - 2k + \mu - 2$

When M is not completely filled in, we do not always know whether p and q are adjacent or not, and then the information above cannot be used directly. However, in that case we may still use a *weaker* constraint : the numbers are then bounded by the maximum of the corresponding bounds for $p \sim q$ and $p \not\sim q$. When eventually the (non) adjacency of p and q is determined, we check the constraint again for the appropriate bound.

2.3 Algebraic constraints

The 0–1 adjacency matrix A of a strongly regular graph has some interesting *algebraic* properties which can be used to speed up the search. (We refer to [1] for more information on how these properties are derived.)

A has exactly three eigenvalues. Their values θ_0, θ_1 and θ_2 and the corresponding multiplicities m_0, m_1 and m_2 only depend on the parameters (v, k, λ, μ). The table below lists these eigenvalues and multiplicities for the three graphs under consideration (see also [2]).

(v, k, λ, μ)	$\theta_0^{m_0}$	$\theta_1^{m_1}$	$\theta_2^{m_2}$
$(105, 32, 4, 12)$	32^1	2^{84}	-10^{20}
$(120, 42, 8, 18)$	42^1	2^{99}	-12^{20}
$(176, 70, 18, 34)$	70^1	2^{154}	-18^{21}

For each of the eigenvalues θ_i we may define a correspondig *minimal idempotent* matrix E_i. Each E_i can be expressed as a linear combination of the form $\delta_0 I + \delta_1 A + \delta_2 A_2$ where each of the coefficients δ_i again only depends on the parameters of the strongly regular graph. (Here I denotes the identity matrix, A is the adjacency matrix and A_2 is the adjacency matrix of the complement of the graph.) In other words, each idempotent corresponds to a square symmetric $v \times v$ matrix with values δ_0 on the diagonal, δ_1 on positions that correspond to adjacent vertices of the graph and δ_2 elsewhere.

We list the minimal idempotents E_1 and E_2 for the relevant graphs in the table below. (The idempotent E_0 has $\delta_0 = \delta_1 = \delta_2 = 1/v$.)

(v, k, λ, μ)	E_1	E_2
$(105, 32, 4, 12)$	$\frac{4}{5}I + \frac{1}{20}A - \frac{1}{30}A_2$	$\frac{4}{21}I - \frac{5}{84}A + \frac{1}{42}A_2$
$(120, 42, 8, 18)$	$\frac{33}{40}I + \frac{11}{280}A - \frac{9}{280}A_2$	$\frac{1}{6}I - \frac{1}{21}A + \frac{1}{42}A_2$
$(176, 70, 18, 34)$	$\frac{7}{8}I + \frac{1}{40}A - \frac{1}{40}A_2$	$\frac{21}{176}I - \frac{27}{880}A + \frac{17}{880}A_2$

As in [3], we make use of two important properties of these minimal idempotents E_i in order to prune the search: they have *rank* m_i and they are *positive semidefinite*. Consequently, every principal submatrix of E_i must have rank $\leq m_i$ and must also be positive semidefinite.

The program checks these constraints for every top left principal submatrix of M. Because of the column-by-column instantiation order we use, large top left principal submatrices turn up fairly early in the search process.

2.4 Isomorphism rejection

In order to detect partially filled matrices that are isomorphic to partially filled matrices considered earlier in the search we use a so-called *orderly* approach. An orderly algorithm [9] rejects matrices during the search if they are not in *canonical form*.

Define a certificate $\mathcal{C}(M)$ for a symmetric $v \times v$ matrix M to be the string of length $v(v-1)/2$ obtained by concatenating the upper diagonal entries of M in *column-by-column* order. I.e.,

$$\mathcal{C}(M) = (M)_{12}\ (M)_{13}\ (M)_{23}\ (M)_{14}\ (M)_{24}\ \cdots$$

Note that the certificate for a top left principal submatrix of M is a prefix of $\mathcal{C}(M)$.

The standard lexical (lexicographical) ordering on strings can now be used to define a total ordering on symmetric matrices of the same order, as follows : define $M < M'$ if and only if $\mathcal{C}(M) < \mathcal{C}(M')$.

Now consider the set $\{M^\pi | \pi \in \mathrm{Sym}(v)\}$ of matrices that can be obtained from M by applying every possible permutation π of the rows (and simultaneously of the columns) to M, or equivalently, the set of matrices that represent the same graph as M but use a (possibily) different numbering of the vertices. In this set, the matrix M' for which $\mathcal{C}(M)$ is smallest shall be called the *canonical* form of M. If M is equal to its canonical form, we say that M is *in canonical form*. (It should be noted that not every author and not every algorithm uses the same definition of certificate and hence of canonical form. Our choice is a consequence of the column-by-column search order we use.)

The exhaustive search now uses the orderly approach in the following manner : whenever a top left principal submatrix of M is fully instantiated, we check whether that submatrix is in canonical form. If not, we prune the search. Because computing the canonical form of a matrix is in general a very time consuming task, we have introduced two additional criteria to speed up this test : *lexical ordering* of the rows of M and *clique checking*.

2.4.1 Lexical ordering on rows

We say that M is *lexically ordered* if for every $i \in 1, \ldots, v-1$ the i-th row of M, seen as a string of length v, is lexically smaller than the $i+1$-th row or equal to it. Note that every top left principal submatrix of a lexically ordered matrix must itself be lexically ordered.

Lemma 1 *Let M be a symmetric $v \times v$ matrix with zero diagonal and every non-diagonal entry ≥ 0. If M is in canonical form, then M is lexically ordered.*

Proof : Assume M is not lexically ordered. And let i be the smallest row number for which the $i+1$-th row of M is lexically smaller than the i-th row of M. Let j be the smallest column number for which those rows differ, i.e., for which $M_{ij} > M_{i+1,j}$, and hence, by symmetry, $M_{ji} > M_{j,i+1}$. Because M has zero diagonal, we must have $j < i$. But then it is easily seen that the transposition π which interchanges the i-th and $i+1$-th row (and column) maps M onto a matrix M^π with a smaller certificate. Hence M is not in canonical form. $\square$

Checking whether a matrix is lexically ordered can be done much faster than computing its canonical form. Hence, before checking whether a top left principal submatrix is canonical, we first check whether it is lexically ordered.

2.4.2 Clique checking

A *clique* of a graph Γ is a subset of the vertices of Γ such that every two distinct vertices in this set are adjacent.

Lemma 2 *A graph Γ contains a clique of size s if and only if the canonical form of the corresponding matrix M has a top left principal submatrix of order $s \times s$ with all non-diagonal entries equal to 1.*

Proof : If the top left principal $s \times s$ submatrix of M has the stated form, then by definition of M the vertices of Γ numbered $1, \ldots, s$ are mutually adjacent and hence form a clique.

If the top left principal $s \times s$ submatrix of M does not have the stated form, then $\mathcal{C}(M)$ contains at least one 2 within the first $s(s-1)/2$ positions. If then Γ has a clique of size s and we renumber the vertices of Γ in such a way that the clique vertices are numbered $1, \ldots, s$, we obtain a matrix M^π for which the string $\mathcal{C}(M^\pi)$ starts with $s(s-1)/2$ ones. Hence $\mathcal{C}(M^\pi) < \mathcal{C}(M)$, and M is not in canonical form. $\square$

We apply this lemma in the following way : if at a certain point in the search process M has a completely instantiated top left principal submatrix of order $s \times s$ which correponds to a clique, while the top left principal submatrix of order $s+1 \times s+1$ does not, then we may prune the search if we detect a clique of size $s+1$ elsewhere in the graph.

In general, searching for cliques in a graph is a costly operation. However for the three parameter sets we consider, the maximal clique size can be proved to be at most 4 (e.g., using the *Hoffman bound* [1]). Searching for cliques of such small size is not very expensive, the more so as we already keep track of the number of common neighbours of any two points for the sake of the combinatorial constraints (cf. §2.2).

2.5 Further refinements

Apart from pruning the search tree also other techniques contribute to the overall speed of the algorithm.

One of the methods which has proven to be a big time saver, is the application of a *look-ahead* strategy [5,7,8]. Basic backtracking always first assigns a value 1 or 2 to a matrix entry and then applies the constraints to check whether this instantiation does not produce any inconsistencies with the partially filled matrix obtained so far. In a look-ahead strategy we keep track at every position in the matrix of all domain values that are still allowed for that position. After every recursion step we adjust the domain of selected matrix entries by temporarily removing values from their domains which are inconsistent with the partially filled matrix M constructed so far.

For example, when we introduce the k-th entry with value 1 in a given row, we remove 1 from the domain of all entries in the same row which have not yet been given a value. This is a look-ahead version of the degree constraint described in §2.2. A similar strategy can be applied for the other combinatorial constraints. For more information on the specific domain reduction rules used for selection of the set of matrix entries to be considered and the removal of inconsistent domain values, we refer to [4].

For the look-ahead strategy to be effective, we need to introduce two changes into the backtrack algorithm. First, it may happen that at some point the domain of an uninstantiated matrix entry becomes empty. In that case the search tree can be pruned immediately. Without look-ahead this inconsistentcy would only be noted when the corresponding matrix position was actually reached during the search, which might be a lot further down the search tree.

Secondly, we may use domain information to deviate from the column-by-column search order whenever a domain has size 1. In other words, when a certain value is *forced* for a certain matrix entry, we instantiate that element first. This requires some extra bookkeeping, but again it can have a substantial impact on the running time of the algorithm.

Finally, to improve the clique search (cf. §2.4) we use a *look-back* strategy. Suppose that the instantiation of the matrix entry M_{xy} completes a clique of size $s + 1$ and this enables us to prune the search. Then this instantiation remains forbidden (this forces a non-adjacency) until a backtrack occurs to the second last instantiated matrix entry that contributed to the same clique.

3 Results and final remarks

As stated in the introduction, the main result of our computer search is the following

Theorem 1 *The three strongly regular graphs with parameters* $(v, k, \lambda, \mu) = (105, 32, 4, 12)$*,* $(120, 42, 8, 18)$ *and* $(176, 70, 18, 34)$ *are uniquely determined by their parameters (upto isomorphism).*

The algorithm was implemented in the programming language Java and the search was carried out on a single CPU with a clock speed of approximately 1 GHz. We list the time needed to perform the exhaustive searches in the table below.

(v, k, λ, μ)	CPU time
$(105, 32, 4, 12)$	5 s
$(120, 42, 8, 18)$	13 m 40 s
$(176, 70, 18, 34)$	11 h 45 m 10 s

To double check the computer results, we tested the program on various other parameter sets of strongly regular graphs for which the results were already known before.

Finally we would like to mention that we had not expected to be able to tackle graphs of these sizes. Former computer enumerations of strongly regular graphs by ourselves and by different authors were restricted to graphs of size $v = 64$ or less and took many computers and a lot more time to complete. We think that the refinements introduced in section §2.5 are largely responsible for our success and hope that applying the same techniques to other parameters sets will lead to further classification results in the near future.

4 Acknowledgements

We would like to thank Frank De Clerck for suggesting to apply our techniques to the strongly regular graphs studied in this paper.

References

[1] Brouwer A. E., A. M. Cohen & A. Neumaier, *Distance-Regular Graphs*, Ergeb. Math. Grenzgeb. (3) **18**, Springer-Verlag, Berlin (1989).

[2] Brouwer A. E., Strongly Regular Graphs, in *The CRC handbook of Combinatorial Designs*, Chap. VI.5, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (1996).

[3] Coolsaet K. & J. Degraer, A computer assisted proof of the uniqueness of the Perkel graph, to be published in *Designs, Codes and Cryptography*.

[4] Degraer J. & K. Coolsaet, Classification of three-class association schemes using backtracking with dynamical variable ordering, submitted to *Journal of Discrete Mathematics*.

[5] Gibbons P. B., Computational Methods in Designs Theory, in *The CRC handbook of Combinatorial Designs*, Chap. VI.9, Ch. J. Colbourn, J. H. Dinitz (eds.), CRC Press, Boca Raton (1996).

[6] Goethals J.-M., J. J. Seidel, The regular two graph on 276 vertices, *Discrete Math.* **12** (1975),143–158.

[7] Haralick R., Elliott G., Increasing tree search efficiency for constraint-satisfaction problems, *Artificial Intelligence* **14** (3) (1980), 216–313.

[8] Kumar V., Algorithms for Constraint Satisfaction Problems: A Survey, *AI Magazine* **13(1)** (1992), 32–44;

[9] Read R.C., Every one a winner, or, how to avoid isomorphism search when cataloguing combinatorial configurations, *Ann. Discrete Math.* **2** (1978), 107–120.