Isomorphism for Digraphs and Sequences of Shop Scheduling Problems*

Heidemarie Bräsel Martin Harborth Per Willenius

Otto-von-Guericke-University Magdeburg
Faculty of Mathematics
Institute for Algebra and Geometry
PF 4120, D-39016 Magdeburg, Germany
Email: harborth@mathematik.uni-magdeburg.de

Abstract

The computational complexity of the graph isomorphism problem is still unknown. We consider Cartesian products $K_n \times K_m$ of two complete graphs K_n and K_m. An acyclic orientation of such a Cartesian product is called a sequence graph because it has an application in production scheduling. It can be shown that the graph isomorphism problem on the class of these acyclic digraphs is solvable in polynomial time. We give numbers of non-isomorphic sequence graphs for small n and m. The orientation on the cliques of a sequence graph can be interpreted as job orders and machine orders of a shop scheduling problem with a complete operation set.

Keywords: complexity; graph isomorphism problem; acyclic digraphs; Latin rectangles; sequences; open shop scheduling

*supported by Deutsche Forschungsgemeinschaft (DFG), Project ScheMA

1 Introduction

Much effort has been made to find efficient algorithms for the *graph isomorphism problem*; i.e., the problem of deciding whether two given finite graphs are isomorphic. Up to now no fast (i.e., polynomial-bound) algorithm for this problem has been developed and it is unknown if such an algorithm can exist.

Informally, a decision problem is said to be in P if there is a deterministic algorithm which solves the problem in polynomial time. A decision problem is in NP if a positive solution can be verified in polynomial time (i.e., it exists a nondeterministic polynomial algorithm). A decision problem in NP is called NP-complete if it has the following property: if the problem belongs to P then all problems in NP also belong to P (i.e., $P = NP$). See [6] for precise notation and an overview of the area. The main open question in complexity theory is if $P = NP$ holds. An interesting role in this question plays the graph isomorphism problem which is known to be in NP, but it is not known to be NP-complete, neither is it known to be in P.

For certain classes of graphs the computational complexity of graph isomorphism has been determined. For example, MILLER [13] showed that the isomorphism problem for undirected graphs generated from Latin squares is decidable in $O(n^{\log n+O(1)})$ time. Polynomial-time algorithms for the graph isomorphism problem have been developed for some other classes of graphs, e.g., planar graphs (HOPCROFT and WONG [8]), graphs of bounded valence (LUKS [11]), cyclic tournaments (PONOMARENKO [15]), and graphs of bounded average genus (CHEN [3]). Clearly, the isomorphism problem for undirected graphs is polynomial-time reducible to a corresponding isomorphism problem for directed graphs since each undirected edge can be replaced by two anti-parallel arcs. On the other hand, there exists also an opposite reduction, see MILLER [14]. Therefore, the problems of undirected graph isomorphism and directed graph isomorphism are polynomially equivalent.

The goal of this paper is to study a restriction on an additional class of graphs in order to get a polynomial-time algorithm for the corresponding isomorphism problem. We consider acyclic digraphs generated from Hamming graphs. These digraphs can be viewed as sequences of open shop scheduling problems with a complete operation set.

An open shop scheduling problem is characterized by a set $\{J_1, \dots, J_n\}$ of jobs and a set $\{M_1, \dots, M_m\}$ of machines. Each job consists of operations where the operation of job J_i on machine M_j has to be processed for the processing time p_{ij}. In general, some processing times may be zero and corresponding operations are considered as nonexistent but we assume that all operations do exist; i.e., we have a complete operation set. A machine can always work on at most one job at a time and no two operations of the same job can be processed simultaneously. Each machine order (i.e, the order on the operations belonging to the same job) and each job order (i.e., the order on the operations belonging to the same machine) can be chosen arbitrarily. A feasible combination of all machine orders and job orders is called a sequence. A schedule contains the completion times of all operations. The time when the last operation of job J_i is finished is the completion time C_i of job J_i. The maximum completion time $C_{\max} = \max_i\{C_i\}$ is called the makespan of a schedule. The problem is to find a schedule with minimum makespan. This problem is studied in various ways (e.g. see [7]). A sequence is optimal if it generates a schedule with minimum makespan.

There is a partial order on the set of sequences with the property that at least one optimal sequence is in the set of minimal elements of this partial order independent of the given processing times. This partial order is part of a concept of irreducibility for sequences introduced by BRÄSEL and KLEINAU [2].

We describe a polynomial time algorithm for solving the restricted isomorphism problem on acyclic digraphs associated with open shop sequences. This algorithm allows us to give computational results for the number of such non-isomorphic digraphs of a given format. This number can be used to decrease the set of corresponding sequences which have to be examined for characterizing the minimal elements of the partial order mentioned above.

2 Terminology and Preliminaries

In general we use notation well understood in graph theory. Whenever we refer to a graph or digraph it is assumed to be finite, and additionally, neither (anti-)parallel edges nor loops are allowed.

2.1 Hamming graphs

The Cartesian product $G = H_1 \times H_2$ of two graphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ is the graph $G = (V, E)$ where the set of vertices is the Cartesian product $V = V_1 \times V_2$ and two vertices, denoted by two 2-tuples $(v_1 v_2)$, $(w_1 w_2) \in V$ with v_i, $w_i \in V_i$ for $(i = 1, 2)$, are adjacent if and only if $v_1 = w_1$ and $(v_2 w_2) \in E_2$ or $v_2 = w_2$ and $(v_1 w_1) \in E_1$ holds.

The *Hamming distance* between two t-tuples $x = (x_1 \dots x_t)$ and $y = (y_1 \dots y_t)$ is said to be the number of positions in which the entries in x and y differ. Formally,

$$d(x, y) = |\{i : x_i \neq y_i\}|.$$

A *Hamming graph* is the Cartesian product of complete graphs. Let H be such a Cartesian product of t complete graphs. The vertices of H can be labeled with t-tuples such that any two vertices $v = (v_1 \dots v_t)$ and $w = (w_1 \dots w_t)$ are adjacent if and only if $d(v, w) = 1$. In general, the Hamming distance $d(v, w)$ is the shortest path distance in H between any two vertices $v = (v_1 \dots v_t)$ and $w = (w_1 \dots w_t)$. Such a labeling of the vertices of H is called a *Hamming labeling.*

A special Hamming graph is the n-cube Q_n, often also referred to as the n-dimensional hypercube or binary Hamming graph, which can be interpreted as the Cartesian product of n copies of the complete graph K_2.

This paper deals only with Hamming graphs $K_n \times K_m$. Such graphs are of order $n\,m$ and each of them contains m disjoint subgraphs isomorphic to K_n (so-called n-cliques) as well as n disjoint subgraphs isomorphic to K_m (so-called m-cliques).

2.2 Sequence graphs

A directed acyclic graph is often abbreviated to *dag*. In the following we define dags which are based on Hamming graphs.

Definition 2.1 An $n \times m$ *sequence graph* D is an acyclic orientation of the Hamming graph $K_n \times K_m$. We write (ij), $1 \leq i \leq n$, $1 \leq j \leq m$, for the vertices of D. The symbol $\mathcal{D}_{n,m}$ denotes the set of all such dags of format $n \times m$.

Note, that the term 'sequence' shows a relation to sequences of production scheduling, which will be given in Section 4.1.

A *tournament* is an orientation of a complete graph. Due to Rédei [16] each tournament contains an oriented Hamilton path. Then it is easy to see that each acyclic tournament is transitive and that such a tournament contains *exactly* one Hamilton path. So, a sequence graph contains n subgraphs isomorphic to tournaments of order m ("row tournaments") and m disjoint subgraphs isomorphic to tournaments of order n ("column tournaments"), and each tournament of a sequence graph is spanned by a unique path.

2.3 Latin rectangles

Recall that a *Latin rectangle* $L(n, m, r)$ is an $n \times m$ matrix with values from the set $\{1, 2, 3, \dots, r\}$ such that each value occurs at most once in each row and each column. A *Latin square* of order n is a Latin rectangle with $n = m = r$. Many problems concerning these special matrices are discussed by Dénes and Keedwell [4],[5]. The following definition connects Latin rectangles with sequence graphs.

Definition 2.2 Let $L = L((ij)) = L(n, m, r)$ be a Latin rectangle with elements (ij), $1 \leq i \leq n$, $1 \leq j \leq m$. We write $l(ij)$ for the value of element (ij). The graph $D(L)$ denotes the associated $n \times m$ *sequence graph* of L with vertices (ij), $1 \leq i \leq n$, $1 \leq j \leq m$ in which two vertices (ij) and (st) are connected by an oriented arc $((ij), (st))$ if one of the following holds:

1. $(i = s) \wedge (l(ij) < l(it))$,
2. $(j = t) \wedge (l(ij) < l(sj))$.

The vertices of this graph are identified with the elements of the Latin rectangle, and informally, each pair of vertices in a row and in a column is connected with an arc oriented from the smaller to the larger value in the corresponding Latin rectangle. Obviously, such digraph associated with a Latin rectangle does not contain any cycle, therefore it is a dag and the above definition for sequence graphs is fulfilled. In this manner we are able to assign a unique sequence graph $D \in \mathcal{D}_{n,m}$ to a Latin rectangle $L(n, m, r)$.

Conversely, if we want to assign a unique Latin rectangle to a given sequence graph we have to reduce the set of Latin rectangles by an additional constraint. For this purpose the introduction of the term 'sequence' follows.

2.4 Sequences

Definition 2.3 A Latin rectangle $L(n, m, r)$ is called a *sequence* $S((ij)) = S(n, m, r)$ if for each value $s(ij) > 1$ the number $s(ij) - 1$ occurs as a value in row i or in column j.

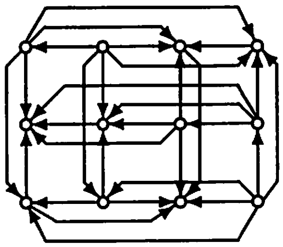
Now we are able to formulate the desired correspondence between these special Latin rectangles and the sequence graphs introduced above. This result was proved by BRÄSEL and KLEINAU [1].

Proposition 2.4 *There is a one-to-one correspondence between sequences $S(n, m, r)$ and sequence graphs $D \in \mathcal{D}_{n,m}$.*

This can be proved by giving a unique assignment of sequences $S(n, m, r)$ to given sequence graphs $D \in \mathcal{D}_{n,m}$ based on the longest paths in D. As an example for this one-to-one correspondence, Figure 1 shows a sequence S and its associated sequence graph $D(S)$.

$$\begin{pmatrix} 2 & 1 & 4 & 3 \\ 5 & 4 & 3 & 2 \\ 3 & 2 & 5 & 1 \end{pmatrix}$$

A sequence $S = S(3, 4, 5)$.



The sequence graph $D(S)$.

Figure 1.

3 Isomorphism and Recognition

As it will be shown in Section 4.1, a sequence can be interpreted as a feasible combination of machine orders and job orders of a shop scheduling problem with a complete operation set. On the other hand, there is a correspondence between sequences and sequence graphs given in Proposition 2.4. For this reason it is interesting to study the structural characteristics of sequences and sequence graphs, respectively, by distributing them to classes with same properties.

3.1 Sequence isomorphism

Our intention is to focus only on the main structural differences between sequences of open shop scheduling problems. Therefore, we introduce the term 'isomorphism' for sequences as follows.

Definition 3.1 Two sequences S_1 and S_2 are called *isomorphic* if there exists a row permutation π_R and a column permutation π_C in S_1 such that S_2 or the transposed of S_2 can be obtained.

A sequence $S(n, m, r)$ with $n = m = r$ is a Latin square and Latin squares can be viewed as multiplication tables of quasigroups. Usually, two Latin squares are said to be isomorphic if their corresponding quasigroups are isomorphic; i.e., if we are able to transform one square into the other by application of the same permutation of rows, columns and symbols simultaneously (see DÉNES and KEEDWELL [5]). Following this Latin square isomorphism, the Latin square graph is defined (e.g. see MILLER [13]). Note, that the definition for isomorphism of Latin squares viewed as quasigroups differs from the corresponding definition for sequences above. In this paper we consider only Definition 3.1 for isomorphic sequences, which is based on possibly different permutations of rows and columns and on a reflection in the main left-to-right diagonal.

In order to get an elegant algorithm for testing sequence isomorphism it is useful to consider only certain standard sequences. Thus, using permutations of rows and columns as well as transposing we can always put a sequence in a so-called normal form.

Definition 3.2 A sequence $S((ij)) = S(n, m, r)$ with $n \leq m$ and values $s(ij)$ is called *normal* if $s(11) = 1$ holds and if the order on the first row and the first column is increasing.

Theorem 3.3 *Let $n \leq m \leq r$ with $n, m, r \in \mathbb{N}$. The isomorphism of $n \times m$ sequences is decidable in $O(n^2 m)$ time.*

Proof: The following simple algorithm decides whether two given $n \times m$ sequences are isomorphic.

```
The Sequence Isomorphism Algorithm
Input:   Two $n \times m$ sequences $S, T$. Let $t(ij)$ be the value of element
         $(ij)$ in $T$.
Output:  An isomorphism (row and column permutations) if it exists.
  1. Put $S$ in an arbitrary normal form, say $S'$, by suitable row and column
     permutations, say $\pi_{S_R}$ and $\pi_{S_C}$.
  2. For all $t(ij) = 1$ in $T$ do:
     (a) Put $T$ in the normal form with $t'(11) = t(ij)$, say $T'$, by suitable
         row and column permutations, say $\pi_{T_R}$ and $\pi_{T_C}$.
     (b) If $T' = S'$, return the isomorphism by the given permutations
         $\pi_{S_R}\pi_{T_R}^{-1}$ and $\pi_{S_C}\pi_{T_C}^{-1}$. Stop.
     (c) If $n = m$ do:
         i. Set $T'^* =$ transposed of $T'$.
         ii. If $T'^* = S'$, return the isomorphism by the given permuta-
             tions $\pi_{S_R}\pi_{T_R}^{-1}$ and $\pi_{S_C}\pi_{T_C}^{-1}$ as well as transposing. Stop.
```

If the *Sequence Isomorphism Algorithm* terminates without returning an isomorphism, S and T cannot be isomorphic. The correctness of this algorithm follows from the previous discussion, especially from Definition 3.1 and Definition 3.2.

Concerning the time complexity, note that the row and column permutations of step 1 and step 2(a) of the algorithm can clearly be performed in $O(nm)$ time. Furthermore, the steps 2(b) and 2(c) also need $O(nm)$ time. Obviously, there are at most n different normal forms of a given sequence T. Thus, step 2 has to be repeated at most n times and we get the bound $O(n^2 m)$ for the entire time complexity of step 2. □

3.2 Sequence graph recognition

At first, we investigate the time complexity for the recognition of sequence graphs. We refer to a couple of algorithms which take a directed graph as input and checks whether it is a sequence graph.

Theorem 3.4 *Let $D = (V, E)$ be a digraph. The problem of deciding if D is a sequence graph is solvable in $O(|E|)$ time.*

Proof: The recognition of a sequence graph can be divided into two different algorithm parts. At first, it has to be determined if the underlying undirected graph of D is a Hamming graph of the form $K_n \times K_m$. The second part consists of the verification of an acyclic orientation of D.

Some recently published results concerning the problem of recognition Hamming graphs can be used for the first part of the proof, see IMRICH and KLAVŽAR [9]. The authors show that for a given undirected graph G with q edges it is possible to decide in $O(q)$ time whether G is a Hamming graph. Given the digraph D we consider the underlying graph $G(D)$ which is the undirected graph obtained from D by ignoring the orientations of its edges. The *Hamming Graph Algorithm* in [9] can be applied to the graph $G(D)$ with the modification that we search only for Cartesian products of *two* complete graphs. Thus, the first algorithm of the proof for checking the structure of the given digraph $D = (V, E)$ needs $O(|E|)$ time.

The second part of the proof leads to the problem of topological sorting. A *topological sorting* of a digraph $D = (V, E)$ is a linear order $v_1, v_2, \ldots, v_p$ of its vertices such that whenever $(v_i v_j) \in E$, we have $i < j$. It is well known that a digraph D is acyclic if and only if there exists a topological sorting of D. Thus, we can use the *Topological Sorting Algorithm* (e.g. see [10]) which tries to find such a linear order so that we are able to verify if the given digraph is acyclic. The computational complexity for this algorithm with input digraph $D = (V, E)$ is $O(|V| + |E|)$, first stated by KNUTH [10]. Since the number of arcs in an oriented Hamming graph cannot be smaller than the number of vertices we can simplify this bound to $O(|E|)$ in this case. Thus we get the proposition of Theorem 3.4. □

Now we are searching for an efficient method to retrieve the sequence from a given sequence graph according to the one-to-one correspondence stated in Proposition 2.4.

Theorem 3.5 *Let $n \leq m$ with $n, m \in \mathbb{N}$. The sequence according to a given $n \times m$ sequence graph $D = (V, E)$ can be computed in $O(nm^2)$ time.*

Proof: Let D be an $n \times m$ sequence graph on nm vertices. Roughly, the construction of the corresponding sequence is as follows. Each vertex of

D is associated with an element in an $n \times m$ matrix $S = ((ij))$. For each vertex (ij) we write its rank as the corresponding value $s(ij)$, where the *rank* of a vertex (ij) denotes the number of vertices on a longest path from a source to vertex (ij) in D.

Analogously to the proof of Theorem 3.4, this sequence construction from a sequence graph can be divided into two algorithm parts: vertex labeling with labels (ij), $1 \leq i \leq n$, $1 \leq j \leq m$ and rank determining for each vertex.

For the first part we use the *Labeling Algorithm* of IMRICH and Klavžar [9], again applied to the underlying undirected graph $G(D)$. This algorithm is an essential part of the mentioned *Hamming Graph Algorithm* since the procedure of labeling is the same but now we know that a Hamming graph is given. The *Labeling Algorithm* can be used for the graph $G(D)$ with the two modifications that we search only for Cartesian products of *two* complete graphs and that the canonical Hamming labeling is replaced by a corresponding labeling beginning from the label (11) instead of the label (00) for the first chosen vertex. We use this modified labeling in order to have labels which allow the identification of the vertices with the elements of an associated sequence in the usual manner. Again, we have the complexity of $O(|E|)$ for a digraph $D = (V, E)$.

The second part of the sequence construction, rank determining, can be solved also by a certain version of the *Topological Sorting Algorithm* mentioned in the proof of Theorem 3.4. Instead of determining only one source in each step as in the simple version of the *Topological Sorting Algorithm* we now mark all sources. If we proceed by deleting the sources and the incident arcs, then marking all new sources again and so on, we are able to determine the rank for each vertex, too. The computational complexity for this algorithm with rank determining is also $O(|V| + |E|)$ for a digraph $D = (V, E)$. This bound can be clearly simplified to $O(|E|)$ if D is a sequence graph.

Since the number of arcs of each sequence graph $D \in \mathcal{D}_{n,m}$ is exactly

$$n\binom{m}{2} + m\binom{n}{2},$$

and assuming that $n \leq m$, we can derive the more specified complexity bound $O(nm^2)$ for the *Topological Sorting Algorithm* and the *Labeling Algorithm* in an $n \times m$ sequence graph. □

Given a sequence graph $D = (V, E)$, let us consider the spanning subdigraph $H = (V, F)$ with $F \subset E$ containing only the arcs which belong to the Hamilton paths of the row and column tournaments in D. The time complexity for the *Topological Sorting Algorithm* of such a graph H reduces to $O(nm)$ since the number of arcs is decreased to $(n-1)m + n(m-1)$

by the deletion of the redundant arcs of each tournament. But from such a dag we can no longer uniquely retrieve the corresponding sequence rectangle structure. Therefore, if we want to associate a certain application in scheduling (as in Section 4.1) we have to use sequence graphs which results in higher time complexity.

3.3 Sequence graph isomorphism

The sequence graph introduced in Section 2.2 is a natural representation of a sequence in terms of graph theory. We transfer isomorphism of these sequences (i.e., special Latin rectangles) to the corresponding sequence graphs, because then we will be able to give an algorithm for sequence graph isomorphism consisting of two parts: retrieving the sequences from the given sequence graphs and checking these sequences for isomorphism.

Lemma 3.6 *Two sequences S and S' are isomorphic if and only if their associated sequence graphs $D(S)$ and $D(S')$ are isomorphic.*

Proof: A sequence graph contains $n+m$ subgraphs isomorphic to tournaments. There are n row tournaments of order m and m column tournaments of order n. Two tournaments are vertex disjoint if and only if the tournaments are either different row or different column tournaments. Therefore, the adjacencies are preserved if we permute rows, permute columns, or if we reflect in the main left-to-right diagonal of a sequence. □

Finally, using this Lemma 3.6 together with Theorem 3.5 and Theorem 3.3 we obtain the following Theorem 3.7.

Theorem 3.7 *Let $n \leq m$ with $n, m \in \mathbb{N}$. Isomorphism of $n \times m$ sequence graphs is decidable in $O(nm^2)$ time.*

4 Applications

In the following, we give the connection between the sequences and sequence graphs of the previous discussion with sequences of open shop problems in scheduling theory. We also present numerical results for the implementation of the *Sequence Isomorphism Algorithm* considered in Section 3.1.

4.1 Production scheduling

As already mentioned a sequence graph $D \in \mathcal{D}_{n,m}$ contains $n+m$ acyclic tournaments as subgraphs (n disjoint row tournaments of order m as well as m disjoint column tournaments of order n). Each tournament is spanned by a unique oriented Hamilton path. This well known fact (e.g. see [16]) is important for our application of sequence graphs and corresponding sequences in shop scheduling.

A *sequence* of an open shop problem in scheduling theory is a feasible combination of its machine orders and its job orders. A sequence graph D can be interpreted as such an open shop sequence for an open shop problem with a complete operation set in the following way: each vertex (ij) of D is identified with the operation of job J_i on machine M_j. The unique oriented spanning path of the i-th row tournament (spanned by the vertex set $\{(i1),(i2),\dots,(im)\}$) corresponds to the machine order on job J_i and the spanning path of the j-th column tournament (spanned by the vertex set $\{(1j),(2j),\dots,(nj)\}$) corresponds to the job order on machine M_j.

This associated combination of job orders and machine orders is feasible because the underlying sequence graph does not contain any cycle.

Following the intention of [2], we are interested in sequences of open shop problems for which small maximum completion times can be expected. The number of vertices on the longest paths of such reasonable sequences is relatively small. There is a partial order P on the set of sequences using this property such that the minimal elements of P are good in the sense that there is always a minimal element which is optimal with respect to the maximum completion time and independent from the given processing times.

In order to get a more precise view of the different sequence structures we want to examine only the main structural characteristics of the sequences. Thus, it is useful to consider the isomorphism classes of sequences. In the following section, we give numbers of such non-isomorphic sequences.

4.2 Numbers of isomorphism classes

Without loss of generality we now assume $n \leq m$ for Latin rectangles $L(n,m,r)$, sequences $S(n,m,r)$, and sequence graphs $D \in \mathcal{D}_{n,m}$, because it is always possible to formulate the dual proposition if we interchange rows with columns, i.e., if we interchange jobs with machines in terms of scheduling (see Section 4.1).

A sequence is called *reduced* if the order on its values of the first row is increasing and the order on its values of the first column beginning from the second element is increasing as well. It is easy to see that the total number of $n \times m$ sequences with maximum value r is $m!\,(n-1)!\,R(n,m,r)$, where $R(n,m,r)$ denotes the number of reduced $n \times m$ sequences with maximum value r.

An $n \times m$ Latin rectangle $L(n,m,m)$ is called *normalized* if its values of the first row and its values of the first column are in natural order (see McKay and Rogoyski [12]). Obviously, this definition is only significant for 'classical' Latin rectangles; i.e., Latin rectangles with $m = r$. Since such classical Latin rectangles also fulfill the additional condition for sequences we use the term 'normalized' for sequences with $m = r$, too. A normalized sequence is reduced but not necessary vice versa.

In [1], Bräsel and Kleinau give an enumeration algorithm for sequences. This algorithm together with the *Sequence Isomorphism Algorithm* described in the proof of Theorem 3.3 gives us the ability to determine the number of sequence isomorphism classes for small n and m. For example, the enumeration of all reduced and non-isomorphic 3×4 sequences needs a CPU-time of 80 seconds on a fast HP-UX Workstation (model 9000/770/J210), whereas only the enumeration of all reduced 4×4 sequences already takes 17.33 hours on the same machine. For determining all isomorphism classes we apply the *Sequence Isomorphism Algorithm* naively. We turn out a list of non-isomorphic sequences in the following way. As input, we have a list of all reduced sequences which are also normal. The first sequence of this list is written into the so far empty output list of non-isomorphic sequences. Then, step by step we compare each element of the input list with the elements of the output list. If a sequence is not isomorphic to any of the sequences of the output list it is appended to this list until all input sequences have been examined.

By the implementations of these algorithms we get the computational results summarized in Table 1. Here, $R(n, m)$ denotes the total number of reduced sequences for all maximum values r, $m \leq r \leq n\,m$, and the symbol $I(n, m)$ stands for the number of their isomorphism classes. Note, that the number of non-isomorphic 4×4 sequences is not determined so far.

For comparison, we also give the number $N(n, m)$ which is the number of normalized $n \times m$ sequences, stated by McKay and Rogoyski [12].

5 Concluding Remarks

In this paper we have considered isomorphism algorithms for special Latin rectangles, the so-called sequences. The identification of such sequences with acyclic orientations of 2-dimensional Hamming graphs allows us to examine the main characteristics of the associated open shop scheduling sequences for problems with a complete operation set.

There are some generalizations for these problems, e.g. an analogous consideration for open shop problems with a partial operation set or a transfer to job shop problems, which are left for further investigation.

m	n	$R(n,m)$	$I(n,m)$	$N(n,m)$
1	1	1	1	1
2	1	1	1	1
2	2	7	3	1
3	1	1	1	1
3	2	34	17	1
3	3	1597	280	1
4	1	1	1	1
4	2	209	106	3
4	3	77772	25924	4
4	4	48205504	?	4

Table 1. Numbers of reduced sequences, non-isomorphic sequences and normalized sequences.

References

[1] H. Bräsel, M. Kleinau, On the number of feasible schedules of the open-shop-problem – an application of special Latin rectangles, *Optimization* **23** (1992), 251–260.

[2] H. Bräsel, M. Kleinau, New steps in the amazing world of sequences and schedules, *Math. Methods Oper. Res.* **43** (1996), 195–214.

[3] J. Chen, A linear-time algorithm for isomorphism of graphs of bounded average genus, *SIAM J. Discrete Math.* **7**, 4 (1994), 614–631.

[4] J. Dénes, A. D. Keedwell, *Latin Squares and their Applications*, Academic Press, New York and London, 1974.

[5] J. Dénes, A. D. Keedwell, *Latin Squares: New Developments in the Theory and Applications*, vol. 46 of Ann. Discrete Math., North-Holland, Amsterdam, 1991.

[6] M. R. Garey, D. S. Johnson, *Computers and Intractability - a Guide to the Theory of NP-Completeness*, W. H. Freeman & Co, New York, 1979.

[7] T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, *J. Assoc. Comput. Mach.* **23**, 4 (1976), 665–679.

[8] J. E. Hopcroft, J. K. Wong, Linear time algorithm for isomorphism of planar graphs, in *Proc. 6th ann. ACM Symp. Theory Comput.*, 1974, 172–184.

[9] W. IMRICH, S. KLAVŽAR, On the complexity of recognition Hamming graphs and related classes of graphs, *European J. Combin.* **17**, 2/3 (1996), 209–221.

[10] D. E. KNUTH, *Fundamental Algorithms*, The Art of Computer Programming 1, Addison-Wesley, 1st ed., 1968.

[11] E. M. LUKS, Isomorphism of graphs of bounded valence can be tested in polynomial time, *J. Comput. Syst. Sci.* **25**, 1 (1982), 42–65.

[12] B. D. MCKAY, E. ROGOYSKI, Latin squares of order 10, *Electron. J. Combin.* **2** (1995), Note 3, approx. 4 pp. (electronic).

[13] G. L. MILLER, On the $n^{\log n}$ isomorphism technique, in *Proc. 10th SIGACT Symp. on the Theory of Computing*, 1978, 51–58.

[14] G. L. MILLER, Graph isomorphism, general remarks, *J. Comput. System Sci.* **18** (1979), 128–142.

[15] I. N. PONOMARENKO, Polynomial time algorithms for recognizing and isomorphism testing of cyclic tournaments, *Acta Appl. Math.* **29**, 1/2 (1992), 139–160.

[16] L. RÉDEI, Ein kombinatorischer Satz, *Acta Litt. Sci. Szeged* **7** (1934), 39–43.