ISSN: 0067-2904

A Metaheuristic Approach to the C1S Problem

Rewayda Abo-Alsabeh[1], Abdellah Salhi[2]
[1]Department of Mathematical Sciences, University of Kufa, Alnajif, Iraq
[2]Department of Mathematical Sciences, University of Essex, UK

Received: 29/6/2019 Accepted: 15/3/2020

Abstract

Given a binary matrix, finding the maximum set of columns such that the resulting submatrix has the Consecutive Ones Property (C1P) is called the Consecutive Ones Submatrix (C1S) problem. There are solution approaches for it, but there is also a room for improvement. Moreover, most of the studies of the problem use exact solution methods. We propose an evolutionary approach to solve the problem. We also suggest a related problem to C1S, which is the Consecutive Blocks Minimization (CBM). The algorithm is then performed on real-world and randomly generated matrices of the set covering type.

Keywords: Approximation algorithm, Genetic algorithm, Consecutive Ones Property, Consecutive Block Minimization.

نهج ألادله العليا لمسألة خاصية الواحدات المتعاقبة

رويدة رزاق محسن[*1]، عبد الله صالحي[2]
[1]قسم الرياضيات، كلية علوم الحاسوب والرياضيات، جامعة الكوفة، العراق
[2]قسم الرياضيات، كلية الرياضيات، جامعة اسيكس، انكلترا

الخلاصه

اعطيت مصفوفة-(0، 1)، أيجاد المجموعة العظمى من الاعمدة بحيث المصفوفة الجزئية الناتجة تمتلك خاصية الواحدات المتعاقبة (C1P) تدعى مسألة المصفوفة الجزئية للواحدات المتعاقبة (C1S). يوجد عدة طرق لحلها, لكن لا مجال لتحسين الحل كفاية. علاوة على ذلك فأن معظم دراسات المشكلة تستحدم طرق الحل الدقيق. نحن نقترح نهج التطور لحل المسألة. كذلك نقترح مسألة تتعلق بالـ C1S، وهي تصغير القوالب المتعاقبة (CBM). الخوارزمية يتم تطبيقها على مصفوفات من العالم الحقيقي و مصفوفات متولدة عشوائيا من نوع مجموعة الغطاء.

1. Introduction

The problem of consecutive ones submatrix on a binary matrix has been known since the 1950s. It was suggested by Fulkerson and Gross [1] and described as follows: Let A be an incidence matrix, and then can we rearrange its columns so that each row has a single block of ones?

1.1 Basic concepts of C1P and C1S

Definition 1.1. A block of 1's (block of 0's) in a binary matrix A is any maximal set of consecutive one entries (zero entries) appearing in the same row [2].

A binary matrix A is said to have the Consecutive Ones Property (C1P) if the ones in every row appear consecutively [3].

*Email: ruwaida.mohsin@uokufa.edu.iq

Definition 1.2. A binary matrix has the C1P for rows if its columns can be ordered such that all the 10s are consecutive in every row [3].
This property is similar for the columns, through matrix transposition.
Definition 1.3. Given a binary matrix, the problem of finding a maximum submatrix having columns with C1P property is called the Consecutive Ones Submatrix problem [4].
Definition 1.4. Given a matrix A, a valid permutation of A is a permutation of its columns that puts the ones in consecutive arrangement in each row. A binary matrix is C1P, if the matrix can be reordered by such a permutation [2].
The C1P property is desirable, which often leads to efficient algorithms. Recently, there has been a great deal of interest in matrix modification and transformation into a binary matrix having the C1P.
These transformations can be shown up as problems [5, 6, 7]. In general, these problems are NP-hard[1] even for very sparse matrices. Well known solution approaches depend on characterizing the so-called Tucker forbidden submatrices [8, 7, 9, 4]. Here we propose a heuristic approach to the C1S problem.

1.2 Brief review

In integer programming, the property of C1P is very interesting as it shows that the problems based on matrices having this property are easier to solve than the suggested model. Such matrices with C1P are totally unimodular [10, 11]. The property arises in many applications such as railway optimization [12], information retrieval [13], and scheduling [14]. It also finds applications in ancestral genome reconstruction [15] and the construction of physical mapping with hybridization data [16]. In graph theory, it helps to find out interval graphs [1, 17].

C1P has been fairly well studied. Kendall [18, 19] attributed the first appearance of the C1P property to Flinders Petrie, an archaeologist in 1899. Many heuristic methods were suggested for the problem of chronological ordering archaeological deposits before the work of Fulkerson and Gross [1], who provided the first polynomial complexity algorithm. In 1972, Tucker [8] proposed a characterization of matrices with the C1P using forbidden submatrices. Later, Booth and Lueker [20] introduced the first linear-time algorithm for it. They used a data structure called the PQ-tree. A binary matrix has C1P if and only if its data structure exists [20].

Let (α, β) −matrices be binary matrices with at most α $1's$ in each column and β $1's$ in each row. Garey and Johnson [21] pointed out that the decision version of the C1S problem is NP-complete. Later, Hajiaghayi and Ganjali [9] showed that the C1S problem is NP-hard for $(2,\ 4)$ −matrices. They also found a polynomial time solution for $(2, 2)$ −matrices. Hence, their work raised the question of whether the C1S problem stays NP-complete or not for both $(3, 4)$ −matrices and $(3, 2)$ −matrices. The question was answered by Tan and Zhang [4]. For both $(2, 3)$ −matrices and $(3, 2)$ −matrices having the C1S, they proved that the decision versions are NP-complete. For $(2, 3)$ −matrices that have no identical two columns, they showed that the C1S problem is polynomial time 0.8-approximable. They also found that the C1S problem is 0.5-approximable for $(3, 2)$ −matrices and for $(2, \infty)$ −matrices. On the other hand, they proved that for $(\infty, 2)$ −matrices there exists an $\epsilon > 0$ such that the approximation of the C1S problem within a factor of n^{ϵ} is NP-hard.

The Petrie Seriation (or Sequence-dating) Problem [19] is an important and interesting problem that was formulated in mathematical terms by Flinders Petrie almost one century ago. It is to chronologically order grave sites in a cemetery where each grave contains (or does not contain) a number of stylistic artifacts of a period. It is mathematically modelled by an incidence matrix A where the columns are specific artifact types and the rows are the grave sites. In 2006, Gargano and Lurie [22] introduced a hybrid evolutionary approach for solving this problem. They found a permutation of rows that transforms A into a matrix that has a consecutive ones block in each column; this is called a Petrie matrix. Finding this permutation is equivalent to showing that A has the C1P property. The Consecutive Block Minimization is a related problem. The aim is to minimize the number of blocks of 1's by permuting the columns of the matrix. Haddadi and others [3] gave a polynomial-time local-improvement heuristic for the problem.

This paper is organized as follows. Section 2 reviews current solution approaches. Section 3 presents the proposed GA algorithm for solving the C1S problem. Section 4 provides experimental results. Section 5 is the conclusion.

2. An illustration of C1S and current solution approaches

2.1 The seriation problem

The incidence matrix A is an input to this problem. It is a binary matrix with m rows and n columns (grave sites and object types, respectively); the a_{ij} entry of A has a value of one if grave i has any object of type j, and zero otherwise. If the rows (graves) were in the suitable time successive order,

[1]A problem belongs to the class NP if a solution to it can be generated and verified quickly, i.e. in polynomial time. It is believed that NP-hard problems are intractable, i.e. that there is no efficient algorithm to solve them, [7].

, then each column (object) would indicate a time period during which that object was common. Assume that each object represents the time interval from the appearance of the object to the time when it becomes uncommon. The chronological ordering of the graves is reached by getting rows permutation of A that transforms it into Petrie form. This will minimize the total temporal range of an object summed over all the varieties. This range for an artifact type is the span from the first to the last appearance of a '1' in the column of A corresponding to that artifact type.

Kendall [19] introduced a solution by using a Similarity Matrix $Q = AA'$ of size $n \times n$ for graves, which is nonnegative and symmetric. He found out that a permutation P can convert matrix A into a Petrie form (i. e.,PA), as it can transform the grave matrix Q into Robinson form (Robinson Matrix[1], i. PQP'). The final matrix PA has the C1P where each column displays the correct order of the artifact type from its appearance to it disappears.

2.2 Polynomial-time local-improvement heuristic for CBM

The problem of CBM-decision is NP-complete even if it is limited to binary matrices containing two ones in each row [13, 23]. However, Haddadi introduced a polynomial-time heuristic which gives a permutation such that the optimum solution and the consecutive blocks do not differ by more than 50%.

Haddadi and others found a polynomial-time local-improvement heuristic for the CBM. They introduced two $O(n^2)$ size local neighborhood search, where the blocks number of a neighbor is provided in $O(m)$ operations [2, 3].

2.3 A GA-based solution approach

The GA has been used in related cases. Here, we develop an implementation to deal with C1S. GA is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). It was proposed by Holland [24]. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators, such as crossover, mutation, and reproduction [25].

Solution representation

The basic is to keep permuting the columns of the given binary matrix to get as many consecutive ones as possible in every row. An integer is used here to represent a solution or chromosome of size n, an array of the column indices of the matrix. The position of each gene in the chromosome corresponds to a column in this matrix. Figure-1 illustrates this representation.

1	6	7	2	10	4	8	3	9	5

Figure 1-Chromosome representation

Genetic operators

After randomly generating the initial population, we breed successive generations of offspring. This can be achieved by performing genetic operators which are crossover, mutation, and reproduction.

Crossover operator. We use the 'Order Crossover' of Gen and Cheng [26] to breed a new child. This operation begins by picking a substring (subchromosome) from one of the two parents randomly, then copying it into its corresponding location in the child. The genes in the second parent that appeared in the substring are deleted to prevent repeating the gene. In the end, the genes are placed into the unfixed location of the child from left to right, relying on the order of the sequence (Figure-2).

Mutation operator. It is used to allow the genetic algorithm to search in the global solution space and prevent trapping in local optima, via changing one or more genes. Here, two genes from an individual are randomly chosen and then swapped. Figure-3 illustrates the mutation operator.

[1]Robinson matrix is a matrix with entries which do not increase as one progresses along a row beyond the main diagonal and do not decrease as one continues to progress along that row towards the main diagonal.

Parent1	1	10	8	7	5	3	6	2	4	9
Child1	**1**	**10**	**8**	3	6	5	2	4	7	9
Parent2	3	6	5	2	4	8	7	9	1	10

Parent1	1	10	8	7	5	3	6	2	4	9
Child1	**3**	**6**	**5**	1	10	8	7	2	4	9
Parent2	3	6	5	2	4	8	7	9	1	10

Figure 2-Child1 and Child2 obtained with crossover

Parent	1	10	**8**	7	5	3	6	**2**	4	9
Child	1	10	**2**	7	5	3	6	**8**	4	9

Figure 3-Individuals obtained with the mutation operator

Reproduction. Without any modification, some fit chromosomes are copied via this operator into the next generation.

Fitness functions

The matrix does not include enough information to decide whether it has the property or not. However, we will use the entries of the matrix to build a fitness function. Many fitness functions are tested and the most reliable one with respect to the number of C1S columns and run time is selected.

Fitness Function FF1. It computes the number of 1's in each row using a simple formula.

Summing the number of 1's for all the rows will give the fitness value of the matrix.

Suppose that we have the following rows with four ones each

$$row_1 = (1\ 0\ 1\ 1\ 1\ 0), \qquad row_2 = (1\ 0\ 1\ 0\ 1\ 1).$$

The first row has two blocks of 1's, k_1, and k_2, where k_1 is one element and k_2 is three 1's. We deal with every block as a sequence of 1's. The associated series is defined as the ordered formal sum $\sum_{j=1}^{l} j$ where l is the length of the block. Counting the elements for the two blocks will be as

$$Fitness\ row = \sum_{i=1}^{k} \sum_{j=1}^{l_i} j\,, \quad \text{where} \quad k = number\ of\ blocks, \quad \text{and} \quad l_i = number\ of\ 1\text{'s in } i\text{th blocks}. \tag{1}$$

Applying the formula for the two rows gives fitness values of 7 and 5, respectively, where $\sum_{j=1}^{1} j + \sum_{j=1}^{3} j = 1 + 6 = 7$ and $\sum_{j=1}^{1} j + \sum_{j=1}^{1} j + \sum_{j=1}^{2} j = 1 + 1 + 3 = 5$. Applying this formula for a matrix gives different fitness values for every row. The fitness value for the whole matrix is found by accumulating the fitness values of all the rows, as shown in the next formula

$$fitness\ matrix = \sum_{s=1}^{m} fitness\ row\ (s),$$

$$where \qquad m = number\ of\ rows,$$

The larger the fitness, the better the matrix with respect to consecutive ones blocks. With different permutations of columns in each generation and computing the fitness function of the matrix, the consecutive solutions are improved every time by pushing the ones together to form less blocks. In the final generation, it is expected that a matrix with best consecutive ones in every row is obtained.

Fitness Function FF2. This function relies on counting the ones in the different regions of the matrix. That is, by finding the label of the connected components of the matrix, where each maximal connected region is assigned as a unique label [27, 28]. The matrix below has three regions of ones. The first connected components in the matrix has elements (label 1), the second has three elements (label 2), and the last has one element (label 3), as shown in the region matrix. We also deal with

every region as a sequence of 1's, like the fitness FF1. The fitness values of the regions are 15, 6, and 1, respectively, which means that the fitness function for the matrix is 22, where $\sum_{j=1}^{5} j + \sum_{j=1}^{3} j + \sum_{j=1}^{1} j = (1+2+3+4+5) + (1+2+3) + 1 = 15 + 6 + 1 = 22$.

$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}, \qquad regions = \begin{pmatrix} 0 & 1 & 0 & 2 & 2 \\ 0 & 1 & 1 & 0 & 2 \\ 1 & 1 & 0 & 3 & 0 \end{pmatrix}$$

This function pushes the ones in different separated regions, which may produce a variety of fitness values more than those of FF1. It accumulates the ones, not only in the rows, but also in the columns. This helps to accumulate the ones in large blocks. Function FF1 may give similar fitness values for many rows, and consequently similar fitness values for different matrices. This causes that the worse matrix with less columns having the C1P property may be selected for the next generation. With FF2, there is a higher chance to distinguish between matrices. It may give better results than the FF1 function. Although it gives good results, unfortunately it costs more time than FF1. Thus, it is not the desired function for the proposed algorithm.

Fitness Function FF3. This function counts the zeros, instead of the ones, in every row and then for the whole matrix. The same formula that is used for finding the maximal sum of 1's in FF1 is used here. Maximizing the fitness value leads to aggregating the zeros in blocks, which results in minimizing the number of gaps between the ones. This fitness almost gives the same result but with longer time, because the matrix is sparse so the calculations cost longer time than FF1. As we seek better results with shorter time, this fitness function is also not suitable for our implementation.

Fitness Function FF4. The fitness function that Gargano and Lurie used for solving the Petrie Seriation Problem [22] is used to find the consecutive ones in the columns by permutation of the rows. Although it gives good results for small matrices, they did not mention whether it is suitable for large matrices with different densities. Thus, we decided to test it. Their fitness is evaluated with the so called Petrie Range Index (PRI) of the permuted binary matrix. We formulate the fitness for our matrix by choosing the first column c_f and the last column c_l, where in each row there is at least one element. The PRI for that row is $c_l - c_f + 1$. The fitness for the matrix is the sum of these values over all the rows. The best solution is reached through minimizing the PRI. It is noticeable that if the matrix has the same number of ones in each column, then the fitness value for the matrix will be the same for any permutation. As a result, good matrices may not cross to the next generation, which causes not having very good results. This approach is more suitable for matrices with high density with different number of ones in each column. It generates less columns with C1P but in shorter time as compared to the other functions. It is formulated as follow

$$Pertie\ range\ index\ fitness = \sum_{i=1}^{m} c_{l_i} - c_{f_i} + 1,$$

where $m = number\ of\ row,$
$c_{l_i} = last\ column,$
$c_{f_i} = first\ column.$

Here again, FF4 is not suitable for our implementation of GA.

Fitness Function FF5. This function is meant to solve the CBM problem which is equivalent to C1S. The aim is to permutate the matrix columns to minimize the number of blocks of consecutive ones. Building the function relies on counting the blocks of the ones in each row, then for the whole matrix. The same two rows, Row_1 and Row_2, that are used in FF1 function for illustrations, are used here. The first row has two blocks $b1;\ b2$ of ones and the second one has three blocks $b1;\ b2;\ b3$. The formula for the matrix is

$$CBM\ fitness = \sum_{i=1}^{m} \sum_{j=1}^{k_i} b_j,$$

where $m = number\ of\ rows\ in\ the\ matrix,$
$k_i = number\ of\ blocks\ in\ i^{th}\ row,$

b_j *takes value* 1.

Applying the formula for the two rows gives values of 2 and 3, respectively. The fitness value for the whole matrix can be obtained by summing up the fitness values of all the rows. This fitness also pushes the 1's together to make blocks. The smaller the fitness value, the lesser number of blocks the matrix will have, thus producing a larger C1S submatrix.

Stopping criterion

A common criterion used to stop the proposed genetic algorithm is the maximum number of generations.

Selection procedure

It usually implements a roulette wheel which is biased towards fit individuals. This means that good individuals are likely to be parents.

The proposed GA algorithm is shown in a pseudocode as shown in Algorithm 1.

Algorithm 1: Algorithm C1P

1 Input a binary matrix A;
2 Input the rate of crossover and ω the rate of mutation;
3 Generate a random population of permutations of columns of matrix A;
4 Evaluate the fitness of the permutations according to some fitness function;
5 Rank individuals according to their fitness;
6 Select parents from the population according to some selection procedure;
7 Generate a new population by applying the following operators: crossover, mutation, and reproduction;
8 Compute the fitness of the individuals of the new population;
9 Until (The sopping criteria are satisfied) Repeat from 5.
10 Return Permuted matrix.

2.4 Testing the quality of the fitness functions

To compare the quality of the fitness functions that are stated above, we implement them separately in Algorithm1. They are applied to the same matrices with a fixed number of generations and initial population. We run the GA on 5 different matrices for each size. The size of the population and the generation are fixed to 40 for the first two matrices and 100 for the rest. Table-1 records the number of columns (Nbcols) with C1P rounded to the nearest integer. We refer to the matrices size as (Mat.) and to the density as (Dens.). The table shows that FF1 and FF5 produce better results in a shorter time. Although both of them give almost the same results, the latter is superior in CPU time.

Table 1-Comparing the fitness functions on the number of columns with C1P and time.

Mat. Dens.(%)		FF1 Nbcols	FF1 Time(s)	FF2 Nbcols	FF2 Time(s)	FF3 Nbcols	FF3 Time(s)	FF4 Nbcols	FF4 Time(s)	FF5 Nbcols	FF5 Time(s)
24	6	21	1.11	21	1.97	20	2.35	14	0.49	23	0.71
50	4	17	1.45	18	5.89	17	8.26	13	0.31	20	1.67
100	2	28	12.58	30	14.05	27	33.37	22	1.22	27	9.77
200	.	20	16.66	20	95.94	20	133.4	15	3.13	21	12.31
500	.	14	47.59	14	2760.48	13	641.19	11	17.52	14	31.31

3. Computational experience

3.1 Results of Algorithm 1 on the CBM problem

This algorithm is implemented for matrices of the set covering problem with different densities. We use these matrices to serve as a test bed to run the GA implementation. The results shown in Table- 2 are obtained by applying Algorithm 1 to ten different matrices of each size. We run the GA 10 times on every matrix, each time with a random initial population. In fact, the results of the algorithm rely on the arrangement of the binary matrix and, therefore, it may produce better results when it is repeated with different initial populations. The run time depends slightly on the density of

the matrix and the number of rows, but more on the number of columns. We set the number of generations and population size to 100. The heuristic gives the results of the number of blocks and columns. The first five columns have the initial information of the matrices with the number of generations (Gen) in column 3. The rest of the table shows the number of final blocks with the number of improved blocks, the number of columns (Nbcols) with C1P, and the time. For small matrices, the number of columns with C1P is good, but for matrices with size ≥ 100, the results are not up to expectation. From our experience, the results can be improved by the following:

1. Increasing the number of generations and the initial population size.

2. For large matrices, it is better to have an initial population which covers more than half the columns of the matrix to obtain about the same number of columns having the C1P. This reduces computing time. This can be achieved by reducing the chromosome that presents the matrix to the half.

3. Dividing the matrix into submatrices and applying Algorithm 1, separately, then applying it for the whole matrix put together. This however requires more time.

4. The initial population can be seeded by applying Algorithm 1 many times, then making the final population from the best of each run and use it as the initial population. This also computationally expensive.

Overall, this algorithm is performed to different instances of real-world and randomly generated matrices. The nonsymmetric matrices that are created from the set covering problem are not checked for the consecutive ones property, so the optimal solutions are not known. Consequently, we cannot discuss the quality of our results. The rest of matrices, Real-world data, ($R1km$ -$R10km$) whose sizes are given in Table- 3, arise from the problem of stop location, posted by a German railway company [29]. The problem is formulated as a Set Covering Problem (SCP) [29, 30]. These cases produce matrices with 0, 1 entries, that are assumed to have almost C1P. Other small size matrices of types B and C, which are randomly created by Ruf and Schöbel [29], are both sparse and almost have the C1P with density values of 3% and 5%, respectively (Table- 3). Finally, the algorithm is implemented as follows:

The results of Table- 4 show that the columns of the B and C matrices from Ruf and Schöbel could not satisfy the C1P. Also, applying this procedure alone is not enough to improve the number of consecutive blocks.

The results on the real-world data of Ruf and Schöbel are illustrated in Table-5. The first matrix R_{1km} almost satisfies the CBM, since the number of final blocks is near to m, and thus optimal. With respect to the rest of matrices of this type, although the final numbers of blocks and columns differ from the optimal values, they are close to the lower bound m, but not to the upper n. However, it still produces large submatrices with C1P. From the results shown in the three tables, we can say that:

1. Performing Algorithm 1 improves the number of blocks but not that of the columns with C1P.

2. Originally, any matrix should have C1S submatrix of at least two columns. The result from Algorithm 1 does not rely on the size of the C1S only, but also on the structure of the matrix.

3. Minimizing the number of blocks does not imply finding a large C1S submatrix. We can say that improving the C1S can improve the CBM, but the converse is not always true. This is because the size of the C1S submatrix relies on the position of the destructive column.

4. Over all, the number of consecutive blocks from the algorithm in Tables-(3, 4 and 5) show a good improvement, While the number of columns with C1P is not as expected.

4. Conclusions

We presented a metaheuristic method for solving the C1S problem. A basic GA is proposed with many new fitness functions and FF5 is chosen to solve the C1S problem. The minimum consecutive blocks or CBM in [3] is also solved using the GA. We applied our algorithm to a large number of randomly generated matrices and real-world instances. The results show that large submatrices with C1P can be found for matrices with small sizes. However, since the optimum solutions are not known, it is not possible to say how far the solutions returned by our approaches are actually resulted from them. Finally, we can say that the GA is a suitable algorithm for solving the C1S problem if we want to separate a small submatrix with C1S.

Acknowledgements

The authors would like to thank Schöbel, one of the authors of [29], who provided us with real-world data. Also, we extend tanks to Omar Kirikcki for supplying us with the random nonsymmetric matrices.

Table 2- Computational results of the C1P algorithm for nonsymmetric randomly generated matrices

Initial information					GA Algo.			
Mat.	Dens. (%)	Ge.	Initial blocks	Initial C1P	Final Blocks	Blocks improve.	Nbcols	Time(s) C1P
24	16	100	84	3	24	60	24	2.96
50	4	100	103	6	51	52	49	3.93
100	2	100	209	17	174	35	32	9.68
100	4	100	444	10	373	70	12	9.97
100	10	100	863	8	719	144	11	9.96
200	2	100	781	16	706	72	23	12.07
200	4	100	1379	8	1304	75	13	11.85
200	10	100	3248	8	2834	413	12	12.13
500	2	100	5189	11	4925	264	17	31.21
500	4	100	9612	4	9193	419	11	30.62
500	10	100	22021	6	20721	1958	9	33.79

Table 3-Test problem statistics

Real world matrices			Randomly generated instances					
Mat.	Dens.(%)	Size	Mat.	Dens.(%)	Size	Mat.	Dens.(%)	Size
R_{1km}	0.002	757 × 707	B1	0.032	100 × 96	C1	0.048	100 × 100
R_{2km}	0.014	1196 × 889	B2	0.036	100 × 95	C2	0.054	100 × 100
R_{3km}	0.022	1419 × 886	B3	0.034	100 × 92	C3	0.510	100 × 99
R_{5km}	0.043	1123 × 593	B4	0.031	100 × 92	C4	0.050	100 × 100
R_{10km}	0.203	275 × 165	B5	0.029	100 × 92	C5	0.051	100 × 100

Table 4- Computational results of the C1P algorithm for randomly generated matrices with almost C1P

Initial information				GA Algo.			
Mat.	Ge.	Initial blocks	Initial C1P	Final Blocks	Blocks improve.	Nbcols	Time(s) C1P
Randomly generated instances from Ruf and Schöbel with sparsity 3%							
B1	100	296	12	264	32	15	10.38
B2	100	325	7	292	33	13	9.72
B3	100	304	10	266	48	14	9.39
B4	100	276	11	250	26	16	10.02
B5	100	270	13	229	41	17	10.39
Randomly generated instances from Ruf and Schöbel with sparsity 5%							
C1	100	456	10	420	36	10	10.37
C2	100	516	7	462	54	9	10.16
C3	100	479	8	436	43	11	10.05
C4	100	482	7	433	49	11	10.04
C5	100	483	7	440	43	14	10.33

Table 5-Computational results of the C1P algorithm for real-world instance matrices

Initial information				**GA Algo.**			
Mat.	**Ge.**	**Initial blocks**	**Initial C1P**	**Final Blocks**	**Blocks improve.**	**Nbcols C1P**	**Time(s)**
Real-world instances with sparsity (1 - 2)%							
$R1km$	100	764	243	759	5	481	27.04
$R2km$	100	1359	52	1301	58	829	92.02
$R3km$	100	1813	38	1727	86	104	116.01
$R5km$	100	1597	34	1471	126	169	32.45
$R10km$	100	389	32	332	57	46	6.83

References

1. Fulkerson, D. and Gross, O. **1965**. "Incidence Matrices and Interval Graphs", *Pacific Journal of Mathematics*, **15**(3): 835-855.
2. Haddadi, S. and Layouni, Z. **2008**. "Consecutive Block Minimization is 1.5-Approximable", *Information Processing Letters*, **108**(3): 132-135.
3. Haddadi, S., Chenche, S., Cheraitia, M. and Guessoum, F. **2015**. "Polynomialtime Local-Improvement Algorithm for Consecutive Block Minimization", *Information Processing Letters*, **115**(6): 612-617.
4. Tan, J. and Zhang, L. **2007**. "The Consecutive Ones Submatrix Problem for Sparse Matrices", *Algorithmica*, **48**(3): 287-299.
5. Blin, G., Rizzi, R. and Vialette, S. **2010**. "*A Faster Algorithm for Finding Minimum Tucker Submatrices*", In Programs, Proofs, Processes. Springer, 69-77.
6. Dom, M., Guo, J. and Niedermeier, R. **2007**. "Approximability and Parameterized Complexity of Consecutive Ones Submatrix Problems2", *In Theory and Applications of Models of Computation*. Springer, pp. 680-691.
7. Dom, M., Guo, J. and Niedermeier, R. **2010**. "Approximation and Fixed-Parameter Algorithms for Consecutive Ones Submatrix Problems", *Journal of Computer and System Sciences*, **76**(3): 204-221.
8. Tucker, A. **1972**. "A Structure Theorem for the Consecutive 1's Property", *Journal of Combinatorial Theory*, Series B **12**(2): 153-162.
9. Hajiaghayi, M. T. and Ganjali, Y. **2002**. "A Note on the Consecutive Ones Submatrix Problem", *Information Processing Letters*, **83**(3): 163-166.
10. Annexstein, F. and Swaminathan, R. **1998**. "On Testing Consecutive-Ones Property in Parallel", *Discrete Applied Mathematics*, **88**(1-3): 7-28.
11. Berge, C. **1972**. "Balanced matrices", *Mathematical Programming*, **2**(1): 19-31.
12. Mecke, S. and Wagner, D. **2004**. "Solving Geometric Covering Problems by Data Reduction", In Algorithms-ESA 2004. Springer, pp. 760-771.
13. Kou, L. T. **1977**. "Polynomial Complete Consecutive Information Retrieval Problems", *SIAM Journal on Computing*, **6**(1): 67-75.
14. Hochbaum, D. S. and Levin, A. **2006**. "Cyclical Scheduling and Multi-Shift Scheduling: Complexity and Approximation Algorithms", *Discrete Optimization*, **3**(4): 327-340.
15. Adam, Z., Turmel, M., Lemieux, C. and Sankoff, D. **2007**. **"**Common Intervals and Symmetric Difference in a Model-Free Phylogenomics, with an Application to Streptophyte Avolution", *Journal of Computational Biology*, **14**(4): 436-445.
16. Alizadeh, F., Karp, R. M., Weisser, D. K. and Zweig, G. **1995**. "Physical Mapping of Chromosomes Using Unique Probes", *Journal of Computational Biology*, **2**(2): 159-184.
17. Gilmore, P. and Hoffman, A. **1964**. "A Characterization of Comparability Graphs and of Interval Graphs", *Canadian Journal of Mathematics*, **16**: 539-548.
18. Meidanis, J., Porto, O. and Telles, G. P. **1998**. "On the Consecutive Ones Property", *Discrete Applied Mathematics*, **88**(1): 325-354.

19. Kendall, D. **1969**. ”Incidence Matrices, Interval Graphs and Seriation in Archeology”, *Pacific Journal of Mathematics*, **28(**3): 565-570.
20. Booth, K. S. and Lueker, G. S. **1976**. “Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms”, *Journal of Computer and System Sciences*, **13**(3): 335-379.
21. Garey, M. R. and Johnson, D. S. **1979**. “*A Guide to the Theory of NP-Completeness*”, WH Freemann, New York.
22. Gargano, M. L. and Lurie, L. **2006**. “A Hybrid Evolutionary Approach to Solving the Archaeological Seriation Problem”, Congressus Numerantium, **180**: p43.
23. Haddadi, S. **2002.** “A note on the NP-hardness of the Consecutive Block Minimization Problem”, *International Transactions in Operational Research*, **9**(6): 775-777.
24. Holland, J. H. **1975**. “Adaptation in Natural and Artificial Systems” Ann Arbor MI: University of Michigan Press.
25. Michalewicz, Z. **1996**. “*Genetic Algorithms + Data Structures = Evolution Programs*”, Springer Science & Business Media.
26. Gen, M. and Cheng, R. **1997**. “*Genetic Algorithms and Engineering Design*”, John Wily and Sons, New York.
27. Di Stefano, L. and Bulgarelli, A. **1999**. “A Simple and Efficient Connected Components Labelling Algorithm”, In Image Analysis and Processing, Proceedings. International Conference on (1999), IEEE, pp. 322-327.
28. Dillencourt, M. B., an an Samet, H. and Tamminen, M. **1990**. “Connected Component Labelling for Arbitrary Binary Image Representations”, In Progress in Image Analysis and Processing: Proceedings of the International Conference on Image Analysis and Processing, World Scientific, p. 131.
29. Ruf, N. and Schöbel, A. **2004**. “Set Covering with Almost Consecutive Ones Property”, *Discrete Optimization*, **1**(2): 215-228.
30. Ruf, N. **2002**. “Locating Train Stations: Set Covering Problems with “C1P” Matrices”, Master's Thesis, University of Kaiserslautern.