

A Heuristic Approach to the Consecutive Ones Submatrix Problem

Rewayda Abo-Alsabeh1* 
Hajem Ati Daham2 
Abdellah Salhi3

1Department of Mathematical Sciences, Faculty of Computer Science and Mathematics, University of Kufa, Iraq 
2Department of Mathematics, College of Education for Pure Science, Al Muthanna University, Iraq 
3Department of Mathematical Sciences, Faculty of Science, University of Essex, UK 
*Corresponding author: ruwaida.mohsin@uokufa.edu.iq 
E-mail addresses: hajem.daham@mu.edu.iq, as@essex.ac.uk

Received 26/5/2021, Revised 2/4/2022, Accepted 3/4/2022, Published Online First 20/7/2022, Published 1/2/2023

This work is licensed under a Creative Commons Attribution 4.0 International License.

Abstract: 

Given a \((0,1)\) matrix, the Consecutive Ones Submatrix (C1S) problem which aims to find the permutation of columns that maximizes the number of columns having together only one block of consecutive ones in each row is considered here. A heuristic approach will be suggested to solve the problem. Also, the Consecutive Blocks Minimization (CBM) problem which is related to the consecutive ones submatrix will be considered. The new procedure is proposed to improve the column insertion approach. Then real world and random matrices from the set covering problem will be evaluated and computational results will be highlighted.

Keywords: Consecutive Block Minimization, Consecutive Ones Property, Consecutive Ones Submatrix, Column insertion, Heuristic Approach.

Introduction: 

The C1S problem on a \((0, 1)\)-matrix is a generalization of the Consecutive Ones Property (C1P). The later has been proposed many decades ago. Fulkerson and Gross\(^1\) suggested it as follows. Given an incidence matrix \(A\), is it possible to rearrange the columns so that all the 1’s in each row are together?

In combinatorial optimization, the C1P property is important since it indicates that the problem utilizes a matrix with this property simpler to solve than the original model. Indeed, such a matrix is totally unimodular. It appears in plenty of applications including computational biology, railway optimization, file organization, and scheduling. The C1P property is also used for ancestral genome reconstruction\(^2,3\). In graph theory, it helps detecting interval and circle graphs\(^4,5,6\).

C1P has been extensively investigated. Kendall\(^7,8\) indicated that the first study of the property was introduced by an archaeologist “Flinders Petrie” in the 19th century. Some heuristic approaches were established for this problem before the first polynomial complexity solution that was proposed by Fulkerson and Gross\(^1\). Tucker\(^9\) showed a substructure characterization of the problem. He used a graph theoretic method to characterize matrices by using forbidden consecutive ones submatrices. In 1976, Booth and Lueker\(^10\) provided a linear-time algorithm for it. They found a permutation that transforms a \((0,1)\)–matrix into one with C1P. Their linear-time sequential algorithm is based on the PQ-tree data structure. Binary matrices can have the property if and only if their PQ-tree exists.

Let \((\alpha, \beta)\)–matrices be the \((0,1)\)–matrices that are having at most \(\alpha\) 1’s and \(\beta\) 1’s in per column and per row, respectively. For the problem with C1S, Hajiaghayi and Ganjali\(^11\) solved it for the \((2,2)\)–matrices in polynomial time and found that the problem for \((2,4)\)–matrices is NP-Hard. These results post the issue of whether the C1S problem is NP-complete for the \((2, 3)\) and \((3, 2)\) matrices. Tan and Zhang\(^12\) answered the question and showed that the two decision versions are NP-complete. They proved the problem is 0.8–approximable for \((2,3)\)–matrices that have no two similar columns. In addition, they illustrated that it can be 0.5–approximable for \((3,2)\) and \((2, \infty)\) matrices. Finally, they showed that the problem’s
approximation for matrices of type \((\infty, 2)\) is NP-Hard within a factor of \(n^d\) when \(d > 0\).

**Preliminaries**

**Definition 1.** Given a \((0, 1)\)–matrix \(A\), a set of consecutive 1 elements \((0\) elements\) in a row of \(A\) is known as a block of ones \((block\ of\ zeros)\) respectively.

**Definition 2.** A \((0, 1)\)–matrix \(A\) is said to have the consecutive ones property if the columns are permuted such that a consecutive block of 1’s occurs per row \(\iota\).

The C1P is satisfied for the columns by matrix transposition.

**Definition 3.** Let \(A\) be a \((0, 1)\)–matrix, finding largest set of columns in \(A\) that construct a submatrix has the C1P is called the C1S problem \(\iota\).2

**Definition 4.** Given a \((0, 1)\)–matrix \(A\), a columns’ permutation of \(A\) which leaves the 1 entries consecutive in all the rows is called a valid permutation. If \(A\) is rearranged by this permutation then it contains the consecutive ones property.

The C1P is demanded as it often provides effective algorithms. Large attention for modifying and transforming a \((0, 1)\)-matrix into a matrix satisfying the C1P has been presented recently. These transformations can be delivered as the following problems: \(\iota\)

1. The problem of finding a maximal number of columns in a \((0, 1)\)-matrix \(A\) which induces a submatrix contains C1P is called the Max-C1S-C (Consecutive Ones Submatrix by Column).
2. The problem of finding a maximal number of rows in a \((1, 0)\)-matrix \(A\) which induces a submatrix contains C1P is called the Max-C1S-R (Consecutive Ones Submatrix by Row).
3. The problem of deleting the minimal number of columns which produces a matrix contains C1P is called the Min-C1S-C (Consecutive Ones Submatrix by Deleting Columns).
4. The problem of deleting the minimal number of rows that produces a matrix contains C1P is called the Min-C1S-R (Consecutive Ones Submatrix by Deleting Rows).
5. The problem of finding the minimal set of 1’s in a matrix that can be changed into 0 results in a matrix with C1P is called the Min-C1-1E (Consecutive Ones by Flipping 1-Entries).

The third and fourth problems are equivalent to the first two. Generally, all the cases are also NP-hard for quite sparse matrices. Traditional methods rely on finding the Tucker forbidden submatrices.8,14 This paper introduces an evolutionary method to solve the C1S problem.

A related problem is the so-called Consecutive Block Minimization (CBM). The goal is to reduce the blocks’ number of ones per row by reordering the matrix’s columns.15 Haddadi et al.15 proposed a polynomial time heuristic to solving the problem. Leonardo CR and others proposed the most recent heuristic work on the problem of CBM. They introduced a heuristic relied on a traditional algorithm in graph theory. They designed a graphical representation to address the CBM problem and detect the reduction of the CBM in the traveling salesman problem. Abo Alsabeh17 suggested a metaheuristic approach for the C1S and CBM problems.

Another related problem is the Simultaneous Consecutive Ones Property (SC1P) where a \((0, 1)\)-matrix contains the C1P for rows and columns simultaneously. Subashini et al.18,19 suggested the classical complexity and fixed parameter tractability of Simultaneous Consecutive Ones Submatrix (SC1S) and Simultaneous Consecutive Ones Editing (SC1E). They proved that the decision cases for the two problems are NP-complete.

Heuristic methods have been applied to enormous problems such as \(\iota\).20,21 The paper is arranged as follows. Studying some previous solution methods in Section 2. Illustrating the suggested column insertion algorithm to handle the problem in Section 3. Computational experience in Section 4. Conclusion in Section 5.

**Previous Solution Procedures**

**Polynomial Time Local Improvement Heuristic for CBM**

The decision problem of the CBM is NP-complete when restricted to \((0, 1)\)–matrices including two ones per row. Nevertheless, Haddadi provided a polynomial time algorithm that finds a permutation where the number of consecutive blocks and the optimum do not vary further than 50%. Haddadi et al.13 provided a polynomial time local search algorithm for the problem. For a binary \(m \times n\) –matrix \(A\), they proposed two \(O(n^2)\) –sized local neighbourhoods search such that the blocks’ number of a neighbour is found in \(O(m)\) time.

- **Improvement by two columns interchange:** Let \(A_\rho\) be a matrix associated with a permutation \(\rho\) in which the entire number of blocks is \(\mu\). Suggest the \(O(n^2)\) –sized neighborhood \(N(\rho)\) to be all the permutations that are produced from \(\rho\) by swapping two columns. Exploring the \(N(\rho)\) for a permutation to give less blocks; if such permutation is not found, the algorithm stops. If an improvement \(\delta\) is reached by interchanging two columns \(\rho(i)\) and \(\rho(j), i \neq j\), (the new permutation is called \(\rho'\)), then update \(\mu \leftarrow \mu - \delta\), \(\rho'(i) \leftarrow \rho(j), \rho'(j) \leftarrow \rho(i)\), and iterate the process with \(\rho'\).
• **Improvement by column-shifting:** Let \( N'(\rho) \) be the permutation set that results from \( \rho \) by inserting one column. In this method, the column is moved from its location and placed between two other columns. \( N'(\rho) \) is searched for a permutation to produce fewer blocks and the search stops when no such permutation exists. Consider that an improvement \( \delta \) is reached by inserting a column \( \rho(i) \), and let the new permutation be \( \rho'(i) \). If all the required updates are carried out, the algorithm is repeated for \( \rho'(i) \). The two procedures are shown below in Algorithms 1 and 2.

**Algorithm 1: Interchange procedure**

1. **Input** Positive integers \( m, n \), a total number of blocks \( \mu \), binary \( m \times n \) matrix \( A \), permutation \( \rho \);
2. for \( i = 1, \ldots, n - 2 \)
3. for \( j = i + 2, \ldots, n \)
4. Interchange two
5. non-adjacent columns \( \rho(i) \) and \( \rho(j) \);
6. Compute \( \delta \);
7. if \( \delta > 0 \)
8. \( \mu \leftarrow \mu - \delta \);
9. Swap \( \rho(i) \) and \( \rho(j) \);
10. end if
11. end for
12. end for

**Algorithm 2: Column shifting procedure**

1. **Input** Positive integers \( m, n \), a total number of blocks \( \mu \), binary \( m \times n \) matrix \( A \), permutation \( \rho \);
2. for \( i = 1, \ldots, n \)
3. for \( j = 1, \ldots, n \) \( (j \neq i - 1, j \neq i, j \neq i + 1) \)
4. Insert column \( \rho(i) \)
5. between columns \( \rho(j - 1) \) and \( \rho(j) \);
6. Compute \( \delta \);
7. if \( \delta > 0 \)
8. \( \mu \leftarrow \mu - \delta \);
9. if \( i > j \) move \( \rho(i) \) to position \( j \) in \( \rho \)
10. else move \( \rho(i) \) to position \( j - 1 \); 
11. end if
12. end if
13. end for
14. end for
15. **Return** Permuted matrix.

The process starts by choosing the first left column from \( A_1 \), say \( \rho(j) \) for \( 1 \leq j \leq p \). This column is inserted between two adjacent columns of \( A_2 \) say \( \rho(i) \), and \( \rho(j + 1) \) for \( i = 1, \ldots, n - p \) and checked. If the column \( \rho(j) \) improves the CIS columns in \( A_2 \), update \( A_2 \) and choose the second column \( \rho(j + 1) \) to insert in the new matrix. If \( \rho(j) \) fails to provide an improvement then it is returned to the right-hand side of \( A_1 \) and \( \rho(j + 1) \) is chosen for insertion.

**Illustration:**

Suppose \( A_2 \) is a submatrix with only two columns. The 0 and 1 in this submatrix have four arrangements \((0 \ 0, \ 0 \ 1, \ 1 \ 0, \ 1 \ 1)\). Inserting column \( \rho(j) \) in \( A_2 \) produces eight cases that are suggested by\(^3\), \((0 \ 1 \ 0, \ 0 \ 0 \ 0, \ 1 \ 0 \ 1, \ 1 \ 1 \ 1, \ 1 \ 1 \ 0, \ 0 \ 1 \ 1, \ 1 \ 0 \ 0, \ 0 \ 0 \ 1)\). \( A_2 \) is destroyed in one case, when the sequence of columns \( \rho(i) \ \rho(j) \ \rho(i + 1) \) is \( 1 \ 0 \ 1 \), otherwise, the matrix still has the CIS.

If number of columns of \( A_2 \) is greater than 2, then another destructive case appears. That is when the sequence of columns \( \rho(i) \ \rho(j) \ \rho(i + 1) \) is \( 0 \ 1 \ 0 \) and there is another block of ones in the same row (say \( 1 \ 0 \ 1 \ 0 \)), this row will lose the CIS property. Table 1 shows the checking of the different cases. The destructive cases are referred to as false and other cases that do not destroy \( A_2 \) as true. Column \( \rho(j) \) fits between \( \rho(i) \) and \( \rho(i + 1) \) if the result of checking all the rows is true.
Table 1. Cases of checking a column inserted between two columns.

<table>
<thead>
<tr>
<th>$\rho(i)$</th>
<th>$\rho(j)$</th>
<th>$\rho(i+1)$</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>false</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
</tbody>
</table>

If an improvement $\gamma$ is achieved by inserting a column $\rho(j)$ in $A_2$, the new permutation is called $\rho$. The matrix $A_{\rho}$ will be updated to $A_{\rho'}$ and the process repeated with $\rho'$. The number of blocks which result from inserting an arbitrary column between two other columns can be evaluated in $O(m)$ operations, see. The size of searching $N''(\rho)$ costs time $O(\sigma)$, where $\sigma$ is clarified below. That is because there is one possibility to insert $\rho(j)$ between $\rho(i)$ and $\rho(i+1)$. If it fits, then $\rho(j+1)$ needs two possibilities, and so on. It can say that the size of possible columns for insertion is $\leq (n-2)(n-1)/2$.

**Lemma 1.** The complexity of the column insertion algorithm is $O(m\sigma p)$ where $\sigma$ is the size of the neighborhood $N''(\rho)$ to explore.

Proof. Searching the neighborhood $N''(\rho)$ costs $O(\sigma)$ operation where

$$\sigma = \sum_{N=n-p}^{n-1} N - 1,$$

The integer number $N$ starts with the $n-p$ columns of $A_2$. For every neighbor, the calculations of checking the cases for all the rows cost $O(m)$. As long as the number of C1S columns $\beta$ cannot be greater than $n$, and no less than $n-p$ columns, the finiteness of the inserting procedure is guaranteed and the number of improvements is at most $n - (n - p) = p$ in the worst case.

Algorithm 3: Column insertion procedure

1. **Input** Positive integers $m, n$, a total number of blocks $\beta$, binary $m \times p$ submatrix $A_1$, $m \times (n - p)$ submatrix $A_2$, with the C1P, and permutation $\rho$;
2. for $j = 1, \ldots, p$
3. for $i = 1, \ldots, (n - p) - 1$
4. Insert column $\rho(j)$ between $\rho(i)$ and $\rho(i + 1)$;
5. Check $\rho(i) \rho(j) \rho(i + 1)$;
6. if the result of the check is true for all $m$
7. $\beta \leftarrow \beta + \gamma$;
8. Move $\rho(j)$ between $\rho(i)$ and $\rho(i + 1)$ and update $A_2$;
9. end if
10. end for
11. end for
12. **Return** Permuted matrix.

**Improving the number of blocks** The procedure of column insertion that is used to search the neighbourhood $N''(\rho)$ for a permutation to maximize the C1S submatrix also minimize the number of blocks. Mentioning analogous arguments as in Lemma 1, the complexity of inserting the columns to find the minimum number of blocks is $O(m(a(g - m)))$, where $g$ is the number of 1's in $A$. Also, since the number of blocks $\mu$ cannot be greater than $g$, and no less than $m$, the finiteness of the insertion method is ensured and the number of improvements is at most $g - m$ in the worst case. This algorithm can be implemented directly on the matrix $A$ by extracting the C1S matrix then fill the reminder columns.

**Computational Experience**

**Implementing Column Insertion Algorithm**

The matrix is separated into two submatrices, one having C1S (columns $\geq 2$), then the column insertion algorithm is applied. From our experiments, it is found that the column insertion algorithm gives better results on larger C1S matrices. The results are promising: there are fewer blocks, and more columns having the C1P obtained in a reasonable time.

Overall, the heuristic is applied to many various matrices from real world data or generated by random. The generated square nonsymmetric matrices of the Set Covering Problem (SCP) are not tested for the C1P, then their optimums are not recognized. Therefore, the quality of the outcomes cannot be discussed. Randomly 10 matrices for different sizes are generated, the algorithm is performed then the average of the results of each size is taken. The remaining data, the real world
matrices, \((R_{1km} - R_{10km})\) which are shown in Table 2, come from the stop location problem, supplied by a German railway company\(^{22}\). This problem is written as SCP problem. The instances provide binary matrices which are supposed to contain almost C1P. The matrices B and C of small size that are generated by Ruf and Schöbel\(^{22}\), are sparse and almost contain the consecutive ones property with density 3% and 5%, respectively, the results are shown in Table 4. The heuristic is tested by the following ways:

Directly extract the C1S submatrix of a given matrix, and then apply Algorithm 3. Concerning the five real-world matrices, only two columns are separated as a C1S submatrix then the algorithm is applied. Results are in columns 5 to 7 of Tables 3 and columns 4 to 7 of Tables 4 and 5.

Note that Tables 3, 4, and 5 show the average number of blocks and columns. The results of Table 3 show that Algorithm 3 gives a larger number of columns with C1P and fewer blocks than the original matrix. The results of Table 4 show that almost (30-50)% of the columns of the B and C matrices have C1P. Table 5 includes the outcomes on the real world matrices. Matrix \(R_{1km}\) has the C1P where final blocks’ number is equal to \(m\) and the columns’ number is equal to \(n\), hence optimal. Concerning the remainder matrices of real world data, the last numbers of blocks and columns are close to the lower bound \(m\) and upper bound \(n\) respectively. One can say that the column insertion algorithm is fast with respect to computation time.

To compare with the results of the CBM problem from\(^{13}\), with respect to the number of blocks, it is found that the results do not differ a lot from theirs. The last column shows

### Table 2. Test problem statistics.

<table>
<thead>
<tr>
<th>Real-world matrices</th>
<th>Randomly generated instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat.</td>
<td>Dens. (%)</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>(R_{1km})</td>
<td>0.002</td>
</tr>
<tr>
<td>(R_{2km})</td>
<td>0.014</td>
</tr>
<tr>
<td>(R_{3km})</td>
<td>0.022</td>
</tr>
<tr>
<td>(R_{4km})</td>
<td>0.043</td>
</tr>
<tr>
<td>(R_{5km})</td>
<td>0.203</td>
</tr>
</tbody>
</table>

### Table 3. The column insertion performance for non-symmetric matrices.

<table>
<thead>
<tr>
<th>Mat.</th>
<th>Dens.</th>
<th>Initial blocks</th>
<th>Initial CIP (%)</th>
<th>Final Blocks</th>
<th>Blocks improve.</th>
<th>Nbcols</th>
<th>CIP</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
<td>209</td>
<td>17</td>
<td>148</td>
<td>60</td>
<td>83</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>444</td>
<td>10</td>
<td>331</td>
<td>67</td>
<td>47</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>863</td>
<td>8</td>
<td>788</td>
<td>172</td>
<td>26</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>781</td>
<td>16</td>
<td>651</td>
<td>140</td>
<td>93</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>4</td>
<td>1379</td>
<td>8</td>
<td>1191</td>
<td>189</td>
<td>60</td>
<td>0.40</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>3248</td>
<td>8</td>
<td>896</td>
<td>351</td>
<td>32</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>2</td>
<td>5189</td>
<td>11</td>
<td>4538</td>
<td>651</td>
<td>118</td>
<td>5.19</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>4</td>
<td>9612</td>
<td>4</td>
<td>8856</td>
<td>766</td>
<td>67</td>
<td>4.82</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>10</td>
<td>22021</td>
<td>6</td>
<td>20826</td>
<td>893</td>
<td>34</td>
<td>5.96</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4. The column insertion performance for matrices having almost C1P.

<table>
<thead>
<tr>
<th>Mat.</th>
<th>Initial blocks</th>
<th>Initial CIP (%)</th>
<th>Final Blocks</th>
<th>Blocks improve.</th>
<th>Nbcols</th>
<th>CIP</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrices of sparsity 3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>296</td>
<td>12</td>
<td>267</td>
<td>29</td>
<td>47</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>325</td>
<td>7</td>
<td>295</td>
<td>30</td>
<td>39</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>304</td>
<td>10</td>
<td>279</td>
<td>25</td>
<td>38</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>276</td>
<td>11</td>
<td>255</td>
<td>21</td>
<td>42</td>
<td>0.13</td>
<td></td>
</tr>
<tr>
<td>B5</td>
<td>270</td>
<td>13</td>
<td>242</td>
<td>28</td>
<td>48</td>
<td>0.13</td>
<td></td>
</tr>
</tbody>
</table>

| Matrices of sparsity 5% |
| C1   | 456            | 10              | 437          | 19              | 29     | 0.14|         |
| C2   | 516            | 7               | 496          | 20              | 22     | 0.15|         |
| C3   | 479            | 8               | 463          | 16              | 23     | 0.15|         |
| C4   | 482            | 7               | 468          | 24              | 23     | 0.13|         |
| C5   | 483            | 7               | 468          | 15              | 27     | 0.14|         |
Their results. Also, the numbers of columns with C1P are not far from the columns’ sizes of the matrices.

The three tables illustrate that:
1. The outcomes of Algorithm 3 do not entirely depend on the size of the C1S; they as well rely on the matrices’ structures.
2. Computation time relies on the size of the matrix and the density. The algorithm performs well on matrices with more sparsity.
3. Reducing the blocks does not lead to producing a sizeable C1S submatrix. It is noticed that developing the C1S enhances the CBM. However, the reverse may not be correct since the submatrix size depends on the location of the destructive column.

### Table 5. The column insertion performance for real-world instance matrices.

<table>
<thead>
<tr>
<th>Mat.</th>
<th>Initial blocks</th>
<th>Initial C1P</th>
<th>Final blocks</th>
<th>Column insertion Blocks improve.</th>
<th>Nbcols C1P</th>
<th>Time(s)</th>
<th>Algo. (1, 2) Final blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_{1km}</td>
<td>764</td>
<td>243</td>
<td>757</td>
<td>7</td>
<td>707</td>
<td>2.565</td>
<td>757</td>
</tr>
<tr>
<td>R_{2km}</td>
<td>1359</td>
<td>52</td>
<td>1210</td>
<td>157</td>
<td>882</td>
<td>6.940</td>
<td>1206</td>
</tr>
<tr>
<td>R_{3km}</td>
<td>1813</td>
<td>38</td>
<td>1487</td>
<td>326</td>
<td>859</td>
<td>8.566</td>
<td>1461</td>
</tr>
<tr>
<td>R_{5km}</td>
<td>1597</td>
<td>34</td>
<td>1185</td>
<td>412</td>
<td>568</td>
<td>3.152</td>
<td>1143</td>
</tr>
<tr>
<td>R_{10km}</td>
<td>389</td>
<td>32</td>
<td>281</td>
<td>108</td>
<td>159</td>
<td>0.081</td>
<td>280</td>
</tr>
</tbody>
</table>

### Conclusion:
A heuristic method for solving the C1S problem is represented. A column insertion procedure was suggested to address the problem. The CBM problem is solved by Algorithm 3 and the maximum C1S submatrix is improved by using a polynomial time local algorithm with a complexity of $O(mσ)$. The same algorithm is used for solving the minimum consecutive blocks problem in which the complexity for finding only the CBM is $O(mσ(g − m))$. The algorithm is applied to a set of real world matrices and randomly generated matrices from set covering. The outcomes present that large submatrices having the C1P can be detected. However, as the optimums are unknown for these matrices, it is impossible to say how far the solutions got by our procedure are actually resulted from them.

### Acknowledgments:
The authors would like to thank Schöbel, one of the authors of that provided us with real-world data. Also, to Omar Kirikcki for supplying us with the random nonsymmetric matrices.

### Authors’ declaration:
- Conflicts of Interest: None.
- We hereby confirm that all the Figures and Tables in the manuscript are mine ours. Besides, the Figures and images, which are not mine ours, have been given the permission for re-publication attached with the manuscript.

- Ethical Clearance: The project was approved by the local ethical committee in University of Kufa.

### Authors’ contributions statement:
R. A. A. worked on the experiments and got the results above. A. S. helped with his experience to make the work straightforward and clear. He also revised the paper. H. A. D. searched for the literature on the problem and helped us write the paper.

### References:
8. Safe MD. Characterization and linear-time detection of minimal obstructions to concave-round graphs and


C1S

نهج إرشادي لمشكلة

رويدة رزاق محسن

حاجم حات دحام

عبد الله صالح

قسم الرياضيات، كلية علوم الحاسوب والرياضيات، جامعة الكوفة، العراق

قسم الرياضيات، كلية التربية للعلوم الصرفة، جامعة المثنى، العراق

قسم الرياضيات، كلية العلوم، جامعة اسكس، المملكة المتحدة

الخلاصة:

أعطيت مصفوفة (0،1)، تم اقتراح مسألة المصفوفة الجزئية ذات الاحمادي المتعاقبة والتي تهدف إلى إيجاد تدريب للأعدة التي تزيد

من عدد الاعددة التي تحتوي معًا على قالب واحد فقط من الاعدادات المتعاقبة في كل صف. سيتم اقتراح أسلوب الاستدلال لحل المسألة. كما

سير دراسة مسألة تقليل القوالب المتتالية ذات الصلة بمسألة المصفوفة الجزئية ذات الاعدادات المتعاقبة. تم اقتراح أجزاء جديدة للتحسين طريقة

إدراج الاعدادات. يتم بعد ذلك تقييم مصفوفات العالم الحقيقي ومصفوفات متنوعة عشوائيًا من مسألة عدد المجموعات وعرض النتائج الحسابية.

الكلمات المفتاحية: تغيير الكتل المتتالية، خاصية الاعدادات المتتالية، مصفوفة الاعدادات المتتالية الفرعية، أجزاء الاعدادات.