A Dual-based Combinatorial Algorithm for Solving Cyclic Optimization Problems

Hesham K. Alfares*

Systems Engineering Department, PO Box 5067, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

Received: July 20, 2012; Accepted: August 21, 2012; Revised: August 18, 2012

Abstract: This paper describes Patent Number U.S. 8,046,316 B2, titled “Cyclic Combinatorial Method and System”, issued by the US Patents and Trademarks Office on October 25, 2011. The patent is based on a combinatorial algorithm to solve cyclic optimization problems. First, the algorithm identifies cyclically distinct solutions of such problems by enumerating cyclically distinct combinations of the basic variables. In combinatorial terminology, this stage of the algorithm addresses the following question: given \( n \) cyclic objects, how many cyclically distinct combinations of \( m \) (\( m \leq n \)) objects can be selected? Integrating the operations of partition and cyclic permutation, a procedure is developed for generating cyclically distinct selections (dual solutions). Subsequently, rules are described for recognizing the set of dominant solutions. Finally, primal-dual complementary slackness relationships are used to find the primal optimum solution. This patent has many potential applications in optimization problems with cyclic 0-1 matrices, such as network problems and cyclic workforce scheduling. The patent’s applicability has been illustrated by efficiently solving several cyclic labor scheduling problems.

Keywords: Binary matrices, combinatorial algorithms, cyclic selection, cyclic scheduling, integer programming, optimization.

1. INTRODUCTION

Cyclic matrices are frequently encountered in optimization problems, such as network problems, the traveling salesman problem, labor scheduling problems, and certain reliability problems. A circular \( n \times n \) matrix \( A \) is fully specified by one vector \( a \), which is the first row (column) of \( A \). The remaining rows (columns) of \( A \) are each cyclic permutations of \( a \), such that row (column) \( k \) is obtained by applying a shift of length \( (k - 1) \) to the vector \( a \) [1, 2]. The cyclic property of the constraint matrix can be exploited in order to develop efficient solutions of optimization problems. For example, Bartholdi et al. [3], Koop [4], Uebe et al. [5], and Boctor and Renaud [6] utilize the cyclic property of the constraint matrix to devise models and efficient solution procedures for cyclic optimization problems.

According to Bartholdi et al. [3], a 0-1 vector is said to be circular if its 1’s occur consecutively, where the first and last entries are considered to be consecutive. A matrix is called column (row) circular if its columns (rows) are circular. Bartholdi et al. [3] present two methods for solving a cyclically structured integer programming model: network flows, and special rounding of the continuous LP solution. Oswald and Reinelt [7] present a polynomial time algorithm for converting a given general 0-1 matrix to one with the consecutive ones property. Ruf and Schöbel [8] develop a reformulation and branching procedure for solving set covering problems with coefficient matrices almost having the consecutive ones property.

Uebe et al. [5] show that, if the matrix \( A \) is cyclic, then the continuous linear programming solution to the set covering problem: minimize \( Cx \), subject to \( Ax = r \), is integer, although not necessarily non-negative, provided \( r \) satisfies certain integrality conditions. The integral structure is maintained and exploited by a branch and bound interactive solution procedure. Boctor and Renaud [6] discuss the column-circular subsets-selection problem (CCSSP), whose objective is to select a minimum-cost collection of subsets from a circular set of elements. CCSSP is applicable for selecting subsets in routing problems and for selecting subsets of tasks in scheduling problems. Boctor and Renaud [6] transform CCSSP into a bounded number of sub-problems and use a dynamic programming algorithm to solve these sub-problems.

Cyclic sequences and permutations are important problems in combinatorial theory. An example of a cyclic combinatorial problem is the necklace problem, which is stated by Krishnamurthy [9] as follows: how many distinct necklace patterns are possible with \( n \) beads, available in \( r \) different colors? The algorithm presented in this paper addresses a simpler problem: how many distinct necklace patterns are possible with \( n \) beads, \( m \) of which are black, and the rest are white? If mirror image necklaces are considered equivalent, the question becomes: what is the number of bracelets (reversible necklaces) that can be formed with \( n \) black and \( m \) white beads, \( m \) of which are black? Both the necklace and the bracelet problems are directly applicable to optimization problems with cyclic 0-1 matrices. The two colors of the beads correspond to 0s and 1s in a cyclic 0-1 constraint coefficient matrix.

The algorithm to be presented in this paper is generally applicable to all integer linear programming (ILP) problems of the form: minimize \( 1^T x \), subject to \( Ax \geq r \), where \( x \geq 0 \).
and integer, and \( A \) is a cyclic 0-1 matrix. These ILP problems are known to be difficult to solve (NP hard) and have numerous applications in many areas of optimization, such as cyclic scheduling and network problems.

The cyclic combinatorial optimization algorithm has two main steps. First, partition and permutation tools are combined for selecting cyclically distinct subsets out of a number of cyclic objects. Considering the columns (variables) of these matrices as cyclic objects, the algorithm can be used to select only cyclically distinct combinations of columns (basic dual variables). The process of determining these cyclic combinations is exactly equivalent to solving the necklace or the bracelet problem. Subsequently, simple rules are introduced to identify a small subset of dominant dual solutions. The dominant dual solutions can be used to calculate bounds on the objective function, impose efficient cuts on the ILP model, or even completely solve the given cyclic 0-1 optimization problem using primal-dual relationships.

The remainder of this paper is organized as follows. Relevant recent patents on cyclic and binary optimization and computation are reviewed in Section 2. A description of the cyclic selection algorithm is presented and an example is described in Sections 3 and 4 respectively. The number of cyclically distinct combinations of subsets selected from cyclic objects is discussed in Section 5. The use of the cyclic selection algorithm in solving cyclic optimization problems is described in Section 6. Finally, conclusions and future developments are presented in Section 7.

2. PATENTS ON CYCLIC AND BINARY OPTIMIZATION

No other patent is found that addresses the same particular problem as U.S. 8,046,316, titled “Cyclic Combinatorial Method and System” [10]. However, several recent patents are in the same general area of cyclic, binary, combinatorial, and computational optimization. As these patents are relevant to provide a wider view of the field, they are described below.

Patent U.S. 6,519,738, titled: “Method and Apparatus for High-Speed CRC Computation Based on State-Variable Transformations” [11], describes a method and an associated system for calculating the cyclic redundancy code (CRC) of a communication data stream. By taking \( M \) bits at a time, the method produces a throughput rate which is \( M \) times the speed of the one bit-at-a-time CRC computation system, at the same circuit clock speed. The method applies a linear transformation to the polynomial input sequence to produce a transformed state vector in order to efficiently compute CRC for the sequence. Patent U.S. 6,836,869, titled “Combined Cyclic Redundancy Check (CRC) and Reed-Solomon (RS) Error Checking Unit” [12], integrates CRC and RS codes in a two-stage combinatorial circuit that performs arithmetic operations. Bitwise operations are carried out by input and output registers, sets of input and output registers, AND and XOR logic gates, and multiplexer units process the data blocks and carry out bitwise operations.

Patent U.S. 7,024,441, titled “Performance Optimized Approach for Efficient Numerical Computations” [13], presents a system that improves computational efficiency. Specifically, the patent develops an optimized process and associated hardware structure for improving the multiply and accumulate (MAC) operations such as the “Discrete Cosine Transform” (DCT) procedure. Patent U.S. 7,155,656, titled “Method and System for Decoding of Binary Shortened Cyclic Code” [14], enhances the computational efficiency of decoding binary shortened cyclic codes by improving the syndrome calculation step. The proposed system is illustrated by both a hardware system and a Digital Signal Processor (DSP) system.

Patent U.S. 7,657,452, titled “System and Method for Tour Optimization” [15], determines the optimal allocation of loads in two-segment vehicle tours. Given data on the load, origination point, and destination point of each segment, the load fitness functions are evaluated, ranked, and optimally reassigned among the two segments. Patent U.S. 7,735,086, titled “Methods and Arrangements for Planning and Scheduling Change Management Requests in Computing Systems” [16], presents efficient algorithms to schedule requests for change (RFC) in computing systems. The algorithm first determines whether or not a given RFC is acceptable. All acceptable RFCs are divided into discrete tasks that are assigned to specific start times on designated servers.

Patent U.S. 7,953,062, titled “Enhanced Channel Interleaving for Optimized Data Throughput” [17], devises a scheme for increasing the speed of data transmission to a remote station. Faster transmission is achieved by grouping code symbols into multi-slot packets, while efficient decoding is maintained by decoding only a subset of those multi-slot packets. Patent U.S. 8,175,330, titled “Optimization Methods for the Insertion, Protection, and Detection of Digital Watermarks in Digital Data” [18], develops optimal models for digital watermarks used in the transmission, distribution and storage of multimedia data. Based on the digital signal frequency and features, watermark parameters and insertion locations are designed to maximize signal quality and tamper-resistance.

3. THE CYCLIC SELECTION PROCEDURE

The problem is stated as follows: given an optimization problem with a cyclic 0-1 \( n \times n \) matrix, cyclically enumerate all dual solutions to identify the dominant solutions. For such problems, the dual variables are cyclic objects. Cyclic objects are those that have the cyclic property, i.e. they can be arranged in a circle; such that the last object is followed by the first object. Examples of cyclic objects include the hours of the days, the days of the week, and the seasons of the year. Combinations (sequences) of cyclic objects are considered either cyclically equivalent or distinct depending on whether or not they are the same or different when they are wrapped around a circle. For example, the sequence (spring-summer-fall-winter) is cyclically equivalent to the sequence (fall-winter-spring-summer), but cyclically distinct from the sequence (fall-summer-winter-spring).

The cyclic algorithm applies the operations of partition and enumeration on the cyclic objects (dual variables) in two stages. First, for each value of \( m (m \leq n) \), the algorithm finds all possible partitions of the \( n \) dual variables into \( m \) parts. For each partition, the algorithm determines all cyclically distinct permutations to of the \( m \) parts.
Given the total number of cyclic objects (dual variables) $n$, and the number of selected objects $m$ ($m \leq n$), the combinatorial algorithm is designed to achieve two specific objectives. The first objective is to generate all cyclically distinct combinations of $m$ objects that can be selected. The second objective is to determine a small set of dominant solutions that include the optimum solution for the given problem. The first objective is achieved mainly by a cyclic selection procedure that combines two basic combinatorial tools: (i) partition of an integer, and (ii) permutation of objects. Therefore, these two tools will be briefly discussed before the steps of the algorithm are described.

Hall [19] defines the partition of a positive integer $n$ into $m$ parts ($m \leq n$) as a representation of $n$ as a sum of positive integers, expressed as

$$n = v_1 + v_2 + \ldots + v_m,$$

where

$$v_i > 0, \quad i = 1, \ldots, m.$$

Denoting the number of (unordered) partitions of $n$ into $m$ parts by $P_m(n)$, it is easy to show that

$$P_1(n) = P_m(n) = 1,$$

$$P_2(n) = \lfloor n/2 \rfloor,$$

where $\lfloor a \rfloor$ is the largest integer less than or equal to $a$.

The remaining values of $P_m(n)$ can be obtained by the recursive relationship:

$$P_m(n) = P_m(n - m) + P_{m-1}(n - m) + \ldots + P_1(n - m)$$

Wells [20] defines a permutation of $n$ objects, as an order of the $n$ objects. There are $n!$ permutations of $n$ distinct objects. A new scheme will be developed for generating cyclically distinct permutations. Wells [20] defines a combination of $n$ distinct objects taken $m$ at a time, as an $m$-combination of $n$ elements, as a selection of $m$ of the $n$ objects without regard to order. When the selection is made without replacement (i.e. when repetitions of the objects are not allowed), the number of $m$-combinations of $n$ elements is

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}.$$

The proposed selection procedure combines partition with permutation, and adds safeguards to eliminate cyclically redundant combinations. Effectively, a scheme for generating cyclically distinct permutations is applied to $m$ partitions of an integer $n$. The steps of the cyclic selection procedure are briefly described as follows:

**Step 0.** Given integers $n$ and $m$ ($m \leq n$),

**Step 1.** Partition $n$ into $m$ parts $(v_1, v_2, \ldots, v_m)$ using a procedure that generates monotonic (non-decreasing or non-increasing) partitions.

(a) If $n \leq 3$, or $n - m \leq 1$, or $v_1 = v_m$ (all parts are equal), store the current combination and go to the next partition,

(b) Otherwise, go to Step 2.

**Step 2.** Fixing $v_1$, find the next permutation of the $m - 1$ remaining parts $(v_2, \ldots, v_m)$. Cyclically compare the resulting (current) combination (including $v_1$) with all stored combinations of the current partition.

(a) If there is no match, store the current combination and go to the next permutation,

(b) Otherwise, ignore the current combination and go to next permutation.

The above description of the cyclic selection procedure is purposely kept brief in order to avoid clouding the overall view of the procedure with details. In order for the description to be complete, however, more detailed verbal description of steps 1 (partition) and 2 (permutation) will be given. Moreover, a simplified flowchart depicting the process is shown in Fig. (1).

In step 1, any monotonic partition procedure can be used. In our implementation of the optimization algorithm, the procedure described by Lehmer [21] was used to generate non-decreasing partitions (1 $\leq v_1 \leq v_2 \leq \ldots \leq v_m$), beginning with $n = n$ and ending with $n = 1 + 1 + \ldots + 1$. No permutation is required for the three cases specified in Step 1(a), because all permutations for these cases are cyclically equivalent.

In step 2, Johnson’s [22] adjacent-mark permutation procedure was used in the implementation of the optimization algorithm. For each current combination, cyclic comparisons are performed as follows. Since $v_1$ is fixed in all permutations, all sequences will start with $v_1$ (the smallest partition). Therefore, any element in the sequence whose value is equal to $v_1$ will be considered as a starting point for comparisons with previously stored permutations. For each $v_i = v_1, i = 2, \ldots, m$, start with $v_i$ and proceed to complete the cycle, comparing the resulting sequence with all stored sequences for the current partition.

If the 0-1 matrix consists of reversible (symmetric) cycles, then we need to proceed from each $v_i$ in both directions, constructing two different sequences. The forward sequence to be compared is $(v_1, v_{i+1}, v_{i+2}, \ldots, v_m, v_1, v_2, \ldots, v_{i-1})$, while the reverse sequence is $(v_1, v_{i-1}, v_{i-2}, \ldots, v_1, v_m, v_{m-1}, v_{m-2}, \ldots, v_{i+1})$. A reversible (symmetric) 0-1 cycle can be turned over, producing the same cycle. Reversible 0-1 cycles are symmetric, i.e., the cycle can be divided into two identical (mirror image) halves. Examples of reversible (symmetric) and irreversible (asymmetric) 0-1 matrices and corresponding cycles are shown on Fig. (2).

### 4. A CYCLIC SELECTION EXAMPLE

Let us consider the case ($n = 7, m = 4$) to illustrate how the cyclic selection procedure works. In step 1, three possible partitions of 7 into 4 parts $(v_1, v_2, v_3, v_4)$ are determined:

$(1, 1, 1, 4), (1, 1, 2, 3),$ and $(1, 2, 2, 2)$.

To illustrate step 2, let us consider only the second partition $(1, 1, 2, 3)$. Thus, fixing “$v_1 = 1$”, there are $3! = 6$ permutations of $(v_2, v_3, v_4)$ as follows:

$(1, 1, 2, 3), (1, 1, 3, 2), (1, 2, 1, 3), (1, 2, 3, 1), (1, 3, 1, 2),$ and $(1, 3, 2, 1)$. 

Fig. (1). Simplified flowchart of the cyclic selection procedure.
Storing the first sequence as the first permutation, the remaining permutations are compared to it by starting from each “vi = 1” and proceeding in both directions. For example, the second permutation above (1, 1, 3, 2) can be expressed as:

Forward direction: (1, 1, 3, 2), (1, 3, 2, 1)
Reverse direction: (2, 1, 3, 1), (1, 2, 1, 3)

From the forward rotation, the permutation (1, 1, 3, 2) is cyclically equivalent to (1, 3, 2, 1). Since the reverse rotation does not produce the original sequence, this permutation is not reversible. However, if the direction of rotation is ignored (reverse cycles are considered equivalent), then (1, 1, 3, 2) is also cyclically equivalent to (1, 1, 2, 3). In any case, the permutation (1, 1, 3, 2) is ignored, and the procedure then moves to check the four remaining permutations. At the end of this process, only three cyclically distinct permutations remain: (1, 1, 2, 3), and (1, 2, 1, 3).

The procedure represents each combination as a sequence of partitions \( (v_1, v_2, \ldots, v_m) \) of \( n \) into \( m \) parts, each of \( k \) blocks of \( \frac{n}{d} \). The algorithm’s representation can be converted to specific choices from the set \( (1, \ldots, n) \). For example, given that \( n = 7 \) and \( m = 4 \), the sequence of partitions \( (v_1, v_2, v_3, v_4) = (1, 2, 1, 3) \) corresponds to selecting the numbers \( 1, 1 + 2, 1 + 2 + 1, 1 + 2 + 1 + 3 \) from the numbers \( 1, 2, \ldots, 7 \). The selection \((1, 3, 4, 7)\) is cyclically equivalent to \((2, 4, 5, 1)\), or \((7, 2, 3, 6)\), and so on. Alternatively, this sequence can be represented as selecting from the cyclic set \( (1, \ldots, n) \) the numbers \((k, k + 1, k + 1 + 2, k + 1 + 2 + 1) \mod n\), or \((k, k + 2, k + 2 + 1, k + 2 + 1 + 3) \mod n\), and so on. As a convention, the cyclic combination \((v_1, v_2, \ldots, v_m)\) will be represented by the sequence \((k, k + v_2, k + v_2 + v_3, \ldots, k + v_2 + v_3 + \ldots + v_m) \mod n\), for \( k = 1, \ldots, n \).

The number of cyclically distinct selections is given by the well-known formula in equation (5). Weisstein [23] defines an \( a \)-array necklace of length \( n \) as a string of \( n \) characters, each of \( a \) possible types, where rotation is ignored, such that \((b_1 b_2 \ldots b_n)\) is equivalent to \((b_{k+1} b_{k+2} \ldots b_n b_1 b_2 \ldots b_k)\) for any \( k \). A free or reversible necklace (one that can be turned over) is called a bracelet. For free necklaces, opposite orientations (mirror images) are regarded as equivalent, so the necklace can be picked up and flipped over. The number of such bracelets (reversible necklaces) with \( n \) beads, \( m \) of which are black and \( n-m \) are white, is given by Jibladze [24] as:

\[
B_{m}(n) = \frac{1}{2n} \sum_{d|m,n} \phi(d) \left( \frac{n}{d} \right) \left( \frac{1}{2} \frac{n - 1 - (-1)^m}{m^{\frac{2}{2}}} \right)
\]

where \([a]\) denotes the integer part of \(a\).

In fixed necklaces, reversal of strings is respected, i.e., opposite orientations are not considered equivalent. Elashvili et al. [25] define \( a_0(n, m) \) as the number of necklaces with \( n \) black and \( m \) white beads (i.e. total number of beads equal to \( n + m \)). Therefore, the number of binary (two-color) necklaces with \( n \) beads of which \( m \) are black is \( a_0(n, n - w m) \), which is given by

\[
C_{m}(n) = \frac{1}{n} \sum_{d|m,n} \phi(d) \left( \frac{n}{d} \right) \left( \frac{m}{d} \right)
\]

where
\( \phi(d) \) = Euler’s totient function, i.e., number of numbers less than \( d \) which are coprime to \( d \). For example \( \phi(1) = |\{1\}| = 1 \), and \( \phi(6) = |\{1, 5\}| = 2 \).

Using the above expressions for the number of cyclic selections, simpler expressions can be determined for three special cases. The first case is when \( m \leq 3 \), and the second case is when \( n - m \leq 1 \). In both cases, no permutations are required, and the number of cyclic selections is simply the number of partitions \( P_m(n) \). The third special case applies to the remaining (unselected) \( n - m \) objects. For the linear (non-cyclic) case, the number of combinations of the selected \( n \) items is equal to the number of combinations of the remaining \( n - m \) items, or \( \binom{n}{m} = \binom{n}{n-m} \). Obviously, this is also true for the cyclic case. The three cases are summarized as follows:

\[
B_m(n) = C_m(n) = P_m(n), \quad m \leq 3, \text{ or } n - m \leq 1 \quad (8)
\]

\[
B_{n-m}(n) = B_m(n), \quad C_{n-m}(n) = C_m(n) \quad (9)
\]

For the remaining cases of \( n \) and \( m \), the values of \( B_m(n) \) and \( C_m(n) \) must be calculated by (6) and (7), respectively. Alternatively, these values can be obtained by applying the steps of the cyclic selection procedure. In order to calculate these values, and to facilitate the implementation of these steps, the algorithm was coded as a FORTRAN program. The program was run for a small representative sample of \( n \) and \( m \), and the values of \( B_m(n) \) and \( C_m(n) \) were recorded. These values are shown in Tables 1 and 2.

### 6. THE CYCLIC OPTIMIZATION ALGORITHM

The cyclic selection procedure can be used as step in an overall methodology to solve a general class of important and NP-hard cyclic optimization problems. Given an integer linear programming (ILP) problem of the form:

Minimize \( Z = 1^T x \) \quad (10)
subject to \( Ax \geq r \) \quad (11)

where \( x \geq 0 \) and integer, and \( A \) is a cyclic \( 0-1 \times n \times n \) matrix.

This proposed cyclic optimization algorithm employing the cyclic selection procedure involves the following steps:

1. Ignoring integrality restrictions, the dual LP model is constructed as:

Maximize \( W = r^T y \) \quad (12)
subject to \( A^T y \leq 1 \) \quad (13)

where \( y \geq 0 \)

2. All cyclically-distinct solutions of the dual model are determined. To solve the dual problem we allocate the unit resource – right hand side of dual constraints (13) which is equal to 1 - among the dual variables (columns of \( A^T \)) in order to maximize the dual objective \( W \). We may allocate the unit resource among any number \( m \) of selected columns, where \( m = 1, \ldots, n \). Since the \( n \) columns (variables) of the dual matrix \( A^T \) are cyclic, the number of cyclically-distinct dual solutions corresponds to the number of cyclically-

### Table 1.

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>( \Sigma B(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>10</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>26</td>
<td>26</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>12</td>
<td>29</td>
<td>38</td>
<td>38</td>
<td>29</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. \( B_m(n) \): Number of Bracelets: Distinct Combinations of \( m \) Objects Selected Out of \( n \) Cyclic Objects (\( m \leq n \)), where Reverse Cycles are Considered Equivalent
distinct selections of $m$ columns (variables) out of the $n$ cyclic columns of the matrix. As shown in Appendix A, the optimum dual values are equal. Thus for each selection, the value of the dual variables $y_1, \ldots, y_n$ is $1$ over the maximum sum (among all rows) of variables in the $m$ selected columns. Thus, the dual solution is specified by:

$$y_i = \begin{cases} 
1/L, & i \in SC \\
0, & i \notin SC
\end{cases}$$

(14)

$$W = \frac{1}{L} \sum_{i \in SC} r_i$$

(15)

$$L = \max_i (\sum_{j \in SC} a_{ij}^T),$$

(16)

where

$$a_{ij}^T = \text{elements of } A^T, \text{ where } i = 1, \ldots, n, j = 1, \ldots, n$$

$L = \text{max. number of non-zero coefficients in selected columns, } L \leq m$

$SC = \text{set of } m \text{ selected columns (variables)}$

3. Out of all cyclically-distinct dual solutions, the best ones are generally identified as having the highest ratio of $m/L$. Specifically, two sets of rules, whose proof is given in Appendix B, are used to identify the dominant solutions.

I. Solution $F$ is considered to dominate solution $G$ if:

$$SC_G \subset SC_F \quad \text{(implying } m_F > m_G)$$

(17a)

and

$$L_F = L_G$$

(17b)

II. Solutions $F$ and $G$ are considered to dominate solution $H$ if:

$$SC_H = SC_F + SC_G \quad \text{(implying } m_H = m_F + m_G)$$

(18a)

and

$$L_H = L_F + L_G$$

(18b)

4. Dominant dual solutions are rounded up to nearest integer values in order to develop a general expression for the value of $W$. To efficiently obtain an integer primal solution, the following constraint is added to the model. Alfares [26] shows that the addition of this constraint greatly improves computational performance.

$$\sum_{j=1}^{n} x_j \geq W$$

(19)

5. Applying primal-dual complementary slackness relations to dominant dual solutions, closed-form solutions to the primal problem can be derived, as demonstrated by Alfares [27]. In essence, a zero-slack dual constraint corresponds to a basic primal variable, and a basic dual variable corresponds to a zero-slack primal constraint.

### 7. CURRENT & FUTURE DEVELOPMENTS

The algorithm of Patent U.S. 8,046,316, titled “Cyclic Combinatorial Method and System” [10], has been described in this paper. This dual-based combinatorial cyclic selection algorithm can efficiently solve optimization problems with cyclic 0-1 matrices. The process of cyclic selection determines all cyclically distinct combinations of $m$ objects that can be selected out of $n$ cyclic objects. A cyclic permutation
procedure is developed to enumerate all cyclically distinct combinations of the \( m \) partitions of \( n \). This procedure combines the two basic combinatorial tools of partition and permutation. Using this procedure, all cyclically distinct dual solutions are enumerated. Simple rules are then used to identify the set of dominant dual solutions, out of which the optimum solution can be determined. A simple computer program can be used to implement the algorithm and to calculate the number of cyclic selections.

Example applications of Patent U.S. 8,046,316, [10] for employee days-off scheduling have been published by Alfares [26, 27]. Using primal-dual complementary slackness relations, the dominant dual solutions obtained can be used to generate complete primal optimum solutions, giving the optimum values of primal variables. From Table 1, optimal solution of these \( 7 \times 7 \)-matrix problems require only 18 cyclic combinations of dual variables instead of 128 \( (2^7) \) linear combinations. Problems of \( 12 \times 12 \)-matrix size would need only 224 cyclic combinations instead of 4,096 \( (2^{12}) \). These examples show the effectiveness of the patent’s cyclic algorithm in dramatically reducing the computational effort.

Applications of the patent’s algorithm to larger days-off problems are illustrated by Alfares [28, 29]. Many other applications of Patent U.S. 8,046,316, [10] are possible for similar optimization problems with cyclic 0-1 matrices. Examples show the effectiveness of the patent’s cyclic algorithm in dramatically reducing the computational effort.

8. CONFLICT OF INTEREST

The author confirms that this article content has no conflicts of interest.

9. ACKNOWLEDGEMENTS

The author gratefully acknowledges the generous support and research facilities provided by King Fahd University of Petroleum & Minerals (KFUPM).

APPENDIX A. PROOF OF EQUAL VALUES OF DUAL VARIABLES

Each cyclically distinct dual solution has \( m \) selected dual variables, excluding the slacks. Each set of \( m \) selected dual variables (columns of \( A_L \)) corresponds to a specific value of \( L \) (\( L \leq m \)), where \( L \) is the maximum number of non-zeros in the selected columns among the rows of \( A_L \). Each row (constraint) with \( L \) non-zero dual variables will be an equation. For each dual solution, define distinct equality constraints (DEC) as the set of distinct constraints in which the number of non-zeros is equal to \( L \). Let \( e = \text{number of distinct equality constraints (DEC)} \) for each dual solution (\( e \leq m \)). DEC can be represented as:

\[ A_L y = 1 \]  
(A.1)

where

\[ A_L = \text{exm submatrix of } A \], formed by the \( m \) columns of selected dual variables, and the \( e \) rows (equations) with \( L \) non-zero selected dual variables

Since each equation in DEC has \( L \) non-zero dual variables, the solution with equal values of dual variables \( (y_i = 1/L) \) is always feasible. Moreover, depending on the dimensions of \( A_L \), there are two possible cases.

Case 1. \( e = m \)

If the \( m \times m \) matrix \( A_L \) is non-singular, then:

(a) The system has a unique solution with equal dual values \( (y_i = 1/L) \)

(b) The solution corresponds to an extreme point

Case 2. \( e < m \)

The \( exm \) matrix \( A_L \) has more variables than equations, thus:

(a) The system has no unique solution

(b) It does not correspond to an extreme point

(c) Unequal values of non-zero dual variables are feasible, but cannot be optimal because they do not correspond to extreme points

APPENDIX B. PROOF OF DOMINANCE RULES (17) AND (18)

I. Assuming both (17a) and (17b) hold, then:

\[
W_F = \frac{1}{L_F} \sum_{i \in SC_F} r_i
= \frac{1}{L_G} \left( \sum_{i \in SC_G} r_i + \sum_{i \in (SC_F - SC_G)} r_i \right)
= W_G + \frac{1}{L_G} \sum_{i \in (SC_F - SC_G)} r_i
\]

Thus

\[
W_F \leq W_G
\]

II. Assuming both (18a) and (18b) hold, then:

\[
W_H = \frac{1}{L_H} \sum_{i \in SC_H} r_i
= \frac{1}{L_F + L_G} \left( \sum_{i \in SC_F} r_i + \sum_{i \in SC_G} r_i \right)
= \frac{L_F}{L_F + L_G} \left( \frac{1}{L_F} \sum_{i \in SC_F} r_i \right) + \frac{L_G}{L_F + L_G} \left( \frac{1}{L_G} \sum_{i \in SC_G} r_i \right)
= \frac{L_F}{L_F + L_G} (W_F) + \frac{L_G}{L_F + L_G} (W_G)
\]

Thus

\[
W_H \leq \max(W_F, W_G)
\]

REFERENCES


