mathematics

MDPI

Article

The Basic Algorithm for the Constrained Zero-One Quadratic Programming Problem with k-diagonal Matrix and Its Application in the Power System

Shenshen Gu *,† and Xinyi Chen

School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200444, China
* Correspondence: gushenshen@shu.edu.cn
† Current address: 99 Shangda Road, Shanghai 200444, China.

Received: 30 December 2019; Accepted: 16 January 2020; Published: 19 January 2020

Abstract: Zero-one quadratic programming is a classical combinatorial optimization problem that has many real-world applications. However, it is well known that zero-one quadratic programming is non-deterministic polynomial-hard (NP-hard) in general. On one hand, the exact solution algorithms that can guarantee the global optimum are very time consuming. And on the other hand, the heuristic algorithms that generate the solution quickly can only provide local optimum. Due to this reason, identifying polynomially solvable subclasses of zero-one quadratic programming problems and their corresponding algorithms is a promising way to not only compromise these two sides but also offer theoretical insight into the complicated nature of the problem. By combining the basic algorithm and dynamic programming method, we propose an effective algorithm in this paper to solve the general linearly constrained zero-one quadratic programming problem with a k-diagonal matrix. In our algorithm, the value of k is changeable that covers different subclasses of the problem. The theoretical analysis and experimental results reveal that our proposed algorithm is reasonably effective and efficient. In addition, the placement of the phasor measurement units problem in the power system is adopted as an example to illustrate the potential real-world applications of this algorithm.

Keywords: zero-one quadratic problem; combinatorial optimization; k-diagonal matrix; power system

1. Introduction

Optimization problems normally fall into two categories, one for continuous variables and the other for discrete variables. The latter one is called combinatorial optimization, which is an active field in applied mathematics [1]. Common problems are the maximum flow problem, traveling salesman problem, matching problem, knapsack problem, etc. Among those famous combinatorial optimizations, zero-one quadratic programming (01QP), whose variables can only be either 0 or 1 [2], is very important and attracts a lot of attention.

Zero-one quadratic programming, which can be divided into constrained (01CQP) and unconstrained (01UQP), is a combinatorial optimization and has practical significance. For example, it can be applied in circuit design [3], pattern recognition [4], capital budgeting [5], portfolio optimization [6], etc. Except for these well known applications, 01QP also has potential applications in the nonlinear control related fields [7–11]. Among these applications, the phasor measurement unit (PMU) placement has been widely studied. Phasor measurement units can be used in dynamic monitoring, system protection, and system analysis and prediction. Therefore, the placement of PMU has become an important issue. As the scale of the electric grid grows, the PMU placement problem becomes more difficult and must be addressed considering certain requirements. In [12],

a modified bisecting search combined with simulated annealing method was proposed, and the latter randomly selected arrays were used to test the observability of the system. Considering the incomplete observability of the system, a calculation method based on graph theory was proposed in [13]. This method is time-consuming, and with the increase of dimensions, the calculation load is too heavy. An integer linear programming method [14] and an improved algorithm [15] were put forward, considering system redundancy, and full and incomplete observability. Researchers in [16] proposed a binary programming method considering the joint placement of the conventional measurement units and phasor measurement units.

Zero-one quadratic programming also has some theoretical significance. Many classical problems, such as the max-cut problem [17,18] and max-bisection problem [19] can be converted to zero-one quadratic programming. Therefore, designing the algorithm that can solve 01QP effectively and efficiently is very meaningful not only in practical fields but also in theoretical fields.

However, zero-one quadratic programming is a well known NP-hard problem in general. The common solutions are exact solution and heuristic algorithms. Exact solution algorithms can guarantee the global optimum. In [20], the branch-and-bound method was used to solve zero-one quadratic programming problems without constraints. In [21,22], some exact solutions were proposed by means of geometric properties. Penalty parameters were introduced to solve zero-one quadratic programming problems with constraints in [23]. But exact solution algorithm are very time consuming and suitable for small scale problems only. Oppositely, heuristic algorithms, such as simulated annealing [24], genetic algorithms [25], neural networks [26], and ant colony algorithms [27], can solve the medium and large scale problems quickly in general. But most of them can only find the local optimum.

Therefore, identifying polynomially solvable subclasses of zero-one quadratic programming problems and their corresponding algorithms is a promising way to not only compromise these two sides but also offer theoretical insight into the complicated nature of the problem. In our past studies [28,29], the problem of five-diagonal matrix quadratic programming with linear constraints has been solved effectively. In [30], an algorithm for solving 01QP with a seven-diagonal matrix Q was presented. However, for these algorithms, the applicable problem is very specific. This narrows their applications. Then, we proposed an algorithm for the unconstrained problems with a k-diagonal matrix in [31]. In this paper, based on our previous results, we further propose an algorithm for the general linearly constrained zero-one quadratic programming problem with a k-diagonal matrix by combining the basic algorithm and dynamic programming method. For the algorithm we proposed, the value of k is changeable. That means the algorithm can cover different subclasses of the problem. The theoretical analysis and experimental results reveal that our proposed algorithm is reasonably effective and efficient. We also apply the algorithm to real-world applications and then verify its feasibility.

The main contributions of this paper are reflected in the following aspects: (1) The previous algorithm targeted a fixed k value in Q matrix. While the algorithm in this paper targeted a general problem with changeable k values. (2) We analyze the time complexity and give the proof process of the rationality of the algorithm. (3) We apply the algorithm to the phasor measurement units placement in real-world application.

This paper is organized as follows: in Section 2, we review the algorithm of solving unconstrained zero-one k-diagonal matrix quadratic programming. In Section 3, a constrained zero-one k-diagonal matrix quadratic programming algorithm with the proof of the algorithm is proposed. Application of zero-one quadratic programming in the phasor measurement units placement is put forward in Section 4. Experimental results and discussion are given in Section 5. We draw our conclusions and put forward the prospects in Section 6.

2. Basic Algorithm to 01UQP

The following Equation (1) shows the form of the k-diagonal matrix zero-one quadratic programming problem. The special point is the form of the matrix Q called the k-diagonal matrix, where $k = 2m + 1$ $(m = 0, 1, 2, \ldots, n-1)$.

$$\min_{x \in \{0,1\}^n} f(x) = \frac{1}{2} x^T Q x + c^T x \tag{1}$$

where $Q = (q_{ij})_{n \times n}$, $q_{ij} = q_{ji} (i, j = 1, 2, \ldots, n)$ indicates that it is a symmetric matrix. Note that all the numbers in this matrix are zero except q_{ij} and $q_{ji} (i = 1, 2, \ldots, n-1, j = i+1, i+2, \ldots, i+m)$.

$$\begin{pmatrix} 0 & q_{1,2} & q_{1,3} & \cdots & 0 & 0 & 0 \\ q_{2,1} & 0 & q_{2,3} & \cdots & 0 & 0 & 0 \\ q_{3,1} & q_{3,2} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & q_{n-2,n-1} & q_{n-2,n} \\ 0 & 0 & 0 & \cdots & q_{n-1,n-2} & 0 & q_{n-1,n} \\ 0 & 0 & 0 & \cdots & q_{n,n-2} & q_{n,n-1} & 0 \end{pmatrix}$$

Based on past works [30,31], we can have the algorithm, as follows, to solve 01UQP. The feasibility and effect of the algorithm can be seen in [31]. Figure 1 shows the Algorithm 1 process intuitively.

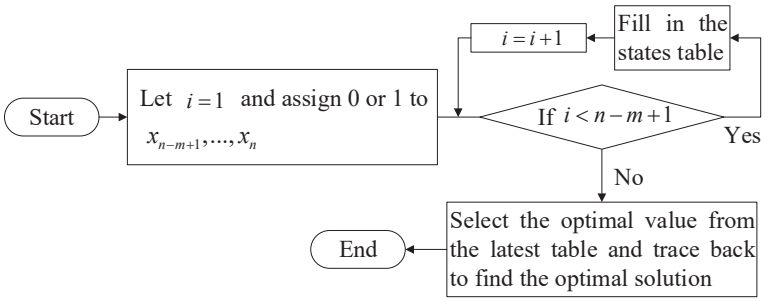

Figure 1. Flow chart of the zero-one unconstrained quadratic programming (01UQP) algorithm.

Algorithm 1 Process of solving 01UQP

Step 1: Assign $x_{n-m+1}, \ldots, x_n$ to 0 or 1.

(1) Adjacent terms are the combination of $x_{n-m+1}, \ldots, x_n$, whose value is the same except for the value of x_n.
(2) Get the corresponding $f(x)$.
(3) Label these states $state0, state1, \ldots, state(2^m - 1)$.
(4) Compare $f(x)$ in the two adjacent terms. (only the coefficient of x_{n-m} and the constant term are different.)

Step 2: Change the value of x_{n-m-i} to 0 and $1 (i = 0, 1, \ldots, n-m-1)$.

(1) Compare every two adjacent states and take the result with the smaller constant term as the new state.
(2) Update all the 2^m states.

Step 3: Get the optimal solution x.

(1) Update the states based on **Step 2** until only the constant term is in $f(x)$.
(2) The optimal value is the minimal one.
(3) Trace back and get the optimal solution x.

3. Basic Algorithm to 01CQP

3.1. 01CQP Algorithm Description

Consider the constrained k-diagonal matrix zero-one quadratic programming problem:

$$\begin{array}{ll} \min\limits_{x\in\{0,1\}^n} & \frac{1}{2}x^TQx + c^Tx \\ \text{s.t.} & a^Tx \leq b \end{array} \tag{2}$$

where Q has the same meaning as that of the 01UQP formula in Section 2, $a \in \mathbb{Z}_+^n$, $b \in \mathbb{Z}_+$.

In this section, we utilize the dynamic programming method to solve the 01CQP problem. To apply the dynamic programming method, we introduce a state variable $s_k(s_k \in \mathbb{Z})$ and a stage variable $k(0 < k \leq n)$, which should satisfy the following iteration. s_{k+1} can be expressed as:

$$s_{k+1} = s_k + a_{k+1}x_{k+1} \quad (k = 1, \ldots, n-1).$$

We only need to consider the integer point of the state space since $a \in \mathbb{Z}_+^n$ and $b \in \mathbb{Z}_+$. Since s_k satisfies $0 \leq s_k \leq b$, we define a set $s_k = \{s_k | 0 \leq s_k \leq b, s_k \in \mathbb{Z}\}$.

Algorithms 2 and 3 show the detailed calculation process.

Algorithm 2 Calculation Method of $f(s_k)$

Case 1: When $k = 1$, there are two cases.

(1a) $s_1 < a_1$
$x_1 = 0$, $x_1^* = 0$, $f(s_1) = f(0, \ldots, x_n)$
(1b) $s_1 \geq a_1$
$x_1 = 1$, $x_1^* = 1$, $f(s_1) = f(1, \ldots, x_n)$

We will get a series of functions $f(x)$ after executing Case 1.

Case 2: When $k \geq 2$, there are also two cases.

(2a) $s_k < a_k$
x_k must be 0 to satisfy $s_{k-1} = s_k - a_kx_k$. At this time, $s_k = s_{k-1}$, by which we can obtain the function $f(s_k) = f(s_{k-1})|_{x_k=0}$.
(2b) $s_k \geq a_k$
In this case, x_k can be both 0 or 1, which generates two more situations:
1) If ${x_k}^* = 0$ and $s_k = s_{k-1}$, we can get $f(s_k) = f(s_{k-1})|_{x_k=0}$.
2) If ${x_k}^* = 1$ and $s_{k-1} = s_k - a_kx_k$, we can get $f(s_k) = f(s_{k-1})|_{x_k=1}$.

We can see that there is only one function $f(s_k)$ corresponding to each state s_k when $k = 1$, and there are several $f(s_k)$ to each state s_k when $k > 1$. To save storage and computational time, $f(s_k)$ should be selected satisfactorily in the next step. We need to find the optimal $f(s_k) = f(s_{k-1})|_{x_k=0}$ and $f(s_k) = f(s_{k-1})|_{x_k=1}$, that the optimizing process refers to for Algorithm 3.

Algorithm 3 $state_0$ and $state_1$

$state_0(x_k = 0)$

Case 1: There is only one $f(s_k) = f(s_{k-1})|_{x_k=0}$:
Set ${x_k}^* = 0$, $f(s_k) = f(s_{k-1})|_{x_k=0}$
Case 2: There are more than one $f(s_k) = f(s_{k-1})|_{x_k=0}$:
1) Compare $f(s_{k-1})|_{x_k=0}$, which are almost the same except for the constant term and pick up the smallest one.
2) Set ${x_k}^* = 0$ and $f(s_k) = f(s_{k-1})|_{x_k=0}$.

$state_1(x_k = 1)$

Case 1: There is only one $f(s_k) = f(s_{k-1})|_{x_k=1}$:
Set ${x_k}^* = 1$, $f(s_k) = f(s_{k-1})|_{x_k=1}$
Case 2: There are more than one $f(s_k) = f(s_{k-1})|_{x_k=1}$:
1) Find the $f(x)$ using a similar approach to that in $state_0$ Case 2.
2) Set ${x_k}^* = 1$ and $f(s_k) = f(s_{k-1})|_{x_k=1}$.

According to the above algorithm, the maximum number of functions per *State* is shown in Table 1. The primary time is spent generating the state table. The number of times that the core steps calculated is focused on the state number in Table 1. In total, we need to update the state table n times and the number of state is b, so the time complexity is $O(2^{m-1} \times n \times b)$.

Table 1. Maximum number of functions per *State*.

Situation	The Maximum Number of Functions
Algorithm 2 Case 1	1
Algorithm 3 Case 1	2^{m-1}
Algorithm 3 Case 2	2^{m-1}

3.2. Analysis on the Effectiveness and Rationality of 01CQP Algorithm

In this section, we analyze the properties of the polynomial time algorithm for 01CQP. To analyze the algorithm, we need to demonstrate the rationality of Algorithms 2 and 3. Suppose $f(x)$ has the form:

$$\begin{aligned} f(x) = {} & q_{12}x_1x_2 + q_{13}x_1x_3 + \cdots + q_{1,1+m}x_1x_{1+m} + \cdots + q_{n-m,n-m+1}x_{n-m} \\ & x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n + \cdots + q_{n-1,n}x_{n-1}x_n + c_1x_1 + \cdots + c_nx_n. \end{aligned}$$

Step 1: Set $k = 1$
(1) $s_1 < a_1, x_1 = 0$

$$\begin{aligned} f_1(s_1) = f(x)|_{x_1=0} = {} & q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + q_{i,i+m}x_ix_{i+m} \\ & + \cdots + q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n \\ & + \cdots + q_{n-1,n}x_{n-1}x_n + c_2x_2 + \cdots + c_nx_n (i = 2, \ldots, n-m-1). \end{aligned} \tag{3}$$

(2) $s_1 \geq a_1, x_1 = 1$

$$\begin{aligned} f_1(s_1) = f(x)|_{x_1=1} = {} & q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + q_{i,i+m}x_ix_{i+m} \\ & + \cdots + q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n \\ & + \cdots + q_{n-1,n}x_{n-1}x_n + \widehat{c_2}x_2 + \cdots + \widehat{c_{1+m}}x_{1+m} + c_{m+2}x_{m+2} \\ & + \cdots + c_nx_n + c_1. \end{aligned} \tag{4}$$

Clearly, when $k = 1$, each state s_1 has only one function $f_1(s_1)$. The coefficient of $x_2, \ldots, x_{m+1}$ and the constants are the only different terms in the function.

Step 2: Set $k = 2, 3, ..., m$

(1) If $s_k < a_k$, execute $state_0$.

If $s_{k-1} < a_{k-1}$, $f_k(s_k)$ has the same form as function (3):

$$\begin{aligned} f(s_k) &= q_{k,k+1}x_kx_{k+1} + q_{k,k+2}x_kx_{k+2} + \cdots + q_{k,k+m}x_kx_{k+m} \\ &+ \cdots + q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + q_{i,i+m}x_ix_{i+m} + \cdots + \\ &q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n + \cdots + \\ &q_{n-1,n}x_{n-1}x_n + c_{k+1}x_{k+1} + \cdots + c_nx_n + const. \end{aligned} \tag{5}$$

($const$ represents the constant term of the $f(s_{k-1})$.)

If $s_k \geq a_{k-1}$, $f_k(s_k)$ has the same form as function (4):

$$\begin{aligned} f(s_{k-1}) &= q_{k,k+1}x_kx_{k+1} + q_{k,k+2}x_kx_{k+2} + \cdots + q_{k,k+m}x_kx_{k+m} \\ &+ \cdots + q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + q_{i,i+m}x_ix_{i+m} + \cdots + \\ &q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n + \cdots + \\ &q_{n-1,n}x_{n-1}x_n + \widehat{c_{k+1}}x_{k+1} + \cdots + \widehat{c_{k+m}}x_{k+m} + c_{k+m}x_{k+m} \\ &+ \ldots + c_nx_n + const + c_k. \end{aligned} \tag{6}$$

Then, $f_k(s_k)$ can be shown as:

$$\begin{aligned} f_k(s_k) &= f_{k-1}(s_{k-1})|_{x_k=0} = q_{k+1,k+2}x_{k+1}x_{k+2} + q_{k+1,k+3}x_{k+1}x_{k+3} + \cdots \\ &+q_{k+1,k+m+1}x_{k+1}x_{k+m+1} + \cdots + q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + \\ &q_{i,i+m}x_ix_{i+m} + \cdots + q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n \\ &+ \cdots + q_{n-1,n}x_{n-1}x_n + \widehat{c_{k+1}}x_{k+1} + \cdots + \widehat{c_{k+m}}x_{k+m} + c_{k+m+1}x_{k+m+1} \\ &+ \ldots + c_nx_n + \widehat{const}. \end{aligned} \tag{7}$$

At this point, the maximum number of $f_k(s_k)$ corresponding to each s_k is 2^{k-2}.

(2) If $s_k \geq a_k$, execute $state_0$ and $state_1$.

Here, the $state_0$ is the same as the one in Case 1, so we do not need to repeat it.

Consider $state_1$, $x_k = 1$, $f_k(s_k) = f_{k-1}(s_k - a_kx_k)$, $s_{k-1} = s_k - a_kx_k = s_k - a_k$.

If $s_{k-1} < a_{k-1}$, $f(s_{k-1})$ has the form of function (5). If $s_{k-1} \geq a_{k-1}$, $f(s_{k-1})$ has the form of function (6), then, $f_k(s_k)$ can be expressed as:

$$\begin{aligned} f_k(s_k) &= f_{k-1}(s_{k-1})|_{x_k=1} = q_{k+1,k+2}x_{k+1}x_{k+2} + q_{k+1,k+3}x_{k+1}x_{k+3} + \cdots \\ &+q_{k+1,k+m+1}x_{k+1}x_{k+m+1} + \cdots + q_{i,i+1}x_ix_{i+1} + q_{i,i+2}x_ix_{i+2} + \cdots + \\ &q_{i,i+m}x_ix_{i+m} + \cdots + q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + q_{n-m,n}x_{n-m}x_n \\ &+ \cdots + q_{n-1,n}x_{n-1}x_n + \widehat{c_{k+1}}'x_{k+1} + \cdots + \widehat{c_{k+m}}'x_{k+m} + c_{k+m+1}x_{k+m+1} \\ &+ \ldots + c_nx_n + \widehat{const}'. \end{aligned} \tag{8}$$

All $f_k(s_k)$ are only different in the constant terms and the coefficient of $x_{k+1}, ..., x_{m+2}$.

Step 3: $k = n - m - 1$

Consider the most complex case, $s_k \geq a_k$ and the state variable s_k corresponds to 2^{m-1} $f_k(s_k)$ and 2^{m-1} $f_k(s_k) = f_{k-1}(s_k - a_k)$ respectively (2^{m-1} is the number of both). Therefore, we need to execute $state_0$ first, and then execute $state_1$.

(1) $state_0$:

Since $f_k(s_k)$ has the form as function (8), the $state_0$ is executed for the $f_k(s_k)$ respectively, and the following expression is obtained:

$$\begin{aligned} f_k(s_k) &= f_{k-1}(s_k)|_{x_k=0} = q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + \\ &q_{n-m,n}x_{n-m}x_n + \cdots + q_{n-1,n}x_{n-1}x_n + \widehat{c_{n-m}}'x_{n-m} + c_{n-m+1}x_{n-m+1} \\ &+ \ldots + c_nx_n + \widehat{const}'. \end{aligned} \tag{9}$$

Clearly, these $f_k(s_k)$ are only different in the constants and the coefficients of x_{n-m}.

(2) $state_1$:

It is possible to obtain $f_k(s_k)$, which has the following form:

$$\begin{aligned} & f_k(s_k) = f_{k-1}(s_{k-1})|_{x_k=1} = q_{n-m,n-m+1}x_{n-m}x_{n-m+1} + \cdots + \\ & q_{n-m,n}x_{n-m}x_n + \cdots + q_{n-1,n}x_{n-1}x_n + \widehat{c_{n-m}}'' x_{n-m} + c_{n-m+1}x_{n-m+1} \\ & + \ldots + c_n x_n + \widehat{const}' + \widehat{c_{n-m-1}}. \end{aligned} \tag{10}$$

For both case (1) and (2), they are only different in the constants and the coefficients of x_{k+1}.

Step 4: $k = m+2, ..., n$

Based on Algorithm 2 Case 2, we calculate the function $f_k(s_k)$, and the core is the implementation of $state_0$ and $state_1$. When executing $state_0$ and $state_1$, the difference between the two $f_k(s_k)$ is the constant and the coefficients of x_k. Therefore, when executing $state_0$, we only need to compare the constant terms. When executing $state_1$, we only need to compare the sum of the constants and the coefficients of x_k. In this way, we can avoid solving the excess $f_k(s_k)$.

The form of the function $f_k(s_k)$ and the maximum number of $f_k(s_k)$ in each stage k are similar to the one in stage $k = n - m$, which also ensures the repeatability of the algorithm. The algorithm is supplemented by concrete examples and numerical simulations.

3.3. Calculation Example of 01CQP

For example, the parameters Q, c, a, and b are:

$$Q = \begin{pmatrix} 0 & 23 & -37 & -56 & 0 & 0 & 0 & 0 \\ 23 & 0 & 41 & 16 & -34 & 0 & 0 & 0 \\ -37 & 41 & 0 & -62 & -27 & 76 & 0 & 0 \\ -56 & 16 & -62 & 0 & -81 & 14 & -58 & 0 \\ 0 & -34 & -27 & 81 & 0 & 90 & 25 & -42 \\ 0 & 0 & 76 & 14 & 90 & 0 & -12 & 31 \\ 0 & 0 & 0 & -58 & 25 & -12 & 0 & -94 \\ 0 & 0 & 0 & 0 & -42 & 31 & -94 & 0 \end{pmatrix}$$

$$c = \begin{pmatrix} 24, & -54, & -17, & 36, & -72, & 63, & 46, & -18 \end{pmatrix}^T$$

$$a = \begin{pmatrix} 1, & 2, & 3, & 2, & 4, & 2, & 3, & 2 \end{pmatrix}^T$$

$$b = 6.$$

In this example, we will omit some of the states.

(1) Set $k = 1$, $s_1 = a_1x_1$,$a_1 = 1$

We can set $s_1 = 0, 1, \ldots, 6$. According to Algorithm 2 Case 1, we can obtain Table 2.

When $s_1 = 0$, we can calculate ${x_1}^*$ and $f(s_1)$ through Algorithm 2 Case (1a) for $s_1 < a_1$.

When $s_1 = 1, \ldots, 6$, we can calculate ${x_1}^*$ and $f(s_1)$ through Algorithm 2 Case (1b) for $s_1 \geq a_1$.

We omit $state3, \ldots, state6$, which is the same as $state1, state2$.

(2) Set $k = 2, 3$, $s_2 = a_1x_1 + a_2x_2 = s_1 + a_2x_2$, $s_3 = s_2 + a_3x_3$

We can set $s_3 = 0, 1, \ldots, 6$. According to Algorithm 2 Case 2, we obtain the Table 3. The case of $s_2 = 0, 1, \ldots, 6$ is omitted.

(3) Set $k = 4, 5, 6, 7$

According to Algorithm 2 Case 2 and Algorithm 3, we can gain $f_k(s_k)$ with $k = 3, ..., 7$. Tables 4 and 5 show part of the calculation process. From the table, we can see that only the coefficient of s_{k+1} and constant terms are different in the adjacent items.

(4) Set $k = 8$

Finally, we have Table 6, which only has the constant terms. Through it, we can see that ${x_8}^* = 0$, $f(x^*) = -160$, and by the backtracking method, we find the optimum solution $x^* = (0, 1, 0, 0, 1, 0, 0, 0)$.

Table 2. Functions $f_1(s_1)$.

s_1	x_1	$f_1(s_1)(a_1 = 1)$
0	0	$41x_2x_3 + 16x_2x_4 - 34x_2x_5 - 62x_3x_4 - 27x_3x_5 + 76x_3x_6 - 81x_4x_5$ $+14x_4x_6 - 58x_4x_7 + 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 54x_2 - 17x_3 + 36x_4 - 72x_5 + 63x_6 + 46x_7 - 18x_8$
1	1	$41x_2x_3 + 16x_2x_4 - 34x_2x_5 - 62x_3x_4 - 27x_3x_5 + 76x_3x_6 - 81x_4x_5$ $+14x_4x_6 - 58x_4x_7 + 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 31x_2 - 54x_3 - 20x_4 - 72x_5 + 63x_6 + 46x_7 - 18x_8 + 24$
2	1	$41x_2x_3 + 16x_2x_4 - 34x_2x_5 - 62x_3x_4 - 27x_3x_5 + 76x_3x_6 - 81x_4x_5$ $+14x_4x_6 - 58x_4x_7 + 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 31x_2 - 54x_3 - 20x_4 - 72x_5 + 63x_6 + 46x_7 - 18x_8 + 24$

Table 3. Functions $f_3(s_3)$.

s_3	x_3	$f_3(s_3)(a_3 = 3)$
0	0	$f_3(s_3) = f_2(0)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 + 36x_4 - 72x_5 +$ $63x_6 + 46x_7 - 18x_8$
1	0	$f_3(s_3) = f_2(1)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 20x_4 - 72x_5 +$ $63x_6 + 46x_7 - 18x_8 + 24$
3	0	$f_3(s_3) = f_2(3)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 20x_4 - 72x_5 +$ $63x_6 + 46x_7 - 18x_8 + 24$
3	0	$f_3(s_3) = f_2(3)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 4x_4 - 106x_5 +$ $63x_6 + 46x_7 - 18x_8 - 7$
3	1	$f_3(s_3) = f_2(3)_{\mid x_3=1} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 26x_4 - 99x_5 +$ $139x_6 + 46x_7 - 18x_8 - 17$
5	0	$f_3(s_3) = f_2(5)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 20x_4 - 72x_5 +$ $63x_6 + 46x_7 - 18x_8 + 24$
5	0	$f_3(s_3) = f_2(5)_{\mid x_3=0} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 4x_4 - 106x_5$ $+63x_6 + 46x_7 - 18x_8 - 7$
5	1	$f_3(s_3) = f_2(2)_{\mid x_3=1} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 84x_4 - 99x_5 +$ $139x_6 + 46x_7 - 18x_8 - 30$
5	1	$f_3(s_3) = f_2(2)_{\mid x_3=1} = -81x_4x_5 + 14x_4x_6 - 58x_4x_7 + 90x_5x_6 +$ $25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8 - 94x_7x_8 - 10x_4 - 133x_5 +$ $139x_6 + 46x_7 - 18x_8 - 30$

Table 4. Functions $f_4(s_4)$.

s_4	x_4	$f_4(s_4)(a_4 = 2)$
1	0	$f_4(s_4) = f_3(1)_{\mid x_4=0} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 72x_5 + 63x_6 + 46x_7 - 18x_8 + 24$
5	0	$f_4(s_4) = f_3(5)_{\mid x_4=0} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 72x_5 + 63x_6 + 46x_7 - 18x_8 + 24$
5	0	$f_4(s_4) = f_3(5)_{\mid x_4=0} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 106x_5 + 63x_6 + 46x_7 - 18x_8 - 7$
5	0	$f_4(s_4) = f_3(5)_{\mid x_4=0} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 99x_5 + 139x_6 + 46x_7 - 18x_8 - 30$
5	0	$f_4(s_4) = f_3(5)_{\mid x_4=0} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 133x_5 + 139x_6 + 46x_7 - 18x_8 - 30$
5	1	$f_4(s_4) = f_3(3)_{\mid x_4=1} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 153x_5 + 77x_6 - 12x_7 - 18x_8 + 4$
5	1	$f_4(s_4) = f_3(3)_{\mid x_4=1} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 187x_5 + 77x_6 - 12x_7 - 18x_8 - 11$
5	1	$f_4(s_4) = f_3(3)_{\mid x_4=1} = 90x_5x_6 + 25x_5x_7 - 42x_5x_8 - 12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 180x_5 + 153x_6 - 12x_7 - 18x_8 - 43$

Table 5. Functions $f_5(s_5)$.

s_5	x_5	$f_5(s_5)(a_5 = 4)$
5	0	$f_5(s_5) = f_4(5)_{\mid x_5=0} = -12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 72x_6 + 46x_7 - 18x_8 - 7$
5	0	$f_5(s_5) = f_4(5)_{\mid x_5=0} = -12x_6x_7 + 31x_6x_8$ $-94x_7x_8 - 72x_6 + 46x_7 - 18x_8 - 30$
5	0	$f_5(s_5) = f_4(5)_{\mid x_5=0} = -12x_6x_7 + 31x_6x_8$ $-94x_7x_8 + 77x_6 - 12x_7 - 18x_8 - 11$
5	0	$f_5(s_5) = f_4(5)_{\mid x_5=0} = -12x_6x_7 + 31x_6x_8$ $-94x_7x_8 + 153x_6 - 12x_7 - 18x_8 - 43$
5	1	$f_5(s_5) = f_4(1)_{\mid x_5=1} = -12x_6x_7 + 31x_6x_8$ $-94x_7x_8 + 153x_6 + 71x_7 - 60x_8 - 48$

Table 6. Functions $f_8(s_8)$.

s_8	x_8	$f_8(s_8)$
0	0	0
1	0	24
2	0	−54
2	1	−18
3	0	−17
3	1	6
4	0	−72
4	1	−72
5	0	−48
5	1	−66
6	0	−160
6	1	−132

4. Application of Zero-One Quadratic Programming in Phasor Measurement Units Placement

How to find the installation location and the number of installations is the focus of the phasor measurement units placement problem. The matrix in the PMU placement problem differs

from the previous k-diagonal matrix in the elements of the main diagonal. However, since x is either 0 or 1, we can convert it into a k-diagonal matrix. Then, we can take advantage of the algorithms designed above to work out PMU placement without considering too many constraints and realistic requirements.

4.1. Modelling

For the power system composed of n nodes, the placement of PMU is represented by n-dimensional vector $X = (x_1, x_2, \ldots, x_n)$. Here, $i = 1, 2, \ldots, n$,

$$x_i = \begin{cases} 0 & \text{a PMU is installed at bus } i \\ 1 & \text{otherwise.} \end{cases}$$

The matrix H represents the network graph structure,

$$h_{ij} = \begin{cases} 1 & i = j \\ 1 & i \text{ is connected to } j \\ 0 & \text{otherwise,} \end{cases}$$

and the objective function is

$$V(x) = \lambda(N - HX)^T R(N - HX) + X^T QX \tag{11}$$

where λ is the weight. $N \in \mathbb{R}^n$ represents the upper bound of the maximum redundancy of each bus. $R \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{n\times n}$ are diagonal arrays, representing the importance of each bus and the cost of placing the PMU on the bus respectively.

We expand Equation (11),

$$\begin{aligned} V(x) &= \lambda(N - HX)^T R(N - HX) + X^T QX \\ &= \lambda[N^T RN - N^T RHX - X^T H^T RN + X^T H^T RHX] + X^T QX \\ &= \lambda N^T RN - 2\lambda N^T RHX + \lambda X^T H^T RHX + X^T QX \\ &= \tfrac{1}{2} X^T (2\lambda H^T RH + 2Q)X + (-2\lambda N^T RH)X + \lambda N^T RN. \end{aligned} \tag{12}$$

Then, Equation (12) can be expressed as integer quadratic programming as,

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^T Gx + f^T x \\ \text{s.t.} \quad & M(x) = 0 \\ & x_i \in \{0, 1\}^n \end{aligned} \tag{13}$$

where $G = 2\lambda H^T RH + 2Q$, $f = (-2\lambda N^T RH)^T$. $M(x)$ is the column vector consisting of $m_i(x) = (1 - x_i)\prod_{j\in A_i}(1 - x_j)(i \in \Omega)$. A_i and Ω represent a set of nodes adjacent to the bus i and the bus set respectively. The above constraints require at least one PMU unit to be placed between the bus and its adjacent nodes to ensure that each adjacent node of the bus can be observed. The above problem can be expressed as unconstrained problems by the weighted least square method:

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^T Gx + f^T x + M(x)^T VM(x) \\ & = \tfrac{1}{2} x^T Gx + f^T x + \textstyle\sum_{i=1}^n (v_i(1 - x_i)^2 (\prod_{j\in A_i}(1 - x_j))^2) \\ \text{s.t.} \quad & x_i \in \{0, 1\}^n \end{aligned} \tag{14}$$

where $V = diag(v_i)$.

4.2. Example and Experimental Result

Suppose the coefficient matrix H of a power system is

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Set Q as the unit matrix, $\lambda = 0.5$,

$$R = \begin{pmatrix} 12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 128 & 0 & 0 & 0 & 0 \\ 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 140 & 0 & 0 \\ 0 & 0 & 0 & 0 & 72 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{pmatrix}, N = \begin{pmatrix} 2, & 2, & 2, & 3, & 2, & 1 \end{pmatrix}^T.$$

Then, we can have

$$G = \begin{pmatrix} 192 & 140 & 62 & 178 & 0 & 0 \\ 140 & 282 & 152 & 268 & 140 & 0 \\ 62 & 152 & 204 & 190 & 140 & 0 \\ 178 & 268 & 190 & 392 & 212 & 72 \\ 0 & 140 & 140 & 212 & 224 & 82 \\ 0 & 0 & 0 & 72 & 82 & 84 \end{pmatrix},$$

$$f = \begin{pmatrix} -380, & -700, & -544, & -920, & -574, & -154 \end{pmatrix}^T.$$

The main diagonal of the matrix for this problem is 1, while the definition of k-diagonal matrix Q requires the main diagonal elements to be zero. Since $x \in \{0,1\}^n$, the diagonal of G can be converted into a linear term and can be considered as a seven-diagonal matrix. This indicates that the diagonal of G becomes zero and f is updated as seen in Equation (15). Using the algorithm in Section 2, we can find the optimal solution $(0,1,1,1,0,1)$ without considering the constraints. This is to install four PMUs in bus 2, 3, 4, and 6. Through observation, it can be seen that the configuration results meet the system observability and the constraint is reached.

$$f = \begin{pmatrix} -284, & -559, & -442, & -724, & -462, & -112 \end{pmatrix}^T. \tag{15}$$

The above problem considers the placement of PMU under the condition that the system is completely observable. When considering the PMU placement problem with the $N-1$ principle, constraint conditions $2x_i + 2\sum_{j \in P_i^2} x_j + \sum_{j \in P_i^1} x_j \geq 2 (i \in N)$ are added based on the definition of a single node observable. N is a set that includes the nodes required to have objectivity when a fault occurs in the system. P_i^2 and P_i^1 represents a node set connected to node i with two and one line respectively.

If the bus between nodes 5 and 6 breaks down, we set $H(5,6)$ and $H(6,5)$ as zero. Then, the constraint is added.

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} x \geq \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We can get the optimal solution is the same as that in the former example. As the connection between nodes 5 and 6 fails, the system after placing PMU will not lose its observability. We only

consider the algorithm to solve the constrained k-diagonal zero-one matrix quadratic programming and consider that it can be applied to some form of PMU placement problems. Some of the conditions to the placement of PMU have not been taken into account here, which has yet to be studied by specialized engineers.

5. 01CQP Algorithm Simulation and Discussion

5.1. Algorithm Experimental Simulation

Now, the experimental results are provided for this algorithm to illustrate its performance. We implement the algorithm with C++ and run it on an Intel(R) Core(TM) i7-8550U CPU. For the problems we tested, the dimension of matrix Q ranges from 10 to 100 and k takes $5, 7, \ldots, 25$. All simulation data (Q, c, a, b) are generated randomly. Q, c is set up as in Section 2 with numbers ranging from -100 to 100, and a, b range between 0 and 20. Then, we obtain Table 7, which shows the detailed computation time for different dimensional problems with different diagonal numbers.

Table 7. Corresponding computation time in different diagonal numbers with dimension n = 10 to 100 of 01CQP.

n	10	20	30	40	50	60	70	80	90	100
$k = 5$	0.0356	0.0678	0.1404	0.2039	0.3289	0.4102	0.4589	0.6942	0.7391	1.0060
$k = 7$	0.0336	0.0703	0.1419	0.2030	0.3097	0.4024	0.4839	0.6908	0.7475	1.0519
$k = 9$	0.0320	0.0638	0.1424	0.1851	0.3129	0.3640	0.4896	0.6831	0.7835	0.9771
$k = 11$	0.0335	0.0619	0.1470	0.2025	0.2749	0.3357	0.4155	0.5482	0.6621	0.8812
$k = 13$	0.0246	0.0482	0.1093	0.1577	0.2591	0.3500	0.4176	0.6029	0.7109	0.8872
$k = 15$	0.0185	0.0544	0.1138	0.1734	0.2561	0.3624	0.4469	0.5782	0.7356	0.8873
$k = 17$	0.0234	0.0549	0.1185	0.1749	0.2838	0.3766	0.4732	0.7220	0.8908	1.1064
$k = 19$	0.0397	0.0744	0.1692	0.2210	0.3944	0.4888	0.6059	0.9147	0.9886	1.2304
$k = 21$	$--$	0.0872	0.1797	0.2836	0.4656	0.5714	0.6995	1.1097	1.2544	1.7283
$k = 23$	$--$	0.1283	0.2640	0.4409	0.5499	0.9615	1.0942	1.3158	1.8381	1.8951
$k = 25$	$--$	0.1961	0.3673	0.6169	0.8295	1.3556	1.3235	2.0418	2.3717	4.2689

5.2. Experimental Results Discussion

Here, based on the experimental results, we investigate the influence of n and k on the algorithm and discuss the importance and implications of our study results. Firstly, we fix the values of k but increase n. Figure 2 shows the experimental situations when the matrix dimension n changes from 10 to 100 with k is fixed at 9, 13, 19 and 23 respectively. As can be seen, the calculation time increases slightly with the increase of dimension. Secondly, we fix the values of n but increase k. Figure 3 shows the experimental situations when k changes from 5 to 25 with n is fixed at 20, 40, 60 and 80 respectively. It can be observed clearly that time changes significantly with the increase of diagonal number. Finally, we increase n and k simultaneously. Figure 4 illustrates this situation. It is obvious that k has an obvious influence on the calculation speed. All these observations coincide with the time complexity we derived in Section 3. It means that if the diagonal number is within an appropriate range, our algorithm can perform effectively and efficiently even for very large scale problems. And these kind of problems cover a large portion of the whole problem set. Therefore, the algorithm we proposed will have great potential in many real-world applications.

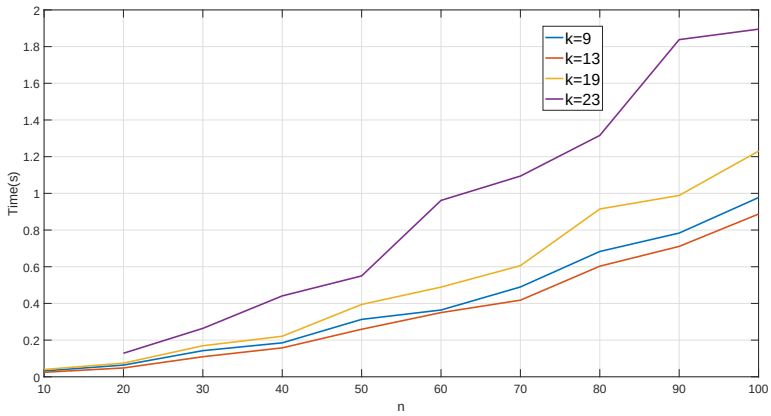

Figure 2. Calculation time with different dimensions when the diagonal number $k = 9, 13, 19, 23$.

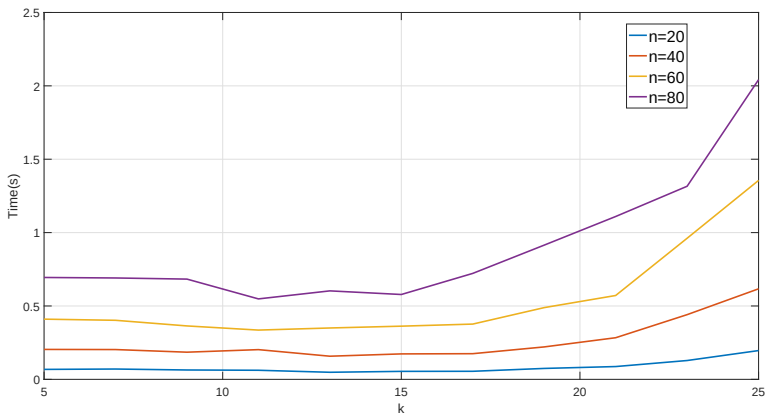

Figure 3. Calculation time with different diagonal numbers when the dimension $n = 20, 40, 60, 80$.

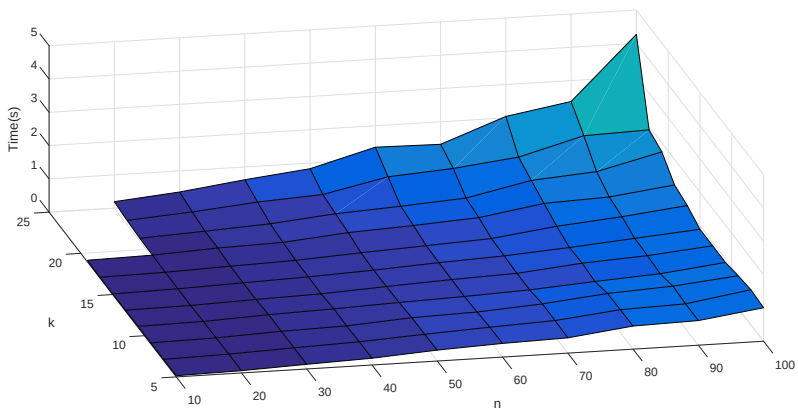

Figure 4. The variation of calculation time with different dimensions n and diagonal number k.

6. Conclusions

In this paper, a novel exact algorithm to the general linearly constrained zero-one quadratic programming problem with k-diagonal matrix is proposed. The algorithm is designed by analyzing the property of matrix Q and then combining the famous basic algorithm and dynamic programming method. The complexity of the algorithm is analyzed and shows that it is polynomially solvable when m is fixed. The experimental results also illustrate the feasibility and efficiency of the algorithm. Designing efficient algorithm to this special class of problem $01CQP$ not only provides useful

information for designing efficient algorithms for other special classes but also can provide hints and facilitate the derivation of efficient relaxations for the general problems. And finally, the phasor measurement units placement problem is used to demonstrate that the algorithm has wide potential applications in decision-making real-life problems.

Author Contributions: S.G. put forward ideas and algorithms. S.G. and X.C simulated the results and wrote important parts of the article. X.C. formatted the whole paper, and summarized the results in tables. All authors have read and agreed to the published version of the manuscript.

Funding: The work described in the paper was supported by the National Science Foundation of China under Grants 61876105 and 61503233.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

01QP	Zero-One Quadratic Programming
01UQP	Unconstrained Zero-One Quadratic Programming
01CQP	Constrained Zero-One Quadratic Programming
PMU	Phasor Measurement Units
NP-hard	Non-deterministic polynomial-hard

Variables

The following variables are used in this manuscript:

Q	$Q = (q_{ij})_{n\times n}$ and only q_{ij}=$q_{ji}(i = 1, 2, \ldots, n-1, j = i+1, i+2, \ldots, i+m)$ are not zero
c	$c = (c_{ij})_{n\times 1}$
a	$a = (a_{ij})_{n\times 1}$
b	$b \in \mathbb{Z}_+$
n	Dimensions of matrix Q
m	$m \in [0, n-1]$ and $m \in \mathbb{Z}$
k	$k = 2m+1$ $(m = 0, 1, 2, \ldots, n-1)$
s_k	State variable $s_k \in \mathbb{Z}$, stage variable $k(0 < k \leq n)$
$f(x)$	$f(x) = \frac{1}{2}x^T Qx + c^T x$
H	$H = (h_{ij})_{n\times n}$ is a network graph structure
N	$N \in \mathbb{R}^n$ represents the upper bound of the maximum redundancy of each bus
R	$R \in \mathbb{R}^{n\times n}$ is the importance of each bus
Q	$Q \in \mathbb{R}^{n\times n}$ in the application is the cost of placing the PMU on the bus
$M(x)$	Column vector consisted of $m_i(x) = (1-x_i)\prod_{j\in A_i}(1-x_j)(i \in \Omega)$
Ω	Bus set
λ	Weight
A_i	Nodes adjacent to the bus i

References

1. Cook, W.J.; Cunningham, W.H.; Pulleyblank, W.R.; Schrijver, A. *Combinatorial Optimization*; Wiley-Interscience: Hoboken, NJ, USA, 1997.
2. Hammer, P.L.; Rudeanu, S. *Boolean Methods in Operations Research and Related Areas*; Spring: Berlin/Heidelberg, Germany; New York, NY, USA, 1968.
3. Chang, K.C.; Du, H.C. Efficient algorithms for layer assignment problem. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1987**, *6*, 67–78. [CrossRef]
4. Dehghan, A.; Mubarak, S. Binary quadratic programing for online tracking of hundreds of people in extremely crowded scenes. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *40*, 568–581. [CrossRef] [PubMed]
5. Laughhunn, D.J. Quadratic binary programming with application to capital-budgeting problems. *Oper. Res.* **1970**, *18*, 454–461. [CrossRef]

6. Kizys, R.; Juan, A.; Sawik, B.; Calvet, L. A biased-randomized iterated local search algorithm for rich portfolio optimization. *Appl. Sci.* **2019**, *9*, 3509. [CrossRef]
7. Huang, L.; Zhang, Q.; Sun, L.; Sheng, Z. Robustness analysis of iterative learning control for a class of mobile robot systems with channel noise. *IEEE Access* **2019**, *7*, 34711–34718. [CrossRef]
8. Manoli, G.; Rossi, M.; Pasetto, D.; Deiana, R.; Ferraris, S.; Cassiani, G.; Putti, M. An iterative particle filter approach for coupled hydro-geophysical inversion of a controlled infiltration experiment. *J. Comput. Phys.* **2015**, *283*, 37–51. [CrossRef]
9. Mesbah, A. Stochastic model predictive control with active uncertainty learning: A Survey on dual control. *Ann. Rev. Control* **2018**, *45*, 107–117. [CrossRef]
10. Imani, M.; Braga-Neto, U. Maximum-Likelihood adaptive filter for partially-observed boolean dynamical systems. *IEEE Trans. Signal Process.* **2017**, *65*, 359–371. [CrossRef]
11. Imani, M.R.; Dougherty, E.; Braga-Neto, U. Boolean Kalman filter and smoother under model uncertainty. *Automatica* **2020**, *111*, 108609. [CrossRef]
12. Baldwin, T.L.; Mili, L.; Boisen, M.B. Power system observability with minimal phasor measurement placement. *IEEE Trans. Power Syst.* **1993**, *8*, 707–715. [CrossRef]
13. Nuqui, R.F.; Phadke, A.G. Phasor measurement unit placement techniques for complete and incomplete observability. *IEEE Trans. Power Deliv.* **2005**, *20*, 2381–2388. [CrossRef]
14. Xu, B.; Abur, A. Observability analysis and measurement placement for system with PMUs. *IEEE PES Power Syst. Conf. Expos.* **2004**, *2*, 943–946.
15. Gou, B. Generalized integer linear programming formulation for optimal PMU placement. *IEEE Trans. Power Syst.* **2008**, *23*, 1099–1104. [CrossRef]
16. Kavasseri, R.; Srinivasan, S.K. Joint placement of phasor and power flow measurements for observability of power systems. *IEEE Trans. Power Syst.* **2011**, *26*, 1929–1936. [CrossRef]
17. Poljak, S.; Rendl, F. Solving the max-cut problem using eigenvalues. *Discret. Appl. Math.* **1995**, *62*, 249–278. [CrossRef]
18. Rendl, F.; Rinaldi, G.; Wiegele, A. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. *Lect. Notes Comput.* **2007**, *4513*, 295–309. [CrossRef]
19. Ye, Y. A .699-approximation algorithm for Max-Bisection. *Math. Program.* **2001**, *90*, 101–111. [CrossRef]
20. Pardalos, P.M.; Rodgers, G.P. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing* **1990**, *45*, 131–144. [CrossRef]
21. Li, D.; Sun, X.L.; Liu, C.L. An exact solution method for unconstrained quadratic 0-1 programming: A geometric approach. *J. Glob. Optim.* **2012**, *52*, 797–829. [CrossRef]
22. Gu, S.; Chen, X.Y.; Wang L. Global optimization of binary quadratic programming: A neural network based algorithm and its FPGA Implementation. *Neural Process. Lett.* **2019**, 1–20. [CrossRef]
23. Zhu, W.X. Penalty parameter for linearly constrained 0-1 quadratic programming. *J. Optim. Theory Appl.* **2003**, *116*, 229–239. [CrossRef]
24. Katayama, K.; Narihisa, H. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *Eur. J. Oper. Res.* **2001**, *134*, 103–119. [CrossRef]
25. Jiang, D.; Zhu, S. A method to solve nonintersection constraint 0-1 quadratic programming model. *J. Southwest Jiaotong Univ.* **1997**, *32*, 667–671.
26. Ranjbar, M.; Effati, S.; Miri, S.M. An artificial neural network for solving quadratic zero-one programming problems. *Neurocomputing* **2017**, *235*, 192–198. [CrossRef]
27. Ping, W.; Weiqing, X. Binary ant colony algorithm with controllable search bias for unconstrained binary quadratic problem. In Proceedings of the 2012 International Conference on Electronics, Communications and Control, Zhoushan, China, 16–18 October 2012.
28. Gu, S.; Cui, R. Polynomial time solvable algorithm to linearly constrained binary quadratic programming problems with Q being a five-diagonal matrix. In Proceedings of the Fiveth International Conference on Intelligent Control and Information Processing, Dalian, China, 18–20 August 2014; pp. 366–372.
29. Gu, S.; Cui, R.; Peng, J. Polynomial time solvable algorithms to a class of unconstrained and linearly constrained binary quadratic programming problems. *Neurocomputing* **2016**, *198*, 171–179. [CrossRef]

30. Gu, S.; Peng, J.; Cui, R. A polynomial time solvable algorithm to binary quadratic programming problems with Q being a seven-diagonal matrix and its neural network implementation. In Proceedings of the ISNN 2014—Advances in Neural Networks, Macao, China, 28 November–1 December 2014; pp. 338–346.
31. Gu, S.; Chen, X.Y. The basic algorithm for zero-one unconstrained quadratic programming problem with k-diagonal matrix. In Proceedings of the Twelfth International Conference on Advanced Computational Intelligence International, Dali, China, 14–16 March 2020.