Computers and Operations Research 113 (2020) 104774

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Closed-form formulas for evaluating r-flip moves to the unconstrained binary quadratic programming problem

Eduardo A.J. Anacleto [a,*], Cláudio N. Meneses [a], Santiago V. Ravelo [b]

[a] Center of Mathematics, Computation and Cognition, Federal University of ABC, Santo André, Brazil
[b] Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

ARTICLE INFO

Article history:
Received 6 February 2019
Revised 29 July 2019
Accepted 19 August 2019
Available online 22 August 2019

Keywords:
Zero-one optimization
Unconstrained binary quadratic programming
Fast flip moves
Computational efficiency

ABSTRACT

The Unconstrained Binary Quadratic Programming problem (UBQP) belongs to the NP-hard class and has become a framework for modeling a variety of combinatorial optimization problems. The methods most commonly used to solve instances of the UBQP explore the concept of neighborhood of a solution. Given a binary vector $x \in \{0, 1\}^n$, solution to a UBQP instance, a neighborhood of x can be defined by flip moves. *Flip moves* consist on selecting one or more elements (positions) of x and "flip" their values to their complementary values (i.e., from 1 to 0 or from 0 to 1). Normally, those methods compute a large number of flip moves, and so the whole process to solve an instance can be quite time consuming. In order to reduce this time, some works have proposed ways to efficiently evaluate one or two flip moves, and also extensions to higher order moves. In this paper we propose two closed-form formulas for evaluating quickly any order of flip moves. To test our theoretical findings, we executed an extensive set of computational experiments over well-known instances for the problem. Against common belief, our results show that it is possible to compute high order flip moves very fast.

1. Introduction

The Unconstrained Binary Quadratic Programming problem (UBQP) can be defined as

$$\max \; f(x) = x^{\intercal} Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i q_{ij} x_j$$

$$\text{s.t.: } x \in \{0, 1\}^n$$

where $Q = [q_{ij}]$ is an $n \times n$ matrix of rational coefficients. The general case of the UBQP is NP-hard (Pardalos and Rosen, 1987) while some special cases of the problem are known to be polynomial time solvable (Barahona, 1986; Chakradhar and Bushnell, 1992; Pardalos and Jha, 1991; Punnen et al., 2015).

The UBQP is also referred to as QUBO (Quadratic Unconstrained Optimization Problems) (Glover and Kochenberger, 2019). As stated in Beasley (1998), it is sometimes termed as unconstrained quadratic bivalent programming problem (Gulati et al., 1984); unconstrained quadratic zero-one programming problem (Chardaire and Sutter, 1995); quadratic zero-one programming problem (Helmberg and Rendl, 1998); unconstrained pseudo-Boolean quadratic problem (Simone, 1990); unconstrained pseudo-Boolean quadratic zero-one programming problem (Sherali et al., 1995); Boolean quadratic programming problem (Anstreicher, 1998) and binary quadratic program (Glover et al., 1998). According to Lucas (2014), the UBQP is also the same as the Ising Spin Glass problem.

Despite the simplicity of its formulation, a wide range of combinatorial optimization problems can be modeled as UBQP, some of them are: Robust Graph Coloring Problem (Wang and Xu, 2011), Maximum Cut Problem (Kochenberger et al., 2011), Sum Coloring Problem (Douiri and Elbernouss, 2012), Generalized Vertex Cover Problem (Kochenberger et al., 2015) and Maximum Vertex Weight Clique Problem (Wang et al., 2016). Since several problems can be formulated as UBQP, many researches focused on solving it by using exact methods (Dinh et al., 2010; Gueye and Michelon, 2009; Kochenberger et al., 2015; Li et al., 2012; Mauri and Lorena, 2012; Pan et al., 2008) or heuristic algorithms (Hanafi et al., 2013; Liefooghe et al., 2014; Lü et al., 2011; Shylo and Shylo, 2011; Wang et al., 2012a; 2012b; 2013), furthermore a survey of the different

* Corresponding author.
E-mail addresses: eduardo.anacleto@ufabc.edu.br (E.A.J. Anacleto), claudio.meneses@ufabc.edu.br (C.N. Meneses), santiago.ravelo@inf.ufrgs.br (S.V. Ravelo).

https://doi.org/10.1016/j.cor.2019.104774
0305-0548/

strategies to approach the UBQP can be found in the literature (Kochenberger et al., 2014).

In order to solve the UBQP, most of the proposed methods (exact or heuristic) evaluate flip moves. Flip moves consist on choosing one or more binary variables and changing their values to their complementary values (i.e., from 1 to 0 or from 0 to 1). Usually, flip move evaluations occur several times during the algorithm execution, then applying an efficient strategy to evaluate flip moves will improve the performance of different algorithms.

Some studies were made in the direction of efficiently evaluate flip moves, that are the cases of Glover and Hao proposals (Glover and Hao, 2010a; 2010b), giving efficient methods to evaluate flip moves selecting one and two variables (1-flip move and 2-flip moves). Glover and Hao also presented extensions of the methods to higher order moves. Those ideas were implemented in a tabu search with 1-flip move (Lü et al., 2010), in a local search heuristic with 1-flip and 2-flip moves (Hanafi et al., 2013), and in a tabu search which combined 1-flip and 2-flip moves (Wang et al., 2016). The cases of flip moves with higher order are not frequently encountered in the literature, mostly due to the common belief that evaluating large neighborhoods of a solution is always expensive.

The r-flip moves using the 1-flip or 2-flip evaluation techniques in Glover and Hao (2010a,b) require $O(nr)$ asymptotic time complexity. This occurs because $O(n)$ operations are used to update an auxiliary vector each time the 1-flip or the 2-flip evaluation are performed.

In this work we developed two closed-form formulas to reevaluate the objective function value of the UBQP after an r-flip move and we provide algorithms for these formulas with computational complexities of $O(nr)$ and $O(r^2)$. These formulas differ from the existing methods, which usually apply incremental ideas. Also, our formulas are easy to implement and very fast to compute.

This paper is organized as follows. In Section 2 we introduce some notations. In Section 3 we give a closed-form formula to evaluate any order of flip moves and an algorithm to compute it. In Section 4 we propose a second closed-form formula, which considers the existence of an auxiliary vector in order to reduce time consumption. In Section 5 we present computational experiments of heuristics implemented with the proposed formulas. Finally, in Section 6, we give some final comments.

2. Notations and definitions

Before the discussion of our results we specify the notation we use. First, we denote by $\mathbb{N}^*$ the set of positive natural numbers. Then, for any $n \in \mathbb{N}^*$, we denote by $\mathbb{B}^n$ the set of binary vectors of dimension n, and by $\mathbb{Q}^{n\times n}$ the set of rational matrices with dimension $n \times n$.

Notice that the UBQP receives an $n \in \mathbb{N}^*$ and a rational matrix $Q \in \mathbb{Q}^{n\times n}$. In Glover and Hao (2010a) was observed that UBQP over a general matrix Q can be reduced to the case in which Q is lower triangular. So, from now on, we only consider rational matrices for the UBQP that are lower triangular, and we formally define the UBQP as follows.

Problem 1. Unconstrained Binary Quadratic Programming problem (UBQP)

Input: a positive natural number $n \in \mathbb{N}^*$ and a lower triangular matrix of rational coefficients $Q \in \mathbb{Q}^{n\times n}$.

Output: a binary vector $x \in \mathbb{B}^n$ such that the function $f(x) = x^\intercal Qx = \sum_{i=1}^{n}\sum_{j=1}^{i} x_i q_{ij} x_j$ is maximized.

Now we introduce the concepts related to neighborhood and flip moves.

Definition 1. Given two positive natural numbers $n, r \in \mathbb{N}^*$ and a binary vector $x \in \mathbb{B}^n$, we define the r-neighborhood of x as the set of binary vectors $\mathcal{N}_r(x) \subseteq \mathbb{B}^n$, such that $y \in \mathcal{N}_r(x)$ iff y differs from x in exactly r positions (i.e., $|x - y| = \sum_{i=1}^{n} |x_i - y_i| = r$).

Given two binary vectors x and y of same dimension, if $y \in \mathcal{N}_r(x)$ we say that y was obtained by r-flip moves from x, and we define the r-flip moves by the set of positions the vectors differ:

Definition 2. Given two positive natural numbers $n, r \in \mathbb{N}^*$ and two binary vectors $x, y \in \mathbb{B}^n$, such that $y \in \mathcal{N}_r(x)$. We define the r-flip moves that takes from x to y, as the set of positions where x and y differ:

$$N_r(x, y) = \{i \quad | \quad x_i \neq y_i \text{ and } 1 \leq i \leq n\}.$$

We also define the complement of that set, as the set of positions where x and y are equal:

$$\overline{N}_r(x, y) = \{i \quad | \quad x_i = y_i \text{ and } 1 \leq i \leq n\}.$$

Where $N = \{1, 2, \ldots, n\} = N_r(x, y) \cup \overline{N}_r(x, y)$. Observe that $|N_r(x, y)| = r$ and $|\overline{N}_r(x, y)| = n - r$.

If the binary vectors x and y are implicit by the context, then we denote $N_r(x, y)$ and $\overline{N}_r(x, y)$ by N_r and $\overline{N}_r$, respectively.

The problem we approach in this paper is not to solve the UBQP, but to evaluate the objective function of UBQP after r-flip moves, for an arbitrary positive natural r. In order to define that problem, we consider an instance (n, Q) of UBQP is given with two solutions $x, y \in \mathbb{B}^n$, such that $y \in \mathcal{N}_r(x)$, for some positive natural r. The objective is to calculate $f(y) = y^\intercal Qy$, on the assumption we already know the value for $f(x) = x^\intercal Qx$ and we also know the sets N_r and $\overline{N}_r$. Formally:

Problem 2. r-flip move evaluation problem for UBQP

Input: an instance $I = (n, Q)$ of UBQP, a number $r \in \{1, 2, \cdots, n\}$, two solutions $x, y \in \mathbb{B}^n$ for I such that $y \in \mathcal{N}_r(x)$, a rational value $f(x) = x^\intercal Qx$ and the sets N_r and $\overline{N}_r$.

Output: the value of $f(y) = y^\intercal Qy$.

Observe that the r-flip move evaluation problem for UBQP is polynomial time solvable, since a simple $O(n^2)$ algorithm may obtain the value for $f(y) = y^\intercal Qy$. Our objective is not just to calculate $f(y)$, but to compute such value as efficiently as possible. In that direction, next sections present closed-form formulas and algorithms to efficiently obtain $f(y)$.

3. r-flip move evaluation for UBQP

Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP, we propose, in this section, a closed-form formula to evaluate the objective function $f(y) = y^\intercal Qy$.

In order to obtain that formula, first we rewrite $f(z) = \sum_{i=1}^{n}\sum_{j=1}^{i} z_i q_{ij} z_j$, for any vector $z \in \mathbb{B}^n$, by splitting and grouping the summatory in terms that depend on the partition of $\{1, 2, \ldots, n\}$ defined by N_r and $\overline{N}_r$. The following fact shows us how to rewrite $f(z)$.

Fact 1. Given an instance $I = (n, Q)$ of UBQP, a solution $z \in \mathbb{B}^n$ for I and a partition $(N_r, \overline{N}_r)$ of the set $N = \{1, 2, \ldots, n\}$, the formula $f(z) = z^\intercal Qz = \sum_{i=1}^{n}\sum_{j=1}^{i} z_i q_{ij} z_j$ can be written as

$$f(z) = \sum_{\substack{i\in\overline{N}_r \\ j\leqslant i}}\sum_{\substack{j\in\overline{N}_r}} z_i q_{ij} z_j + \sum_{i\in N_r} z_i \Bigg[\sum_{\substack{j\in\overline{N}_r \\ j<i}} q_{ij} z_j + \sum_{\substack{j\in\overline{N}_r \\ j>i}} z_j q_{ji} + \sum_{\substack{j\in N_r \\ j\leqslant i}} q_{ij} z_j \Bigg].$$

Proof. First we split the sum in four terms depending on the partition $(N_r, \overline{N}_r)$, then we rewrite the third term, and finally we group

together the last three terms of the formula:

$$\begin{aligned} f(z) &= \sum_{i=1}^{n}\sum_{j=1}^{i} z_i q_{ij} z_j = \sum_{i\in N}\sum_{\substack{j\in N\\ j\leq i}} z_i q_{ij} z_j \\ &= \sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} z_i q_{ij} z_j + \sum_{i\in N_r}\sum_{\substack{j\in \overline{N}_r\\ j<i}} z_i q_{ij} z_j + \sum_{i\in \overline{N}_r}\sum_{\substack{j\in N_r\\ j<i}} z_i q_{ij} z_j + \sum_{i\in N_r}\sum_{\substack{j\in N_r\\ j\leqslant i}} z_i q_{ij} z_j \\ &= \sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} z_i q_{ij} z_j + \sum_{i\in N_r}\sum_{\substack{j\in \overline{N}_r\\ j<i}} z_i q_{ij} z_j + \sum_{i\in N_r}\sum_{\substack{j\in \overline{N}_r\\ j>i}} z_j q_{ji} z_i + \sum_{i\in N_r}\sum_{\substack{j\in N_r\\ j\leqslant i}} z_i q_{ij} z_j \\ &= \sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} z_i q_{ij} z_j + \sum_{i\in N_r} z_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} z_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} z_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} z_j \Big]. \end{aligned}$$

□

Notice that Fact 1 allows us to write $f(z)$ for any vector $z \in \mathbb{B}^n$, particularly we may write $f(y)$ and $f(x)$ as follows

$$f(y) = \sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} y_i q_{ij} y_j + \sum_{i\in N_r} y_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} y_j \Big]$$

$$f(x) = \sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} x_i q_{ij} x_j + \sum_{i\in N_r} x_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} x_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} x_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} x_j \Big].$$

By subtracting $f(x)$ from $f(y)$ we obtain

$$\begin{aligned} f(y) - f(x) = &\sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} (y_i q_{ij} y_j - x_i q_{ij} x_j) \\ &+ \sum_{i\in N_r} y_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} y_j \Big] \\ &- \sum_{i\in N_r} x_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} x_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} x_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} x_j \Big]. \end{aligned}$$

By definition $x_i = y_i$ for all $i \in \overline{N}_r$, then

$$\sum_{i\in \overline{N}_r}\sum_{\substack{j\in \overline{N}_r\\ j\leqslant i}} (y_i q_{ij} y_j - x_i q_{ij} x_j) = 0.$$

Also $x_i = 1 - y_i$ for all $i \in N_r$, so we obtain

$$\begin{aligned} f(y) - f(x) &= \sum_{i\in N_r} y_i \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} y_j \Big] \\ &\quad - \sum_{i\in N_r} (1-y_i) \Big[\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-y_j) \Big] \\ &= -\sum_{i\in N_r} \Big[(1-2y_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} \Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-y_i-y_j) \Big]. \end{aligned}$$

From the above equation we may express $f(y)$ depending on the previous calculated value of $f(x)$:

$$\begin{aligned} f(y) = f(x) - \sum_{i\in N_r} \Big[&(1-2y_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} \Big) \\ &+ \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-y_i-y_j) \Big]. \end{aligned}$$

We denote this result as the r-flip evaluation formula for UBQP and we enunciate it in the form of a theorem, which takes advantage of the previous known value of $f(x)$ to compute the value of $f(y)$:

Theorem 1. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP, the value of $f(y)$ may be obtained by*

$$\begin{aligned} f(y) = f(x) - \sum_{i\in N_r} \Big[&(1-2y_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} y_j q_{ji} \Big) \\ &+ \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-y_i-y_j) \Big]. \end{aligned}$$

An interesting result we derive from the r-flip evaluation formula for UBQP, is given by the following corollary, which states that we may compute $f(y)$ without actually using any value of y, only depending on x, $f(x)$ and the partition $(N_r, \overline{N}_r)$. Such result could be useful during a local search, where it is common to evaluate the objective function on a neighbor solution without the need of registering its variables.

Corollary 1. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP, the value of $f(y)$ may be obtained by*

$$\begin{aligned} f(y) = f(x) + \sum_{i\in N_r} \Big[&(1-2x_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} x_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} q_{ji} x_j \Big) \\ &+ \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-x_i-x_j) \Big]. \end{aligned}$$

Proof. The r-flip evaluation formula states that

$$\begin{aligned} f(y) = f(x) - \sum_{i\in N_r} \Big[&(1-2y_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} y_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} q_{ji} y_j \Big) \\ &+ \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-y_i-y_j) \Big]. \end{aligned}$$

Since $y_i = 1 - x_i \; \forall i \in N_r$ and $y_i = x_i \; \forall i \in \overline{N}_r$, we have

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r} \Big[(1-2(1-x_i)) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} x_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} q_{ji} x_j \Big) \Big] \\ &\quad - \sum_{i\in N_r}\sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-(1-x_i)-(1-x_j)) \\ &= f(x) + \sum_{i\in N_r} \Big[(1-2x_i) \Big(\sum_{\substack{j\in \overline{N}_r\\ j<i}} q_{ij} x_j + \sum_{\substack{j\in \overline{N}_r\\ j>i}} q_{ji} x_j \Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij} (1-x_i-x_j) \Big]. \end{aligned}$$

□

Also, the r-flip evaluation formula can be adapted to the cases of 1-flip and 2-flip moves, obtaining new closed-form formulas for those cases, already studied in the literature (Glover and Hao, 2010a; 2010b). Corollaries 2 and 3 give us these formulas.

Corollary 2. *Given an instance $I = (n, Q, 1, x, y, f(x), \{k\}, N \setminus \{k\})$ of the 1-flip move evaluation problem for UBQP, the value of $f(y)$ may*

be computed by

$$f(y) = f(x) - (1-2y_k)\Big(q_{kk} + \sum_{j=1}^{k-1} q_{kj}y_j + \sum_{j=k+1}^{n} q_{jk}y_j\Big).$$

Proof. The r-flip evaluation formula states that

$$f(y) = f(x) - \sum_{i\in N_r}\Big[(1-2y_i)\Big(\sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big) + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij}(1-y_i-y_j)\Big].$$

Since $N_1 = \{k\}$ and $\overline{N}_1 = \{1,\ldots,n\}\setminus\{k\}$, we have

$$\begin{aligned} f(y) &= f(x) - \Big[(1-2y_k)\Big(\sum_{j=1}^{k-1} q_{kj}y_j + \sum_{j=k+1}^{n} q_{jk}y_j\Big) + q_{kk}(1-y_k-y_k)\Big] \\ &= f(x) - (1-2y_k)\Big(q_{kk} + \sum_{j=1}^{k-1} q_{kj}y_j + \sum_{j=k+1}^{n} q_{jk}y_j\Big). \end{aligned}$$

□

Corollary 3. *Given an instance $I = (n, Q, 2, x, y, f(x), \{k,\ell\}, N\setminus\{k,\ell\})$ of the 2-flip move evaluation problem for UBQP, where $k<\ell$, the value of $f(y)$ may be computed by*

$$\begin{aligned} f(y) = f(x) &- (1-2y_k)\Big(q_{kk} + \sum_{j=1}^{k-1} q_{kj}y_j + \sum_{\substack{j=k+1\\ j\neq\ell}}^{n} q_{jk}y_j\Big) \\ &-(1-2y_\ell)\Big(q_{\ell\ell} + \sum_{\substack{j=1\\ j\neq k}}^{\ell-1} q_{\ell j}y_j + \sum_{j=\ell+1}^{n} q_{j\ell}y_j\Big) \\ &-q_{\ell k}(1-y_\ell-y_k). \end{aligned}$$

Proof. By replacing the values of $N_2 = \{k,\ell\}$ and $\overline{N}_2 = N\setminus N_2$ in the r-flip evaluation formula we obtain

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[(1-2y_i)\Big(\sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big) + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \Big[(1-2y_k)\Big(\sum_{j=1}^{k-1} q_{kj}y_j + \sum_{\substack{j=k+1\\ j\neq\ell}}^{n} q_{jk}y_j\Big) + q_{kk}(1-y_k-y_k)\Big] - \Big[(1-2y_\ell)\Big(\sum_{\substack{j=1\\ j\neq k}}^{\ell-1} q_{\ell j}y_j + \sum_{j=\ell+1}^{n} q_{j\ell}y_j\Big) \\ &\quad + q_{\ell k}(1-y_\ell-y_k) + q_{\ell\ell}(1-y_\ell-y_\ell)\Big] \\ &= f(x) - (1-2y_k)\Big(q_{kk} + \sum_{j=1}^{k-1} q_{kj}y_j + \sum_{\substack{j=k+1\\ j\neq\ell}}^{n} q_{jk}y_j\Big) \\ &\quad -(1-2y_\ell)\Big(q_{\ell\ell} + \sum_{\substack{j=1\\ j\neq k}}^{\ell-1} q_{\ell j}y_j + \sum_{j=\ell+1}^{n} q_{j\ell}y_j\Big) - q_{k\ell}(1-y_k-y_\ell). \end{aligned}$$

□

These results are useful to evaluate the objective function when the neighborhood structure of the designed algorithm requires a 1-flip or 2-flip moves, commonly used in the literature (Hanafi et al., 2013; Lü et al., 2010; Wang et al., 2016). Also, by implementing these specific formulas instead of the general case, we may reduce the computational effort.

Algorithm 3.1, give us how to compute $f(y)$ using the r-flip evaluation formula with time complexity $O(nr)$, improving the $O(n^2)$ that calculates $f(y)$ from scratch (without using previous information). Notice that, for fixed values of r, the time complexity results in $O(n)$, achieving a linear algorithm to compute $f(y)$.

Algorithm 3.1: r-Flip_Evaluation.

```
Input  : n, Q, r, y, f_x, N_r, N̄_r
Output: f_y
f_y ← f_x
for i ∈ N_r do
    for j ∈ N̄_r do
        if y_j = 1 then
            if j<i then f_y ← f_y − (1 − 2y_i)q_ij
            else f_y ← f_y − (1 − 2y_i)q_ji
    f_y ← f_y − (1 − 2y_i)q_ii
    k ← 1
    j ← N_r[k]
    while j<i do
        f_y ← f_y − (1 − y_i − y_j)q_ij
        k ← k + 1
        j ← N_r[k]
```

4. r-flip evaluation with a reevaluation vector

Inspired by the results in Glover and Hao (2010a,b), in this section we show how to reduce the computational effort needed to calculate the r-flip evaluation formula by maintaining a vector with the values of $f(z) - f(x)$ for each $z \in \mathcal{N}_1(x)$, besides knowing $f(x)$, N_r and $\overline{N}_r$.

From now on, we denote by Δx the vector containing the values of $f(z) - f(x)$ for each $z \in \mathcal{N}_1(x)$, where Δx_i represents $f(z) - f(x)$ iff z can be obtained by flipping the ith position of x. Then, by Corollary 2

$$\begin{aligned} \Delta x_i &= f(z) - f(x) \\ &= \Big[f(x) - (1-2x_i)\Big(q_{ii} + \sum_{j=1}^{i-1} q_{ij}x_j + \sum_{j=i+1}^{n} q_{ji}x_j\Big)\Big] - f(x) \\ &= -(1-2x_i)\Big(q_{ii} + \sum_{j=1}^{i-1} q_{ij}x_j + \sum_{j=i+1}^{n} q_{ji}x_j\Big). \end{aligned}$$

On the hypothesis we know Δx, which we call by reevaluation vector of x, the following theorem gives us a new closed-form formula to solve the r-flip move evaluation problem for UBQP.

Theorem 2. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP and the vector Δx, the value of $f(y)$ may be obtained by*

$$f(y) = f(x) - \sum_{i\in N_r}\Big[\Delta x_i + y_i\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2y_j) - (1-2y_i)\sum_{\substack{j\in N_r\\ j>i}} q_{ji}(1-y_j)\Big].$$

Proof. The r-flip evaluation formula states that

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[(1-2y_i)\Big(\sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j\leqslant i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[(1-2y_i)\Big(\sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big) \\ &\quad + q_{ii}(1-y_i-y_i) + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[(1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big]. \end{aligned}$$

Since $y_i = 1-x_i\ \forall i\in N_r$ and $y_i = x_i\ \forall i\in\overline{N}_r$, then we can replace y_i by x_i in $(1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}y_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}y_j\Big)$ and obtain

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[(1-2(1-x_i))\Big(q_{ii} + \sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}x_j\Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[-(1-2x_i)\Big(q_{ii} + \sum_{\substack{j\in\overline{N}_r\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in\overline{N}_r\\ j>i}} q_{ji}x_j\Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big]. \end{aligned}$$

Since $N = N_r\cup\overline{N}_r$, then

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[-(1-2x_i)\Big(q_{ii} + \sum_{\substack{j\in N\\ j<i}} q_{ij}x_j - \sum_{\substack{j\in N_r\\ j<i}} q_{ij}x_j \\ &\quad + \sum_{\substack{j\in N\\ j>i}} q_{ji}x_j - \sum_{\substack{j\in N_r\\ j>i}} q_{ji}x_j\Big) + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[-(1-2x_i)\Big(q_{ii} + \sum_{\substack{j\in N\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N\\ j>i}} q_{ji}x_j\Big) \\ &\quad + (1-2x_i)\Big(\sum_{\substack{j\in N_r\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N_r\\ j>i}} q_{ji}x_j\Big) + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big]. \end{aligned}$$

Notice that, for each $i\in N$

$$\begin{aligned} \Delta x_i &= -(1-2x_i)\Big(q_{ii} + \sum_{j=1}^{i-1} q_{ij}x_j + \sum_{j=i+1}^{n} q_{ji}x_j\Big) \\ &= -(1-2x_i)\Big(q_{ii} + \sum_{\substack{j\in N\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N\\ j>i}} q_{ji}x_j\Big). \end{aligned}$$

Then

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i + (1-2x_i)\Big(\sum_{\substack{j\in N_r\\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N_r\\ j>i}} q_{ji}x_j\Big) \\ &\quad + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big]. \end{aligned}$$

By replacing $x_i = 1-y_i\ \forall i\in N_r$, we obtain

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i + (1-2(1-y_i))\Big(\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_j) \\ &\quad + \sum_{\substack{j\in N_r\\ j>i}} q_{ji}(1-y_j)\Big) + \sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-y_i-y_j)\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i + y_i\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2y_j) \\ &\quad - (1-2y_i)\sum_{\substack{j\in N_r\\ j>i}} q_{ji}(1-y_j)\Big]. \end{aligned}$$

□

Similarly to the r-flip evaluation formula, we may express this new result in terms of x, without using the values of y.

Corollary 4. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP and the vector Δx, the value of $f(y)$ may be obtained by*

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i - (1-x_i)\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2x_j) \\ &\quad + (1-2x_i)\sum_{\substack{j\in N_r\\ j>i}} q_{ji}x_j\Big]. \end{aligned}$$

Proof. Theorem 2 states that

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i + y_i\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2y_j) \\ &\quad - (1-2y_i)\sum_{\substack{j\in N_r\\ j>i}} q_{ji}(1-y_j)\Big]. \end{aligned}$$

Since $y_i = 1-x_i\ \forall i\in N_r$, we have

$$\begin{aligned} f(y) &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i + (1-x_i)\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2(1-x_j)) \\ &\quad - (1-2(1-x_i))\sum_{\substack{j\in N_r\\ j>i}} q_{ji}(1-(1-x_j))\Big] \\ &= f(x) - \sum_{i\in N_r}\Big[\Delta x_i - (1-x_i)\sum_{\substack{j\in N_r\\ j<i}} q_{ij}(1-2x_j) \\ &\quad + (1-2x_i)\sum_{\substack{j\in N_r\\ j>i}} q_{ji}x_j\Big] \end{aligned}$$

□

By using the result from Theorem 2, we designed Algorithm 4.1 with $O(r^2)$ time complexity. Notice that, for fixed values of r this algorithm runs in constant time, and overall is asymptotically better than Algorithm.

Algorithm 4.1: r-Flip_RV_Evaluation_y.

```
Input  : n, Q, r, y, f_x, N_r, Δx
Output: f_y
1  f_y ← f_x
2  for k = 1 to r do
3      i ← N_r[k]
4      f_y ← f_y − Δx_i
5      for ℓ = 1 to r do
6          j ← N_r[ℓ]
7          if j < i then
8              if y_i = 1 then f_y ← f_y − (1 − 2y_j)q_ij
9          else if j > i then
10             if y_j = 0 then f_y ← f_y + (1 − 2y_i)q_ji
```

The neighborhood $\mathcal{N}_r(x)$ contains $O(n^r)$ solutions. So, if we use Algorithm to evaluate all those solutions, the time complexity would be $O(n^r nr) = O(n^{r+1}r)$, while if we use Algorithm 4.1, the time complexity would result in $O(n^r r^2)$ plus the time of computing the vector Δx. If $r = 1$, then, by using Algorithm, to evaluate the neighborhood will take $O(n^2)$ time, so any algorithm A that computes Δx in time $O(n^2)$ will guarantee also $O(n^2)$ time complexity when using Algorithm 4.1. But, if $r > 1$ then, the same algorithm A (with time complexity $O(n^2)$) will guarantee the neighborhood evaluation by Algorithm 4.1 in time $O(n^r r^2)$, better than the $O(n^{r+1}r)$ given by Algorithm. Next subsection will discuss strategies to maintain the reevaluation vector of x in better times than $O(n^2)$.

4.1. Reevaluation vector maintenance

First, we must show how to initialize the reevaluation vector of x. In order to first compute that vector, we use a simple and straightforward $O(n^2)$ method given by Algorithm 4.2 . So, by using

Algorithm 4.2: Initial_Reavaliation_Vector.

```
Input  : n, Q, x
Output: Δx
1  for i = 1 to n do
2      Δx_i ← q_ii
3      for j = 1 to i − 1 do
4          if x_j = 1 then Δx_i ← Δx_i + q_ij
5      for j = i + 1 to n do
6          if x_j = 1 then Δx_i ← Δx_i + q_ji
7      if x_i = 0 then Δx_i ← −Δx_i
```

this method we already raise an improvement over the proposed solution without using reevaluation vector (given by Algorithm).

In the direction of avoiding to compute Δx, from the very beginning, for each new solution x, we propose to update the reevaluation vector of x. That is, once a move is made from x to y, where $y \in \mathcal{N}_r(x)$, we will update the revaluation vector instead of initializing it again. Proposition 1 gives us a result that will allow us to realize such update.

Proposition 1. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP and the vector Δx, the reevaluation vector of y can be obtained by*

$$\Delta y_i = \begin{cases} -\Delta x_i + (1-2y_i)\beta_{y_i} & \text{if } i \in N_r \\ \Delta x_i + (1-2y_i)\beta_{y_i} & \text{if } i \in \overline{N}_r \end{cases}$$

where $\beta_{y_i} = \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2y_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2y_j)$ for all $i \in N$.

Proof. Let D_i be the difference between the reevaluation vectors Δy_i and Δx_i, that is $D_i = \Delta y_i - \Delta x_i \ \forall i \in N$. By the Δy definition we know

$$\Delta y_i = -(1-2y_i)\Big(q_{ii} + \sum_{j=1}^{i-1} q_{ij}y_j + \sum_{j=i+1}^{n} q_{ji}y_j\Big).$$

Then

$$\begin{aligned} D_i &= -\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{j=1}^{i-1} q_{ij}y_j + \sum_{j=i+1}^{n} q_{ji}y_j\Big) \\ &= -\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}y_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}y_j\Big). \end{aligned}$$

Since $N_r \cup \overline{N}_r = N$, we have

$$\begin{aligned} D_i = &-\Delta x_i - (1-2y_i) \\ &\Big(q_{ii} + \sum_{\substack{j\in N_r \\ j<i}} q_{ij}y_j + \sum_{\substack{j\in \overline{N}_r \\ j<i}} q_{ij}y_j + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}y_j + \sum_{\substack{j\in \overline{N}_r \\ j>i}} q_{ji}y_j\Big). \end{aligned}$$

By definition $y_i = 1 - x_i \ \forall i \in N_r$ and $y_i = x_i \ \forall i \in \overline{N}_r$, then

$$\begin{aligned} D_i = &-\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-x_j) + \sum_{\substack{j\in \overline{N}_r \\ j<i}} q_{ij}x_j \\ &+ \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-x_j) + \sum_{\substack{j\in \overline{N}_r \\ j>i}} q_{ji}x_j\Big). \end{aligned}$$

Since $N_r \cup \overline{N}_r = N$, we have

$$\sum_{\substack{j\in \overline{N}_r \\ j<i}} q_{ij}x_j = \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j - \sum_{\substack{j\in N_r \\ j<i}} q_{ij}x_j \qquad \text{and}$$

$$\sum_{\substack{j\in \overline{N}_r \\ j>i}} q_{ij}x_j = \sum_{\substack{j\in N \\ j>i}} q_{ij}x_j - \sum_{\substack{j\in N_r \\ j>i}} q_{ij}x_j.$$

Then

$$\begin{aligned} D_i = &-\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j + \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2x_j) \\ &+ \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2x_j)\Big) \\ = &-\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) \\ &-(1-2y_i)\Big(\sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2x_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2x_j)\Big). \end{aligned}$$

Since $x_i = 1 - y_i$ for all $i \in N_r$, we have

$$\begin{aligned} D_i = &-\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) \\ &-(1-2y_i)\Big(\sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2(1-y_j)) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2(1-y_j))\Big) \\ = &-\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) \\ &+(1-2y_i)\Big(\sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2y_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2y_j)\Big). \end{aligned}$$

We defined $\beta_{y_i} = \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2y_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2y_j)$. Then

$$D_i = -\Delta x_i - (1-2y_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) + (1-2y_i)\beta_{y_i}.$$

Now, we divide in two cases:

- Case 1. If $i \in N_r$, then $y_i = 1 - x_i$ and

$$\begin{aligned} D_i &= -\Delta x_i - (1-2(1-x_i))\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) \\ &\quad + (1-2y_i)\beta_{y_i} \\ &= -\Delta x_i - \Big(-(1-2x_i)\Big(q_{ii} + \sum_{j=1}^{i} q_{ij}x_j + \sum_{j=i+1}^{n} q_{ji}x_j\Big)\Big) \\ &\quad + (1-2y_i)\beta_{y_i} \\ &= -2\Delta x_i + (1-2y_i)\beta_{y_i}. \end{aligned}$$

- Case 2. If $i \in \overline{N}_r$, then $y_i = x_i$ and

$$\begin{aligned} D_i &= -\Delta x_i - (1-2x_i)\Big(q_{ii} + \sum_{\substack{j\in N \\ j<i}} q_{ij}x_j + \sum_{\substack{j\in N \\ j>i}} q_{ji}x_j\Big) + (1-2y_i)\beta_{y_i} \\ &= -\Delta x_i + \Delta x_i + (1-2y_i)\beta_{y_i} \\ &= (1-2y_i)\beta_{y_i}. \end{aligned}$$

From the above cases we obtain

$$D_i = \begin{cases} -2\Delta x_i + (1-2y_i)\beta_{y_i} & \text{if } i \in N_r \\ (1-2y_i)\beta_{y_i} & \text{if } i \in \overline{N}_r \end{cases}$$

Since $\Delta y_i = D_i + \Delta x_i$, then

$$\Delta y_i = \begin{cases} -\Delta x_i + (1-2y_i)\beta_{y_i} & \text{if } i \in N_r \\ \Delta x_i + (1-2y_i)\beta_{y_i} & \text{if } i \in \overline{N}_r \end{cases}$$

□

Similarly to previous results, we can express the formulas to update the reevaluation vector of y without using the actual values y, depending only on x.

Corollary 5. *Given an instance $I = (n, Q, r, x, y, f(x), N_r, \overline{N}_r)$ of the r-flip move evaluation problem for UBQP and the vector Δx, the reevaluation vector of y can be obtained by*

$$\Delta y_i = \begin{cases} -\Delta x_i + (1-2x_i)\beta_{x_i} & \text{if } i \in N_r \\ \Delta x_i - (1-2x_i)\beta_{x_i} & \text{if } i \in \overline{N}_r \end{cases}$$

where $\beta_{x_i} = \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2x_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2x_j)$ for all $i \in N$.

Proof. Since, $y_i = 1 - x_i$ for all $i \in N_r$, we have

$$\begin{aligned} \beta_{y_i} &= \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2y_j) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2y_j) \\ &= \sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2(1-x_j)) + \sum_{\substack{j\in N_r \\ j>i}} q_{ji}(1-2(1-x_j)) \\ &= -\Big[\sum_{\substack{j\in N_r \\ j<i}} q_{ij}(1-2x_j) + \sum_{\substack{j\in N_r \\ j4>i}} q_{ji}(1-2x_j)\Big] \\ &= -\beta_{x_i}. \end{aligned}$$

So, by Proposition 1 and knowing that $\beta_{y_i} = -\beta_{x_i}$, we have

$$\Delta y_i = \begin{cases} -\Delta x_i - (1-2y_i)\beta_{x_i} & \text{if } i \in N_r \\ \Delta x_i - (1-2y_i)\beta_{x_i} & \text{if } i \in \overline{N}_r \end{cases}$$

Since $y_i = 1 - x_i$ for all $i \in N_r$ and $y_i = x_i$ for all $i \in \overline{N}_r$, we obtain

$$\Delta y_i = \begin{cases} -\Delta x_i + (1-2x_i)\beta_{x_i} & \text{if } i \in N_r \\ \Delta x_i - (1-2x_i)\beta_{x_i} & \text{if } i \in \overline{N}_r \end{cases}$$

□

Algorithm 4.3 shows us how to use the result from Proposition 1 to update the reevaluation vector from solution x to y in $O(nr)$ time complexity, improving the $O(n^2)$ required in case we initialized the vector at every solution. This algorithm uses the auxiliary Algorithm 4.4, with time complexity $O(r)$, to compute the values of β_{y_i}.

Algorithm 4.3: Reevaluation_Vector_Update_y.

```
Input  : n, Q, r, y, N_r, Δx
Output: Δy
1  j ← 1
2  for k = 1 to r do
3      i ← N_r[k]
4      β ← GET_β(i, r, y, Q, N_r)
5      Δy_i ← −Δx_i + (1 − 2y_i)β
6      while j<i do
7          β ← GET_β(j, r, y, Q, N_r)
8          Δy_j ← Δx_j + (1 − 2y_j)β
9          j ← j + 1
10     j ← j + 1
11 while j ≤ n do
12     β ← GET_β(j, r, y, Q, N_r)
13     Δy_j ← Δx_j + (1 − 2y_j)β
14     j ← j + 1
```

Algorithm 4.4: Get_β.

```
Input  : i, r, y, Q, N_r
Output: β
1  β ← 0
2  for k = 1 to r do
3      j ← N_r[k]
4      if j < i then β ← β + (1 − 2y_j)q_ij
5      else if j > i then β ← β + (1 − 2y_j)q_ji
```

5. Computational experiments

The previous sections gave us theoretical results that extend the ones in Glover and Hao (2010a,b). Moreover, our results are simple to implement and run very fast.

In this section we present computational experiments showing that those results also may improve the performance of algorithms based on neighborhood evaluations. The selected algorithms to execute the tests were classical heuristics: two Local Searches and a Variable Neighborhood Search (VNS). We compare three implementations of each one of those heuristics: using Algorithms, 4.1, an $O(n^2)$ method based on the objective function formula given by Algorithm 5.1 and the 1-flip computation of Glover and Hao (2010a) given by the Algorithms 5.2–5.4.

The improvement comparison from one implementation to another is based on the following speedup measure: Given two programs A and B that solve the same problem, the speedup of B compared to A quantifies how many times B is faster than A and is calculated by t_A/t_B, where t_A and t_B are, respectively, the processing times of the programs A and B. Observe that, depending on the value of the speedup of B compared to A we have three possibilities: if that value is lower than 1, then B is slower then A; if the value is equal to 1, then both programs have the same processing

Algorithm 5.1: Basic_Evaluation.

```
Input  : n, x, Q
Output: v
v ← 0
for i = 1 to n do
    if x_i = 1 then
        for j = 1 to i do
            if x_j = 1 then v ← v + q_ij
```

Algorithm 5.2: 1-Flip_Evaluation_GH.

```
Input  : k, fx, x, Q, rowValue, colValue
Output: fx
dif ← rowValue[k] + colValue[k] + q_kk
if x_k = 1 then fx ← fx − dif
else fx ← fx + dif
```

Algorithm 5.3: RowCol_Init_GH.

```
Input  : n, x, Q
Output: rowValue, colValue
Initialize rowValue and colValue with zeros
for i = i to n do
    for j = 1 to i do
        if x_j = 1 then rowValue[i] ← rowValue[i] + q_ij
    for j = i + 1 to n do
        if x_j = 1 then rowValue[i] ← colValue[i] + q_ji
```

Algorithm 5.4: RowCol_Update_GH.

```
Input  : n, k, x, Q, rowValue, colValue
Output: rowValue, colValue
for i = k + 1 to n do
    rowValue[i] ← rowValue[i] + (1 − 2x_k)q_ik
for i = 1 to k do
    colValue[i] ← colValue[i] + (1 − 2x_k)q_ki
```

time; otherwise B is faster than A (when the value is greater than 1).

In order to execute our experiments we selected instances from OR-Library, available in the public website http://people.brunel.ac.uk/~ma stjjb/jeb/orlib/bqpinfo.html. Each instance is identified as bqp$n$, where bqp means binary quadratic programming and n is the size of the instance, representing the number of elements in a solution vector. There are 10 instances for each one of the available sizes: 50, 100, 250, 500, 1.000 and 2.500. The nonzero coefficient of the instances matrices are integers in the range $[-100, 100]$.

Finally, all of our implementations were done in C programming language, compiled with gcc 5.4.0, executed on a Processor Intel Xeon(R) CPU E5-1620v2 4 cores of 3,70 GHz each and 8 GB of RAM, under Linux Ubuntu 16.04 LTS 64 bits and the CPU times were obtained by using the `gettimeofday()` function.

Next subsections will discuss the results of our tests.

5.1. Local search experiments

Before discussing the computational tests on the local search heuristics, we describe the two variants of this strategy implemented by us. The general local search starts at a solution x and move to a neighbor solution y that improves the objective function value at x, if the solution y exists the process restarts, otherwise it finishes. Usually there are more than one neighbor better than x (i.e. that improve the objective function value at x), so when there are at least two neighbors better than the current solution, we can select the movement in two different ways: to move to the first neighbor better than x, or to move to the best among the neighbors better than x. Algorithms 5.5 and 5.6 give us two local search implementations, each one based on a different movement selection strategy. Also, both algorithms receives a parameter nb to bound the number of solutions of each neighborhood to be analyzed, in our case all the tests were made with $nb = n$.

Algorithm 5.5: Local_Search_Best_First.

```
Input  : n, nb, r, x, Q
Output: x
repeat
    better ← false
    for j = 1 to nb do
        y ← Select a random vector in N_r(x)
        if f(y)<f(x) then
            better ← true
            x ← y
            break
until better = false
```

Algorithm 5.6: Local_Search_Greedy.

```
Input  : n, nb, r, x, Q
Output: x
repeat
    better ← false
    Diff ← 0
    for j = 1 to nb do
        y ← Select a random vector in N_r(x)
        if f(y) − f(x)<Diff then Diff ← f(y) − f(x)
    if Diff<0 then
        better ← true
        x ← y
until better = false
```

Observe that, both Algorithms 5.5 and 5.6 must evaluate, at each iteration, the objective function for each neighbor y selected from a subset of random solutions of $\mathcal{N}_r(x)$. Such evaluation can be executed by using either of the Algorithms, 4.1 or 5.1, so we analyzed the speedups of the implementations using Algorithms and 4.1, compared to the one using Algorithm 5.1. For a given instance, the exactly same sequence of solutions is evaluated by the local search with each algorithm:, 4.1, 5.1 and 5.2. So, each implementation produces the same sequence of objective values for the same sequence of flipped variables.

The speedup values of the Algorithms 5.5 and 5.6 are shown in Tables 1 and 2, respectively. On those tables, the speedup values associated to the implementations based on Algorithm 5.2 are in the column (r1-flip), the speedup values associated to the implementations based on Algorithm are in the column (r-flip), and the speedup values associated to the implementations based on Algorithm 4.1 are in the column (r-flip-RV).

The (BF) column states for Algorithm 5.1 and contains the processing time in seconds of that algorithm.

Most of the speedup values from Tables 1 and 2 are larger than 1 and raise as the instance sizes increase, implying that the implementations based on Algorithms, 4.1 and 5.2 are faster than those based on Algorithm 5.1. Also, we observe that the speedup associated to the implementation based on Algorithm 4.1 (r-flip-RV column) is always greater than 1. Even more, for values of r greater than 1, the speedup values associated to Algorithm 4.1 implementation is much greater than the speedup values associated to the implementations of the Algorithms 5.2 (r1-flip column) and

Table 1
Speedups of the Local_Search_Best_First implementations.

inst	Local_Search_Best_First											
	BF	r1-flip	r-flip	r-flipRV	BF	r1-flip	r-flip	r-flipRV	BF	r1-flip	r-flip	r-flipRV
bqp	$r = 1$				$r = 2$				$r = 50$			
250	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	0.13	10.70	9.01	10.51	0.15	7.38	6.89	9.89	0.06	1.89	1.27	4.62
2	0.10	9.44	7.98	9.04	0.18	6.57	6.16	8.83	0.05	1.83	1.21	4.47
3	0.10	8.08	6.82	8.08	0.08	6.44	6.06	8.51	0.02	1.75	1.20	4.05
4	0.11	10.33	8.72	10.29	0.15	7.76	7.26	10.28	0.04	2.02	1.34	4.78
5	0.10	9.09	7.75	8.92	0.10	6.75	6.35	9.00	0.04	1.80	1.22	4.29
6	0.07	8.53	7.40	8.43	0.14	7.89	7.41	10.59	0.04	1.92	1.30	4.60
7	0.11	10.36	8.81	10.29	0.12	7.34	6.44	9.80	0.03	1.82	1.22	4.33
8	0.08	11.52	9.94	11.47	0.12	7.56	7.08	10.06	0.03	1.84	1.23	4.22
9	0.12	8.98	7.70	9.08	0.07	6.00	5.69	7.96	0.03	1.78	1.22	4.19
10	0.12	9.18	7.77	9.12	0.13	7.89	7.30	10.15	0.05	1.77	1.18	4.23
bqp	$r = 1$				$r = 2$				$r = 100$			
500	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	0.96	19.00	15.57	18.93	1.62	13.61	12.59	19.05	0.32	1.50	1.08	5.54
2	1.27	18.87	15.22	18.76	1.42	13.12	12.15	18.35	0.32	1.58	1.12	5.66
3	0.85	16.33	13.45	16.24	1.00	13.06	12.36	18.34	0.24	1.57	1.09	5.50
4	0.69	18.91	15.74	18.77	0.95	12.63	11.72	17.47	0.24	1.53	1.09	5.45
5	0.76	19.52	16.26	19.45	1.26	12.94	11.98	17.91	0.21	1.50	1.08	5.58
6	0.62	17.93	14.93	17.75	1.81	12.68	12.45	19.56	0.21	1.69	1.21	6.07
7	1.27	19.12	15.58	19.06	1.30	13.70	12.96	19.28	0.24	1.56	1.11	5.63
8	0.89	18.78	15.55	18.70	1.08	11.89	11.07	16.62	0.36	1.51	1.07	5.44
9	0.94	18.53	15.16	18.48	1.37	13.39	12.48	18.82	0.26	1.53	1.10	5.50
10	0.80	19.63	16.27	19.57	1.28	13.32	12.33	18.57	0.26	1.50	1.08	5.43
bqp	$r = 1$				$r = 2$				$r = 200$			
1000	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	8.33	37.82	29.60	37.76	10.85	24.38	21.75	35.72	2.28	1.27	0.89	7.47
2	8.64	36.96	28.65	37.17	23.15	24.86	21.76	37.11	1.75	1.23	0.87	6.62
3	8.58	35.35	27.27	35.21	8.56	23.01	20.46	33.58	3.38	1.22	0.86	7.17
4	12.10	35.35	26.79	35.31	10.91	25.18	22.40	36.89	1.60	1.24	0.87	6.94
5	12.69	36.25	27.72	36.30	7.82	23.39	20.77	34.04	1.92	1.29	0.90	7.61
6	10.49	36.39	27.78	36.34	10.21	24.99	22.44	36.54	3.29	1.25	0.87	7.40
7	13.03	36.02	27.45	36.06	17.05	23.59	20.82	35.03	2.22	1.20	0.84	7.11
8	11.28	35.21	26.77	35.21	11.03	23.65	20.92	34.73	1.75	1.21	0.85	6.98
9	11.44	35.09	26.79	35.04	13.53	23.97	21.14	35.45	1.42	1.24	0.87	7.11
10	9.33	34.59	26.52	34.47	11.87	23.44	20.95	34.43	2.04	1.26	0.88	7.28
bqp	$r = 1$				$r = 2$				$r = 500$			
2500	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	347.69	97.40	67.66	97.01	360.79	58.31	52.27	97.81	25.94	0.83	0.82	8.10
2	162.55	100.34	72.61	99.23	453.93	59.39	52.96	100.23	23.06	0.82	0.82	7.99
3	265.49	99.47	70.43	98.67	340.40	61.03	54.85	102.35	18.16	0.82	0.83	8.06
4	245.73	102.52	72.03	101.93	440.93	59.67	53.19	100.27	36.37	0.80	0.79	7.85
5	259.17	100.47	68.40	99.49	546.84	57.98	50.65	100.41	22.38	0.80	0.79	7.79
6	245.21	96.11	65.99	95.16	442.63	56.53	49.55	97.35	16.57	0.80	0.76	7.51
7	384.65	99.90	67.01	99.37	500.95	56.79	49.61	98.62	28.80	0.79	0.76	7.69
8	433.14	95.77	64.52	95.31	380.85	55.57	49.11	95.30	34.98	0.83	0.79	8.09
9	213.32	91.24	63.20	90.10	351.47	56.49	49.94	96.70	22.90	0.79	0.78	7.61
10	290.14	99.01	67.60	98.21	403.81	59.11	52.00	101.61	31.41	0.80	0.78	7.79

(r-flip column). Moreover, for $r = 1$, the greater speedup values from Table 2 still are from Algorithm 4.1 implementation (r-flipRV column), but those values are very close to the ones given by Algorithm 5.2 implementation (r1-flip column) which are slightly better in Table 1.

Figs. 1 and 2 also compare, respectively, the speedups of the implementations of the Algorithms 5.5 and 5.6. Such speedup comparison was based on the implementations of the heuristics using Algorithms, 4.1 and 5.2 to evaluate flip moves.

Notice that charts (a)–(d), in both Figs. 1 and 2, show that the local search algorithms have a much better performance for the lower values of r, when they use the reevaluation vector technique given by Algorithm 4.1. For greater values of r this performance decreases, until the local search algorithm using the basic reevaluation obtain better performances. That can be explained by the fact that, for greater values of r (i.e. $r = \Omega(n)$), the computational complexities of the reevaluation techniques given by Algorithms, 4.1 and 5.2 for computing r-flip moves become $O(n^2)$ with related constants greater than the one related to Algorithm 5.1. Besides, we observe that, in Figs. 1 and 2, the proportion of performance in their respectively chart (d) is higher than in charts (a), (b) and (c), which means that the higher the size of the instance, the better the speedup of our implementations of local search heuristic based on Algorithms, 4.1, and 5.2 to evaluate flip moves.

5.2. VNS experiments

We implemented and tested not only the local search heuristic, but also the VNS metaheuristic. In this subsection we show and

Table 2
Speedups of the Local_Search_Greedy implementations.

inst	Local_Search_Greedy											
	BF	r1-flip	r-flip	r-flip RV	BF	r1-flip	r-flip	r-flipRV	BF	r1-flip	r-flip	r-flipRV
bqp	$r = 1$				$r = 2$				$r = 50$			
250	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	1.23	7.96	6.21	8.29	0.71	6.11	5.42	8.77	0.10	1.36	0.94	3.73
2	1.07	7.87	6.12	8.19	0.58	4.38	4.03	6.28	0.08	1.85	1.12	4.83
3	0.87	5.62	4.47	5.84	0.60	7.02	6.12	10.08	0.11	1.72	1.04	4.48
4	1.22	10.93	8.40	11.37	0.58	6.91	6.06	9.92	0.09	1.45	0.95	3.90
5	1.32	9.27	7.15	9.64	0.54	4.97	4.51	7.14	0.19	2.23	1.41	6.07
6	1.20	6.91	5.45	7.20	0.77	10.07	8.93	14.50	0.10	2.38	1.56	6.63
7	1.43	13.82	10.58	14.37	0.73	9.70	8.53	13.96	0.11	1.72	1.05	4.48
8	1.33	14.17	10.89	14.74	0.66	6.93	6.07	9.96	0.11	1.60	1.00	4.21
9	1.13	8.54	6.62	8.89	0.35	5.09	4.62	7.31	0.10	2.40	1.60	6.73
10	1.25	8.20	6.38	8.54	0.66	10.78	9.59	15.51	0.08	1.86	1.12	4.86
bqp	$r = 1$				$r = 2$				$r = 100$			
500	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	20.16	20.59	15.19	21.35	10.56	12.65	10.86	18.82	0.80	1.23	0.87	4.93
2	17.17	17.68	13.05	18.34	8.07	8.86	7.93	13.18	0.94	2.08	1.39	8.56
3	15.39	12.47	9.44	12.93	11.12	18.92	16.44	28.21	1.01	2.01	1.32	8.19
4	24.60	25.80	19.04	26.76	13.08	17.53	15.17	26.10	0.98	1.50	0.95	5.89
5	21.76	27.08	20.01	28.09	9.22	11.76	10.11	17.51	0.83	1.39	0.91	5.47
6	20.23	18.12	13.34	18.80	10.14	10.68	9.26	15.90	0.86	2.15	1.48	8.93
7	18.35	15.52	11.46	16.09	12.00	19.18	16.68	28.58	0.79	1.57	0.99	6.15
8	24.53	26.71	19.87	27.70	8.45	13.12	11.22	19.53	1.00	2.00	1.33	8.12
9	18.22	20.34	14.89	21.10	10.42	18.33	15.89	27.31	0.65	1.61	1.02	6.31
10	23.22	25.92	19.18	26.40	10.44	12.99	11.10	19.34	0.82	1.37	0.91	5.40
bqp	$r = 1$				$r = 2$				$r = 200$			
1000	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	372.97	52.51	37.39	54.37	175.44	34.32	29.52	53.41	6.54	1.21	0.83	7.45
2	349.72	51.39	36.68	53.22	148.48	21.37	17.89	33.40	7.59	1.12	0.80	6.94
3	307.99	32.91	23.08	34.11	128.39	19.01	16.14	29.68	8.53	1.72	1.25	10.99
4	292.49	28.93	20.45	29.97	189.15	35.17	30.37	54.95	7.18	1.33	0.90	8.19
5	367.75	53.63	38.51	55.45	149.59	24.66	20.64	38.51	7.46	1.65	1.20	10.48
6	287.73	38.92	27.36	40.36	179.77	33.94	29.18	53.00	5.46	1.34	0.91	8.30
7	344.52	51.02	36.41	52.84	140.62	23.90	20.24	37.86	4.26	1.05	0.78	6.51
8	297.39	36.70	25.68	37.97	135.17	18.61	15.88	29.08	4.04	1.12	0.79	6.90
9	267.83	29.03	20.55	30.04	128.93	20.26	17.04	31.67	3.47	0.86	0.73	5.40
10	311.88	31.97	22.37	33.11	114.66	14.04	12.69	21.94	4.77	0.96	0.76	5.99
bqp	$r = 1$				$r = 2$				$r = 500$			
2500	t(s)	speedup			t(s)	speedup			t(s)	speedup		
1	13220.38	83.43	54.12	86.84	8109.88	75.85	62.86	139.66	120.82	0.86	0.82	8.91
2	16275.45	134.47	86.63	139.98	6504.41	58.38	49.12	107.86	177.55	0.85	0.73	9.29
3	15097.07	105.98	63.86	104.40	7378.03	73.80	56.95	121.87	86.89	0.97	0.83	8.35
4	16111.23	131.40	81.50	136.78	6694.76	57.22	48.28	105.28	86.42	0.68	0.75	6.62
5	14955.03	104.06	67.81	108.30	6016.66	44.19	40.64	81.69	65.32	0.76	0.80	7.81
6	12506.08	79.72	53.36	82.98	6429.20	50.84	44.27	91.12	64.98	0.52	0.72	5.09
7	13203.21	90.12	57.83	92.78	5078.57	27.74	28.56	56.49	172.78	0.65	0.78	6.55
8	12028.88	63.78	44.43	66.38	5694.21	38.29	35.58	68.39	109.04	0.81	0.82	8.26
9	11303.41	65.66	45.13	68.33	6206.45	53.08	45.94	94.88	161.02	1.01	0.97	11.14
10	14021.61	93.68	61.40	97.51	8120.35	73.31	62.31	131.30	117.05	0.73	0.80	7.39

discuss our results for the VNS that uses Algorithms, 4.1, 5.1, and 5.2 to evaluate the flip moves.

According to Hansen and Mladenović (2001), the VNS is a metaheuristic that searches for an optimal solution in a neighborhood structure that changes systematically. Algorithm 5.7 describes the VNS metaheuristic, which iteratively uses a local search heuristic to find better solutions than the current one. If, at any iteration, no better solution is found, then the algorithm expands the neighborhood by increasing the value of r. The expansion process is repeated while a better solution than the current one is not found. The VNS repeats all that process until some stop criteria is satisfied.

Since in the previous subsection we already had two different strategies for local search, we use each one to design two different VNS strategies: one will use the local search described by Algorithm 5.5 and the other will use the local search described by Algorithm 5.6. Also, for each VNS strategy, we implemented four different versions regarding to the objective function evaluation: one version based on Algorithm 5.1, another on Algorithm, another one on Algorithm 4.1 and the last one on Algorithm 5.2. Similarly to the previous subsection, we analyze the speedup of the implementations based on Algorithms, 4.1 and 5.2 compared to the implementations based on 5.1.

Tables 3 and 4 give the results for the VNS based on the local search given by Algorithm 5.5, while Tables 5 and 6 give the results for the VNS based on the local search given by Algorithm 5.6. Analogously to the tables of previous subsection, for these tables the speedup values associated to the implementations based on

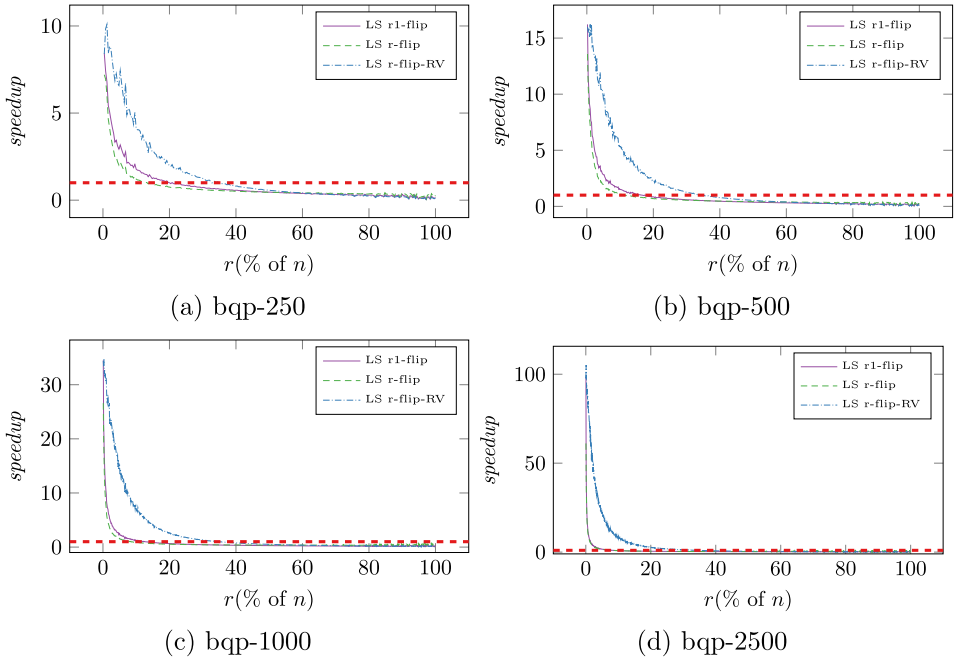

Fig. 1. Speedups of implementations of the Local_Search_Best_First with the Algorithms, 4.1 and 5.2 compared to an equivalent implementation with the Algorithm 5.1. The experiments illustrated in the charts (a), (b), (c) and (d) were done using, respectively, the first instances of sizes 250, 500, 1000 and 2500.

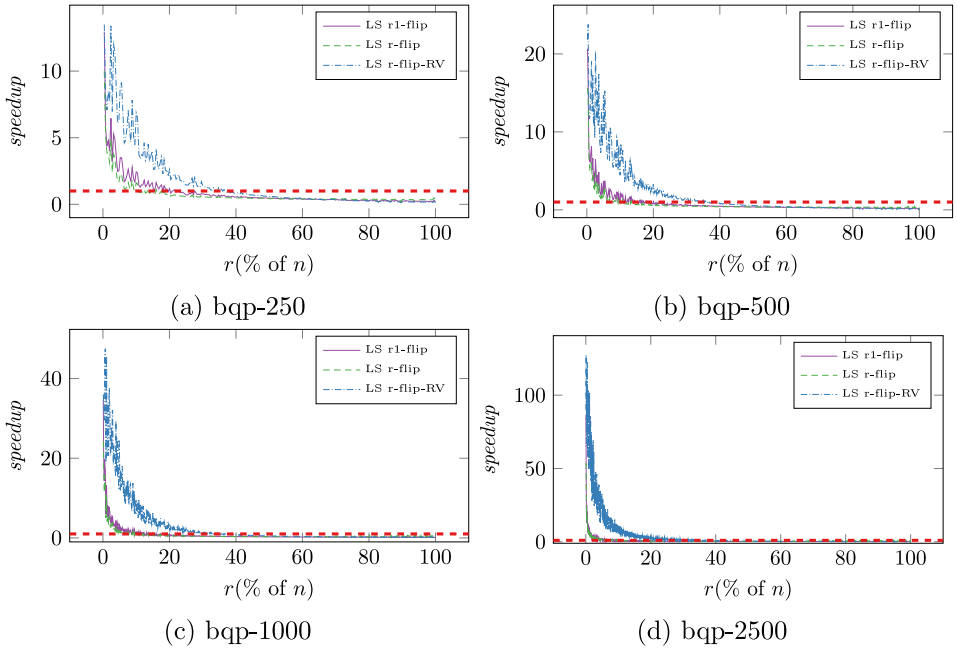

Fig. 2. Speedups of implementations of the Local_Search_Greedy with the Algorithms, 4.1 and 5.2 compared to an equivalent implementation with the Algorithm 5.1. The experiments illustrated in the charts (a), (b), (c) and (d) were done using, respectively, the first instances of sizes 250, 500, 1000 and 2500.

Algorithm are in the column (r-flip), the speedup values associated to the implementations based on Algorithm 4.1 are in the column (r-flip-RV), the speedup values associated to the implementations based on Algorithm 5.2 are in the column (r1-flip), and the processing time in seconds of Algorithm 5.1 are in column (BF). Also, on Tables 3 and 5 we include, for each instance, the value of an optimal solution found by CPLEX[1] solver, and the difference of that solution with the one found by the VNS heuristic in column (Dif.). The results given in Tables 4 and 6 correspond to the instances with greater sizes and the CPLEX solver was not able to found the optimal values in *reasonable* time, so instead of showing an optimal solution value in those tables we showed the best solution found by our heuristics.

All Tables 3–6 show, for all the instances, speedup values greater than 1, where those values increase as the instance size increases. Such speedup results are very similar to the ones obtained in the previous section, which implies that our proposals may improve the performance when evaluating flip moves. Also, one can observe that by using the reevaluation vector we obtain the best

[1] IBM ILOG CPLEX *Optimization Studio Version 12 Release 71*

Algorithm 5.7: VNS.

```
Input  : n, R, Q
Output: x
x ← Initial solution
repeat
    r ← 1
    while r ≤ R do
        y ← Select a random vector in N_r(x)
        y ← run a local search in y
        if f(y)<f(x) then
            x ← y
            r ← 1
        else r ← r + 1
until Stop criteria is satisfied
```

speedup values given in the columns (r-flip-RV), associated to the implementations based on Algorithm 4.1. So, it would be a recommendable strategy to use Algorithm 4.1 in order to design faster algorithms for the UBQP, that require several evaluations in a neighborhood of a solution.

Compared to the CPLEX solver, from Tables 3 and 5 we observe that our VNS implementations performance were much faster for all the tested instances. Furthermore, the VNS implementations found optimal solution values for some of the test cases (those with difference zero), and for the other cases we observe the value of the difference is very small compared to the objective value (less than 10% for almost all the instances). Then, we may suppose that algorithms that evaluate neighborhoods defined by r-flip moves, such as the VNS proposals, could be useful to approach and solve the UBQP, implying that it is interesting to have good strategies to guarantee nice performances of the flip move evaluation.

Similarly as the previous subsection we also illustrate in Figs. 3 and 4, the speedups comparison for the implemented VNS versions. These images give us almost the same results we saw in the previous subsection figures. In both figures, chart (a) shows that the VNS based on Algorithm 4.1 had the best processing time for the smaller values of R, noticing a speedup decreasing while R increases. Also in both figures, chart (b) maintains the same pattern, but shows that the higher the size of the instance, the better the speedup of our implementations.

Fig. 4 compares the speedups of our implementations of the Algorithm 5.7 with the Local Search 5.6 using the Algorithms, 4.1, and 5.2 for evaluating flip moves.

6. Final comments

In this paper we proposed two formulas for efficiently evaluating flip moves that simultaneously change any number of binary variables in a search process. Based on these formulas we developed algorithms with asymptotic time complexities $O(nr)$ and $O(r^2)$, where n is the number of binary variables in a solution vector and r is the number of binary variables that has its values changed in a flip move.

Our best formula, in terms of computational complexity, requires a linear amount of previously calculated values to reevaluate the objective function. We consider such values are allocated in a vector (reevaluation vector). We show how to initialize the reevaluation vector in $O(n^2)$ and how to update it after a move in $O(nr)$ time complexity.

In order to verify the quality of our results, we executed computational experiments over benchmark instances of the UBQP with implementations of two Local Searches and a Variable Neighborhood Search using our evaluation formulas. We compared the processing time of our approaches to equivalent implementations that compute the objective value from scratch, with $O(n^2)$ time complexity, and also using the 1-flip evaluation in Glover and Hao (2010a) iterated r times which has $O(nr)$ time complexity.

The computational experiments showed an improvement in processing time obtained by the implementatins with the formu-

Table 3
Speedups of VNS implementations with the Local_Search Best_First. The value '-' means the VNS found an optimal solution.

inst	Exact Solution		VNS with Local_Search Best_First									
				BF	r1-flip	r-flip	r-flip RV		BF	r1-flip	r-flip	r-flip RV
bqp			$R = 5$					$R = 10$				
50	Value	t(s)	Dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−5176	0.33	119	0.02	2.32	2.18	3.63	52	0.03	1.98	1.48	2.93
2	−3262	0.27	122	0.01	2.05	1.75	2.79	–	0.03	1.85	1.35	2.67
3	−2732	0.41	130	0.01	1.93	1.72	3.01	172	0.03	1.80	1.33	2.61
4	−3169	0.27	216	0.01	1.93	1.80	2.65	214	0.02	1.78	1.31	2.59
5	−3783	0.22	–	0.02	2.23	1.87	3.24	22	0.03	1.85	1.36	2.74
6	−2178	0.35	10	0.02	2.01	1.76	2.87	26	0.03	1.70	1.26	2.49
7	−2666	0.35	338	0.01	2.02	1.66	2.90	173	0.03	1.70	1.26	2.49
8	−3053	0.36	45	0.02	2.20	1.80	3.20	38	0.03	1.85	1.35	2.69
9	−3422	0.26	2	0.01	2.33	1.94	3.30	2	0.03	1.88	1.38	2.71
10	−3420	0.23	120	0.01	2.34	2.08	3.44	122	0.03	1.94	1.44	2.80
bqp	Value	t(s)	$R = 10$					$R = 20$				
100			Dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−12392	1.2	109	0.17	2.84	2.28	4.92	109	0.33	2.08	1.47	3.66
2	−8934	48.4	165	0.13	2.48	1.95	4.30	42	0.30	1.96	1.38	3.48
3	−8183	34.3	–	0.14	2.63	1.97	4.52	590	0.30	1.82	1.29	3.23
4	−9976	33.0	138	0.18	3.00	2.39	5.08	304	0.30	2.02	1.44	3.58
5	−9629	23.9	384	0.14	2.50	1.97	4.34	95	0.30	1.93	1.37	3.39
6	−10468	22.4	8	0.16	2.70	2.13	4.69	64	0.29	1.90	1.32	3.35
7	−10170	13.4	40	0.15	2.75	2.15	4.75	–	0.28	1.94	1.36	3.43
8	−11243	18.0	95	0.15	2.73	2.18	4.76	46	0.30	1.93	1.36	3.44
9	−9976	24.0	186	0.14	2.58	2.03	4.41	157	0.29	1.90	1.33	3.35
10	−9665	29.0	170	0.13	2.54	2.00	4.36	60	0.27	1.84	1.23	3.23

Table 4
Speedups of VNS implementations with the Local_Search Best_First. The value '–' means the VNS found the best solution.

inst	Best Solution	VNS with Local_Search Best_First									
			BF	r1-flip	r-flip	r-flip RV		BF	r1-flip	r-flip	r-flip RV
bqp	value			$R = 25$					$R = 50$		
250		dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−44091	107	3.81	3.52	2.61	8.81	–	6.89	2.03	1.49	4.99
2	−39871	498	3.49	3.15	2.32	7.80	–	6.47	1.92	1.38	4.69
3	−32454	658	3.15	2.88	2.11	7.17	–	6.48	1.89	1.32	4.55
4	−43025	256	3.82	3.24	2.51	8.40	–	7.85	2.27	1.66	5.65
5	−41879	–	3.49	2.94	2.28	7.33	125	6.79	2.01	1.46	4.93
6	−43338	–	3.54	3.28	2.44	8.20	455	6.70	2.01	1.45	4.93
7	−37764	135	3.46	3.14	2.29	7.85	–	6.55	1.92	1.39	4.72
8	−47729	–	3.70	3.40	2.50	8.39	190	6.81	2.04	1.47	4.98
9	−38415	–	3.42	2.95	2.33	7.87	147	6.59	1.94	1.42	4.73
10	−39181	–	3.76	3.37	2.46	8.34	55	6.53	1.93	1.40	4.74
bqp	value			$R = 50$					$R = 100$		
500		dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−121019	635	48.73	3.37	2.42	12.59	–	90.92	1.77	1.35	6.32
2	−115075	–	49.47	3.38	2.39	12.73	710	89.73	1.79	1.33	6.22
3	−110087	1388	47.64	3.28	2.31	12.31	–	91.21	1.82	1.34	6.33
4	−99225	37	43.78	3.03	2.13	11.27	–	86.76	1.73	1.29	5.96
5	−121802	–	49.08	3.37	2.38	12.51	27	92.57	1.85	1.38	6.40
6	−114398	–	49.01	3.36	2.40	12.51	547	92.97	1.85	1.38	6.42
7	−124490	311	48.45	3.34	2.41	12.64	–	89.56	1.79	1.34	6.22
8	−99500	–	44.25	3.06	2.15	11.38	1152	82.85	1.66	1.23	5.74
9	−112220	–	47.22	3.26	2.32	12.15	790	89.76	1.79	1.34	6.25
10	−114850	490	49.40	3.40	2.44	12.64	–	92.29	1.84	1.37	6.42
bqp	value			$R = 100$					$R = 200$		
1000		dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−310331	2503	698.97	2.93	2.01	18.02	–	1310.97	1.47	1.09	7.52
2	−318817	–	667.80	2.82	1.94	17.54	1145	1316.32	1.48	1.10	7.62
3	−331581	–	697.89	2.93	2.04	18.26	1597	1320.56	1.48	1.11	7.57
4	−314542	–	671.30	2.82	1.92	17.56	328	1258.62	1.42	1.04	7.21
5	−307206	400	681.46	2.86	1.96	17.68	–	1292.39	1.45	1.07	7.41
6	−335410	–	700.33	2.94	2.04	18.11	389	1347.50	1.51	1.13	7.69
7	−320828	–	700.05	2.93	2.01	17.86	1384	1319.79	1.48	1.10	7.55
8	−311786	–	697.83	2.92	2.00	17.89	258	1328.73	1.49	1.11	7.60
9	−318893	–	688.86	2.89	1.98	17.77	1804	1327.62	1.49	1.10	7.59
10	−322569	251	692.98	2.91	2.01	17.93	–	1335.41	1.50	1.11	7.64

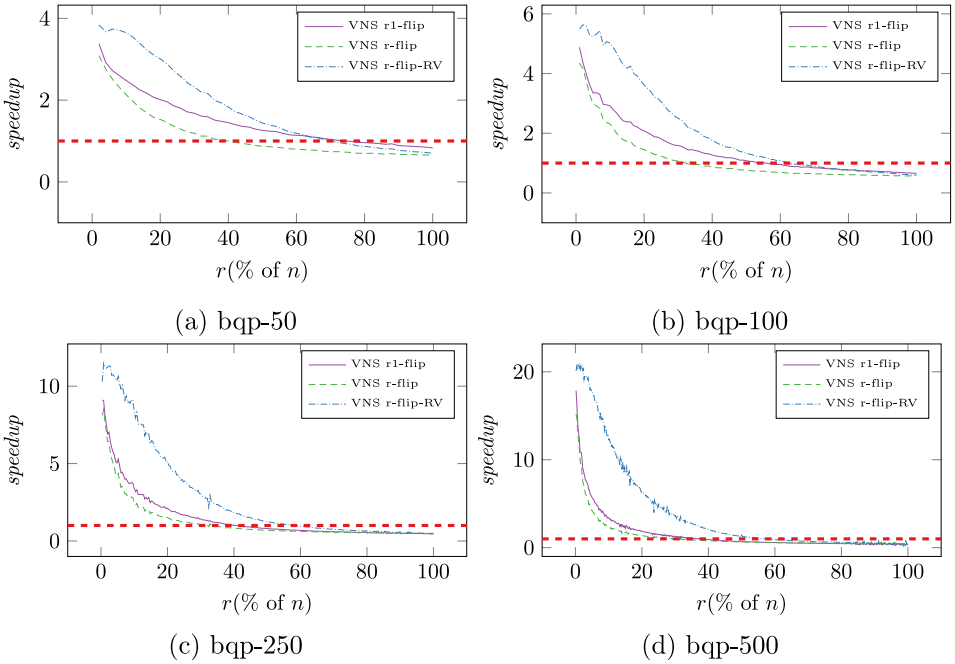

Fig. 3. Speedups of implementations of the VNS with the Local_Search Best_First with the Algorithms, 4.1 and 5.2 compared to an equivalent implementation with the Algorithm 5.1. The experiments illustrated in the charts (a), (b), (c) and (d) were done using, respectively, the first instances of sizes 50, 100, 250 and 500.

Table 5
Speedups of VNS implementations with the Local_Search Best_Greedy. The value '-' means the VNS found an optimal solution.

inst	Exact Solution		VNS with Local_Search_Greedy									
				BF	r1-flip	r-flip	r-flip RV		BF	r1-flip	r-flip	r-flip RV
bqp	Value	t(s)	$R = 5$					$R = 10$				
50			Dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−5176	0.33	–	0.03	2.50	2.09	3.46	–	0.05	1.98	1.48	2.82
2	−3262	0.27	78	0.02	1.98	1.66	2.64	–	0.05	1.91	1.39	2.69
3	−2732	0.41	72	0.02	2.26	1.86	3.15	130	0.05	1.89	1.37	2.69
4	−3169	0.27	64	0.02	2.32	1.95	3.21	–	0.05	1.88	1.38	2.64
5	−3783	0.22	–	0.03	2.22	1.85	3.07	22	0.05	1.87	1.36	2.63
6	−2178	0.35	24	0.02	2.12	1.76	2.87	94	0.05	1.84	1.33	2.63
7	−2666	0.35	137	0.03	2.09	1.73	2.89	165	0.05	1.86	1.36	2.63
8	−3053	0.36	108	0.02	2.23	1.85	3.02	15	0.06	2.01	1.50	2.86
9	−3422	0.26	–	0.02	2.33	1.95	3.18	2	0.05	1.91	1.39	2.70
10	−3420	0.23	120	0.02	2.40	2.00	3.21	26	0.05	1.93	1.43	2.69
bqp	Value	t(s)	$R = 10$					$R = 20$				
100			Dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−12392	1.2	93	0.28	3.20	2.44	4.98	17	0.45	2.21	1.50	3.63
2	−8934	48.4	478	0.27	2.95	2.24	4.55	115	0.42	1.99	1.36	3.20
3	−8183	34.3	392	0.24	2.71	2.04	4.19	389	0.46	1.90	1.30	3.06
4	−9976	33.0	230	0.26	2.79	2.11	4.21	–	0.47	2.18	1.47	3.56
5	−9629	23.9	370	0.29	3.04	2.30	4.81	541	0.48	2.13	1.45	3.49
6	−10468	22.4	30	0.25	2.94	2.22	4.50	559	0.50	2.29	1.56	3.74
7	−10170	13.4	168	0.27	2.95	2.24	4.56	50	0.43	2.14	1.45	3.46
8	−11243	18.0	82	0.27	2.92	2.22	4.48	80	0.45	2.14	1.49	3.47
9	−9976	24.0	286	0.23	2.73	2.13	4.08	153	0.44	1.88	1.29	3.10
10	−9665	29.0	426	0.24	2.72	2.01	4.24	48	0.45	1.98	1.35	3.20

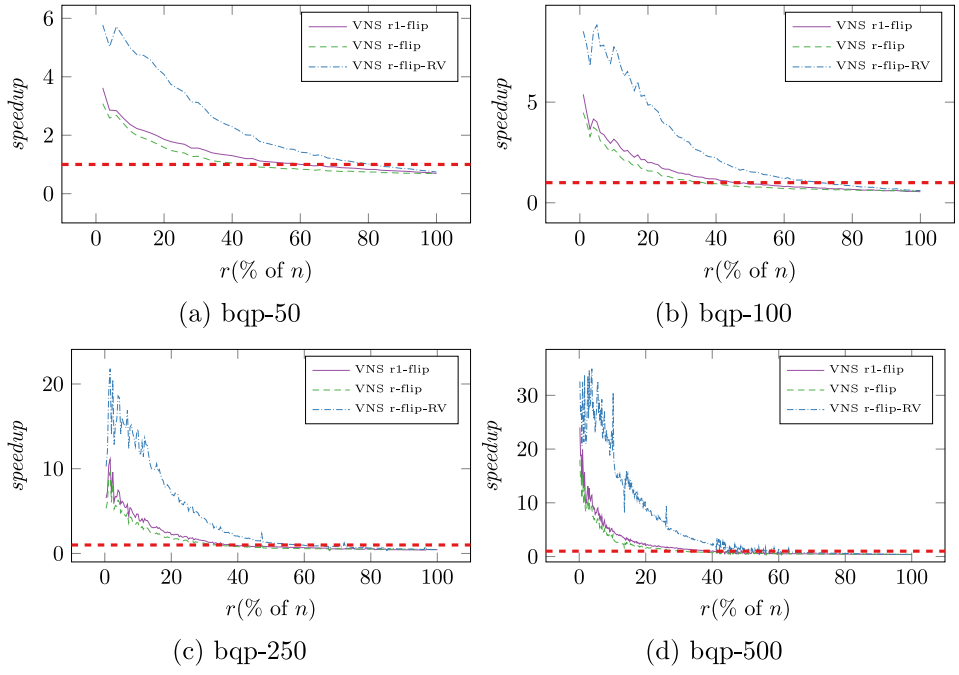

Fig. 4. Speedups of implementations of the VNS with the Local_Search_Greedy with the algorithms, 4.1 and 5.2 compared to an equivalent implementation with the Algorithm 5.1. The experiments illustrated in the charts (a)–(d) were done using, respectively, the first instances of sizes 50, 100, 250 and 500.

las we proposed. We also noticed that such improvement increases as the instance size increases, which we expected because of the asymptotic time complexity of the evaluation algorithms. Besides this improvement, the speedup decreases as the number of flip moves increases, also expected since if the number of flip moves is near n, then our formulas computation time are near $O(n^2)$. We executed the experiments, by increasing the number of flip moves, until the implementations computing the objective function from scratch obtain the best performance, indicating a limit to the use of our formulation.

Currently, we are extending the theoretical results, shown in this paper, to special cases of binary quadratic programming with constraints. Another research path we are working on is to find an optimal r-flip where r depends on the instance size, that is to

Table 6
Speedups of VNS implementations with the Local_Search_Greedy. The value '-' means the VNS found the best solution.

inst	Best Solution	VNS with Local_Search_Greedy									
			BF	r1-flip	r-flip	r-flip RV		BF	r1-flip	r-flip	r-flipRV
bqp				$R = 25$					$R = 50$		
250	value	dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−43847	–	5.42	4.09	2.81	7.90	56	8.49	2.34	1.53	5.03
2	−39770	–	5.14	3.68	2.52	7.12	534	8.79	2.46	1.63	5.36
3	−33473	–	5.90	4.35	2.93	8.48	240	8.39	2.28	1.48	4.92
4	−43977	–	6.34	4.72	3.29	9.24	263	9.14	2.49	1.65	5.51
5	−42136	–	5.83	4.46	3.02	8.79	879	8.70	2.41	1.57	5.20
6	−43668	–	5.96	4.28	2.97	8.29	433	8.39	2.26	1.49	4.92
7	−37179	–	5.78	3.96	2.76	7.64	1007	8.48	2.18	1.43	4.75
8	−48128	–	6.06	4.35	3.05	8.42	363	9.76	2.55	1.70	5.35
9	−38366	817	4.79	3.62	2.51	6.87	–	8.43	2.30	1.52	4.84
10	−39801	209	5.53	4.02	2.77	7.62	–	8.73	2.39	1.56	5.11
bqp	value			$R = 50$					$R = 100$		
500		dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−120867	–	77.90	4.93	3.27	14.60	2612	114.92	2.23	1.46	6.94
2	−115120	–	78.69	4.78	3.10	13.62	311	123.95	2.39	1.57	7.58
3	−108909	–	77.92	4.88	3.17	14.30	707	114.25	2.20	1.44	6.85
4	−97175	–	70.43	4.41	2.88	12.56	3364	104.13	2.03	1.32	6.29
5	−121581	2189	71.28	4.41	2.87	12.37	–	113.90	2.21	1.44	6.83
6	−114178	914	73.49	4.56	2.94	12.78	–	116.00	2.22	1.44	6.83
7	−124268	–	73.98	4.43	2.93	12.04	643	116.96	2.25	1.48	6.92
8	−97014	1978	60.55	3.77	2.45	10.54	–	103.31	2.01	1.31	6.28
9	−111021	–	67.68	4.24	2.74	12.40	756	120.86	2.35	1.54	7.41
10	−114670	–	76.48	4.84	3.17	14.12	2750	110.86	2.14	1.40	6.60
bqp	value			$R = 100$					$R = 200$		
1000		dif.	t(s)	speedup			Dif.	t(s)	speedup		
1	−311024	–	1136.71	4.60	3.06	22.49	4250	1614.50	1.79	1.16	8.21
2	−318279	–	1104.15	4.06	2.72	19.67	2658	1721.82	1.79	1.32	8.89
3	−328043	5011	975.29	3.92	2.61	18.71	–	1606.60	1.78	1.24	8.36
4	−314040	–	1024.52	3.86	2.60	18.65	2481	1516.98	1.68	1.16	7.79
5	−303265	–	963.95	3.79	2.48	17.16	346	1566.97	1.73	1.14	7.89
6	−330783	1873	974.23	3.81	2.51	17.84	–	1752.89	1.89	1.35	9.11
7	−322657	711	1019.41	4.07	2.70	19.26	–	1839.44	2.03	1.42	9.55
8	−311495	–	1130.75	4.55	3.00	22.06	3672	1624.71	1.81	1.24	8.46
9	−322444	–	1066.73	4.26	2.81	19.75	2652	1697.56	1.88	1.31	8.82
10	−322944	-	1083.97	4.35	2.87	20.74	4719	1676.10	1.86	1.29	8.67

find values of r (depending on the instance size) that maximize the speedup of the r-flip moves.

Acknowledgments

This work was supported by the São Paulo Research Foundation - FAPESP (grant 2018/03819–4) and CNPq (grant 312206/2015-1). The authors thank the anonymous reviewers and the editor for the valuable comments.

References

Anstreicher, K., 1998. On the equivalence of convex programming bounds for boolean quadratic programming. Unpublished.
Barahona, F., 1986. A solvable case of quadratic 0–1 programming. Discret. Appl. Math. 13 (1), 23–26. doi:10.1016/0166-218X(86)90065-X.
Beasley, J., 1998. Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem. Technical Report. Management School, Imperial College, London, UK..
Chakradhar, S.T., Bushnell, M.L., 1992. A solvable class of quadratic 0–1 programming. Discret. Appl. Math. 36 (3), 233–251. doi:10.1016/0166-218X(92)90256-A.
Chardaire, P., Sutter, A., 1995. A decomposition method for quadratic zero-one programming. Manag. Sci. 41 (4), 704–712. doi:10.1287/mnsc.41.4.704.
Dinh, T.P., Canh, N.N., Thi, H.A.L., 2010. An efficient combined DCA and b&b using DC/SDP relaxation for globally solving binary quadratic programs. J. Glob. Optim. 48 (4), 595–632. doi:10.1007/s10898-009-9507-y.
Douiri, S.M., Elbernouss, S., 2012. The unconstrained binary quadratic programming for the sum coloring problem. Modern Appl. Sci. 6 (9), 26–33. doi:10.5539/mas.v6n9p26.
Glover, F., Hao, J.K., 2010. Efficient evaluations for solving large 0–1 unconstrained quadratic optimization problems. International Journal of Metaheuristics 1 (1), 3–10. doi:10.1504/IJMHEUR.2010.033120.
Glover, F., Hao, J.K., 2010. Fast two-flip move evaluations for binary unconstrained quadratic optimization problems. Int. J. Metaheuristics 1 (2), 100–107. doi:10.1504/IJMHEUR.2010.034201.
Glover, F., Kochenberger, G., 2019. A Tutorial on Formulating and Using QUBO Methods. Cornell University Library, arXiv: 1 811.11538 [cs.DS].
Glover, F., Kochenberger, G., Alidaee, B., 1998. Adaptive memory tabu search for binary quadratic programs. Manag. Sci. 44, 336–345. doi:10.1287/mnsc.44.3.336.
Gueye, S., Michelon, P., 2009. A linearization framework for unconstrained quadratic (0–1) problems. Discret. Appl. Math. 157 (6), 1255–1266. doi:10.1016/j.dam.2008.01.028.
Gulati, V., Gupta, S., Mittal, A., 1984. Unconstrained quadratic bivalent programming problem. Eur. J. Oper. Res. 15, 121–125. doi:10.1016/0377-2217(84)90055-9.
Hanafi, S., Rebai, A., Vasquez, M., 2013. Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems. J. Heuristics 19 (4), 645–677. doi:10.1007/s10732-011-9169-z.
Hansen, P., Mladenović, N., 2001. Variable neighborhood search: principles and applications. Eur. J. Oper. Res. 130 (3), 449–467. doi:10.1016/S0377-2217(00)00100-4.
Helmberg, C., Rendl, F., 1998. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Math. Program. 82. doi:10.1007/BF01580072.
Kochenberger, G.A., Hao, J.K., Glover, F., Lewis, M., Lü, Z., Wang, H., Wang, Y., 2014. The unconstrained binary quadratic programming problem: a survey. J. Comb. Optim. 28 (1), 58–81. doi:10.1007/s10878-014-9734-0.
Kochenberger, G.A., Hao, J.K., Lü, Z., Wang, H., Glover, F., 2011. Solving large scale max cut problems via tabu search. J. Heuristics 19 (4), 565–571. doi:10.1007/s10732-011-9189-8.
Kochenberger, G.A., Lewis, M., Glover, F., Wang, H., 2015. Exact solutions to generalized vertex covering problems: a Comparison of two models. Optim. Lett. 9 (7), 1331–1339. doi:10.1007/s11590-015-0851-1.

Li, D., Sun, X.L., Liu, C.L., 2012. An exact solution method for unconstrained quadratic 0–1 programming: a geometric approach. J. Glob. Optim. 52 (4), 797–829. doi:10.1007/s10898-011-9713-2.

Liefooghe, A., Verel, S., Hao, J.K., 2014. A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming. Appl. Soft Comput. 16, 10–19. doi:10.1016/j.asoc.2013.11.008.

Lü, Z., Glover, F., Hao, J.K., 2010. A hybrid metaheuristic approach to solving the UBQP problem. Eur. J. Oper. Res. 207 (3), 1254–1262. doi:10.1016/j.ejor.2010.06.039.

Lü, Z., Hao, J.K., Glover, F., 2011. Neighborhood analysis: a case study on curriculum-Based course timetabling. J. Heuristics 17 (2), 97–118. doi:10.1007/s10732-010-9128-0.

Lucas, A., 2014. Ising formulations of many NP problems. Front. Phys. 2, 5. doi:10.3389/fphy.2014.00005.

Mauri, G., Lorena, L., 2012. Improving a lagrangian decomposition for the unconstrained binary quadratic programming problem. Comput. Oper. Res. 39 (7), 1577–1581. doi:10.1016/j.cor.2011.09.008.

Pan, S., Tan, T., Jiang, Y., 2008. A global continuation algorithm for solving binary quadratic programming problems. Comput. Optim. Appl. 41 (3), 349–362. doi:10.1007/s10589-007-9110-4.

Pardalos, P.M., Jha, S., 1991. Graph separation techniques for quadratic zero-one programming. Comput. Math. Appl. 21 (6), 107–113. doi:10.1016/0898-1221(91)90165-Z.

Pardalos, P.M., Rosen, J.B., 1987. Constrained Global Optimization: Algorithms and Applications (Lecture Notes in Computer Science), 268. Springer-Verlag, Berlin Heidelberg doi:10.1007/BFb0000035.

Punnen, A.P., Sripratak, P., Karapetyan, D., 2015. The bipartite unconstrained 0–1 quadratic programming problem: polynomially solvable cases. Discret. Appl. Math. 193, 1–10. doi:10.1016/j.dam.2015.04.004.

Sherali, H., Lee, Y., Adams, W., 1995. A simultaneous lifting strategy for identifying new classes of facets for the boolean quadric polytope. Oper. Res. Lett. 17, 19–26. doi:10.1016/0167-6377(94)00065-E.

Shylo, V.P., Shylo, O.V., 2011. Systems analysis; solving unconstrained binary quadratic programming problem by global equilibrium search. Cybern. Syst. Anal. 47 (6), 889–897. doi:10.1007/s10559-011-9368-5.

Simone, C., 1990. The cut polytope and the boolean quadric polytope. Discret. Math. 79 (1), 71–75. doi:10.1016/0012-365X(90)90056-N.

Wang, F., Xu, Z., 2011. Metaheuristics for robust graph coloring. J. Heuristics 19 (4), 529–548. doi:10.1007/s10732-011-9180-4.

Wang, Y., Hao, J.K., Glover, F., Lü, Z., Wu, Q., 2016. Solving the maximum vertex weight clique problem via binary quadratic programming. J. Comb. Optim. 32 (2), 531–549. doi:10.1007/s10878-016-9990-2.

Wang, Y., Lü, Z., Glover, F., Hao, J.K., 2012a. A Multilevel Algorithm for Large Unconstrained Binary Quadratic Optimization. Springer, Berlin, Heidelberg, pp. 395–408.

Wang, Y., Lü, Z., Glover, F., Hao, J.K., 2012b. Path relinking for unconstrained binary quadratic programming. Eur. J. Oper. Res. 223 (3), 595–604. doi:10.1016/j.ejor.2012.07.012.

Wang, Y., Lü, Z., Glover, F., Hao, J.K., 2013. Probabilistic GRASP-tabu search algorithms for the UBQP problem. Comput. Oper. Res. 40 (12), 3100–3107. doi:10.1016/j.cor.2011.12.006.