Local Search for Integer Quadratic Programming

Xiang He, Peng Lin, Shaowei Cai *

Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China {hexiang, linpeng, caisw}@ios.ac.cn

Abstract

Integer Quadratic Programming (IQP) is an important problem in operations research. Local search is a powerful method for solving hard problems, but the research on local search algorithms for IQP solving is still on its early stage. This paper develops an efficient local search solver for solving general IQP, called LS-IQCQP. We propose four new local search operators for IQP that can handle quadratic terms in the objective function, constraints or both. Furthermore, a twomode local search algorithm is introduced, utilizing newly designed scoring functions to enhance the search process. Experiments are conducted on standard IQP benchmarks QPLIB and MINLPLIB, comparing LS-IQCQP with several state-of-the-art IQP solvers. Experimental results demonstrate that LS-IQCQP is competitive with the most powerful commercial solver Gurobi and outperforms other state-ofthe-art solvers. Moreover, LS-IQCQP has established 6 new records for QPLIB and MINLPLIB open instances.

Introduction

Integer Quadratic Programming (IQP) problems are mathematical optimization problems where the objective function, constraints, or both include quadratic polynomial functions, with the variables required to take integer values. IQP is a generalization of Integer Linear Programming (ILP), extending ILP by incorporating quadratic terms into the objective function or constraints.

With its strong expressive power, IQP can accommodate the requirements of a variety of real-world scenarios, leading to an extensive range of practical applications. Many combinatorial optimization problems can be described using the IQP formulation, such as the quadratic assignment problem (Anstreicher 2003), max-cut (Burer and Vandenbussche 2005), and maximum clique (Bomze et al. 1999). In the industrial domain, many optimization problems can also be modeled using IQP, including finance portfolio optimization (Deng et al. 2013), quadratic network design (Frangioni, Furini, and Gentile 2016), telecommunications routing (Frangioni, Galli, and Scutellà 2015), energy management (Frangioni, Galli, and Stea 2017), automatic control (Axehill 2008) and so on.

Integer Quadratic Programming is NP-hard (Vavasis 1990; Pia, Dey, and Molinaro 2017), making it challeng-

ing to solve. There are two classes of methods of solving Integer Quadratic Programming: complete methods and incomplete methods. Complete methods aim to compute the exact optimal solution and prove its optimality, while incomplete methods focus on obtaining high-quality solutions within a reasonable time. Nearly all of the complete solvers addressing IQP are based on the Branch-and-Bound algorithm (Land and Doig 2010). This includes most state-of-the-art IQP solvers (Mittelmann 2023), such as the commercial solvers Gurobi (Gurobi Optimization 2022) and CPLEX (Nickel et al. 2022), as well as the academic solver SCIP (Achterberg 2009). Nonetheless, Branch-and-Bound algorithms require exponential time in the worst case, and unfortunately, their exponential behavior frequently appears in practice (Furini et al. 2019).

Incomplete methods are usually implemented as local search algorithms, which play an important role in solving NP-hard combinatorial problems (Gu 1992; Goldberg 2004; Lun et al. 2009). These algorithms aim to find good solutions quickly and have shown significant effectiveness in solving pseudo-Boolean optimization (Lei et al. 2021) and integer linear programming (Lin et al. 2023). However, research on local search algorithms for solving Integer Quadratic Programming is still on its early stages.

In practice, IQP instances vary widely due to their diverse properties. Many local search algorithms are devotedly tailored for specific IQP types with certain attributes. For instance, MA (Merz and Katayama 2004) and AC-SIOM (Boros, Hammer, and Tavares 2007) are developed for quadratic unconstrained binary problems. CQP (Gould, Orban, and Robinson 2013) and DQP (Gould and Robinson 2017) are designed for convex problems with linear constraints, while BOPD (Fletcher 2000) and SOIC (Gill and Wong 2015) are suitable for nonconvex problems with linear constraints. There are several local search solvers for Nonlinear Programming (a problem class that includes IQP), such as CONOPT, IPOPT and SNOPT (Drud 1985; Wächter and Biegler 2006; Gill, Murray, and Saunders 2005). Nevertheless, these solvers are dedicated to real variables and not to integer ones. Knitro (Byrd, Nocedal, and Waltz 2006) periodically uses local search algorithms to assist its complete algorithm, but these local search algorithms only serve as an auxiliary method and must be dependent on complete algorithms to work.

^{*}Corresponding author

This paper is dedicated to proposing an efficient local search solver for IQP with two main goals: 1. The solver can handle any type of IQP, whether the problem is convex or not, regardless of the presence of quadratic terms in the objective function, constraints or both. 2. The solver is standalone and can quickly find high-quality solutions for IQP without relying on any complete or other algorithms.

Our Contributions

In this work, we introduce LS-IQCQP, a novel local search solver designed for solving the IQP problem.

- We propose four new local search operators for general IQP, capable of handling both convex and non-convex problems, regardless of quadratic terms in the objective function, constraints or both. Additionally, a two-mode local search algorithm is introduced, utilizing newly designed scoring functions to enhance the search process.
- Experiments are conducted on standard IQP benchmarks QPLIB and MINLPLIB, comparing LS-IQCQP with several state-of-the-art IQP solvers. Experimental results demonstrate the excellent performance of LS-IQCQP, showing it is competitive with the most powerful commercial solver, Gurobi, indicating a significant improvement in the field of local search solvers for IQP.
- LS-IQCQP has established 6 new records for QPLIB and MINLPLIB open instances, further demonstrating its exceptional solving capabilities.

Preliminary

Integer Quadratic Programming

Integer Quadratic Program (IQP) is an optimization problem in which either the objective function, or some of the constraints, or both, are quadratic functions. The problem has the general form as follows:

Minimize
$$\frac{1}{2}\mathbf{x}^{T}\mathcal{Q}^{0}\mathbf{x} + b^{0}\mathbf{x} + q^{0}$$
subject to
$$\frac{1}{2}\mathbf{x}^{T}\mathcal{Q}^{i}\mathbf{x} + b^{i}\mathbf{x} \leq d_{i} \quad i \in \mathcal{M}, \qquad (1)$$

$$l^{j} \leq \mathbf{x}_{j} \leq u^{j} \quad j \in \mathcal{N},$$

$$\mathbf{x} \in \mathbb{Z}^{n}$$

- $\mathcal{N} = \{1, \dots, n\}$ is the set of indices of variables;
- $\mathcal{M} = \{1, \dots, m\}$ is the set of indices of constraints $(\mathcal{Q}^i = 0 \text{ when constraint is linear constraint)};$
- $\mathbf{x} = [x_j]_{j=1}^n$ is a finite vector of integer variables, the variables are restricted to only attain integer values.
- Q^i for $i \in \{0\} \cup \mathcal{M}$ are symmetric $n \times n$ real (Hessian) matrices;
- b^i , d_i for $i \in \{0\} \cup \mathcal{M}$, and q^0 are, respectively, real(integer) n-vectors and constants;
- $-\infty \le l^j \le u^j \le \infty$ are the lower and upper bounds on each variable x_j for $j \in \mathcal{N}$.
- Any problem with a maximization objective can be converted into a minimization problem by negating the objective function.

To better introduce the algorithm, additional notations will be supplemented below. For any variable $x_j \in \text{constraint } i$, the constraint con_i can be expressed using x_j as follows:

$$A \cdot x_j^2 + H(i, x_j) \cdot x_j + I(i, x_j) \le d_i \tag{2}$$

where A is the constant coefficient of x_j^2 in con_i , $H(i,x_j)$ is the coefficient polynomial of x_j in con_i , and $I(i,x_j)$ is the polynomial that represents the terms in con_i that do not contain x_j .

We use f to denote the objective function. The set of variables in the objective function is denoted as V(f), and the set of variables in a constraint con_i as $V(con_i)$. For any variable $x_j \in V(f)$, let the polynomial $\Theta(x_j)$ represent the terms in the objective function that involve x_j :

$$\Theta(x_j) = W \cdot x_j^2 + K(x_j) \cdot x_j \tag{3}$$

where W is the constant coefficient of x_j^2 in objective function, $K(x_j)$ is the coefficient polynomial of x_j in the objective function.

A complete assignment (assignment for short) α for an IQP instance F is a mapping that assigns to each variable an integer, and $\alpha(x_j)$ denotes the value of x_j under α . The value of the objective function under α is denoted as $obj(\alpha)$. Also, We denote $\alpha(H(i,x_j))$, $\alpha(I(i,x_j))$, and $\alpha(K(x_j))$ $\alpha(\Theta(x_j))$ as the values of these polynomials under assignment α , respectively. An assignment α satisfies the constraint con_i if $con_i(\alpha) \leq d_i$, otherwise the constraint is violated. An assignment α is **feasible** if and only if it satisfies all constraints in F. In the remaining part of the paper, α means the current assignment, unless otherwise stated.

Operator for Feasibility

An operator defines how to modify variables to generate new assignments. When an operator is instantiated with a variable, it produces an operation. A local search algorithm progressively takes operations to generate new assignments and tracks the best assignment obtained.

In this section, we propose a new operator *quadratic satisfying move* for violated quadratic constraints. It considers modifying the value of variables in violated constraints towards making them satisfied.

Definition 1. The quadratic satisfying move operator, denoted as $move_{sat}(x_j, con_i, \alpha)$, takes an assignment α and assigns an integer variable x_j to the threshold value making constraint con_i satisfied, where con_i is a violated constraint containing x_j .

Note that for any variable $x_j \in con_i$, con_i is given by $A \cdot x_j^2 + H(i,x_j) \cdot x_j + I(i,x_j) \le d_i$ as in Formula (2). We consider modifying the value of x_j while keeping any other variables fixed as constants. There are two possible forms for con_i with respect to x_j : it is linear if A=0; otherwise, it is quadratic. We will discuss these two cases individually below.

I. Linear Form

For any variable $x_j \in con_i$ with a zero quadratic coefficient A = 0, the constraint con_i is given by: $H(i, x_i)$.

 $x_j + I(i,x_j) \leq d_i$. Under an assignment α , we denote $\nu = \frac{d_i - \alpha(I(i,x_j))}{\alpha(H(i,x_j))}$, $\alpha(x_j^{\text{new}})$ is the value of x_j after performing $move_{sat}(x_j,con_i,\alpha)$ operator. A $move_{sat}(x_j,con_i,\alpha)$ operator for x_j in con_i is:

$$\alpha(x_j^{\text{new}}) = \begin{cases} \lceil \nu \rceil, & \text{if } H(I, x_j) > 0, \\ \lfloor \nu \rfloor, & \text{if } H(I, x_j) < 0, \\ \alpha(x_j), & \text{otherwise.} \end{cases}$$
 (4)

II. Quadratic Form

For any variable $x_j \in con_i$ with a non-zero quadratic coefficient $A \neq 0$, the constraint con_i is given by: $A \cdot x_j^2 + H(i,x_j) \cdot x_j + I(i,x_j) \leq d_i$. Under an assignment α , we denote $\Delta = \alpha(H(I,x_j)^2) - 4 \cdot A \cdot (\alpha(I,x_j) - d_i)$. Depending on the following conditions, $move_{sat}(x_j,con_i,\alpha)$ varies:

- If $\Delta=0$, the equation $A\cdot x_j^2+H(i,x_j)\cdot x_j+I(i,x_j)-d_i=0$ has a single root x_0 with respect to x_j . If x_0 is an integer, then there is a $move_{sat}(x_j,con_i,\alpha)$ such that $\alpha(x_j^{\mathrm{new}})=x_0$.
- If $\Delta > 0$, the equation $A \cdot x_j^2 + H(i, x_j) \cdot x_j + I(i, x_j) d_i = 0$ has two roots x_1 and x_2 with respect to x_j , where $x_1 < x_2$, there are two $move_{sat}(x_j, con_i, \alpha)$:
 - If A>0, then the two $move_{sat}(x_j,con_i,\alpha)$ are $\alpha(x_j^{\mathrm{new}})=\lceil x_1 \rceil$ and $\alpha(x_j^{\mathrm{new}})=\lfloor x_2 \rfloor$.
 - If A<0, then the two $move_{sat}(x_j,con_i,\alpha)$ are $\alpha(x_j^{\mathrm{new}})=\lfloor x_1 \rfloor$ and $\alpha(x_j^{\mathrm{new}})=\lceil x_2 \rceil$.
- Otherwise, there is no $move_{sat}(x_i, con_i, \alpha)$.

Note that if con_i is an inequality, the new integer value of x_j will satisfy con_i . However, if con_i is an equality, we must check whether the new integer value of x_j satisfies con_i . If it does not, the $move_{sat}(x_j, con_i, \alpha)$ operation will be aborted.

Operators for Optimization

For IQP, satisfying constraints is essential to find feasible solutions, while optimizing the objective function is crucial for achieving a (sub)optimal solution. In this section, we propose 3 new operators for optimizing the objective function.

Inequality Exploration Move Operator

We first introduce the *inequality exploration move* operator. It considers modifying the value of $x_j \in V(f)$ and the new value of x_j is calculated based on the satisfied constraint con_i , where con_i is an inequality that involves the x_j .

Definition 2. The inequality exploration move operator, denoted as $move_{exp}(x_j, con_i, \alpha)$, takes an assignment α and assigns an integer variable $x_j \in V(f)$ to a value that both maintains the satisfied state of the inequality constraint con_i involving x_j and maximize the decrease in the value of the objective function.

 con_i is given by: $A \cdot x_j^2 + H(i,x_j) \cdot x_j + I(i,x_j) \le d_i$ with respect x_j by Formula (2). If A = 0, the equation $A \cdot x_j^2 + H(i,x_j) \cdot x_j + I(i,x_j) - d_i = 0$ has a single root x_0 . If $A \ne 0$ and $\Delta > 0$, the equation has two roots x_1 and

 x_2 , where $x_1 < x_2$. The feasible domain of x_j in con_i is denoted as \mathcal{D} , \mathcal{D} is :

$$\mathcal{D} = \begin{cases} [x_0, +\infty] & \text{if } A = 0, \alpha(H(I, x_j)) < 0, \\ [-\infty, x_0] & \text{if } A = 0, \alpha(H(I, x_j)) > 0, \\ [x_1, x_2] & \text{if } A > 0, \Delta \neq 0 \\ [-\infty, x_1] \cup [x_2, +\infty] & \text{if } A < 0, \Delta \neq 0, \end{cases}$$
(5)

Recall that the terms in the objective function that involve x_j can be viewed as a quadratic function of x_j , given by $\Theta(x_j) = W \cdot x_j^2 + K(x_j) \cdot x_j$ according to Formula (3). If $W \neq 0$, let the axis of symmetry of this quadratic function be $\xi = \frac{K(x_j)}{-2W}$. The inequality exploration move operator assigns an integer value $\alpha(x_j^{\text{new}})$ to x_j such that $\alpha(x_j^{\text{new}}) \in \mathcal{D}$ and minimizes $\Theta(x_j)$.

We determine a candidate value x^{\min} based on the specified conditions of con_i and $\Theta(x_j)$, where x^{\min} is the value in $\mathcal D$ that minimizes $\Theta(x)$. This candidate value x^{\min} is then used to find the nearest feasible integer within the domain $\mathcal D$ to apply the $move_{exp}(x_j, \operatorname{con}_i, \alpha)$ operator. Specifically, x^{\min} is:

• If
$$W=0,$$
 $A=0,$ and $\alpha(H(I,x_j))\cdot\alpha(K(x_j))<0,$ then
$$x_{\min}=x_0.$$

• If $W=0, A\neq 0$, and $\pm \infty \notin \mathcal{D}$, then

$$x_{\min} = \underset{x \in \{x_1, x_2\}}{\arg \min} \Theta(x).$$

- If W < 0 and $\pm \infty \notin \mathcal{D}$, or if W > 0,
 - If $\xi \in \mathcal{D}$, then

$$x_{\min} = \underset{x \in \{x_1, x_2, \xi\}}{\arg \min} \Theta(x).$$

- If $\xi \notin \mathcal{D}$, then

$$x_{\min} = \underset{x \in \{x_1, x_2\}}{\arg\min} \Theta(x).$$

• Otherwise, there is no $x^{\text{candidate}}$ for $\alpha(x_i)$ operator.

Finally, we get $move_{exp}(x_j, con_i, \alpha)$ operator as follows: if $\lceil x^{\min} \rceil \in \mathcal{D}$, then $\alpha(x_j^{\text{new}}) = \lceil x^{\min} \rceil$; otherwise, $\alpha(x_i^{\text{new}}) = \lfloor x^{\min} \rfloor$.

Equality Incremental Move Operator

The *equality incremental move* operator is designed for $x_j \in V(f)$ and a satisfied constraint con_i , where con_i is an equality involving x_j . In contrast to inequalities, for a quadratic equality, the feasible region of x_j consists of either a single point or two distinct points. It is challenging to find a value of x_j that keeps the equality satisfied while decreasing the value of the objective function. Therefore, the equality incremental move operator considers modifying both x_j and an auxiliary variable x' to decrease the value of the objective function, where x' is adjusted to maintain the satisfied state of con_i .

Definition 3. The equality incremental move operator, represented as $move_{inc}(x_j, x', con_i, \alpha)$, modifies a variable $x_j \in V(f) \cap V(con_i)$ and another variable $x' \in V(con_i)$, where con_i is a satisfied equality constraint. Firstly, it increases or decreases the value of x_j by 1 so that the value of $\Theta(x_j)$ is decreased, and it updates the value of x' to make con_i remain satisfied.

We first determine the updated value of x_j . The value of x_j is adjusted by a simple incremental move according to $\Theta(x_j)$ to make it decrease: $\alpha(x_j^{\rm inc}) = \alpha(x_j) - 1$ or $\alpha(x_j^{\rm inc}) = \alpha(x_j) + 1$, such that $\alpha(\Theta(x_j^{\rm inc})) < \alpha(\Theta(x_j))$. Next, we determine the updated value of x'. For variable $x' \in con_i$, its new value is computed based on the updated x_j to ensure that con_i remains satisfied after adjusting both variables. Let the new assignment after the change in x_j from α as α_{x_j} . For the constraint:

$$con_i : A \cdot {x'}^2 + H(i, x') \cdot x' + I(i, x') = d_i$$

with respect to x' as defined by Formula (2). We then modify the value of x' while keeping any other variables (besides x_j) fixed as constants. The set of integer roots of the equation: $A \cdot {x'}^2 + H(i,x') \cdot x' + I(i,x') - d_i = 0$ with respect to x' under the assignment α_{x_j} is computed and denoted as \mathcal{R} . Let the polynomial $\Theta(x_j,x')$ be the terms in the objective function that involve both x_j and x'. For any root $r_i \in \mathcal{R}$, let α' denote the new assignment after changing the values of x_j to x_i^{inc} and x' to r_i from the original assignment α . If:

$$\alpha'(\Theta(x_i, x')) < \alpha(\Theta(x_i, x')),$$

then there is a $move_{inc}(x_i, x', con_i, \alpha)$ such that:

$$\alpha(x_j^{\text{new}}) = \alpha(x_j^{\text{inc}}), \alpha(x'^{\text{new}}) = r_i.$$

Free Move Operator

Considering a special case where a variable $x_j \in V(f)$ has no constraints involving it. We denote x_j as *free variables*. The *free move* operator, denoted as $move_{free}(x_j,\alpha)$, is designed for such variables. Recall that, $\Theta(x_j) = W \cdot x_j^2 + K(x_j) \cdot x_j$. If $W \neq 0$, the axis of symmetry of this quadratic function is $\xi = \frac{K(x_j)}{-2W}$. It adjusts its value to minimize the objective function. Specifically, $move_{free}(x_j,\alpha)$ is:

$$\alpha(x_j^{\text{new}}) = \begin{cases} \alpha(x_j) - 1, & \text{if } W = 0, K(x_j) > 0\\ \alpha(x_j) + 1, & \text{if } W = 0, K(x_j) < 0\\ \xi, & \text{if } W > 0\\ \alpha(x_j), & \text{otherwise.} \end{cases}$$
(6)

Weighting Scheme and Score Function

In this section, we introduce techniques that help the local search algorithm perform effective search.

Weighting Scheme

The weighting scheme guides the search in a promising direction by assigning an additional property called weight (an integer) to constraints and the objective function. These weights are dynamically adjusted during the search and are used to compute the score functions. We denote $w(con_i)$ as the weight of constraint con_i and w(obj) as the weight of the objective function. Both $w(con_i)$ and w(obj) are initially set to 1 and are updated dynamically as follows:

- for violated constraint con_i , $w(con_i) := w(con_i) + 1$;
- if $obj(\alpha) > obj^*$ and $w(obj) < \zeta$, then w(obj) := w(obj) + 1, where obj^* is objective value under best found solution, ζ limits the maximum value that objective function weight can get.

Scoring Function

The scoring function is a crucial component of local search algorithms, used to select best operations. We design the scoring function based on the feasibility of the current solution. For infeasible solutions, the scoring function assesses whether there is a change in the state of the constraints and the objective function. For feasible solutions, we evaluate how significant these changes are. The score consists of two components: constraint score and objective function score. Given the weights for constraints and the objective function, the scoring function for an operation op that changes an assignment α to α' is designed as follows:

The constraint score measures the change in the total penalty of constraints. Specifically, if α is infeasible, any violated constraint under α and α' incurs a penalty of $w(con_i)$. if α is feasible, the penalty of all constraints under α is $p(con_i, \alpha) = 0$, any violated constraint under α' incurs a penalty of $p(con_i, \alpha') = w(con_i) \cdot |d_i - (\frac{1}{2}\mathbf{x}^T\mathcal{Q}^i\mathbf{x} + b^i\mathbf{x})|$.

Definition 4 (Constraint Score). *The constraint score for an operation op is the decrease of the total penalty caused by performing op :*

Constraint Score(op) =
$$\sum_{i=1}^{m} (p(con_i, \alpha) - p(con_i, \alpha'))$$

The objective function score measures the change in its value. Let Δ_{obj} as change $(obj(\alpha)-obj(\alpha'))$. If $\Delta_{obj}\neq 0$, then $\mathrm{sign}(\Delta_{obj})=\frac{\Delta_{obj}}{|\Delta_{obj}|}$, otherwise, $\mathrm{sign}(\Delta_{obj})=0$.

Definition 5 (Objective Function Score). The objective function score for an operation op is $w(obj) \cdot sign(\Delta_{obj})$ if α is infeasible, and score is $w(obj) \cdot \Delta_{obj}$ if α is feasible.

Definition 6 (Total Score). The score of an operation is the sum of the constraint score and the objective function score.

An operation is **decreasing** if its score is positive, indicating that it moves the search in a promising direction.

Local Search Algorithm

Our algorithm primarily consists of two aims: Satisfying all violated constraints and optimizing the objective function. To achieve this, we design an algorithm with a two-mode structure: the *Satisfying* mode and the *Optimization* mode, each associated with its specific operators. In the *Satisfying* mode, the algorithm focuses on making violated constraints satisfied, while in the *Optimization* mode, the algorithm focuses on minimizing the objective function.

Algorithm 1: Satisfying Mode

```
1 while ∃ violated constraints do
     if \exists decreasing quadratic satisfying move
2
       operation in violated constraints then
         op = a decreasing quadratic satisfying move
3
           operation with the greatest score via BMS;
      else
4
         update weights by Weighting Scheme;
5
         c = a random violated constraint;
6
         op = a quadratic satisfying move operation
           with the greatest score in c via BMS;
     perform op to modify \alpha;
```

Algorithm 2: Optimization Mode

```
1 while \forall constraints is satisfied do
      CandOp =
2
       {all move_{exp}, move_{inc}, move_{free} operations};
      if \exists decreasing operaton in CandOp then
3
          op = a decreasing operaton with the greatest
4
           score picked via BMS;;
      else
5
          update weights by Weighting Scheme;
6
          v = a \text{ random variable } \in V(f);
7
          op = an operaton for v with the greatest score
           in CandOp via BMS;
     perform op to modify \alpha;
```

The routine of our algorithm is described in Algorithm 3. First, the best assignment of best solution α^* and the current assignment α are initialized (Line 1), with all variables set to 0. Then, the algorithm iteratively modifies α by performing operations on integer variables. The algorithm dynamically switches between two modes based on whether there are any violated constraints (Line 2–8).

If there exist violated constraints, then our algorithm enters Satisfying mode. As described in Algorithm 1, the algorithm first tries to find a decreasing quadratic satisfying move operation with the greatest score via BMS heuristic (Cai 2015) (Line 2–3). Specifically, the BMS heuristic samples t operations (t is a parameter), and selects the decreasing one with the greatest score. If no decreasing operation is found, it indicates that the algorithm may reach a local optimum. The algorithm further tries to escape from the local optimum by updating the weights of the constraints and the objective function based on the weighting scheme. Then it randomly selects one of the violated constraints and chooses a quadratic satisfying move operation with the greatest score via BMS (Line 4–7).

Otherwise, our algorithm enters the Optimization mode. In this mode, the algorithm focuses on minimizing the objective function. The Optimization mode is described in Algorithm 2. In each iteration, all inequality exploration move operations for inequality constraints, equality incremental

Algorithm 3: Local Search Algorithm

```
Input: IQP instance F, cutoff time cutoff
  Output: Best found solution obj(\alpha^*) or NA
1 \alpha^* := \emptyset, \alpha := an initial complete assignment;
  while time not exceeds cuttoff do
       if \alpha is feasible and obj(\alpha) < obj^* then
3
        \alpha^* := \alpha, \ obj^* := obj(\alpha);
4
       if \exists violated constraints then
5
           Enter Satisfying Mode and perform
6
             corresponding operation;
       else
7
           Enter Optimization Mode and perform
8
            corresponding operations;
9 if \alpha^* is feasible then return (obj^*, \alpha^*);
10 else return NA:
```

move operations for equality constraints, and free move operations for free variables are added to a candidate set CandOp (Line 2). The algorithm first tries to find a decreasing operation in CandOp with the greatest score via the BMS heuristic (Line 3–4). If it fails to find any decreasing operation, the weighting scheme is activated. Then it selects a random variable $v \in V(f)$ and chooses an operation for v in CandOp with the greatest score (Line 5–8).

Experiments

In this section, we evaluate the performance of our solver, LS-IQCQP, against state-of-the-art IQP solvers on standard IQP benchmarks. Ablation experiments are conducted to assess the effectiveness of the proposed strategies. We also conduct experiments to verify the stability of our solver. Notably, LS-IQCQP has set new records for 6 open instances.

Experiment Implementation and Setup:

LS-IQCQP is programmed in C++, compiled by g++ with '-O3' option. All experiments are carried out on a server with AMD EPYC 9654 CPU and 2048G RAM under the system Ubuntu 20.04.4. There are 2 parameters in the solver: t the number of samples for the BMS heuristic, ζ for the Upper Limit of objective function weight. The parameters are tuned according to our preliminary experiments and suggestions from the literature, and are set as follows: t=100. $\zeta=100$.

Competitors

We compare LS-IQCQP with 4 state-of-the-art IQP solvers. The binaries for each competitor are downloaded from their respective websites and are run with default settings.

- **Gurobi** (Gurobi Optimization 2022): The most powerful commercial IQP solvers. We use both its exact and heuristic versions, denoted by **Gurobi_exact** and **Gurobi_heur**, respectively (version 10.0.0).
- **SCIP** (Achterberg 2009): One of the fastest academic solvers for IQP (version 8.1.0).
- Cplex (Nickel et al. 2022): A famous commercial IQP solver to solve IQP models (version 22.1.0).

 Knitro (Byrd, Nocedal, and Waltz 2006): A powerful mixed-integer nonlinear programming solver renowned for its advanced metaheuristics and local search algorithms (version 14.1).

These solvers showcase different characteristics for IQP solving: **Gurobi** is the best-performing IQP solver and **ranks first** in IQP competitions, as noted in (Mittelmann 2023). **SCIP** is the best open-source academic solver for IQP problems, also highlighted in (Mittelmann 2023). **Cplex**, although not participating in the IQP competition, is widely used and known for its strong performance in complex IQP models. Lastly, **Knitro** is the only solver we found that is both free for academic use and employs local search algorithms, while being capable of solving any type of IQP.

Benchmarks

Experiments are carried out with 2 benchmarks of the mainstream dataset for IQP.

- **QPLIB** (Furini et al. 2019): a standard library of quadratic programming instances. We select 137 instances where all variables are integers.
- MINLPLIB (Bussieck, Drud, and Meeraus 2003): A library of mixed-integer and continuous nonlinear programming instances. We include 84 quadratic programming instances with integer variables, excluding those already present in QPLIB.

For each benchmark, instances are further categorized into 4 types: 1.QUBO: No constraints, quadratic objectives. 2.LCQP: Linear constraints only, quadratic objectives. 3.QCLP: Quadratic constraints, linear objectives. 4.QCQP: Quadratic constraints and objectives.

Evaluation Metrics

There are 3 Metrics for comparison in experiment. 1. Winning instances: denoted as #win, the number of instances where a solver finds the best solution among all solutions output by tested solvers. 2. Feasible instances: denoted as #feas, the number of instances where a solver can find a feasible solution within this time limit. 3. Solving time: the runtime comparison between LS-IQCQP and competitor solvers if both solvers can find a solution with the same value of the objective function. For each instance, each solver is run on a single thread with time limits of 10, 60, and 300 seconds, as referenced in (Lin et al. 2023). In each table, the best performance for each metric under each time limit is highlighted in bold. Additionally, the number of instances in each benchmark is denoted as #inst.

Comparison with State-of-the-Art IOP Solvers

The ability to find feasible(#feas) solutions: As shown in Table 1. LS-IQCQP performs best on both QPLIB and MINLPLIB benchmarks. It consistently finds feasible solutions for all instances in these benchmarks within 10 seconds or more. This result confirms the capability of LS-IQCQP to obtain feasible solutions within reasonable time limits.

The ability to find high-quality(#win) solutions: As shown in Table 1. LS-IQCQP consistently leads other

solvers in total #win across all time limits, demonstrating its competitive performance. In the QPLIB dataset, LS-IQCQP stands out particularly in the QUBO, QCLP and QCQP categories, showcasing its advantage in handling specific types of problems. In the MINLPLIB dataset, LS-IQCQP also performs exceptionally well in the QCLP and QCQP categories. LS-IQCQP's stable and robust performance across various problem types and time limits demonstrates its potential as an efficient solver. It is worth mentioning that LS-IQCQP outperforms Knitro in all settings, indicating a significant improvement in the field of local search solver for IQP.

Solving time analysis: The solving time results are presented in Figure 1. We compare LS-IQCQP with the topperforming solvers, Gurob_exact and Gurobi_heur, using a long 300-second time limit. It can be observed that, for instances where the same solution is found, LS-IQCQP consistently obtains solutions in a shorter time. This demonstrates the rapid convergence speed of our solver.

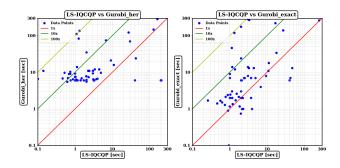


Figure 1: Running Time Comparision

New Records to Open Instances

In the QPLIB and MINLPLIB datasets, there are instances that remain open, indicating that the optimal solution has not yet been found. The current best-known solutions for each open instance are available on the respective QPLIB and MINLPLIB websites. These open instances represent some of the most challenging IQP problems to solve. **Remarkably, LS-IQCQP has established new best-known solutions for 6 open instances.** As shown in Table 2, these instances also include different types of IQP from QPLIB and MINLPLIB, demonstrating the powerful solving ability.

Effectiveness of Proposed Strategies

To evaluate the effectiveness of our proposed strategies, we develop three modified versions of LS-IQCQP. Experiments are conducted with a time limit of 300 seconds. The modifications and corresponding versions are as follows:

- Remove Inequality exploration move: version v_{no_exp}
- Remove Equality incremental move: version v_{no_inc}
- Remove Free move: version v_{no_free}

The results of ablation experiment are presented in Table 3 and confirm the effectiveness of the proposed strategies.

Part	Benchmark	Category	#inst	Gurobi_exact Gurobi_heur		SCIP		Cplex		Knitro		LS-IQCQP			
QPLIB LCQP 99 99 56 99 73 87 23 99 30 80 10 99 74	Deficilitation			#feas	#win	#feas	#win	#feas	#win	#feas	#win	#feas	#win	#feas	#win
QPLIB															
OCLP 10 10 1 10 2 10 1 9 2 10 2 10 7	QPLIB	QUBO	23			23			1	23		23		23	18
MINLPLIB CQP 5 5 0 5 1 5 0 4 0 5 0 19 0 19 9 9 10 10 1		LCQP	99			99						80			
MINLPLIB CQP 52 52 46 52 51 46 45 52 45 45 21 52 51 51 62 64 65 64 65 64 65 64 65 64 65 64 65 64 65 64 65 64 65 65		QCLP					2				2				7
MINLPLIB LCQP 52 52 46 52 51 46 45 52 45 45 45 21 52 51							- 1 <u>1</u> -				2 -			- 3 -	
OCLP 2 2 2 2 2 2 2 2 2	MAN DI ID	LCOP		19 52		19 52				19	2 15			19	9 51
OCQP 11 11 11 11 11 11 11	MINLPLIB	OCLD													31
Total 221 221 128 221 165 203 85 220 95 195 46 221 177		OCOP	11			11				11	11				11
OUBO 23 23 12 23 9 23 2 23 5 23 0 23 19	Total	QCQI													
QPLIB QUBO 23 23 12 23 9 23 2 23 5 23 0 23 19 QPLIB LCQP 99 99 64 99 74 98 26 99 37 87 12 99 74 QCLP 10 10 1 10 5 10 1 10 3 10 3 10 3 10 3 10 4 3 10 4 QCQP 5 5 1 5 1 5 1 5 0 5 0 5 5 5 0 5 MINLPLIB LCQP 52 52 46 52 52 52 45 52 45 45 21 52 51 QCLP 2 2 2 2 2 2 2 2 2															
QPLIB \$\text{LCQP}\$ 99 64 99 74 98 26 99 37 87 12 99 74 QCLP 10 10 1 10 5 10 1 10 3 10 3 10 4 QCQP 5 5 1 5 1 5 0 5 0 5 0 5 5 5 5 5 5 5 0 5 0 5 0 5 5 5 5 5 5 5 5 4 6 52 52 45 52 45 45 21 52 51 2 <		OUBO	23	23	12.	23		23		23	- 5	23	0	23	19
QCLP 10 10 1 10 5 10 1 10 3 10 3 10 4	QPLIB		99	99		99		98	26	99	37	87			
MINLPLIB QUBO 19 19 10 19 14 19 2 19 3 19 0 19 10 MINLPLIB LCQP 52 52 46 52 52 52 45 52 45 45 21 52 51 QCLP 2 2 2 2 2 2 2 2 2		ÖCLP	10	10	1	10		10 10	1	10	3			10	4
MINLPLIB QUBO 19 19 10 19 14 19 2 19 3 19 0 19 10 MINLPLIB LCQP 52 52 46 52 52 52 45 52 45 45 21 52 51 QCLP 2 2 2 2 2 2 2 2 2		OCOP	5		1	5	1		0	5		5	0	5	5
QCLP 2 2 2 2 2 2 2 2 2		QUBO -	T9 T			⁻ 19 ⁻		⁻ 19 ⁻	2	19					10
OCLP 2 2 2 2 2 2 2 2 2		LCQP													51
Total 221 221 147 221 168 220 89 221 106 202 49 221 176 Time Limit 300 Seconds		QCLP													2
Time Limit 300 Seconds QUBO 23 23 13 23 9 23 5 23 9 23 0 23 20		QCQP													11
QPLIB QUBO LCQP 23 by 13 by 13 by 14 by 15 by 15 by 16 by	Total		221	221	147					221	106	202	49	221	176
QPLIB ÎCQP QCLP 99 10 73 10 99 10 78 10 99 6 30 10 99 10 42 10 88 10 15 5 99 10 74 10 6 6 5 10 1 5 10 5 5 10 6 5 5 5 0 5 5 5 0 5 5 5 0 5 5 5 5 5 5 5 5 5 99 5 74 5 70 70 10 5 5 5 5 5 5 70 5 5 5 5 5 70 70 19 99 5 74 5 5 5 5 5 5 70 70 19 99 5 5 5 5 5 5 7 70 70 19 99 5 5 5 5 7 70 70 19 99 5 7 74 7 70 10 6 5 7 5 7 10 8 7 10 6 7 10 6 7 99 7 74 7 99 7 74 7 99 7 74 7 99 7 74 7 10 99 7 42 7 88 8 15 8 99 8 74 8 10 6 8 10 6 8 10 6 8 10 6 8 10															
MINLPLIB QCLP 10 10 2 10 6 10 1 10 3 10 5 10 6 5 5 1 10 15 10 15 10 10		QUBO	23		13	23	9	23	5	23	9	23	0	23	20
MINLPLIB QCQP 5 5 1 5 3 5 0 5 0 5 0 5 5 19 0 19 0 11 11 11 11	QPLIB	LCQP		99		99			30	99					74
MINLPLIB QUBO 19 19 11 19 15 19 5 19 8 19 0 19 9 15 15 10 15 10 15 10 15 10 15 10 10		QCLP			2		0		1	10				10	ō
MINLPLIB LCQP 52 52 46 52 52 45 52 45 46 21 52 51 20 20 20 20 20 20 20 2					<u>1</u> -	10 -				- 70				- 30 -	- 5
QCLP 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	MINLPLIB	LCOB													9 51
OCOP 11 11 11 11 11 11 11		OCLP	32						43	32	43		21		31
Total 221 221 159 221 176 221 99 221 120 203 54 221 178		OCOP													11
	Total	γυζι	221	221	159	221	176	221	99	221	120	203	54	221	178

Table 1: Comparison of the number of #win and #feas instances with state-of-the-art IQP solvers

Instance Name	Benchmark	Category	#var	#cons	Objective	Previous best	LS-IQCQP
QPLIB_2036	QPLIB	QCQP	324	324	Minimize	-30590	-30660
QPLIB_2096	QPLIB	QCQP	300	6925	Minimize	7068000	7064664
chimera_mgw-c16-2031-01	MINLPLIB	QUBO	2032	0	Maximize	1993	2006
chimera_mgw-c16-2031-02	MINLPLIB	QUBO	2032	0	Maximize	1996	2004
chimera_selby-c16-01	MINLPLIB	QUBO	2031	0	Maximize	739.4	741.1
chimera_selby-c16-02	MINLPLIB	QUВО	2031	0	Maximize	733.9	738.1

Table 2: LS-IQCQP sets new records for 6 open instances. #var and #cons denote the number of variables and constraints, respectively. Minimize refers to a minimization problem, while Maximize refers to a maximization problem.

Benchmark	v_{no}	exp	v_{no}	inc	v_{no_free}		
	#better	#loss	#better	#loss	#better	#loss	
QPLIB MINLPLIB	65 45	2 5	40 9	5 1	23 19	4 2	
Total	110	7	49	6	42	$\overline{6}$	

Table 3: Comparing LS-IQCQP with modified versions. The number of instances where LS-IQCQP finds better(#better) solutions and worse(#loss) solutions are presented.

Stability Experiments

Repetitive running with randomness To examine the stability of LS-IQCQP, we execute the algorithm 10 times using seeds from 1 to 10 for each instance. Experimental results show that over 80% of instances have a coefficient of variation less than 0.1, demonstrating that LS-IQCQP exhibits stable performance.

Sensitivity analysis on the parameters To examine the sensitivity of parameters in LS-IQCQP, we test the algorithm with various parameter settings. Results are in Appendix C and show that these variations do not significantly affect the algorithm's overall performance.

Conclusion and Future Work

In this paper, we focus on local search algorithms for IQP. We propose 4 novel operators and a two-mode algorithm with new scoring functions to enhance the search process. Experiments show that the performance of our solver is outstanding, setting 6 new records for open instances. Future work aims to combine our solver with other solvers to prune the search space and speed up the search process.

References

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1: 1–41.

- Anstreicher, K. M. 2003. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97: 27–42.
- Axehill, D. 2008. *Integer quadratic programming for control and communication*. Ph.D. thesis, Institutionen för systemteknik.
- Bomze, I. M.; Budinich, M.; Pardalos, P. M.; and Pelillo, M. 1999. The maximum clique problem. *Handbook of Combinatorial Optimization: Supplement Volume A*, 1–74.
- Boros, E.; Hammer, P. L.; and Tavares, G. 2007. Local search heuristics for quadratic unconstrained binary optimization (QUBO). *Journal of Heuristics*, 13: 99–132.
- Burer, S.; and Vandenbussche, D. 2005. Semidefinite-Based Branch-and-Bound for Nonconvex Quadratic Programming. Bussieck, M. R.; Drud, A. S.; and Meeraus, A. 2003. MINLPLib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1): 114–119.
- Byrd, R. H.; Nocedal, J.; and Waltz, R. A. 2006. K nitro: An integrated package for nonlinear optimization. *Large-scale nonlinear optimization*, 35–59.
- Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 747–753.
- Deng, Z.; Bai, Y.; Fang, S.-C.; Tian, Y.; and Xing, W. 2013. A branch-and-cut approach to portfolio selection with marginal risk control in a linear conic programming framework. *Journal of systems science and systems engineering*, 22(4): 385–400.
- Drud, A. 1985. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical programming*, 31: 153–191.
- Fletcher, R. 2000. Stable reduced Hessian updates for indefinite quadratic programming. *Mathematical programming*, 87: 251–264.
- Frangioni, A.; Furini, F.; and Gentile, C. 2016. Approximated perspective relaxations: a project and lift approach. *Computational Optimization and Applications*, 63: 705–735.
- Frangioni, A.; Galli, L.; and Scutellà, M. G. 2015. Delay-constrained shortest paths: approximation algorithms and second-order cone models. *Journal of Optimization Theory and Applications*, 164(3): 1051–1077.
- Frangioni, A.; Galli, L.; and Stea, G. 2017. Delay-constrained routing problems: Accurate scheduling models and admission control. *Computers & Operations Research*, 81: 67–77.
- Furini, F.; Traversi, E.; Belotti, P.; Frangioni, A.; Gleixner, A.; Gould, N.; Liberti, L.; Lodi, A.; Misener, R.; Mittelmann, H.; et al. 2019. QPLIB: a library of quadratic programming instances. *Mathematical Programming Computation*, 11: 237–265.
- Gill, P. E.; Murray, W.; and Saunders, M. A. 2005. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM review*, 47(1): 99–131.

- Gill, P. E.; and Wong, E. 2015. Methods for convex and general quadratic programming. *Mathematical programming computation*, 7(1): 71–112.
- Goldberg, P. W. 2004. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *Proceedings of the twenty-third annual ACM symposium on principles of distributed computing*, 131–140.
- Gould, N. I.; Orban, D.; and Robinson, D. P. 2013. Trajectory-following methods for large-scale degenerate convex quadratic programming. *Mathematical Programming Computation*, 5: 113–142.
- Gould, N. I.; and Robinson, D. P. 2017. A dual gradient-projection method for large-scale strictly convex quadratic problems. *Computational Optimization and Applications*, 67(1): 1–38.
- Gu, J. 1992. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bulletin*, 3(1): 8–12.
- Gurobi Optimization, L. 2022. Gurobi Optimizer Reference Manual (Gurobi Optimization, LLC).
- Land, A. H.; and Doig, A. G. 2010. An automatic method for solving discrete programming problems. Springer.
- Lei, Z.; Cai, S.; Luo, C.; and Hoos, H. 2021. Efficient local search for pseudo Boolean optimization. In *Theory and Applications of Satisfiability Testing—SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*, 332–348. Springer.
- Lin, P.; Cai, S.; Zou, M.; and Lin, J. 2023. New Characterizations and Efficient Local Search for General Integer Linear Programming. *arXiv preprint arXiv:2305.00188*.
- Lun, D. S.; Rockwell, G.; Guido, N. J.; Baym, M.; Kelner, J. A.; Berger, B.; Galagan, J. E.; and Church, G. M. 2009. Large-scale identification of genetic design strategies using local search. *molecular systems biology*, 5(1): 296.
- Merz, P.; and Katayama, K. 2004. Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems*, 78(1-3): 99–118.
- Mittelmann, H. 2023. Visualizations of Mittelmann benchmarks.
- Nickel, S.; Steinhardt, C.; Schlenker, H.; and Burkart, W. 2022. *Decision Optimization with IBM ILOG CPLEX Optimization Studio*. Springer.
- Pia, A. D.; Dey, S. S.; and Molinaro, M. 2017. Mixed-integer quadratic programming is in NP. *Mathematical Programming*, 162: 225–240.
- Vavasis, S. A. 1990. Quadratic programming is in NP. *Information Processing Letters*, 36(2): 73–77.
- Wächter, A.; and Biegler, L. T. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106: 25–57.