New Solution Approaches for the Quadratic Assignment Problem

Franklin Djeumou Fomeni
University of the Witwatersrand, Johannesburg
School of Computational and Applied Mathematics
Supervised by: Professor Montaz Ali, University of the Witwatersrand, South Africa

September 12, 2011

Submitted in partial fulfilment of Masters of Sciences

Declaration

I, the undersigned, hereby declare that the work contained in this dissertation is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly. It is being submitted for the degree of Masters of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.

Franklin Djeumou Fomeni

September 12, 2011

Acknowledgements

I acknowledge the almighty God for the strength He gives me and for His love.
I am very thankful towards my supervisor Professor Montaz Ali who drew my attention to this area of optimization, for his encouragement and his guidance throughout this work. I thank both him and his visitor Dr Elzain of the university of Khartoum for suggesting the way of generating initial strictly feasible solution for the interior point method reported in this dissertation.
My sincere gratitude goes to the African Institute for Mathematical Sciences (AIMS) and the School of Computational and Applied Mathematics (CAM) for their financial support without which this dissertation would not have been possible.
I am indebted to all the staff members of the School of Computational and Applied Mathematics of the University of the Witwatersrand, for their friendship and assistance, and particularly Prof Momoniat, Prof Banda, Dr Moitsheki, Mrs B. Pickering and Mrs D. Bowes for their supports and advices.
I gratefully acknowledge CAM's higher degree students for their warm friendship and collaboration, especially, Naval, Charles, Morgan, Guo-Dong, Innocent, Tumelo, Obakeng, Asha, Dario, Sima, Elimboto, Viren, Terry, Tanya, Gideon, Byron and Rahab.
I am extremely grateful to my former lecturers at the University of Yaounde I in Cameroon, especially Prof F. Wamon, Dr B. Tchapnda , Dr G. M. Mbakop, Dr Y. Emvudu for the strong mathematical background they gave me, and for their continual advices.
To my family relatives, my dear mum Clementine Tientcheu, my dad Barthelemy Tayou, the family Bakam, the family Tayo, Hugue, Chritel, Edith, and my friends, Ludovic, Lydienne, Jeanne, Alain Julio, Silver, Diane, Chrystelle, Bertin, Bruno, Thierry, Daniel, Billy, Frank, Fleur, Ronald, Danielle, Alain, Francine, Arnaud, Tafadzwa and Stephane, I say thanks for their unconditional encouragement and love. Thanks to St Francis-Xavier Martindale, Holy Trinity, Braamfontein Catholic churches for the spiritual support.
Finally, I thank the Family Ketcha for encouraging and advising me.

Abstract

A vast array of important practical problems, in many different fields, can be modelled and solved as quadratic assignment problems (QAP). This includes problems such as university campus layout, forest management, assignment of runners in a relay team, parallel and distributed computing, etc. The QAP is a difficult combinatorial optimization problem and solving QAP instances of size greater than 22 within a reasonable amount of time is still challenging. In this dissertation, we propose two new solution approaches to the QAP, namely, a *Branch-and-Bound* method and a discrete dynamic convexized method. These two methods use the standard quadratic integer programming formulation of the QAP. We also present a lower bounding technique for the QAP based on an equivalent separable convex quadratic formulation of the QAP. We finally develop two different new techniques for finding initial strictly feasible points for the interior point method used in the *Branch-and-Bound* method. Numerical results are presented showing the robustness of both methods.

Symbols nomenclature

- QAP: Quadratic assignment problem.
- $RSQIP$: Reformulated standard quadratic integer programming formulation of the QAP.
- $SQIP$: Standard quadratic integer programming formulation of the QAP.
- $SCQIP$: Separable convex quadratic integer programming formulation of the QAP.
- $CQIP$: Convex quadratic integer programming formulation of the QAP.
- $MILP$: Mixed integer linear programming formulation of the QAP.
- QIP: Quadratic integer programming formulation of the QAP.
- TF: Trace formulation of the QAP.
- KF: Kronecker formulation of the QAP.
- $CR1$: Continuous relaxation of ($RSQIP$).
- $CR2$: Continuous relaxation of ($SCQIP$).
- BL: Barrier logarithmic problem.
- $ANLIP$: Auxiliary non-linear integer programming problem from ($RSQIP$).
- GA: Genetic algorithm.
- ACO: Ant colony optimization.
- SA: Simulated annealing.
- TS: Tabu search.
- DDC: Discrete dynamic convexized.

Contents

List of Figures

List of Tables

1. Introduction

Optimization is an important tool in decision science and in analysis of physical systems. It has application in all branches of Science, Engineering and Management. In nature, physical systems tend to state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electron is minimized. Rays of light follow paths that minimize their travel time. Manufacturers aim for maximum efficiency in the design and operation of their production processes. Airline companies schedule crews and aircraft to minimize cost. Investors seek to create portfolios that avoid excessive risks while achieving a high rate of return.

To use optimization, one first needs to identify some objective, which is a quantitative measure of the performance of the system under consideration. This objective could be, the cost, the profit, the time, the potential energy or any quantity or combination of quantities that can be represented by a number. The objective depends on certain characteristics of the system, called variables. In an optimization problem, one's goal is to find the values of the variables that optimize the objective. In most of the cases, There are constraints in the problem that restrict the values of the variables. The feasible set of the problem is determined by the constraints. Optimization is divided into continuous, discrete and mixed integer programming. In the continuous optimization, the variables used in the objective can assume real values. Continuous optimization comprises linear programming and non-linear programming. A continuous optimization is convex if the objective is a convex[1] function of the variables and the feasible set is also convex.

As opposed to continuous optimization, the variables used in the discrete optimization are restricted to assume only discrete values, such as the integers. There are two notable branches of discrete optimization, namely, combinatorial optimization, in which the feasible set is a finite subset of the set of all the integer numbers, and integer programming in

[1]Details and characteristics of a convex function are given in Chapter 2

which variable are simply constrained to assume only integer values. Discrete optimization problems can also be formulated as a linear, non-linear, convex or non-convex optimization problem. Mixed integer optimization combines both real and integer variables. This dissertation deals with a combinatorial optimization problem, namely, the quadratic assignment problem.

1.1 Mathematical formulation of the optimization problem

Mathematically speaking, optimization is concerned with finding the maxima or minima of a function subject to restrictions on its variables. An optimization problem is a problem of the form

$$\min_{x \in S} \qquad f(x), \tag{1.1}$$

where x stands for the variables, S is the feasible set or the feasible set, and $f : S \subseteq \mathbb{R}^n \longrightarrow Y$ with $Y \subseteq \mathbb{R}$ is the objective function. The feasible set is defined by $S = \{x \in \mathbb{R}^n : g(x) \geq 0, h(x) = 0\}$, where $g(x), h(x)$ are also real-valued funtions.

If the search space S is continuous, then we have a continuous optimization problem, if it is discrete, then we have a discrete optimization problem.

Definition 1.1.1 (Feasible and strictly feasible points)**:**
A point $x \in \mathbb{R}^n$ is said to be feasible for (1.1) if $x \in S$.

If the interior of S is non empty, x is said to be strictly feasible for (1.1) if x is in the interior of S.

Definition 1.1.2 (Local minimum, local maximum)**:**
A point $x^* \in S$ is said to be a local minimum (local maximum) of f if $f(x^*) \leq f(x)$ $(f(x^*) \geq f(x))$ for all x in a neighbourhood of x^*. In other words, this means that there exists $\varepsilon > 0$ such that $f(x^*) \leq f(x)$ $(f(x^*) \geq f(x))$ and $\|x^* - x\| < \varepsilon$, $x \in S$.

Definition 1.1.3 (Global minimum, global maximum)**:**
A point x^* is said to be a global minimum (global maximum) if $f(x^*) \leq f(x)$ $(f(x^*) \geq f(x))$ for all $x \in S$.

Definition 1.1.4 (Optima)**:**
A point $x^* \in S$ is called an optimizer of f if x^* is a minimizer or a maximizer of f.

Figure 1.1 gives a graphical illustration of local and global optima. In this figure, the points (A) and (B) are global maximum and minimum respectively. While the points (a) and (b) are local maximum and minimum respectively. The global maximum (Minimum) is also a local maximum (Minimum).

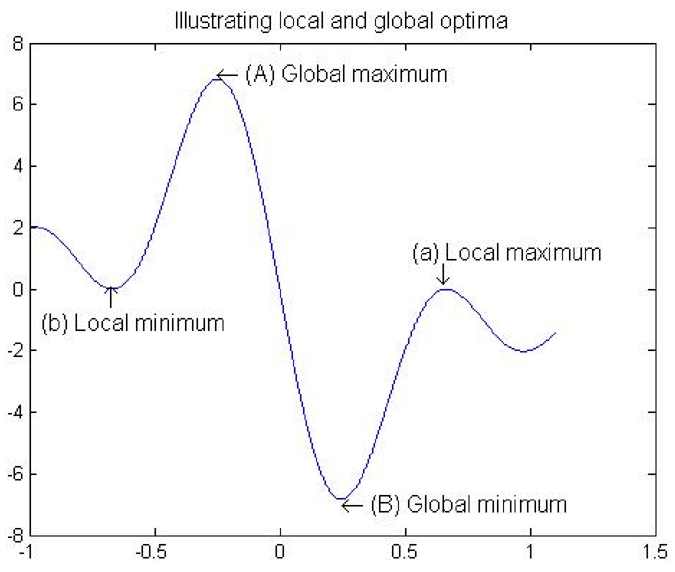


Figure 1.1: Local optima vs global optima

1.2 The Quadratic Assignment Problem

The QAP is one of the fundamental combinatorial optimization problems. It was introduced in 1957 by Koopmans and Beckmann [KB57] as a mathematical model for the location of a set of indivisible economical activities. The QAP considers the problem of allocating a set of n facilities to a set of n locations, with the cost being a function of the distance and flow between facilities, plus costs associated with a facility being placed at a certain location. The formal definition of the problem is as follows. Let n be the number of facilities and locations, $F = (f_{ij})_{1\leq i,j\leq n}, D = (d_{kl})_{1\leq k,l\leq n}$ and $B = (b_{ik})_{1\leq i,k\leq n}$ be three $n \times n$ matrices, where f_{ij} is the flow between the facilities i and j, d_{kl}, the distance between the locations k and l, and b_{ik} the cost of the facility i being placed at the location k. Let $\mathcal{S}_n$ be the set of all the permutations $\phi : \{1, \dots, n\} \longrightarrow \{1, \dots, n\}$, see **Definition 2.2.15** in Chapter 2. The QAP was originally defined as follows:

$$\min_{\phi\in\mathcal{S}_n} \qquad \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}d_{\phi(i)\phi(j)} + \sum_{i=1}^{n} b_{i\phi(i)}. \tag{1.2}$$

The term $b_{i\phi(i)}$ in (1.2) is the cost associated with placing facility i at location $\phi(i)$. On the other hand, the product $f_{ij}d_{\phi(i)\phi(j)}$ represents the cost of placing facility j at location $\phi(j)$ while facility i is placed at location $\phi(i)$.

A lot of interests have been given to the QAP since its introduction in 1957. The QAP has been used as the mathematical model of many real life problems arising in facility location, computer manufacturing, scheduling, building layout design, process communications, etc. Indeed, in Chapter 3 we review some of the direct applications of the QAP in real life. Solving the QAP using exact solution methods is still a challenge. In fact, many problem instances of size $n > 20$ are still found difficult to be solved with exact algorithm within reasonable computational time. Sahni and Gonzalez [SG76] have shown that the QAP is NP-hard and that even finding an approximate solution within some constant factor from the optimal solution cannot be done in polynomial time unless P=NP.

1.3 Methods for solving the QAP

The different methods used to achieve a global optimum for the QAP include *Branch-and-Bound*, cutting planes, *Branch-and-Cut*, and semi-definite programming (SDP). However, the aim of this dissertation is to develop new solution approaches for the QAP. We identify a mathematical formulation of the QAP, which has not been studied in the literature yet, for which we develop two solution methods. Our first solution method is a *Branch-and-Bound* method which is a systematic enumerative scheme that uses lower bounds to eliminate undesired solutions.

Our second solution method is an auxiliary function-based dynamic convexized method. This method consists of building a sequence of minimizers, using a local search algorithm, for the QAP which converges towards the optimum solution of the problem. The dynamic convexized method incorporates some mechanisms that allow it to escape from local minima. It has recently been proposed by Zhu and Ali [ZA09] for solving the general non-linearly constrained non-linear integer programming, and has never been applied to the QAP. This dissertation offers the first application of this method to the QAP. Central to the application of dynamic convexzied method to the QAP is a neighbourhood structure of the QAP that we introduce. This makes the dynamic convexzied method different from the one proposed by Zhu and Ali [ZA09].

In order to improve the efficiency and speed up the convergence of the two solution methods, we have proposed a heuristic random enumerative scheme to identify an initial solution with which to start the new methods proposed in this dissertation. Another major contribution in this dissertation is the identification of a strictly feasible solution of the quadratic formulation of the QAP. We have developed two different techniques for finding an initial strictly feasible point for the interior point method, since we used an interior point algorithm for our lower bounding technique in the *Branch-and-Bound*. These two techniques can consequently be applied to other interior point algorithms.

1.4 Structure of the dissertation

The rest of this dissertation is organized as follows:

- In Chapter 2, we present the mathematical background necessary for the understanding of this dissertation.
- In Chapter 3, we review the previous developments in the area of the QAP. This review comprises the different mathematical formulations of the QAP, its applications to real life problems, the existing lower bounding techniques and the existing solution methods.
- Chapter 4, we present a standard quadratic integer programming $(RSQIP)$ formulation of the QAP. This is a reformulation of the standard quadratic integer programming $(SQIP)$ formulation suggested in [BcPP98]. We then transform this standard quadratic integer programming reformulation, $(RSQIP)$, into a separable convex quadratic integer programming $(SCQIP)$ problem using a decomposition technique. We study $(RSQIP)$ and $(SCQIP)$ in this dissertation. Two lower bounding techniques based on these two problem formulations are also presented in this chapter.
- In Chapter 5, we develop two different techniques for finding an initial strictly feasible point for the interior point method.
- In Chapter 6, we present in detail the two new solution approaches proposed in this dissertation together with the step-by-step descriptions of the corresponding algorithms.
- In Chapter 7, the details of the numerical experiments and a full set of numerical results from the two solution methods are presented.
- Chapter 8 presents concluding remarks and possible future research.

2. Mathematical background

Since the introduction of the QAP, researchers have been using mathematical theories to advance the reformulations and approximations of the QAP. Reformulations and approximations are used to design algorithms for the QAP. In this dissertation, we develop some mathematical theories and suggest new algorithms for solving the QAP. This chapter presents some mathematical tools useful for an easy understanding of various reformulations and approximation of the QAP. We begin with some basic notations used throughout the dissertation: $\mathbb{R}^n$, $\mathbb{Z}^n$ and $\mathbb{Q}^n$ denote the set of all real, integer and rational n-vectors, respectively; $\mathbb{R}^{m\times n}$, $\mathbb{Z}^{m\times n}$ and $\mathbb{Q}^{m\times n}$ denote the set of all real , integer and rational $m \times n$ matrices, respectively. The set of all positive integers is also denoted by $\mathbb{Z}_+$ and the set of all rational numbers by $\mathbb{Q}$. The upper script T to any vector or matrix stands for its transpose.

2.1 The quadratic form

Definition 2.1.1 (Bilinear function)**:**
A function

$$\begin{aligned} f :& E \times F \longrightarrow G, \qquad \textit{i.e.} \\ &(x, y) \longmapsto f(x, y), \end{aligned}$$

$E \subseteq \mathbb{R}^m$, $F \subseteq \mathbb{R}^n$ and $G \subseteq \mathbb{R}^q$, $m, n, q \in \mathbb{Z}_+$, is called a bilinear function if it is linear in each of its variables i.e for all $x, x_1, x_2 \in E$, $\quad y, y_1, y_2 \in F$ and $a, b \in \mathbb{R}$ we have:

- $f(ax_1 + bx_2, y) = af(x_1, y) + bf(x_2, y)$,
- $f(x, ay_1 + by_2) = af(x, y_1) + bf(x, y_2)$.

In addition, if $E = F$, then f is said to be symmetric if $f(x, y) = f(y, x)$ for all $x, y \in E$.

Definition 2.1.2 (Bilinear form)**:**
A bilinear function f is said to be a bilinear form if G is reduced to $\mathbb{R}$ i.e.

$$f : E \times F \longrightarrow \mathbb{R}.$$

Definition 2.1.3 (Quadratic form)**:**
A quadratic form over E is a function

$$\begin{aligned} q : E &\longrightarrow \mathbb{R}, \qquad i.e. \\ x &\longmapsto q(x) \end{aligned}$$

such that $q(x) = f(x, x)$, where $f(\bullet, \bullet)$ is a symmetric bilinear form over E; f is called the associated bilinear form of q.

Given a quadratic form q, its associated bilinear form f can always be retrieved from q as follows:

$$f(x, y) = \frac{1}{2}\left(q(x + y) - q(x) - q(y)\right).$$

Any quadratic form q is fully defined by its matrix M i.e.

$$q(x) = \frac{1}{2} x^T M x.$$

Definition 2.1.4 (Positive semi-definiteness and positive definiteness)**:**
A matrix M is said to be positive semi-definite if:

$$x^T M x \geq 0, \forall x \in \mathbb{R}^m,$$

it is said to be positive definite if

$$x^T M x > 0, \forall x \in \mathbb{R}^m \setminus \{0\}.$$

Definition 2.1.5 (Convex function)**:**
Let g be a real valued function defined over $E \subseteq \mathbb{R}^n$. The function g is said to be a convex function if it satisfies:

$$g(tx + (1 - t)y) \leq tg(x) + (1 - t)g(y)$$

for all $x, y \in E$ and $t \in [0,1]$, it is said to be strictly convex if

$$g(tx + (1-t)y) < tg(x) + (1-t)g(y)$$

for all $t \in (0,1)$.

It is well known [Cot67] that a quadratic form $q(x) = x^T Mx$ is convex if its matrix M is positive semi-definite, it is strictly convex if M is positive definite.

2.2 Matrix analysis

Definition 2.2.1 (Rational elementary row and column operations)**:**

For a rational matrix, the rational elementary row or column operations are:

i) Interchanging two rows or two columns.

ii) Multiplying a row or a column by a non-zero rational number.

iii) Adding a rational multiple of one row (or one column) to another row (or column).

Theorem 2.2.2:
Let M be an $n \times n$ rational symmetric matrix such that a zeros pivot is never encountered when applying Gaussian elimination with type iii) operations, then there exists a non-singular rational matrix P such that

$$P^T MP = diag(\bar{d}_1, \ldots, \bar{d}_n). \tag{2.1}$$

Proof. Let M be an $n \times n$ rational symmetric matrix such that the hypothesis of **Theorem 2.2.2** holds . It is well known (Chapter 3 of [Mey00]) that by performing a sequence of rational elementary row and column operations on M, one can decompose M as $M = LU$ (This decomposition is called the LU factorization of M.), where L is a lower triangular matrix and U is an upper triangular matrix, both with non-zero elements on the main diagonal.

It follows from the LU factorization that $U^T = L_1U_1$, where U_1 is a diagonal matrix, since all the elements in the upper triangular half of U^T are equal to zero, while L_1 is a lower triangular matrix.

Therefore we can write $M = LU_1L_1^T$. Given that M is a symmetric matrix, we have $M^T = M \iff L_1U_1L^T = LU_1L_1^T$ i.e $L = L_1$, since the same elementary operations that are applied to each row will also be applied to the corresponding column.

Hence, if we choose $P = \left(L^{-1}\right)^T$ and $\bar{D} = U_1$, **Theorem** 2.2.2 holds, where $\bar{D} = diag(\bar{d}_1, \ldots, \bar{d}_n)$.

□

When transforming the objective function of a quadratic integer programming problem into a separable[1] quadratic form, it is desirable that the resulting quadratic program keeps the integral nature.

Considering a quadratic form $q(x) = \dfrac{1}{2}x^TMx + c^Tx$ where M is such that $P^TMP = diag(\bar{d}_1, \ldots, \bar{d}_n)$, as in **Theorem** 2.2.2. We set $x = Py \iff y = P^{-1}x$, then the quadratic form q becomes $q(y) = \dfrac{1}{2}(Py)^TM(Py) + c^T(Py) = \dfrac{1}{2}y^T(P^TMP)y + c^TPy = \dfrac{1}{2}y^T\bar{D}y + \bar{c}^Ty$ with $\bar{c}^T = c^TP$. The new variable y will be ensured to be integral if P^{-1} is an integer matrix.

Definition 2.2.3 (Congruent matrices)**:**
A matrix A is said to be congruent to a matrix B if there exists a non-singular matrix P such that $P^TAP = B$. The matrix P is called the congruent matrix of A and B.

The congruence relation is an equivalence relation. Indeed we have:

- $I^TAI = A$ i.e the congruence relation is reflexive.
- $P^TAP = B \iff (P^{-1})^TB(P^{-1}) = A$, the relation is symmetric.

[1]A quadratic form is separable if its matrix is diagonal.

- $P^TAP = B$ and $Q^TBQ = C \Longrightarrow (PQ)^TA(PQ) = C$, with $(PQ)^{-1} = Q^{-1}P^{-1}$, the relation is transitive.

Definition 2.2.4 (Unimodular matrix)**:**
A matrix $U \in \mathbb{R}^{n\times n}$ is unimodular if it is integral and $\det(U) = \pm 1$, *where* $\det(U)$ *denotes the determinant of U.*

It is easy to see that a matrix P is *unimodular* if and only if P^{-1} is *unimodular*. Therefore if P is *unimodular*, then $Py \in \mathbb{Z}^n \Longleftrightarrow y \in \mathbb{Z}^n$. Hence for a quadratic integer programming problem that has $q(x) = \dfrac{1}{2}x^TMx + c^Tx$ as the objective function, the transformed separable quadratic program with the objective function $q(y) = \dfrac{1}{2}y^T\bar{D}y + \bar{c}^Ty, \quad x = Py, \quad \bar{D} = P^TMP$, will be an integer optimization problem as well. However, such a *unimodular* congruence transformation does not always exist for all the rational symmetric matrices.

Example 2.2.5:
Let

$$M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

and suppose that there is an integer matrix

$$P = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

such that P^TMP is diagonal. Then, we must have $bc + ad = 0$ and $ad - bc = \pm 1$, which implies $2ad = \pm 1$, a contradiction that a and d are both integer numbers. Thus, there does not exist a unimodular congruence transformation for M.

Definition 2.2.6 (*Semi-unimodular* matrix)**:**
A matrix P is said to be semi-unimodular if P^{-1} is an integer matrix.

Let us suppose that P is a *semi-unimodular* matrix, therefore $Py \in \mathbb{Z}^n \Longleftrightarrow y \in \mathbb{Z}^n$.

For a rational symmetric matrix M with non-zero pivots using type $iii)$ rational elementary operations under Gaussian elimination, Zheng et al [ZSL10] proposed a procedure of obtaining a *semi-unimodular* congruent matrix P such that P^TMP is diagonal. Their result is stated in the following theorem.

Theorem 2.2.7 (Integer diagonalisation)**:**
*For any rational symmetric matrix M such that the hypothesis of **Theorem** 2.2.2 holds, there exists a semi-unimodular congruent matrix P such that*

$$P^TMP = diag(d_1, \dots, d_n) \tag{2.2}$$

Proof. ***NB:*** The proof of this theorem is the same as that in [ZSL10].
Let $D_i(k)$ be the elementary matrix obtained by multiplying the i–th row of the identity matrix by k and $T_{ij}(k)$ be the matrix obtained by adding k times of the j–th row to the i–th row of the identity matrix, where k is a rational number.

It follows from 2.2.2 that there exists a non-singular rational matrix V such that $V^TMV = diag(\bar{d}_1, \dots, \bar{d}_n)$. Since the elements of M are rational, the elements k in the elementary matrices $D_i(k)$ and $T_{ij}(k)$ involved in V are also rational. Thus each element of V^{-1} is rational. If V^{-1} is an integer matrix, then $P = V$ satisfies (2.2). Suppose now that V^{-1} is not an integer matrix. Let all the entries of V^{-1} be written as (vulgar) fractions. Let k_i be the least common denominator (lcd) of the i–th row of V^{-1}. Here, we set $k_i = 1$ if all entries of the i–th row are integers. Let

$$P = VD_1\left(\frac{1}{k_1}\right)\dots D_n\left(\frac{1}{k_n}\right),$$

then

$$P^{-1} = D_n(k_n)\dots D_1(k_1)V^{-1}$$

is an integer matrix and

$$P^TMP = diag(d_1, \dots, d_n),$$

where $d_i = \bar{d}_i/k_i^2, i = 1, \dots, n$. □

Definition 2.2.8 (Diagonally dominant matrix)**:**

A matrix $M = (m_{ij})_{1 \leq i,j \leq n}$ is said to be diagonally dominant if for every row and column of M the magnitude of the diagonal entry in a row and column is greater than or equal to the sum of the magnitudes of all other (non-diagonal) entries in the same row and column.

In other words, M is diagonally dominant if $\mid m_{ii} \mid \geq \sum_{j \neq i} \mid m_{ij} \mid$ for all $i = 1, \dots, n$, M is strictly diagonally dominant if $\mid m_{ii} \mid > \sum_{j \neq i} \mid m_{ij} \mid$ for all $i = 1, \dots, n$.

Lemma 2.2.9:

If a matrix M is diagonally dominant with real non-negative diagonal entries, then it is positive semi-definite.

Proof. This lemma is a direct consequence of the Gerschgorin's circle theorem [S.G31]. □

Lemma 2.2.10:

*In addition to the assumptions of **Theorem** 2.2.7, if the matrix M is diagonally dominant then there exists a semiunimodular congruence matrix P such that $P^T MP = diag(d_1, \dots, d_n)$ and $d_i \geq 0$ for all $i = 1, \dots, n$.*

Proof. In the proof of **Theorem 2.2.7**, the matrix $diag(d_1, \dots, d_n)$ is obtained from (2.1) by the relation $d_i = \bar{d}_i / k_i^2$ for some rational number $k_i, i = 1, \dots, n$. So to end our proof, it is sufficient to show that if M is diagonally dominant, then every element $\bar{d}_i$ in (2.1) is positive. To do so we consider the Gaussian elimination that leads to (2.1), and show that every diagonal element remains positive. Given that the matrix M is non-zero and diagonally dominant, there is no need of interchanging two rows or columns. There is also no need to multiply a row or a column by a non-zero rational number. Thus the only rational elementary operations involved in the Gaussian elimination are operations of type iii), which corresponds to, for a row $R_j, R_j \longleftarrow R_j + \alpha R_i$ with $\alpha \in \mathbb{Q}^* = \mathbb{Q} \setminus \{0\}$.

If the pivot is fixed at the diagonal entry M_{ii} (which is already positive) on the row R_i, then for any other row $R_j, j > i$, we will have $R_j \longleftarrow R_j - \dfrac{M_{ji}}{M_{ii}} R_i$ and the diagonal entry on the row R_j will become $M_{jj} - \dfrac{M_{ji}}{M_{ii}} M_{ij}$. We consider the two cases below:

- If $M_{ij} \leq 0$, then $M_{jj} - \dfrac{M_{ji}}{M_{ii}} M_{ij} \geq 0$ since $\mid M_{ji} \mid \leq M_{ii}$ and $\mid M_{ij} \mid \leq M_{jj}$.
- If $M_{ij} > 0$, since $M_{ji} \leq M_{ii}$, therefore $M_{jj} - \dfrac{M_{ji}}{M_{ii}} M_{ij} \geq M_{jj} - M_{ij} \geq 0$.

Thus every diagonal entry of the resulting matrix will be left positive. Hence $\bar{d}_i \geq 0$ for $i = 1, \ldots, n$. $\square$

Theorem 2.2.11 (Eigenvalue factorization)**:**

Let M be an $n \times n$ symmetric matrix, there exists an orthogonal matrix P such that $M = P^T DP$, where D is a diagonal matrix with eigenvalues of M on the diagonal.

Definition 2.2.12 (Trace of a matrix)**:**

The trace of an $n \times n$ matrix M denoted by $tr(M)$ is the sum of all the element on its main diagonal, i.e.

$$tr(M) = \sum_{i=1}^{n} m_{ii}.$$

Given any two $n \times n$-matrices F and D, some well known properties of the trace are given by:

- $tr(FD) = tr(DF)$,
- $tr(F) = tr(F^T)$,
- for $F = F^T$ and for any $n \times n$-matrix X, we have $tr(FXD^T X^T) = tr(FXDX^T)$.

Definition 2.2.13 (Kronecker product)**:**

Let A be a real $m \times n$-matrix and B a real $p \times q$-matrix. The Kronecker product of A and

B, denoted $A \otimes B$, is defined by:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix}$$

which is the $mp \times nq$ matrix formed from all possible pairwise element products of A and B.

Example 2.2.14:
$A = \begin{pmatrix} 1 & 2 & 2 \\ 4 & 3 & 1 \end{pmatrix}$ *and* $B = \begin{pmatrix} 5 & 1 \\ 1 & 2 \end{pmatrix}$ *then*

$$A \otimes B = \begin{pmatrix} B & 2 \times B & 2 \times B \\ 4 \times B & 3 \times B & B \end{pmatrix} \tag{2.3}$$

$$= \begin{pmatrix} 5 & 1 & 10 & 2 & 10 & 2 \\ 1 & 2 & 2 & 4 & 2 & 4 \\ 20 & 4 & 15 & 3 & 5 & 2 \\ 4 & 8 & 3 & 6 & 1 & 2 \end{pmatrix}. \tag{2.4}$$

Definition 2.2.15 (Permutation)**:**
A permutation of the set $\{1, \dots, n\}$ is a one-to-one correspondence from $\{1, \dots, n\}$ onto itself.

Definition 2.2.16 (Permutation matrix)**:**
A permutation matrix is a square binary matrix that has exactly one entry 1 on each row and each column and 0's everywhere else. We denote by $\mathcal{X}_n$ the set of all permutation matrices of size $n \times n$.

Each permutation of the set $\{1, \dots, n\}$ can be represented by an $n \times n$ permutation matrix.

Definition 2.2.17 (Vector norm)**:**
Let E be a sub-vector space of $\mathbb{R}^n$. A norm on E is an application $\rho : E \longrightarrow \mathbb{R}$ which satisfies the following properties:

i) $\rho(x) \geq 0$*, for all* $x \in E$*,*

ii) $\rho(ax) = \mid a \mid \rho(x)$*, for all* $x \in E$ *and* $a \in \mathbb{R}$*,*

iii) $\rho(x + y) \leq \rho(x) + \rho(y)$*, for all* $x, y \in E$*,*

iv) If $\rho(x) = 0$*, then* $x = 0$*.*

On the vector space $\mathbb{R}^n$, we define the the following norms:

- $\|x\|_1 = \sum\limits_{i=1}^{n} \mid x_i \mid$, it is called the Taxicab norm.
- $\|x\|_2 = \left(\sum\limits_{i=1}^{n} x_i^2\right)^{1/2}$, it is called the Euclidean norm.
- $\|x\|_\infty = \max\limits_{i=1,\dots,n} (\mid x_i \mid)$, it is called the Maximum norm.

3. Literature review

In this chapter, we present a review on the QAP so that the reader can see how different is the current work to what have been done in this area already. Since its introduction, the QAP has gained a lot of attentions from researchers all over the world. This is due to its applications in a wide range of applied areas and its challenging difficulty. In this chapter, we review some important mathematical reformulation of the QAP, some of the real life applications of the QAP, the lower bounding techniques used in different solution approaches, and finally the exact and heuristic solution methods that have been adopted for the QAP.

3.1 Formulations of the QAP

Since its introduction, the QAP has been formulated in many different ways ranging from the linear form to the SDP form. Here we present some selected mathematical formulations.

3.1.1 Quadratic integer programming formulation

Considering the fact that for every permutation of $\{1,\dots,n\}$, there is a corresponding element X in $\mathcal{X}_n$, where $X=(x_{ij})_{1\le i,j\le n}$. The permutation ϕ in the original QAP formulation (1.2) can therefore be replaced by a permutation matrix $X=(x_{ij})_{1\le i,j\le n}$ where

$$x_{ij}=\begin{cases}1 & \text{if facility } i \text{ is placed at location } j,\\ 0 & \text{otherwise.}\end{cases} \tag{3.1}$$

Using this notation, Koopmans and Beckmann [KB57] gave the following quadratic integer programming formulation:

$$(QIP) \qquad \min \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}d_{kl}x_{ik}x_{jl}+\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}x_{ij}, \qquad s.t. \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{for } j = 1, \dots, n, \tag{3.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for } i = 1, \dots, n, \tag{3.4}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for } i, j = 1, \dots, n, \tag{3.5}$$

where f_{ij} is the flow between facility i and facility j, d_{kl} the distance between location k and location l and b_{ij} the cost of placing facility i at location j. This formulation can be found in most of the linearisation approaches for the QAP.

3.1.2 Trace formulation

Considering a QAP instance with flow matrix F, distance matrix D and cost matrix B, we set $\bar{D} = XD^TX^T$, which leads to $\bar{d}_{ji} = d_{\phi(i)\phi(j)}$ for $i, j = 1, \dots, n$. It then follows that

$$tr(FXD^TX^T) = tr(F\bar{D}) = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}\bar{d}_{ji} = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}d_{\phi(i)\phi(j)},$$

where ϕ is the permutation associated with the permutation matrix X.

Therefore the original formulation of the QAP can equivalently be reformulated in the following form:

$$\min \qquad tr\left[(FXD^T + B)X\right], \qquad s.t. \tag{3.6}$$

$$X \in \mathcal{X}_n, \tag{3.7}$$

which is equivalent to

$$(TF) \qquad \min \qquad tr\left[(FXD^T + B)X\right], \qquad s.t. \tag{3.8}$$

$$X^T e = e, \tag{3.9}$$

$$Xe = e, \tag{3.10}$$

$$x_{ij} \in \{0,1\} \text{ for all } i, j, \tag{3.11}$$

where e is the column n–vector of ones. This formulation was introduced by Edward [Edw80]. The spectral theory was applied to this formulation to develop the eigenvalue lower bounding and some other lower bounding techniques, see [FBR87, HRW90, Had94, KR95].

3.1.3 Kronecker formulation

Given an $n \times n$-matrix X, we define $vec(X)$ to be the n^2-vector formed by the columns of X. The QAP can thus be formulated as:

$$(KF) \qquad \min \qquad x^T(F \otimes D)x + b^T x, \qquad s.t. \tag{3.12}$$

$$X^T e = e, \tag{3.13}$$

$$Xe = e, \tag{3.14}$$

$$x_i \in \{0,1\} \text{ for all } i = 1, \ldots, n^2, \tag{3.15}$$

where $x = vec(X)$ and $b = vec(B)$. This formulation was suggested in a survey by Burkard et al. [PRW94] in 1994 but has not been studied further.

3.1.4 Mixed integer linear programming (MILP) formulation

In the MILP formulation, the QAP formulation (3.2)–(3.5) is simplified by adding some new variables, see [LW76]. These variables together with the MILP formulation are presented below. Let us consider the objective function of the quadratic integer programming formulation given by equation (3.2). In this function, we set $C_{ijkl} = f_{ij}d_{kl}$ if $i \neq j$ or $k \neq l$

and $C_{iikk} = f_{ii}d_{kk} + b_{ik}$. The new variables are now defined as $y_{ijkl} = x_{ik}x_{jl}$ for $i, j, k, l = 1, \dots, n$ and the QAP is transformed into the following MILP:

$$(MILP) \qquad \min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} C_{ijkl}y_{ijkl}, \qquad s.t. \tag{3.16}$$

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad \text{for } k = 1, \dots, n, \tag{3.17}$$

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad \text{for } i = 1, \dots, n, \tag{3.18}$$

$$\sum_{i=1}^{n} y_{ijkl} = x_{jl} \qquad \text{for } j, k, l = 1, \dots, n, \tag{3.19}$$

$$\sum_{j=1}^{n} y_{ijkl} = x_{ik} \qquad \text{for } i, k, l = 1, \dots, n, \tag{3.20}$$

$$\sum_{k=1}^{n} y_{ijkl} = x_{jl} \qquad \text{for } i, j, l = 1, \dots, n, \tag{3.21}$$

$$\sum_{l=1}^{n} y_{ijkl} = x_{ik} \qquad \text{for } i, j, k = 1, \dots, n, \tag{3.22}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for } i, j = 1, \dots, n, \tag{3.23}$$

$$y_{iikk} = x_{ik} \qquad \text{for } i, k = 1, \dots, n, \tag{3.24}$$

$$0 \leq y_{ijkl} \leq 1 \qquad \text{for } i, j, k, l = 1, \dots, n. \tag{3.25}$$

3.2 Applications of the QAP

The QAP was originally introduced by Koopmans and Beckmann [KB57] to model the assignment of activities in economy. Afterwards, the QAP has been successfully used to model problems arising from many different areas. For example, the QAP has been applied in the *backboard wiring*. The *backboard wiring* is concerned with placing the computer's elements on the backboard while minimizing a bounded numeric norm. The norm is calculated as the product of the inter-connexion between the elements, and the length of wire needed to connect elements placed at given positions. The problem was casted into mathematical formulation by Steinberg [Ste61]. As an application of mathematics in sports, Heffley [Hef77] pointed out that the assignment of runners in a relay team leads to the QAP. Geoffrion and Graves [GG76] used a quadratic assignment formulation to treat the problem of scheduling parallel production lines with changeover costs. Here the production orders for a number of products must be scheduled on a number of production lines, so as to minimize the sum of products costs. The total cost consists of the changeover costs, production costs and the cost involving time restrictions. Pollatscheck et al. [PGR76], on the other hand, used the QAP to define the best design typewriter keyboard and control panels.

The application of the QAP in Chemistry has also been reported by Forsberg et al [FDZ+94] who used the QAP in the analysis of some chemical reactions. In the area of numerical analysis, the combinatorial solution for the least-square uni-dimensional scaling of symmetric proximity matrices is known to be very sensitive to the starting point. Brusco and Stahl [BS00] proved that using the solution to a QAP as a starting point substantially improves the final seriation quality and the computational efficiency of this problem.

The QAP has a number of applications in the location problem. For example, in university *campus layout problem*, where there is a need for a university to enlarge its campuses while minimizing the amount of required travel for students and staff. The QAP happened to be the solution as was mathematically formulated so by Dickey and Hopkins [DH72]. Similarly, the problem of locating hospital department with the aim of minimizing the total distance travelled by patients was formulated by Elshafei [Els77] as a QAP. Jan Bos [Bos93] used the QAP formulation to solve the *zoning problem* which arose in forest management. This problem is concerned with the planning of territorial structures by designating area units for specific purpose.

In addition to the above examples, several applications of the QAP also arise in electronics. For example, Rabak and Schiman [RS03] showed that the problem of optimizing the automatic electronic components insertion in a particular inserting machine corresponds to a QAP. Miranda et al. [MLMF05] used the QAP formulation to model the electronic board design problem. In this problem, one needs to place electronic components to some locations in a printed circuit card so as to minimize the distance among the components that have greater levels of interactivity and energy or data flow, in order to avoid excessive signal delay. On the other hand, the index assignment which has to do with error control in communications was proved by Ben-David and Malah [DM05] to be a special case of the QAP. Wess and Zeitlhofer [WZ04] represented the problem of memory layout optimization in signal processor as a QAP. Many other QAP's applications can be found in the literature [LAN$^+$07].

3.3 The lower bounding techniques

Ever since the QAP was originally suggested, many researchers have been working on various solution techniques [LAN$^+$07]. The ability of some QAP instances to be solved, both exactly and approximately, depends on their lower bounding techniques. The lower bounding techniques are used within implicit enumeration algorithms, such as *Branch-and-Bound*, in order to perform a limited search of the feasible region of the problem, until an optimal solution is found. Therefore many researchers have focused on developing lower bounds for the QAP instances. Based on the mixed integer linear programming (MILP) formulation (3.16), Gilmore [Gil62] and Lawler [Law63] independently derived similar lower bounds (known as the Gilmore-Lawler lower bounds) for the QAP by constructing a solution matrix in the process of solving a series of *linear assignment problem*. The Gilmore-Lawler lower bounds have been widely used for roughly three decades because of their cheap computational cost.

Using the linear programming relaxation of the MILP formulation (3.16), Resende and Ramakrishnan [MRD95] computed lower bounds for the QAP instances via an interior point method. In their work, they used the primal simplex algorithm and the interior point algorithm, both available in the commercial linear programming solver CPLEX. For about 80% of the problem instances available in the QAP library [BcKR], Resende and Ramakrishnan produced lower bounds tighter than the Gilmore-Lawler lower bounds. Another bounding technique that shares the basic idea with the Gilmore-Lawler lower bounding technique has been developed by Hahn and Grant [HG98]. This lower bounding technique is based upon a dual formulation. It extends the Hungarian algorithm [Kuh55] for the linear assignment problem to the QAPs. Karisch et al. [KR95] investigated this dual-based lower bounding technique. They revealed that it is an iterative approach in which the dual of some linear programming relaxation of the original problem is solved, and reformulated at each iteration. The reformulation step makes use of the information provided in the preceding step.

Given that linear programming problems are easy to solve, many researchers have focused on the MILP formulation (3.16) in order to develop good quality lower bounds. For example, Frieze and Yadegar [FY83] gave a MILP reformulation of the QAP. They studied the Lagrangian relaxation of it, and developed two sub-gradient optimization-based algorithms to approximately solve the MILP. They were able to give lower bounds better than the Gilmore-Lawler lower bounds. Adams and Johnson [AJ94] proposed another MILP reformulation of the QAP similar to the reformulation idea of Frieze and Yadegar [FY83]. In their work, the number of constraints of the problem is considerably reduced compared to the MILP reformulation by Frieze and Yadegar [FY83]. Adams and Johnson obtained lower bounds simply by considering the continuous relaxation of their problem reformulation. On the other hand, Karisch et al [KR95] studied a theoretical relationship between the two lower bounding techniques by Adams and Johnson [AJ94] and Hahn and Grant [HG98]. It was reported that, unlike other Gilmore-Lawler-like bounds, the Hahn and Grant bounds cannot be obtained by applying the algorithm of Adams and Johnson (to solve the Lagrangian re-

laxation). However, both Adams and Johnson [AJ94] and Hahn and Grant [HG98] bounds can be obtained as feasible solutions of the dual of the continuous relaxation of the mixed integer linear programming reformulation by Adams and Johnson.
In 1990, Hadley et al. [HRW90] used the trace formulation (3.6) to develop the *projection lower bound* for the QAP. They obtained additional improvements by making an efficient use of a tractable representation of the orthogonal matrices having constant row and column sum. Based on the relationship of the objective function of the trace formulation (3.6) and the eigenvalues of its coefficient matrices, Finke et al.[FBR87] developed the "Eigenvalues lower bounds". The bounds obtained were tighter than the Gilmore-Lawler bounds. However, Clausen et al. [CKPR98] showed that the computation of these bounds is time expensive, and therefore not good for the *Branch-and-Bound* method. Based upon this general eigenvalue bounding idea, many researchers have applied some reduction techniques to the quadratic term in the objective function of the trace formulation (3.6) with the aim of improving the quality of the lower bound. These reduction techniques have significantly contributed to the improvement of the existing lower bounds [BcKR].

Another important lower bounding technique for the QAP is the one via SDP relaxation. In the literature of this lower bounding technique, valid bounds are obtained by solving the SDP relaxation using interior point methods [Kar95], and cutting plane methods [ZKRW98, Zha96]. The lower bounds obtained by this methods are very competitive [BcKR]. Burer and Vandenbussche [BV06] computed lower bounds for the general binary programming via a lift-and-project relaxation, an SDP-based relaxation, which performed very well and provided challenging bounds for the QAP instances. Further research using the SDP was carried out by Rendl and Sotirov [RS07] who combined SDP relaxation together with the bundle method to compute lower bounds. More recently, Ding and Wolkowicz [DW09] introduced a new SDP relaxation for generating lower bound for the QAP in the trace formulation (3.6). The authors applied a majorization to obtain a relaxation of the orthogonal similarity set of the quadratic part of the objective function. This exploits the matrix structure of the QAP and results in a relaxation with much smaller dimension than the previous suggested SDP relaxations by Rendl and Sotirov [RS07], Karisch [Kar95], Zhao et al. [ZKRW98] and Zhao [Zha96].

3.4 Solution methods for the QAP

Despite the calculation of quality bounds for the QAP instances, the great challenge of solving the QAP to optimality still remains. In order to achieve optimality for some QAP instances, exact solution methods for combinatorial optimization such as *Branch-and-Bound* and *cutting-plane* have been used in the literature [BcKR]. Enumerative schemes that use lower bounds to eliminate undesired solutions started with Gilmore [Gil62] and Lawler [Law63]. Hahn et al. [HGH98] proposed a *Branch-and-Bound* algorithm based on the Hungarian method [Kuh55]. For problem instances of size up to 22, this *Branch-and-Bound* [HGH98] requires significantly less computational time than other methods. Brixius and Anstreicher [BA01] developed a *Branch-and-Bound* algorithm for the QAP that uses a convex quadratic programming relaxation to obtain a bound at each node. An exhaustive list of applications of the *Branch-and-Bound* methods for the QAP can be found in a recent QAP survey by Loiola et al. [LAN$^+$07]. Zhang et al [ZRC10] have recently analysed the variables and constraints reduction of the QAP. In their work, they considered the linear programming formulation of the QAP by Adams and Johnson [AJ94]. They finally used the *Branch-and-Bound* algorithm available in the integer programming solver CPLEX in Matlab to solve the reduced problem.

It appears that there have been less applications of the *cutting-plane* method for the QAP than the *Branch-and-Bound* and the heuristic methods. Kaufman and Broekx [KB78] first used a *cutting-plane* method to solve an equivalent linear formulation of the QAP. Bazaraa and Sherali [BS82] solved a concave equivalent formulation of the QAP using a *cutting-plane* method as well. A recent implementation of the *cutting-plane* method for solving the QAP is the one by Miranda et al. [MLMF05]. They used Benders Decomposition to deal with a motherboard design problem. The reason why polyhedral cutting plane is not widely used in the context of the QAP is due to the dearth of knowledge about the QAP polytopes. Some contributions have been made in this direction by Jünger and Kaibel [JK96] and Blanchard et al. [BEFW03]. On the other hand, Gasimov and Ustun [GU07] implemented a generalized version of the modified sub-gradient algorithm. This enabled them to solve some QAP instances of sizes $12, 15, 18, 32$ and 64.

Hahn et al. [HZGS10] provide a survey of the latest methods available for solving exactly a growing class of assignment problems which includes the QAP. These techniques mainly consist of the well known reformulation linearization technique (RLT) [HG98, Zhu07]

Given that exact solution methods have not been successful enough in solving the larger QAPs within reasonable amount of time, the development of heuristic methods, which intend to have near-optimal solution within acceptable computational time, has been of great interest for some researchers. In this direction, Nissen and Paul [NP95] proposed a modification of the threshold accepting heuristic method for the QAP. Gilmore [Gil62] proposed a constructive method which is an iterative approach that usually starts with an empty permutation, and iteratively complete a partial permutation into a solution of the QAP by assigning some facilities that have not been assigned yet to some free locations. Other heuristic methods include a local search scheme, which intend to improve a given solution by searching in its neighbourhood for a better solution. For this type of heuristic, the definition of the neighbourhood structure is very important. Therefore Frieze et al. [FYEHP89] introduced the "pair-exchange" neighbourhood structure. Here, for a given solution of the QAP, its neighbours in the form of permutation matrices can be obtained by applying a transposition[1] to this solution.

[1]A matrix transposition is a permutation which exchanges two rows or two columns of a matrix while keeping all others fixed.

Another technique that has also been somewhat successfully applied to the QAP is the *Metaheuristic* method which includes *Genetic Algorithm* (GA) [Hol75], *Tabu Search* (TS) [GL], *Simulated Annealing* (SA) [KGV83], *Greedy Randomized Adaptive Search Procedure* (GRASP) [FR95], *Ant Colony Optimization* (ACO) [CDM+95], *Bees Algorithm* [PGK+05, FW10] etc.

Tabu Search was developed by Fred Glover [GL] as a metaheuristic optimization tool. Skorin-Kapov applied *Tabu Search* to find near-optimal solutions for the QAP with a fixed Tabu-list. Taillard [Tai91], on his own, proposed a robust *Tabu Search* technique by randomizing the size of the Tabu-list between a maximum and a minimum value. Misevicius [Mis05] implemented *Tabu Search* algorithm for the QAP with an efficient use of mutation applied to the best solution found so far. The application of the mutation may allow the algorithm to escape from local optima. More recently, Rego et al. [RJG10] presented a new tabu search algorithm for the quadratic assignment problem (QAP) that utilizes an embedded neighbourhood construction called an ejection chain.

Since its introduction by Kirkpatrick, SA had never been used for the QAP until Burkard and Rendl [BR84] proposed its first application to the QAP. Subsequently, Whilhelm and Ward [WW87] presented a new equilibrium component of the SA. Connolly [Con90] also proposed a SA algorithm for solving the QAP, by employing the “pair-exchange” neighbourhood structure of Whilhelm and Ward. However, the two approaches differ on the implementation of the “cooling schedule”.

Introduced by Holland [Hol75], the GA is a nature inspired approach for combinatorial optimization problems. It adapts the evolutionary mechanism acting in selection process in nature to combinatorial optimization problems. Tate and Smith [TS95] proposed a standard GA method for the QAP. Experimental results show that this algorithm has difficulties to generate the best known solutions even for QAP instances of small to moderate sizes. Fleurent and Ferland [FG99] proposed a combination of the GA techniques and TS . On the other hand, good results were obtained with the greedy Genetic Algorithm proposed by Ahuja et al. [AOT00]. Ji et al. [JWL06] presented a recent implementation of GA for the QAP. They proposed a hybrid GA to examine the solvability of the QAP instances. Their numerical results are better than those of Ahuja et al. [AOT00].

The GRASP is a combination of greedy elements with random search elements in a two phase heuristic. It was introduced by Feo and Resende [FR95]. It consists of a construction phase in which good solutions from available feasible space are constructed, and a local improvement phase where the neighbourhood of the solution constructed in the first phase is investigated for possible improvement. There is a number of applications of GRASP to the QAP, see the references [FR95, LPR94, RPL96, OPR04, FG99, AOT00].

ACO is a class of algorithms whose first member called *Ant System* was initially proposed by Colorni et al. [CDM+95]. The main underlying idea, loosely inspired by the behaviour of real ants, is that of a parallel search over several constructive computational threads based on local problem data and a dynamic memory structure containing informations on the quality of previously obtained result. The collective behaviour emerging from the interaction of the different search threads has been proved effective in solving combinatorial optimization problems. Stützle and Dorigo [SD99] applied ACO algorithm to the QAP and obtained good results for the QAP instances. These are available in the QAPLIB library [BcKR].

A recent advanced metaheuristics for the QAP is the incorporation in a single framework of GA, SA and TS. This work was done by Song et al. [SLSD09] to find good approximation of the solution of large QAP instances.

4. Problem formulation and lower bounding techniques

We have noticed that in the literature, most of the attentions have been given to the linearisation techniques and the SDP formulation of the QAP. To the best of our knowledge, very few researchers have considered the standard quadratic integer programming formulation $(SQIP)$ of the QAP. This formulation was originally suggested by Burkard et al. [BcPP98] in their survey paper on the QAP, but has not been investigated further. Bazaraa and Sherali [BS82] used this form to construct an equivalent concave quadratic integer programming formulation of the QAP that they solved using a *cutting-plane* method. In this chapter, we present a reformulated standard quadratic integer programming $(RSQIP)$ formulation of the QAP which has been modified from $(SQIP)$. We present an equivalent separable convex quadratic integer programming reformulation $(SCQIP)$. These two formulations will be studied in this dissertation for the computation of lower bounds as well as for developing solution methods for the QAP. We also discuss in this chapter two lower bounding techniques. Firstly, we consider the continuous relaxation of the reformulated standard quadratic integer programming $(RSQIP)$ formulation of the QAP. We used this lower bounding technique within the *Branch-and-Bound* method. Secondly, we consider the continuous relaxation of its equivalent separable convex quadratic integer programming $(SCQIP)$ reformulation. The lower bounds obtained in this case were too weak to be considered for the *Branch-and-Bound* method. Nonetheless, we have presented some results on this, see section 7.1.

4.1 Standard quadratic integer programming formulation $(SQIP)$

Let us consider the $n \times n \times n \times n$ cost matrix $C = (C_{ijkl})$ as constructed in section 3.1.4, $C_{ijkl} = f_{ij}d_{kl}$ if $i \neq j$ or $k \neq l$ and $C_{iikk} = f_{ii}d_{kk} + b_{ik}$. We define an $n^2 \times n^2$-matrix S in such a way that the element C_{ijkl} of the matrix C is on the row $(i-1)n+k$ and column $(j-1)n+l$ of S and let $x = vec(X)$. With the above notations, the QAP can be formulated as [BcPP98]:

$$(SQIP) \qquad \min \qquad x^T Sx, \qquad s.t. \tag{4.1}$$

$$X^T e = e, \tag{4.2}$$

$$Xe = e, \tag{4.3}$$

$$x_{ij} \in \{0,1\}, \qquad \text{for all } i,j = 1,\dots,n. \tag{4.4}$$

We now present a variation of $(SQIP)$. We begin with the definition of the following matrices. Let E and R be the two $n \times n^2$ matrices such that

$$E = \begin{bmatrix} e & 0 & \cdots & 0 \\ 0 & e & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & \cdots & e \end{bmatrix}^T = I_n \otimes e^T, e \in \mathbb{R}^n$$

and $R = (R_{ij})$ with

$$R_{ij} = \begin{cases} 1 & \text{if} \qquad j = kn + i \text{ for all } k = 0, \cdots, (n-1), \\ 0 & \text{otherwise.} \end{cases}$$

It can easily be seen that $R = e^T \otimes I_n$, where I_n is the n-dimensional identity matrix. The constraint $X^T e = e$ can be written as $Ex = e$ and the constraint $Xe = e$ can also be written as $Rx = e$. We set $A_1 = \begin{bmatrix} E \\ R \end{bmatrix}$ and $b = \begin{bmatrix} e \\ e \end{bmatrix}$.

Lemma 4.1.1:

The matrix A_1 defined above is of rank $2n-1$.

Proof. This result emanates from the study of the QAP polytopes by Jüngle and Kaibel [JK96]. □

From the results of Lemma 4.1.1, we consider throughout this dissertation that A_1 is of full row rank. Using the notations described above, we can reformulate $(SIQP)$ as follows:

$$(RSQIP) \qquad \min \qquad x^T Sx, \qquad s.t. \tag{4.5}$$

$$A_1 x = b, \tag{4.6}$$

$$0 \leq x \leq e, \tag{4.7}$$

$$x \in \mathbb{Z}^{n^2}. \tag{4.8}$$

Bazaraa and Sherali [BS82] explored the objective function of $(RSQIP)$ and transformed it into an equivalent concave quadratic programming. More specifically, the objective function of $(RSQIP)$ can be transformed into an equivalent concave quadratic function by subtracting a positive constant term on the diagonal of S, see [BS82]. It can also be transformed into a convex quadratic programming by adding a positive constant term to the diagonal of S and this is also what we investigate in this dissertation. The reformulated standard quadratic formulation $(RSQIP)$ has never been investigated in the literature. In addition, the convex equivalent formulation of $(RSQIP)$ is an interesting problem. The convex formulation is achieved by adding a non-negative constant, say α, to the diagonal of S. Indeed, if α is a real constant number, then we have:

$$x^T(S + \alpha I)x = x^T Sx + \alpha x^T x = x^T Sx + \alpha n. \tag{4.9}$$

It can be shown that the optimizer of $(RSQIP)$ remains the same if S is replaced by $S+\alpha I$, with α chosen to be larger than the maximum row sum or column sum of S. Indeed, (4.9) shows that replacing S in $(RSQIP)$ by $S + \alpha I$ only changes the objective function of $(RSQIP)$ by a constant, therefore does not change minimizers.

4.2 Separable quadratic integer programming formulation

In this section, we transform $(RSQIP)$ into a convex quadratic programming problem by replacing the matrix S in its objective function with a matrix $Q = 2(S + \alpha I)$ where α is chosen to be larger than the maximum row sum of S. This leads to the following convex quadratic integer programming formulation:

$$(CQIP) \qquad \min \quad \frac{1}{2}x^T Q x, \qquad s.t. \tag{4.10}$$

$$A_1 x = b, \tag{4.11}$$

$$0 \leq x \leq e, \tag{4.12}$$

$$x \in \mathbb{Z}^{n^2}. \tag{4.13}$$

Note that $(CQIP)$ is equivalent to $(RSQIP)$, since the feasible sets are the same and the objective functions only differ by a constant as shown in (4.9). The matrix Q is a symmetric positive definite matrix.

Given the eigenvalue decomposition, see **Theorem 2.2.11**, one can easily transform $(CQIP)$ into a separable quadratic program by using the eigenvalue decomposition of the matrix Q. There exists an orthogonal matrix U such that $U^T Q U = diag(\lambda_1, \dots, \lambda_{n^2}) = D_1$, where $\lambda_1, \dots, \lambda_{n^2}$ are the eigenvalues of Q. We then set $x = Uy$ and plug this in $(CQIP)$. The objective function becomes:

$$1/2(Uy)^T Q(Uy) = 1/2y^T(U^T QU)y = 1/2y^T D_1 y,$$

which is now a separable quadratic function, since D_1 is a diagonal matrix. However, the change of variable, $x = Uy \iff y = U^{-1}x$, may not guarantee the integrability of y. Therefore, the initial problem $(CQIP)$ might lose its integrability nature.

A good way of making $(CQIP)$ separable while keeping its integrability nature can be achieved by using the integer diagonalization of the matrix Q (**Theorem 2.2.7**). Let U

be semiunimodular congruent to Q such that $U^TQU = D = diag(d_1, \dots, d_{n^2})$. We set $x = Uy \iff y = U^{-1}x$. U^{-1} being an integer matrix, $x \in \mathbb{Z}^{n^2} \iff y \in \mathbb{Z}^{n^2}$ and the following problem is equivalent to $(CQIP)$:

$$(SCQIP) \qquad \min \qquad 1/2y^TDy, \tag{4.14}$$

$$s.t \qquad Ay = b, \tag{4.15}$$

$$0 \le Uy \le e, \tag{4.16}$$

$$y \in \mathbb{Z}^{n^2}, \tag{4.17}$$

where $A = A_1U$. The above problem is a separable convex quadratic integer programming problem.

4.3 Lower bounds via the continuous relaxation of $(RSQIP)$

The computation of lower bounds for integer programming problems is of crucial importance for both exact and heuristic solution methods. The quality of a lower bound is measured in terms of how tight or how close it is to the exact solution of a problem, and also in terms of its computational time and complexity requirements. This section presents our first lower bounding technique which consists of relaxing the integrability requirements of $(RSQIP)$. In particular, we consider the quadratic programming problem:

$$(CR1) \qquad \min \qquad x^TSx, \qquad s.t \tag{4.18}$$

$$A_1x = b,$$

$$0 \le x \le e.$$

Any standard optimization method for quadratic programming such as *trust-region*, *active-set*, *interior-point* method for non convex non-linear programming can be used to solve

$(CR1)$ efficiently. We have used the built-in function *QUADPROG* from Matlab within the implementation of the *Branch-and-Bound* method for solving $(CR1)$. The Matlab built-in function *QUADPROG* implements an interior point algorithm which requires an initial strictly feasible solution to be provided. *QUADPROG* incorporates a default heuristic procedure of generating the strictly initial solution.

4.4 Lower bounds via the continuous relaxation of $(SCQIP)$

In this section, we propose our second lower bounding technique for the QAP by considering the continuous relaxation of $(SCQIP)$ which is as follows:

$$(CR2) \qquad \min \qquad 1/2y^T Dy, \tag{4.19}$$

$$s.t \qquad Ay = b, \tag{4.20}$$

$$Cy - s = d, \tag{4.21}$$

$$s \geq 0, \tag{4.22}$$

where $C = \begin{pmatrix} -U \\ U \end{pmatrix} \in \mathbb{Q}^{2n^2 \times n^2}$, $d = \begin{pmatrix} -e \\ 0 \end{pmatrix}$ a $2n^2$-vector and s is the excess variable for the inequality constraints. $(CR2)$ is a separable convex quadratic programming problem. Nimrod and Arie [MT93] have shown that such a problem is not more difficult than linear programming to be solved using interior point methods. Hence we have decided to used an interior point method. Next, we present an interior point algorithm to solve $(CR2)$.

4.4.1 An interior point algorithm for $(CR2)$

In order to eliminate the non-negativity constraints $s \geq 0$ in $(CR2)$, we introduce the barrier logarithmic problem as follows:

$$(BL) \qquad \min \qquad \tfrac{1}{2}y^T Dy - \mu \sum_{i=1}^{2n^2} \ln s_i, \tag{4.23}$$

$$\text{s.t} \qquad Ay = b, \tag{4.24}$$

$$Cy - s = d. \tag{4.25}$$

The corresponding Lagrangian function will then be:

$$L = \tfrac{1}{2}y^T Dy - \mu \sum_{i=1}^{2n^2} \ln s_i - \lambda^T(Ay - b) - z^T(Cy - s - d).$$

Since (BL) is convex, therefore the first order optimality conditions will be sufficient and necessary.

The KKT first order optimality conditions are given by :

$$\nabla_y L = \nabla_s L = \nabla_\lambda L = \nabla_z L = 0,$$

which is equivalent to:

$$Dy - A^T\lambda - C^T z = 0, \tag{4.26}$$

$$Ay - b = 0, \tag{4.27}$$

$$Cy - s - d = 0, \tag{4.28}$$

$$SZe = \mu e, \tag{4.29}$$

$$s, z > 0, \tag{4.30}$$

where $e = (1, \ldots, 1), S = diag(s_1, \ldots, s_{2n^2})$ and $Z = diag(z_1, \ldots, z_{2n^2})$. The interior point method aims to solve the above (KKT) system by constructing a sequence $(y^k, \lambda^k, z^k, s^k)$

that converges toward the solution of the (KKT) system. In our solution method, we have adopted a predictor corrector algorithm, as presented by Mehrotra [Meh92], which controls the step-length of the sequence in order to ensure that all the points of the sequence remain feasible for $(CR2)$. The full step-by-step description of the interior point method is presented by **Algorithm 1**. We also provide description of the main steps in section 4.4.2.

Algorithm 1 : The interior point algorithm for $(CR2)$

Step 1: (initialization): Set $k = 0$, set a starting point $(y^k, \lambda^k, z^k, s^k)$ which is strictly feasible for $(CR2)$, and a parameter $\tau \in [2, 4]$.

Step 2: If stopping criteria is met then stop, else compute $\mu = (z^k)^T s^k / 2n^2$.

Step 3: Solve for $(\delta y^k, \delta\lambda^k, \delta z^k, \delta s^k)$:

$$\begin{bmatrix} D & -A^T & -C^T & 0 \\ A & 0 & 0 & 0 \\ C & 0 & 0 & -I \\ 0 & 0 & S^k & Z^k \end{bmatrix} \begin{bmatrix} \delta y^k \\ \delta\lambda^k \\ \delta z^k \\ \delta s^k \end{bmatrix} = - \begin{bmatrix} r_D^k \\ r_A^k \\ r_C^k \\ r_z^k \end{bmatrix}, \tag{4.31}$$

where

$$\begin{aligned} S^k &= diag(s_1^k, \ldots, s_{2n^2}^k), \\ Z^k &= diag(z_1^k, \ldots, z_{2n^2}^k), \\ r_D^k &= Dy^k - A^T\lambda^k - C^T z^k, \\ r_A^k &= Ay^k - b, \\ r_C^k &= Cy^k - s^k - d, \\ r_z^k &= Z^k S^k e. \end{aligned}$$

Step 4: Calculate α_{aff} to be the largest value in $(0, 1]$ such that $(z^k, s^k) + \alpha_{aff}(\delta z^k, \delta s^k) \geq 0$.

Step 5: Set $\mu_{aff} = (z^k + \alpha_{aff}\delta z^k)^T (s^k + \alpha_{aff}\delta s^k)/2n^2$, set $\sigma = (\mu_{aff}/\mu)^\tau$.

Step 6: Solve for $(\Delta y^k, \Delta\lambda^k, \Delta z^k, \Delta s^k)$:

$$\begin{bmatrix} D & -A^T & -C^T & 0 \\ A & 0 & 0 & 0 \\ C & 0 & 0 & -I \\ 0 & 0 & S^k & Z^k \end{bmatrix} \begin{bmatrix} \Delta y^k \\ \Delta\lambda^k \\ \Delta z^k \\ \Delta s^k \end{bmatrix} = - \begin{bmatrix} r_D^k \\ r_A^k \\ r_C^k \\ r_\mu^k \end{bmatrix}, \tag{4.32}$$

where $r_\mu^k = Z^k S^k e - \sigma\mu e + \Delta Z^k \Delta S^k e$, $\Delta Z^k = diag(\Delta z_1^k, \ldots, \Delta z_{2n^2}^k)$ and $\Delta S^k = diag(\Delta s_1^k, \ldots, \Delta s_{2n^2}^k)$.

Step 7: Calculate $\alpha_{\max}$ to be the largest value in $(0, 1]$ such that $(z^k, s^k) + \alpha_{\max}(\Delta z^k, \Delta s^k) \geq 0$.

Step 8: Choose $\rho \in (0, \alpha_{\max})$.

Step 9: $(y^{k+1}, \lambda^{k+1}, z^{k+1}, s^{k+1}) \longleftarrow (y^k, \lambda^k, z^k, s^k) + \rho(\Delta y^k, \Delta\lambda^k, \Delta z^k, \Delta s^k)$. Set $k := k + 1$ and go to ***Step 2***.

The algorithm deals with the solution of two systems of equations, namely (4.31) and (4.32) which involve the predictor and the corrector direction respectively. At iteration k, the direction obtained from (4.32) can be viewed as an approximate second-order step toward a point $(y^{k+1}, \lambda^{k+1}, z^{k+1}, s^{k+1})$ at which the conditions (4.26), (4.27) and (4.28) are satisfied, and in addition, the pairwise products $z_i^{k+1}s_i^{k+1}$ are all equal to $\sigma\mu$. The heuristic for σ yields a value in the range $(0, 1)$, so the step usually produces a reduction in the average value of the pairwise product from their current average μ.

The successive corrections attempt to:

- increase the steplength ρ that can be taken along the final direction,
- bring the pairwise product $z_i^+s_i^+$ whose values are either much larger than or much smaller than the average into closer correspondence with the average.

4.4.2 Discussions of the interior point algorithm

In this section, we elaborate on the main steps of **Algorithm 1**.

- **Solution of the linear systems (4.31) and (4.32)**

  There are two large linear systems of equation (4.31) and (4.32) to be solved in **Algorithm 1**. This can be time consuming for the algorithm if the left hand matrices are not reduced. Given some properties of $(CR2)$ stated in **Lemma 4.4.1**, we can reduce these systems and make them easy to solve.

  Let us consider the linear system of equations (4.32). Without jeopardizing the generality, we can write this system here without the superscripts k.

$$D\Delta y - A^T\Delta\lambda - C^T\Delta z = -r_D, \tag{4.33}$$

$$A\Delta y = -r_A, \tag{4.34}$$

$$C\Delta y - \Delta s = -r_C, \tag{4.35}$$

$$S\Delta z - Z\Delta s = -r_\mu. \tag{4.36}$$

Equality (4.35) implies that $\Delta s = C\Delta y + r_C$.

Replacing this in (4.36), we have $\Delta z = S^{-1}(-r_\mu - ZC\Delta y - Zr_C)$. We now plug this in (4.33) and using (4.34), we obtain

$$\begin{aligned}
&\Delta\lambda = (APA)^{-1}(-r_A - APR_2),\\
&\Delta y = P(A^T\Delta\lambda + R2),\\
&\Delta z = S^{-1}(-r_\mu - ZC\Delta y - Zr_C),\\
&\Delta s = C\Delta y + r_C,
\end{aligned}$$

where $R_2 = -r_D - C^TS^{-1}r_\mu - C^TS^{-1}Zr_C$ and $P = (D + C^TS^{-1}ZC)^{-1}$.

The above technique is also repeated for the solution of (4.31).

Lemma 4.4.1:

The matrix $D + C^TS^{-1}ZC$ is invertible. Hence the existence of P.

Proof. The matrix $S^{-1}Z$ is a diagonal $2n^2 \times 2n^2$-matrix with positive elements on the diagonal. Let T_1 be the first $n^2 \times n^2$ diagonal bloc of $S^{-1}Z$ and T_2 be the second $n^2 \times n^2$ diagonal bloc of $S^{-1}Z$. We then have $S^{-1}Z = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix}$. Recall that

$C = \begin{pmatrix} -U \\ U \end{pmatrix}$. Therefore we have

$$\begin{aligned} C^T S^{-1} Z C &= \begin{pmatrix} -U \\ U \end{pmatrix}^T \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} \begin{pmatrix} -U \\ U \end{pmatrix} \\ &= (-U^T, U^T) \begin{pmatrix} -T_1 U \\ T_2 U \end{pmatrix} \\ &= U^T T_1 U + U^T T_2 U. \end{aligned}$$

Therefore,

$$\begin{aligned} D + C^T S^{-1} Z C &= D + U^T T_1 U + U^T T_2 U \\ &= U^T Q U + U^T T_1 U + U^T T_2 U, \qquad \text{since } U^T Q U = D, \\ &= U^T (Q + T_1 + T_2) U, \end{aligned}$$

with Q being a symmetric strictly diagonally dominant matrix with positive diagonal entries, adding positive[1] numbers to its diagonal will yield another strictly diagonally dominant matrix which is invertible. Hence $D + C^T S^{-1} Z C = U^T (Q + T_1 + T_2) U$ is invertible. □

- **Calculation of the step lengths α_{max} and α_{aff} in *Step 4* and *Step 7*:**
  In **Algorithm 1**, we have to find α_{max} and α_{aff} to be the largest values in $(0, 1]$ such that

$$(z^k, s^k) + \alpha_{max}(\Delta z^k, \Delta s^k) \geq 0 \tag{4.37}$$

and

$$(z^k, s^k) + \alpha_{aff}(\delta z^k, \delta s^k) \geq 0. \tag{4.38}$$

For the easiness of reading, we will deal with (4.37) without using the superscripts k. We consider $(z, s) + \alpha(\Delta z, \Delta s) \geq 0 \iff z + \alpha_z \Delta z \geq 0, \quad s + \alpha_s \Delta s \geq 0$ i.e

[1]Since the matrices T_1 and T_2 are diagonal matrices with positive entries in the diagonal.

$$\begin{pmatrix} z_1, s_1 \\ \vdots \\ z_i, s_i \\ \vdots \\ z_{2n^2}, s_{2n^2} \end{pmatrix} + \alpha \begin{pmatrix} \Delta z_1, \Delta s_1 \\ \vdots \\ \Delta z_i, \Delta s_i \\ \vdots \\ \Delta z_{2n^2}, \Delta s_{2n^2} \end{pmatrix} \geq 0$$

For the i-th component, we can write $z_i + \alpha_z^i \Delta z_i \geq 0$ and $s_i + \alpha_s^i \Delta s_i \geq 0$, where α_s^i and α_z^i are the largest values in $(0, 1]$ for which the inequality holds. Therefore,

$$\alpha_z^i = \begin{cases} 1 & \text{if} \quad \Delta z_i \geq 0, \\ \min(1, -z_i/\Delta z_i) & \text{otherwise,} \end{cases}$$

and

$$\alpha_s^i = \begin{cases} 1 & \text{if} \quad \Delta s_i \geq 0, \\ \min(1, -s_i/\Delta s_i) & \text{otherwise.} \end{cases}$$

Let $a_1 = \min\left\{\alpha_z^1, \ldots, \alpha_z^{2n^2}\right\}$ and $a_2 = \min\left\{\alpha_s^1, \ldots, \alpha_s^{2n^2}\right\}$, let $\alpha_{max} = \min\{a_1, a_2\}$. Clearly, α_{max} is the largest value in $(0, 1]$ satisfying $(z, s) + \alpha_{max}(\Delta z, \Delta s) \geq 0$

A similar procedure can be applied to find the value of α_{aff}.

- **Termination criteria in *Step 2*:**

For each iteration k of the algorithm, let us define the duality gap $g_k = (y^k)^T D y^k - b^T \lambda^k - d^T z^k$. In addition, we define $\phi_k = \dfrac{\|(r_D^k, r_A^k, r_C^k)\|_\infty + g_k}{\|(D, A, C, b, d)\|_\infty}$ at iteration k, where $\|(D, A, C, b, d)\|_\infty$ stands for the element of largest magnitude in all the data quantities that define $(CR2)$. Therefore the algorithm successfully terminates if the following two conditions are satisfied:

i) The first condition is based on complementarity convergence, i.e

$$\mu \leq \varepsilon_\mu$$

ii) The second condition is based on the primal-dual convergence i.e

$$\|(r_D^k, r_A^k, r_C^k)\|_\infty \leq \varepsilon_r \|(D, A, C, b, d)\|_\infty.$$

where ε_μ and ε_r are tolerance values.

An infeasible solution is obtained by **Algorithm 1** if :

$$\phi_k > \varepsilon_\phi \qquad \text{and} \qquad \phi_k \geq 10^4 \min_{1\leq i\leq k} \phi_i,$$

where ε_ϕ is a used provide tolerance value. See [Meh92] for more details on the termination criteria and other implementational issues of **Algorithm 1**.

5. Finding a starting point for the interior point algorithm

Any interior point algorithm needs a starting point which is in the strictly feasible region of the original problem, from which the algorithm will start and consecutively build a sequence of point that converges towards the optimal solution of the problem. Finding such an initial point is not an easy task. We present in this chapter two techniques for finding an initial starting point for the interior point algorithm. The first technique finds an initial strictly feasible point in two steps. The second technique is based on a perturbation of the set of constraints. This technique shares the same idea in the first step of the first technique.

5.1 Description of the first technique

Here, we obtain an initial strictly feasible point in two steps, the first step will provide us with a point that is on the vertex of the feasible region and the second step will aim to pull this point in the interior of the feasible region. Let us consider the set of constraints[1] of $(CR2)$ defined by:

$$Ay = b \tag{5.1}$$

and

$$Cy \geq d. \tag{5.2}$$

5.1.1 The first step

In this step, we define an auxiliary linear program for which an optimal solution will be a point on the vertex of the feasible region of $(CR2)$. We use "artificial" variables which are

[1]The constraints of $(CR1)$ can also be written in the form $A_1 x = b, \; Cx \geq d,$ where C and d have been constructed from $x \geq 0, \quad -x \geq -e.$

extra variables added to $(CR2)$ resulting in the following problem

$$\min \quad \tfrac{1}{2}y^T Dy \tag{5.3}$$

$$\text{s.t} \quad Ay + a^0 = b, \tag{5.4}$$

$$Cy - s + a^1 = d, \tag{5.5}$$

$$a^0, a^1, s \geq 0, \tag{5.6}$$

where $a = \begin{pmatrix} a^0 \\ a^1 \end{pmatrix} \in \mathbb{R}^{2n-1+2n^2}$, see **Lemma 4.1.1** for more information on the dimension of a^0. The following auxiliary linear programming problem is then considered:

$$(AP) \qquad \min \sum_{i=1}^{2n^2+2n-1} a_i, \qquad \text{s.t} \tag{5.7}$$

$$Ay + a^0 = b, \tag{5.8}$$

$$Cy - s + a^1 = d, \tag{5.9}$$

$$a^0, a^1, s \geq 0. \tag{5.10}$$

Lemma 5.1.1:

If $(CR2)$ is feasible, then (AP) is also feasible and has optimal value zero, achieving at $a = 0$.

Proof. Let us suppose that $(CR2)$ is feasible and that the optimal value of (AP) is not zero. It yields that there exists $i_0 \in \{1, \ldots, 2n^2 + 2n - 1\}$ such that $a_{i_0} \neq 0$ i.e $a_{i_0} > 0$. We then have for y in the feasible solution,

(i) $A_{i_0}^T y + a_{i_0}^0 = b_{i_0}$ with $a_{i_0}^0 = a_{i_0} > 0$, where A_{i_0} is the i_0-th row of A, or

(ii) $C_{i_0}^T y - s_{i_0} + a_{i_0}^1 = d_{i_0}$ with $a_{i_0}^1 = a_{i_0} > 0$, where C_{i_0} is the i_0-th row of C.

The condition (i) is equivalent to $A_{i_0}^T y < b_{i_0}$ and the condition (ii) is equivalent to $C_{i_0}^T y - s_{i_0} < d_{i_0}$.

Both the cases (i) and (ii) contradict the fact that y is feasible for $(CR2)$. □

It follows from **Lemma 5.1.1** that if the initial problem $(CR2)$ is feasible, then the auxiliary linear problem (AP) in the first step is also feasible and provides us with an initial solution which lies[2] on the vertex of the feasible region of $(CR2)$. This procedure is known as the phase 1 in the two phase method for linear programming [NW99].

Now that we have a point lying on the vertex of the feasible region of $(CR2)$, we can obtain an initial solution in the strictly feasible region by pulling this point, obtained from the first step, in the interior of the feasible region. This is described in the next section.

5.1.2 The second step

This second step consists of moving the point provided by the first step, described previously, in a direction that will finally end at a point in the strictly feasible region. That is, if y^0 is the point obtained from the first step, we will be looking for a strictly feasible point y^1.

Let us consider δ to be the direction in which we move the point y^0, then y^1 will satisfy $y^1 = y^0 + \alpha\delta$ for some $\alpha > 0$ which is chosen in such a way that y^1 is strictly feasible.

The feasibility of y^1 implies $Ay^1 = Ay^0 + \alpha A\delta = b$, which yields $\alpha A\delta = 0$. So the direction δ should satisfy $A\delta = 0$, i.e δ must be in the null space of A. Since y^0 is a vertex point of the feasible region, there exists a set of linear constraints in (5.2) that are active at y^0. Without loss of generality, we assume that $\{1, \ldots, l\}$ is the index set of active constraints at y^0, i.e

$$C_i^T y^0 = d_i, \text{ for } i = 1, \ldots, l, \quad l < 2n^2$$

[2]The fact that this initial solution lies on the vertex comes from the fact that it is the solution of a linear program over a convex set.

For a direction δ to be a non-binding feasible direction, it must satisfy $C_i^T(y^0+\delta) > d_i, \quad i = 1,\dots,l$ i.e $C_i^T\delta > 0, \quad i = 1,\dots,l$.

Let $A_l = [C_1, C_2, \dots, C_l]$, so $A_l^T\delta > 0$ characterizes the feasible direction δ which is non-binding with respect to $\{1,\dots,l\}$ such that δ can be found by solving

$$K\delta = \beta, \tag{5.11}$$

where $K = \begin{bmatrix} A \\ A_l^T \end{bmatrix}$ and $\beta = \begin{bmatrix} 0 \\ e \end{bmatrix}$. The matrix K is in general not a square matrix or invertible. Therefore, solving (5.11) may prove tricky. We consider K^+ to be the generalized inverse of K (more details on the generalized inverse of a matrix are given in Appendix 9.1). The equation (5.11) is therefore solved by $\delta = K^+e$, which is the best fit solution. Now $y^1 = y^0 + \alpha\delta$, reduce α until y^1 is strictly feasible.

5.2 Description of the second technique

The idea of obtaining an initial point here is basically the same as in the first step of the technique presented previously. We shrink the feasible set of $(CR2)$ by a perturbation of its boundary. Hence a point in the perturbed boundary will be a strictly feasible point in the original feasible set. Recall that the feasible set of $(CR2)$ is given by (5.1) and (5.2).

We need a starting point y^0 which satisfies $Ay^0 = b$ and $Cy^0 > d$. Let ε be a vector of all positive components. If ε is well chosen such that the set $\Gamma = \left\{y \in \mathbb{R}^{n^2} : Ay = b, Cy \geq d + \varepsilon\right\}$ is non empty, then at least one point in Γ will be strictly feasible for $(CR2)$, since $Cy \geq d + \varepsilon > d$. Before we choose ε, a question might arise concerning the existence of $\varepsilon > 0$ such that the set Γ is non empty. Since the feasible set $\{y | Ay = b, Cy \geq d\}$ is continuous and not singleton, Γ is non empty. Let $\varepsilon > 0$ be small enough. We set $d' = d + \varepsilon$ and seek the solution of (AP), described by (5.7)–(5.10), wherein d in the right hand side of (5.10) is replaced with d'. In particular, an initial strictly feasible solution of the interior point algorithm will be the solution of the following problem:

$$\min \quad \sum_{i=1}^{2n^2+2n-1} a_i, \qquad \text{s.t} \tag{5.12}$$

$$Ay + a^0 = b, \tag{5.13}$$

$$Cy - s + a^1 = d', \tag{5.14}$$

$$a^0, a^1, s \geq 0, \tag{5.15}$$

where $a = \begin{pmatrix} a^0 \\ a^1 \end{pmatrix}$.

Now, (y, s) in the solution of the above linear program (5.12)–(5.15) satisfies $Ay = b$ and $Cy - s = d' = d + \varepsilon > d \Longleftrightarrow Cy - s > d$, since $\varepsilon > 0$.

It is well known from the duality theorem of linear programming [NW99] that if the dual of a linear programming problem is unbounded, then this problem will have no feasible point. So, on the choice of an appropriate $\varepsilon > 0$ such that $Cy \geq d + \varepsilon$, one can start with a small positive value of its components. If for this value, the dual of (5.12)–(5.15) is unbounded, one can reduce the chosen value and test it again, until having an ε for which the dual of (5.12)–(5.15) is bounded.

6. New methods for solving the QAP

In this chapter, we propose two solution approaches for solving $(RSQIP)$. The first solution approach is a *Branch-and-Bound* method. The second solution approach is a discrete dynamic convexized method which consists of an auxiliary function, which we sequentially minimize by a local search algorithm. The optimal solution of the defined auxiliary function is also the optimal solution of $(RSQIP)$.

6.1 The *Branch-and-Bound* method for the QAP

In this section, k is used to denote an iteration, j is used to denote the j-th sub-problem or node, and i is the index of the i-th variable x_i.

At the beginning of the *Branch-and-Bound* method, we obtained a solution called the incumbent solution, say $x^*, f(x^*)$, by a heuristic technique, see section 7.2. This solution, which is an upper bound, is updated within the *Branch-and-Bound* method. The *Branch-and-Bound* method has been widely adopted as a basic enumeration strategy for combinatorial optimization. It is well known that it is a successful and robust method for linear integer programming when combined with linear programming techniques [BGG+71]. It has also been established as an effective computational tool for solving mixed integer programming problems [LS10]. *Branch-and-Bound* is basically an implicit enumeration scheme which systematically eliminates non-promising feasible points that cannot lead to optimality.

Let $\mathcal{F} = \left\{ x \in \mathbb{Z}^{n^2} : A_1 x = b, 0 \leq x \leq e \right\}$ be the feasible set of $(RSQIP)$. *Branch-and-Bound* is an iterative algorithm and at the iteration k, it maintains p sub-problems in $L = \{(RSQIP_1), \ldots, (RSQIP_p)\}$, with the corresponding feasible sets $\mathcal{F}_1, \ldots, \mathcal{F}_p$.

An iteration begins with selecting a sub-problem from L and may end with adding a new sub-problem(s) in L. We now describe how a sub-problem (or more than one sub-problems) is selected from L for further investigation, and how newly created sub-problems are added to L. We begin with the sub-problem selection. While there are a number of ways one can select a sub-problem(s) from L, we have used a recency based technique. We have described these techniques following the description of *Branch-and-Bound*. The k-th iteration begins with selecting a sub-problem. Once a sub-problem, say the j-th sub-problem, has been selected, the following steps are carried out:

- Solution of the continuous relaxation of the sub-problem using *QUADPROG* is obtained. This solution is a lower bound, LB_j, of $(RSQIP_j)$. If $LB_j > f(x^*)$ then $(RSQIP_j)$ is fathomed and a new sub-problem is selected.
- If the solution is an integer solution, then the incumbent x^* and $f(x^*)$ are updated. The process in this case is ended without creating any new sub-problem from the selected sub-problem $(RSQIP_j)$. A new sub-problem is selected from L again.
- If the solution of the continuous relaxation of $(RSQIP_j)$ is not integer, then two new sub-problems are created from $(RSQIP_j)$ by searching[1] a variable, say the i-th variable x_i, to branch upon. The feasibility of the new sub-problems $(RSQIP_j^1)$ and $(RSQIP_j^2)$ with the corresponding feasible sets $\mathcal{F}_j^1 = \mathcal{F}_j \cap \{x_i = 0\}$ and $\mathcal{F}_j^2 = \mathcal{F}_j \cap \{x_i = 1\}$ are then checked. A sub-problem is fathomed if it is not feasible. This type of fathoming occurs after a certain number of iteration. The unfathomed sub-problem is now added to L.

The resulting new sub-problems $(RSQIP_j^1)$ and $(RSQIP_j^2)$ are defined as follows:

$$(RSQIP_j^1) \qquad \min \qquad x^T S x, \qquad s.t$$

$$x \in \mathcal{F}_j^1,$$

and

$$(RSQIP_j^2) \qquad \min \qquad x^T S x, \qquad s.t$$

[1]This can be done in a number of ways which we have presented in the next subsection. In our implementation, we have adopted the most fractional variable strategy.

$$x \in \mathcal{F}_j^2.$$

The *Branch-and-Bound* stops when the set L becomes empty.

Some features within the *Branch-and-Bound* method are important in the sense that they facilitate or accelerate the convergence of the algorithm, namely the choice of the variable to branch upon and the choice of the sub-problem. These features deserve some explanations.

6.1.1 Choice of the branching variables

In the case of linear integer programming, it is known that the rule used to choose branching variables usually has an important effect on the performance of the *Branch-and-Bound* method [BGG+71]. In this section, we discuss some selection rules for selecting the branching variable. Let $x = (x_1, \dots, x_{n^2})$ be the optimal solution of the continuous relaxation of the sub-problem $(RSQIP_j)$ at a node j. Let $I(RSQIP_j) \subset \left\{1, \dots, n^2\right\}$ denotes the index set of fractional variables in x.

1) **The most fractional variable**

   After the continuous relaxation problem of a sub-problem is solved, the weight, ω_i, associated with the variable $x_i, i \in I(RSQIP_j)$ is calculated as $\omega_i = \min(\mid x_i \mid, \mid x_i - 1 \mid)$. This branching strategy selects the variable with the highest weight, i.e. the variable that is the farthest from its nearest integer value. This selection is aimed at getting the largest degradation of the objective when branching is carried out so that more nodes can be fathomed at early stages (see [LS10]).

2) **The lowest-index-first**

   In many situations, some decision variables x_is play more important roles in the model than others. Therefore, it is reasonable to branch variables in terms of their importance. The rule of lowest-index-first orders the index set $I(RSQIP_j)$ in the decreasing priorities[2] and selects the first variable in $I(RSQIP_j)$ to branch.

[2]These priorities are given according to the importance or the role of each variable in a given problem.

3) **Using a pseudo-cost**

In this strategy, each variable is given importance according to a pseudo-cost which allows a prioritization of all the variables. This concept was developed by Benichou et al. [BGG+71]. For each variable $x_i, i \in I(RSQIP_j)$, a lower pseudo-cost pcl_i and an upper pseudo-cost pcu_i are computed in the following way.

Suppose that at the node j, the variable x_i is selected for branching. The fractional part of the value of the variable x_i is still x_i since $x_i \in [0, 1]$. Let f_j denote the value of the objective function at this node j. Let f_l be the value of the objective function when the continuous relaxation problem is solved with fixing $x_i = 0$. The lower pseudo-cost of x_i is therefore given by $pcl_i = (f_l - f_j)/x_i$.

Let f_u be the value of the objective function when the continuous relaxation problem is solved with fixing $x_i = 1$. The upper pseudo-cost is given by: $pcu_i = (f_u - f_j)/(1 - x_i)$.

Although the values of the pseudo-costs depend on the node where they are computed, they are computed only once and are assumed to remain constant so that the computational effort of recomputing them at every node could be saved. This strategy of selecting the branching variable is invoked in the following manner:

– Calculate the lower and upper pseudo-costs for all the variables.

– Compute the quantity

$$V_i = \min(pcl_i x_i, pcu_i(1 - x_i)) \tag{6.1}$$

for each variable x_i.

– Select the variable x_i for which the value of V_i is maximum.

6.1.2 Selection of branching nodes

It has been found that the selection method for branching nodes significantly affect the performance of *Branch-and-Bound* as does the selection method for the branching variables. Here are some rules used to select branching nodes.

1) **Branch from the node with the lowest bound**

   The name of this branching strategy tells everything about itself. In this strategy, the node which currently has the lowest bound on the objective function is selected for branching.

2) **Branch from the newest node**

   In this recency-based branching strategy, whenever a branching is carried out, the nodes corresponding to the new problems are given preference over the rest of the unfathomed nodes. The node that is the newest in the list of unfathomed sub-problems is selected for branching. This strategy is also known as *depth-first strategy*. It has the advantage of saving storage space.

3) **Branch from an estimation**

   At a node j, the pseudo-costs V_i for $i \in \{1, \ldots, n^2\}$ defined in (6.1) are added to the lower bound f_j to form an estimation of the best objective function value for the descendants of node j i.e.

$$E_j = f_j + \sum_{i=1}^{n^2} V_i.$$

   The quantity E_j is computed for all the unfathomed nodes. The node with the lowest value of E_j is selected for branching. We have also selected one sub-problem at each iteration k i.e $s = 1$.

Remark 6.1.1:
Benichou et al. [BGG+71] give a deeper study and experimental comparison of all these branching strategies. However, in our numerical implementation, we have used the strategy of the most fractional variable as branching variable criteria, and the strategy of branching from the newest node as selection of branching node rule. We have also selected one sub-problem at each iteration, i.e $s = 1$.

Algorithm 2 : The *Branch-and-Bound* algorithm

Step 1: (Initialization). Set the sub-problems list $L = \{(RSQIP)\}$. Set an initial feasible solution as the incumbent solution x^* and $f^* = f(x^*)$.

Step 2: (Node selection). If $L = \emptyset$, stop and x^* is the optimal solution to $(RSQIP)$. Otherwise, choose one or more nodes from L. Denote the set of s selected sub-problems by $L^s = \{(RSQIP_1), \ldots, (RSQIP_s)\}$. Let $L := L \setminus L^s$. Set $j = 1$.

Step 3: (Bounding). Compute a lower bound LB_j of sub-problem $(RSQIP_j)$. Set $LB_j = +\infty$ if $(RSQIP_j)$ is infeasible. If $LB_j \geq f^*$, go to ***Step 6***.

Step 4: (Feasible solution). Save the feasible solution found in ***Step 3*** or generate a better feasible solution when possible[3]. Update the incumbent x^* and f^* if needed. Remove from L^s all $(RSQIP_r)$ that are infeasible, $1 \leq r \leq j$. If $j < s$, set $j := j + 1$ and return to ***Step 3***. Otherwise, go to ***Step 5***.

Step 5: (Branching). If $L^s = \emptyset$, go to ***Step 2***. Otherwise, choose a node $(RSQIP_j)$ from L^s. Further divide $\mathcal{F}_j$ into smaller subsets: $L_j^s = \left\{\mathcal{F}_j^1, \mathcal{F}_j^2\right\}$. Remove $(RSQIP_j)$ from L^s and set $L := L \cup L^s \cup L_j^S$. Go to ***Step 2***.

Step 6: (Fathoming). Remove $(RSQIP_j)$ from L^s. If $j < s$, set $j := j + 1$ and return to ***Step 3***. Otherwise, go to ***Step 4***.

6.1.3 The *Branch-and-Bound* algorithm

We have presented above the different components of the *Branch-and-Bound* method. We now present, in **Algorithm 2**, a step-by-step description of the *Branch-and-Bound* algorithm implemented in this dissertation.

A flowchart detailing the steps of **Algorithm 2** is given in Figure 6.1.3. Further detailed discussions on the general *Branch-and-Bound* method can be found in [LS10].

[3]This can be generated using a heuristic method.

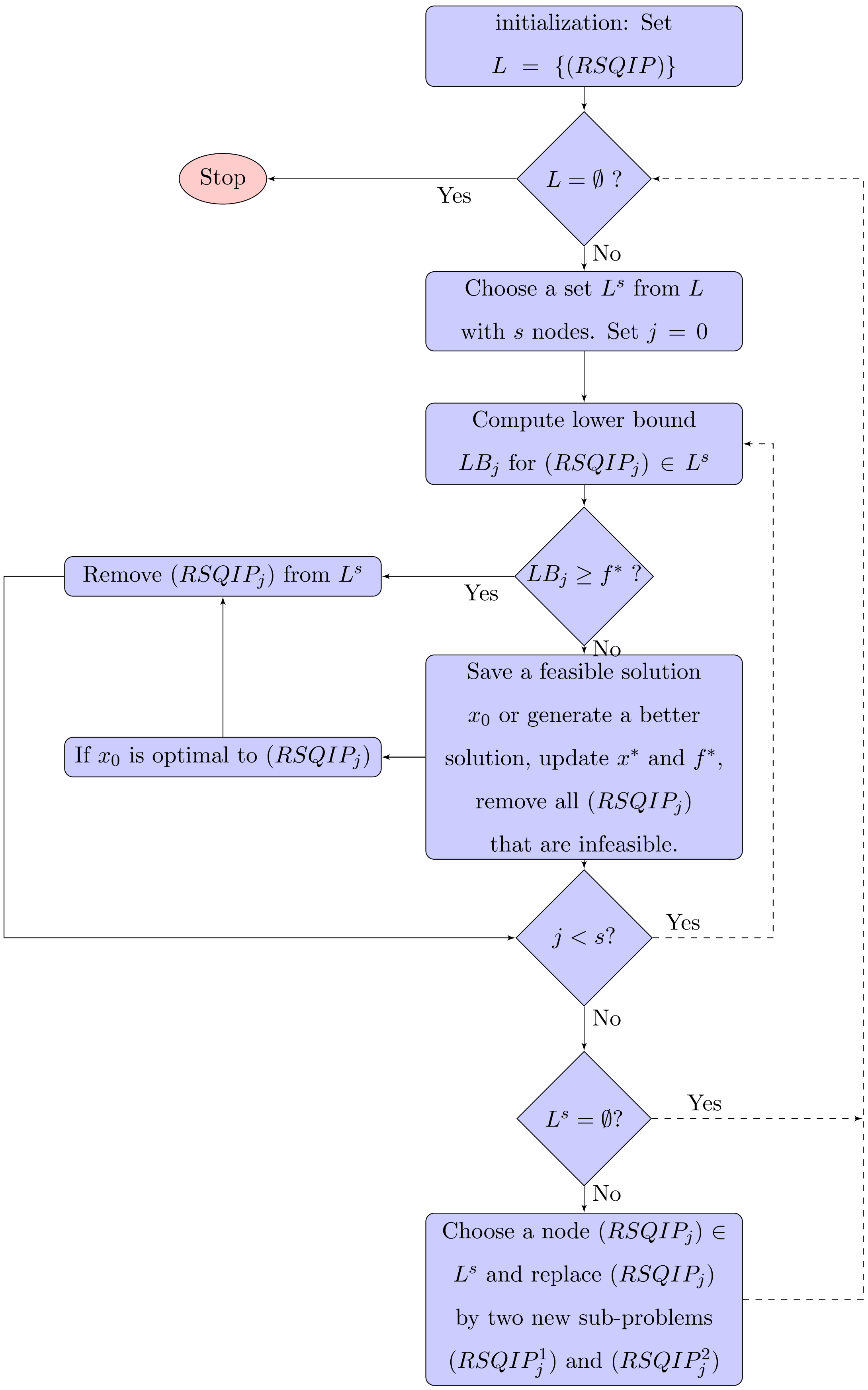

initialization: Set $L = \{(RSQIP)\}$
$L = \emptyset$?
Yes
Stop
No
Choose a set L^s from L with s nodes. Set $j = 0$
Compute lower bound LB_j for $(RSQIP_j) \in L^s$
$LB_j \geq f^*$?
Yes
Remove $(RSQIP_j)$ from L^s
No
Save a feasible solution x_0 or generate a better solution, update x^* and f^*, remove all $(RSQIP_j)$ that are infeasible.
If x_0 is optimal to $(RSQIP_j)$
$j < s$?
Yes
No
$L^s = \emptyset$?
Yes
No
Choose a node $(RSQIP_j) \in L^s$ and replace $(RSQIP_j)$ by two new sub-problems $(RSQIP_j^1)$ and $(RSQIP_j^2)$

Figure 6.1: The *Branch-and-Bound* flow chart

6.2 An auxiliary function-based dynamic convexized method

In this section, we discuss an auxiliary function-based method for the QAP which is inspired by the discrete dynamic convexized method of Zhu and Ali [ZA09]. This method consists of an auxiliary function which is equivalent to the objective function of $(RSQIP)$. This auxiliary function is then sequentially minimized using a local search algorithm. This method has the ability of escaping from local optimal solutions. In our approach, we define a neighbourhood structure and an auxiliary function, which are different from the ones presented in [ZA09]. Given the neighbourhood structure, we present a simple local optimization algorithm which we have implemented in the numerical section. Full details of the local search algorithm are presented in **Algorithm 3**. The auxiliary function as well as details of the theoretical results presented in this dissertation conform to the problem we consider for our study.

Remark 6.2.1:
In this section we mostly deal with the permutation matrices. We establish the link between the variable x and the permutation matrix $X \in \mathcal{X}_n$ by setting $x = vec(X)$. Due to this link, we will sometimes write $x \in \mathcal{X}_n$ to mean that $x = vec(X)$ with $X \in \mathcal{X}_n$.

Definition 6.2.2 (Neighbouring permutation matrix)**:**
Let X be a permutation matrix, we define X_{ij} to be the permutation matrix obtained by swapping the columns i and j of X for $i, j = 1, \dots, n$.

Definition 6.2.3 (Neighbourhood of a permutation matrix)**:**
For any permutation matrix $X \in \mathcal{X}_n$, the neighbourhood, $N(X)$, of X is a set $N(X) \subseteq \mathcal{X}_n$ such that $N(X) = \{X, X_{ij} : \forall i, j, i \neq j\} = \{X\} \cup \{X_{ij} : \forall i, j, i \neq j\}$.

Example 6.2.4:
For n=4, if X is the permutation matrix corresponding to the permutation $\phi = (2, 3, 1, 4)$ then $N(X)$ is defined as

$$N(X) = \{(2,3,1,4), (3,2,1,4), (1,3,2,4), (4,3,1,2), (2,4,1,3), (2,1,3,4), (2,3,4,1)\}.$$

Definition 6.2.5 (Local minimizer)**:**

A permutation matrix $X_0 \in \mathcal{X}_n$ is called local minimizer of $f(x) = x^T Sx$ over $\mathcal{X}_n$ if $f(x) \geq f(x_0)$ for all $x \in N(x_0) \cap \mathcal{X}_n$.

Definition 6.2.6 (Local search)**:**

A local search is an algorithm which can identify the local minimizer associated with X_0 within $N(X_0)$, where X_0 is an initial solution.

Algorithm 3 illustrates a local search procedure.

Algorithm 3 : Local search algorithm

```
Step 1: Take an initial permutation matrix $X_0 \in \mathcal{X}_n$.
Step 2: If $x_0$ is a local minimizer of $f(x)$ over $\mathcal{X}_n$ then stop else take a permutation
matrix $X \in N(X_0)$ such that $f(x) < f(x_0)$.
  Step 3: Let $X_0 := X$ and go to Step 2.
```

6.2.1 The auxiliary function and its properties

The auxiliary function used in the dynamic convexized method is based on the current best known minimizer of $(RSQIP)$. Let x_c^* be the current best minimizer of $(RSQIP)$ and let $f_c^* = f(x_c^*)$ be its objective function value. We define the following auxiliary function:

$$T(x, \lambda | x_c^*) = \begin{cases} f(x) - f_c^* + \lambda \|x - x_c^*\| & \text{if } f(x) \geq f_c^*, \\ f(x) - f_c^* & \text{otherwise,} \end{cases} \tag{6.2}$$

with λ being a non-negative parameter and $\|.\|$ designating the p-norm, $p = 1, 2$, see **Definition 2.2.17**. The auxiliary function is updated within the dynamic convexized method as soon as a new minimizer is found which is better than x_c^*.

We can easily notice that if $f(x) \geq f_c^*$, then $T(x, \lambda | x_c^*) \geq 0$, otherwise $T(x, \lambda | x_c^*) < 0$. Using the above function, we define the following auxiliary non-linear integer programming problem:

$$(ANLIP) \qquad \begin{array}{ll} \min & T(x, \lambda|x_c^*), \qquad s.t \\ & X \in \mathcal{X}_n. \end{array}$$

The ideas of this method was first introduced by Zhu and Ali [ZA09] and it aims to find sequentially local minimizers $\{x_1^*, x_2^*, \ldots, x_c^*, \ldots\}$, of $(RSQIP)$ such that $f(x_i^*) \geq f(x_{i+1}^*)$ by solving $(ANLIP)$ based on updated $T(x, \lambda|x_i^*)$. Before we present theoretical results on the auxiliary function-based method, we need to define the following sets:

$$\mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f_c^*\},$$

$$\mathcal{S}_2 = \{x \in \mathcal{X}_n : f(x) > f_c^*\},$$

$$\mathcal{S}_3 = \{x \in \mathcal{X}_n : f(x) = f_c^*\}.$$

Theorem 6.2.7:

If x_c^* is a local minimizer of $(RSQIP)$, then x_c^* is a local minimizer of $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$.

Proof. Let us assume that x_c^* is a local minimizer of $(RSQIP)$. Therefore, $f_c^* \leq f(x)$ for all $x \in N(x_c^*) \cap \mathcal{X}_n$. We want to show that $T(x, \lambda|x_c^*) \geq T(x_c^*, \lambda|x_c^*)$ for all $x \in N(x_c^*) \cap \mathcal{X}_n$.

Let $x \in N(x_c^*) \cap \mathcal{X}_n$, since x_c^* is a local minimizer of $(RSQIP)$ then $f(x) \geq f_c^*$. Therefore $T(x, \lambda|x_c^*) \geq 0$ i.e. $T(x, \lambda|x_c^*) \geq T(x_c^*, \lambda|x_c^*)$ since $T(x_c^*, \lambda|x_c^*) = 0$. Hence x_c^* is a local minimizer of $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$. □

Theorem 6.2.8:

For all $x \in \mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f_c^*\}$ and for all $y \in \mathcal{S}_2 \cup \mathcal{S}_3$, we have $T(x, \lambda|x_c^*) < T(y, \lambda|x_c^*)$.

Proof. Let $x \in \mathcal{S}_1$ and $y \in \mathcal{S}_2 \cup \mathcal{S}_3$, $T(x, \lambda|x_c^*) < 0$ and $T(y, \lambda|x_c^*) \geq 0$. Hence, **Theorem** 6.2.8 obviously holds. □

From **Theorem** 6.2.8 we can deduce the following result.

Corollary 6.2.9:

If f_c^* is not the global minimal value of $(RSQIP)$, then $\mathcal{S}_1 = \{x \in \mathcal{X}_n | f(x) < f_c^*\} \neq \emptyset$ and all global minimizers of $(ANLIP)$ are in the set $\mathcal{S}_1$.

Theorem 6.2.10:

Suppose that f_c^* is not the global minimal value of $(RSQIP)$ then for $y \in \mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f_c^*\}$, y is a local minimizer of $(ANLIP)$ if and only if y is a local minimizer of $(RSQIP)$.

Proof. Let us suppose that f_c^* is not the global minimal value of $(RSQIP)$, then by **Corollary** 6.2.9 $\mathcal{S}_1 \neq \emptyset$. Let $y \in \mathcal{S}_1$ such that y is a local minimizer of $(ANLIP)$, i.e. $T(y, \lambda | x_c^*) \leq T(x, \lambda | x_c^*) \qquad \forall x \in N(y) \cap \mathcal{X}_n$.

This is equivalent to $f(y) - f_c^* \leq f(x) - f_c^*$ for all $x \in N(y) \cap \mathcal{X}_n$

i.e. $f(y) \leq f(x)$ for all $x \in N(y) \cap \mathcal{X}_n$.

Hence y is a local minimizer of $(RSQIP)$. The reverse proof follows easily. □

By **Corollary** 6.2.9 and **Theorem** 6.2.10, we can find that if f_c^* is not the global minimal value of $(RSQIP)$, then $(RSQIP)$ and $(ANLIP)$ have the same global minimizer.

We now look at some properties of the auxiliary function.

Lemma 6.2.11:

For any $x \in \mathcal{X}_n$, if $x \neq x_c^*$, then there exists $y \in N(x) \cap \mathcal{X}_n$ such that $\|y - x_c^*\| < \|x - x_c^*\|$.

Proof. Let X be a permutation matrix corresponding to x such that $X \neq X_c^*$, where X_c^* is the permutation matrix corresponding to x_c^*. Therefore, there are some corresponding columns of both matrices that are not equal. Since X and X_c^* are permutation matrices, there are at least two columns by which they differ. Let i and j be the columns in the matrix X which are not equal to their corresponding columns in X_c^*. By swapping the columns i and j in X, we obtain a permutation matrix $Y = X_{ij}$ such that the column i or the column j is equal to its corresponding column in the matrix X_c^*. Consequently, the matrix $Y - X_c^*$

has less non-zero elements than the matrix $X - X_c^*$. Hence $\|y - x_c^*\| < \|x - x_c^*\|$ and **Lemma** 6.2.11 holds. □

Theorem 6.2.12:

For the function $T(x, \lambda | x_c^*)$, we have the following results:

i) For any $x \in \mathcal{S}_2 \cup \mathcal{S}_3$, if there exists $y \in N(x) \cap \mathcal{X}_n$ such that $f(y) < f_c^*$ then x is not a local minimizer of $(ANLIP)$.

ii) For any $x \in \mathcal{S}_2 \cup \mathcal{S}_3, x \neq x_c^*$ let

$$L(x) = \min_z \left\{ \frac{f(z) - f(x)}{\|x - x_c^*\| - \|z - x_c^*\|} \mid z \in N(x) \cap \mathcal{X}_n, \|z - x_c^*\| < \|x - x_c^*\| \right\}. \quad (6.3)$$

If $\lambda > L(x)$, then x is not a local minimizer of $(ANLIP)$.

iii) Especially if

$$\lambda > \max \{L(x) : x \in \mathcal{X}_n\} \quad (6.4)$$

then for all $x \in \mathcal{S}_2 \cup \mathcal{S}_3, x \neq x_c^*$, is not a local minimizer of $(ANLIP)$.

Proof. *i)* Let $x \in \mathcal{S}_2 \cup \mathcal{S}_3$ and let us suppose that there exists $y \in N(x) \cap \mathcal{X}_n$ such that $f(y) < f_c^*$. $x \in \mathcal{S}_2 \cup \mathcal{S}_3$ implies that $T(x, \lambda | x_c^*) = f(x) - f_c^* + \lambda \|x - x_c^*\| \geq 0$ on the other hand we $f(y) < f_c^*$, it implies that $T(y, \lambda | x_c^*) = f(y) - f_c^* < 0$. Hence $T(y, \lambda | x_c^*) < T(x, \lambda | x_c^*)$ with $y \in N(x) \cap \mathcal{X}_n$. Whence x is not a local minimizer for $(ANLIP)$.

ii) We have

$$L(x) = \min_z \left\{ \frac{f(z) - f(x)}{\|x - x_c^*\| - \|z - x_c^*\|} \mid z \in N(x) \cap \mathcal{X}_n, \|z - x_c^*\| < \|x - x_c^*\| \right\}$$

i.e. there exists $z \in N(x) \cap \mathcal{X}_n$ with $\|z - x_c^*\| < \|x - x_c^*\|$ such that

$$L(x) = \frac{f(z) - f(x)}{\|x - x_c^*\| - \|z - x_c^*\|}.$$

$$\begin{aligned}
\text{Now, } \lambda > L(x) \implies & \qquad \lambda\left(\|x - x_c^*\| - \|z - x_c^*\|\right) > f(z) - f(x) \\
\textit{i.e.} & \qquad \lambda\left(\|x - x_c^*\| - \|z - x_c^*\|\right) > (f(z) - f_c^*) - (f(x) - f_c^*) \\
\textit{i.e.} & \qquad \lambda\|x - x_c^*\| + f(x) - f_c^* > f(z) - f_c^* + \lambda\|z - x_c^*\| \\
\textit{i.e.} & \qquad T(x, \lambda|x_c^*) > f(z) - f_c^* + \lambda\|z - x_c^*\|.
\end{aligned}$$

So if $f(z) < f_c^*$, we have $T(z, \lambda|x_c^*) = f(z) - f_c^* \leq f(z) - f_c^* + \lambda\|z - x_c^*\| < T(x, \lambda|x_c^*)$. Otherwise $T(z, \lambda|x_c^*) = f(z) - f_c^* + \lambda\|z - x_c^*\| < T(x, \lambda|x_c^*)$.

Thus in any case, $T(x, \lambda|x_c^*) > T(z, \lambda|x_c^*)$ with $z \in N(x) \cap \mathcal{X}_n$. Hence x is not a local minimizer of $(ANLIP)$.

iii) Let us suppose that equation (6.4) holds, by *ii)* we know that if $\lambda > L(x)$ for any $x \in \mathcal{X}_n, x \neq x_c^*$, x is not a local minimizer of $(ANLIP)$. This will therefore be valid for $\lambda > \max_{x \in \mathcal{X}_n} L(x)$. Hence x is not a local minimizer of $(ANLIP)$.

□

By **Theorem** 6.2.12, we can notice that if the minimization of $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$ using the local search algorithm gets stuck at a local minimizer in the set $\mathcal{S}_2 \cup \mathcal{S}_3$, then by a sufficient increment on the value of λ, the minimization of $T(x, \lambda|x_c^*)$ can escape from the local minimizer. Moreover, by Theorems 6.2.7 and 6.2.12, if λ is large enough, the minimization of $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$ starting from any permutation matrix will converge either to the prefixed point x_c^* or to a local minimizer in $\mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f_c^*\}$.

Let $h(x) = \max\{0, f(x) - f_c^*\}$ be defined over $\mathcal{X}_n$. It is important to note that the objective of minimizing $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$ is to find a point $x \in \mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f_c^*\}$ which satisfies $h(x) = 0$. However if the set $\mathcal{S}_1$ is small, it is difficult to find such a point. Therefore, most of the efforts will be spent searching in the set $\mathcal{S}_2 \cup \mathcal{S}_3$. While minimizing $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$, for two points x and y in $\mathcal{S}_2 \cup \mathcal{S}_3, x \in N(y)$, if $h(x) < h(y)$, we would like to have

$T(x,\lambda|x_c^*) < T(y,\lambda|x_c^*)$. However, by assertion *ii)* of **Theorem** 6.2.12, if the value of λ is large enough so as to satisfy inequality (6.4), then we will have $T(x,\lambda|x_c^*) \geq T(y,\lambda|x_c^*)$ when $\|x - x_c^*\| > \|y - x_c^*\|$ and this will obviously misconduct the search for a good point.

Theorem 6.2.13:

Let $h(x) = \max\{0, f(x) - f_c^*\}$. Suppose $x \in \mathcal{S}_2 \cup \mathcal{S}_3$ and $h(z) < h(x)$, $z \in N(x) \cap \mathcal{X}_n$. Then $T(z,\lambda|x_c^*) < T(x,\lambda|x_c^*)$ if and only if one of the following conditions is satisfied:

a) $\lambda = 0$,

b) $\lambda > 0$ *and* $\|z - x_c^*\| \leq \|x - x_c^*\|$,

c) $\lambda > 0$, $\|z - x_c^*\| < \|x - x_c^*\|$ *and* $\lambda < \dfrac{h(x) - h(z)}{\|z - x_c^*\| - \|x - x_c^*\|}$.

Proof. Let $x \in \mathcal{S}_2 \cup \mathcal{S}_3, z \in N(x) \cap \mathcal{X}_n, z \in \mathcal{S}_2 \cup \mathcal{S}_3$ and $h(z) < h(x)$.

$T(z,\lambda|x_c^*) < T(x,\lambda|x_c^*)$ is equivalent to $f(z) - f_c^* + \lambda\|z - x_c^*\| < f(x) - f_c^* + \lambda\|x - x_c^*\|$ and we have $h(z) < h(x)$. So

a) $f(z) - f_c^* + \lambda\|z - x_c^*\| < f(x) - f_c^* + \lambda\|x - x_c^*\|$ holds if and only if $\lambda = 0$.

b) $\lambda > 0$ and $\|z - x_c^*\| \leq \|x - x_c^*\|$ is equivalent to $\lambda\|z - x_c^*\| \leq \lambda\|x - x_c^*\|$ and this combined with $h(z) < h(x)$ leads to $T(z,\lambda|x_c^*) < T(x,\lambda|x_c^*)$.

c) Since, $\lambda > 0$, $\|z - x_c^*\| < \|x - x_c^*\|$ and $\lambda < \dfrac{h(x) - h(z)}{\|z - x_c^*\| - \|x - x_c^*\|}$, we have $\lambda < \dfrac{f(x) - f(z)}{\|z - x_c^*\| - \|x - x_c^*\|} \Longleftrightarrow \lambda\|z - x_c^*\| - \lambda\|x - x_c^*\| < (f(x) - f_c^*) - (f(z) - f_c^*)$
i.e. $f(z) - f_c^* - \lambda\|z - x_c^*\| < f(x) - f_c^* + \lambda\|x - x_c^*\|$
Hence $T(z,\lambda|x_c^*) < T(x,\lambda|x_c^*)$.

□

Theorem 6.2.13 helps us in making $T(x,\lambda|x_c^*) < T(y,\lambda|x_c^*)$ while $h(x) < h(y)$.

6.2.2 The discrete dynamic convexized (DDC) algorithm

Here we present the DDC algorithm. The DDC algorithm solves $(RSQIP)$ by repeatedly solving $(ANLIP)$. This is because the auxiliary function $T(x, \lambda|x_c^*)$ in $(ANLIP)$ is updated dynamically with new found minimizers of $(RSQIP)$. Parameters involved in the DDC algorithm are λ which is a non-negative parameter used to define the auxiliary function. We start the algorithm with $\lambda = 0$ and increase its value by δ_λ when necessary until the condition[4] $c)$ given in **Theorem 6.2.13** is satisfied. A numerical study by Zhu and Ali [ZA09] shows that δ_λ can be chosen between 0.5 and 5, for our numerical implementation, we chose $\delta_\lambda = 1$. The DDC algorithm also has a parameter N_L which represents the maximum number of iterations. The choice of the value for N_L is discussed in the numerical implementation section. We denote the iteration of DDC by k.

Before presenting a step-by-step description of DDC, we present the basic mechanism of the DDC algorithm. Initially, an initial solution, say x^0, is generated using the heuristic method described in section 7.2. A local minimizer, x_c^*, of $(RSQIP)$ is then found starting from x^0 by **Algorithm 3**. We denote the local search by $Loc(\bullet)$. The pair x_c^*, f_c^* is then used to construct the auxiliary function, and the DDC algorithm starts with the following initial step, **Step 0**.

- **Step 0**: At this step of DDC, a random[5] initial solution $x^0 \in \mathcal{X}_n$ is found and is used to minimized $T(x, \lambda|x_c^*)$ with $\lambda = 0$. The local minimization of $T(x, \lambda|x_c^*)$ from x^0 may result in the minimizer $y^0 = Loc(x^0)$ landing in one of the following sets.

$$\mathcal{S}_1 = \{x \in \mathcal{X}_n : f(x) < f(x_c^*\}$$

$$\mathcal{S}_2 = \{x \in \mathcal{X}_n : f(x) > f(x_c^*\}$$

[4]This condition varies from one problem to another since it depends on the neighbourhood of the current incumbent solution x_c^*.

[5]In the numerical implementation, we found the initial x^0 by using the heuristic method described in section 7.2. The purpose of using the heuristic instead of simply generate a random initial solution is to facilitate the faster convergence of the DDC algorithm.

$$\mathcal{S}_3 = \{x \in \mathcal{X}_n : f(x) = f(x_c^*\}$$

Depending on which of the above sets, $\mathcal{S}_1, \mathcal{S}_2$ or $\mathcal{S}_3$ contains the resulting minimizer y^0, the following steps are executed:

i) If $y^0 \in \mathcal{S}_2$ then $Loc(\bullet)$ is applied with an increased value of λ. In particular, if $z^0 = Loc(y^0), z^0 \in \mathcal{S}_2$, then the process is repeated with z^0 i.e $r^0 = Loc(z^0)$ with a further increased value of λ. The process is repeated until the minimizer lands in $\mathcal{S}_1$ or $\mathcal{S}_3$.

ii) If $y^0 \in \mathcal{S}_3$ (or any minimizer in step i) lands in $\mathcal{S}_3$) then a new starting point x^0 (not using heuristic, as it may be time consuming) is found at random and the process is repeated from **Step 0** again.

iii) If $y^0 \in \mathcal{S}_1$ (or any minimizer in step i) or in step ii) lands in $\mathcal{S}_1$) then by **Theorem 6.2.10** we have found a local minimizer of $(RSQIP)$ which is better than x_c^*. x_c^*, f_c^* and the auxiliary function are then updated and the process begins with **Step 0** again.

With the above description, a step-by-step procedure of the DDC algorithm is given by **Algorithm 4**.

Algorithm 4 : DDC algorithm

```
***Step 1***: Let $x_c^*$ be an initial minimizer of $(RSQIP)$ and $f_c^* = f(x_c^*)$. Let $N_L$ be a sufficiently large integer number, and $\delta_\lambda$ be a positive number. Set $k = 0$
***Step 2***: Set $\lambda = 0$ and $k = k + 1$. If $k \geq N_L$ then go to ***Step 5***; Otherwise draw uniformly at random a permutation matrix $Y^0 \in \mathcal{X}_n$ set $y^0 = vec(Y^0)$ and go to ***Step 3***.
***Step 3***: Minimize $T(x, \lambda|x_c^*)$ over $\mathcal{X}_n$ from $y^0$ using the local search **Algorithm 3**. Suppose that $z^0$ is an obtained local minimizer i.e $z^0 = Loc(y^0)$
    If $z^0 \in \mathcal{S}_2$, then set $\lambda := \lambda + \delta_\lambda, y^0 = z^0$ and repeat ***Step 3***.
    If $z^0 \in \mathcal{S}_3$ then go to ***Step 2***.
    If $z^0 \in \mathcal{S}_1$ then go to ***Step 4***.
***Step 4***: Let $x_c^* = z^0, f_c^* = f(x_c^*)$ and go to ***Step 2***.
***Step 5***: Output $x_c^*$ and $f_c^*$ as an approximate global minimal solution and global minimal value of $(RSQIP)$.
```

7. Numerical experiments and results

In this Chapter, we present full details of the computational experiments. These are the calculation of lower bounds of QAP instances used within the *Branch-and-Bound* algorithm, the calculation of the first incumbent solution, used in both the *Branch-and-Bound* and the DDC algorithm, by a heuristic method. Finally, we present the implementation details of the *Branch-and-Bound* and the DDC algorithm. A full set of results are also presented and compared with a recent algorithm by Zhang et al. [ZRC10]. We have used more than 40 test problems of various degree of complexity for the *Branch-and-Bound* algorithm. We have used an additional set of 17 problems to test the DDC algorithm. Hence the total number of problems used is 57. These problems are taken from the QAP library [BcKR]. Programming was carried out using Matlab 7.11 on a desktop computer, Intel Core $i7$ with 3.07 GHz processor and 6.00 GB RAM. We begin with the computation of lower bounds.

7.1 Computation of lower bounds

In Chapter 4, we have presented two continuous relaxations, namely, $(CR1)$, the continuous relaxation of $(RSQIP)$ presented in section 4.1, and $(CR2)$, the continuous relaxation of $(SCQIP)$ presented in section 4.2. Here, we present the numerical calculation of lower bounds of a representative set of problem instances. We make a comparison of these lower bound calculations using $(CR1)$ and $(CR2)$ and show that the lower bounds obtained using $(CR1)$ are better than those obtained using $(CR2)$. The lower bounds obtained using $(CR1)$ are used within the *Branch-and-Bound*.

We have used an interior point method to solve $(CR1)$. This is available in the built-in Matlab routine, *QUADPROG*, for which we provided an initial strictly feasible solution[1] using the first technique of finding initial strictly feasible point developed in section 5.1.

[1]See the last paragraph on page 32, and the footnote 1 on page 42.

On the other hand, we used a predictor-corrector method for the solution of $(CR2)$ which is the continuous relaxation of $(SCQIP)$. Motivations for the use of the predictor-corrector are given in section 4.4.1. We have calculated a measure of goodness of the lower bounds using a gap. The gap is calculated as $gap = \frac{\mid f - f^* \mid}{\mid f^* \mid} \times 100$, where f is the lower obtained, and f^* is the exact or best known optimal function value.

The results are presented in Table 7.1, where column 1 contains the problem name, column 2 the problem size, column 3 the optimal solution of the problem. Columns 4, 5 and 6 contain the lower bound obtained from solving $(CR1)$, the corresponding gap and the times taken to compute these lower bounds respectively. Finally column 8 reports the gap obtained from the solutions of $(CR2)$.

A comparison of the gaps obtained from the two lower bounding techniques (columns 5 and 6 of Table 7.1) shows that the lower bounds based on $(CR2)$ are inferior and this cannot be used for the *Branch-and-Bound* method. This is due to the convexification technique used to transform $(RSQIP)$ into $(CQIP)$.

Table 7.1: Lower Bounds from the continuous relaxation of $(RSQIP)$

Instances	Sizes	Opt	$(CR1)$ Lower bounds	$(CR1)$ Gap (%)	CPU times (s)	$(CR2)$ Gap (%)
Esc16a	16	68	65.03	4.3	2.09	91.3
Esc16b	16	292	268.8	7.9	2.16	927
Esc16c	16	160	143.2	10.5	2.31	91.3
Esc16d	16	16	12.5	21.8	2.15	91.2
Esc16e	16	28	27.8	0.7	2.13	91.7
Esc16f	16	0	0	0.0	0.87	0.0
Esc16g	16	26	25.3	14.2	2.10	91.5
Esc16i	16	14	11.7	16.4	2.10	90.4
Esc16j	16	8	8	0.0	2.19	91.7
Had12	12	1652	1602.6	2.9	1.73	87.2
Had14	14	2724	2675.5	1.7	3.47	89.3
had16	16	3720	3647.8	1.9	8.22	90.2
Had18	18	5358	5283.7	1.3	18.73	91.4
Had20	20	6922	6845.6	1.1	35.43	92.3
Nug12	12	578	576.3	0.2	2.17	86.7
Nug14	14	1014	1009.1	0.4	3.28	89.4
Nug15	15	1150	1139.1	0.8	5.15	89.8
Nug16a	16	1610	1600.9	0.5	9.15	90.6
Nug16b	16	1240	1224.3	1.2	9.25	90.4
Nug17	17	1732	1716.3	0.9	9.31	91.0
Nug18	18	1930	1923.9	0.3	16.71	91.4
Nug20	20	2570	2551.7	0.7	43.25	92.3
Nug22	22	3596	3576.2	0.5	100.16	93.7

7.2 A heuristic random enumeration

During our discussion of the *Branch-and-Bound* method in section 6.1, we mentioned the calculation of the incumbent solution or the (starting) initial solution. We also mentioned the calculation of an initial (starting) solution used at the beginning of the DDC algorithm in section 6.2. The initial solution for both algorithms are generated by a random enumeration scheme. This solution is provided to both algorithms due to the experience we gathered during our numerical experiments.

Clearly, if the (initial) starting solution in the *Branch-and-Bound* is close enough to the optimal solution, then many branches will be pruned at early stages of *Branch-and-Bound* resulting in faster convergence. Similarly, our experiments with the DDC algorithm have shown that if we start the DDC algorithm with a randomly generated permutation matrix or the identity matrix, which is more likely to be far from the optimal solution, the probability of convergence is low. On the other hand, if the initial solution is good enough, then the convergence is more likely to be achieved after the suggested number of iterations of the DDC algorithm.

With the above motivation, we now present the details of calculation of the initial solution. This is a purely random strategy whereby the current permutation matrix (initially the identity matrix I_n is chosen) is changed iteratively by swapping two columns taken randomly. For example, we start with the matrix $I^0 = I_n$ and create the next matrix with I^1 from I^0 by swapping two randomly selected columns of I^0. Similarly, we create the permutation matrix I^{k+1} from I^k. Each time a permutation matrix is created, its function value is checked and the better solution is kept[2]. We have use the function $f(x) = \frac{1}{2}x^TQx$ of $(CQIP)$ for this purpose. The pseudo-code for the random enumeration is presented by **Algorithm 5**. The most important issue in the random enumeration procedure is the choice of the value N which is the number of randomly generated solutions. We now demonstrate

[2]One could use a local descent technique from the initial solution I_n, but our experiences have shown that the random enumeration is better due to its exploratory feature.

how we have chosen a value for N.

Our choice of N was based on the following numerical study. The problem presented in the QAP library [BcKR] have different structures. For some problem instances, there are pairs of facilities whose flow is zero. Since the cost coefficients in the objective function are proportional to the flows and distances, the cost associated with the zero flow coefficient will vanish. The structure of problem instances thus differ with the density[3] of the flow matrix, which is defined as the percent of the number of non-zero elements in this matrix. We took eight problems with different structure for our study. The remaining problem have similar structure to one of the eight problems considered.

For each of these eight problems, we have implemented the random enumeration algorithm, **Algorithm 5**, using $N = 10^5$ and study the improvement of the solution. We have presented[4] the optimal solution together with the evolving random heuristic solution for all the eight problems in Figures 7.1–7.4. In Figure 7.1–7.4,, the x-axes represent the N values and the y-axes the function values. These figures clearly show that the random enumeration solution does not necessarily improve with the increase of the value of N. This study helps us to choose N for different problems. Indeed for our numerical implementation, the values of N were randomly selected in the interval $[10^4, 10^5]$. Given that for a problem with $n = 12$, there are $12! = 479,001,600$ possibilities, these numbers are reasonably small.

7.3 Implementation of the *Branch-and-Bound* algorithm

In this section, we present the computational details of the *Branch-and-Bound* algorithm. For a faster convergence of the *Branch-and-Bound* algorithm, the lower bounding technique used should provide tight lower bounds, and its implementation should not be computationally expensive. In Chapter 4, we have presented two lower bounding techniques, via the continuous relaxation of $(RSQIP)$, $(CR1)$, and via the continuous relaxation of $(SCQIP)$,

[3] Zhang et al. [ZRC10] present the density flow matrix for different problem instances from [BcKR].
[4] The average optimal solution over 10 runs is also presented in appendix **??**.

Algorithm 5 : Random enumeration

```
Set $N$ to be a large number
Set the iteration counter, $k = 0$
Generate the permutation matrix $I^k$, $I^0 = I_n$ (identity matrix)
Set $x^s = vec(I^k)$
while $k \leq N$ do
    Generate a random permutation matrix $I^{k+1}$
    Set $x^0 = vec(I^{k+1})$
    if $\frac{1}{2}(x^0)^T Q x^0 < \frac{1}{2}(x^s)^T Q x^s$ then
        $x^s := x^0$
    end if
$k := k + 1$
end while
```

$(CR2)$. Motivated by the results presented in Table 7.1, we have used $(CR1)$ to compute the lower bounds of sub-problems within the *Branch-and-Bound* algorithm. Therefore, the lower bound which is the optimal solution of $(CR1)$ was obtained using the interior point algorithm available in the in-built Matlab solver *QUADPROG*. As we pointed out earlier, we provided initial strictly feasible solutions in *QUADPROG* using the first technique discussed in section 5.1.

At the beginning of the *Branch-and-Bound* algorithm, we implemented the heuristic random enumeration algorithm, **Algorithm 5** in order to get an initial incumbent solution, details of which have been presented in the previous section. The heuristic algorithm enables the *Branch-and-Bound* algorithm to discard some non promising branches at an early stage of the algorithm.

Using the *Branch-and-Bound* algorithm, we have been able to optimally solve up to 30 problem instances out of 40 tested problems, taken from the QAP library [BcKR], of size $n \leq 22$ on a single computer i.e without paralleling. Tables 7.2 and 7.3 present the results obtained from the *Branch-and-Bound* algorithm. In these tables, we report the optimizers obtained in the sixth column, together with corresponding objective function values in column 2. We also report the computational time and the number of function evaluations in columns 4 and 5 respectively. The total function calls includes the number of function calls used within the random enumeration algorithm and in the *Branch-and-Bound* algorithm. It is important to point out the fact that for some problem instances, the optimizers obtained from this algorithm are different from the those reported in [BcKR], See for instance problems Ecs16a, Ecs16b, Ecs16c, Ecs16d, Ecs16e, Ecs16g, Ecs16h, Ecs16i, Ecs16j, Had16, Nug12, Scr12 and Scr20. However, the optimal function values are the same.

We now compare our *Branch-and-Bound* with the *Branch-and-Bound* algorithm presented in [ZRC10]. This *Branch-and-Bound* uses parallel computing together with a powerful commercial solver to generate results. Hence this *Branch-and-Bound* algorithm solved problem instances with dimension up to 32. We have implemented our *Branch-and-Bound* using a personal computer and no commercial solver was used, and therefore we have presented our results of problem instances of dimension up to 22. We therefore compare these two *Branch-and-Bound* on the common problems that were solved by us and by Zhang et al. [ZRC10]. There were 11 common problems. On these problems, our *Branch-and-Bound* algorithm solved all successfully, and the *Branch-and-Bound* of Zhang et al. [ZRC10] solved 6 problems only.

7.4 Implementation of the auxiliary function-based method

In this section we present details of the numerical implementation of the DDC algorithm presented in Chapter 6. This algorithm starts with the heuristic random enumerative scheme, see *Algorithm 5*, to obtain an initial solution . This solution is then used as initial point to get the initial local minimizer x_c^* by the local search $Loc(\bullet)$. We have tested the DDC algorithm to a set of problem instances of size $n \leq 32$ and a problem of size $n = 64$. Given that the DDC algorithm solves problems instances of larger sizes than *Branch-and-*

Bound, for the purpose of comparison, we have used the 30 problems considered in [ZRC10], and solved them using DDC. In addition, we have also used another set of 27 problems to test the DDC algorithm. Therefore the total number of problems used to test the DDC algorithm is 57. These 57 problems also include the 40 used to test the *Branch-and-Bound* algorithm. Indeed, [ZRC10] provides recent advances in the QAP, and therefore, a platform for us to compare our results of the auxiliary function-based method, since it can solve larger problems within a reasonable amount of time.

In implementing the DDC algorithm, we started the algorithm with $\lambda = 0$ and increase its value by δ_λ when necessary until the condition c) given in **Theorem 6.2.13** is satisfied. We then reset $\lambda = 0$. As we said earlier, we chose to increase the value of λ by $\delta_\lambda = 1$. On the other hand, although the original dynamic convexized method [ZA09] used $N_L = 10^6$ to stop the algorithm, we have found that $N_L = 10^2$ is reasonable to use for the QAP considering the heuristic random enumeration that is implemented at the beginning of the DDC algorithm. In fact, $N_L = 10^2$ is large enough for our implementation as we have solved all the problems with this value.

The results obtained on 57 problem instances are presented in Table 7.4 and Table 7.5. In column 2 we give the deviation of our optimal function value from the exact or best known optimal function value for different problem instances. The minimizers corresponding to these function values are presented in Appendix 9.2. The sixth column provides the gap[5] obtained by Zhang et al. [ZRC10], where we use the symbol "–" for the problem instances for which they did not carry the experiment. Tables 7.4 and 7.5 show that the results by Zhang et al. [ZRC10] and DDC are comparable. Both algorithms were unable to optimally solve 13 problems. However, the gaps produced by DDC for the unsolved problems are much better than those reported in [ZRC10].

As we can see in Tables 7.4 and 7.5, the auxiliary function-based method has been able to achieve optimality for about 64% of the problems tested. The worst optimality gap we

[5]These gaps are the best of the gaps they obtained from their three different techniques.

could get is 19.9%. Only 14% of these tested problems have optimality gap greater than 5%.

Table 7.2: Results from the *Branch-and-Bound* algorithm: Part 1

Instances	Opt	Gap (%)	CPU time (s)	Func. Eval	Opt. Sol
Chr12a	9552	0	182.0	2000017	$(7, 5, 12, 2, 1, 3, 9, 11, 10, 6, 8, 4)$
Chr12b	9742	0	128.8	1000025	$(5, 7, 1, 10, 11, 3, 4, 2, 9, 6, 12, 8)$
Chr12c	11156	0	186.3	3000033	$(7, 5, 1, 3, 10, 4, 8, 6, 9, 11, 2, 12)$
Chr15a	10196	3.0	1928.1	1000641	$(13, 7, 5, 2, 1, 8, 14, 3, 4, 6, 9, 15, 12, 11, 10)$
Chr15b	8990	12.5	1363.1	1000683	$(9, 14, 3, 13, 5, 1, 4, 12, 6, 7, 2, 8, 11, 10, 15)$
Chr15c	10446	9.9	1499.9	6000079	$(2, 13, 7, 5, 1, 8, 6, 14, 4, 9, 15, 3, 12, 11, 10)$
Tai10a	135028	0	90.3	100021	$(9, 1, 8, 6, 10, 5, 4, 3, 7, 2)$
Tai10b	1183760	0	95.7	100117	$(5, 6, 1, 4, 7, 8, 9, 3, 2, 1)$
Tai12a	224416	0	179.7	2000018	$(8, 1, 6, 2, 11, 10, 3, 5, 9, 7, 12, 4)$
Tai12b	39464925	0	170.6	200009	$(9, 4, 6, 3, 11, 7, 12, 2, 8, 10, 1, 5)$
Tai15a	388988	0.2	1848.6	60000031	$(4, 7, 2, 11, 6, 10, 14, 9, 8, 13, 1, 3, 15, 12, 5)$
Tai15b	51934419	0.3	1666.8	60000101	$(13, 1, 4, 6, 5, 15, 7, 9, 12, 8, 14, 2, 11, 3, 10)$
Tai17a	508394	3.3	47947.5	60000095	$(5, 4, 17, 9, 10, 3, 1, 7, 15, 2, 6, 14, 13, 16, 12, 11, 8)$
Tai20b	123561767	0.9	178675.9	60000203	$(7, 6, 17, 8, 9, 16, 4, 2, 15, 19, 14, 11, 3, 1, 10, 13, 5, 20, 18, 12)$
Scr12	31410	0	104.5	200045	$(5, 7, 10, 11, 3, 12, 8, 4, 9, 6, 1, 2)$
Scr15	51140	0	569.5	6000017	$(15, 7, 11, 8, 1, 4, 3, 2, 12, 6, 13, 5, 14, 10, 9)$
Scr20	110030	0	186300.2	20000079	$(17, 6, 9, 7, 1, 5, 2, 3, 15, 10, 19, 12, 18, 14, 13, 20, 16, 8, 11, 4)$
Rou12	235528	0	111.1	200041	$(6, 5, 11, 9, 2, 8, 3, 1, 12, 7, 4, 10)$
Rou15	354210	0	1163.4	6000035	$(12, 6, 8, 13, 5, 3, 15, 2, 7, 1, 9, 10, 4, 14, 11)$
Rou20	743974	2.5	141603.5	6000101	$(9, 11, 7, 2, 10, 3, 19, 13, 6, 7, 1, 18, 20, 5, 15, 4, 12, 8, 14, 16)$

Table 7.3: Results from the *Branch-and-Bound* algorithm: Part 2

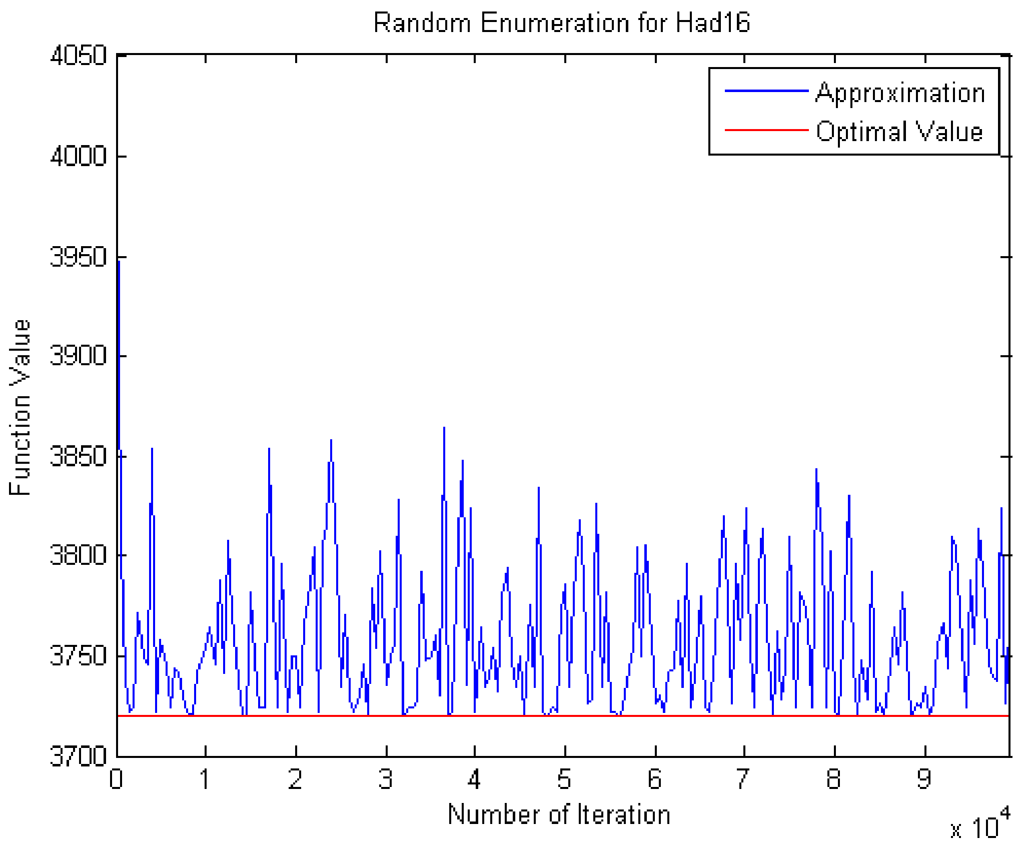
Instances	Opt	Gap (%)	CPU time (s)	Func. Eval	Opt. Sol
Nug12	578	0	168.3	100135	$(2, 10, 6, 5, 1, 11, 8, 4, 3, 9, 7, 12)$
Nug14	1014	0	476.3	6000039	$(9, 8, 13, 2, 1, 11, 7, 14, 3, 4, 12, 5, 6, 10)$
Nug16a	1610	0	272854.3	10000891	$(9, 14, 2, 15, 16, 3, 10, 12, 8, 11, 6, 5, 7, 1, 4, 13)$
Nug20	2570	0	282068.8	10001021	$(18, 14, 10, 3, 9, 4, 2, 12, 11, 16, 19, 15, 20, 8, 13,$
					$17, 5, 7, 1, 6)$
Nug22	3650	1.5	311118.2	60000152	$(17, 21, 9, 10, 1, 7, 19, 8, 16, 15, 20, 5, 2, 13, 6,$
					$12, 22, 18, 3, 11, 4, 14)$
Had12	1652	0	210.7	1000032	$(3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9)$
Had14	2724	0	727.5	1000185	$(8, 13, 5, 10, 12, 11, 2, 14, 3, 6, 7, 1, 9, 4)$
Had16	3720	0	169598.3	1000702	$(1, 4, 16, 15, 7, 8, 6, 11, 9, 14, 12, 5, 2, 13, 10, 3)$
Had18	5358	0	215832.2	6000101	$(8, 15, 16, 6, 7, 18, 14, 11, 1, 10, 12, 5, 13, 3, 2, 17, 9, 4)$
Had20	6922	0	334900.5	6000120	$(8, 15, 1, 14, 6, 19, 7, 11, 16, 12, 10, 17, 2, 20, 5, 3, 4, 9, 18, 13)$
Esc16a	68	0	1517.1	100049	$(3, 8, 16, 7, 2, 10, 1, 12, 11, 4, 14, 6, 5, 9, 15, 13)$
Esc16b	292	0	3421.8	100121	$(14, 4, 3, 5, 7, 6, 1, 2, 16, 12, 8, 13, 15, 10, 9, 11)$
Esc16c	160	0	355632.3	100621	$(7, 10, 14, 9, 13, 8, 6, 5, 1, 12, 3, 4, 2, 11, 16, 15)$
Esc16d	16	0	469422.3	100897	$(6, 7, 11, 16, 8, 4, 5, 10, 14, 2, 1, 13, 12, 9, 15, 3)$
Esc16e	28	0	3146.5	100083	$(4, 16, 12, 8, 5, 1, 3, 2, 6, 13, 15, 10, 7, 11, 14, 9)$
Esc16g	26	0	492.2	100012	$(12, 2, 1, 6, 4, 8, 7, 5, 3, 9, 13, 11, 14, 16, 15, 10)$
Esc16h	996	0	487612.8	100998	$(5, 15, 13, 6, 12, 10, 8, 14, 4, 16, 2, 3, 11, 9, 1, 7)$
Esc16i	14	0	32721.5	100502	$(2, 4, 10, 12, 16, 8, 7, 15, 11, 3, 13, 14, 6, 1, 9, 5)$
Esc16j	8	0	557.87	100012	$(3, 13, 2, 10, 12, 8, 16, 11, 5, 14, 6, 9, 4, 7, 15, 1)$

Table 7.4: Results from the auxiliary function-based method: Part 1

Instances	Opt	DDC Gap (%)	CPU time (s)	Func. Eval	ZBC Gap (%)
Had12	1652	0	83.8	549524	–
Had14	2724	0	147.2	750866	–
Had16	3720	0	628.7	1288724	–
Had18	5358	0	1024.1	2919844	–
Had20	6922	0	430.4	120820	–
Esc16a	68	0	99.6	256498	14.2
Esc16b	292	0	97.3	285490	97.3
Esc16c	160	0	88.5	240643	77.5
Esc16d	16	0	89.7	243814	0
Esc16e	28	0	92.1	266464	0
Esc16f	0	0	1	1	0
Esc16g	26	0	96.8	275977	0
Esc16h	996	0	110.2	295909	30.7
Esc16i	14	0	96.6	277336	–
Esc16j	8	0	105.6	291832	–
Esc32a	132	0	7484.4	62932864	100
Esc32b	192	14.2	5624.1	2491000	100
Esc32c	642	0	46451.2	13664832	100
Esc32d	206	3.0	5835.4	2491000	100
Esc32e	2	0	725.9	2617792	0
Esc32g	6	0	536.9	149604	0
Esc32h	438	0	16560.4	10116736	100
Esc64a	116	0	21254.1	11120480	–
Kra30a	91500	2.9	2151.6	1261600	65.7
Kra30b	95850	4.8	2331.4	2308000	68.3
Kra32	91760	3.4	5462.6	1517839	70.6
Bur26a	5439218	0.2	95.8	167600	–
Chr12a	9552	0	90.3	720020	–
Chr12b	9742	0	36.5	288500	–
Chr18a	11858	6.8	206.9	1046200	0
Chr18b	1534	0	243.1	1424000	0

Table 7.5: Results from the auxiliary function-based method: Part 2

Instances	Opt	DDC Gap (%)	CPU time (s)	Func. Eval	ZBC Gap (%)
Chr20a	2428	10.7	300.3	1073000	0
Chr20b	2470	7.4	417.3	1573000	0
Chr20c	16962	19.9	351.7	1246000	0
Chr22a	6538	6.2	380.3	5696000	0
Chr22b	6370	2.8	498.0	7960000	0
Chr25a	4382	15.4	213.0	582082	0
Tai10a	135028	0	9.8	100200	–
Tai10b	1183760	0	10.5	100200	–
Tai12a	224416	0	46.2	338000	–
Tai12b	39464925	0	36.2	293000	–
Scr12	31410	0	97.4	1135024	0
Scr15	51140	0	207.1	1975010	0
Scr20	110676	0.5	166.7	296000	6.2
Nug12	578	0	110.4	587378	–
Nug14	1014	0	98.7	199910	–
Nug15	1150	0	104.9	297211	–
Nug16a	1612	0.1	2192.6	43851648	–
Nug16b	1252	0.9	2254.1	18934976	–
Nug20	2570	0	1957.3	4708000	–
Nug21	2438	0	17858.1	11436142	–
Nug27	5306	1.3	21205.7	10940000	–
Nug28	5208	0.8	22505.7	7348603	–
Nug30	6180	0.9	10969.3	7220000	–


Random Enumeration for Had16
Approximation
Optimal Value
Function Value
Number of Iteration
x 10^4
4050
4000
3950
3900
3850
3800
3750
3700
0
1
2
3
4
5
6
7
8
9

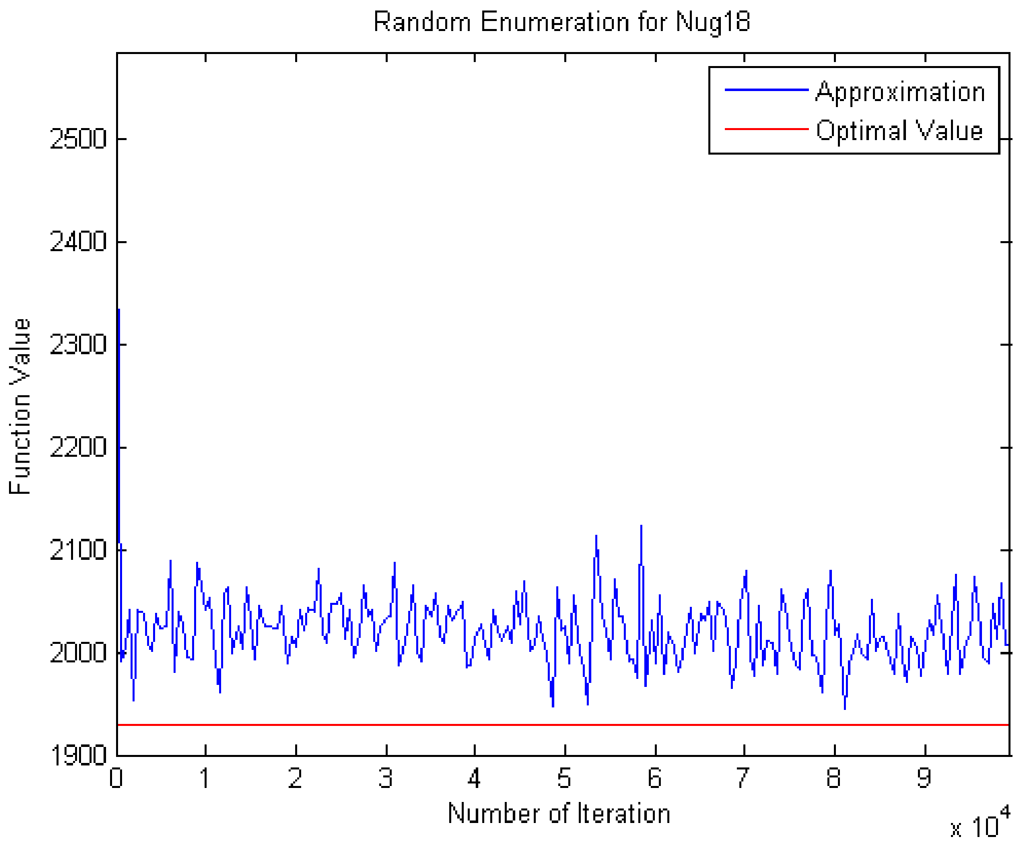
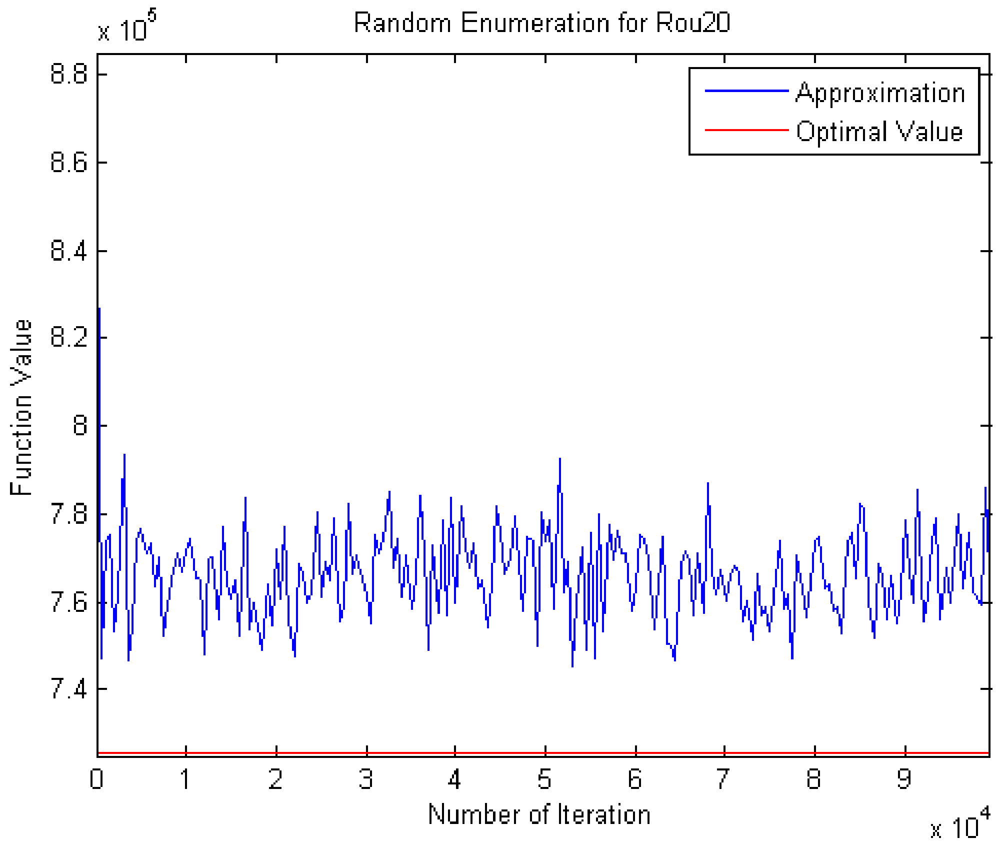

Random Enumeration for Nug18
Approximation
Optimal Value
Function Value
Number of Iteration
x 10^4
2500
2400
2300
2200
2100
2000
1900
0
1
2
3
4
5
6
7
8
9

Figure 7.1:


Random Enumeration for Rou20
x 10^5
Approximation
Optimal Value
8.8
8.6
8.4
8.2
8
7.8
7.6
7.4
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
x 10^4

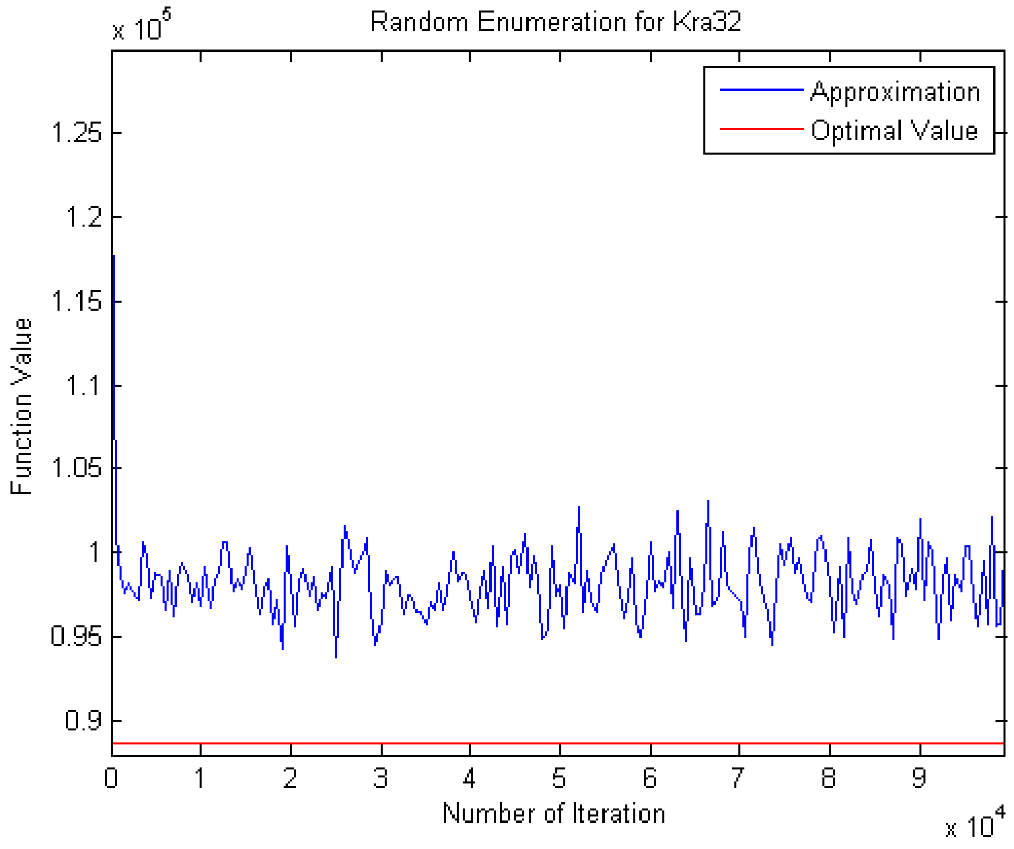
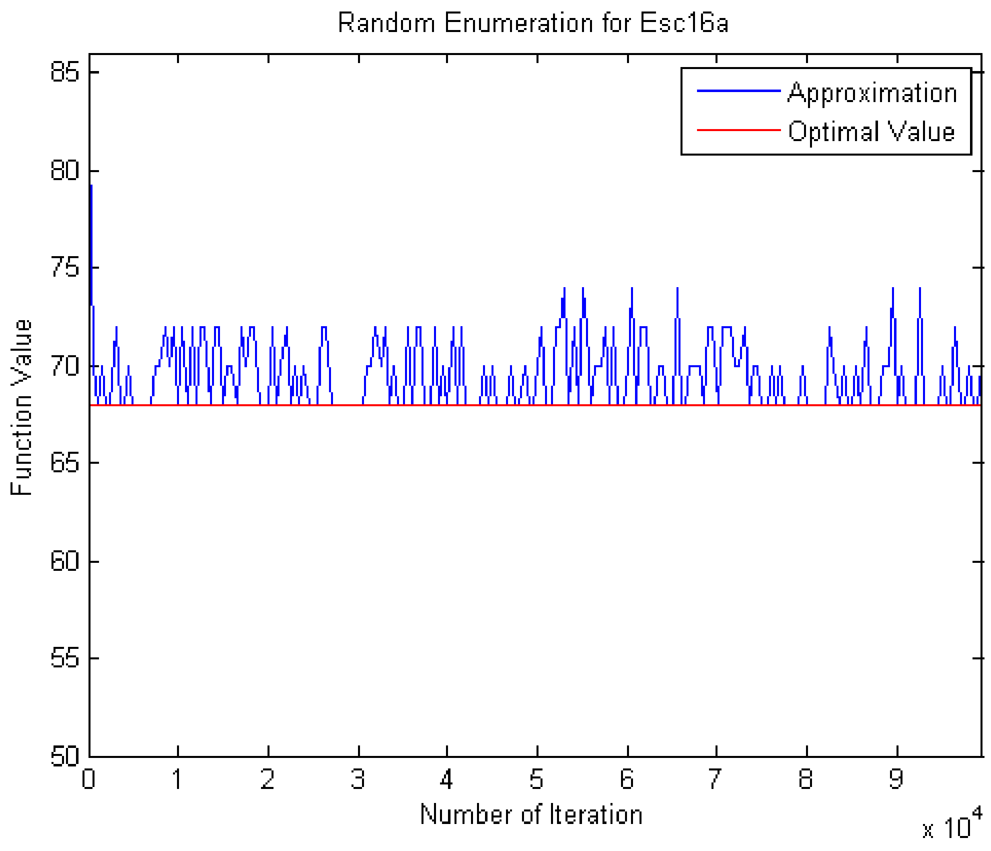

Random Enumeration for Kra32
x 10^5
Approximation
Optimal Value
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
x 10^4

Figure 7.2:


Random Enumeration for Esc16a
Approximation
Optimal Value
Function Value
Number of Iteration
x 10^4
85
80
75
70
65
60
55
50
0
1
2
3
4
5
6
7
8
9

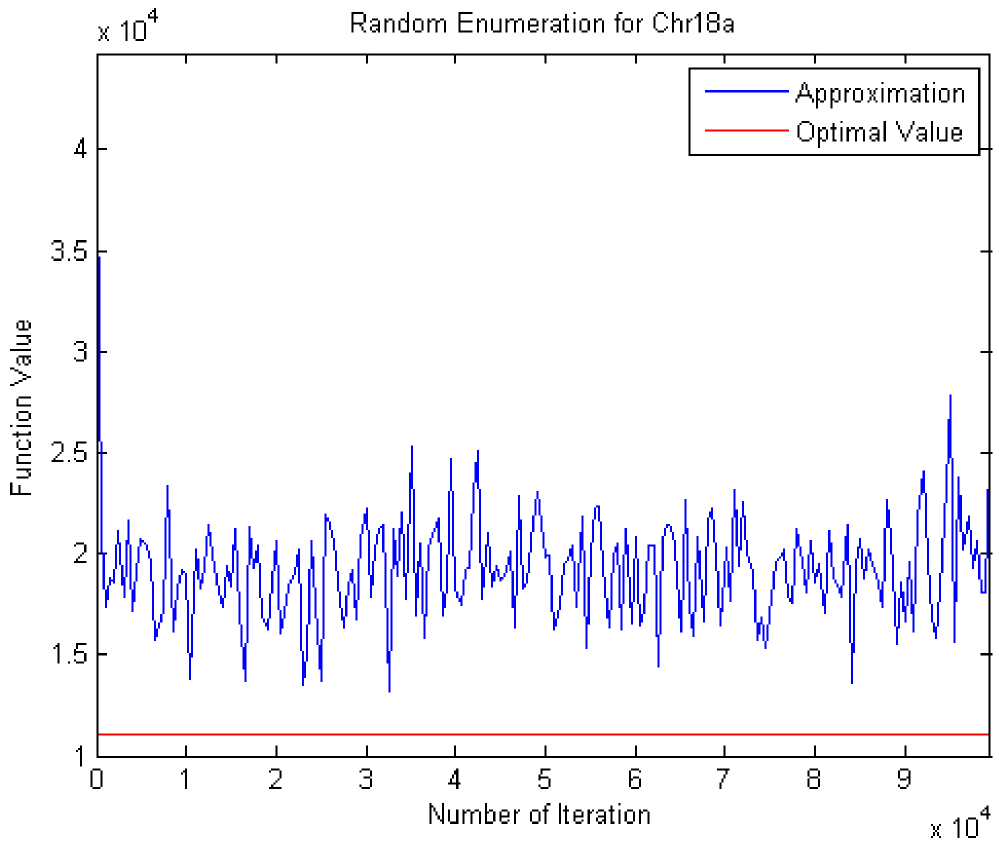
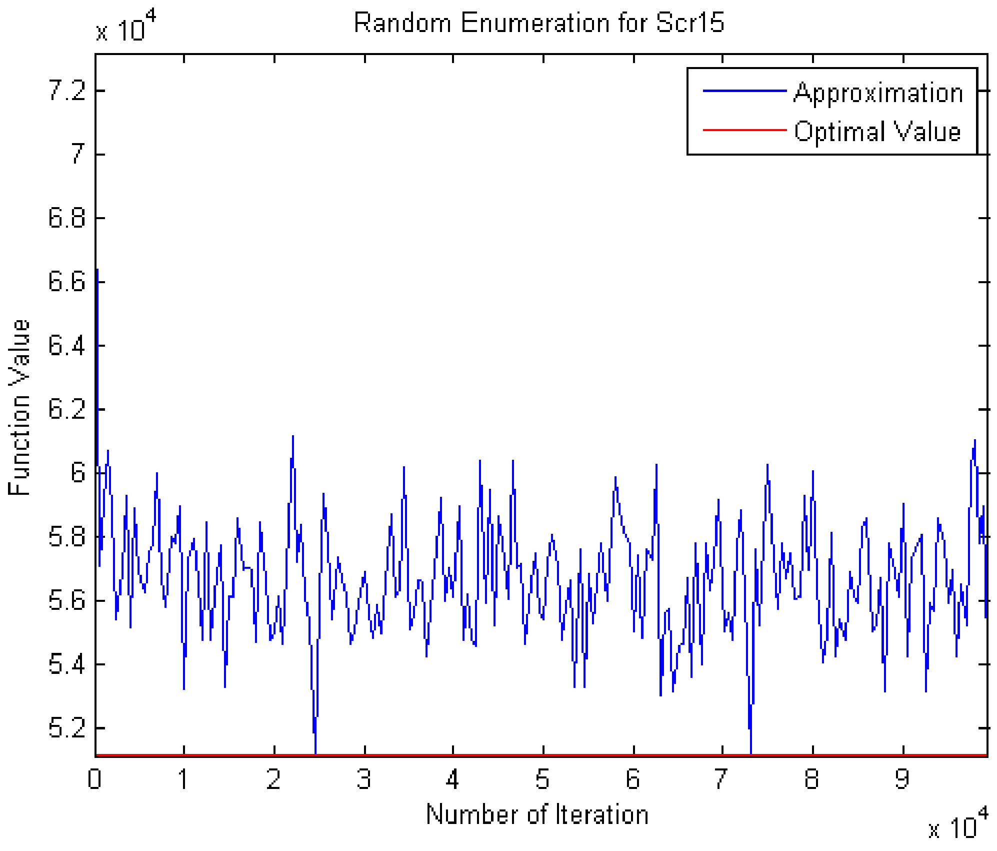

Random Enumeration for Chr18a
x 10^4
Approximation
Optimal Value
Function Value
Number of Iteration
x 10^4
4
3.5
3
2.5
2
1.5
1
0
1
2
3
4
5
6
7
8
9

Figure 7.3:


Random Enumeration for Scr15
x 10^4
7.2
7
6.8
6.6
6.4
6.2
6
5.8
5.6
5.4
5.2
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
x 10^4
Approximation
Optimal Value

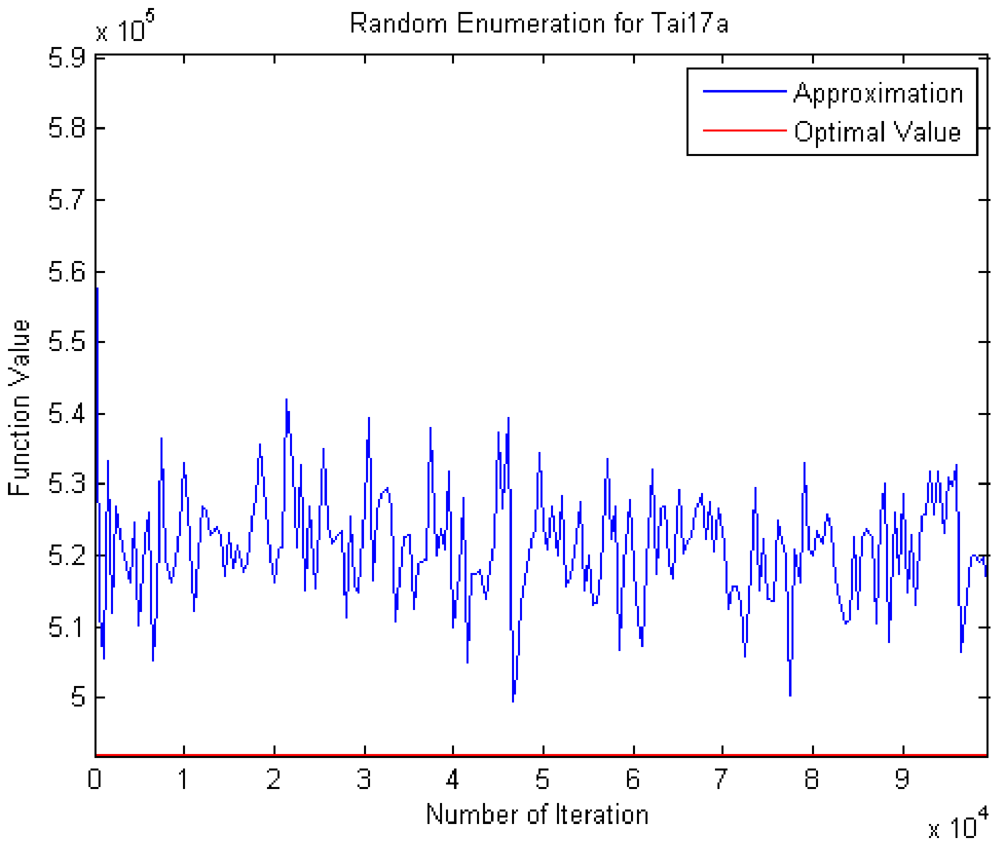

Random Enumeration for Tai17a
x 10^5
5.9
5.8
5.7
5.6
5.5
5.4
5.3
5.2
5.1
5
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
x 10^4
Approximation
Optimal Value

Figure 7.4:

8. Conclusion and further research

In this dissertation, we aimed to develop new solution methods for the QAP. In achieving this, we have identified and studied thoroughly a mathematical formulation of the QAP i.e the standard quadratic integer programming formulation which has not been studied before. For this mathematical formulation, we have proposed two solution approaches, namely, a *Branch-and-Bound* method and an auxiliary function-based dynamic convexized method. The *Branch-and-Bound* method for the QAP is well know in the literature. However, its application to the standard quadratic integer programming formulation of the QAP is something that has never been carried out previously. We have presented the first study of this type and numerical solutions of the QAP using *Branch-and-Bound*. In addition, we have introduced a heuristic random enumeration scheme to improve the efficiency and to speed up the convergence of the *Branch-and-Bound* method. We use interior point algorithm to calculate lower bounds for the sub-problems within the *Branch-and-Bound* algorithm. It is well known that the problem of finding an initial strictly feasible point for the interior point algorithm is not an easy task. In this dissertation, we have developed two new techniques to overcome this. These two different techniques have the advantage that they can be used for any other interior point algorithm.

The second solution approach is an auxiliary function-based dynamic convexized method. This method was proposed recently for solving the general non-linearly constrained non-linear integer programming, and has never been applied to the QAP. For this reason, we have decided to apply this method to the QAP. The major challenge in applying this method directly to the QAP was the neighbourhood structure. The auxiliary function-based dynamic convexized method requires the definition of an appropriate neighbourhood structure which was suggested in the context of non-linear programming problem. Unfortunately, the same neighbourhood structure cannot be used when solving the QAP as this will lead most often to infeasible solutions. To overcome this problem, we have defined an appropriate neighbourhood structure for the QAP solutions. As a consequence of this, the auxiliary function presented in this dissertation is different from the one used in solving non-linear integer programming problems. In this neighbourhood structure, all the point are feasible for the QAP. The theoretical results have been proved for the new neighbourhood structure and the defined auxiliary function. Numerical results obtained from this method are quite satisfying. Indeed, we have shown that this method is more robust than *Branch-and-Bound* in terms of computational time and efforts.

On the other hand, the *Branch-and-Bound* method, which is an exact solution method, could not solve some of the tested problem instances to optimality. This is due to the fact that, the objective function of the standard quadratic integer programming reformulation of the QAP is not convex. We have tried to overcome this problem with an equivalent convex formulation and an equivalent separable convex reformulation, but the lower bounds obtained were not of good quality. This is due to our convexification technique used.

However, Billionnet et al. [BEP09] proposed a different technique for transforming the objective function of an indefinite quadratic integer programming problem subject to linear equality constraints into a convex function. The investigation of this concept, if provides tight lower bounds, can therefore be adapted to improve the quality of the *Branch-and-Bound* method used in this dissertation. This will be one of the object of a future research.

In future research, we can also consider a more appropriate neighbourhood structure for the auxiliary function-based dynamic convexized method. This is due to the large size of the neighbourhood structure used in this dissertation, which slows down the computation of the DDC method.

Something else that is still to be done in the area of the QAP is studying the QAP under the graph formulation. This formulation was proposed by Loiola et al. [LAN+07] in a survey paper and has never been studied in the literature. Given the recent advances in the area of graph theory, and its computational tool in a software like *Mathematica*, this seems to be a promising future research direction to consider.

9. Appendix

9.1 Appendix 1: Generalized inverse of a matrix

The generalized inverse of a matrix A is the a matrix that has some properties of the inverse of A, and which can be used as the inverse of A if A is not invertible. The most widely known generalized inverse is the Moore–Penrose generalized inverse. This was named after the works by Moore in 1920 [Moo20] and Penrose in 1955 [Pen55].

For an $m \times n$–matrix A, the Moore–Penrose generalized inverse A^+ of the matrix A is defined as a matrix which satisfies the following:

- $AA^+A = A$,
- $A^+AA^+ = A^+$,
- $(AA^+)^T = AA^+$,
- $(A^+A)^T = A^+A$.

The Moore–Penrose generalized inverse A^+ of the A has the following properties:

i) A^+ is uniquely determined.

ii) If A is non-singular, $A^+ = A^{-1}$.

iii) If $A = 0$, $A^+ = 0$.

iv) $(A^+)^+ = A$.

v) If $rank(A) = m$, then $A^+ = A^T(AA^T)^{-1}$.

vi) $(A^T)^+ = (A^+)^T, \qquad (\alpha A)^+ = \alpha^{-1}A^+$ for $\alpha \neq 0$.

The computation of the Moore–Penrose generalized inverse has known many developments. In Matlab the in-built function *pinv* computes the Moore–Penrose generalized inverse of matrices.

9.2 Appendix 2: Minimizers from the auxiliary function-based method

In this section, we present in Tables 9.1, 9.2 and 9.3 the minimizers that realize the optimal value from DDC presented in Tables 7.4 and 7.5

Table 9.1: Minimizers of the solutions from the auxiliary function-based method (1)

Instances	Opt. Sol
Bur26a	(16, 11, 7, 15, 8, 3, 6, 2, 12, 14, 5, 9, 21, 1, 26, 13, 19, 20, 18, 25, 10, 4, 17, 24, 22, 23)
Chr12a	(7, 5, 12, 2, 1, 3, 9, 11, 10, 6, 8, 4)
Chr12b	(5, 7, 1, 10, 11, 3, 4, 2, 9, 6, 12, 8)
Chr12c	(7, 5, 1, 3, 10, 4, 8, 6, 9, 11, 2, 12)
Chr18a	(8, 5, 9, 6, 15, 7, 18, 13, 1, 12, 3, 17, 16, 14, 4, 2, 10, 11)
Chr18b	(1, 2, 4, 3, 5, 6, 8, 9, 7, 12, 10, 11, 13, 14, 16, 15, 17, 18)
Chr20a	(13, 2, 16, 11, 4, 8, 15, 9, 6, 18, 10, 20, 1, 12, 17, 19, 3, 7, 5, 14)
Chr20b	(13, 4, 16, 14, 1, 6, 9, 12, 11, 7, 2, 3, 19, 20, 18, 8, 17, 10, 5, 15)
Chr20c	(11, 9, 6, 7, 2, 12, 19, 4, 20, 10, 1, 13, 16, 5, 14, 18, 8, 3, 17, 15)
Chr22a	(4, 1, 16, 12, 13, 6, 11, 20, 5, 21, 3, 19, 17, 9, 2, 15, 14, 10, 22, 18, 7, 8)
Chr22b	(13, 6, 4, 1, 20, 2, 17, 22, 8, 16, 9, 11, 15, 5, 18, 7, 21, 10, 19, 12, 14, 3)
Chr25a	(25, 22, 12, 3, 18, 2, 4, 11, 20, 10, 7, 21, 24, 9, 5, 19, 14, 8, 16, 6, 23, 15, 17, 13, 1)
Tai10a	(9, 1, 8, 6, 10, 5, 4, 3, 7, 2)
Tai10b	(5, 6, 1, 4, 7, 8, 9, 3, 2, 10)
Tai12a	(8, 1, 6, 2, 11, 10, 3, 5, 9, 7, 12, 4)
Tai12b	(9, 4, 6, 3, 11, 7, 12, 2, 8, 10, 1, 5)
Scr12	(10, 7, 2, 3, 11, 4, 8, 12, 1, 6, 9, 5)
Scr15	(15, 7, 11, 8, 1, 4, 3, 2, 12, 6, 13, 5, 14, 10, 9)
Scr20	(20, 3, 10, 2, 12, 1, 7, 8, 14, 11, 18, 9, 19, 15, 16, 17, 13, 5, 6, 4)
Nug12	(2, 10, 6, 5, 1, 11, 8, 4, 3, 9, 7, 12)
Nug14	(9, 8, 13, 2, 1, 11, 7, 14, 3, 4, 12, 5, 6, 10)
Nug15	(9, 8, 13, 2, 1, 11, 7, 14, 3, 4, 12, 5, 6, 15, 10)
Nug16a	(9, 10, 12, 14, 16, 3, 1, 8, 15, 11, 6, 5, 7, 2, 4, 13)

Table 9.2: Minimizers of the solutions from the auxiliary function-based method (2)

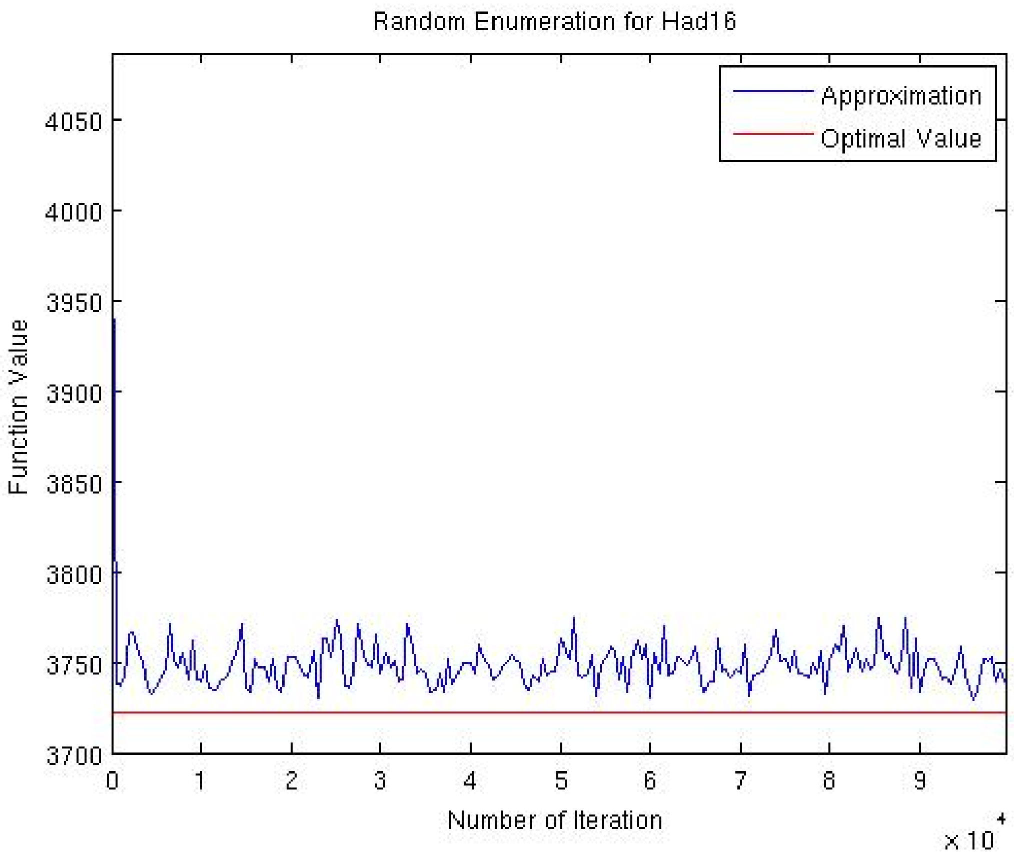
Instances	Opt. Sol
Nug16b	(8, 3, 12, 15, 13, 9, 2, 16, 11, 7, 10, 4, 5, 1, 6, 14)
Nug20	(9, 3, 10, 14, 18, 16, 11, 12, 2, 4, 13, 8, 20, 15, 19, 6, 1, 7, 5, 17)
Nug21	(20, 19, 8, 7, 1, 12, 17, 14, 18, 11, 16, 10, 6, 15, 4, 21, 3, 9, 13, 2, 5)
Nug27	(11, 18, 7, 8, 27, 23, 4, 3, 13, 12, 5, 21, 22, 26, 25, 6, 19, 15, 2, 17, 9, 14, 24, 20, 1, 10, 16)
Nug28	(13, 15, 8, 1, 22, 26, 11, 4, 18, 20, 5, 27, 19, 23, 28, 10, 7, 9, 3, 24, 17, 12, 2, 14, 6, 21, 16, 25)
Nug30	(14, 20, 3, 21, 2, 5, 4, 27, 18, 29, 9, 23, 30, 11, 22, 19, 10, 26, 16, 25, 8, 7, 1, 17, 15, 28, 6, 13, 12, 24)
Had12	(3, 10, 5, 2, 12, 11, 7, 1, 8, 6, 4, 9)
Had14	(8, 13, 10, 11, 12, 5, 2, 14, 3, 6, 7, 1, 9, 4)
Had16	(9, 4, 16, 1, 7, 8, 6, 14, 15, 11, 12, 10, 5, 3, 2, 13)
Had18	(8, 15, 16, 6, 7, 18, 14, 11, 1, 10, 12, 5, 13, 3, 2, 17, 9, 4)
Had20	(8, 15, 1, 14, 6, 19, 7, 11, 16, 12, 10, 17, 2, 20, 5, 3, 4, 9, 18, 13)
Esc16a	(1, 10, 9, 13, 4, 3, 7, 6, 5, 2, 11, 16, 8, 12, 15, 14)
Esc16b	(1, 4, 10, 6, 16, 8, 3, 12, 2, 14, 7, 5, 13, 9, 15, 11)
Esc16c	(1, 10, 2, 14, 6, 3, 5, 7, 9, 16, 11, 15, 13, 12, 8, 4)
Esc16d	(1, 10, 8, 5, 2, 3, 11, 7, 12, 15, 13, 16, 6, 14, 4, 9)
Esc16e	(3, 2, 7, 4, 15, 16, 11, 12, 8, 9, 13, 10, 5, 14, 6, 1)
Esc16f	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)
Esc16g	(1, 13, 9, 14, 5, 6, 2, 10, 12, 3, 7, 4, 16, 8, 15, 11)
Esc16h	(5, 7, 8, 16, 12, 10, 15, 13, 14, 9, 11, 2, 1, 4, 3, 6)
Esc16i	(1, 2, 3, 4, 8, 6, 7, 5, 13, 12, 11, 10, 9, 14, 15, 16)
Esc16j	(9, 6, 2, 4, 5, 3, 7, 13, 8, 14, 11, 12, 1, 10, 15, 16)

Table 9.3: Minimizers of the solutions from the auxiliary function-based method (3)

Instances	Opt. Sol
Esc32a	$(13, 14, 16, 32, 30, 29, 28, 2, 22, 27, 21, 9, 5, 6, 8, 3, 1, 24, 23, 10, 20, 25, 4, 19, 17, 11, 18, 26, 12, 15, 7, 31)$
Esc32b	$(7, 3, 8, 6, 4, 2, 5, 1, 13, 15, 9, 11, 12, 10, 14, 16, 30, 32, 20, 18, 19, 27, 26, 28, 25, 23, 21, 24, 17, 29, 31, 22)$
Esc32c	$(20, 29, 17, 19, 28, 9, 6, 31, 11, 10, 16, 14, 13, 15, 25, 12, 30, 32, 26, 27, 21, 22, 23, 5, 8, 3, 1, 2, 18, 24, 7, 4)$
Esc32d	$(15, 8, 23, 16, 7, 12, 6, 13, 14, 3, 4, 21, 11, 5, 10, 2, 9, 1, 27, 22, 18, 24, 17, 19, 25, 30, 20, 28, 29, 26, 31, 32)$
Esc32e	$(11, 7, 3, 4, 19, 30, 25, 15, 9, 1, 5, 6, 13, 14, 28, 16, 31, 18, 27, 17, 21, 20, 23, 24, 2, 26, 8, 10, 29, 12, 22, 32)$
Esc32g	$(6, 1, 3, 4, 2, 11, 29, 8, 9, 10, 5, 12, 13, 14, 15, 16, 17, 18, 19, 20, 26, 22, 23, 24, 25, 21, 27, 28, 7, 30, 31, 32)$
Esc32h	$(17, 20, 1, 7, 28, 4, 3, 9, 11, 26, 10, 12, 15, 31, 27, 19, 23, 18, 6, 13, 8, 24, 14, 5, 25, 21, 30, 32, 2, 29, 16, 22)$
Esc64a	$(1, 59, 29, 31, 20, 6, 41, 8, 56, 15, 5, 24, 36, 14, 13, 11, 39, 25, 37, 27, 21, 10, 22, 52, 33, 54, 28, 32, 47, 30, 9,$ $64, 46, 57, 45, 19, 16, 44, 12, 40, 49, 53, 43, 7, 35, 38, 23, 60, 2, 26, 3, 63, 42, 50, 55, 4, 62, 58, 18, 48, 61, 34, 17, 51)$
Kra30a	$(1, 6, 12, 19, 27, 10, 5, 9, 30, 8, 26, 18, 13, 20, 15, 11, 17, 7, 23, 24, 22, 29, 14, 3, 25, 2, 16, 28, 21, 4)$
Kra30b	$(7, 11, 24, 4, 23, 29, 1, 6, 20, 12, 13, 17, 10, 14, 2, 25, 18, 28, 9, 30, 5, 3, 26, 8, 22, 15, 16, 27, 21, 19)$
Kra32	$(11, 3, 4, 7, 6, 8, 5, 2, 9, 25, 15, 16, 13, 31, 23, 14, 10, 26, 17, 1, 21, 12, 22, 19, 24, 27, 30, 29, 18, 28, 20, 32)$

9.3 Appendix 3: Choice of the value of N in the random enumeration

This section contains the graphical results of the experiment on choosing the value of N in the heuristic random enumeration method. In Figures 9.1–9.4, the x-axis represents the number of iteration, and the y axis represent the average optimal value over 10 runs of the heuristic random enumeration method.


Random Enumeration for Had16
Approximation
Optimal Value
Function Value
4050
4000
3950
3900
3850
3800
3750
3700
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10^4

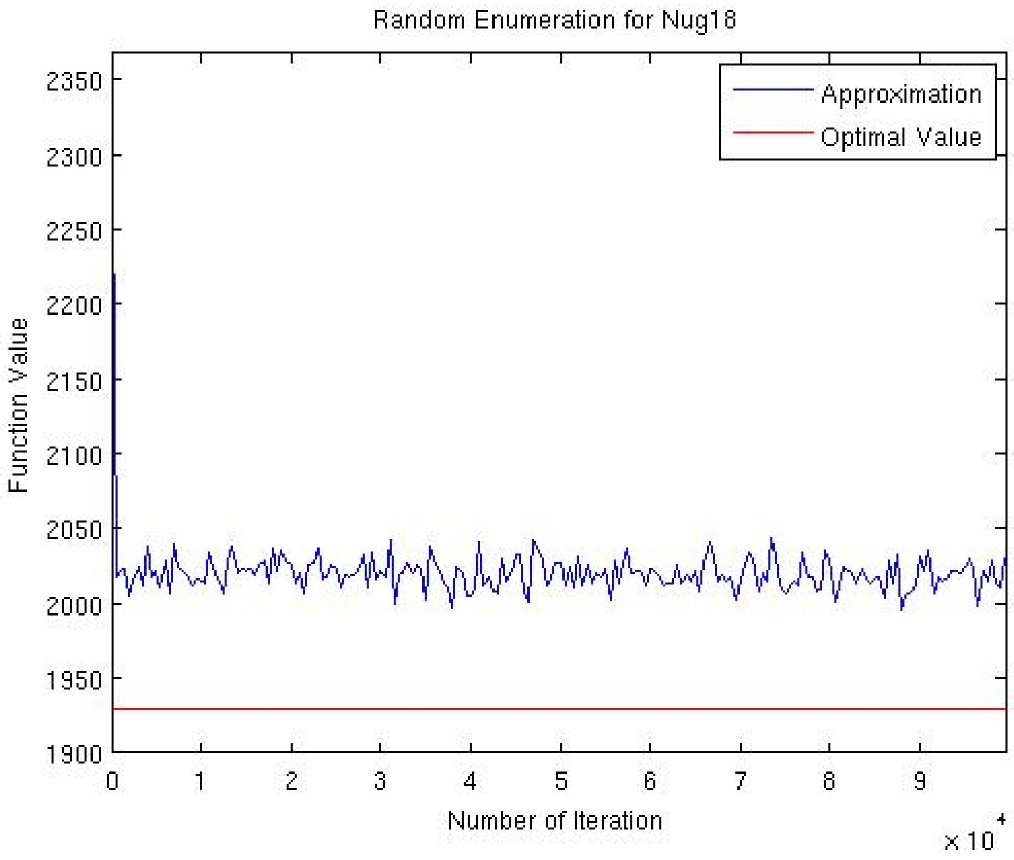
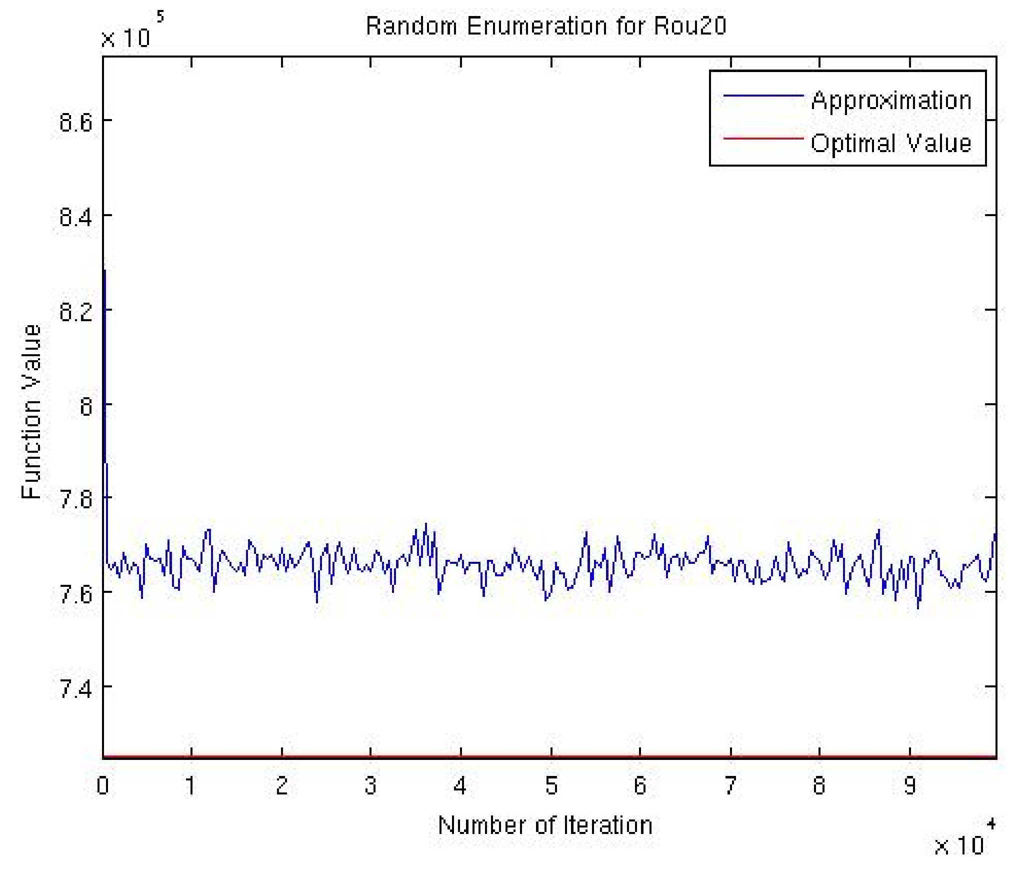

Random Enumeration for Nug18
Approximation
Optimal Value
Function Value
2350
2300
2250
2200
2150
2100
2050
2000
1950
1900
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10^4

Figure 9.1:


Random Enumeration for Rou20
× 10^5
Approximation
Optimal Value
8.6
8.4
8.2
8
7.8
7.6
7.4
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10^4

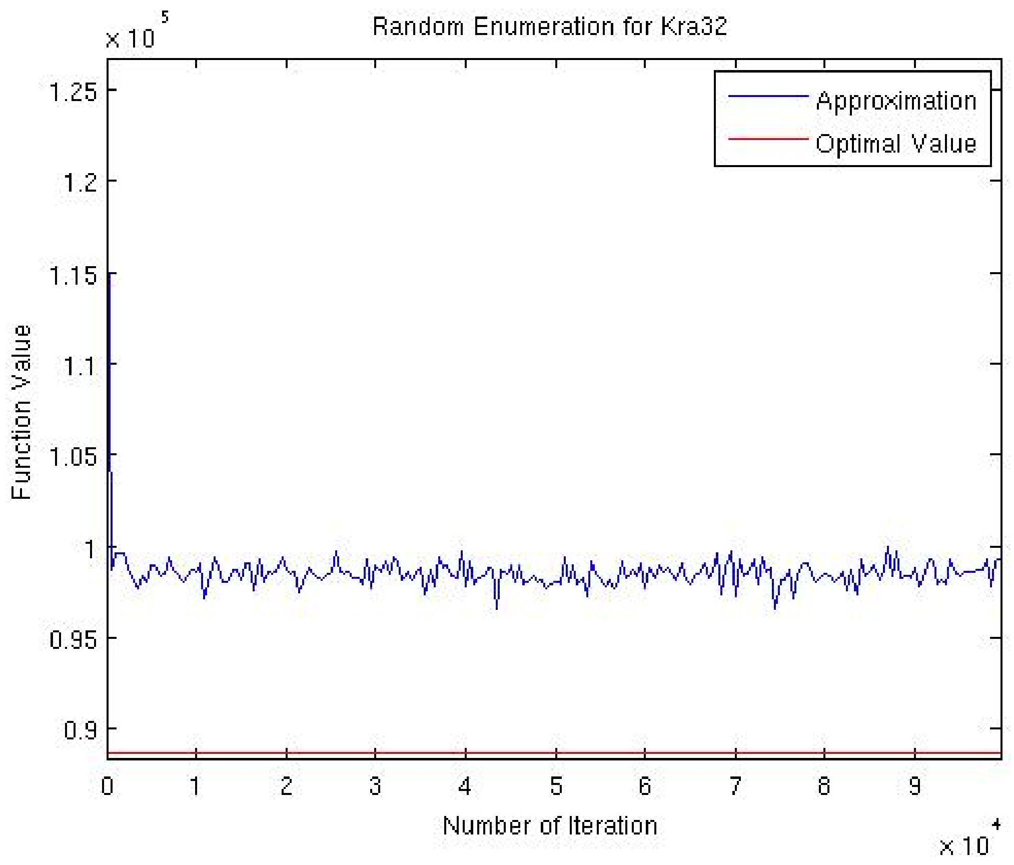
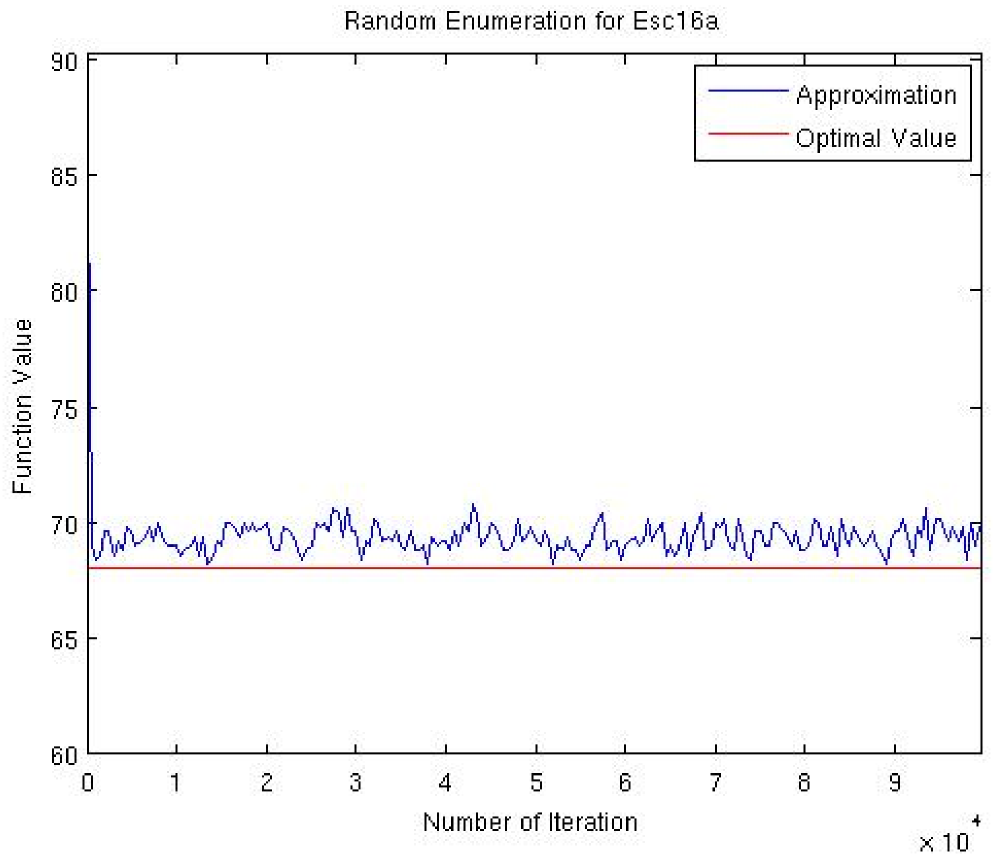

Random Enumeration for Kra32
× 10^5
Approximation
Optimal Value
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
Function Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10^4

Figure 9.2:


Random Enumeration for Esc16a
Approximation
Optimal Value
Function Value
Number of Iteration
90
85
80
75
70
65
60
0
1
2
3
4
5
6
7
8
9
× 10⁴

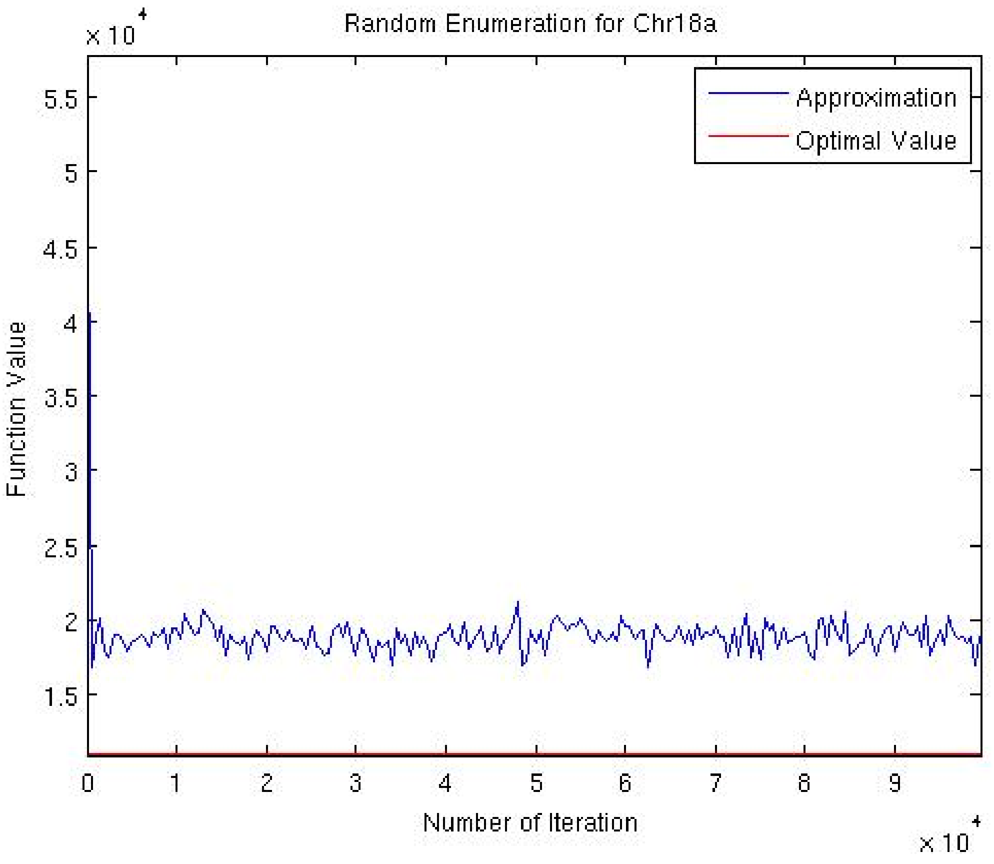
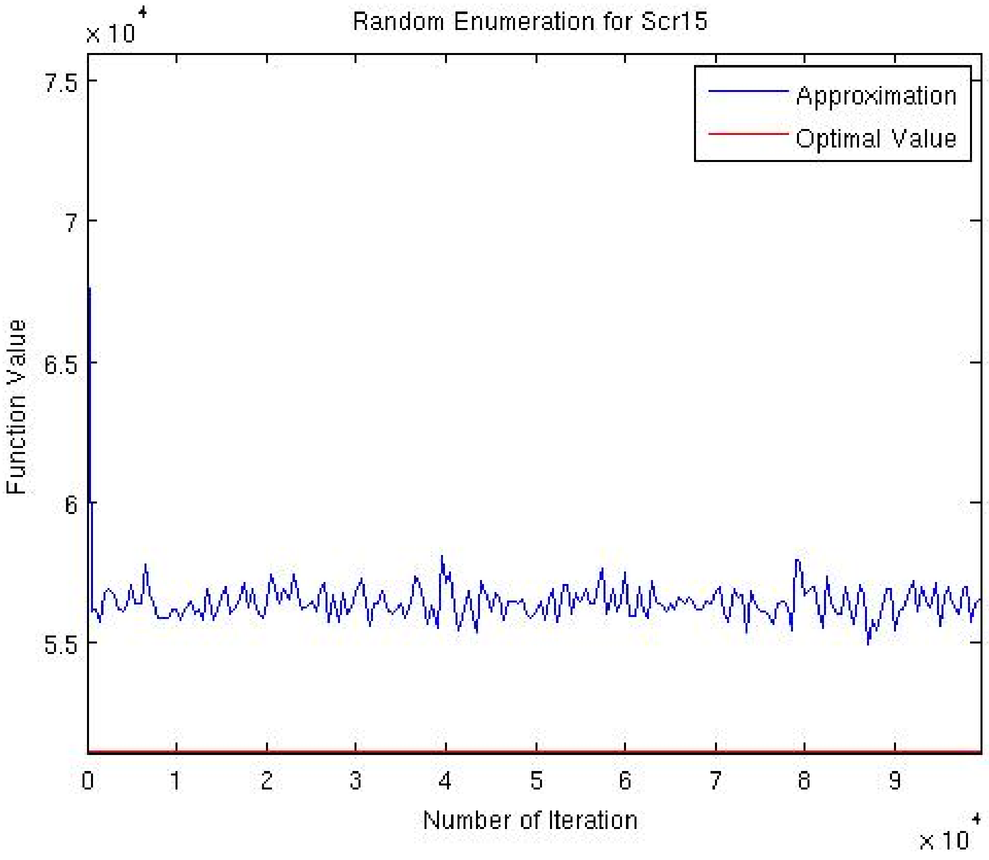

Random Enumeration for Chr18a
× 10⁴
Approximation
Optimal Value
Function Value
Number of Iteration
5.5
5
4.5
4
3.5
3
2.5
2
1.5
0
1
2
3
4
5
6
7
8
9
× 10⁴

Figure 9.3:


Random Enumeration for Scr15
× 10⁴
7.5
7
6.5
6
5.5
Function Value
Approximation
Optimal Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10⁴

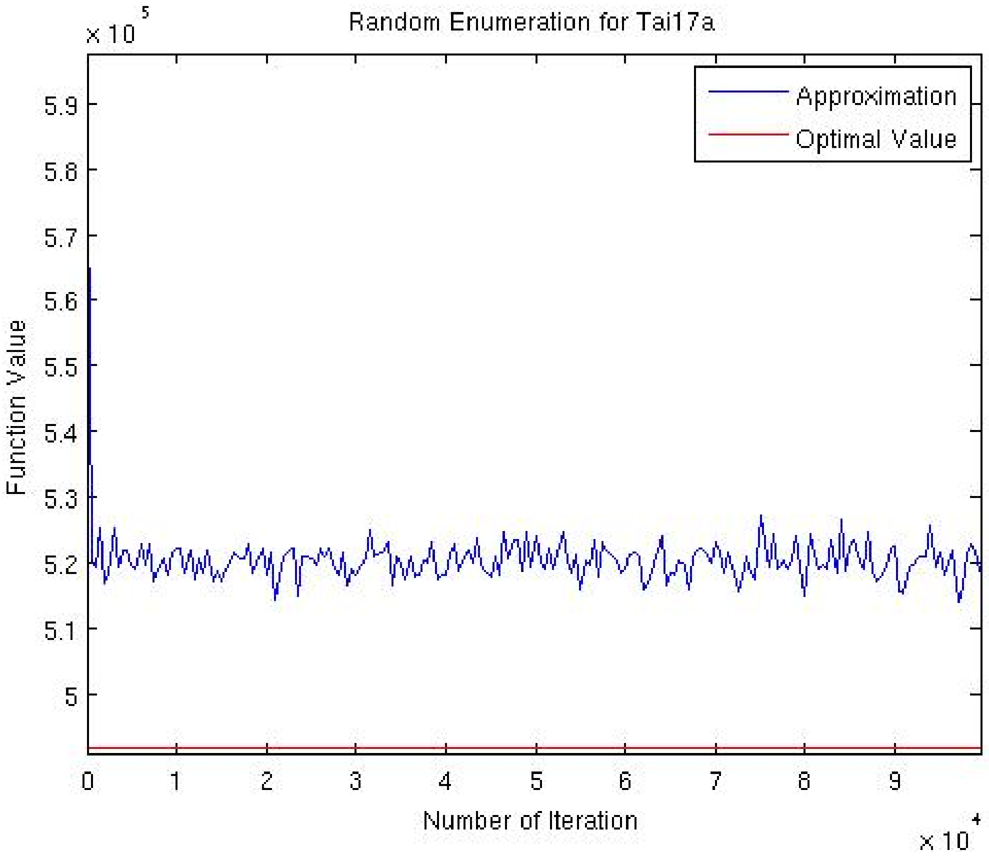

Random Enumeration for Tai17a
× 10⁵
5.9
5.8
5.7
5.6
5.5
5.4
5.3
5.2
5.1
5
Function Value
Approximation
Optimal Value
0
1
2
3
4
5
6
7
8
9
Number of Iteration
× 10⁴

Figure 9.4:

References

[AJ94] W. P. Adams and T. A. Johnson, *Improved linear programming-based lower bounds for the quadratic assignment problem*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **16** (1994), 43–75.

[AOT00] R. Ahuja, J. B. Orlin, and A. Tiwari, *A greedy genetic algorithm for the quadratic assignment problem*, Computers and Operations Research **27** (2000), no. 10, 917–934.

[BA01] N. W. Brixius and K. M. Anstreicher, *Solving quadratic assignment problems using convex quadratic programming relaxations*, Optimization Methods and Software **16** (2001), 49–68.

[BcKR] R. E. Burkard, E. Çela, S. E. Karisch, and F. Rendl, *http://www.seas.upenn.edu/qaplib/ (a mirror site is hosted at ecole polytechnique de montreal, by miguel anjos http://anjos.mgi.polymtl.ca/qaplib/)*, QAPLIB: A Quadratic Assignment Problem Library. Benchmark of problems collected in May 2011.

[BcPP98] R. E. Burkard, E. Çela, P. M. Pardolos, and L. S. Pitsoulis, *The quadratic assignment problem.*

[BEFW03] A. Blanchard, S. Elloumi, A. Faye, and N. Wicker, *A cutting algorithm for the quadratic assignment problem*, INFORM **41** (2003), 35–49.

[BEP09] A. Billionnet, S. Elloumi, and M. C. Plateau, *Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The qcr method*, Discrete Applied Mathematics **157** (2009), no. 6, 1185 – 1197.

[BGG$^+$71] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, *Experiments in mixed integer linear programming*, Mathematical Programming **1** (1971), 76–94.

[Bos93] J. Bos, *A quadratic assignment problem solved by simulated annealing*, Journal of Environmental Management **37** (1993), 127–145.

[BR84] R. E. Burkard and F. Rendl, *A thermodynamically motivated simulation procedure for combinatorial optimization problems*, European Journal of Operational Research **17** (1984), no. 2, 169–174.

[BS82] M. S. Bazaraa and H. D. Sherali, *On the use of exact and heuristic cutting plane methods for the quadratic assignment problem*, Journal of Operations Research Society **33** (1982), 991–1003.

[BS00] M. J. Brusco and S. Stahl, *Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices*, Journal of Classification **17** (2000), no. 2, 197–223.

[BV06] S. Burer and D. Vandenbussche, *Solving lift and project relaxations of binary integer programs*, Siam Journal on Optimization **16** (2006), no. 3, 726–750.

[CDM$^+$95] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian, *Heuristics from nature for hard combinatorial optimization problems*, International Transactions in Operational Research **3** (1995), no. 1, 1–21.

[CKPR98] J. Clausen, S. E. Karisch, M. Perregaard, and F. Rendl, *On the applicability of lower bounds for solving rectilinear quadratic assignment problems in parallel*, Computational Optimization and Applications **10** (1998), no. 2, 127–147.

[Con90] D. T. Connolly, *An improved annealing scheme for the qap*, European Journal of Operational Research **46** (1990), 93–100.

[Cot67] R. W. Cottle, *On the convexity of quadratic forms over convex sets*, Operations Research **15** (1967), 170–172.

[DH72] J. W. Dickey and J. W. Hopkins, *Campus building arrangement using topaz*, Transportation Research **26** (1972), 29–41.

[DM05] G. B. David and D. Malah, *Bounds on the performance of vector-quantizers under channel errors*, IEEE Transactions on Information Theory **51** (2005), no. 6, 2227–2235.

[DW09] Y. Ding and H. Wolkowicz, *A low dimensional semidefinite relaxation for the quadratic assignment problem*, Mathematics and Operations Research **34** (2009), 1008–1022.

[Edw80] C. S. Edwards, *A branch and bound algorithm for the koopmans beckmann quadratic assignment problem*, Mathematical Programming Study **13** (1980), 35–52.

[Els77] A. N. Elshafei, *Hospital layout as a quadratic assignment problem*, Operational Research Quarterly (1970 to 1977) **28** (1977), 167–179.

[FBR87] G. Finke, R. E. Burkard, and F. Rendl, *Quadratic assignment problems*, Annals of Discrete Mathematics **5** (1987), 61–82.

[FDZ$^+$94] J. . Forsberg, R. M. Delaney, Q. Zhao, G. Harakas, and R. Chandran, *Analyzing lanthanide included shifts in the nmr spectra of lanthanide (iii) complexes derived from 1,4,7,10tetrakis (n,ndiethylacetamido) 1,4,7,10tetraazacyclododecane*, Inorganic Chemistry **34** (1994), 3705–3715.

[FG99] C. Fleurent and F. Glover, *Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory*, INFORMS Journal on Computing **11** (1999), 189–203.

[FR95] T. A. Feo and M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization **6** (1995), 109–133.

[FW10] C. W. Fon and K. Y. Wong, *Investigating the performance of bees algorithm in solving quadratic assignment problem*, International Journal of Operational Research **9** (2010), no. 3, 241–257.

[FY83] A. M. Frieze and J. Yadegar, *On the quadratic assignment problem*, Discrete Applied Mathematics **5** (1983), 89–98.

[FYEHP89] A. M. Frieze, J. Yadegar, S. El-Horbaty, and D. Parkinson, *Algorithms for assignment problems on an array processor*, Parallel Computing **11** (1989), 151–162.

[GG76] A. M. Geoffrion and G. W. Graves, *Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/lp approach*, Operations Research **24** (1976), 595–610.

[Gil62] P. C. Gilmore, *Optimal and suboptimal algorithms for the quadratic assignment problem*, SIAM Journal on Applied Mathematics **10** (1962), 305–313.

[GL] F. Glover and M. Laguna, *Tabu search*, Lecture note.

[GU07] R. N. Gasimov and O. Ustun, *Solving the quadratic assignment problem using f msg algorithm*, Journal of Industrial and management Optimization **3** (2007), no. 2, 173–191.

[Had94] S. W. Hadley, *Domination and separation applied to the quadratic assignment problem*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **16** (1994), 189–196.

[Hef77] D. R. Heffley, *Assigning runners to a relay team*, Optimal Strategy in Sport. North Holland,Amsterdam (1977), 169–171.

[HG98] P. Hahn and T. Grant, *Lower bounds for the quadratic assignment problem based upon a dual formulation*, Operations Research **46** (1998), no. 6, 912–922.

[HGH98] P. Hahn, T. Grant, and N. Hall, *A branch and bound algorithm for the quadratic assignment problem based on the hungarian method*, European Journal of Operational Research **108** (1998), 629–640.

[Hol75] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, Ann Arbor MI: University of Michigan Press (1975).

[HRW90] S. W. Hadley, F. Rendl, and H. Wolkowicz, *Bounds for the quadratic assignment problem using continuous optimization techniques*, Integer Programming and Combinatorial Optimization. University of Waterloo Press (1990), 237–248.

[HZGS10] P. M. Hahn, Y. R. Zhu, M. Guignard, and J. M. Smith, *Exact solution of emerging quadratic assignment problems*, International Transactions in Operational Research **17** (2010), 525–552.

[JK96] M. Jünger and V. Kaibel, *A basic study of the qap-polytope*, Tech. report, Institut für Informatik, Universität zu Köln, Pohligstrasse 1, D-50969, 1996.

[JWL06] P. Ji, Y. Wu, and H. Liu, *A solution method for the quadratic assignment problem (qap)*, The Sixth International Symposium on Operations Research and Its Applications (ISORA06) (2006), 106–117.

[Kar95] S. E. Karisch, *Nonlinear approaches for quadratic assignment and graph partition problems*, Technical University Graz, Austria **Ph.D. Thesis** (1995).

[KB57] T. C. Koopmans and M. J. Beckmann, *Assignment problems and the location of economic activities*, Electronica **25** (1957), 53–76.

[KB78] L. Kaufmann and F. Broeckx, *An algorithm for the quadratic assignment problem using benders decomposition*, European Journal of Operational Research **2** (1978), 204–211.

[KGV83] S. Kirkpatrick, C. D. Gellat, and M. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), 671–680.

[KR95] S. E. Karisch and F. Rendl, *Lower bounds for the quadratic assignment problem via triangle decompositions*, Programming (1995), 137–152.

[Kuh55] H. H. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly **2** (1955), 83–97.

[LAN$^+$07] E. M. Loiola, N. M. M. Abreu, P. O. B. Netto, P. Hahn, and T. Querido, *A survey for the quadratic assignment problem*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **176** (2007), 657–690.

[Law63] E. L. Lawler, *The quadratic assignment problem*, Management Science **9** (1963), no. 4, 586–599.

[LPR94] Y. Li, P. M. Pardolos, and M. G. C. Resende, *A greedy randomized adaptive search procedure for the quadratic assignment problem*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **16** (1994), 237–261.

[LS10] D. Li and X. Sun (eds.), *Nonlinear integer programming*, International Series in Operations Research and Management Science Series, no. 9781441939913, Springer London, Limited, 233 Spring Street, New York, NY 10013, USA, 2010.

[LW76] R. F. Love and J. Y. Wong, *Solving quadratic assignment problems with rectangular distances and integer programming*, Naval Research Logistics Quarterly **23** (1976), 623–627.

[Meh92] S. Mehrotra, *On implementation of a primal-dual interior point method*, SIAM Journal on Optimization **2** (1992), 575–601.

[Mey00] C. D. Meyer (ed.), *Matrix analysis and applied linear algebra*, SIAM, 3600 University City Sciences Centre, Philadelphia,PA, 19104-2688, 2000.

[Mis05] A. Misevicius, *A tabu search algorithm for the quadratic assignment problem*, Computational Optimization and Applications **30** (2005), no. 1, 95–111.

[MLMF05] G. Miranda, H. Luna, G. R. Mateus, and R Ferreira, *A performance guarantee heuristic for electronic components placement problems including thermal effects*, Computers and Operations Research **32** (2005), 2937–2957.

[Moo20] E. H. Moore, *On the reciprocal of the general algebraic matrix*, Bulletin of the American Mathematical Society **26** (1920), 394–395.

[MRD95] G. C. Mauricio, K. G. Resende, and R. Z. Drezner, *Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming*, Operations Research **43** (1995), no. 5, 781–791.

[MT93] N. Megiddo and A. Tamir, *Linear time algorithms for some separable quadratic programming problems*, Operations Research Letters **13** (1993), 203–211.

[NP95] V. Nissen and H. Paul, *A modification of threshold accepting and its application to the quadratic assignment problem*, OR Spektrum **17** (1995), 205–210.

[NW99] J. Nocedal and S. J. Wright (eds.), *Numerical optimization*, Springer series in operations research, no. 0387987932, Springer New York, 233 Spring Street, New York, NY 10013, USA, 1999.

[OPR04] C. A. S. Oliveira, P. M. Pardolos, and M. G. C. Resende, *Grasp with path relinking for the quadratic assignment problem*, In: Experimental and Efficient Algorithms at the Third International Workshop (WEA 2004), Brazil, LNCS **3059** (2004), no. 1, 356–368.

[Pen55] R. Penrose, *A generalized inverse for matrices*, Proceedings of the Cambridge Philosophical Society **51** (1955), 406–413.

[PGK$^+$05] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, *The bees algorithm*, Technical Note, Manufacturing Engineering Centre **Cardiff University** (2005), UK.

[PGR76] M. A Pollatscheck, N. Gershoni, and Y. T. Radday, *Optimization of the typewriter keyboard by simulation*, Angewandte Informatik **17** (1976), 438–439.

[PRW94] P. M. Pardolos, F. Rendl, and H. Wolkowicz, *The quadratic assignment problem: A survey and recent developments*, DIMACS Series in Discrete Mathematics and Theoretical computer Sciences **16** (1994), no. In The Quadratic Assignment and Related Problems (Edited by P.M. Pardalos & H. Wolkowicz), 1–42.

[RJG10] C. Rego, T. James, and F. Glover, *An ejection chain algorithm for the quadratic assignment problem*, Networks **56** (2010), no. 3, 188–206.

[RPL96] M. G. C. Resende, P. M. Pardolos, and Y. Li, *Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using grasp*, ACM Transactions on Mathematical Software **22** (1996), no. 1, 104–118.

[RS03] C. S. Rabak and J. S. Schiman, *Using a teams to optimize automatic insertion of electronic components*, Advanced Engineering Informatics **17** (2003), no. 2, 95–106.

[RS07] R. Rendl and R. Sotirov, *Bounds for the quadratis assignment problem using the bundle method*, Mathematical Programming **109** (2007), no. Serie B, 505–524.

[SD99] T. Stützle and M. Dorigo, *Aco algorithms for the quadratic assignment problem*, pp. 33–50, McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[S.G31] S.Gersgorin, *Über die abgrenzung der eigenwerte einer matrix*, Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na **6** (1931), 749–754.

[SG76] S. Sahni and T. Gonzalez, *P complete approximation problems*, Journal of the Association for Computing Machinery **23** (1976), 555–565.

[SLSD09] L. Q. Song, M. H. Lim, P. N. Suganthan, and V. K. Doan, *Ensemble for solving quadratic assignment problems*, International Conference of Soft Computing and Pattern Recognition (2009).

[Ste61] L. Steinberg, *The backboard wiring: A placement algorithm*, SIAM Review **3** (1961), 37–50.

[Tai91] E. Taillard, *Robust taboo search for the quadratic assignment problem*, Parallel Computing **17** (1991), 44–455.

[TS95] D. E. Tate and A. E. Smith, *A genetic approach to the quadratic assignment problem*, Computers and Operations Research **22** (1995), 73–83.

[WW87] M. R. Wilhelm and T. L. Ward, *Solving quadratic assignment problems by simulated annealing*, EEE Transactions **19** (1987), 107–119.

[WZ04] B. Wess and T. Zeitlhofer, *On the phase coupling problem between data memory layout generation and address pointer assignment*, Lecture Notes in Computer Science **3199** (2004), 152–166.

[ZA09] W. Zhu and M. M. Ali, *Discrete dynamic convexized method for nonlinear integer programming*, Computer and Operations Research **36** (2009), 2723–2728.

[Zha96] Q. Zhao, *Semidefinite programming for assignment and partitioning problems*, University of Waterloo, Ontario, Canada, **Ph.D. Thesis** (1996).

[Zhu07] Y. R. Zhu, *Recent advances and challenges in quadratic assignment and related problems*, University of Pennsylvania **Ph.D. Thesis** (2007).

[ZKRW98] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz, *Semidefinite relaxations for the quadratic assignment problem*, Journal of Combinatorial Optimization **2** (1998), 71–109.

[ZRC10] H. Zhang, C. B. Royo, and M. Constantino, *Effective formulation reductions for the quadratic assignment problem*, Computers and Operations Research **37** (2010), 2007–2016.

[ZSL10] X. J. Zheng, X. L. Sun, and D. Li, *Separable relaxation for nonconvex quadratic integer programming: Integer diagonalization approach*, Journal of Optimization theory and Applications **146** (2010), no. 2, 182–191.