An Evolutionary Approach to Solving the Maximum Size Consecutive Ones Submatrix and Related Problems

A Thesis submitted for the degree of

Doctor of Philosophy

at the

Department of Mathematical Sciences
University of Essex

by

Rewayda Abo Alsabeh

August, 2017

Dedicated to

My affectionate parents

My dear brothers and sisters

All who continually pray for my fortune.

Acknowledgements

This thesis is the culmination of input, work and encouragement of many people who have helped and accompanied me for the four years that I have spent at University of Essex. First of all, my deepest gratitude goes to my family, my parents, my brothers, and my sisters for their endless love and unconditional support. I could not have achieved this without the continuous support and love of my family. Mom, Husham, and Somya thank you for everything you did.

I would like to express my sincerest thanks to my PhD supervisor, Prof. Abdel Salhi, for steering me through and pushing me further than I can imagine. His eloquence, sharpness, and great sense of humor have impressed me deeply and made my every meeting and discussion with him always full of joy and unforgettable memories. He has not only provided me his professional expertise and valuable guidance I needed to complete my Ph.D. studies but also been a real friend and a true mentor. I feel extremely fortunate to work with him. He has always provided me the freedom to choose my research subjects and encouraged me to be involved in various types of research activities. During our academic discussions, I was very impressed by his enthusiasm and his ability in thinking extreme while keeping the real life applicability of the research high. I learned a lot from him, and I am very grateful to have him as my advisor.

One of the best parts of attending Essex University has been the wonderful opportunity to be surrounded by the many interesting, intellectually stimulating and impressive fellow students. The list includes: Dr.Ghazwan Alsoufi, and Dr. Khattab Ali. My deepest gratitude goes to Bernard Fares for his help. Specially thanks to Prof. Edward Codling for his guidance, constructive suggestion for my research.

For the financial support, I would like to thank the University of Kufa. Finally, I would

like to thank the University of Essex for the President's Graduate Fellowship and all the services of the university staffs.

Declaration

The work in this thesis is based on research carried out Department of Mathematical Sciences, University of Essex, United Kingdom. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is all my own work, unless referenced, to the contrary, in the text.

Abstract

The Consecutive Ones Submatrix (C1S) has a vital role in real world applications. Consequently, there are continuous concern and demand to solve this problem via efficient algorithms. These algorithms are judged on the basis of their robustness, ease of use, and their computational time.

The main aim of this thesis is to convert a Pure Integer Linear Programming (ILP) with (0, 1)−matrix into Mixed Integer Linear Programming (MILP) by finding the C1S submatrix. Given a (0, 1)−matrix, we consider the C1S problem which aims to maximize the number of columns having only one block of consecutive 1's in each row by permuting them. We suggest an evolutionary approach to solve the problem. The Genetic Algorithm (GA) is the one proposed here to rearrange the columns of the matrix by pushing them in large blocks of 1's.

We also consider the Consecutive Blocks Minimization (CBM) problem which is related to C1S. A new procedure is proposed to improve the C1S submatrix, which is the column insertion approach. Moreover, preprocessing by minimum degree ordering is also used.

On the other hand, we suggest another approach to solve the C1S. It is using the MVEE problem. To pave the way we first solve the problem. Given a set of points $C = \{x_1, x_2, ..., x_m\} \subseteq R^n$, what is the minimum volume ellipsoid that encloses it? Equally interestingly, one may ask: What is the maximum volume ellipsoid that can be embedded in the set of points without containing any? These problems have a number of applications beside being interesting in their own right. If one requires that at least k of m points, $k < m$ be enclosed in the minimum volume ellipsoid, then the problem becomes more difficult but has the potential, once solved, to detect outliers among the n points. We suggest an evolutionary-type approach for their solution. We will also highlight application areas and include computational results.

Acronyms

B&B: Branch and Bound

C1P: Consecutive Ones Property

C1S: Consecutive Ones Submatrix

CBM: Consecutive Block Minimization

COLAMD: Column Approximate Minimum Degree

EA: Evolutionary Algorithm

ES: Evolution Strategy

GA: Genetic Algorithm

ILP: Integer Linear Programming

LP: Linear Programming

LVEE: Largest Volume Enclosing Ellipsoid

MDOA: Minimum Degree Ordering Algorithm

MILP: Mixed Integer Linear Programming

MVEE: Minimum Volume Enclosing Ellipsoid

MVE: Minimum Volume Ellipsoid Estimator

MVEH: Minimum Volume Enclosing Hyper-Rectangle

MVIE: Minimum Volume Inscribed Ellipsoid

PPA: Plant Propagation Algorithm

PSO: Particle Swarm Optimization

SCP: Set Covering Problem

SA: Simulated Annealing

TS: Tabu Search

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

This thesis is in two parts, the first deals with the Consecutive Ones Submatrix (C1S) problem; the second deals with the Minimum Volume Enclosing Ellipsoid (MVEE) problem. The C1S problem for a (0, 1)−matrix has a special variant called the Consecutive Ones Property (C1P) problem. This property means that it is possible to permute the matrix columns in such a way that every row has at most one block of consecutive ones. The C1P property has a long history. It is important since it indicates that optimization problems based on matrices with this property are easier to solve than the model suggests. For a $(0, 1)$−matrix A, the C1S is that of finding the largest number of columns that forms a submatrix A' having the C1P property. The latter problem has applications in computational biology, in the physical mapping problem [3–6], and in the seriation problem [7] that arises in other applications such as musical chronology and psychometrics. There are solution approaches for it, but there is room for improvement. Moreover, most of the studies of the problem use exact solution methods.

In terms of motivation and contribution, herein, we investigate a novel heuristic approach. We lay the foundation for efficiently finding the solution to the C1S problem. The consequence of solving this problem is to transform Integer Linear Programming (ILP) with binary matrices into Mixed Integer Linear Programming (MILP). In this thesis we discuss the complexity of the two problems. It is known that ILP is NP-complete [8,9]. A question

that arises here, is whether the transformation of ILP into MILP, itself an NP-complete problem in the decision version [10]. We provide an estimate for reducing the complexity via the transformation. Also, we design two algorithms a heuristic and exact to transform the ILP by finding the C1S submatrix. We test these algorithms on real-life instances with almost C1P.

Another innovation in this thesis is using ellipsoids of minimum volume to solve the C1S problem. To pave the way, we start with solving the Minimum Volume Enclosing Ellipsoid (MVEE) problem. We then show how this can be used as a solution approach to the C1S problem.

The MVEE is an ellipsoid approximation problem. It originated from work by Fritz John [11] who proved that a nonempty subset of points that is contained in R^n can be $n-$rounded by an ellipsoid. Finding a MVEE for a set of points in R^n is equivalent to computing the MVEE around the polytope specified by the convex hull of the set. This problem arises in many different applications such as computer graphics, convex optimization, and pattern recognition. The MVEE for detecting outliers is a variation of this problem which also has several applications [12].

The problem of MVEE has been solved exactly since it is convex. However, some approaches lose efficiency in high dimensions and for large sets of points. So, again there is a need for new algorithms. This encourages us to study the problem in a totally different way and introduce a new solution approach and compare the results with an exact method. We apply this approach to a large number of points in high dimensions. Although, the results are not so accurate like the exact method, but the latter can not deal with such large number of sets in $d > 20$. The MVEE heuristic approach is also applied to other problems: the Minimum Volume Ellipsoid (MVE) estimator problem, the Largest Volume Enclosing Ellipsoid (LVEE) problem, the Minimum Volume Enclosing Hyper-Rectangle (MVEH) problem, and the MVEE containing a dynamic set of points. With respect to the MVE estimator problem we also use a heuristic approach which gives good results with better time in comparison with previous methods.

1.2 Thesis organization

The rest of this thesis is organized as follows. Chapter 2 introduces the formal definition of the C1S problem with some fundamental notions and a few application examples. It reviews some relevant methods for its solution. It then covers Combinatorial Optimization and the Genetic Algorithm (GA).

Chapter 3 introduces the first version of the proposed heuristic approach. It contains all necessary algorithms for its successful implementation. It shows how the C1S problem is gradually solved using different algorithms which help to improve the solution. It also covers the Consecutive Blocks Minimization (CBM) problem. The solution of the two problems is improved locally via two heuristics: the column insertion procedure and the minimum degree reordering algorithm. Moreover, it discusses aspects of the complexity of the algorithms that are used to solve the C1S problem.

Chapter 4 provides the transformation of a ILP with a $(0,1)-$matrix into an equivalent MILP also with a $(0,1)-$matrix. It reviews some complexity results relevant to the GA and investigates the complexity of our heuristic. Then it discusses the gains in terms of the complexity of this conversion.

Chapter 5 introduces solution approaches to the MVEE and the MVE estimator problems in higher dimensions. It provides the formal definitions of the problems together with brief historical surveys.

Chapter 6 introduces applications of the MVEE heuristic approach: computing an enclosing minimum volume hyper-rectangle for a set of points, the enclosing ellipsoid of a $(0,1)-$matrix, and that of a dynamic set of points.

Chapter 7 is the conclusion. It also discusses some related issues worth to study further.

Chapter 2

The C1S problem

2.1 Introduction

The Consecutive Ones Property (C1P) is the property of matrices whose entries are 0 or 1. A binary matrix is C1P for rows when its columns can be permuted so that in each of its rows the 1's appear consecutively. In ILP the C1P property is important since it indicates that the problems based on matrices with this property are easier to solve than the model suggests. Indeed, such matrices are totally unimodular [13, 14]. It appears in many applications such as railway optimization [15], information retrieval [16], and scheduling [17]. The C1P property is also used for ancestral genome reconstruction [18] and the construction of physical maps from data hybridization [4, 19]. In graph theory it helps detect interval graphs, [20, 21].

C1P has been widely investigated. Kendall [22] attributed the first mention of the property to Flinders Petrie, an archaeologist of the 19th century, [23]. Many heuristic methods were established for the problem of chronologically ordering archaeological deposits before the first polynomial complexity solution was proposed by Fulkerson and Gross [20]. In 1972, Tucker [2] showed a characterization of the problem using forbidden submatrices. Later, Booth and Lueker [24] provided a linear-time algorithm for it. They found a permutation that transforms a (0, 1)-matrix into one with C1P. Their linear time sequential algorithm is based on the *PQ*-tree data structure. A (0, 1)-matrix has C1P if and only if its *PQ*-tree exists.

Procedures based on *PQ*-trees are regarded as complicated, however, in 1992, without using *PQ*-trees, Hus [25] provided a simple test for the C1P property of a (0, 1)-matrix. He introduced a linear complexity solution for the problem. A simplification of the C1P test can for instance simplify algorithms for the planar graphs recognition and the interval graphs. In 1998, Annexstein and Swaminathan [13] introduced an algorithm for testing C1P property in parallel using the so called Tutte decomposition, which decomposes the graphs into 3-connected components. It is a divide and conquer method that adopts a graph-theoretic data structure. In 2000, Habib et al. [26], used the Lexicographic Breadth First Search (Lex-BFS) and the partition refinement with pivots to design a simple algorithm for testing matrices for the C1P property.

2.2 Some related problems

A related problem is the Gapped $(\alpha, \beta)-$C1P problem which is an extension of the $(0, 1)-$C1P problem. For a binary matrix and two integers α and β, check if there is a permutation of the matrix columns such that there is at most β sequences of $1's$ in each row and there is a gap of no more than α $0's$ that separates any two consecutive sequences of $1's$. This problem is NP-complete when $\beta = 2$ and $\alpha \geq 2$ [27]. Chauve et al [28] conjectured that for $\beta \geq 2$, $\alpha \geq 1$, and $(\alpha, \beta) \neq (1, 2)$ the $(\alpha, \beta)-$C1P problem is NP-complete.

Another related problem is the so called Consecutive Block Minimization (CBM). The goal is to minimize the number of blocks of 1's via permuting the columns of the matrix. Haddadi and others [29], provided a polynomial-time local-improvement heuristic for the problem.

2.3 Some applications

2.3.1 The Petrie Seriation Problem

The Petrie Seriation (or Sequence-dating) Problem [22], is to chronologically order grave sites in a cemetery where each grave includes (or does not include) a number of stylistic artifacts of a period. It is modelled mathematically by an incidence matrix A the columns

of which correspond to specific artifact types and its rows are the grave sites. Gargano and Lurie [30], provided a hybrid evolutionary approach for solving this problem. They provided a permutation of rows to transform A into a matrix which has a consecutive ones block in each column; this is called a Petrie matrix. Finding this permutation is equivalent to showing that a $(0,1)-$matrix has the C1P property.

2.3.2 The consecutive retrieval property (CRP)

In 1972, Ghosh proposed the property of the consecutive retrieval in terms of its application to file organizations, for example in a computer file system. Dividing the information into records is one of many procedures of storing information in the storage system. Let $\mathcal{R}$ be a set of records that is called a file which has a particular structure that differs from data to data. This file can be stored on a storage medium $\mathcal{S}$ that is represented as a set of storage positions. Retrieval of pertinent or relevant records is an essential use of storing records in a storage medium. This is drawn by a set of queries $\mathcal{Q}$ which may have several kinds of structures relying on what is needed. Organizing the file $\mathcal{R}$ on the storage medium such that every query in $\mathcal{Q}$ has an answer is called a file organization. Some types of file organization have been improved for linear storage medium. The best of them is when there is an organization between $\mathcal{Q}$ and $\mathcal{R}$ such that for every query in $\mathcal{Q}$, the relevant records are stored in consecutive storage positions in $\mathcal{R}$ without redundancy. If so, the set of queries $\mathcal{Q}$ is said to have the Consecutive Retrieval Property (CRP) with respect to the set of records $\mathcal{R}$ and such an organization is called a Consecutive Retrieval File Organization (CRFO) [31,32].

Under this assumption, it can be associated an incident matrix with $\mathcal{R} = \{r_\alpha\}_{\alpha=1}^{m}$ and $\mathcal{Q} = \{q_\beta\}_{\beta=1}^{n}$. Let A be an $m \times n$ matrix where the rows correspond to the m records of $\mathcal{R}$ and the columns to the n queries of $\mathcal{Q}$. The $a_{\alpha\beta}$ member of A includes a 1 if r_α is relevant to q_β and 0 otherwise. So A is of the form:

$$A = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_m \end{array} \begin{array}{c} \begin{array}{ccccc} q_1 & q_2 & q_3 & \dots & q_n \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & \dots & 1 \\ 0 & 1 & 1 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix} \end{array}$$

Variations of this problem are investigated in different contexts. For instance, relying on the concept of the C1P, Fulkerson and Gross [20], proposed a procedure to find whether the fine structure of genes is consistent with a linear model of the gene [33]. They provided the most general solution of the CRP problem.

2.3.3 Physical mapping of DNA

Fulkerson and Gross [34] used the C1P to address the fine structure of a gene relying on work by Benzer [35], also see [6, 13, 19, 36]. A physical mapping of the human DNA sequences, say an entire genome or chromosome, is found by assigning the linear order of a set of markers[1]. One of the known methods relies on clone maps which are constructed via a clone library. This library contains a large number of clones that represent short fragments of DNA. These fragments may overlap. To test whether pairs of clones overlap (intersection) or not, the hybridization of short probes[2] is used. A clone is detected by a number of probes and the probe that hybridizes[3] to the clone is identified. For a problem with m clones and n probes, the experimental information is an $m \times n$ binary matrix $A = a_{\alpha\beta}$; if a clone hybridizes with a probe then $a_{\alpha\beta} = 1$ and $a_{\alpha\beta} = 0$ otherwise. Now, to find the correct order of the probes on a certain clone these probes (the 1's) should be consecutive

[1] A marker is a uniquely recognizable DNA segment; the types of markers are ranged between 1 to 300-400 nucleotide bases in size such as probes, genes, ect.

[2] Probes are short segments of DNA and called sequence tagged sits (STSs) which are located in a single sit of the genome [37].

[3] A probe is called hybridize to a clone if it matches a substring in that clone.

in such a way that in every row the zero does not appear in a clone. However, if a zero appears then the clone has an error and it becomes necessary to deal with it. This is either by discarding the rows which have inconsecutive blocks of ones or the columns need to be ordered to get a consecutive block in each row. If the matrix A has the C1P then the empirical information is error-free. See the matrix below for illustration.

$$\begin{array}{c} \\ \text{Clones} \end{array} \begin{array}{c} \text{Probes} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}, \qquad \begin{pmatrix} \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\ 0 & 1 & 1 & 1 & 0 \\ \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{0} \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

2.4 Fundamental aspects

Definition 2.4.1 *A matrix with only 0 or 1 entries is called a* $(0, 1)-$*matrix or a binary matrix. Given an* $(0, 1)-$*matrix A with m rows and n columns, the entry of the matrix is denoted by* a_{ij} *in the* i^{th} *row and the* j^{th} *column. A row (column) of a matrix A is called a* $0-$*row (column) if all its entries are zeros. Complementing a row or column of A means that all* $0-$*entries are replaced by* 1*'s and all* $1-$*enteries replaced by* 0*'s.*

Remark: In the rest of this work, we assume that all $(0, 1)-$matrices include at least one element 1 in each row and column.

Definition 2.4.2 *The permutation of rows (or columns) of a matrix A is called row permuting* (*column permuting*) *of the matrix.*

Definition 2.4.3 *If A′ is a permutation of the rows and columns (it can only be rows or only be columns) of A then the matrices A and A′ are said to be isomorphic.*

Definition 2.4.4 *A matrix A′ is said to be a submatrix of A if a number of rows and columns is*

chosen from A such that deleting all except them from A will result in A'. This is true for choosing a subset of columns of A and deleting the reminder of columns.

This concept can be extended to say A', a submatrix of A, if there is a number of rows and columns of A chosen such that deleting all except them from A will produce a matrix that is isomorphic to A'. This is also true for choosing only columns. If A is isomorphic to a submatrix A' then the latter is contained in A and induced by the chosen rows and columns.

2.4.1 Definitions in graph theory and terminology

Definition 2.4.5 *A graph is a pair $(V, E) = G$ that contains a finite set of vertices (nodes) V, and a finite set of edges E. If any two vertices of graph G are joined by at most one edge then G is called a simple graph.*

Definition 2.4.6 *A graph $G = (V, E)$ is said to be bipartite if the set of vertices V is divided into two subsets of vertices V_1, V_2 such that every edge in E connects a vertex in V_1 to a vertex in V_2 and is denoted by a triple $G = (V_1, V_2, E)$.*

Remark: Each vertex in V_1 is called a row vertex and each vertex in V_2 is called a column vertex.

A binary matrix A can be seen as the adjacency matrix of a bipartite graph $G(A) = (V_{1A}, V_{2A}, E_A)$ which is called the *representing graph* where V_{1A} is a row set and V_{2A} is a column set. For every row and every column of A there is a vertex which belongs to $G(A)$, and the vertices that correspond to the i^{th} row and the j^{th} column of A can be joined by an edge in $G(A)$ for every 1-entry a_{ij} in A. Let A be a matrix represented by a graph $G(A)$, denote a row or column of A as a *line*. If two vertices are connected by a path in $G(A)$ corresponding to two lines of A then the lines are connected in A. If each two lines in a submatrix $A^{'}$ of A are connected in $A^{'}$, then $A^{'}$ is called connected. A matrix $A^{'}$ is called a component of A if it is the maximal submatrix of A. Each component of A corresponding to a connected component of $G(A)$ and each submatrix of A corresponds to an induced subgraph of $G(A)$ [38]. Figure 2.1 shows a clarification of the components of a matrix. We refer to [39, 40] for a general introduction to graph theory.

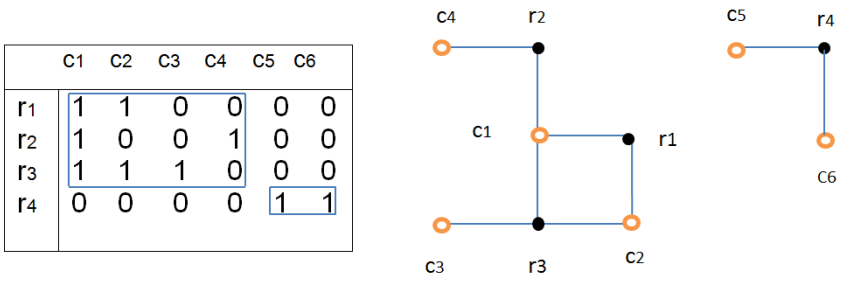

Figure 2.1: *Bipartite graph representing a matrix with two components [1]*

2.5 The C1P and consecutive ones submatrix

Definition 2.5.1 *A maximal set of consecutive one entries (zero entries) in a row of a* $(0, 1)$*−matrix* A *is called a block of 1's (block of 0's).*

Definition 2.5.2 *A 2-dimensional* $(0,1)$*−matrix has the consecutive ones property for rows if its columns can be permuted such that all the* 1*'s are consecutive in every row.*

Remark: The C1P for the columns is similar, through matrix transposition.

Definition 2.5.3 *Given a 2-dimensional* $(0,1)$*−matrix, the problem of finding the maximum set of columns such that the resulting submatrix has the C1P is called the Consecutive Ones Submatrix (C1S) problem.*

Definition 2.5.4 *Given a matrix* A*, a permutation of its columns that leaves the ones in consecutive arrangement in each row is called a valid permutation. If a* $(0, 1)$*−matrix can be reordered by such a permutation then the matrix can be turned into a matrix with C1P.*

The C1P can be seen as the property of a collection of sets. A set of rows of $A = (a_{ij})$ is denoted $R_A = \{r_i\}_{i=1}^m$, and the set of its columns is $C_A = \{c_j\}_{j=1}^n$. The set R_{A_i} for any i can be regarded as a subset of columns such that $\Re = \{R_{A_1}, ..., R_{A_m}\}$ be the collection of all subsets of C_A; we can say that $R_{A_i} = \{c_j \,|\, a_{ij} = 1\}$ for $i = 1, ..., m$. The total sum of nonzero entry in A will be $k = \sum_{i=1}^m |R_{A_i}|$. A $(0, 1)$−matrix A with only $|C_A| \leq 2$ is C1P matrix.

Definition 2.5.5 *For a finite set* C_A*, its permutation is a bijection* $\delta : \{1, 2, ..., n\} \rightarrow C_A$.

Let $P(C_A)$ represent the power set of C_A. For simplicity, a set can be written here as a list of its elements. For example $B = \{s, h, k, l\}$ can be written as $B = hlsk$. Given C_A, a

subset B of C_A, and a permutation δ of the elements of C_A, when the elements of B appear consecutively in δ we say B is consecutive in δ. For example, if $C_A = uvwxyz$ and $\delta = yzwvxu$, then the subset $B = wvx$ is consecutive in δ, while $D = yzu$ is not. Given a pair $(C_A, \mathfrak{R})$, with $\mathfrak{R} \subseteq P(C_A)$, a permutation δ of C_A is valid if all sets $B \in \mathfrak{R}$ are consecutive in δ. If there is at least one valid permutation, then the pair $(C_A, \mathfrak{R})$ will have the C1P. An application of the property can be shown as follows: let $A = (a_{ij})$ be a $(0, 1)$-matrix, and let C_A be the set of its columns. Each row i can be viewed as the set $B \subseteq C_A$ created by the columns j such that $a_{ij} = 1$. Matrix A can be viewed as a pair $(C_A, \mathfrak{R})$ where $\mathfrak{R}$ represents the collection of the subsets of C_A which correspond to its rows. In the following example we recognize the valid and non valid permutations that satisfy the property of C1P [23].

Example:
Consider the $(0, 1)-$matrix A below with its labeled columns. A permutation of the columns illustrates that A is a C1P matrix:

$$A = \begin{array}{c} \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{ccccc} u & v & w & x & y \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}, \qquad A' = \begin{array}{c} \\ f \\ g \\ h \end{array} \begin{array}{c} \begin{array}{ccccc} u & x & v & y & w \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

For the set $C_A = \{u, x, v, y, w\}$, we have the subsets $u_1 = \{v, x\}$, $u_2 = \{v, w, y\}$, $u_3 = \{u, x\}$ where $\mathfrak{R} = \{u_1, u_2, u_3\}$. The permutation $\delta = uxvyw$ is valid and displays that the pair $(C_A, \mathfrak{R})$ has the property of C1P. The bipartite graphs of A before and after the reordering is shown in Figure 2.2. The one to the left corresponds the matrix before permutation. Black and orange vertices correspond to rows and columns, respectively.

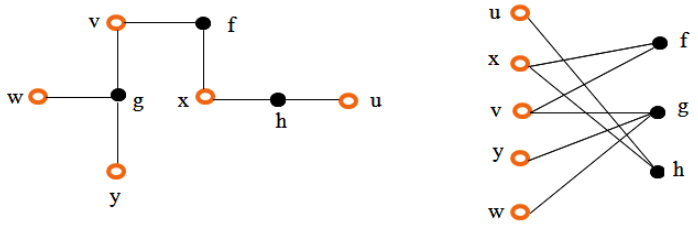

Figure 2.2: *Bipartite graphs corresponding to the matrix A*

However, for the following matrix A, there is no permutation which gives the property.

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

The C1P is a desirable property that often leads to efficient algorithms. A great deal of interest in matrix modification and transformation into a binary matrix satisfying the C1P has been shown recently. These transformations can be cast as the following problems [1, 41, 42].

- Find the maximum number of columns of a (1, 0)−matrix A which induces a submatrix with C1P. This problem is known as the Max-C1S-C (Consecutive Ones Submatrix by Column).

- Find the maximum number of rows of a (1, 0)−matrix A which induces a submatrix with C1P. This problem is known as the Max-C1S-R (Consecutive Ones Submatrix by Row).

- Find the minimum-cardinality set of columns to delete such that the resulting matrix has the C1P. This problem is known as Min-C1S-C (Consecutive Ones Submatrix by Column Deletion).

- Find the minimum-cardinality set of rows to delete such that the resulting matrix has the C1P. This problem is known as Min-C1S-R (Consecutive Ones Submatrix by Rows

Deletion).

- Find the minimum-cardinality set of 1-entries in the matrix which, when flipped into 0 results into a matrix with C1P. This problem is known as the Min-C1-1E (Consecutive Ones by Flipping 1-Entries).

Note that the first problem is equivalent to the third and the second is equivalent to the fourth. All them are posted as decision problems. In complexity theory, there is no variance in their computing hardness. Finding the optimal solution of the Max-C1S-C (Max-C1S-R) leads to the optimal solution for the problems Min-C1S-C (Min-C1S-R) and vice versa [38]. We are concerned with the maximization versions of the problem particulary that of the Max-C1S-C problem to which we refer as C1S. These problems are generally NP-hard[4] even for very sparse matrices.

Let $(\alpha, \beta)-$matrices be $(0,1)-$matrices with at most α $1's$ in each column and β $1's$ in each row. Garey and Johnson [10] mentioned that the decision version of the C1S problem is NP-complete. Later, Hajiaghayi and Ganjali [43] found that the C1S problem is NP-hard for $(2, 4)-$matrices. They also proved that for $(2,2)-$matrices it can be solved in polynomial time. So, their work raises the question of whether the C1S problem stays NP-complete or not for both $(2, 3)-$matrices and $(3, 2)-$matrices. Tan and Zhang [36] answered the question and showed that indeed the decision version of $(2,3)-$matrices and $(3,2)-$matrices having the C1S stays NP-complete. For the NP-hard cases of the C1S they found that the C1S problem is polynomial-time 0.8-approximatable for $(2,3)-$matrices that have no identical two columns. They also proved that the C1S problem is 0.5-approximatable for $(3,2)-$matrices and for $(2,\infty)-$matrices in general. Moreover, they showed that for $(\infty,2)-$matrices there exists an $\varepsilon > 0$ such that the approximation of the C1S problem within a factor of n^{ε} is NP-hard.

Traditional solution approaches to the C1S problem rely on detecting the so called Tucker forbidden submatrices [2, 36, 42, 43]. Dom *et al* [42] worked on the problems of

[4] A problem belongs to the class NP if a solution to it can be generated and verified quickly, i.e. in polynomial time. A problem is NP-hard if the entire class of NP problems polynomially reduce to it. A problem A reduce to B if it can be constructed for any instance of A an equivalent instance of B. it is believed that NP-hard problems are intractable, i.e. that there is no efficient algorithm to solve them. To prove this is the most important challenge in computational theory. See [10] for an introduction to complexity theory.

approximation and parameterized complexity of C1S problem. They have found a technical solution which relies on effectively detecting forbidden submatrices, which are given in the following section.

2.5.1 Forbidden submatrices

The class of forbidden submatrices that is known as Tucker Patterns are a certificate that a $(0,1)-$matrix does not have the C1P. Matrices that do not have any of the structures of these submatrices should be C1P matrices. Forbidden submatrices can be represented by graphs. In particular, a bipartite graph $G(A) = (V_{1A}, V_{2A}, E_A)$ is associated to a $(0, 1)-$matrix A. Tucker's characterization relies on the definition of the *asteroidal triples* that is due to Lekkerkerker and Boland [44].

Definition 2.5.6 *Let $G = (V, E)$ be a simple graph. A subset of three vertices of V form an asteroidal triple if there exists a path between any two of them in such a way that no vertex in the path is adjacent to the third vertex.*

Tucker proved that a $(0, 1)-$matrix A has the C1P if there is no asteroidal triple of $G(A)$ between any three vertices of V_{2A} (or V_{1A}). Notice that, Tucker supposed the C1P for columns set V_{2A}, while we consider the C1P for rows set V_{1A}. So, here we exchange the roles of V_{1A}, V_{2A} compared to [2].

Theorem 2.5.1 *In the bipartite graph $G(A) = (V_{1A}, V_{2A}, E_A)$, the set of vertices V_{2A} contains no asteroidal triple if and only if $G(A)$ contains none of the bipartite subgraphs: A_{I_n}, A_{II_n}, A_{III_n}, $(n \geq 1)$ A_{IV}, and A_V as illustrated in Figure 2.3.*

The set of vertices V_{2A} of the bipartite graph $G(A) = (V_{1A}, V_{2A}, E_A)$ includes an asteroidal triple if and only if $G(A)$ includes one of the represented graphs as an induced subgraph. Figure 2.3 shows that the triple with orange vertices x, y, z are an asteroidal triple in each graph. The set of forbidden submatrices that Tucker used to distinguish the binary matrices that have the consecutive ones property is given in the following theorem.

Theorem 2.5.2 *([2]) A $(0,1)-$matrix has the C1P for rows if and only if it contains none of the matrices A_{I_n}, A_{II_n}, A_{III_n}, $(n \geq 1)$ A_{IV} and A_V (see Figure 2.4).*

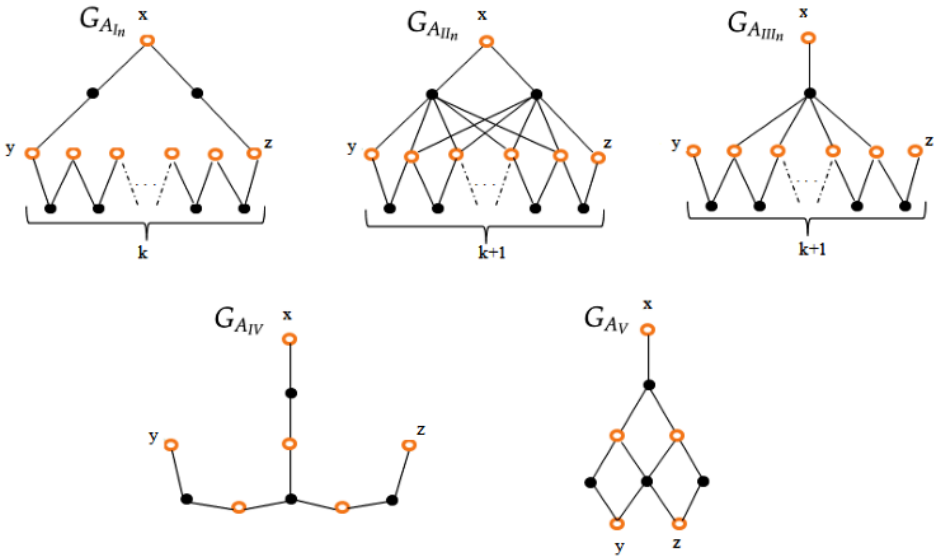

Figure 2.3: *Bipartite graphs corresponding Tucker Forbidden submatrices [2]*

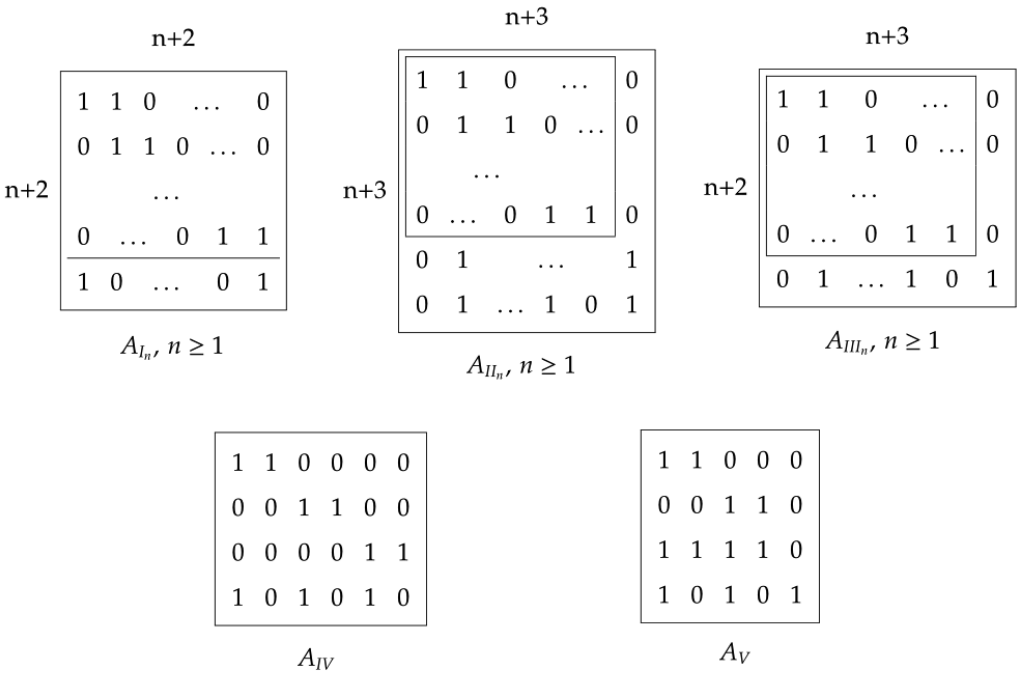

Figure 2.4: *The forbidden submatrices of Tucker [1, 2]. The graphs and the matrices are corresponding*

2.6 Characterizing a C1P matrix

Many types of trees have been proposed starting with the PQ-tree of Booth and lueker, MPQ-tree of Korte and Möhring, gPQ-tree of Novick, PQR-tree of Meidanis et al., and the generalized PQ-tree of McConnell. All the previous trees are a variation of the first.

2.6.1 PQ-trees

Booth and lueker [24] improved a data structure to represent (0, 1)−matrices called PQ-trees. They introduced a linear-time algorithm for recognizing C1P matrices for columns. The PQ-tree is proposed to represent the set of permitted permutations of certain subsets of C_A that should occur consecutively. The class of PQ-trees over a set of columns is defined to be the rooted, labeled trees whose leaves are elements of the set of columns. Each one of these elements is represented by a single leaf node that does not have children. The internal (non-leaf) nodes of the PQ-tree are labeled as

- P-nodes which have at least two children are drawn as a circle,
- Q-nodes which have at least three children are drawn as a rectangle. See Figure 2.5.

The leaves of the two types of nodes are one-to-one corresponding to the matrix columns. The C1P matrix for rows results by ordering the matrix columns according to the order of the leaves in the PQ-tree. Let T be a PQ-tree, the leaves of T for a matrix A can be read as a sequence from left to right to represent the consecutive ones ordering for A columns, (see Figure 2.6). If T represents a valid permutation for a C1P matrix A, then T can encode all the valid permutations of A columns, thereby transforming the tree into any equivalent tree say T'. The transformation is done by suggesting two node ordering operations:

- The children of a P-node can be arbitrary reordered.
- The order of the children of a Q-node can be reversed, but not reordered.

Building a PQ-tree of a matrix A begins with building a universal PQ-tree; we call it T. It contains a single P-node for its root and a leaf node for every column of A. The algorithm checks the rows of the matrix one by one and at each step modifies the current PQ-tree to

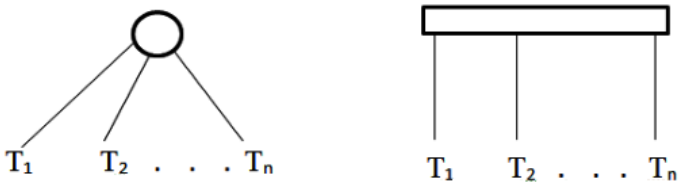

Figure 2.5: *A P-node is represented by a circle, where its children are the PQ-trees $T_1, T_2, ..., T_n$ to the left, while a Q-node is represented by a rectangle and its children are shown to the right*

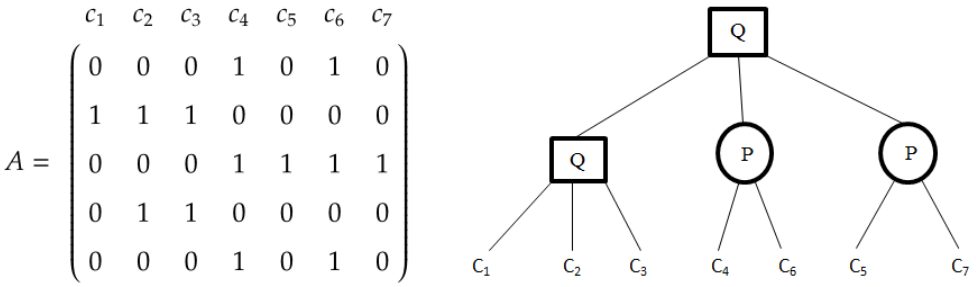

Figure 2.6: *A PQ-tree representing the columns permutation of matrix A*

a new one for the matrix that is induced by the rows examined so far; or it announces that the rows do not have the C1P. Note, a null tree represents an empty set that has no nodes.

In each step, the T-tree is restricted by the subsets of $\Re$ that are processed in such a way ordering their children are consistent with all the elements of the collection $\Re$. The PQ-tree algorithm for a C1P matrix has linear time $O(m + n + k)$ for an $m \times n$ matrix where k is the total number of 1-entries of A, i.e. $k = \sum_{i=1}^{m} |R_{A_i}|$ as mentioned before.

2.6.2 gPQ-trees

Novick [45], investigated a related problem to the consecutive ones problem, the *trivial intersection proplem*. It can be stated for a (0, 1)−matrix A as follows.

Let C_A be a finite set and $\Re$ be the collection of subsets over C_A. Does any of the subsets of C_A have a trivial intersection with every member of $\Re$?.

Novick [45], generalized the PQ-tree, proposed by Booth and Lueker into the gPQ-tree which is simpler. By generalizing the concept of the PQ-tree, the subsets of C_A that have trivial intersection with every element of $\Re$ can be simply represented. The collection of

these subsets is denoted by $N_A(R)$. To build the gPQ-tree, the following subsets of C_A should have trivial intersections (of columns) with the members of $\mathfrak{R}$:

- The leaves set that are the descendants of some Q-node,
- For any subset of children of a P-node, the leaves set that are descendants of this subset,
- The set of columns C_A, the empty set, and $\{c_j\}_{j=1}^{n}$, where each c_j is a single element of C_A.

Solving the trivial intersection problem starts with constructing the universal gPQ-tree, call it T, which is similar to that of Booth and Lueker [24]. The set C_A is suggested to be the children of a P-node which is the root of T. This tree is reduced with respect to the elements of $\mathfrak{R}$ one by one in each step of the reduction. The output of this process is the gPQ-tree that represents the $N_A(R)$. This reduction costs $O(mn)$.

This reduction can be used for solving the C1P problem by reducing the universal PQ-tree T with respect to elements of $\mathfrak{R}$. This is done in the same way that is described above for the gPQ-tree. If the resulting T of the reduction of the universal PQ-tree is the null-tree then the problem does not have the C1P; otherwise it satisfies the property. However, even when reducing the PQ-tree T produces a null-tree, the gPQ-tree T is still defined. This is what makes them a true generalization, the following theorem shows that.

Theorem 2.6.1 *[45] If the result of the reduction of the universal PQ-tree by elements of $\mathfrak{R}$ is a non-null tree T, then if the left-to-right order of the children in T is ignored, the resulting gPQ-tree is equivalent to the result from the reduction of the universal gPQ-tree by elements of $\mathfrak{R}$.*

2.6.3 PQR-trees

Meidanis and Munuera [46], proposed a PQR-tree for a $(0, 1)-$matrix which is a modification of a PQ-tree by Booth and Lueker [24]. The PQR-tree is different from the latter in that it can exist for every subset of C_A, even if there is no valid permutation of the $(0, 1)$-matrix. This means we can define this tree for a matrix that does not have the C1P. Here, the number of better organized patterns that are used for the PQR-tree is less than that for the PQ-tree.

A PQR-tree over a set of columns C_A is a rooted tree which comprises P, Q, R-node and leaves. These types have limiting conditions:

- The P and Q are similar to those in PQ-tree,
- A R-node has at least three children that can be arbitrarily permuted,
- Each leaf is in correspondence with a member of C_A.

A PQR-tree without the R-nodes is a PQ-tree; these nodes prevent the matrix from satisfying the C1P, they are the reason of the absence of the property. The matrix will have the C1P when a single column is taken off from each R-node. The total number of operations of the algorithm to construct the PQR-tree is $O(m + n + k)$ [47].

2.6.4 Generalized PQ-trees

McConnell [48], first proposed generalized PQ-trees. He found a different characterization of consecutive ones matrices with a linear time algorithm. In case the matrix does not have the C1P, the algorithm provides us with a certificate of size $O(n)$ for the absence of the C1P. This certificate relies on the so-called consecutive ones relation (C1-*relation*).

Definition 2.6.1 *For a domain C_A and a family $\mathcal{F}$ of subsets of C_A, a C1-ordering $\{c_j\}_{i=1}^{n}$ of C_A defines the C1-relation $R = \{(c_{j_1}, c_{j_2}) \mid j_1 < j_2\}$.*

McConnell defined the incompatibility graph notion for a (0, 1)−matrix.

Definition 2.6.2 *The incompatibility graph for an $m \times n$ matrix A with a set of rows $R_A = \{r_i\}_{i=1}^{m}$, and a set of columns $C_A = \{c_j\}_{j=1}^{n}$ is an undirected graph $G = (V, E)$ with a set of vertices $V = \{(c_{j_1}, c_{j_2}) | \, j_1, j_2 = 1, ..., n, \;\; j_1 \neq j_2\}$ and a set of edges E where the pairs are formed as follows:*

- $\{\{(c_{j_1}, c_{j_2}), (c_{j_2}, c_{j_1})\} | \, j_1, j_2 = 1, ..., n, \;\; j_1 < j_2\}$
- $\{\{(c_{j_1}, c_{j_2}), (c_{j_2}, c_{j_3})\} | \, c_{j_1} \neq c_{j_3} \bigwedge \exists \, i \in \{1, ..., m\} : (a_{ij_1} = a_{ij_3} = 1 \bigwedge a_{ij_2} = 0)\}$.

It is clear that each unordered pair $\{c_{j_1}, c_{j_2}\}$ corresponds to two vertices and each edge in the incompatible graph corresponds to two relative orderings of the columns that are

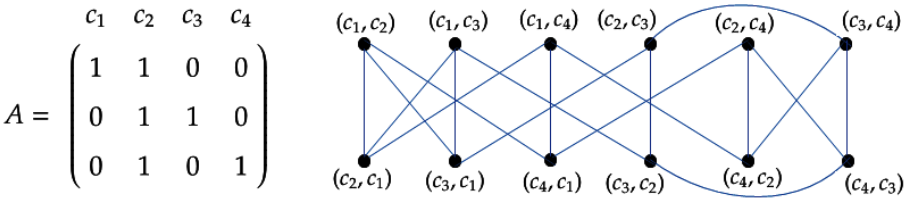

Figure 2.7: *The incompatibility graph of the matrix A, the smallest odd cycles of length 7*

not in consecutive positions of the matrix; see Figure 2.7. Assume that the C1-ordering is unknown. Then it can be seen that the pairs (c_{j_1}, c_{j_2}) and (c_{j_2}, c_{j_1}) are incompatible because they do not appear in the same relation. Also, for the pairs (c_{j_1}, c_{j_2}), (c_{j_2}, c_{j_3}) they can not appear in the same C1-ordering because c_{j_2} will appear between c_{j_1}, c_{j_3} which returns inconsecutive orderings. So the pairs are incompatible. The C1-relation should have no incompatible pairs. If the matrix does not satisfy the C1P then the incompatibility graph is not a bipartite graph and it should have an odd cycle of length at most $n + 3$, [1].

Theorem 2.6.2 *([1], Theorem 2.6) A* (0, 1)*−matrix A is C1P matrix if and only if its incompatibility graph is bipartite and if A does not have the C1P, the incompatibility graph has an odd cycle of length at most* $n + 3$*.*

Despite that all the tree types that are mentioned above, the *PQ*-trees data structure is still regarded as complicated, so alternative studies [13, 25, 26] used alternative data structures.

2.6.5 Characterizing matrices with almost C1P

A (0, 1)−matrix, in most rows, the ones of which are consecutive and only a few blocks of consecutive ones are in the remaining rows, is called a matrix with almost C1P. Characterizing matrices with the C1P is cheaper than characterizing matrices with almost C1P. Characterizing the C1P matrices can be done in linear time, but the decision problem of determining whether a permutation of the columns of $(0, 1)-$matrix such that each row has at most r consecutive ones blocks is NP-complete [49, 50]. This is true for $r \geq 2$, [5, 6, 51]. Booth and Lueker [24] provided a linear time procedure when $r = 1$, [38].

2.7 Combinatorial optimization

Discrete optimization problems can be written as

$$\begin{aligned} max\ (or\ min) \quad & \beta(U) \\ s.t \quad & U \in K, \end{aligned}$$

where U is an arrangement, K is the set of feasible arrangements, and $\beta(U)$ is the value computation of members of K [52]. Some particular discrete optimization problems are as follows.

1. Traveling Salesman Problem (TSP). In a graph (directed or undirected) associated with weights on the arcs, find a closed traversal that contains every node of the graph just once and that has minimum total arc weight.

2. Postman Problem. In a weighted graph (directed or undirected), find a traversal that contains every arc in the graph at least once and that has minimum total weight.

3. Max Clique Problem. Given an undirected graph G and a positive integer k. Find the largest complete subgraph in G that contains exactly k nodes.

4. Set Packing Problem. Given a finite set C, a collection of subsets of C, each subset associated with a value. Find a maximum value collection of the subsets that contains every object of C at most once.

5. Set Partitioning Problem. Given a finite set C, a collection of subsets of C, each subset associated with a weight. Find the minimum weight collection of the subsets that forms a partition of C.

6. Maximum Flow Problem. In a graph (directed or undirected) associated with capacities on the arcs, find a maximum arc flow between two associated nodes that conforms to capacities and has total flow into all other nodes equal to total flow out.

7. Shortest Path Problem. In a graph (directed or undirected) associated with weights or length on the arcs, find a minimum total length nonrepeating sequence of arcs that connects two associated nodes and that conforms to any directions on arcs.

Other problems are: Set Covering, Minimum Spanning Tree (MST), Knapsack problem, Vertex Coloring, Matching, Bin Packing, Fixed Charge Problem, to name a few. All the previous problems can be written as follows.

$$
\begin{aligned}
& & min\ (or\ max) \quad & \sum_{j=1}^{n} c_j x_j & \\
(IP) \qquad & & s.t \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_j & for\ i = 1, ..., m \\
& & & x_j \geq 0 & for\ j = 1, ..., n \\
& & & x_j\ integer & for\ j \in I
\end{aligned}
$$

where x_j, $j = 1, 2, ..., n$ are nonnegative variables, c_j, a_{ij} and b_i are coefficients. It is a standard form for all these problems. This problem is known as Integer Linear Programming (ILP). If $I = \{1, 2, ..., n\}$ then we call it a pure integer linear programming and otherwise Mixed Integer Programming (MIP) [52].

2.8 Solution approaches

Through the last fifty years, methods and approaches have been developed to solving linear integer problems. However, these problems are difficult to solve as they belong to the NP-hard optimization problems [53]. These approaches can be divided into two main classes; exact and approximate/heuristic. Note that the design and analysis of approximation algorithms involves a mathematical proof certifying the quality of the returned solutions in the worst case. This recognizes them from heuristics such as genetic algorithm, which finds reasonably good solutions on some inputs, but provides no clear indication at the outset on when it may succeed or fail.

Algorithms that belong to exact approaches are guaranteed to reach the optimal solution. When an optimum cannot be found efficiently in practice, then acceptable solutions to be found out in polynomial time are preferred. Approximate algorithms could be used to find a suitable initial solution or to tighten the feasible solution domain and to direct the search for the optimum, but they do not guarantee optimality of the final solution.

2.8.1 Exact approaches

There are many exact methods for solving IP problems, split into the following classes. Some of these classes are overlapping and the features of a number of methods have been used by some specifically successfully approaches to solve large problems. However, most of the IP problems solved are relatively small. The run time often increases highly with increasing the problem size [54].

Cutting planes methods In 1958, Gomory [55] described the original method of this type for general MIP problems. They solve IP problems by dropping the integrality requirements. The resulting problem known as the LP relaxation can be solved by ordinary LP solution methods. If the resulting solution of the LP is integer, then it is the optimum. If not then systematically more cutting planes (constraints) are added to the problem and seeking to get as close as possible to the convex hull of the solution set of the original problem. The solution of the new problem may or may not be an integer. This procedure is continued until the problem finds an integer solution or it is infeasible. These methods have not shown great success on large problems, although joined with Branch-and-Bound, and other methods they provide a more powerful approach [56].

Enumeration methods In general, the class of binary (0-1) ILP problems uses these methods. Theoretically, a problem that belongs to this class has only a finite number of possible solutions. Some of the solutions can be examined by a tree search and other solutions are systematically ruled out as non optimal or infeasible. These methods proved to be highly successful on particular problems. For more information see [56, 57].

Pseudo-Boolean methods Many algorithms have been designed to exploit the similarity between the binary ILP problems and Boolean algebra. This way is totaly different from any other to solve the IP problems. The Boolean algebra is used to express the constraints of the problem. These algorithms may perform well on some problems and less so on others. These techniques have been improved by Hammer and explained in [58–60]. Yet, no commercial packages eligible of accepting practical problems use these methods.

The Branch-and-Bound (B&B) methods B&B has been invented by Land and Doig [61]. It is the most successful approach to practical general IP problems. This algorithm is widely used in commercial packages that offer an MIP facility. It has a lot of flexibility in the way it explores the search space. B&B methods are partial enumerative methods. They first solve the LP relaxation of the problem using the simplex algorithm for instance. The problem is solved if the LP solution is an integer; otherwise a tree search is implemented. Hybridizing the Cutting Plane method with B&B results in Branch-and-Cut which has proved to be a good solution approach to IP problems. The B&B method explores the set of feasible integer solutions using a divide and conquer approach. It explores particular regions of this set. It uses bounds on the optimal value to avoid exploring the whole feasible set [62]. Consider the problem

$$\begin{aligned} min \quad & cu \\ s.t \quad & u \in X, \end{aligned} \tag{2.1}$$

where X is the set of feasible solutions. X could be the set of integer feasible solutions to the problem (2.1). This set is divided into a finite collection of subsets $X_1, ..., X_r$, where each one of the subproblems is solved separately as

$$\begin{aligned} min \quad & cu \\ s.t \quad & u \in X_j, \quad j = 1, ..., r. \end{aligned} \tag{2.2}$$

The optimal solutions to the subproblems are compared and the best is chosen. The subproblems may still be as hard as the original problem. The same method is used to some more subproblems, and so on. This part of the algorithm called "branching" produces a tree of subproblems.

We suppose also that there is an efficient algorithm, which can calculate a lower bound $B(X_j)$ to the optimal value of the subproblem for every X_j of concern, such that

$$B(X_j) \leq \min_{u \in X_j} c^T u \tag{2.3}$$

That is because finding a lower bound to a subproblem might be easier than exactly computing the optimal value. This bound can be obtained by finding the optimal value of

the LP relaxation. Notice that during the process of solving the subproblems, the algorithm may compute the value of a particular feasible solution, or find the optimal solution of a particular subproblem. This allows preserving an upper bound U on the optimal value which might be the value of the best feasible solution up to now. The essential point of the algorithm is that, a certain subproblem will not be considered if its lower bound $B(X_j)$ satisfies $B(X_j) \geq U$; this is because the optimum to the subproblem is no better than the best feasible solution known so far. Another main point in this algorithm is that at any point, the algorithm holds in memory the value U of the best feasible solution and a set of active subproblems thus far. A generic B&B algorithm works as follows [62].

Algorithm 1: A generic B&B algorithm

1 Choose an active subproblem X_j;
2 If X_j is infeasible, delete it; otherwise, calculate $B(X_j)$ for the corresponding X_j;
3 If $B(X_j) < U$, either break the corresponding X_j into more subproblems that are joined the set of active subproblems, or find an optimum to X_j subproblem;
4 If $B(X_j) \geq U$, delete the X_j.

Benders' Decomposition method Benders' Decomposition [63] is a specialised approach to mixed-integer linear programming. It has had wide applications in many domains such as routing and scheduling [64, 65], network design [66–69], and assignment [70] where the author considered a special linear assignment problem which is an integer programming problem. The mathematical model of their problem, called assignment problem of disjunctive tasks (APDT). He used a simple trick to transform APDT into a mixed binary linear program then apply the Benders' Decomposition. This method implements the divide and conquer strategy. In the framework of the Benders' Decomposition two different problems are solved: the master problem which is almost a pure IP, and the subproblem which is the dual of the LP part, [71]. The variables of the original problem are split into two subsets such that the master problem is solved over the first set of variables, and the values for the second set of variables are found in the dual subproblem for a given master problem solution. If the dual problem finds that the fixed master problem decisions are in fact infeasible, then the so-called Benders cuts are generated and added to the master problem. This problem is re-solved until no cuts can be added. Benders decomposition adds new

constraints as it proceeds towards a solution.

2.8.2 Meta-heuristics

Combinatorial optimization problems can be solved by heuristic techniques. There is a number of heuristic algorithms for IP such as Genetic Algorithms (GAs), Tabu Search (TS), and Simulated Annealing (SA). Algorithms that are inspired by nature and heuristic procedures have some notable drawbacks; unlike classical approaches, they are supported by rather limited theories. They are stochastic, and dependent on often many parameters the setting of which is usually arbitrary. This means that results vary over many runs and repetition of results by different researchers may be hard to reach [54].

These techniques are meant to quickly find good solutions to IP problems, but not necessarily the optimum ones. They are important for many reasons. When exact algorithms are not suitable for very large optimization problems heuristics may be the only usable alternatives. Exact methods are also less flexible than heuristics. A good heuristic technique should have three properties; The solution should be close to the optimum, the probability of reaching a bad solution should be low, and a solution can be obtained in reasonable time. Note that this thesis concerns NP-hard problems. So it is pertinent that we present here the most well known heuristic approaches for solving them.

Genetic algorithms (GAs) Since the 1960's there has been a number of algorithms for intractable optimization problems. Such techniques are evolutionary in nature. They comprise evolution strategies, provided by Rechenberg [72] and Schwefel [73], Genetic Algorithms (GAs), introduced by Holland [74], Genetic Programming (GP), developed by Koza [75], and Evolutionary Programming (EP), introduced by Fogel el al. [76]. In computer science and operations research, GA is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of Evolutionary Algorithms (EAs). Genetic algorithms generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. GA is a well established meta-heuristic which has been shown to be very effective on combinatorial optimisation problems.

Simulated Annealing (SA) The simulation of annealing is a random search technique for global optimization due to Kirkpatrick et al. [77] and, independently, Černý [78]. SA is a randomized local search algorithm. It has a large effect on the field of heuristic search for its efficiency and simplicity. SA emulates the annealing process in material processing which needs heating and then slowly cooling to get a strong crystalline structure. The rate of cooling metals affects the strength of structure. Defects are obtained if the initial temperature is not high enough or by applying fast cooling, while slow cooling increases the size of the metals crystals and reduces imperfections. SA emulates the changes in a system's energy subject to a cooling process until it converges to a steady frozen state.

SA avoids being trapped in local minima, unlike the deterministic search methods and gradient-based methods. It has been proved that, if enough randomness is employed in combination with very slow cooling then SA converges to the global optimum. SA interprets slow cooling as a slow reduction in the accepting of worse solutions probability as the solution space is explored. Accepting worse solutions is an essential property of meta-heuristics because it permits extensively searching the space for the optimum.

Tabu Search (TS) TS is due to Glover [79]. It is a meta-heuristic that solves approximately optimization problems. The method uses memory structures to describe the visited solutions. The solution that has been already visited is marked as tabu, so the algorithm will not select it again. Shortly, search spaces are explored by moving at each iteration to the best non-tabu neighbor of the immediate solution. In order to avoid cycling, tabu search discards the neighbors that have been already visited [80].

Particle Swarm Optimization (PSO) PSO was proposed by Kennedy, Eberhart, and Shi [81–83]. It is inspired by the movement of organisms such as a fish school or bird flock. PSO is a meta-heuristic that can search large spaces. The problem is solved by using a population of solutions called particles which spread out in the search space randomly and search to find the optimum or near optimum via generating new populations. The PSO solves a problem by repeatedly trying to improve a solution with respect to its fitness function. The particles move around in the search space relying on a mathematical formula over the particles position and velocity. The movement of each particle is influenced by its

local best position and is also directed toward the best positions in the search space, which are replaced as better positions are found so far by other particles. This moves the swarm toward the best solutions.

The Plant Propagation Algorithm (PPA) The Plant Propagation Algorithm (PPA) designed by Salhi and Fraga [84], emulates the way plants propagate. It is a neighbourhood search algorithm that can be seen as a multi-path following algorithm unlike, for instance, Simulated Annealing which is a single path following algorithm. PPA begins with a population of plants uniformly randomly placed in the search space. The population is ranked according to fitness values of the plants positions. Their propagation intensity is found by their scores on the objective function. If they score low, they are treated as being in a poor spot of the search space; if on the other hand they score high they are treated as being in a good spot. Those in bad spots generate few longer runners, while those in good spots propagate by generating many short runners. Note that the exploitation and exploration are the main components in any successful global optimization. The long runners implements the idea of running away from bad areas, they permit distant neighbourhoods to be explored, while many short runners are sent by the plants to exploit the good spots.

2.9 Summary

In this chapter we have defined the C1P property and the C1S problem as well as the most prominent related application areas. A review of the background literature is given as well as a definition of integer programming and the most popular approaches to solve it. The next chapter will develop an evolutionary type heuristic approach to the C1S problem.

Chapter 3

A heuristic approach to the C1S problem

3.1 Introduction

The purpose of this work is to explore the possibility of solving mixed integer programming problems, as those pure integer programs with $(0,1)-$matrices. Integer programs problems whose $(0,1)-$matrices have the C1P property have totally unimodular coefficient matrices [65]. This means potential computational gain can be made. Our approach to find the maximum submatrix having the C1P is evolutionary. More explicitly, we put forward a novel implementation of GA. The novelty is in the fitness functions used.

3.2 Current solution approaches

Here, we consider the solution of the C1S problem through the solution of the seriation problem and CBM using current approaches.

3.2.1 Solution of the seriation problem

The data structure that is used for analyzing this problem is the incidence matrix. A $(0,\ 1)-$matrix A with m rows (grave sites) and n columns (object types), the a_{ij} element of A has value 1 if grave i includes any object of type j and zero otherwise. If the rows (graves) were in the suitable time sequential order, each column (object type) would suggest a time period during which that object was popular. Suppose that each object represented by the

time interval from the object appearance to when it becomes unpopular then disappear. The graves' chronological ordering is attained by finding rows permutation of A that changes it into Petrie form. This minimizes the total temporal range of an object type summed over all the types of objects (i.e., columns). This range for an object is the span from the first to the last appearance of a '1' in the column of A corresponding to that object type.

In [22] Kendall, provided a solution. Let us illustrate it. Given (0, 1)−matrix A of size 6×6, it can be permuted into Petrie matrix in two ways. Let $Q = AA'$ be $m \times n$, s.t $m = n$ nonnegative symmetric Similarity Matrix for graves, such that Q has ij^{th} element equal to the number of varieties that are in common to graves i and j.

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \qquad Q = \begin{pmatrix} 3 & 0 & 2 & 1 & 3 & 2 \\ 0 & 2 & 2 & 0 & 1 & 0 \\ 2 & 2 & 5 & 0 & 4 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 3 & 1 & 4 & 1 & 5 & 2 \\ 2 & 0 & 1 & 1 & 2 & 2 \end{pmatrix}$$

Kendall proved that a permutation P that can change the grave matrix Q into Robinson form (i. e., PQP') can also transform A into a Petrie form (i. e., PA). He used a multi-dimensional scaling (MDS) technique to find a permutation P from the information that is contained in Q. Reordering A and Q using P produces a Petrie matrix and a Robinson matrix[1] respectively, [30]:

[1]Robinson matrix is a matrix with entries which do not increase as one progresses along a row beyond the main diagonal and do not decrease as one continues to progress along that row towards the main diagonal.

$$PA = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad PQP' = \begin{pmatrix} 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 5 & 4 & 2 & 1 & 0 \\ 1 & 4 & 5 & 3 & 2 & 1 \\ 0 & 2 & 3 & 3 & 2 & 1 \\ 0 & 1 & 2 & 2 & 2 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The resulting matrix PA has the C1P where each column shows the right order of the object type from its appearance to when it disappear. Finding a matrix P is impractical and so a simpler heuristic procedure was developed using a MDS technique that due to Shepard and J.B. Kruskal [85].

3.2.2 Polynomial-time local-improvement algorithm for CBM

The CBM-decision problem is NP-complete even if restricted to $(0,1)-$matrices having two ones in each row [16, 49]. However, Haddadi [50] provided a polynomial-time heuristic that provides a permutation such that the number of consecutive blocks and the optimum do not differ by more than 50%. Haddadi and others [29] provided a polynomial-time local-improvement heuristic for CBM. They proposed two $O(n^2)-$size local search neighborhoods, where the number of blocks of a neighbor is found in $O(m)$ operations.

- *Improvement by two columns interchange*: Let A_ρ be a matrix associated with a permutation ρ such that the total number of blocks is μ. Suggest the $O(n^2)-$sized neighborhood $\mathcal{N}(\rho)$ to be the set of all permutations that result from ρ by interchanging two columns. $\mathcal{N}(\rho)$ is searched for a permutation gives a small number of blocks, if no such permutation exists, the procedure ends. If an improvement δ is reached by interchanging two columns $\rho(i)$ and $\rho(j)$, $i \neq j$, (the new permutation is called ρ'), we update $\mu \leftarrow \mu - \delta$, $\rho'(i) \leftarrow \rho(j)$, $\rho'(j) \leftarrow \rho(i)$, and repeat the process with ρ'.

- *Improvement by column-insertion*: Let $\mathcal{N}'(\rho)$ be the permutation set that results from ρ by inserting (Haddadi called it shifting) one column. This means that the column leaves its location and is inserted between two other columns. $\mathcal{N}'(\rho)$ is also searched

for a permutation producing a smaller number of blocks and the search stops when no such permutation exists. Consider that an improvement δ is reached by inserting a column $\rho(i)$, and let the new permutation be $\rho'(i)$. If all the necessary updates are performed, the procedure is repeated for $\rho'(i)$.

Algorithm 2: Interchange procedure

Input Positive integers m, n, total number of blocks μ, binary $m \times n-$matrix A, permutation ρ;
for $i = 1, ..., n-2$
for $j = i+2, ..., n$
  Interchange two
  non-adjacent columns $\rho(i)$ and $\rho(j)$;
  Compute δ;
  if $\delta > 0$
    $\mu \leftarrow \mu - \delta$;
    Swap $\rho(i)$ and $\rho(j)$;
  end if
end for
end for
Return Permuted matrix.

Algorithm 3: Insertion procedure

Input Positive integers m, n, total number of blocks μ, binary $m \times n-$matrix A, permutation ρ;
for $i = 1, ..., n$
for $j = 1, ..., n$ $(j \neq i-1, j \neq i, j \neq i+1)$
  Insert column $\rho(i)$;
  between columns $\rho(j-1)$ and $\rho(j)$;
  Compute δ;
  if $\delta > 0$
    $\mu \leftarrow \mu - \delta$;
    if $i > j$ move $\rho(i)$ to position j in ρ
    else move $\rho(i)$ to position $j-1$;
    end if
  end if
end for
end for
Return Permuted matrix.

3.3 New solution approaches

Here, we consider new approaches one direct, the other evolutionary.

3.3.1 An alternative column insertion procedure

We want a permutation of the columns of a matrix that improves the number of C1S columns. Here, we suggest a local improvement based algorithm which is polynomial in time. Suppose a $m \times n-$matrix A with two submatrices where m and n are not necessarily equal, $m \times p-$submatrix A_1 (we call it the remainder submatrix) and $m \times (n-p)-$submatrix A_2 which has the C1P. We suggest inserting the remainder columns of A_1 one by one between the columns of A_2. Consider that investigating the matrix A_ρ associated with a permutation ρ such that the total number of C1S columns is β. Suppose the neighborhood $\mathcal{N}''(\rho)$ to

be the set of all permutations that result from ρ by moving one column, which means the column leaves its position in A_1 and is inserted between two columns in A_2. $\mathcal{N}''(\rho)$ is explored searching for a permutation to increase the C1S columns. This procedure ends if there is no such permutation.

The process starts by choosing the first left column from A_1 say $\rho(j)$ for $1 \leq j \leq p$. This column is checked between two adjacent columns of A_2 say $\rho(i)$, $\rho(i+1)$ for $i = 1, ..., n-p$. If the column $\rho(j)$ improves the C1S columns in A_2, we update A_2 and choose the second column $\rho(j+1)$ to insert in the new matrix. If $\rho(j)$ fails to prove an improvement then it is returned back to the right hand side of A_1 and $\rho(j+1)$ is chosen to insert.

Suppose A_2 is a submatrix with only two consecutive columns. The (0, 1) in this submatrix has four arrangements (0 0, 0 1, 1 0, 1 1). Inserting column $\rho(j)$ in A_2 produces eight cases that are suggested by [22,29], (0 1 0, 0 0 0, 1 0 1, 1 1 1, 1 1 0, 0 1 1, 1 0 0, 0 0 1). A_2 is destroyed in one case, when the sequence of columns $\rho(i)\rho(j)\rho(i+1)$ is 1 0 1, otherwise the matrix still has the C1P.

If the number of columns of A_2 is greater than 2, then another destructive case appears. That is when the sequence of columns $\rho(i)\rho(j)\rho(i+1)$ is 0 1 0 and there is another block of ones in the same row (say 1 0 1 0), this row will lose the C1P property. Table 3.1 shows checking the cases. The destructive cases are referred as false and other cases that do not destroy A_2 as true. Column $\rho(j)$ fits between $\rho(i)$ and $\rho(i+1)$ if the result of checking all the rows is true.

Table 3.1: *Cases of checking a column inserted between two columns*

$\rho(i)$	$\rho(j)$	$\rho(i+1)$		*Result*
1	0	1		*false*
0	1	0	*and there is another block of ones in this row*	*false*
0	0	0		*true*
1	1	1		*true*
1	1	0		*true*
0	1	1		*true*
1	0	0		*true*
0	0	1		*true*

If an improvement γ is achieved by inserting a column $\rho(j)$ in A_2, the new permutation is called ρ'. We update the matrix A_ρ to $A_{\rho'}$ and repeat the process with ρ'. The number of

blocks which results from inserting an arbitrary column between two other columns can be computed in $O(m)$ operations, see [29]. The size of searching $\mathcal{N}''(\rho)$ costs $O(\sigma)$ time. That is because we have one possibility to insert $\rho(j)$ between $\rho(i)$ and $\rho(i+1)$. If it fits, then $\rho(j+1)$ needs two possibilities, and so on. We can say that the size of possible columns for insertion is $\leq (n-2)(n-1)/2$.

Lemma 3.3.1 The complexity of the column insertion procedure is $O(m\sigma p)$ where σ is the size of the neighborhood $\mathcal{N}''(\rho)$ to explore.

Proof. Searching the neighborhood $\mathcal{N}''(\rho)$ costs $O(\sigma)$ time where

$$\sigma = \sum_{N=n-p}^{n-1} N - 1, \tag{3.1}$$

The integer number N start with the $n-p$ columns of A_2. For every neighbor, the calculations of checking the cases in Table 3.1 for all the rows cost $O(m)$. As long as the number of C1S columns β cannot be greater than n, and no less than $n-p$ columns, the finiteness of the inserting procedure is assured and the number of improvements is at most $n-(n-p)=p$ in the worst case. ■

Improving the number of blocks The procedure of column insertion that is used to search the neighborhood $\mathcal{N}''(\rho)$ for a permutation to maximize the C1S submatrix also minimizes the number of blocks. Mentioning analogous arguments as in Lemma 3.3.1, the complexity of inserting the columns to find the minimum number of blocks is $O(m\sigma(g-m))$, where g is the number of 1's in A. Also, since the number of blocks μ cannot be greater than g, and no less than m, the finiteness of the insertion procedure is assured and the number of improvements is at most $g-m$ in the worst case.

This algorithm can be implemented directly to the matrix A by extracting the C1S matrix then fill the reminder columns.

Algorithm 4: Column insertion procedure

Input Positive integers m,n, total number of columns β, binary $m \times p-$submatrix A_1, $m \times (n-p)-$submatrix A_2 with the C1P, and permutation ρ;
for $j = 1, ..., p$
for $i = 1, ..., (n-p)-1$
  Insert column $\rho(j)$ between $\rho(i)$ and $\rho(i+1)$;
  Check $\rho(i)\rho(j)\rho(i+1)$;
  if the result of the check is *true* for all m
    $\beta \leftarrow \beta + \gamma$;
    Move $\rho(j)$ between $\rho(i)$ and $\rho(i+1)$ and update A_2;
  end if
end for
end for
Return Permuted matrix.

3.3.2 A GA-based solution approach

As said earlier GA has been used in related context. Here, we develop an implementation of GA to deal with C1S. This implementation represents a solution as the permutation of a (0, 1)−matrix.

3.3.2.1 Solution representation

Formulating an appropriate representation scheme is not trivial. It depends on the problem of interest and the size of the search space. It is regarded as a first step in implementing a genetic algorithm for a specific problem. The problem of maintaining the feasibility of the initial population throughout the crossover and mutation processes is faced by many researchers. It can be resolved by many ways such as using integer representation. A possibility is to have the string size equal to the number of columns of the matrix. The position of each gene corresponds to a column in this matrix, but the computing of the fitness may become unclear because different forms of the matrix may have the same fitness value.

The basic idea in our algorithm is to keep permuting the columns of the given (0, 1)−matrix to get as many consecutive ones as possible in every row. An integer is used here to represent a solution or chromosome of size n, an array of the column indices of the matrix. The

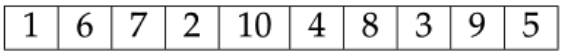

Figure 3.1: *Chromosome representation*

position of each gene in the chromosome corresponds to a column in this matrix. Figure 3.1 illustrates this representation.

3.3.2.2 Genetic operators

After randomly generating the initial population, we breed successive generations of offspring. This can be achieved by performing genetic operators which are crossover, mutation, and reproduction.

Crossover operator

Producing a new individual needs using a suitable mating; we use the 'Order Crossover' of Gen, and Cheng [86] to breed a new child. This operation begins by picking a substring (subchromosome) of random size from one of the two parents randomly, then copying it into its corresponding location in the child. The genes in the second parent that appeared in the substring are deleted to prevent repeating the gene. In the end place the genes into the unfixed location of the child from left to right relying on the order of the sequence. See Figure 3.2, we create Child1 by choosing a substring from Parent1 (1, 10, 8) then delete this substring from Parent2 and add the rest (3, 6, 5, 2, 4, 7, 9) to Child1. The same for Child2, we choose (3, 6 ,5) from Parent2 then delete it from Parent1 and add the rest (1, 10, 8, 7, 2, 4, 9) to Child2.

Mutation operator

Mutation operator is used to allow the genetic algorithm to search in the global solution space and prevent being trapped in the local optima via changing one or more genes. Here, two genes from an individual are randomly chosen and then swapped. Figure 3.3 illustrates the mutation operator where 2 and 8 are swapped.

Parent1	1	10	8	7	5	3	6	2	4	9
Child1	**1**	**10**	**8**	3	6	5	2	4	7	9
Parent2	3	6	5	2	4	8	7	9	1	10

Parent1	1	10	8	7	5	3	6	2	4	9
Child2	**3**	**6**	**5**	1	10	8	7	2	4	9
Parent2	3	6	5	2	4	8	7	9	1	10

Figure 3.2: *Child1 and Child2 obtained with crossover*

Parent:	1	10	**8**	7	5	3	6	**2**	4	9
Child:	1	10	**2**	7	5	3	6	**8**	4	9

Figure 3.3: *Individual obtained with the mutation operator*

Reproduction

Without any modification some fit chromosomes are copied via this operator into the next generation.

3.3.2.3 Fitness Function

The matrix does not include enough information to decide whether it has the C1P property or not. However, we will use the entries of the matrix to build the fitness function. Many fitness functions are tested and the most reliable one with respect to the number of C1S columns and run time is selected.

Fitness Function FF1

This function computes the number of 1's in each row using a simple formula. Summing the number of 1's for all the rows will give the fitness value of the matrix. Suppose we have the following rows with four elements which are arranged in two different ways

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The first row has two blocks of 1's, k_1, and k_2 where k_1 contains one element and k_2 contains three 1's. We deal with every block as a sequence of 1's. The associated series is defined as the ordered formal sum $\sum_{j=1}^{l} j$ where l is the length of the block. Counting the elements for the two blocks will be as

$$fitness\ row = \sum_{i=1}^{k} \sum_{j=1}^{l_i} j,$$

$$where \qquad k = number\ of\ blocks,$$

$$and \qquad l_i = number\ of\ 1's\ in\ the\ i^{th}\ block.$$

Applying the formula for the two rows gives fitness values 7 and 5 consequently, where $\sum_{j=1}^{1} j + \sum_{j=1}^{3} j = 1 + 6 = 7$ and $\sum_{j=1}^{1} j + \sum_{j=1}^{1} j + \sum_{j=1}^{2} j = 1 + 1 + 3 = 5$. Applying this formula for a matrix gives different fitness values for every row. The fitness value for the whole matrix is found by accumulating the fitness values of all the rows, this is showing in the next formula

$$fitness\ matrix = \sum_{s=1}^{m} fitness\ row(s),$$

$$where \qquad m = number\ of\ rows.$$

The larger the fitness, the better matrix with respect to consecutive ones blocks. With different permutations of columns in each generation and computing the fitness function of the matrix, the consecutive solutions are improved every time by pushing the ones together in less blocks. In the final generation it is expected that a matrix with best consecutive ones in every row is obtained. The result of implementing this algorithm can produce some matrices that satisfy the C1P for the whole matrix, while others have submatrices with the property. Consider the following 6×6 binary matrix A with two ones in each row. Applying the algorithm provides matrix A' which has the C1P property for the whole columns.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \qquad A' = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

By bartering the ones in some rows in matrix A and applying the algorithm again we can get different result.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \qquad A' = \left(\begin{array}{ccccc|c} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right)$$

The minimum number of columns to delete is one and the maximum number of columns that satisfy the property is five.

Fitness Function FF2

This function relies on counting the ones in the different regions of the matrix. That is by finding the label of the connected components of the matrix, where each maximal connected region is assigned as a unique label, see [87, 88]. The matrix below has three regions of ones. The first connected component in the matrix have five elements labeled as 1, the second connected components have three elements with label 2, and the last has one element with label 3 as shown in the region matrix. We also deal with elements of every region as a sequence of 1's, like the fitness FF1. The fitness values of the regions are 15, 6, and 1 consequently which means the fitness function for the matrix is 22 where

$\sum_{j=1}^{5} j + \sum_{j=1}^{3} j + \sum_{j=1}^{1} j = (1+2+3+4+5) + (1+2+3) + 1 = 15 + 6 + 1 = 22.$

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}, \qquad regions = \begin{pmatrix} 0 & 1 & 0 & 2 & 2 \\ 0 & 1 & 1 & 0 & 2 \\ 1 & 1 & 0 & 3 & 0 \end{pmatrix}$$

This function pushes the ones in different separated region, which may produce a variety of fitness values more than the FF1 function. It accumulates ones not only in the rows, but also in the columns. This helps accumulate the ones in large blocks. Function FF1 may give similar fitness values for many rows, and consequently similar fitness values for different matrices. This causes that the worse matrix with less columns having the C1P property may be selected for the next generation. With FF2 there is more chance to distinguish between matrices. It may give better results than the FF1 function. Although, it gives good results, unfortunately it cost more time than FF1 because of the code that is used to find the label of the regions. So it is not the desired function for the proposed algorithm.

Fitness Function FF3

This function counts the zeros in every row then for the whole matrix instead of the ones. The same formula that is used for finding the maximal sum of 1's in FF1 is used here. Maximizing the fitness value leads to accumulating the zeros in blocks which results in minimizing the number of gaps between the ones. This fitness almost gives the same result but with more time because the matrix is sparse so the calculations cost more time than FF1. As we seek better result with less time, this fitness function is also not suitable for our implementation.

Fitness Function FF4

The fitness function that Gargano and Lurie used for solving the Petrie Seriation Problem [30] is used to find the consecutive ones in the columns by permutation of the rows. Although, it gives a good result for a small matrix, they did not mention whether it is

suitable for large matrices with different densities. So we decided to test it. Their fitness is evaluated with the so called Petrie Range Index (PRI) of the permuted $(0,1)-$matrix. We formulate the fitness for our matrix by choosing the first column c_f and the last column c_l where in each row there is at least one element. The PRI for that row is $c_l - c_f + 1$. The fitness for the matrix is the sum of these values over all the rows. The best solution is reached through minimizing the PRI. To calculate this fitness the matrix should have different number of ones in each column to get different fitness values. It is noticeable that if the matrix has the same number of ones in each column the fitness value for the matrix will be the same for any permutation. As a result, good matrices may not cross to the next generation, which causes getting not so good results. It is more suitable for matrices with high density. Applying this function provides less columns but with shorter time comparable to the previous functions. It is formulated as follow

$$\begin{aligned} &Petrie\,range\,index\,fitness = \sum_{i=1}^{m} c_{l_i} - c_{f_i} + 1, \\ &where \qquad\qquad m = number\,of\,rows, \\ &\qquad\qquad\quad c_{l_i} = last\ column, \\ &\qquad\qquad\quad c_{f_i} = first\ column. \end{aligned}$$

Here again, FF4 is not suitable for our implementation of GA.

Fitness Function FF5

This function is meant to solve the CBM problem which is equivalent to C1S. The target is permutating the matrix columns to minimize the number of blocks of consecutive ones. Building the function relies on counting the blocks of ones in each row then for the whole matrix. The same two rows that are used in FF1 function are used here.

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The first row has two blocks b_1, b_2 of ones and the second one has three blocks b_1, b_2, b_3.

The formula for a matrix is

$$CBM\ fitness = \sum_{i=1}^{m} \sum_{j=1}^{k_i} b_j,$$

$$where \qquad m = number\ of\ rows\ in\ the\ matrix,$$

$$k_i = number\ of\ blocks\ in\ the\ i^{th}\ row.$$

where b_j takes value 1. Applying the formula for the two rows gives values 2 and 3 consequently. The fitness value for the whole matrix can be obtained by summing the fitness values of all the rows. This fitness also pushes the 1's together to make blocks. The minimum fitness value, the better matrix with a lesser number of blocks, may produce larger C1S submatrix. Therefore, the problem of C1S is solved via finding the minimum consecutive blocks. An example of the application of this fitness function, the following matrix A, shows that the matrix with 12 blocks turns into a matrix with 6 blocks after applying the CBM fitness function.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \quad A' = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

3.3.2.4 Stopping criterion

A common criterion used to stop the proposed genetic algorithm, is the maximum number of generations.

3.3.2.5 Selection procedure

It usually implements a roulette wheel which is biased towards fit individuals. This means that good individuals are likely to be parents.

The proposed GA algorithm is shown in pseudocode as Algorithm 5.

Algorithm 5: Algorithm C1P

```
1  Input Positive integers m, n, binary m × n−matrix A;
2  Input η the rate of crossover and ω the rate of mutation;
3  Generate a random population of permutations of columns of matrix A;
4  Evaluate the fitness of the permutations according to a function;
5  Rank individuals according to their fitness;
6  Select parents from the population according to some selection procedure;
7  Generate a new population by applying the following operators: crossover,
     mutation, and reproduction;
8  Compute the fitness of the individuals of the new population;
9  Until (The sopping criteria are satisfied) Repeat from 5;
10 Return Permuted matrix.
```

3.4 Minimum degree reordering algorithm

Before we embark into using the minimum degree reordering algorithm, let us first motivate our decision to use it. The idea is that by reordering the given (0, 1)−matrix A with this algorithm produces a structure with large blocks of contiguous zeros which do not fill in during the factorization. Consequently, the nonzero elements will set in large blocks of contiguous 1's. This brings together the columns which improve the C1P property of the submatrix of A. This has been observed in practice. We therefore carry out all experiments on preprocessed matrices using this algorithm. An overview of the algorithm is following.

For an $n \times n$ symmetric positive definite system of equations

$$Ax = b, \tag{3.2}$$

where A is sparse and n is large, factorization of A by Cholesky method causes fill; that is some zero entries in matrix A turn into nonzero ones in the factors. In fact, nonzero numbers in a row say e above the diagonal guide to new nonzero numbers in the rows that have nonzero in column e. This results from the operations that are associated with the Gaussian elimination. An important point is that the fill relies on the order of the elements locations in the matrix. So detecting the ordering of the rows and columns of the matrix in

the sense of minimizing the fill makes the sparse Cholesky fast. For any permutation matrix Q, QAQ^T is also symmetric and positive definite, so instead of the system of Equation (3.2) the following system can be solved

$$(QAQ^T)(Qx) = Qb. \tag{3.3}$$

The ordering problem can be defined as finding a permutation matrix Q such that the factorization of a given square matrix preserves sparsity. Computationally, the problem of providing a best ordering for a matrix that produces the least fill is intractable, i.e it is an NP-complete problem [89, 90]. Heuristic algorithms are imposed.

The minimum degree algorithm is well studied over the past forty years. It is related to a previous heuristic introduced by Markowitz in 1957 [91] for reordering linear equations that appear in linear programming applications. It is based on the elimination graph model and employed on unsymmetric matrices. The procedure decreases the fill in the solution of unsymmetric system by Gaussian elimination. This algorithm starts with a given matrix, performing a permutation of a row and column to minimize the number of off-diagonal nonzeros in the pivot row and pivot column at each step of the Gaussian elimination. This minimizes the fill that appears via minimizing the amount of the arithmetic that should be carried out at each step. Generally, such a local strategy cannot produce a global minimum for the fill or the arithmetic requirements. However, it is very effective in decreasing both of them [89].

Tinny and Walker [92] gave the first implementation of the minimum degree algorithm. Later many improvements were introduced to this algorithm. George and Lui [89] provided the evolution of the minimum degree algorithm. The algorithm of Tinny relied on the original one of Markowitz [93–95]. The algorithm takes as input the adjacency matrix of the graph representation of the given matrix. For a considered symmetric graph, the algorithm may be written as follows.

The algorithm provides a graph with a new labelling. When selecting the node with minimum degree, many nodes may have this degree. In this case a random selection is made corresponding to tie-breaking strategies which may produce many versions of the

Algorithm 6: Minimum degree ordering algorithm (MDOA)

1 **Input** Adjacency matrix A, symmetric graph G represents A;
2 **while** $G \neq \emptyset$ **do**
3 Select and order node v of minimum degree in G;
4 $G = G_v$;
5 **end while**
6 **Return** Permutation.

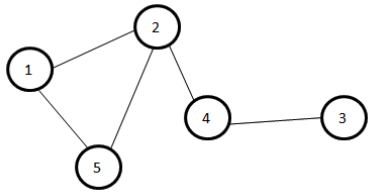

Figure 3.4: *Representation of graph G_A*

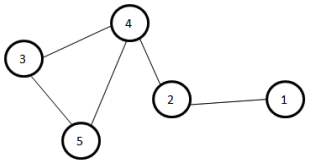

Figure 3.5: *Representation of graph $G_{A'}$*

minimum degree orders [94]. Transforming a graph G_A into graph say $G_{A'}$ by elimination involves deleting and adding edges. The number of edges may either decrease or increase relying on how the adjacent nodes are connected to the selected one. Eliminating a node v with the least degree with its incident edges from G_A then adding them to make the nodes adjacent to this node into a clique, produces the new graph $G_{A'}$ [89]. The graph of matrix A and the new reordered graph are shown in Figures 3.4-3.5. It is clear that A has eight blocks of ones, while A' has only six.

$$A = \begin{pmatrix} (1) & * & & & * \\ * & (2) & & * & * \\ & & (3) & * & \\ & * & * & (4) & \\ * & * & & & (5) \end{pmatrix}, \qquad A' = \begin{pmatrix} (3) & * & & & \\ * & (4) & & * & \\ & & (1) & * & * \\ & * & * & (2) & * \\ & & * & * & (5) \end{pmatrix}$$

However, There are three problems in the performing of the minimum degree algorithm. Since it is needed to add and delete edges from the graph, it is difficult to predict the maximum storage requirements for the algorithm. The second problem is calculating the degree of the node which is the most computationally expensive part of the algorithm. Finally, obtaining a good column ordering for unsymmetric matrices [96].

3.4.1 COLAMD preordering algorithm

In 2000, Davis *el al* [97] provided a new COLAMD algorithm which is a local greedy heuristic. They produced COLAMD preordering algorithm that relies on the same strategy of Matlab's COLMMD preordering algorithm, but with a better heuristic orderings. Their algorithm is faster and produces better ordering, with less nonzeros in the factors of the matrix. They used COLAMD for square unsymmetric matrices to provide a column preordering for sparse partial pivoting. We use the COLAMD to preprocess the matrices in our experiments.

3.5 Computational experience

3.5.1 Implementation of GA

To compare the quality of the fitness functions, we implement them separately in Algorithm 5. They are applied to the same matrices with a fixed number of generations and initial population. We run the GA on 10 different matrices for each size. The size of the population and the generation are fixed to 40 for the first two matrices and 100 for the rest. Table 3.2 illustrates the rate in the number of columns (Nbcols) with C1P rounded to the nearest integer number. We refer to the matrices size as (Mat.) and to the density as (Dens.). The table shows that FF1 and FF5 produce larger rate of columns with less time. Although, both of them almost have the same results, we prefer the latter as it uses less time.

Table 3.2: *Comparing the fitness functions with respect to the number of columns with C1P and time*

Mat.	*Dens.*(%)	*FF*1		*FF*2		*FF*3		*FF*4		*FF*5	
		Nbcols	*Time*(s)	*Nbcols*	*Time*(s)	*Nbcols*	*Time*(s)	*Nbcols*	*Time*(s)	*Nbcols*	*Time*(s)
24	6	21	1.11	21	1.97	20	2.35	14	0.49	23	0.71
50	4	17	1.45	18	5.89	17	8.26	13	0.31	20	1.67
100	2	28	12.58	30	14.05	27	33.37	22	1.22	27	9.77
200	2	20	16.66	20	95.94	20	133.41	15	3.13	21	12.31
500	2	14	47.59	14	2760.48	13	641.19	11	17.52	14	31.31
1000	2	10	123.09	10	132611.32	10	5487.19	10	18.83	10	62.68

This algorithm is implemented for nonsymmetric matrices of the set covering problem with different densities. These matrices are generated in the context of a practical problem

namely that of pollination using beehives. It is a problem of beehive location for optimum pollination [98]. We use these matrices to serve as a test bed to analyze the GA. The results shown in Table 3.3 are obtained by applying Algorithm 5 to five different matrices of each size. We run the GA 10 times on every matrix, each time with a random initial population. In fact, the results of the algorithm rely on the arrangement of the $(0, 1)-$matrix, therefore, it may produce better results when it is repeated with different initial populations. The run time depends slightly on the density of the matrix and the number of rows, but more on the number of columns. We set the number of generations and population size to 40 for the first two matrices and 100 for the rest of them. The heuristic gives the results of the number of blocks and columns. The first three columns have the initial information of the matrices. The rest of the table shows the number of generations (Ge), the rate of final number of blocks with the number of improved blocks, and the rate of columns (Nbcols) with C1P. For small matrices the number of columns with C1P is good, but for matrices with size ≥ 100 the results are not so good. From our experience the results can be improved by the following:

Table 3.3: *Results obtained with GA alone*

Mat.	*Dens.*(%)	*Initial blocks*	*Ge.*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time*(*s*)
24	16	84	40	47	37	12	0.96
50	4	103	40	70	33	20	1.93
100	2	207	100	172	35	30	9.68
100	4	402	100	329	73	14	9.97
100	10	952	100	773	179	8	9.96
200	2	771	100	709	62	21	12.07
200	4	1508	100	1355	153	11	11.85
200	10	3513	100	3151	362	9	12.13
500	2	5061	100	4671	390	14	31.21
500	4	8875	100	8506	369	12	30.62
500	10	21871	100	20513	1358	8	33.79
1000	2	19220	100	18737	483	12	94.21

1. Increasing the number of generations and the initial population size.

2. For large matrices, it is better to have an initial population which covers more than

half the columns of the matrix to get about the same number of columns having the C1P. This reduces computing time. This can be done by reducing the chromosome that presents the size of the matrix to the half.

3. Dividing the matrix into submatrices and applying Algorithm 5 separately, then applying it for the whole matrix put together. This however requires more time.

4. The initial population can be seeded by applying Algorithm 5 many times then make the final population from the best of each run and use it as the initial population. This also takes time.

5. Column insertion can be applied to the output of Algorithm 5.

6. Preprocessing by algorithm COLAMD can be applied to the matrix before applying Algorithm 5.

3.5.2 Implementing column insertion after GA

The GA algorithm with fitness FF5 is not suitable for matrices with higher size and density. As long as using large populations and generations is not preferred, we apply this algorithm to the result from the GA algorithm. After applying Algorithm 5 to matrix A, we separate the result into two submatrices, one being C1S (columns$\geq$ 2), then apply the column insertion algorithm. From the experiments, it is found that the column insertion algorithm gives better results on larger C1S matrices. Also, the number of generations and the population size of the Algorithm 5 can be reduced. This method can improve the results many times for some matrices. The average of the results of Table 3.4 are provided by combining Algorithms 4 and 5. We run the two algorithms on the same matrices of Table 3.3, but here for population size equal to 40. The results are promising: there are less blocks, and more columns having the C1P obtained in less CPU time.

3.5.3 Applying GA after preprocessing

We implement the C1P algorithm with COLAMD and without it for few matrices with fixed initial population to test the effectiveness of the code. Table 3.5 provides some results

Table 3.4: *Results of Algorithm 5 and columns insertion*

Ge.	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*
20	33	51	20	0.66
20	51	52	43	1.41
40	139	68	83	3.98
40	313	89	45	4.57
40	741	210	18	4.34
40	607	164	98	5.43
40	1277	231	50	5.73
40	3103	410	21	5.58
40	4499	562	83	16.42
40	7908	967	64	16.29
40	19027	2843	42	16.77
40	1833	892	75	66.07

for several matrices that are enhanced using COLAMD. The size of the population is set to 100 and the number of generations to 10. The second two columns show the results of the matrix using COLAMD with Algorithm 5 and the last two columns show the results of Algorithm 5 alone.

Table 3.5: *Results of GA on preprocessed and non processed data*

		COLAMD and GA		*GA alone*	
Mat.	*Dens.*(%)	*Nbcols C1P*	*Time(s)*	*Nbcols C1P*	*Time(s)*
24	6	23	1.28	18	1.19
50	4	48	1.41	19	1.40
100	2	48	1.36	20	1.36
200	2	19	1.71	15	1.55
500	2	13	4.64	12	4.49
1000	2	13	13.13	10	13.66

Overall, these algorithms are applied to different instances of real-world and randomly generated matrices. The square nonsymmetric matrices that we generate from the set covering problem are not checked for the C1P, so the optimal values are not known. Consequently, the quality of our results cannot be discussed. The remaining matrices, Real-world data, (R_{1km}-R_{10km}) whose sizes are given in Table 3.6, arise from the stop location problem, posted by a Germany railway company [99]. The aim of this problem is to cover

a particular set of demand points by new stops over the track system. These points are covered if their distance to the closest stop is smaller than a provided covering radius r. The problem is formulated as a Set Covering Problem (SCP), see [99,100]. These instances provide binary matrices that are supposed to have almost C1P. Other matrices of types B and C of small size that are randomly generated by Ruf and Schöbel [99], are both sparse and almost have the C1P with density 3% and 5% respectively, see Table 3.8. Finally, the three algorithms are tested together as follows:

a) Apply Algorithm 5 after performing COLAMD. The average of the results are shown in columns 6 to 9 of Table 3.7, and in columns 5 to 8 of Tables 3.8 and 3.9.

b) Directly extract the C1S submatrix of a given matrix, then apply Algorithm 4. With respect to the five real-world matrices, we separate only two columns as a C1S submatrix then apply Algorithm 4. Results are in columns 10 to 13 of Table 3.7 and columns 9 to 12 of Tables 3.8 and 3.9.

c) Perform Algorithm 4 to the output of (**a**). The rest of columns in Tables 7 and 8 and columns 13 to 14 of Table 9 show the results.

Note that Tables 3.7, 3.8, and 3.9 show the average of number blocks and columns. The results of Table 3.7 show that Algorithm 4 in (**b**) gives a larger number of columns with C1P, but less blocks in comparison with those from other procedures (**a**, and **c**). The COLAMD with Algorithms 4 and 5 improves the number of consecutive blocks for all the matrices, but produces less columns with C1P in comparison with those from (**b**). Note that we apply Algorithm 4 to the initial C1S submatrices in column 4 in the table and in procedure (**c**) we apply it to the C1S submatrices in column 8.

The results of Table 3.8 show that almost 50% of the columns of the B and C matrices from Ruf and Schöbel satisfy the C1P. Although, Algorithm 4 costs less time and produces good submatrices with C1P, still applying this procedure alone is not enough to improve the number of consecutive blocks. The table shows that best results are from COLAMD with Algorithms 4 and 5.

The results on the real-world data of Ruf and Schöbel [99] are illustrated in Table 3.9. The first matrix R_{1km} satisfies the C1P since the number of final blocks is equal to m and

the columns number is equal to n, and thus optimal. This result is from each of COLAMD with Algorithm 5 and from Algorithm 4 separately. So there is no need to run the three algorithms together. With respect to the remainder matrices of this type, although the final numbers of blocks and columns differ with respect to the three procedures, they are close to the lower bounds m and n respectively. But, we can say that the column insertion algorithm is the best with respect to computation time. In comparison with the results of the CBM problem from [29], with respect to the number of blocks, we think our results do not differ a lot from theirs. Also, the numbers of columns with C1P are not far from the size of the matrices. The last column in the table shows their results. Also, the numbers of columns with C1P are not far from the sizes of the matrices. From the three tables we can say that:

1. Performing (**a**) improves the blocks but not the columns with C1P.

2. The result from Algorithm 4 does not rely on the size of the C1S only, but on the structure of the matrix.

3. Minimizing the number of blocks does not mean finding large C1S submatrix. We can say that improving the C1S can improve the CBM, but not necessarily the converse. That is because the size of the C1S submatrix relies on the position of the destructive column.

4. Over all, the number of consecutive blocks from (**c**) in the three tables is so far better than (**b**) and the number of columns with C1P does not differ a lot from (**b**) in Table 3.7 and is better in Tables 3.8 and 3.9. We can say that (**c**) is the best with respect to both problems. Nevertheless, Algorithm 4 is the best with respect to CPU time.

3.6 Sorting columns procedure

Another simple procedure for searching the neighborhood $\mathcal{N}(\rho'')$ to find a permutation that provides a matrix with maximum C1S and minimum number of consecutive ones blocks is proposed here. It depends on the summation of the ones in each column in the $(0, 1)-$matrix. This is a polynomial time algorithm that relies on the probability distribution of the 1's in

Table 3.6: *Test problem statistics*

Real − world matrices			*Randomly generated instances*					
Mat.	*Dens.*(%)	*Size*	*Mat.*	*Dens.*(%)	*Size*	*Mat.*	*Dens.*(%)	*Size*
R_{1km}	0.002	757×707	$B1$	0.032	100×96	$C1$	0.048	100×100
R_{2km}	0.014	1196×889	$B2$	0.036	100×95	$C2$	0.054	100×100
R_{3km}	0.022	1419×886	$B3$	0.034	100×92	$C3$	0.51	100×99
R_{5km}	0.043	1123×593	$B4$	0.0307	100×92	$C4$	0.05	100×100
R_{10km}	0.203	275×165	$B5$	0.029	100×92	$C5$	0.051	100×100

the columns. The procedure start with finding the number of 1's in each column; we then sort the sums, and rearrange the columns relying on their index. Details are provided in Algorithm 7. The type of sorting algorithm we used here is Quicksort as implemented in Matlab Toolbox, [101]; (see also [102,103]). The complexity of this algorithm costs $O(n \log n)$.

Lemma 3.6.1 *The complexity of the column sorting procedure with respect to the C1S problem is* $O((n - col)\, n \log n)$.

Proof. Sorting the columns costs $O(n \log n)$. The number of C1S improvement β is at most $n - col$ in the worst case, where col is the minimum number of C1S columns, and n is the maximum number of C1S. ■

Lemma 3.6.2 *The complexity of the column sorting procedure with respect to the CBM problem is* $O((g - m)\, n \log n)$.

Proof. Sorting the columns cost $O(n \log n)$. Since the number of blocks μ cannot be greater than g, and no less than m, then the number of improvements is at most $g - m$ in worst case. ■

The algorithm is tested on matrices with different densities. It is found that this procedure is not suitable for those with high density of 11% or more. From the experiments we found that it has an effect on those matrices with density $\leq 5\%$. Table 3.10 shows that for matrices B and C, the number of columns with the C1P from sorting algorithm is almost the same for some of those from the GA only, where the size of the population and the number of generations is set to 100. For the real-world instances the number of columns with C1P and the time are far better, but the number of blocks is worse. However, the results from

Algorithm 7: Sorting columns procedure

Input Positive integers m, n, binary $m \times n-$ matrix A;
for $i = 1, ..., n$
    p = sum the 1's in column i;
end for
q=sort $p_1, ..., p_n$;
Rearrange the columns according to on q;
Extract the C1S submatrix;
Return Permuted matrix.

combining the sorting algorithm with the column insertion in comparison with those from the columns insertion algorithm alone in Tables 3.8 and 3.9 show that combining them is better with respect to the number of columns. Finally, we can say that the sorting algorithm is not costly in comparison with the previous algorithms.

3.7 Summary

We presented a heuristic method for solving the C1S problem. A basic GA is proposed with many new fitness functions and the best is chosen to solve the C1S problem. The minimum consecutive blocks or CBM in [29] is also solved using the GA. The maximum C1S submatrix is improved by using a polynomial-time local algorithm with complexity about $O(m\sigma p)$. The same algorithm is used to solve the minimum consecutive blocks in which the complexity for finding only the CBM is $O(m\sigma(g - m))$. We applied our algorithms on a large number of randomly generated matrices and real-world instances. Since the optimal solutions are not known except for matrix R_{1km} and we do not know good upper or lower bounds for the C1S and CBM problems respectively, consequently, without comparison with previous methods, we cannot assess the quality of our results or judge their computing times.

Table 3.7: *Computational results of the C1P algorithm and the column insertion algorithm for nonsymmetric randomly generated matrices*

Initial information					(a) COLAMD and GA				(b) Column insertion				(c) COLAMD, GA, Insertion			
Mat.	Dens. (%)	Ge.	Initial blocks	Initial C1P	Final blocks	Blocks improv.	Nbcols C1P	Time(s)	Final blocks	Blocks improv.	Nbcols C1P	Time(s)	Final blocks	Blocks improv.	Nbcols C1P	Time(s)
100	2	20	208	22	121	87	48	2.933	148	60	83	0.113	125	83	88	3.124
100	4	20	398	12	294	104	13	3.049	331	67	47	0.124	274	124	49	3.283
100	10	20	960	7	478	482	13	2.811	788	172	26	0.121	472	488	21	3.371
200	2	20	791	19	553	238	18	3.894	651	140	93	0.497	519	272	101	4.919
200	4	20	1565	11	1106	459	14	4.677	1347	218	54	0.484	1070	495	46	5.36
200	10	20	3605	7	1740	1865	10	6.813	3320	285	28	0.485	1728	1887	27	6.593
500	2	20	4996	15	3601	1395	16	11.779	4348	648	115	5.092	3508	1488	89	15.951
500	4	20	9640	11	5558	4082	12	12.560	8891	749	67	4.971	5489	4151	49	15.69
500	10	60	22021	7	8472	13549	8	24.076	20821	1200	41	5.195	8371	13650	25	28.357

Table 3.8: *Computational results of the C1P algorithm and the column insertion algorithm for randomly generated matrices with almost C1P*

Initial information				(a) *COLAMD and GA*				(b) *Column insertion*				(c) *COLAMD, GA, Insertion*			
Mat.	*Ge.*	*Initial blocks*	*Initial C1P*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*
Randomly generated instances from Ruf and Schöbel with density 3%															
B1	60	296	12	265	31	19	6.725	267	29	47	0.053	252	44	48	6.793
B2	100	325	7	281	44	14	10.361	295	30	39	0.138	276	49	41	10.423
B3	100	304	10	259	45	14	10.420	279	25	38	0.125	255	49	46	10.479
B4	60	276	11	241	35	17	6.934	255	21	42	0.131	240	36	49	7.059
B5	60	270	13	221	49	15	6.959	242	28	48	0.134	215	55	54	7.093
Randomly generated instances from Ruf and Schöbel with density 5%															
C1	60	456	10	414	42	12	6.466	437	19	29	0.141	407	49	38	7.141
C2	100	516	7	452	64	11	10.614	496	20	22	0.148	450	66	29	10.676
C3	100	479	8	424	55	13	9.923	463	16	23	0.149	420	59	30	9.987
C4	100	482	7	421	61	13	9.754	468	24	23	0.134	417	65	32	9.816
C5	100	483	7	433	50	16	9.912	468	15	27	0.143	432	51	33	9.978

Table 3.9: *Computational results of the C1P algorithm and the column insertion algorithm for real-world instance matrices*

Initial information				(a) *COLAMD and GA*				(b) *Column insertion*				(c) *COLAMD, GA, Insertion*				*Algo.* (1, 2)
Mat.	*Ge.*	*Initial blocks*	*Initial C1P*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time(s)*	*Final blocks*
Real-world instances with density 2%																
R_{1km}	5	764	243	757	5	707	13.013	757	5	707	2.565	...	...	...	...	757
R_{2km}	5	1359	52	1206	153	820	31.513	1210	157	882	6.940	1210	157	882	36.620	1206
R_{3km}	5	1813	38	1479	334	617	38.648	1487	326	859	8.566	1472	341	867	47.568	1461
R_{5km}	5	1597	34	1203	394	342	15.551	1185	412	568	3.152	1190	407	569	20.315	1143
R_{10km}	5	389	32	287	102	147	1.499	281	108	159	0.081	281	108	162	1.685	280

Table 3.10: *Computational results of the C1P, the sorting and the column insertion algorithms*

Initial information			*GA*			*Sorting*			*Sorting and Insertion*		
Mat.	*Initial blocks*	*Initial C1P*	*Final blocks*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Nbcols C1P*	*Time(s)*	*Final blocks*	*Nbcols C1P*	*Time(s)*
B1	296	12	267	15	10.522	287	14	0.067	26 9	46	0.177
B2	325	7	295	13	9.716	320	13	0.075	309	38	0.152
B3	304	10	261	14	9.939	292	12	0.066	269	44	0.151
B4	276	11	251	14	9.803	274	15	0.085	254	46	0.167
B5	270	13	242	17	10.403	264	15	0.067	236	52	0.157
C1	456	10	423	10	10.373	454	13	0.079	438	33	0.189
C2	516	7	463	9	10.243	515	8	0.070	502	25	0.178
C3	479	8	436	9	10.046	477	10	0.078	459	26	0.167
C4	482	7	433	10	10.149	472	8	0.082	459	27	0.171
C5	483	7	446	9	10.334	485	10	0.101	468	33	0.174
R_{1km}	764	243	759	346	51.863	763	673	10.333	757	707	12.489
R_{2km}	1359	52	1301	174	97.136	1351	530	25.533	1231	874	38.224
R_{3km}	1813	38	1802	41	116.011	1809	368	33.031	1549	846	57.367
R_{5km}	1597	34	1557	33	66.929	1658	209	12.585	1307	548	27.584
R_{10km}	389	32	338	44	13.380	416	82	0.230	302	157	0.516

Chapter 4

Converting ILP with a $(0, 1)$ matrix into MILP

4.1 Intorduction

Integer Linear Programming (ILP) is in general NP-hard. It is shown to be NP-complete [8,9], by Cook [104]. The idea here is to transform an ILP with a (0, 1) matrix into a MILP with a (0, 1) matrix after showing that a submatrix of its matrix has the C1P property; In other words after solving the C1S problem. Note that matrices with C1P are totally unimodular, or TUM [65]. ILP with a TUM matrix can be solved as ordinary LP. Hence, the value of this idea. This chapter will also provide the computational complexity of the procedures that are used in Chapter 3 to solve the C1S problem.

4.2 Totally unimodular matrices

In LP when only integral solutions are permitted it is said to be ILP. Recall that LP has constraints

$$Ax = b, \ and \ x \geq 0 \tag{4.1}$$

where A and b are assumed to be integer matrices.

Definition 4.2.1 *[105] A matrix A is totally unimodular if the determinant of every square*

submatrix of A has value 1, 0 or -1.

This type of matrices appears in biological applications and in many combinatorial optimization problems. The main reason for the importance of this type of matrices comes from the following [106].

Theorem 4.2.1 *[57] If A is totally unimodular, then every basic solution of (4.1) is integer.*

This theorem can be strengthened for LP. In fact, given the following polyhedron

$$P = \{x : Ax \geq b,\ x \geq 0\}, \tag{4.2}$$

where A is an $m \times n$ matrix, b is a column vector in Z^m, total unimodularity of A is necessary and sufficient for all vertices of the polyhedron to be integer for every integer vector b. This result is given in the following theorem.

Theorem 4.2.2 *[105] An $m \times n$ matrix A is totally unimodular if and only if for each vector $b \in Z^m$, all vertices of the polyhedron $\{x \in R^n : Ax \geq b,\ x \geq 0\}$ are integral.*

The two previous theorems are important in ILP, since if A is totally unimodular [57], then

$$\begin{aligned} min \quad & c^T x \\ subject\ to \quad & Ax = b, \\ & x \geq 0,\ integer \end{aligned} \tag{4.3}$$

where A is an $m \times n$ matrix, b is a column vector in Z^m and c is a row vector in R^n, the problem can be solved as an LP by dropping the integrality constraint.

Theorem 4.2.3 *[105] If A is an $m \times n$ totally unimodular matrix and b is an integral vector, then*

$$conv\{x \in Z^n : Ax \geq b\} = \{x \in R^n : Ax \geq b\}. \tag{4.4}$$

In other words, a pure ILP can be solved in polynomial time via linear programming. That is because if A is totally unimodular and b is integral, the convex hull of the pure integer set $\{x \in Z^n : Ax \geq b\}$ and its linear relaxation $\{x \in R^n : Ax \geq b\}$ are the same

polyhedron. Note, that total unimodularity is sufficient condition for ILP to be solvable as LP, However, it is not necessary condition.

4.3 Turning ILP into MILP

Definition 4.3.1 *A Mixed Integer Linear Programming (MILP) is an optimization program which includes continuous and integer variables, a linear objective function, and linear constraints. The MILP can be written as follow.*

$$(MILP): \begin{cases} min & c^T x + f^T y \\ s.t & Ax + By \geq b, \\ & x \in Z_+^d, \;\; y \in R_+^q \end{cases} \tag{4.5}$$

where $c \in R^d$ and $f \in R^q$ are the coefficients row vectors of the objective function, x and y denote the column vector of the nonnegative integer variables of $d-$dimention and the column vector of nonnegative continuous variables of $q-$dimention, respectively. The matrices A and B with dimensions $m \times d$ and $m \times q$ respectively represent the constraints with real coefficients and $b \in R^m$ is the column vector of the right hand side.

We describe our experimental investigation of this chapter as follows:

Problem

Given a pure ILP with (0, 1)−matrix, find the C1S of the matrix of the problem which has the C1P property, i.e, convert the ILP with (0, 1)−matrix into an MILP.

1. Solve the ILP by B&B algorithm,
2. Apply one of the procedures that we have introduced to reorder the columns of the matrix,
    - If the whole matrix is C1P then solve it as an LP problem, otherwise,
    - Find the C1S submatrix, then solve the new problem as MILP,

3. Compare the complexity of the ILP problem with the MILP problem and the procedure that is used for the conversion.

For the basic binary linear programming problem

$$(ILP): \qquad min\,\{c^T u \;:\; Au \geq b,\; u \in \{0,\,1\}\}, \tag{4.6}$$

if the result of the reordering provides the $m \times n-$matrix A with C1P for the whole matrix then we can solve it as LP problem using the simplex method. If a part of it satisfies the C1P, then A will be separated into two parts $A = (A_1,\, A_2)$. Without loss of generality, assume that, after reordering the columns, A_1 has the first p columns not having the C1P property and A_2 has the second $n-p$ columns with the C1P. Consequently, the cost and the variables will be separated into two parts $c = (c_1,\, c_2)$, and $u = (x,\, y)$. A_2 is totally unimodular, so we can relax the integrality constraint on y. The suggested transformation leads to a MILP. The new problem is expressed as in (4.5) where $x \in \{0,\,1\}^p,\;\; y \in R^{n-p}$.

From the preceding claim, the ILP with $(0,\,1)-$matrix and the MILP with $(0,\,1)-$matrix are equivalent. So $(x,\, y)$ is an optimal solution for the latter if and only if u is an optimal solution for the ILP [107]. Our concern, first, is to apply the procedures to matrices of the set covering problem.

Definition 4.3.2 *[108] The problem of covering the rows of a binary $m \times n-$matrix (a_{ij}) by a subset of the columns at minimal cost is a Set of Covering Problem (SCP). Defining $x_j = 1$ if column j (with cost $c_j > 0$) is in the solution and $x_j = 0$ otherwise, the SCP is*

$$\begin{aligned} min \quad & \sum_{j=1}^{n} c_j x_j \\ s.t \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, ..., m \\ & x_j \in \{0, 1\}, \qquad j = 1, ..., n \end{aligned} \tag{4.7}$$

The MILP in (4.5) is turned to

$$(MISC): \begin{cases} min & c_1^T x + c_2^T y \\ s.t & A_1 x + A_2 y \geq 1, \\ & x \in \{0,1\}^p, \;\; y \in R^{n-p} \end{cases} \tag{4.8}$$

4.4 Complexity of the reduction

For the following ILP

$$max\ \{c^T x \,|\, Ax \leq b\,;\ x\ integer\} \tag{4.9}$$

Using B&B to solve the problem may have a number of iterations. In each stage there is a collection of polyhedra. Suppose that at stage 1 $\prod_1 := \{P\}$, and at stage i $\prod_i := \{P_1, ..., P_i\}$ such that $P_1, ..., P_i$ are pairwise disjoint polyhedra in R^n and all integral vectors in P are contained in $P_1 \bigcup ... \bigcup P_i$. The computational time of B&B is not polynomially bounded by the size of the input. Consider the problems ILP_k, for $k = 1, 2, ...$ [8],

$$ILP_k : max\ \{\delta \,|\, 2^k\beta = (2^k + 1)\delta\,; 0 \leq \beta \leq 2^k\,;\ \delta,\ \beta\ integer\} \tag{4.10}$$

The origin $(\delta, \beta) = (0, 0)$ is the only feasible solution for the ILP_k. The method requires at least 2^k iterations.

Suppose that $\xi_j := max\{cx \,|\, x \in P_j\}$, for $j = 1, ..., i$ are the optimum values. Choose j^* such that $\xi_{j*} = max\{\xi_j | j = 1, ..., i\}$ and let x^* reaches the maximum for $j = j^*$. The maximum optimum value ξ_{j*} is reached in the i^{th} iteration is at least $2^k - i$. This comes from the fact that, by indication on i, in the i^{th} stage one of the polyhedra P_j contains the following subset

$$\{\tbinom{\beta}{\delta} \,|\, 2^k\beta = (2^k + 1)\delta\,; 0 \leq \delta \leq 2^k - i\} \tag{4.11}$$

one simply provides that if this subproblem divide into two subproblems, one of the two new subproblems will contain

$$\{\binom{\beta}{\delta} \mid 2^k\beta = (2^k + 1)\delta\,; 0 \leq \delta \leq 2^k - i - 1\} \tag{4.12}$$

as a subset.

This shows that there is no polynomial upper bound on the computation time as the size of IP_k is linear in k. In general, the method needs 2^k iterations, but it can be more or less the number of columns of matrix A, i.e. the dimension of the problem.
In fact, for a knapsack problem,

$$\begin{aligned} max \quad & \textstyle\sum_{i=1}^{n} c_i u_i \\ s.t \quad & \textstyle\sum_{i=1}^{n} a_i u_i \leq b \quad u_i = 0,\, 1 \ (i = 1, ..., n), \end{aligned} \tag{4.13}$$

where a_i, b, c_i are positive integer, Chvátal 1980 [109], provided that the procedure of B&B needs $2^{n/10}$ iterations when n is large enough. This is satisfied for almost all knapsack problems of this type. Jeroslow 1974 [110], proved that, any B&B procedure for the problem

$$max\ \{\gamma_1 \mid 2\gamma_1 + ... + 2\gamma_n = n\,;\, 0 \leq \gamma_i \leq 1\,;\, and\ integer\, (i = 1, ..., n)\} \tag{4.14}$$

for n odd, needs at least $2^{(n+1)/2}$ iterations before showing the infeasibility of (4.14). He showed that for any B&B method, for n odd, independent of the heuristic employed, of the variables order and the way of selecting the nodes, it should expand out at least $2^{n/2}$ nodes before it finds that (4.14) is inconsistent [110, 111].

Finding the C1S potentially reduces the number of columns required to branch on by using B&B procedure. The subproblem with C1P is solved as an LP problem via solving the MILP problem. The complexity of this part over the process is $O(m(n - p))$. We reduce the complexity of solving the ILP from 2^n to $2^{n-(n-p)} = 2^p$ iterations, where $n - p$ is the dimension of the continuous part and p that of the integer part. So the complexity of each iteration of the MILP is $O(2^p) + O(m(n - p))$. For large n, the cost is less than 2^n iterations.

4.5 Complexity of the GA

The theory of the computation time and time complexity analysis of Evolutionary Algorithms (EAs) is an essential topic. It shows the predictable number of generations to reach the optimum [112–115]. Although, EAs have many applications in discrete optimisation, theoretical results that are related to time complexity are relatively few, [113,116,117].

In general, EAs use a large number of parameters that are set before running the algorithm. The effectiveness of the search implemented by an EA can be greatly affected by these parameters. Until now researchers could not find theoretically robust instructions to help them choose good values for these parameters. Choosing the size of the population is one of these parameters which is regarded as a frequent issue in evolutionary computation applications. Genetic algorithms [118] and genetic programming [119] use equal size for both of parent μ and the offspring λ. Nevertheless, in Evolution Strategies (ESs) [73] these sizes are conventionally chosen independently, for $(\mu + \lambda)-$variant. Choosing λ greater than μ raises an open question about the benefit of that and if so what should the value be. Most of the GA and EP written work considers the case of $\mu = \lambda$ and concentrates on choosing a suitable value and in the literature of the ES, the $(1 + 1)-$ES plays an important role [120].

An overview of some works that have been done on the time complexity is [121]: Ambati et al. [122] and Fogel [123] found the complexity time of the EAs using the Traveling Salesman Problem. For the linear function with Boolean inputs, Droste et al. [124], proposed a rigorous theoretical complexity analysis of $(1 + 1)-$EAs. All the previous results relied on the simplest EAs with population size 1 and without using mating. He et al. [125] proved that for solving some deceptive problems, GAs may run in exponential time.

Among the studies of EAs, the one on unimodal problems (ONEMAX and LEADINGONES) has been widely investigated with respect to the time complexity of EAs [115,116, 120,121,126–128]. These problems are regarded as simple for EAs and their analysis shows how the latter solves optimisation problems. For the ONEMAX problem, Rudolph [129] showed that the $(1 + 1)-$EAs is $O(n \log n)$, where n is the number of bits in a binary chromosome. In a simple $(1 + 1)-$EA, the child is acceptable for the EA, if its fitness value is at least as good as the parents fitness [130]. Many papers showed how the unimodal problems are

optimized by population-based EAs [126], but the results are limited to case studies. Some of the work present theoretical results to these problems, such as the $(1+\lambda)-$EA [120], the $(N+N)-$EAs [115,121], and $(\mu+1)-$EA [128]. See Appendix C for some interesting general results.

4.5.1 Complexity of basic GA

Based on drift analysis of He and Yao [121] (see Appendix C), we will attempt to estimate the complexity of our GA, but here for minimization. We estimate the cost of the algorithm based on the CBM fitness function that was used in the previous chapter to solve the C1S problem.

Given a finite state space V and a function $f(y)$, $y \in V$, find

$$min\{f(y); y \in V\}. \tag{4.15}$$

Here the GA is described for the CBM function as follows: Let λ_i be the i^{th} generation population.

1. Generate randomly an initial population of $2N$ individuals, denoted by $\lambda_0 = (y_1, ..., y_{2N})$; let $i \leftarrow 0$ and N be an integer number, $N > 0$. Define $f(\lambda_i) = min\{f(y_i) : y_i \in \lambda_i\}$ for any population λ_i.

2. Generate a new population of offspring using crossover and mutation operators; denote it $\lambda_i^{(V)}$, where the crossover operator produces an offspring population denote it as $\lambda_i^{(C)}$ and the mutation operator produces an offspring population denote it as $\lambda_i^{(M)}$.

3. Stop if $f(\lambda_i^{(V)}) = f_{min}$; else let $\lambda_{i+1} = \lambda_i^{(V)}$ and $i \leftarrow i+1$, and go to step 2.

The drift analysis is described as follows: Let $d(y, y^*)$ be the distance between a point y and the optimum y^*. In case there are more than one optimal value then the distance between the chromosome y and the optimal set V^* is suggested as $d(y, V^*) = min\{d(y, y^*) : y^* \in V^*\}$, simply noted as $d(y)$. The distance from a given population $Y = \{y_1, ..., y_{2N}\}$ to the

optimum is

$$d(Y) = min\{d(y) : y \in Y\} \tag{4.16}$$

The GA generates a random sequence $\{d(\lambda_i);\ i = 0, 1, 2, ...\}$ which can be modelled by a homogeneous Markov chain. The drift of this sequence at time i is described as

$$\Delta(d(\lambda_i)) = d(\lambda_{i+1}) - d(\lambda_i)$$

The time of the GA to stop is defined as $t = min\{i : d(\lambda_i) = 0\}$ which represents the first hitting time on the optimum value. See C.1.1 at the page 156 for more details. He and Yao put many conditions for their EA; here we follow two of them. The first condition indicates that the distance from any population to the optimum is bounded by a polynomial function in the problem size. The second condition is that the drift of the random sequence toward the optimum is always positive and bounded by an inverse polynomial.

Condition 1. There exists a polynomial of problem size n, $h_0(n) > 0$, such that $d(Y) \leq h_0(n)$ for any given population Y.

Condition 2. At any time $i \geq 0$, if population λ_i satisfies $d(\lambda_i) > 0$, then there exists a polynomial of problem size n, $h_1(n) > 0$, such that

$$E[d(\lambda_i) - d(\lambda_{i+1}) \,|\, d(\lambda_i) > 0)] \geq \frac{1}{h_1(n)}. \tag{4.17}$$

Notice, that we can reverse this equation to the case of minimizing objective functions, as in

$$E[d(\lambda_{i+1}) - d(\lambda_i) \,|\, d(\lambda_i) > 0)] \geq \frac{1}{h_1(n)}.$$

Theorem 4.5.1 *[121] If $\{d(\lambda_i);\ i = 0, 1, ...\}$ satisfies Conditions 1 and 2, then starting from any initial population Y with $d(Y) > 0$, $E[\,t \,|\, d(\lambda_0) > 0] \leq h(n)$, where $h(n)$ is a polynomial of problem size n.*

Now we state and prove our main result. A solution to the C1S problem is represented by a chromosome $y = (v_1, ..., v_n)$, $v_n \in Z^+$.

Theorem 4.5.2 *Given a C1S problem and a GA to solve it, for any initial population with* $d(Y) > 0$,

$$E[\, t \,|\, \lambda_0 = Y] \leq h(n),$$

where $h(n)$ *is a polynomial of* n.

Proof. Define the distance function $d(y)$ as:

$$d(y) = \sum_{i=1}^{m} \sum_{j=1}^{k_i} b_j,$$

where m is the number of rows in the matrix, k_i is the number of blocks in the i^{th} row, and b_j takes value 1. We need to prove that the random sequence $\{d(\lambda_i);\ i = 0, 1, ...\}$, satisfies Conditions 1 and 2 according to Theorem (4.5.1). From the definition of the above distance and by equation (4.16), for any population Y; $d(Y) \leq mn/2$ for even n, $(d(Y) \leq (mn + m)/2$ for odd n, we prove here the theorem for even number), where m is the number of rows in a (0, 1)−matrix and the maximum number of blocks is $mn/2$. We reach the optimum when the number of blocks in the final generation equals m and so the number of C1P columns is n. Hence the random sequence $\{d(\lambda_i); i = 0, 1, ...\}$ satisfies Condition 1. Now at any time $i \geq 0$, and population λ_i with $\lambda_i > 0$, we test the effect of the crossover on the drift. Some of the cases which may occur after crossover are:

1. Case I : $\{d(\lambda_i^{(C)}) < d(\lambda_i)\}$,
2. Case I : $\{d(\lambda_i^{(C)}) = d(\lambda_i)\}$, or
3. Case I : $\{d(\lambda_i^{(C)}) > d(\lambda_i)\}$.

I is the event that may happen. The third case cannot occur because crossover does not bring out a worse intermediate population. In fact, increasing the drift of one individual causes decreasing the drift of another individual, so crossover will not produce a worse intermediate population. Suppose that case $\{\lambda_i^{(C)} = \lambda_i\}$ has occurred. Then one of the following cases may occur subsequently:

a) Case I : $\{\lambda_i^{(M)}) < d(\lambda_i^{(C)})\}$,

b) Case I : $\{\lambda_i^{(M)}) = d(\lambda_i^{(C)})\}$, and

c) Case I : $\{d\lambda_i^{(M)}) > d(\lambda_i^{(C)})\}$.

Case **b** cannot occur because mutation always happens. The probability of Case **a** is not less than $(1/n)(1/n-1)$ (if $d(\lambda_i^{(C)}) > 0$), and the probability of Case **c** is not greater than $1-(1/n)(1/n-1)$. Note that the mutation operator we used swaps two genes. If $d(\lambda_i^{(C)}) = 0$, then the population λ_{i+C} has one individual with minimum fitness. Suppose that Case $\{\lambda_i^{(C)}) < d(\lambda_i)\}$ has occurred. Then also the Cases **a**, **b**, and **c** may occur subsequently and the same probabilities of the three events are similar to those of the analysis in Case 2. To this end, best individual that appears in the next population may have probability $1-e^{-n}$. Thus probability of $\{\lambda_i^{(M)} < d(\lambda_i)\}$ is not less than $1-e^{-n}$, while probability of $\{\lambda_i^{(M)}) > d(\lambda_i)\}$ is not more than e^{-n}. Considering all the cases, we get

$$
\begin{aligned}
&E[d(\lambda_{i+1}) - d(\lambda_i) | d(\lambda_i) > 0] \\
&\quad = E[(d(\lambda_{i+1}) - d(\lambda_i))\, I\,\{d(\lambda_i^{(C)}) < d(\lambda_i),\ \lambda_i^{(M)}) < d(\lambda_i^{(C)}),\ d(\lambda_i^{(M)}) < d(\lambda_i)\} \,|\, d(\lambda_i) > 0] \\
&\quad + E[(d(\lambda_{i+1}) - d(\lambda_i))\, I\,\{d(\lambda_i^{(C)}) < d(\lambda_i),\ d(\lambda_i^{(M)}) > d(\lambda_i^{(C)}),\ \lambda_i^{(M)}) > d(\lambda_i)\} \,|\, d(\lambda_i) > 0] \\
&\quad + E[(d(\lambda_{i+1}) - d(\lambda_i))\, I\,\{d(\lambda_i^{(C)}) = d(\lambda_i),\ \lambda_i^{(M)}) < d(\lambda_i^{(C)}),\ \lambda_i^{(M)}) < d(\lambda_i)\} \,|\, d(\lambda_i) > 0] \\
&\quad + E[(d(\lambda_{i+1}) - d(\lambda_i))\, I\,\{d(\lambda_i^{(C)}) = d(\lambda_i),\ \lambda_i^{(M)}) > d(\lambda_i^{(C)}),\ \lambda_i^{(M)}) > d(\lambda_i)\} \,|\, d(\lambda_i) > 0]
\end{aligned}
$$

We use the fact that $|d(\lambda_{i+1}) - d(\lambda_i)| \leq mn/2 - 1$, and since

$P(d(\lambda_i^{(C)}) < d(\lambda_i) \,|\, d(\lambda_i > 0)) + P(d(\lambda_i^{(C)}) = d(\lambda_i) \,|\, d(\lambda_i > 0)) = 1$, in other words,

$$
\begin{aligned}
&E[d(\lambda_{i+1}) - d(\lambda_i) \,|\, d(\lambda_i) > 0] \\
&\quad \leq (-1)P(d(\lambda_i^{(C)}) < d(\lambda_i), \lambda_i^{(M)}) < d(\lambda_i^{(C)}) \,|\, d(\lambda_i) > 0)(1 - e^{-n}) \\
&\quad + (mn/2 - 1)P(d(\lambda_i^{(C)}) < d(\lambda_i), \lambda_i^{(M)} > d(\lambda_i^{(C)}) \,|\, d(\lambda_i) > 0)(e^{-n}) \\
&\quad + (-1)P(d(\lambda_i^{(C)}) = d(\lambda_i), \lambda_i^{(M)}) < d(\lambda_i^{(C)}) \,|\, d(\lambda_i) > 0)(1 - e^{-n}) \\
&\quad + (mn/2 - 1)P(d(\lambda_i^{(C)}) = d(\lambda_i), \lambda_i^{(M)}) > d(\lambda_i^{(C)}) \,|\, d(\lambda_i) > 0)(e^{-n}).
\end{aligned}
$$

Then we have

$$
\begin{aligned}
&E[d(\lambda_{i+1}) - d(\lambda_i) \,|\, d(\lambda_i) > 0] \\
&\quad \leq (-1)P(d(\lambda_i^{(C)}) < d(\lambda_i) \,|\, d(\lambda_i) > 0)(\tfrac{1}{n})(\tfrac{1}{n-1})(1 - e^{-n}) \\
&\quad +(mn/2 - 1)P(d(\lambda_i^{(C)}) < d(\lambda_i) \,|\, d(\lambda_i) > 0)\tfrac{n^2-n-1}{n(n-1)}(e^{-n}) \\
&\quad +(-1)P(d(\lambda_i^{(C)}) = d(\lambda_i) \,|\, d(\lambda_i) > 0)(\tfrac{1}{n})(\tfrac{1}{n-1})(1 - e^{-n}) \\
&\quad +(mn/2 - 1)P(d(\lambda_i^{(C)}) = d(\lambda_i) \,|\, d(\lambda_i) > 0)\tfrac{n^2-n-1}{n(n-1)}(e^{-n}). \\
&\quad \leq (-1)(\tfrac{1}{n})(\tfrac{1}{n-1})(1 - e^{-n}) + (mn/2 - 1)\tfrac{n^2-n-1}{n(n-1)}e^{-n} \\
&\quad \leq -\tfrac{1-e^{-n}-(mn/2-1)(n^2-n-1)e^{-n}}{n(n-1)}.
\end{aligned}
$$

Let

$$h_1(n) = \tfrac{n(n-1)}{1-e^{-n}-(mn/2-1)(n^2-n-1)e^{-n}}.$$

then when $n \to +\infty$, $h_1(n) = O(n(n-1))$. Hence,

$$E[d(\lambda_{i+1}) - d(\lambda_i) \,|\, d(\lambda_i) > 0] \leq -\tfrac{1}{h_1(n)} \text{ and } \lim_{x\to\infty} \frac{-1}{h_1(n)} < 0,$$

where $h_1(n) = O(n(n-1))$. This verifies that the random sequence $\{d(\lambda_i); i = 0, 1, ...\}$ satisfies condition (2). According to Theorem (4.5.1) , we know $E[\, t \,|\, \lambda_0 = Y] \leq h(n)$. ■

On the other hand, based on He and Yao [115], who proposed a drift analysis for maximum problem, specifically Theorem C.1.12 in Appendix C, we test the drift direction of our GA. We used the distance function $G(\mathbf{y}) = min\{|f(\mathbf{y}) - f(\mathbf{z}) \,|\, ; \mathbf{z} \in E^*\}$. Even if the C1S fitness function is to be minimized, still the distance function gives the right distance from the population of a certain generation to the optimal population. For small (0, 1)−matrices, applying the one-step mean drift shows that the drift is positive and converges to the optimum. This result can be concluded in the following lemma.

Lemma 4.5.3 *The mean number of generations required by the GA to solve the C1S problem is polynomial in the problem size. It belongs to the easy class according to [115].*

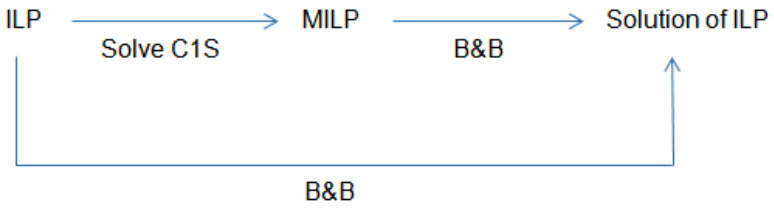

Figure 4.1: *Complexity of the transformation*

4.6 Complexity issues

The main issue here is to show that transforming an ILP into an MILP after solving a C1S problem is worthwhile. While the solution of the MILP is via B&B, that of the C1S is not. We used a number of procedures including GA, Column Insertion, and Column Sorting algorithms. In the following we study complexity of the transformation depending on the algorithm used, coupled with the solution approach to the resulting MILP.

4.6.1 Transformation with GA and solution of MILP with B&B

Here, the C1S is solved with a GA resulting in a submatrix with C1S of $(n-p)$ columns. GA takes $O(n(n-1))$ operations, but the solution to the MILP with B&B takes $O(2^p)$ for the integer part and $O(m(n-p))$ for the continuous part giving a total of $O(n(n-1))+O(2^p)+O(m(n-p))$. Of course, this is only a rough estimate since if p is 0, i. e. the whole matrix of ILP has C1P, then the MILP is just an LP. Similarly if no column have C1P, then the ILP is pure, and the complexity is $O(2^p)$, where $p = n$. The real issue is really to estimate the number of columns of the ILP that must have C1P for the whole approach to be worthwhile. This is a very hard question to answer indeed. Our aim is just to show what happen in the worst case.

4.6.2 Transformation with column insertion and solution of MILP with B&B

We proceed in the same way above. Suppose we use only the insertion procedure to transform the ILP into MILP. The complexity of this procedure is $O(m\sigma p)$ where σ is the size of exploring the neighborhood $\mathcal{N}''(\rho)$. Adding this time to the one from the MILP we

get $O(m\sigma p) + O(2^p) + O(m(n-p))$. For small matrices this can be easily solved by B&B, but the conversion may cost more time. For large matrices we believe that the time is less especially if the continuous part is larger than the integer part. Our trials show that the number of nodes of the B&B search tree of the ILP is always larger than the number of nodes of the B&B search tree of the MILP. This can explain the reduction in computing time.

Illustration:

We provide some examples to show how increasing the C1S columns effects the complexity, we compare the $O(m\sigma p) + O(2^p) + O(m(n-p))$ of the MILP with the 2^n of ILP. Given n, for any $p \geq n-p$, for small matrices the ILP time is better than the MILP, but for $p < n-p$ the time of latter is better. For large matrices the time of the MILP is better for any p.

Example1:

For 10×10 matrix A with $p = 5$, $n - p = 5$, σ the cost is as follows:

$\sigma = \sum_{N=5}^{9} N - 1 = 4 + 5 + 6 + 7 + 8 = 30$

$O(m\sigma p) = 10 \times 30 \times 5 = 1500$ the complexity of the insertion algorithm,

$O(m(n-p)) = 50$ the complexity of the continuous part,

$O(2^p) = 2^5 = 32$ the complexity of the integer part.

Total complexity = 1500 + 50 + 32 = 1582, while $2^{10} = 1024$. The B&B time is better, but for the next one.

Example2:

For 10×10 matrix B with $p = 7$, $n - p = 3$, the total sum of the MILP is 2808, while $2^{10} = 1024$ still the B&B time is better.

Example3:

For 10×10 matrix B with $p = 3$, $n - p = 7$, the total sum of the MILP is 528, while $2^{10} = 1024$ the B&B time is less than the MILP time.

Example4:

For 20×20 matrix B with $p = 15$, $n - p = 5$, the total sum of the MILP is 65168, while $2^{20} = 1048576$ which is about 16 times the conversion cost.

Example5:

For 20×20 matrix B with $p = 10$, $n - p = 10$, the total sum of the MILP is 28224, while $2^{20} = 1048576$ which is about 37 times the conversion cost.

Example6:

For 20×20 matrix B with $p = 5$, $n - p = 15$, the total sum of the MILP is 8332, while 1048576 is about 125 times the conversion cost.

4.6.3 Transformation with column sorting and solution of the MILP with B&B

In the same way as in the above two solutions, we can estimate the complexity of this combined solution approach as follows. Since the complexity time of the sorting procedure is $O((n - col)\, n \log n)$, the total time therefore is $O((n - col)\, n \log n) + O(2^p) + O(m(n - p))$.

4.7 Summary

In this chapter, we introduced the conversion of the ILP with a (0, 1) matrix into a MILP with a (0, 1) matrix. The complexity of the conversions was studied. The chapter provides an analysis of the procedures that have been proposed to solve the C1S problem. We worked out the complexity of the GA, column insertion, and the sorting procedures for converting ILP into MILP, and that of the solver of the resulting MILP. Hence, overall complexity was estimated. In the following, we take a very different approach which starts by studying the problem of minimum volume ellipsoids enclosing a given set of points in R^n.

Chapter 5

An Evolutionary Approach to Constructing the Minimum Volume Ellipsoid Containing a Set of Points and the Largest Volume Ellipsoid Embedded in a Set of Points

5.1 Introduction

In this chapter we consider first the problem of estimating the Löwner-John ellipsoid. It can be described as follows. Given a set C of points in R^n, what is the minimum volume ellipsoid that contains all the points of C? This is referred to as the Minimum Volume Enclosing Ellipsoid or MVEE problem. When only a subset of C is required to be enclosed then this version of the problem is referred to as the Minimum Volume Ellipsoid, or MVE, estimator problem. Both MVEE and MVE have good applications as we shall see later. The third problem, also related to MVEE, that we consider here is that of finding the Largest Volume Enclosed Ellipsoid or LVEE, that can be embedded within the set of points C in R^n, without containing any of the points. This can be described as the problem of finding the biggest hole within a set of points in some dimension. Equivalently, it is the problem

of finding the largest convex hull that is embedded within the set of discrete points. In the following we treat these problems in turn and show how they can be formulated, and solved in particular using an evolutionary approach. We will also show how they can be applied to C1S. Illustrations will be provided as well as computational results.

5.2 The Minimum Volume Enclosing Ellipsoid Problem

5.2.1 The ellipsoid: Preliminaries

An ellipsoid in R^n is the image of the unit ball B^n of R^n under a nonsingular affine map of the form.

$$\Psi(x) := Ax + b \tag{5.1}$$

where $\Psi : R^n \rightarrow R^n$ is linear and $b \in R^n$ is a constant vector. The ellipsoid has many representations, such as

$$E = \{x \in R^n | (x-b)^T A^{-1}(x-b) \leq 1\} \tag{5.2}$$

for $b \in R^n$ and $A = A^T > 0$. Ellipsoids can approximate any bounded convex set in R^n within a factor of n, [131].

The MVEE or Löwner-John (L-J) ellipsoid E_{lj} of a set C is the minimum volume ellipsoid that contains the set C. To characterize E_{lj}, a general ellipsoid can be parameterized conveniently as

$$E = \{v \mid ||Av + b||_2\} \leq 1, \tag{5.3}$$

which is the inverse image of the unit ball under an affine mapping. Without loss of generality, we can assume that the volume of E is proportional to $detA^{-1}$. For $A \in S_+^n$[1], the problem of finding the MVEE of C can be written as

$$\begin{aligned} min \quad & log\ detA^{-1} \\ s.t \quad & sup_{v \in C} ||Av + b||_2 \leq 1, \end{aligned} \tag{5.4}$$

[1] S^n and S_+^n denote $n \times n$ symmetric matrices and symmetric positive definite matrices, respectively. We use $>$ to denote strict matrix inequality between symmetric matrices.

for an implicit constraint $A > 0$. The variables are $A \in S^n$, and $b \in R^n$. This problem is convex as both the objective and the constraint functions are convex in A and b.

5.2.2 The MVEE probelm

MVEE arises in a number of practical situations such as when placing relay antennae, packing and packaging to name a few. It is commonly solved using convex programming and boils down to finding the so-called Löwner-John ellipsoid [11]. Computing the MVEE of a finite set of points $x_1, ..., x_m \in R^n$ is equivalent to computing the MVEE of a polytope that is defined by the convex hull of those points. By applying (5.4) this problem can be written as

$$\begin{aligned} min \quad & log\ detA^{-1} \\ s.t \quad & ||Ax_i + b||_2 \leq 1,\ i = 1, 2, ..., m, \end{aligned} \tag{5.5}$$

for the implicit constraint $A > 0$, and the variables $A \in S^n$ and $b \in R^n$, [132]. The problem goes back a long way. Indeed, Fritz John (1948) [11] showed that for an arbitrary nonempty set C of points, $C \subseteq R^n$ can be n-rounded, meaning that for some ellipsoid $E \subseteq R^n$,

$$n^{-1}E \subseteq conv(C) \subseteq E, \tag{5.6}$$

where $conv(C)$ is the convex hull of C. The result holds for the Löwner-John ellipsoid of C, or the minimum volume ellipsoid containing C, which is our concern here.

The problem of approximating E has many applications in statistics and optimal design, [133, 134], integer programming, [135, 136], computational geometry [137], and randomized algorithms for polytope volume computation, [138, 139]. Leonid G. Khachiyan [140] considered the problem of $(1 + \epsilon)n$-rounding of a set C of m points in R^n, which boils down to finding an ellipsoid $E \in R^n$ where

$$[(1 + \epsilon)n]^{-1}E \subseteq conv(C) \subseteq E. \tag{5.7}$$

He suggested an $O(mn^2(\epsilon^{-1} + \ln n + \ln \ln m))$ algorithm, which implies that approximating the minimum volume ellipsoid of C can be done in $O(m^{3.5}ln(m\epsilon^{-1}))$ operations, where ϵ is

a relative accuracy.

Kumar and Yildirim [12] addressed the problem of calculating a $(1+\epsilon)$-approximation of the MVEE of a set C of m points by modifying Khachiyan's algorithm. Their algorithm has the better complexity bound of $O(mn^3/\epsilon)$ operations for $\epsilon \in (0,1)$. It returns a core set $\mathcal{Y} \subset C$ such that the MVEE of $\mathcal{Y}$ gives a good approximation of C. Moreover, the size of $\mathcal{Y}$ does not rely on the number of points m, but only on the dimension n and ϵ. Their results give $|\mathcal{Y}| = O(n^2/\epsilon)$ for $\epsilon \in (0,1)$.

Todd and Yildirim [141] studied two related problems; finding an approximate rounding ellipsoid of the convex hull of a finite set of points and finding the MVEE of a finite set of points. They found a relationship between the polar of the deepest cut ellipsoid method and Khachiyan's barycentric coordinate descent (BCD) method which calculates an approximate rounding of the convex hull of a set of vectors. They implemented Khachiyan's BCD method in an efficient way and also provided a modification of the algorithm of Kumar and Yildirim, [12]. They found that their modification does not raise the complexity of the latter algorithm. Their algorithm calculates a good approximate solution and calculates a smaller core set than that computed by the algorithm of Kumar and Yildirim, [12].

The minimal ellipsoid circumscribing a polytope that is defined by a set of finitely many points can be computed by many algorithms. Algorithms that use the interior point approach were introduced by Zhang [142], Zhang and Gao [143], and Khachiyan and Todd [144]. The one provided by Barnes [145] relies on quadratic programming. There are other algorithms of the stochastic type, introduced by Gärtner and Schönherr [146]. Other studies of the problem are by Yildirim [147], Vaidya [148], and Anstreicher [149].

Sun and Freund [150] introduced the Dual Reduced Newton (DRN) algorithm which solves large instances that arise in data mining contexts. It is based on combining active-set [150] and interior-point methods. The DRN is regarded as an interior point algorithm. It was applied to 10 randomly generated large problem instances, ($10 \leq n \leq 30$ and $1000 \leq m \leq 30000$).

Welzl [151] developed a simple randomized algorithm that finds the smallest enclosing disks (balls and ellipsoids) in the plane for a set of points in linear time. This algorithm depends on Seidel's Linear Programming algorithm [152] which works mainly in low

dimensions. Welzl generalized it to higher dimensions and provided a heuristic that leads to an improved procedure which finds the smallest enclosing ball for 5000 points in 10-dimensions.

As mentioned above, for some ellipsoid $E \subseteq R^n$, an arbitrary set C of m points can be n-rounded. In this case, if C is with nonempty interior, and is symmetric about the origin, that is $-C = C$, then the rounding factor n can be tightened to $\sqrt{n}$.

Let E_{lj} be the Löwner-John ellipsoid of the convex, bounded and nonempty set $C \subseteq R^n$, with the center x_0. The ellipsoid can be shrunk by a factor of n about its center, which gives an ellipsoid that lies inside the set C:

$$\{x_0 + (1/n)(E_{lj} - x_0)\} \subseteq C \subseteq E_{lj} \tag{5.8}$$

This means, that within a factor that relies only on the dimension n, an arbitrary convex set can be approximated by the Löwner-John ellipsoid.

In the case, the set C is symmetric about a point x_0, we can tighten the factor 1/n to $1/\sqrt{n}$:

$$x_0 + (1/\sqrt{n})(E_{lj} - x_0) \subseteq C \subseteq E_{lj} \tag{5.9}$$

5.2.3 Computing MVEE in low dimensions

We illustrate briefly here how to compute MVEE using existing software such as the CVX package of Matlab, [153], which implements the Path-Following Infeasible Interior-Point Algorithm. Instances of the model described by Equation (5.5) can be submitted to this package in a straight forward manner. However, in dimensions > 3, the problems become computationally demanding. We consider here the case of $2d$.

Consider again the model of the L-J ellipsoid problem of Equation (5.5). Recall that in this model the unknowns are A and b. The data is the set of points x_i, $i = 1, ..., m$, $m = 50$, randomly generated within interval [20, 30] in 2-dimensions. The resulting instance has been successfully solved with CVX. The output is represented in Figure 5.1 showing the minimum ellipsoid enclosing perfectly the convex hull of the given set of points in $2d$.

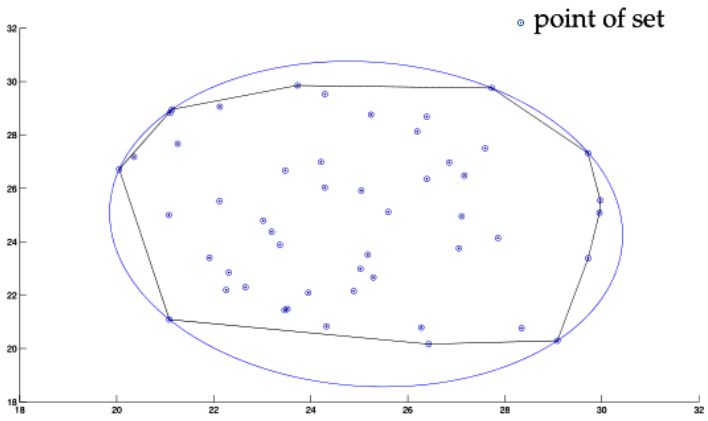

Figure 5.1: *Minimum ellipsoid generated by the infeasible path-following approach*

5.3 Computing MVEE using GA

As mentioned earlier, exact approaches to MVEE are not viable in high dimensions and large cardinality sets. Approximate approaches are therefore called upon. Here, we consider an evolutionary approach namely the Genetic Algorithm or GA. Moreover, we intend to implement and apply GA to the MVEE problem in an innovative way which is different from applying existing implementations of GA found in the Matlab Optimisation Toolbox, for instance.

5.3.1 GA Implementation and application to MVEE

There are a number of ways to implement GA. They differer mainly in the way that individuals are represented. Some representations are more suitable than others in terms of ease of coding and computing time.

5.3.1.1 Implementation 1: handling the MVEE model (5.5)

Consider again model (5.5). In $2d$, A is a 2×2 matrix corresponding to four variables and b is a 2×1 column corresponding to two variables. Given the points to enclose by an ellipsoid of minimum volume. A solution to model (5.5) can be represented as a 1-dimensional array with as many entries as matrix A and vector b, i.e. $n^2 + n$, or 6 in $2d$. In other words, chromosomes of length 6 will have to be generated to form a population of

individuals on which the GA will work. The fitness function is no other than the objective function of model (5.5). Solutions must satisfy the constraints of this model too. The solver minimizes the objective function subject to the constraints until the minimum volume ellipsoid enclosing the points is found.

In the candidate to optimum solution, the first four components belong to A and the remainder belong to b which is the center of the ellipsoid. From this solution, the parameters of the MVEE are extracted as follows. Suppose all the eigenvalues of A are positive real numbers. The length of the major axis is $\sqrt{\Theta}$ where Θ is the largest eigenvalue of A, while the minor axis is $\sqrt{\theta}$ with θ being the smallest eigenvalue of A. The solution found by this implementation of GA is not quite as good as one would like it to be since it is only an approximation. The instance solved with the exact approach has also been solved with this implementation of GA.

5.3.1.2 Implementation 2: an alternative solution representation

Let C be a set of points in some dimension to be included in a L-J ellipsoid. To implement GA on this problem, we need an appropriate representation of a solution as the ellipsoid itself rather than the entries of matrix A and vector b of model (5.5). A fitness function and genetic operators: crossover, mutation, and reproduction. A stopping criterion is also needed. These components of GA are described below.

Solution representation

A solution is an ellipsoid which is represented as a chromosome depicted in Table 5.1. It comprises the following genes:

- The coordinates (c_1, c_2) of a point representing the centre of the ellipsoid;
- The length a of the major axis of the ellipsoid;
- The length b of the minor axis of the ellipsoid;
- The angle θ of tilt of the major axis with respect to the x-axis.

Genetic operators

Starting from an initial randomly generated population of ellipsoids (solutions), subsequent

Table 5.1: *Chromosome representation of an ellipsoid*

major axis	minor axis	centre		angle of tilt
a	b	$c1$	c2	θ

populations are produced using genetic operators of crossover, mutation and reproduction.

Crossover operator

There is a number of possible implementations of this operator. The constant theme running through them all is that they operate on two parents to produce two children. Here, the parents are ellipsoids as represented in Table 5.1. The crossover produces two new children with potentially different axes, centres and angles of tilt. The Intermediate operator is used here. The gene values of the child are selected somewhere between and around the gene values of the parents. The children are created according to the formula:

$$Child1 = Parent1 + \beta\,(Parent2 - Parent1) \tag{5.10}$$

in which β is a scaling factor selected uniformly randomly in $[-d,\ 1 + d]$. The value of the parameter d defines the size of the area for possible children, where $d = 0.25$, this value ensures that the variable area of the children is the same as the variable area spanned by the variables of the parents. Each gene in the new child is the output of combining the genes of the parents according to Equation (5.10) with the new β selected for each gene [154]. Figure 5.2 illustrates that; it represents the two children with Child2 containing all the points. Note that crossover helps cover the search space in the process of optimisation.

Mutation operator

The mutation operator enables GA to avoid being trapped in local optima. This is implemented by changing one or more genes. Gaussian mutation is the type of mutation implemented here. It has been introduced in Evolution Strategy by Rechenberg in 1973, [155], where a Gaussian element is added to the individual or parent to generate the new offspring. This number is taken from a Gaussian distribution with mean value 0. Gaussian mutation can control the variance of the population fitness of each generation according

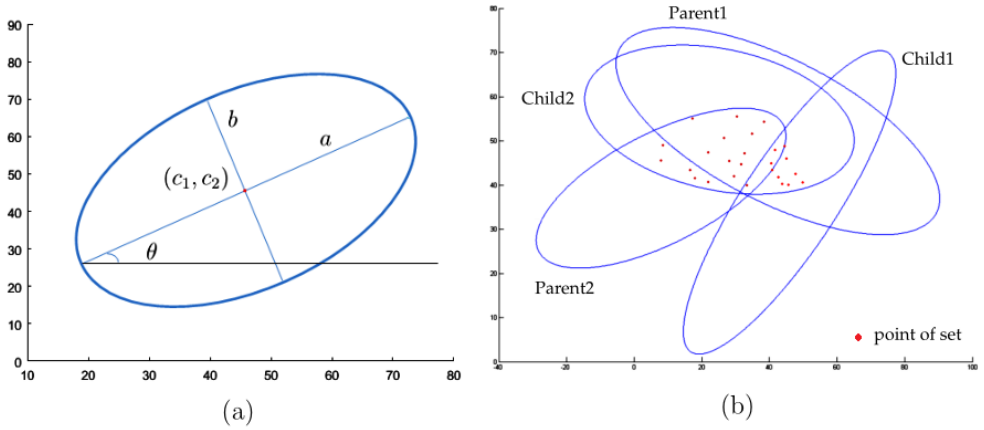

Figure 5.2: *(a) An illustration of a chromosome; (b) Breeding new children from parents*

to [156].

Reproduction

This operator copies some fit individuals into the next generation without any modification.

Fitness function

The fitness function is the measure by which individuals/solutions are ranked. Here, it is the difference between the volume of the ellipsoid and the number of points it encloses as given below. Minimizing this expression provides the desired minimal ellipsoid containing the set of points. The fitness is written as follows. Let ϕ_E be the fitness of ellipsoid E, C the set of points to enclose, C_E the set of points that ellipsoid E contains, V_E the volume of ellipsoid E, and α a constant, such that $\alpha \geq |C|^2$, here $\alpha = 100$. In this fitness we take the difference between the volume and the number of points, to make this possible, we multiply the number of points $|C_E|$ and the value $(|C| - |C_E| + \alpha)$ by a constant $\beta = 1$ to convert them into volume value.

$$
\begin{array}{l}
\textit{if} \qquad (|C_E| < |C|) \qquad \textit{then} \\
\qquad\qquad \phi_E = (\beta\,(|C| - |C_E| + \alpha) + V_E)^2 \\
\textit{else} \\
\qquad\qquad \phi_E = V_E - \beta\,|C_E| \\
\textit{end if}
\end{array}
\tag{5.11}
$$

Table 5.2: *An example of different cases of ellipsoids*

E	V_E	$\|C_E\|$	$V_E - \|C_E\|$
E_5	1	2	-1
E_1	6	5	1
E_2	14	10	4
E_3	6	0	6
E_4	20	10	10

Note that randomly generated ellipsoids may contain all the points while others may contain only some or none. The ellipsoids that contain all the points may have different volumes. According to the GA strategy, individuals with better fitness, i.e. a relatively small volume and a large proportion of points enclosed, are likely to survive into the next generation. This means ellipsoids containing no points of C are unlikely to survive. Those with all points and large volumes will not have a good fitness value in contrast to those with small volumes and all points in. The latter are likely to survive.

To illustrate how the fitness works, suppose we have five ellipsoids with different volumes and aim to enclose a set of points $|C| = 10$. Five different cases may arise, see Table 5.2. It is clear that the best ellipsoids that contain C are E_2 and E_4, but if we sort the fitness values say in ascending order, the values become $[-1, 1, 4, 6, 10]$ for E_5, E_1, E_2, E_3, E_4, respectively. This produces E_5 and E_1 as the best individuals that may be selected by the selection operator to produce new children and gives E_3 better than E_4. This inconsistency between the fitness values leads to an ineffective selection which will often miss the best individuals. Therefore, any ellipsoid has $|C_E| < |C|$, we compute its fitness value using this function $\phi_E = (\beta(|C| - |C_E| + \alpha) + V_E)^2$ to make its fitness very large in comparison to those ellipsoids that contain all the points. This will give E_1, E_3 and E_5 much worse fitness values than those of E_2 and E_4, resulting into the following new ranking $[4, 10, 11881, 12321, 13456]$ for E_2, E_4, E_5, E_1, E_3, respectively. This gives E_2, E_4, the chance to survive into the next generation.

Stopping criteria

Two stopping criteria are used here; one is the maximum number of generations, the other is the number of still generations, i.e. the number of generations with no progress in the

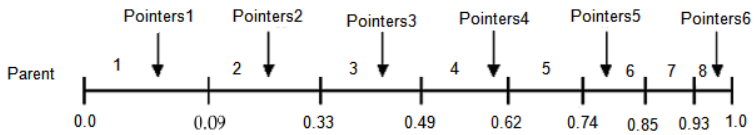

Figure 5.3: *The Stochastic uniform selection*

objective function, or fitness function value for a sequence of consecutive generations of length stall generations.

Selection procedure

It selects parents for the next generation based on their fitness values. The Stochastic uniform selection used here, is also known as the Stochastic Universal Sampling (SUS) which is an improvement on the Roulette Wheel selection, [157]. It is a single phase sampling procedure with zero bias and minimum spread. While the Roulette wheel selects several parents from the population by repeated random sampling, SUS uses a single random value to sample all of the parents by selecting them at evenly spaced intervals. The parents are mapped to adjacent sections of a line. Each parent's section is equal in size to its fitness value. Let N be the number of selections needed, an equally spaced pointers N are placed over the line to select these parents. The distance between the pointers are $1/N$ and the location of the first pointer is provided by a uniformly randomly generated number in $[0, 1/N]$, [158]. See Figure 5.3.

Algorithm 8: GA for MVEE

1 **Input** C the set of random points to be enclosed;
2 Input η the rate of crossover and ω the rate of mutation;
3 Generate a random population of ellipsoids E;
4 Evaluate ϕ the fitness of all ellipsoids in E using Equation (5.11);
5 Rank individuals according to their fitness;
6 Select parents from the population according to (SUS) selection;
7 Generate a new population by applying the following operators: crossover (Intermediate), mutation (Gaussian), and reproduction with their particular rates;
8 Compute the fitness of the individuals of the new population;
9 **Until** (The sopping criteria are satisfied) Repeat from 5;
10 **Return** Best solution.

5.3.2 GA for MVEE: Experimental investigation

To choose the most suitable crossover and mutation operators we proceeded to some experimentation as explained below. Some of these operators, and selections that are tested in these experiments are:
Scattered operator which creates a random binary chromosome. It then selects the genes where the chromosome is a 1 from the first parent, and the genes where the chromosome is a 0 from the second parent, and combines the genes to form the child [159]. Heuristic operator which creates children that randomly lie on the line including the two parents, a small distance away from the parent with better fitness value, in the direction away from the parent with the worse fitness value [160]. Arithmetic operator which creates children that are a random arithmetic mean of parents, uniformly on the line between the parents [160]. Uniform mutation is a two-step process. First, the algorithm chooses a fraction of the chromosome entries of an individual for mutation, where each entry has the same probability as the mutation rate of being mutated. In second step, the algorithm replaced entry by a random selected uniformly from the range for the entry [160]. Adaptive feasible mutation randomly generates directions that are adaptive with respect to the last successful or unsuccessful generation [161]. Remainder selection assigns parents deterministically from the integer part of each individuals scaled value then uses roulette selection on the remaining fractional part [162]. Uniform select parents at random from a uniform distribution using the expectations and number of parents [163]. Tournament chooses each parent by selecting individuals at random, the number of which you can specify by tournament size, and then choosing the best individual out of that best set to be a parent [164].

- We applied five types of crossover operators (*Single point*, *Two point*, *Intermediate*, *Scattered*, and *Heuristic*) to different sets of points in $2d$ and $3d$ for the same set of points and initial population. The results show that the Intermediate operator produces the smallest MVEE for most of the cases, see Table 5.3. After choosing the right operator, we tested it with different rates, we set it to the values (0.2, 0.4, 0.6, 0.8), and chose 0.8 which gives best solutions. Note that we tested each type of crossovers with all types of mutations, when we found that Gaussian mutation gave best result

for most of them we used it to compare between them.

Table 5.3: *Crossover operators*

Dim	*Points*	*Single point*	*Two point*	*Intermediate*	*Scattered*	*Heuristic*
2	10	76.24	70.14	52.89	80.25	52.83
.	20	153.12	145.77	112.50	137.77	112.31
.	50	142.90	119.11	93.81	122.75	122.75
.	60	180.81	167.16	137.51	164.99	133.727
3	10	175.78	146.09	55.31	204.82	123.798
.	20	144.23	228.85	95.142	184.66	113.42
.	30	295.89	230.70	141.15	222.71	141.82
.	50	607.35	361.63	212.11	374.80	276.93

- We applied three types of mutation (*Uniform*, *Gaussian*, and *Adaptivefeasible*). We applied them to different sets of points and selected the best. Table 5.4 shows that Gaussian mutation produces the smallest MVEE for most of the cases. Also Figure 5.4 shows the diversity of of the population, it is clear that Gaussian mutation is the best. Note that relying on the result of Table 5.3 we fixed the Intermediate operator to compere between the mutation operators.

Table 5.4: *Mutation operators*

Dim	*Points*	*Uniform*	*Gaussian*	*Adaptivefeasible*
2	10	201.32	79.61	26.73
.	20	145.39	87.74	138.63
.	50	160.35	112.60	112.83
3	20	466.79	166.16	304.66
.	30	400.57	138.09	213.05
.	50	680.65	211.00	299.47

In the same way, five types of selections (*Stochastic uniform*, *Remainder*, *Uniform*, *Roulette*, and *Tournament*) are tested. Table 5.5 shows that the Stochastic uniform selection, Remainder, and Roulette wheel are the best. They almost have the same results so we chose the Stochastic uniform selection.

Based on these results we applied Algorithm 8 with the Intermediate crossover and the Gaussian mutation. SUS was adopted for selection procedure. Note that in the implementation, when crossover is applied, it is followed by mutation on the best individual generated to improve the result.

Table 5.5: *Selection functions*

Dim	*Points*	*Stochastic uniform*	*Remainder*	*Uniform*	*Roulette*	*Tournament*
2	10	92.01	93.24	144.09	93.06	284.46
.	20	61.11	59.96	91.02	61.06	174.87
.	50	105.738	105.42	159.78	106.29	321.65
3	10	102.84	102.05	677.50	101.272	758.564
.	20	135.15	133.64	721.88	137.29	888.97
.	50	161.09	159.678	787.21	169.16	682.35

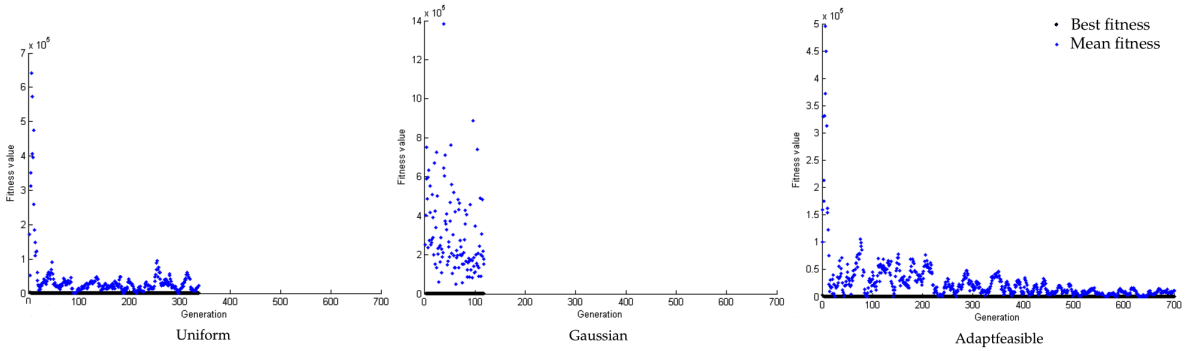

Figure 5.4: *Mutation operators*

5.3.2.1 In 2- and 3-Dimensions

We considered the same 50 points in $2d$ that are used in the exact approach of Section 5.2.3. To find the L-J ellipsoid containing them with GA, we first randomly generated a population of 100 ellipsoids, 200 randomly generated points in the plane; every pair of points representing an ellipsoid. The rates of crossover and mutation are set to 0.8, and 0.1, respectively. This means that the rate of reproduction is 0.1. After 51 generations, an approximate L-J ellipsoid is obtained. Figure 5.5 represents three different stages of the GA search process. Figure (a) shows the initial population of ellipsoids. Figure (b) shows, overlapping, the best ellipsoids found in generations 10, 20, 30 and 40 generations. Figure (c) shows the final solution after 51 generations. The same idea is applied to a set of points in $3d$, with GA using a population of 70 ellipsoids. The problem is solved after 120 generations. Figure 5.6 illustrates this.

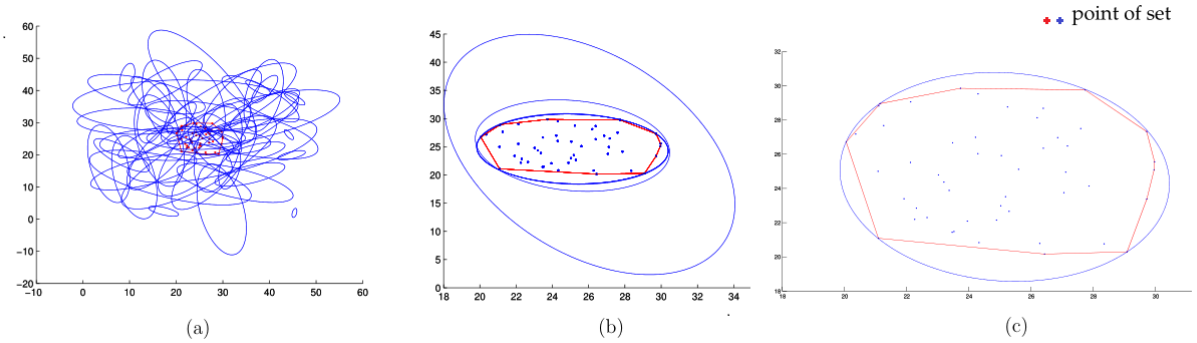

Figure 5.5: *(a) 100 Ellipsoids in 2d; (b) Ellipsoids of intermediary generations; (c) Löwner-John ellipsoid found in generation 51*

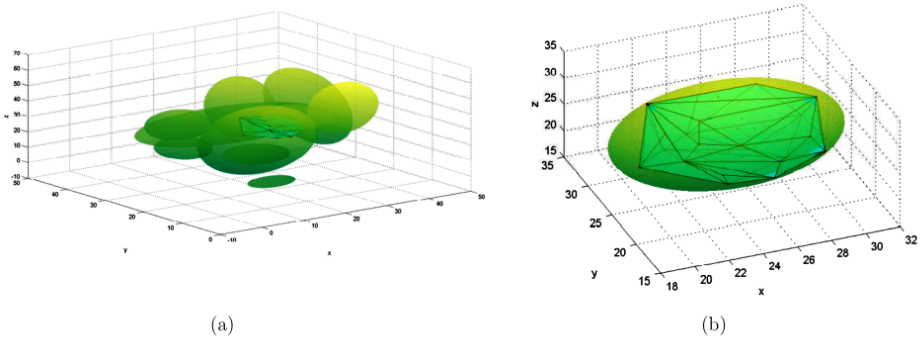

Figure 5.6: *(a) Initial population of ellipsoids in 3d; (b) MVEE in 3d*

5.3.2.2 Alleviating the computational burden

Checking the inclusion of large numbers of points in a given ellipsoid is time consuming even in low dimensions. To make the GA approach viable in practice and higher dimensions, it is necessary to reduce this computational burden. It is clear, for instance that, if the convex hull of the given set of points is known, then only its vertices need enclosing. Unfortunately computing the convex hull is harder than finding L-J ellipsoids/MVEE's. However, this suggests that only the peripheral points of the set need to be known, at most $(n^2 + 3n)/2$ points, [11]; they are the ones that need enclosing; those inside would belong necessarily, if the peripheral ones belong. This may reduce the computational load. Here, we outline three different ways to reduce the number of points to consider for inclusion without compromising the inclusion of the rest of the points of set C. They are:

1. The convex hull approach which considers only the vertices of the convex hull of C;

2. The boundary approach which considers only the boundary points of set C;

3. The boarder approach which considers the points in a band of arbitrary width ϵ that is between two concentric hypercubes the largest being the smallest containing C.

As said earlier, the convex hull approach is expensive; so is the boundary approach which identifies all points on the boundary of C. The boarder approach which we introduce here, on the other hand, is a rough procedure to find the peripheral or border points of the set that need enclosing. It is computationally cheap. This procedure is as follows:

1. Find points with minimum and maximum coordinates over all axis;

2. Choose the width ϵ of the border that we want to impose. Such a border could be defined by inequalities

$$\min_j \quad x_j + \epsilon \leq x_i \leq \max_j \; x_j - \epsilon, \qquad i = 1, ..., n \tag{5.12}$$

The number of points in the border depends on its width ϵ. For example 1000 points form the interval $[40, 50]^2$ with $\epsilon = 0.1$ leads to about 40 points to check for inclusion.

Note that ϵ should neither be small such that a lot of points may miss nor very large such that the amount of points to check is high. We do not have a formula for ϵ because this value produces number points that relies on the distribution structure of the points within an interval. For an interval, the relation between the value of ϵ and the dimensions or with large number of points is a reverse relationship in which the ϵ value gradually turns small in higher dimension or with large number of points.

A comparison between these three approaches is provided in Table 5.6. The convex hull procedure (Columns 3 and 4) comes from Matlab and handles points up to $9d$. The boundary points are found using a Matlab Tool box function which only works in spaces for points up to $3d$, (columns 5 and 6). The border points approach we introduce here performs best, (Columns 7, 8, and 9).

Over all, the total time for finding the MVEE depends on the number of points, the number of generations, and the population size. Reducing any of these reduces computing time. The border method does not guarantee the existence of all the points inside the final ellipsoid. Although, the ellipsoid is enclosed the border points, some points from the set

Table 5.6: *Comparison of Convex Hull, Boundary, and the Border points approaches in 2 to 9d*

Dim	*Points*	*ConvHull approach*	*Time(s)*	*Boundary approach*	*Time(s)*	*Border approch*	*Time(s)*	ϵ
2	50	12	0.057	15	0.080	16	0.059	0.6
2	100	9	0.050	13	0.076	14	0.061	0.3
2	1000	17	0.062	43	0.094	40	0.098	0.1
2	10000	28	0.082	165	0.255	91	0.081	0.02
3	1000	61	0.059	220	0.120	86	0.068	0.6
4	1000	165	0.74	*	*	159	0.044	0.1
5	1000	298	0.115	*	*	116	0.062	0.1
6	1000	461	0.542	*	*	510	0.045	0.28
7	1000	633	5.494	*	*	637	0.047	0.35
8	1000	778	58.722	*	*	768	0.062	0.4
9	1000	*	*	*	*	877	0.062	0.48

(∗) solution not found

are missed outside the ellipsoid. So we need to add these points to the border points and enclose them again, see Figure 5.7. Another important issue is that the algorithm can be run well for a set of points of a positive or negative numbers (in quadrant 1 or 3). For a set of points C with different signs, the algorithm works well in $2,3d$, but with small intervals of points. In large intervals, generating initial ellipsoids to cover it may produce centres with location near zero. When the population evolves, the major and minor axes may have these values which give small ellipsoids. So we need to generate another set of ellipsoids having centres with large values in addition to those ones that cover C. The extra ellipsoids may not cover C but their centres values are higher than those ellipsoids near the center. They may help the GA to produce a solution with large parameters enough to enclose C.

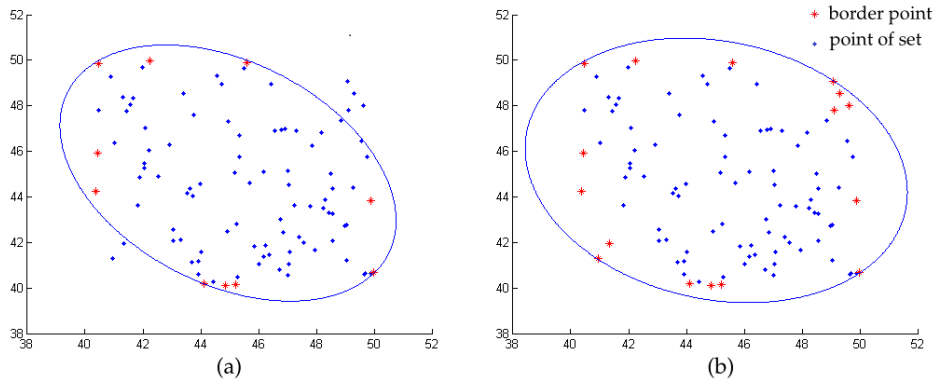

Figure 5.7: *Using the border approach to check the points. (a) Implementation 1; (b) Implementation 2*

5.3.2.3 Computing MVEE in high dimensions

Computing MVEE in high dimensions ($n \geq 4$) is computationally challenging. The approach is the same as in lower dimensions. Here the initial population of the ellipsoids should be varied enough to include large ellipsoids that enclose all the points. Experiments have been carried out. The results recorded in Table 5.7 show the number of points and the total computing time in seconds taken to solve different instances in different dimensions. The population size is restricted to be between 70 and 100. We compare these results for the same data sets with those from the path-following infeasible algorithm. The first two columns represent the dimensions and the sets of points. Columns 3 to 7 show the results from the GA algorithm, the number of points on the border that are used to check, the number of generations, the factor ϵ, the volume, and the time. The rest of the table shows the results from the exact algorithm with the difference between the volumes. To reduce the computing time, the border strategy is used. We compare volume and time; although, the difference between the volumes of the two methods tends to be more with increasing the dimension, the exact algorithm cannot find the solution for large data sets for dimensions $> 20d$. An argument may arise that the border strategy can be used in the exact method too. However, for 60-100 dimensions with the same border points used in the GA tests, the exact algorithm could not find the solutions.

The relationship between ϵ and the number of points and the dimensions is opposite. This factor must be small when increasing the dimension and number points. Although, the ellipsoid has enclosed the border points, some points are missed and lie outside the ellipsoid. So we need to add these points to the border points and enclose them again. We ran the algorithm at most five times and chose the best volume and time.

5.3.3 Special case

The majority of applications of the MVEE consider the perfect cases where the data have no outliers. If the set has outliers, the formulation of the MVEE problem should take that into consideration. With respect to our algorithm when the set has some outliers, the interval that is used to generate the ellipsoids should be large enough to cover all the points

Table 5.7: *Performance of GA for MVEE and the exact approach*

Algorithm		*Genetic Algorithm*					*Path – Following Algorithm*		
Dim	*Points*	*Border*	*Generations*	ϵ	*Volume*	*Time*(s)	*Volume*	*Time*(s)	*Error*(%)
2	10000	151	100	0.04	155.448	9.60	154.863	195.23	0.3777
3	.	177	70	0.015	312.540	23.87	295.851	272.81	5.641
10	.	943	70	0.023	2.3054e+10	185.91	2.1482e+08	60.85	1.0631e+04
20	.	1546	100	0.02	1.8120e+21	600.59	2.5119e+16	304.82	7.2136e+06
30	.	330	100	0.002	2.6415e+34	403.56	*	*	*
60	.	823	100	0.003	2.2161e+85	1987.03	*	*	*
100	.	465	100	0.0009	8.7299e+147	988.90	*	*	*
2	100000	414	100	0.01	156.719	23.38	*	*	*
3	.	942	70	0.0073	327.155	63.98	*	*	*
10	.	410	300	0.002	1.1588e+14	646.60	*	*	*
20	.	1515	100	0.05	7.4049e+21	1194.01	*	*	*
30	.	1295	70	0.001	4.9523e+35	252.76	*	*	*
60	.	1270	100	0.00049	1.8894e+85	1437.10	*	*	*
100	.	1848	100	0.0004	7.0457e+147	4866.48	*	*	*
2	500000	510	70	0.0025	157.415	23.47	*	*	*
3	.	1169	70	0.002	337.205	84.23	*	*	*
10	.	1355	300	0.00065	7.5971e+10	537.26	*	*	*
20	.	3902	100	0.00095	4.9726e+21	3111.63	*	*	*
30	.	1954	100	0.0003	1.3715e+34	4527.24	*	*	*
60	.	3136	100	0.00024	1.7157e+85	3447.41	*	*	*
100	.	2151	100	0.0001	3.6392e+148	4044.21	*	*	*
2	1000000	307	100	0.0006	160.557	34.38	*	*	*
3	.	382	170	0.0003	558.389	144.24	*	*	*
10	.	425	200	0.0002	1.7173e+15	1538.54	*	*	*
20	.	2882	100	0.00035	2.7976e+23	931.16	*	*	*
30	.	5274	100	0.00044	1.9077e+39	6125.83	*	*	*
60	.	5362	100	0.00022	2.1627e+85	9992.16	*	*	*
100	.	2139	100	0.00005	1.6744e+147	3818.56	*	*	*

(∗) solution not found

including the outliers. This is guaranteed by generating large ellipsoids. See Figure 5.8.

A set of points with outliers and how to handle them is an essential problem. In the area of data mining and robust statistic, efficiently detecting the outliers is a challenge that has concerned researchers. The Minimum Volume Ellipsoid (MVE) is an alternative approach that is used as the basis for trimming outliers. It can tolerate the existence of a high fraction of arbitrary sever outliers. The MVE estimator is an essential tool in robust regression and outlier detection in statistics. It is a robust estimator of covariance structure and location. It has a maximum breakdown point that can reach 50% of the points set. This means about half the set can be contaminated without affecting the estimate. Nevertheless, using this estimator is restricted because of the lack of an efficient computational procedures.

The MVEE is known for its affine equivariance and positive breakdown properties as a

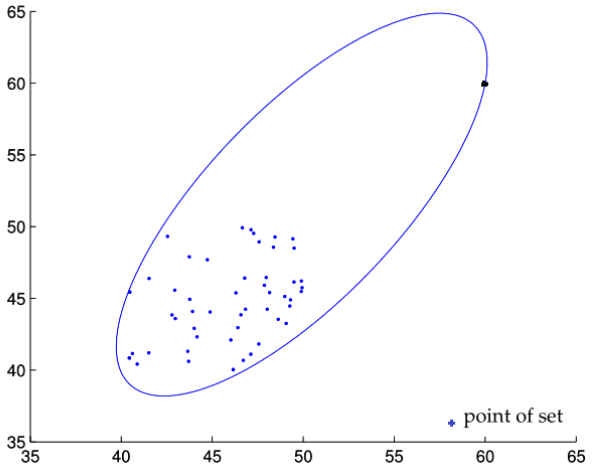

Figure 5.8: *A set with outlier point*

multivariate location and scatter estimator [150]. The outliers can be detected immediately if the MVEE is computed, because the outliers lies on the boundary of the MVEE. The following section will cover this problem.

5.4 The Minimum Volume Ellipsoid Estimator Problem

Let K be a subset of C, such that K contains k points, $1 \leq k < m$. For an arbitrary set of distinct points $C = \{x_i\}_{i=1}^{m}$ where $C \subseteq R^n$, the MVE estimator is the MVEE that contains at least k points in C. This problem can be written as [165, 166]

$$\begin{aligned} min \quad & log\ detA^{-1} \\ s.t \quad & |\{i \in I : (x_i - b)^T A^{-1} (x_i - b) \leq n\}| \geq k \end{aligned} \tag{5.13}$$

for $b \in R^n$, $A \succ 0$, and $I = \{1, ..., m\}$. When finding the MVEE, b is taken as the estimate of the location vector and A is taken as an estimate of the covariance matrix. Computing the exact MVE estimator is a difficult combinatorial problem as it involves determining a minimum of k best points to contain, where the value of k that gives the maximum breakdown is $k = [(m + n + 1)/2]$ (the brackets denote the greatest integer function). Note that, in a clustering context, a judicious choice of number k potentially makes the computation of the MVE estimator a potent tool for detecting outliers and other clusters.

Many approaches have been suggested to solve this problem. Cook [167], proposed an

enumeration algorithm that tests all subsets $K \subseteq C$. This approach becomes computationally prohibitive for large C. Agullo [168], introduced an improved exact technique for the MVE estimator. It is a variant of Branch-and-Bound algorithm that does not test all the subsets K. Although, this is substantially more efficient than that of [167], still it cannot address large size problems.

Hawkins, [169], introduced the Feasible Solution Algorithm (FSA) to find the MVE estimator. It relies on a steepest descent algorithm with a probability mechanism. It selects an initial random subset K of k points by dividing the given m point set into two sets K and $K^{'}$, such that $|K^{'}| = m-k$. Two points are swapped between the sets, the point that is moved to K is specified if it produces a subset with a small enclosing volume ellipsoid. This is repeated until no swap gives a smaller volume. To this end the subset K is the feasible set and the MVEE of K is the feasible solution.

Poston et al. [170] introduced the Effective Independence Distribution (EID) procedure. It minimizes determinants of matrices containing the data to choose the right subset to use in the estimator of the MVE. This technique is deterministic, fast and gives reproducible estimates of shape and location of a set of points. However, it does not guarantee an exact MVE. But Poston et al. showed that it selects subsets which generate ellipsoids close to the true MVE.

Ahipaşaoğlu [165], provided a 2-Exchange procedure which is a local search method. Its advantage is that it can solve large instances of the problem and produces good solutions. The method includes three steps; (*i*) choose an initial feasible solution and calculate its value, (*ii*) choose an exchange to enhance the objective function, (*iii*) implement the exchange. The algorithm keeps choosing points and implementing exchanges until no other point exists that can produce a better solution. Although, this method is the same as of [169]; however, it uses new bounds that have not been suggested before. Other methods have also been suggested in [166, 171].

The problem of MVE estimator can be solved using an evolutionary approach which is also different from the methods that are mentioned above. It requirers a suitable representation of the solutions, a fitness function that distinguishes the outliers from the set as well as suitable genetic operators and stopping criteria.

5.4.1 A GA approach to the MVE estimator problem

The aim is to detect the subset $K \subseteq C$ enclosed by a MVEE. Let $C \subseteq R^n$ be a set of points randomly generated from a multivariate normal distribution. Then the mean of 30% of the points is shifted up by 7 units. These data sets are similar to those in [165, 172]. To apply the GA, we choose a suitable representation of a chromosome, i.e. a subset $K \subseteq C$. A suitable fitness function and genetic operators are described below.

Solution representation

Here, a solution is represented as an array of indices from the subset K. For example, if $K \subseteq C$, $|C| = 20$, and $|K| = 11$, then a solution is any combination of 11 indices of C. Table 5.8 is such a solution.

Table 5.8: *Chromosome representation ($|K| = 11$ is chosen)*

6	7	8	9	10	11	12	13	14	15	16

Genetic Operators

We start from an initial randomly generated population of subsets $K_i \subseteq C$; new populations are subsequently produced using some genetic operators.

Crossover operator

One-point crossover operator is the type used here. Suppose we have two parents K_1, and K_2 with different indexes. To produce two children we pick a single point on the chromosome which splits it into two strand. The same point is chosen on the second chromosome then replace the two pieces, see below. If some of the points are repeated, the validation of the chromosome will not be effected.

Table 5.9: *Parents*

Parent1	1	2	3	4	5	6	7	8	9	10	11
Parent2	10	11	12	13	14	15	16	17	18	19	20

Table 5.10: *Children*

Child1	1	2	3	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Child2	10	11	12	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

Mutation operator

One of the points, here its integer index, in the string to mutate is swapped with one point from the set C. If the point is within the subset K then the volume may become smaller, but if the point is one of the outliers or far from the border points in K, the volume increases and this chromosome is unlikely to make it into the next generation. For a subset K of 11 points where C is a set of 20 points the mutation is done as follows.

Table 5.11: *Chromosome mutation*

K_i	6	7	8	9	10	**11**	12	13	14	15	16

New K_i after mutation

$New\,K_i$	6	7	8	9	10	**20**	12	13	14	15	16

Fitness function

The volume of the ellipsoid is used to find the subset K. When points are close enough to fit in a small volume ellipsoid, outliers can be identified. The MVEE of each K_i is computed by applying Algorithm 8 of Chapter 5.

Stopping

Stopping criteria is based on the maximum number of generations.

Selection procedure

The roulette wheel method is used here; the probability of selecting a chromosome is proportional to its fitness. Selection is biased towards chromosomes with high fitness.

The procedure has two steps: finding the smallest subset with at least $|K|$ points, then finding the MVEE that encloses this subset. We notice that

1. The length of the chromosome is very important, it should be larger than the expected

number d of the outliers say $d < l \leq m/2 + 1$. This because one of the chromosomes may have most of the outliers or some of them so the ellipsoid may enclose them with best volume. The larger the chromosome the better is the solution.

2. There is no need to run the algorithm for many generations. This because the crossover may destruct the strings in such a way many points can repeatedly appear in the string. This may let the ellipsoid enclose a few points only. Almost the same result can be gained from the first or after many generations, so for those sets that are created by [165] we prefer the first generation to reduce the time, otherwise we may increase the generations.

3. To maintain the chromosome, we need to repair it by arbitrary filling it with points from the set C. This will reduce the repetition.

4. Although we choose the number of points of K similar to those in [165], the MVEE encloses almost all the points of C except the outliers and some missing points. Figure 5.9 shows that. The algorithm is described in Algorithm 9.

Note that, it is important to specify the interval of the initial ellipsoids such that the subsets wanted to enclose are covered.

Algorithm 9: GA for MVE

1 **Input** C the set of points from a random multivariate normal distribution;
2 Input η the rate of crossover and ω the rate of mutation;
3 Generate a random population of subsets of size $K, |K| < |C|$;
4 Evaluate the volume of all enclosing ellipsoids of the subsets K using Algorithm 8;
5 Rank individuals according to their fitness;
6 Select parents from the population according to some selection procedure;
7 Generate a new population by applying the following operators: crossover, mutation, and reproduction with their particular rates;
8 Compute the fitness of the individuals of the new population;
9 **Until** (The stopping criteria are satisfied) Repeat from 5;
10 **Return** Best subset K enclosed by MVEE.

5.4.2 GA for MVE: Computational results

Computational results are provided in Tables 5.12 and 5.13 for small and large sets of points. Computing times are in minutes. It is clear that GA is more efficient than the algorithm of Ahipaşaoğlu [165]. Moreover, it can handle larger sets of points in higher dimensions (> 20). Also the MVEE can enclose larger subsets K. Our heuristic provides optimal or near optimal solutions in less time. Tables 5.12 and 5.13 show in columns 3, K the size of the subsets to be enclosed in ellipsoids generated with the 2-Exchange algorithm while columns 4 of the tables shows the number of points enclosed by the MVEE, we refer to it as (Enclosing points).

The experiments were carried out on two computers. The problems in $2d$ up to $50d$ for (20-1000) points were on a Computer with a 3.33 GHz Intel Core 2 processor and 4 GB memory. All experiments on instances in $100d$ and the set of 1000 points in $50d$ were implemented on a computer with 2.6 GHz (32 processor with 8 cores each, and each processor can run 16 processors with 100% CPU usage (hyperthreading) and 128 GB memory. The problems with dimensions $n \leq 20$ that are tested by the 2-Exchange algorithm were carried out on a Macbook Air with a 1.7 GHz Intel Core i7 processor and 8 GB memory.

Table 5.12: *Performance of GA and the 2-Exchange heuristic on MVE: Small datasets*

Dim	*Points*	*K*	*Enclosing points*	*Time* $GA(mns)$	*Time* $2-EX(mns)$
2	20	12	14	0.29	0.99
2	30	17	19	0.27	1.55
3	20	11	14	0.27	1.91
3	30	16	18	0.28	3.28
5	20	13	14	1.91	2.95
5	30	18	21	2.04	5.47

We also applied this approach to different datasets taken from Rousseeuw and Leroy [166]. We chose eight datasets that are tested by [167, 170]. They used an algorithm introduced by Titterington [173] in their experiments, which exploits the finite support property of the ellipsoids. To check their exact calculations, they compared the results with

Table 5.13: *Performance of GA and the 2-Exchange heuristic on MVE: Large datasets*

Dim	*Points*	*K*	*Enclosing points*	*Time*(*mns*) *GA*	*Time*(*mns*) $2-EX$
10	100	56	70	2.20	82.49
10	200	106	139	2.93	303.07
10	500	256	347	5.27	1663.19
10	1000	511	699	9.52	*
20	100	61	70	3.55	132.63
20	200	111	137	5.79	500.42
20	500	261	349	10.97	3252.19
20	1000	551	699	17.60	*
30	100	61	70	4.93	*
30	200	111	139	8.49	*
30	500	261	349	18.63	*
30	1000	511	700	28.99	*
50	100	61	70	11.76	*
50	200	111	140	21.34	*
50	500	261	349	36.03	*
50	1000	511	700	18.87	*
100	100	53	68	11.78	*
100	200	111	140	64.38	*
100	500	261	350	75.16	*

(∗) solution not found

those results from the basic resampling method. The latter is a commonly used technology known as the basic resampling scheme, which is a convenient approximation. Here we tested these datasets using Algorithm 9, then we find the MVEE of the minimum subset K using Algorithm 8 and the path following infeasible algorithm, see Table 5.14. The number of the original data is in column 3, K is the number of points in the chromosome we used, and column 5 is all the points enclosed by the MVEE and the path following. The figures of these datasets show that the Algorithm 9 detects the outliers and finding the MVEE confirms them. The ellipsoids from Algorithm 8 and the path following method can cover at least half the data. The size of the volume proves that our approach can cover points for some sets more than those of [167]. For more details about the datasets see [166]. Figure 5.10 shows the results from the data in $2d$ and $3d$. Figure (a) is belongs to Heart dataset which contains 12 observations, our method detects 5 outliers , Figure (b) belongs to the Phosphorus dataset where 10 outliers are detected, and Figure (c) represents the Stackloss

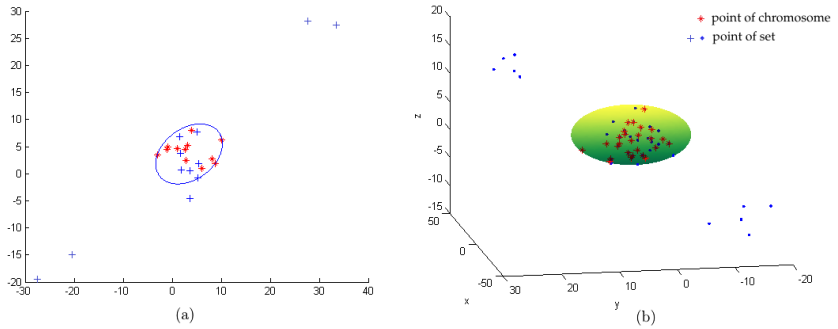

Figure 5.9: *(a) MVE estimator in 2d detecting four outliers, the subset K contains 12 points with red colour, but the ellipsoid contains more points leaves the outliers out and missing one point; (b) MVE estimator in 3d detecting ten outliers*

dataset with 4 outliers. The table also shows extra two sets from [166] that we tested, the Delivery dataset and the Salinity dataset. See Figure 5.11.

Table 5.14: *Performance of GA, Titterington's algorithm, and basic resampling method on the MVE estimator: Real datasets*

Dataset	*Dim*	*Points*	*K*	*Enclosing points*	*Volume GA*	*Volume Path following*	*Volume Titterington*	*Volume basic resampling*
Artifact	4	23	10	14	1.5271e+06	3.5527e+04	7.2510e+07	1.1450e+08
Coleman	5	20	9	13	1.7680e+06	8.0882e+03	137.7844	295.7331
Heart	2	12	6	8	84.8731	68.9244	36.2828	47.7693
Phosphorus	2	18	9	9	401.0587	389.5698	77.0726	83.6180
Stockloss	3	21	13	17	4194.4	1480.3	87.2167	165.5365
Wood gravity	5	20	9	16	3.6323e-04	0.4812e-06	7.78036551e-7	2.1707e-06
Delivery	2	25	10	13	3986	37183	*	*
Salinity	3	28	10	17	478.834	297.923	*	*

(*) solution not found

Also, this method is implemented to some real data sets from the Center for Machine Learning and Intelligent Systems, Bren School of Information and Computer Science, University of California, Irvine. See (https://archive.ics.uci.edu/ml/datasets.html). The dimension of the data ranged between $3 \leq d \leq 77$ and the number of points between $79 \leq m \leq 2167$, where the rate of the outliers is relatively high for some data. We chose an appropriate chromosome to detect as many outliers as possible. Table 5.15 provides some results, where column 3 is the number of points of the dataset, K is the number of points in the chromosome we used, and column 5 is the points enclosed by the MVEE. We do not know exactly the actual number of the outliers to assess our results.

The first three problems supply categorized listed executions of a number of simple

Table 5.15: *Performance of GA on the MVE estimator problem: Real datasets*

Dataset	*Dim*	*Points*	*K*	*Enclosing points*	*Outliers*	*Time(mns)*
Brush teeth	3	2167	445	2132	25	2.40
climb stairs	3	195	44	128	57	0.43
Descend stairs	3	594	160	454	130	1.86
Trajectory	8	163	30	152	11	1.56
Stone flakes	8	79	29	67	12	1.12
1 – *Gesture phase segmentation*	18	1747	221	1452	295	8.54
2 – *Gesture phase segmentation*	18	1264	318	1091	73	12.21
Dress sales	23	148	52	138	10	4.66
Metabolic relation	23	1000	15	745	255	1.79
Water treatment	37	527	16	479	48	3.33
Data cortex nuclear	77	1080	74	1056	24	36.86

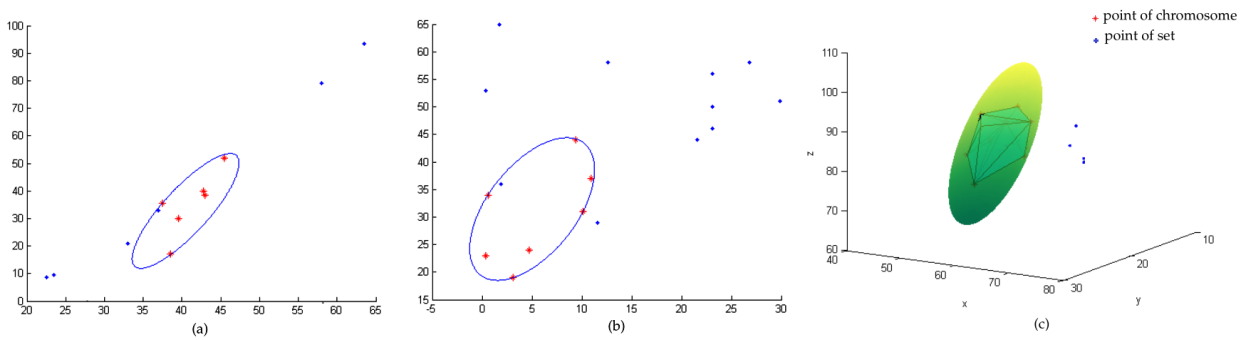

Figure 5.10: *(a) Heart dataset; (b) Phosphorus dataset; (c) Stackloss dataset*

human activities. Brushing teeth, climbing stairs, and descending stairs. Each of these data contains three columns. The fourth dataset is about tracking vehicles. The data of the stone flakes concerns stone tools that prehistoric men used. Datasets 6 and 7 contain a temporal segmentation of gestures. The remaining datasets are on metabolic relations, water treatment, and nuclear cortex dataset.

5.5 GA implementation for LVEE

Given a set of points C in R^n, a subset $P \in C$, $|P| = k$, is called a convex hole (or empty convex polygon, $k-$gon) in C, if its convex hull includes no points of C in its interior, [174]. Characterizing empty or nonempty convex polygons in a set of points is of interest in discrete and computational geometry.

In 1935, Erdős and Szekeres [175] posted this question: What is the smallest positive

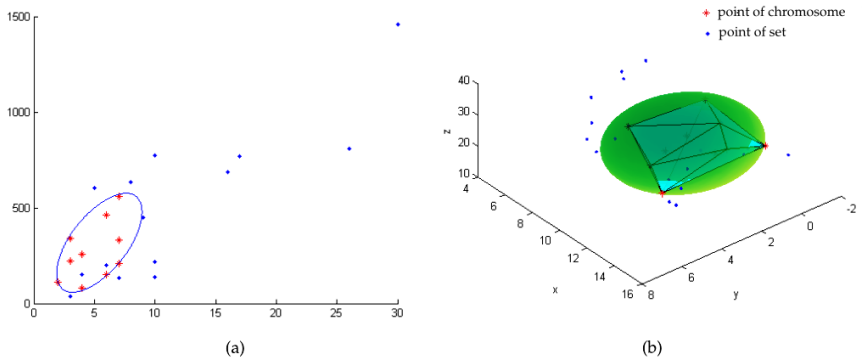

Figure 5.11: *(a) Delivery dataset; (b) Salinity dataset*

integer $g(k)$ for each positive integer k such that any set of points $g(k)$ in the plane contains at least one convex k-gon. The best known bounds for $g(k)$ are $2^{k-2} + 1 \leq g(k) \leq \binom{2k-5}{k-2} + 1$. The lower bound was found by Erdős and Szekeres [176] and the upper bound was proved by Tóth and Valtr [177]. The lower bound is sharp for $k \leq 5$ and is conjectured to be sharp for all k by Erdős and Szekeres [175, 176]. Later, Erdős [178] proposed the problem of finding the smallest integer $h(k)$, if it exists, such that any set P of at least $h(k)$ points in general position in the plane includes a convex polygon of k points whose interior does not include any point of P. It is easy to show that $h(k) = k$, for $k \leq 3$, and $h(4) = 5$. Nevertheless, calculating a formula of $h(k)$ and the exact number of $h(k)$ is very difficult. Harborth [179] showed that $h(5) = 10$, while Horton [180] proved that $h(k)$ does not exist for all $k \geq 7$.

In contrast to the MVEE and MVE problems, here, we are interested in finding the largest volume ellipsoid that does not cover any points, or LVEE. In other words, given a set of points C in R^n, the problem is to find the largest ellipsoid that can fit between the points without including any. It is important to add that ellipsoids which are all or partially outside the convex hull of the given points, do not qualify. Finding the LVEE of a finite set of points C is similar to computing the MVEE of the convex hull of those points. But here it is for the largest convex hole between the points.

Writing a model for this problem is difficult as we do not know which hole is the largest one to specify its vertices and maximize the ellipsoid inside. However, for a closed empty smooth shape the model can be written as a maximization of the L-J ellipsoid problem,

$$\begin{aligned} max \quad & log\ detA^{-1} \\ s.t \quad & ||Ax_i + b||_2 \leq 1,\ i = 1, 2, ..., m \end{aligned} \tag{5.14}$$

for the implicit constraint $A \succ 0$, and the variables $A \in S^n$ and $b \in R^n$.

To tackle this problem with GA, a suitable fitness function can be as in Equation (5.14). Let ϕ_E be the fitness of ellipsoid E, $|C|$ the size of the given set of points, $|C_E|$ the size of the set of points outside ellipsoid E, V_E the volume of ellipsoid E, α a constant, here $\alpha = 100$, LB the vector of lower bounds on all coordinates of points of C and UB the vector of upper bounds on all coordinates of points of C, and c the centre of ellipsoid E.

$$\begin{aligned} &if && (|C_E| < |C|) \quad then \\ & && \phi_E = (\beta\,(|C| - |C_E| + \alpha) + V_E)^2 \\ &elseif && LB_i < c_i < UB_i,\ \forall i \\ & && \phi_E = |C_E| - V_E \\ &else \\ & && \phi_E = 1/(\beta\,|C_E| - V_E) \\ &end\ if \end{aligned} \tag{5.15}$$

To reduce the computational burden, we generate the ellipsoids of the initial population having their centres within the lower and upper bounds of the coordinates of the points of set C. Note that, the ellipsoids which contain some or all of the points and ellipsoids that do not contain any of C do not compete well with empty and large ones. The final one is the maximum volume ellipsoid. Figure 5.12 illustrates that. The results for small sets of points in different dimensions is shown in Table 5.16. We believe that the LVEE can find the largest ellipse, rectangle, and triangle or any convex shape inside a polygon with closed smooth shape. When we find the LVEE, we determine the vertices of the convex hull of the hole.

Notice that the fitness function in Equation 5.15 is designed to work with random points from uniform distribution, it may not work for some special sets such a set that represent a shape of banana for example. Unfortunately, this algorithm with this fitness may not work

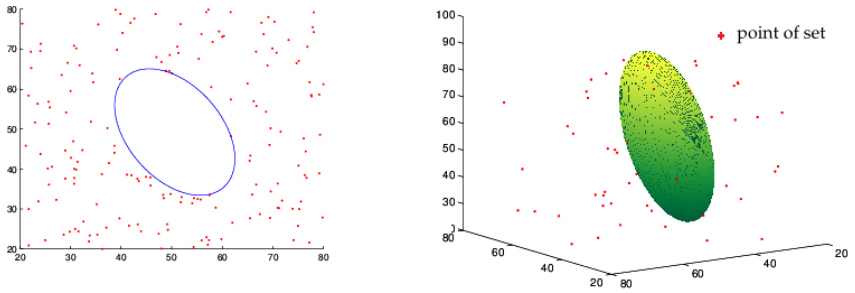

Figure 5.12: *Maximum volume ellipsoids embedded within points in 2d and 3d*

well for small set of points, see Figure 5.13. It may not inscribe three points or even stretch between the points if they are not in a well enclosed shape. So we propose using the MVEE to solve the LVEE. We use the same fitness function above with checking the end points of the axes of LVEE inside the MVEE. Instead of using the condition $LB_i < c_i < UB_i$, we use the condition $|End\ Points| == 4$ which means checking the 4 end points of the LVEE axes. This fitness keeps the LVEE between the points as much as possible. The procedure is as follows:

1. Solve the MVEE of the set,
2. In each generation; Find the LVEE say E_i, then find the end points of the major and the minor axes,
3. Check these points inside the MVEE, if they are inside then the E_i will pass to next generation.

We will support our investigation of the LVEE in high dimensions in a future work. Also, the idea can be improved to find all the holes in a set of points by fitting ellipsoids or any suitable shapes such as a circle. Thats by finding the holes one by one, in every time we find a hole, we fill it with points, so next ellipsoid will fit in another hole.

5.6 Can C1S be solved with enclosing ellipsoids?

A given $(0,1)-$matrix, can be seen as a set of points in a grid where the points represent the 1's. Their positions are recorded as coordinates in the plane. While the problem of MVEE

Table 5.16: *Efficiency of the approach to LVEE*

Dim	*Points*	*Generations*	*Initial Pop*	*Time(s)*
2	20	100	50	4.959
2	50	100	50	6.062
2	100	100	60	9.192
2	177	84	50	9.449
3	20	100	50	6.805
3	50	100	50	8.364
3	100	100	50	12.274
3	150	100	50	14.499
4	50	100	50	12.966
4	100	100	50	20.059
5	50	100	60	23.489
5	100	100	60	17.117
6	50	100	60	18.452
6	100	100	60	26.271
7	50	100	60	19.461
7	100	100	60	28.012
8	50	100	60	21.525
8	100	100	60	31.109
9	50	100	60	24.374
9	100	100	60	34.939
10	50	100	60	25.889
10	100	100	60	36.843

concerns static points, the C1S problem is a problem where we try to move the points in a certain way and then compute the MVEE containing them. It can be see in generic terms as the minimum volume ellipsoid for a dynamic set of points. We are looking for ellipsoids with many consecutive ones (i.e. blocks of 1's) inside. Those with a lower number of blocks and smaller volume are preferred. The procedure that is proposed for this application is as follows.

Procedure

The procedure that is used to minimize the number blocks is a multiple fitness technique. It ranks the ellipsoids according to the difference in the number of consecutive blocks and the volume and sort the fitness values in descending order. Ranking the population of the permutations in ascending order is according to the number of consecutive blocks. This

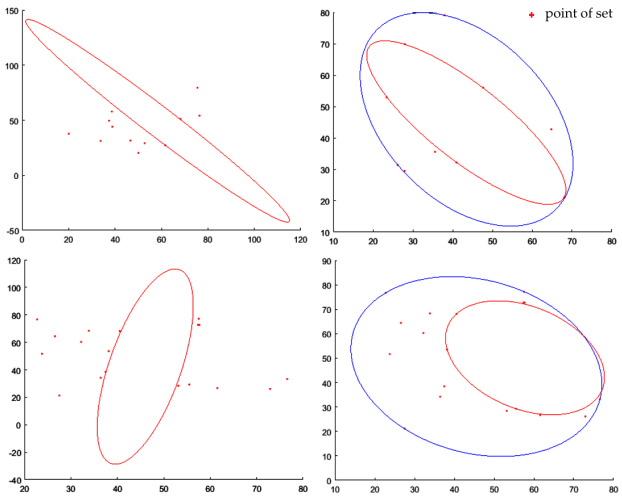

Figure 5.13: *The LVEE for sampling of 10 and 18 points using the MVEE procedure to the right and without it to the left*

algorithm uses the two fitness functions in tandem. Note that the ellipsoid helps minimize the number of blocks over all the generations by removing bad permutations. If we have two permutations with different numbers of blocks say 10 and 24 then obviously after many generations we may get the volume for the first one smaller than the latter. This makes the fittest individuals survive to the next generation. This procedure is called the enclosing ellipsoid algorithm for the C1S problem. Algorithm 10 gives its outline.

To illustrate, consider the following matrix which has 24 elements as blocks in rows; applying Algorithm 10 finds a permutation with 11 blocks of consecutive ones and 4 columns with C1P (count the columns from the left of the matrix), see Figure 5.14. CPU time is 2.468sec with population size 40 and 30 generations.

$$R = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

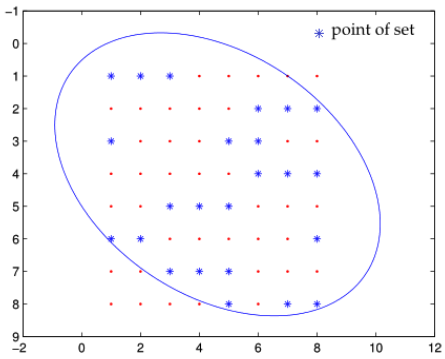

Figure 5.14: *Solving C1P and CBM problems via enclosing the ellipsoid*

The computational results of the C1P algorithm and Algorithm 10 are provided in Table 5.17. It is noticeable that the latter algorithm improves the results over all the table results which shows the effect of the fitness functions of the ellipsoids. The first four columns of the table are the initial information. The following four columns are the final number of blocks and their improvement with the number of columns with C1P and the time from the C1P algorithm. The final four columns are the results from Algorithm 10.

Table 5.17: *Computational results of the C1P algorithm and the enclosing ellipsoid algorithm*

Initial information				*C1P algorithm*				*Enclosing ellipsoid algorithm*			
Mat.	*Dens.*(%)	*Ge.*	*Initial blocks*	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time*(*s*)	*Final blocks*	*Blocks improv.*	*Nbcols C1P*	*Time*(*s*)
9	3	50	22	10	12	8	0.53	9	13	9	3.12
24	16	50	84	43	41	10	5.74	39	45	13	11.05
100	2	100	202	168	34	33	13.37	155	47	36	15.10
100	4	100	389	316	67	16	12.22	289	94	21	16.32
100	10	100	984	677	307	10	13.92	595	389	10	16.45
200	2	100	798	705	93	22	15.61	661	137	23	24.43
200	4	100	1492	1314	178	14	15.93	1241	252	15	25.66
200	10	100	3570	2972	598	8	15.79	2769	802	9	25.31
500	2	100	4887	4676	211	16	32.62	4542	345	18	46.02
500	4	100	9851	9279	572	12	34.12	8972	879	12	45.39
500	10	100	21759	19815	1944	8	34.76	19119	2640	8	46.53
1000	2	100	19611	19055	556	17	104.93	18749	862	16	121.34

Algorithm 10: Enclosing ellipsoid for a $(0,1)-$matrix with multiple fitness procedure

Input Given a $(0,1)-$matrix A;
Input η the rate of crossover and ω the rate of mutation;
Generate a random population of permutations m of columns of the matrix A;
Generate a random population of m ellipsoids;
Evaluate the fitness of the population of permutations according to FF5 function;
Evaluate fitness of the population of ellipsoids using Equation (5.11);
Rank the permutations according to number of blocks and sort them in ascending order;
Rank the ellipsoids according to the values of the difference of (volume - number of blocks) and sort them in descending order;
Select parents from the population according to roulette wheel selection;
Generate a new population by applying the following operators: crossover, mutation, and reproduction with their particular rates;
Compute the fitness of the individuals of the new population;
Until (The sopping criteria are satisfied) Repeat from 7;
Return Permutation matrix.

We found that the enclosing ellipsoid algorithm may find matrices with better block counts. To test the effect of the COLAMD algorithm on this algorithm, the code is performed on the matrices as a prior step before applying the enclosing ellipsoid algorithm, as we did in the case of the C1P algorithm. From our experiments the results are affected by the COLAMD algorithm where two points are worth noticing.

1. For some sparse matrices with low density the results are promising.

2. The enclosing ellipsoid algorithm produces the same results as the C1P algorithm or worse.

5.7 Summary

We have looked at the L-J ellipsoid otherwise known as the MVEE problem and suggested a different approach to solving it which doesn't consider the standard convex programming formulation. The approach is a novel implementation of a GA. This implementation works well for sets of points of low cardinalities in low dimensions. But, it also works for large sets in high dimensions, particularly if some adequate preprocessing is applied. For instance, by targeting only the relevant points, using the border strategy, we are able to improve

the algorithm substantially. Computational results show that this approach is efficient. The same approach is also used to solve the MVE estimator problem. The comparison with the 2-exchange method showed that our approach is faster and the ellipsoids include larger subsets, thus highlighting outliers. We have also considered a related problem which seeks to compute the maximum volume ellipsoid embedded in a given set of points. Computational results show that our approach works well. Note that an implementation of GA with the border strategy, coupled with parallel processing will solve potentially very large problem instances in terms of cardinality of sets of points and space dimensions. The MVEE has been used to solve the C1S problem. The ellipsoid helps push the 1's into large blocks. The procedure uses is multiple fitnesses. A possible extension of the basic ellipsoid problem is to consider dynamic (moving) sets of points instead of the usual static ones. The idea is to move the points periodically to put them inside ellipsoids of ever smaller volume. This indeed produces enclosing ellipsoids of minimum volume. In the context of C1S, this seems to work well.

Chapter 6

The Minimum Volume Enclosing Hyper-Rectangle Problem and Applications

6.1 Introduction

In this chapter we consider another related problem of MVEE which is that of finding the Minimum Volume Enclosing Hyper-Rectangle (MVEH) for a set of points. We also present a potential application in $(0,1)-$matrix analysis. A further application is the MVEE for a dynamic set of points.

6.2 Minimum Volume Enclosing Hyper-Rectangle Problem

The MVEH problem has many applications; enclosing point sets by three dimensional boxes are employed to preserve hierarchical partitioning of these sets. In statistics, it is used to sort and implement search range queries on a large database of samples, and in computer graphics, for collision detection. In these applications, the problem is separated into, first, a problem of providing a good splitting of a point set into many subsets; second, a problem of calculating an approximate box that encloses each subset [181].

Many authors considered this and related problems in [182–187]. Also, it is investigated in [188] where many instances for enclosing with hyper-rectangles and spheres (the Chebychev problem) are provided. O'Rourke [184] provided an exact algorithm for calculating an arbitrary oriented minimum volume boundary box in R^3 which needs $O(m^3)$ operations and $O(m)$ space. Later, Gill [181], produced a $(1+\epsilon)-$approximation of minimum volume bounding box of a set of m points in R^3 with cost $O(m + 1/\epsilon^{4.5})$. He also provided another simpler algorithm with cost $O(m \log m + m/\epsilon^3)$.

Chaudhuri and Samal [189], proposed a method to enclose regions by a boundary rectangle. They determined the minimum bounding rectangle enclosing every point in the region, by employing the least square method to find the directions of the major and minor axes of the region such that they aligned with the boundary of the rectangle. Many heuristic algorithms have been suggested to find an enclosing box of a set of points. Calculating the Axis Aligned Bounding Box (AABB) of a set of points is the easiest problem [181]. The same heuristic approach that is used for the original problem is used here.

6.2.1 Related problems

Let $C = \{x_i\}_1^m$ be a set of points in R^n, and let p be an arbitrary subset of points of C where $m/2 < p \leq m$. The clustering problem is that of enclosing p in the smallest axis parallel rectangle. Many approaches have been suggested for solving this problem [190–192]. Later, Segal and Kedem [183], proposed two algorithms to find the smallest axis parallel rectangle to contain a large number of points p. One of these algorithms is performed with cost $O(m + p(m-p)^2)$ and $O(m)$ space, and the other in dimension $n \geq 3$, has cost $O(nm + np(m-p)^{2(n-1)})$ and $O(nm)$ space. Also, Mount, Netanyahu, and Piatko [193] studied the $p-$enclosing problem. They proposed an algorithm to find the quantile approximations for the minimum annulus, strip, and ellipsoid containing a given quantile of the points with run time $O(m \log m)$.

In [185], a related problem is addressed. For a set C of $m \geq n+1$ points in R^n, $n \geq 2$, they introduced the maximum volume empty convex body B such that its interior does not contain any points of C. They provided a $(1-\epsilon)-$approximation algorithm to find the maximum volume of B among the points in $n-$dimensional unit box $[0,1]^n$. See

also [194–196].

6.2.2 Finding an enclosing minimum volume hyper-rectangle

It is known that finding a minimum rectangle around finite random points can be easily constructed by finding the minimum and the maximum points for the coordinates for the intervals that contain the points. But in the case that the minimum rectangle did not fit around them, it is rotated around them till it fits. Generating many rectangles and applying the genetic algorithm, and using the angles within the individual's string, one can rotate the shape around the points until the minimum one is obtained.

Representation of a rectangle

As we use the same procedure of the GA that is used to find the MVEE, Algorithm 8 is applied to this problem. The same operators of the GA are performed here; only the solution representation of the ellipsoid has changed. A solution is a rectangle that is represented as a chromosome depicted in Table 6.1. It comprises the following genes.

- the coordinates (c_1, c_2) of a point representing the corner position of the rectangle;
- the width w of the rectangle;
- the height h of the rectangle;
- the angle θ of tilt of the height and width with respect to the x-axis which is in counter anticlockwise order.

Table 6.1: *Chromosome representation of a rectangle*

Width	*Height*	*Corner position*		*Angle of tilt*
w	h	$c1$	c2	θ

Not only the parameters of the ellipsoid are replaced by the rectangle parameters in 2, 3, and high dimensions, but also checking the points, where checking a point in an ellipsoid which is different from a rectangle and hyper-rectangle. In $2d$ we check a point with the

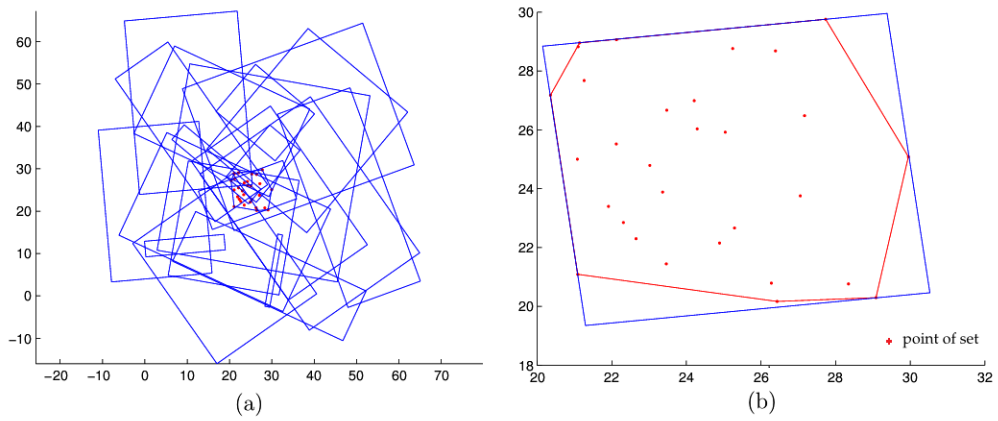

Figure 6.1: *(a) Initial rectangles and (b) rectangle containing polytope in* 2*d*

edges of the rectangle, while in dimensions $n \geq 3$ a point tested inside the vertices of the hyper-rectangle makes it a relatively harder task.

In our experiments, the same 50 points in 2*d* that are used in the MVEE algorithm earlier are used here. We applied the GA for 70 individuals. Figure 6.1 shows the different stages of the GA search process, where Figure (a) represents the initial population and (b) represents the final solution after 59 generations.

Table 6.2: *Chromosome representation of a hyper-rectangle*

Width	*Height*	*Length*	*Centre*		*Angle of tilt*
w	h	l	$c1$	c2	θ

The hyper-rectangle is constructed in the same way in $d \geq 3$, but by a width, height, length, and centre. Table 6.3 illustrates the results for several dimensions. Here, we did not use the strategy of reducing the points by checking only the points on the border. Checking the points inside a rectangle consumes more time than inside an ellipsoid especially in high dimensions as we need to check the points inside the convex hull of the hyper-rectangle. This is the reason for not testing the approach for more points and further dimensions. This is a good task for future work. Figure 6.2 presents the result in 3*d*.

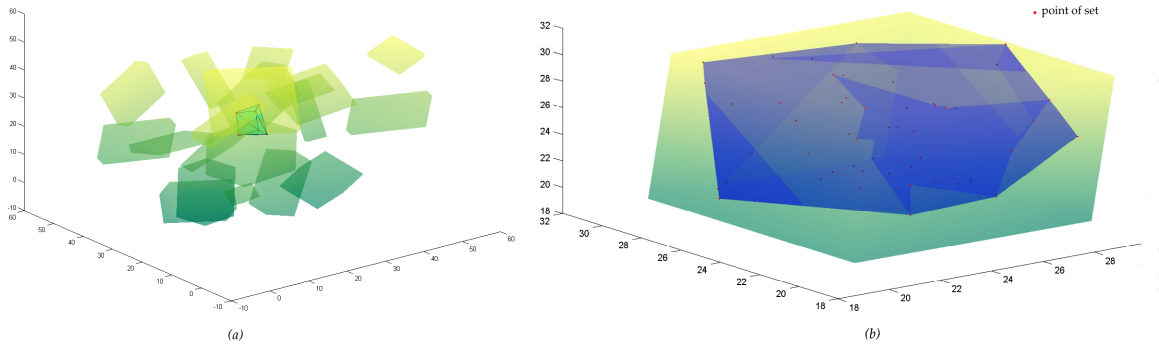

Figure 6.2: *(a) Initial hyper-rectangles in 3d; (b) Final hyper-rectangle containing points in 3d*

Table 6.3: *Time for MVEH in high dimensions*

Dim	*Points*	*Generations*	*Time(s)*
2	50	59	44.65
3	50	70	66.59
4	50	51	117.36
5	50	51	470.39
6	50	62	1850.20
7	50	52	13040.71
8	50	53	123911.15

6.3 Minimum volume ellipsoid enclosing a dynamic set of points

A possible extension of the basic ellipsoid problem is to consider dynamic sets of points instead of the usual static ones. The idea is that periodically the points to put inside an ellipsoid of minimum volume move randomly. The problem can be illustrated by looking at penning a flock of sheep in a pen of minimum area. We can start with a large pen and then reduce it while the sheep are moving. We can impose that, those near the edge move towards the centre by a set of steps, while anywhere else, they move by a set of smaller steps than those near the edge. This task can be done in two ways. One by repeatedly moving the points to new positions, fixing them, then minimizing the ellipsoid around them. The other way is by moving the points while the ellipsoid is being minimized around them

within the generations.

Definition 6.3.1 *A dynamic set of points is one whose elements positions change in time.*

6.3.1 Löwner-John ellipsoid for dynamic sets of points

Return again to the Fritz inclusion where an arbitrary set C of m points in R^n can be $n-$rounded with some ellipsoid $E \subseteq R^n$, as per Equation (5.6), of section 5.2.2. Let C be defined on the interval $[lb, ub]$, where $C \subseteq R^2$. Our goal is to reduce this interval j units from each side by pushing the points toward the centre of the interval. For j the new set is $C_j = [lb + j, ub - j]$, the volume of the ellipsoid in two dimensions is defined by the formula $vol = \pi * a * b$. For a special case let $a = b$, the volume becomes $vol = \pi * a^2$ where a represents the radius of the enclosing circle of the interval. The volume of this circle is

$$vol(E) = \pi * (\frac{ub - lb}{2})^2 \tag{6.1}$$

reducing the interval by j units will turn the equation into

$$vol(E_j) = \pi * (\frac{(ub - j) - (lb + j)}{2})^2 \tag{6.2}$$

$$= \pi * (\frac{ub - lb - 2j}{2})^2 \tag{6.3}$$

$$= \pi * (\frac{ub - lb}{2} - j)^2 \tag{6.4}$$

Divide both sides of the equation by a factor n, we get:

$$n^{-1}(vol(E_j)) = n^{-1}(\pi * (\frac{ub - lb}{2} - j)^2). \tag{6.5}$$

So the dynamic set C_j can be $n-$rounded as in

$$n^{-1}E_j \subseteq conv(C_j) \subseteq E_j, \tag{6.6}$$

and it can therefor be approximated by the Löwner-John ellipsoid E_j. This fact can be extended to higher dimensions. Apply the idea for the formula $vol = \pi * a * b$, reduce j from

both side of the major and minor axis;

$$vol = \pi * (a - 2j)(b - 2j)$$

$$= \pi * (ab - 2ja - 2jb + 4j^2).$$

Dividing both sides by n gives:

$$n^{-1} * vol(E_j) = n^{-1} * \pi * (ab - 2ja - 2jb + 4j^2)$$

Example: We can explain the idea by taking a set C of random points within the interval [20, 40], we want to nearly push the points inside the interval [29.5, 30.5]. Using the ellipsoid guarantees moving the points towards the centre of the interval and not outside it. The procedure starts with specifying the desired interval we want to push the points in. The distances of all the points of the set to its centre is ranked relying on these distances, then the midpoint is chosen; let j be the distance from the midpoint to the centre. The set is divided into two subsets; the one with distances larger than j moves by a step equal to j. This gives the points a large jump to the middle of the set near the centre. The set of points with distances smaller than j rotates around the centre with a small and suitable angle. The number of points to rotate could be more than half the set around the centre. The new set has new bounds $[20 + j, 40 - j]$ where a new population of ellipsoids will be generated depending on them. Repeating this technique and applying the MVEE algorithm will produce smaller ellipsoids in each iteration. The technique is described as Algorithm 11. For different sets of points, pushing these points to a small interval by finding the MVEE is represented in Table 6.4. Figure 6.3 shows the results of the application where the final interval appears like a thick point in the centre, which is the original points enclosing by the MVEE.

Algorithm 11: Enclosing ellipsoid for a dynamic set of points

Input the set of random points to be enclosed;
lb:= lower bound of the set;
ub:= upper bound of the set;
Specify the final desired interval of the set;
Apply GA and find the MVEE;
Move some points toward the centre;
Rotate the reminder points around the centre with a suitable angle;
Update lb, ub;
Generate a new random population of ellipsoids;
Find the MVEE that encloses the new set;
Until (The sopping criteria are satisfies, i.e., the set is within the desired interval) Repeat from 6;
Return Best solution.

Table 6.4: *Enclosing ellipsoid results for a dynamic set of points*

Points	*Iteration*	*Initial population*	*Generations*	*First interval*	*Final interval*	*Time*(*s*)
10	7	50	60	20-40	30.4209-29.1130	15.605
50	7	150	100	.	30.4030-29.8019	17.744
100	9	.	.	.	30.2776-29.6096	25.898
200	8	.	.	.	30.2899-29.6728	38.933
500	7	.	.	.	30.3414-29.6411	78.307
1000	7	.	.	.	30.3447-29.6875	147.219

6.4 Summary

A new approach has been provided to calculate the MVEH of a point set in R^n. The problem has been addressed using a different way from previous methods, which is based on the genetic algorithm. The performance of our algorithm is evaluated on small data sets in high dimensions of up to 8. Its implementation has been explained and experimental results are provided. We think this problem can be solved for more points and in high dimension. For future work we intend to check the points in $d \geq 9$. Also, Löwner-John ellipsoid for A dynamic set of points whose positions change in time is solved. The results of this application showed that the final interval of a dynamic set appears like a thick point in the centre.

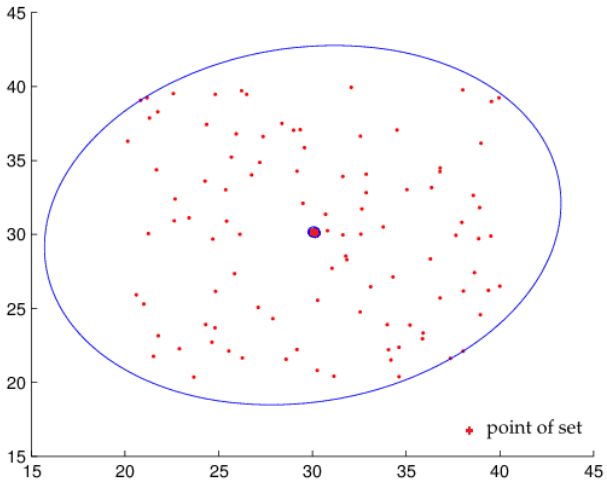

Figure 6.3: *Finding the enclosing ellipsoid for a dynamic set of points, the thick point is the same dynamic set enclosed by an ellipsoid after moving them to a small interval in the center*

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The thesis is in two fairly distinct parts:

1. Solving the C1S problem and
2. Solving the Löwner-John ellipsoid problem.

One can add a third part which is concerned with the application of the methodology of part 2 to solve the C1S problem. This, however, is not the only one. The procedures to both problems are mainly of the evolutionary type although exact methods and other heuristics are also considered.

The thesis reviews both topics of C1S and Löwner-John ellipsoids of minimum volume extensively. Application areas are highlighted and both theoretical and practical results are reported and discussed. This sets a clear backdrop to our contributions which are as follows.

- Solving the C1S problem with heuristic algorithms such as column insertion and reshuffling procedures, and an evolutionary approach of the GA type.
- Solving after preprocessing using the minimum degree ordering algorithm.
- Solving by combining preprocessing, GA followed by insertion/reshuffling.

- Solving the model (5.5) of the Löwner-John ellipsoid problem with an exact method such as the path-following infeasible interior-point algorithm.

- Solving the problem as MVEE using an original implementation of GA.

- Solving the MVE version of the problem in which the aim is to enclose a subset of the original set of points in an MVEE or LJ. This problem which is referred to as the minimum volume ellipsoid estimator problem is a very good tool for clustering data points.

- Solving the related problem of LVEE which is that of computing the largest volume ellipsoid that can be embedded within a set of points. This is a much harder problem, obviously.

- Solving the related problem of MVEH which is that of computing the minimum volume enclosing hyper-rectangle for a set of points.

- To improve the efficiency of the GA for MVEE we have implemented a border approaching which concentrates on the points of the boundary of the given set. This exploits the fact that if the outer points are enclosed, so would be all those nearer the centre of the set. Finally, we have tried to harness all these approaches to tackle again the C1S problem. The idea is that the 1's of a $(0, 1)-$matrix can be considered as points in a plane. A new implementation of GA with two objective functions used in tandem, one, restricting the number of blocks of 1's and the other the volume of the enclosing ellipsoid, gives relatively good results. Note that on top of all these implementations and experimentations, we have also tried to analyse some of the algorithms theoretically to provide complexity bounds. One very important point we have addressed is that of taking advantage of C1S to convert ILP into MILP. Our analysis shows that it is indeed beneficial to carry out the conversion.

7.2 Future Work

These analyses point to very interesting questions that can be tackled to further this investigation and strengthen its results. They are: Searching for better crossover operator to use in the MVEE algorithm to find more accurate solution in high dimensions. Applying the MVEE evolutionary algorithm to other problems such as the clustering problem. This can be done by repeating the finding of the MVEE for each cluster subset of the dataset. Solving the MVEH problem in high dimensions and for large datasets. The challenge about this task is checking the points which needs a cheap way. With respect to the LVEE problem that we solved in 2 and $3d$, in future we intend to find the LVEE within a set of points in high dimensions. The LVEE within a set of points can be extended into finding all the holes in a point set by fitting ellipsoids or any suitable shapes inside the holes. Although, the C1S evolutionary algorithm solved the C1S problem well for small matrices, we need to searching for a better fitness function for the C1S evolutionary algorithm that can efficiently solve the problem.

Bibliography

[1] M. Dom, J. Guo, and R. Niedermeier, "Approximability and parameterized complexity of consecutive ones submatrix problems," in *Theory and Applications of Models of Computation*, pp. 680–691, Springer, 2007.

[2] A. Tucker, "A structure theorem for the consecutive 1's property," *Journal of Combinatorial Theory, Series B*, vol. 12, no. 2, pp. 153–162, 1972.

[3] S. Weis and R. Reischuk, "The complexity of physical mapping with strict chimerism," in *International Computing and Combinatorics Conference*, pp. 383–395, Springer, 2000.

[4] W.-F. Lu and W.-L. Hsu, "A test for the consecutive ones property on noisy data-application to physical mapping and sequence assembly," *Journal of Computational Biology*, vol. 10, no. 5, pp. 709–735, 2003.

[5] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir, "Four strikes against physical mapping of dna," *Journal of Computational Biology*, vol. 2, no. 1, pp. 139–152, 1995.

[6] J. E. Atkins and M. Middendorf, "On physical mapping and the consecutive ones property for sparse matrices," *Discrete Applied Mathematics*, vol. 71, no. 1, pp. 23–40, 1996.

[7] J. E. Atkins, E. G. Boman, and B. Hendrickson, "A spectral algorithm for seriation and the consecutive ones problem," *SIAM Journal on Computing*, vol. 28, no. 1, pp. 297–310, 1998.

[8] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[9] J. von zur Gathen and M. Sieveking, "A bound on solutions of linear integer equalities and inequalities," *Proceedings of the American Mathematical Society*, vol. 72, no. 1, pp. 155–158, 1978.

[10] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness. 1979," *San Francisco, LA: Freeman*, 1979.

[11] F. John, "Extremum problems with inequalities as subsidiary conditions," *Courant Anniversary Volume, Interscience, New York, 1948, Mathematical Institute, Hungarian Academy of Sciences*, vol. 1053, pp. 187–204.

[12] P. Kumar and E. A. Yildirim, "Minimum-volume enclosing ellipsoids and core sets," *Journal of Optimization Theory and Applications*, vol. 126, no. 1, pp. 1–21, 2005.

[13] F. Annexstein and R. Swaminathan, "On testing consecutive-ones property in parallel," *Discrete Applied Mathematics*, vol. 88, no. 1-3, pp. 7–28, 1998.

[14] C. Berge, "Balanced matrices," *Mathematical Programming*, vol. 2, no. 1, pp. 19–31, 1972.

[15] S. Mecke and D. Wagner, "Solving geometric covering problems by data reduction," in *Algorithms–ESA 2004*, pp. 760–771, Springer, 2004.

[16] L. T. Kou, "Polynomial complete consecutive information retrieval problems," *SIAM Journal on Computing*, vol. 6, no. 1, pp. 67–75, 1977.

[17] D. S. Hochbaum and A. Levin, "Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms," *Discrete Optimization*, vol. 3, no. 4, pp. 327–340, 2006.

[18] Z. Adam, M. Turmel, C. Lemieux, and D. Sankoff, "Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution," *Journal of Computational Biology*, vol. 14, no. 4, pp. 436–445, 2007.

[19] F. Alizadeh, R. M. Karp, D. K. Weisser, and G. Zweig, "Physical mapping of chromosomes using unique probes," *Journal of Computational Biology*, vol. 2, no. 2, pp. 159–184, 1995.

[20] D. Fulkerson and O. Gross, "Incidence matrices and interval graphs," *Pacific Journal of Mathematics*, vol. 15, no. 3, pp. 835–855, 1965.

[21] P. Gilmore and A. Hoffman, "A characterization of comparability graphs and of interval graphs," *Canadian Journal of Mathematics*, vol. 16, pp. 539–548, 1964.

[22] D. Kendall, "Incidence matrices, interval graphs and seriation in archeology," *Pacific Journal of Mathematics*, vol. 28, no. 3, pp. 565–570, 1969.

[23] J. Meidanis, O. Porto, and G. P. Telles, "On the consecutive ones property," *Discrete Applied Mathematics*, vol. 88, no. 1, pp. 325–354, 1998.

[24] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms," *Journal of Computer and System Sciences*, vol. 13, no. 3, pp. 335–379, 1976.

[25] W.-L. Hsu, "A simple test for the consecutive ones property," in *Algorithms and Computation*, pp. 459–468, Springer, 1992.

[26] M. Habib, R. McConnell, C. Paul, and L. Viennot, "Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing," *Theoretical Computer Science*, vol. 234, no. 1, pp. 59–84, 2000.

[27] V. P. You, *On matrices that do not have the consecutive ones property*. PhD thesis, Citeseer, 2009.

[28] C. Chauve, J. Maňuch, and M. Patterson, "On the gapped consecutive-ones property," *Electronic Notes in Discrete Mathematics*, vol. 34, pp. 121–125, 2009.

[29] S. Haddadi, S. Chenche, M. Cheraitia, and F. Guessoum, "Polynomial-time local-improvement algorithm for consecutive block minimization," *Information Processing Letters*, vol. 115, no. 6, pp. 612–617, 2015.

[30] M. L. Gargano and L. Lurie, "A hybrid evolutionary approach to solving the archaeological seriation problem," *Congressus Numerantium*, vol. 180, p. 43, 2006.

[31] J. S. Deogun and K. Gopalakrishnan, "Consecutive retrieval property revisited," *Information Processing Letters*, vol. 69, no. 1, pp. 15–20, 1999.

[32] S. P. Ghosh, "File organization: the consecutive retrieval property," *Communications of the ACM*, vol. 15, no. 9, pp. 802–808, 1972.

[33] J. S. Deogun, V. V. Raghavan, and T. K. Tsou, "Organization of clustered files for consecutive retrieval," *ACM Transactions on Database Systems (TODS)*, vol. 9, no. 4, pp. 646–671, 1984.

[34] D. Fulkerson, O. Gross, *et al.*, "Incidence matrices with the consecutive 1's property," *Bulletin of the American Mathematical Society*, vol. 70, no. 5, pp. 681–684, 1964.

[35] S. Benzer, "On the topology of the genetic fine structure," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 45, no. 11, pp. 1607–1620, 1959.

[36] J. Tan and L. Zhang, "The consecutive ones submatrix problem for sparse matrices," *Algorithmica*, vol. 48, no. 3, pp. 287–299, 2007.

[37] A. Rajaraman, *Variants of the consecutive ones property: Algorithms, computational complexity and applications to genomics*. PhD thesis, Simon Fraser University, 2015.

[38] M. Dom, *"Recognition, generation, and application of binary matrices with the Consecutive Ones Property"*. Cuvillier, 2009.

[39] K. M. Koh, F. Dong, K. L. Ng, and E. G. Tay, *Graph Theory: Undergraduate Mathematics*. World Scientific, 2015.

[40] K. Thulasiraman and M. Swamy, *Graphs: Theory and Algorithms*. John Wiley & Sons, 1992.

[41] G. Blin, R. Rizzi, and S. Vialette, "A faster algorithm for finding minimum tucker submatrices," in *Programs, Proofs, Processes*, pp. 69–77, Springer, 2010.

[42] M. Dom, J. Guo, and R. Niedermeier, "Approximation and fixed-parameter algorithms for consecutive ones submatrix problems," *Journal of Computer and System Sciences*, vol. 76, no. 3, pp. 204–221, 2010.

[43] M. T. Hajiaghayi and Y. Ganjali, "A note on the consecutive ones submatrix problem," *Information Processing Letters*, vol. 83, no. 3, pp. 163–166, 2002.

[44] C. Lekkeikerker and J. Boland, "Representation of a finite graph by a set of intervals on the real line," *Fundamenta Mathematicae*, vol. 51, no. 1, pp. 45–64, 1962.

[45] M. B. Novick, "Generalized PQ-trees," tech. rep., Cornell University, Ithaca, NY, USA, 1989.

[46] J. Meidanis and E. Munuera, "A theory for the consecutive ones property," in *Proceedings of WSP*, pp. 194–202, 1996.

[47] G. P. Telles and J. Meidanis, "Building PQR trees in almost-linear time," *Electronic Notes in Discrete Mathematics*, vol. 19, pp. 33–39, 2005.

[48] R. M. McConnell, "A certifying algorithm for the consecutive-ones property," in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04)*, pp. 768–777, Society for Industrial and Applied Mathematics, 2004.

[49] S. Haddadi, "A note on the NP-hardness of the consecutive block minimization problem," *International Transactions in Operational Research*, vol. 9, no. 6, pp. 775–777, 2002.

[50] S. Haddadi and Z. Layouni, "Consecutive block minimization is 1.5-approximable," *Information Processing Letters*, vol. 108, no. 3, pp. 132–135, 2008.

[51] V. Bilo, V. Goyal, R. Ravi, and M. Singh, "On the crossing spanning tree problem," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pp. 51–60, Springer, 2004.

[52] R. G. Parker and R. L. Rardin, "Discrete optimization," 1988.

[53] I. Dumitrescu and T. Stützle, "Combinations of local search and exact algorithms," *Applications of Evolutionary Computing*, pp. 57–68, 2003.

[54] S. Salhi, *Heuristic Search: The Emerging Science of Problem Solving*. Springer, 2017.

[55] R. E. Gomory, "An algorithm for integer solutions to linear programs. princeton ibm mathematics research project," *Techn. Report*, no. 1, 1958.

[56] H. P. Williams, *Model building in mathematical programming*. John Wiley & Sons, 2013.

[57] R. S. Garfinkel and G. L. Nemhauser, *Integer programming*, vol. 4. Wiley New York, 1972.

[58] P. L. Hammer and S. Rudeanu, *Boolean methods in operations research and related areas.* Springer-Verlag, Berlin. 1968.

[59] F. Granot and P. L. Hammer, *On the Use of Boolean Functions in O-1 Programming.* Methods of Operations Research, vol 12, pp.154-184. 1972.

[60] P. L. Hammer and U. N. Peled, "On the maximization of a pseudo-boolean function," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 265–282, 1972.

[61] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

[62] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, vol. 6. 1997.

[63] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[64] A. Mercier, J.-F. Cordeau, and F. Soumis, "A computational study of benders decomposition for the integrated aircraft routing and crew scheduling problem," *Computers & Operations Research*, vol. 32, no. 6, pp. 1451–1476, 2005.

[65] S. Haddadi, "Benders decomposition for set covering problems," *Journal of Combinatorial Optimization*, pp. 1–21, 2015.

[66] A. M. Geoffrion and G. W. Graves, "Multicommodity distribution system design by benders decomposition," *Management Science*, vol. 20, no. 5, pp. 822–844, 1974.

[67] J. Naoum-Sawaya, "New benders' decomposition approaches for w-cdma telecommunication network design," *University of Waterloo*, 2007.

[68] S. Binato, M. V. F. Pereira, and S. Granville, "A new benders decomposition approach to solve power transmission network design problems," *IEEE Transactions on Power Systems*, vol. 16, no. 2, pp. 235–240, 2001.

[69] A. M. Costa, "A survey on benders decomposition applied to fixed-charge network design problems," *Computers & Operations Research*, vol. 32, no. 6, pp. 1429–1450, 2005.

[70] S. Haddadi and O. Slimani, "Alternative decomposition based approaches for assigning dusjunctive tasks," *Algorithmic Operations Research*, vol. 2, no. 2, pp. 129–136, 2007.

[71] E. Kalvelagen, "Benders decomposition with gams," *Technical report, Amsterdam Optimization Modeling Group LLC, Washington DC, the Hague*, 2002.

[72] I. Rechenberg, "Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution," *Frommann-Holzboog Verlag, Stuttgart (2nd edition 1993)*, 1973.

[73] H.-P. Schwefel, "Evolution and Optimum Seeking. Wiley, New York, NY," 1995.

[74] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor MI: Uinversity of Michigan Press, 1975.

[75] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, vol. 1. Cambridge: MIT press, 1992.

[76] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*, Wiley, New York, *1966*.

[77] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[78] V. Černỳ, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[79] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.

[80] E.-G. Talbi, *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons, 2009.

[81] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of Machine Learning*, pp. 760–766, Springer, 2011.

[82] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pp. 39–43, IEEE, 1995.

[83] Y. Shi *et al.*, "Particle swarm optimization: developments, applications and resources," in *evolutionary computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 81–86, IEEE, 2001.

[84] A. Salhi and E. S. Fraga, "Nature-inspired optimisation approaches and the new plant propagation algorithm," in *in Proceedings of the The International Conference on Numerical Analysis and Optimization (ICeMATH 11), Yogyakarta, Indonesia*, p. K21K28, 2011.

[85] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[86] M. Gen and R. Cheng, "Genetic algorithms and engineering design," *John Wily and Sons, New York*, 1997.

[87] L. Di Stefano and A. Bulgarelli, "A simple and efficient connected components labeling algorithm," in *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pp. 322–327, IEEE, 1999.

[88] M. B. Dillencourt, H. an an Samet, and M. Tamminen, "Connected component labeling for arbitrary binary image representations," in *Progress in Image Analysis and*

Processing: Proceedings of the International Conference on Image Analysis and Processing, p. 131, World Scientific, 1990.

[89] A. George and J. W. Liu, "The evolution of the minimum degree ordering algorithm," *Siam Review*, vol. 31, no. 1, pp. 1–19, 1989.

[90] I. S. Duff and J. K. Reid, *MA27-A set of Fortran subroutines for solving sparse symmetric sets of linear equations*. UKAEA Atomic Energy Research Establishment, 1982.

[91] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, no. 3, pp. 255–269, 1957.

[92] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1801–1809, 1967.

[93] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices*. Clarendon press Oxford, 1986.

[94] A. George and J. W. Liu, "Computer solution of large sparse positive definite system, prentice-hill inc., englewood cliffs, n.j," *Prentice Hall Professional Technical Reference*, 1981.

[95] A. Salhi, *Karmarker's algorithm; definitions and implementation*. PhD thesis, University of Aston, 1989.

[96] S. I. Larimore, *An approximate minimum degree column ordering algorithm*. PhD thesis, University of Florida, 1998.

[97] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "A column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 3, pp. 353–376, 2004.

[98] A. Salhi, "Project beeswax: On the problem of beehive location for optimum pollination," *Technical Report*, 2013.

[99] N. Ruf and A. Schöbel, "Set covering with almost consecutive ones property," *Discrete Optimization*, vol. 1, no. 2, pp. 215–228, 2004.

[100] N. Ruf, "Locating train stations: Set covering problems with almost C1P matrices," *Master's Thesis, University of Kaiserslautern*, 2002.

[101] C. A. Hoare, "Quicksort," *The Computer Journal*, vol. 5, no. 1, pp. 10–16, 1962.

[102] C. Canaan, M. Garai, and M. Daya, "Popular sorting algorithms," *World Applied Programming*, vol. 1, no. 1, pp. 42–50, 2011.

[103] B. Rylander, *Computational Complexity and the Genetic Algorithm*. PhD thesis, University of Idaho, 2001.

[104] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, ACM, 1971.

[105] M. Di Summa, "Formulations of mixed-integer sets defined by totally unimodular constraint matrices," PhD thesis, University of Padova, 2008.

[106] A. J. Hoffman and J. B. Kruskal, "Integral boundary points of convex polyhedra," in *50 Years of Integer Programming 1958-2008*, pp. 49–76, Springer, 2010.

[107] D. A. Tarvin, *Benders decomposition: An integer-programming extension with further computational enhancements*. Colorado School of Mines, 2014.

[108] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *European Journal of Operational Research*, vol. 94, no. 2, pp. 392–404, 1996.

[109] V. Chvátal, "Hard knapsack problems," *Operations Research*, vol. 28, no. 6, pp. 1402–1411, 1980.

[110] R. G. Jeroslow, "Trivial integer programs unsolvable by branch-and-bound," *Mathematical Programming*, vol. 6, no. 1, pp. 105–109, 1974.

[111] B. Krishnamoorthy, "Bounds on the size of branch-and-bound proofs for integer knapsacks," *Operations Research Letters*, vol. 36, no. 1, pp. 19–25, 2008.

[112] H.-G. Beyer, H.-P. Schwefel, and I. Wegener, "How to analyse evolutionary algorithms," *Theoretical Computer Science*, vol. 287, no. 1, pp. 101–130, 2002.

[113] A. E. Eiben and G. Rudolph, "Theory of evolutionary algorithms: a bird's eye view," *Theoretical Computer Science*, vol. 229, no. 1/2, pp. 3–9, 1999.

[114] G. Rudolph, "Finite markov chain results in evolutionary computation: a tour d'horizon," *Fundamenta Informaticae*, vol. 35, no. 1-4, pp. 67–89, 1998.

[115] J. He and X. Yao, "A study of drift analysis for estimating computation time of evolutionary algorithms," *Natural Computing*, vol. 3, no. 1, pp. 21–35, 2004.

[116] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+ 1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

[117] J. He and X. Yao, "Towards an analytic framework for analysing the computation time of evolutionary algorithms," *Artificial Intelligence*, vol. 145, no. 1/2, pp. 59–97, 2003.

[118] D. E. Goldberg *et al.*, *Genetic algorithms in search optimization and machine learning*. Addison-Wesley Reading Menlo Park, 1989.

[119] D. B. Fogel, "Evolutionary computation: towrad a new philosophy of machine intelligence. IEEE Press, Piscataway, NJ," 1995.

[120] T. Jansen, K. A. De Jong, and I. Wegener, "On the choice of the offspring population size in evolutionary algorithms," *Evolutionary Computation*, vol. 13, no. 4, pp. 413–440, 2005.

[121] J. He and X. Yao, "Drift analysis and average time complexity of evolutionary algorithms," *Artificial Intelligence*, vol. 127, no. 1, pp. 57–85, 2001.

[122] B. K. Ambati, J. Ambati, and M. M. Mokhtar, "Heuristic combinatorial optimization by simulated darwinian evolution: a polynomial time algorithm for the traveling salesman problem," *Biological Cybernetics*, vol. 65, no. 1, pp. 31–35, 1991.

[123] D. B. Fogel, "Empirical estimation of the computation required to reach approximate solutions to the travelling salesman problem using evolutionary programming," in *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, vol. 685, pp. 56–61, 1993.

[124] S. Droste, T. Jansen, and I. Wegener, "A rigorous complexity analysis of the (1+ 1) evolutionary algorithm for separable functions with boolean inputs," *Evolutionary Computation*, vol. 6, no. 2, pp. 185–196, 1998.

[125] J. He and H. Huang, "The computational time analysis of genetic algorithms," in *Proc. Fifth Chinese Joint Conference on Artificial Intelligence, Xian Jiaotong University Press, Xian*, pp. 440–443, 1998.

[126] T. Chen, J. He, G. Sun, G. Chen, and X. Yao, "A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 5, pp. 1092–1106, 2009.

[127] G. Rudolph, "How mutation and selection solve long-path problems in polynomial expected time," *Evolutionary Computation*, vol. 4, no. 2, pp. 195–205, 1996.

[128] C. Witt, "An analysis of the (μ+ 1) EA on simple pseudo-boolean functions," in *Genetic and Evolutionary Computation Conference*, pp. 761–773, Springer, 2004.

[129] G. Rudolph, *Convergence properties of evolutionary algorithms*. Verlag Dr. Kovač, Hamburg, 1997.

[130] T. Jansen and I. Wegener, "Evolutionary algorithms-how to cope with plateaus of constant fitness and when to reject strings of the same fitness," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 6, pp. 589–599, 2001.

[131] H. Hindi, "A tutorial on convex optimization II: duality and interior point methods," in *American Control Conference, 2006*, pp. 11–pp, IEEE, 2006.

[132] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[133] S. Silvey and D. Titterington, "A geometric approach to optimal design theory," *Biometrika*, vol. 60, no. 1, pp. 21–32, 1973.

[134] D. Titterington, "Optimal design: Some geometrical aspects of D-optimality," *Biometrika*, vol. 62, no. 2, pp. 313–320, 1975.

[135] H. W. Lenstra Jr, "Integer programming with a fixed number of variables," *Mathematics of Operations Research*, vol. 8, no. 4, pp. 538–548, 1983.

[136] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization*, vol. 2. Springer Science & Business Media, 2012.

[137] M. Sharir and E. Welzl, "A combinatorial bound for linear programming and related problems," in *STACS 92*, pp. 567–579, Springer, 1992.

[138] M. Dyer, A. Frieze, and R. Kannan, "A random polynomial-time algorithm for approximating the volume of convex bodies," *Journal of the ACM (JACM)*, vol. 38, no. 1, pp. 1–17, 1991.

[139] L. Lovasz and M. Simonovits, "On the randomized complexity of volume and diameter," in *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, pp. 482–492, IEEE, 1992.

[140] L. G. Khachiyan, "Rounding of polytopes in the real number model of computation," *Mathematics of Operations Research*, vol. 21, no. 2, pp. 307–320, 1996.

[141] M. J. Todd and E. A. Yıldırım, "On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids," *Discrete Applied Mathematics*, vol. 155, no. 13, pp. 1731–1744, 2007.

[142] Y. Zhang, "An interior-point algorithm for the maximum-volume ellipsoid problem," *Department of Computational and Applied Mathematics, Rice University, Technical Report TR98-15*, 1998.

[143] Y. Zhang and L. Gao, "On numerical solution of the maximum volume ellipsoid problem," *SIAM Journal on Optimization*, vol. 14, no. 1, pp. 53–76, 2003.

[144] L. G. Khachiyan and M. J. Todd, "On the complexity of approximating the maximal inscribed ellipsoid for a polytope," *Mathematical Programming*, vol. 61, no. 1, pp. 137–159, 1993.

[145] E. Barnes, "An algorithm for separating patterns by ellipsoids," *IBM Journal of Research and Development*, vol. 26, no. 6, pp. 759–764, 1982.

[146] B. Gärtner and S. Schönherr, "Smallest enclosing ellipses–fast and exact," *Proc. 13. Annual ACM Symposium on Compuational Geometry*, pp. 430–432.

[147] E. A. Yildirim, "On the minimum volume covering ellipsoid of ellipsoids," *SIAM Journal on Optimization*, vol. 17, no. 3, pp. 621–641, 2006.

[148] P. M. Vaidya, "A new algorithm for minimizing convex functions over convex sets," *Mathematical Programming*, vol. 73, no. 3, pp. 291–341, 1996.

[149] K. M. Anstreicher, "Ellipsoidal approximations of convex sets based on the volumetric barrier," *Mathematics of Operations Research*, vol. 24, no. 1, pp. 193–203, 1999.

[150] P. Sun and R. M. Freund, "Computation of minimum-volume covering ellipsoids," *Operations Research*, vol. 52, no. 5, pp. 690–706, 2004.

[151] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New Results and New Trends in Computer Science: Graz, Austria, June 20-21, 1991 Proceedings*, vol. 555, p. 359, Springer Science & Business Media, 1991.

[152] R. Seidel, "Linear programming and convex hulls made easy," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pp. 211–215, ACM, 1990.

[153] M. Grant, S. Boyd, and Y. Ye, "cvx users guide," 2009.

[154] D. Schlierkamp-Voosen and H. Mühlenbein, "Predictive models for the breeder genetic algorithm," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.

[155] S. Blum, R. Puisa, J. Riedel, and M. Wintermantel, "Adaptive mutation strategies for evolutionary algorithms," in *The Annual Conference: EVEN at Weimarer Optimierungsund Stochastiktage*, vol. 2, 2001.

[156] Y. Wu, *Software engineering and knowledge engineering: theory and practice*, vol. 2. Springer Science & Business Media, 2012.

[157] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Tecond International Conference on Genetic Algorithms*, pp. 14–21, 1987.

[158] T. Pencheva, K. Atanassov, and A. Shannon, "Modelling of a stochastic universal sampling selection operator in genetic algorithms using generalized nets," in *Proceedings of the Tenth International Workshop on Generalized Nets, Sofia*, pp. 1–7, 2009.

[159] R. Chakrabortty and M. Hasin, "Solving an aggregate production planning problem by using multi-objective genetic algorithm (MOGA) approach," *International Journal of Industrial Engineering Computations*, vol. 4, no. 1, pp. 1–12, 2013.

[160] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*, vol. 7. John Wiley & Sons, 2000.

[161] T. Jeyalilaseetha and R. Sukanesh, "Investigation on the performance of linear antenna array synthesis using genetic algorithm," *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journals of selected areas in telecommunications, JSAT*, pp. 60–66, 2011.

[162] E. Cantu-Paz, *Efficient and accurate parallel genetic algorithms*, vol. 1. Springer Science & Business Media, 2000.

[163] T. Ray and K. S. Won, "An evolutionary algorithm for constrained bi-objective optimization using radial slots," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 49–56, Springer, 2005.

[164] G. Chartier, "Introduction to genetic algorithms," 2008.

[165] S. D. Ahipaşaoğlu, "Fast algorithms for the minimum volume estimator," *Journal of Global Optimization*, vol. 62, no. 2, pp. 351–370, 2015.

[166] P. J. Rousseeuw and A. M. Leroy, "Robust regression and outlier detection," John Wiley and Sons, New York, 1987.

[167] R. Cook, D. Hawkins, and S. Weisberg, "Exact iterative computation of the robust multivariate minimum volume ellipsoid estimator," *Statistics and Probability Letters*, vol. 16, no. 3, pp. 213–218, 1993.

[168] J. A. Candela, "Exact iterative computation of the multivariate minimum volume ellipsoid estimator with a Branch and Bound algorithm," in *COMPSTAT*, pp. 175–180, Springer, 1996.

[169] D. M. Hawkins, "A feasible solution algorithm for the minimum volume ellipsoid estimator in multivariate data," *Computational Statistics*, vol. 8, pp. 95–107, 1993.

[170] W. L. Poston, E. J. Wegman, C. E. Priebe, and J. L. Solka, "A deterministic method for robust estimation of multivariate location and shape," *Journal of Computational and Graphical Statistics*, vol. 6, no. 3, pp. 300–313, 1997.

[171] D. M. Hawkins and D. J. Olive, "Improved feasible solution algorithms for high breakdown estimation," *Computational Statistics & Data Analysis*, vol. 30, no. 1, pp. 1–11, 1999.

[172] F. Torti, D. Perrotta, A. C. Atkinson, and M. Riani, "Benchmark testing of algorithms for very robust regression: Fs, lms and lts," *Computational Statistics & Data Analysis*, vol. 56, no. 8, pp. 2501–2512, 2012.

[173] D. Titterington, "Algorithms for computing d-optimal designs on a finite design space," in *Proc. of the 1976 Conf. on Information Science and Systems, John Hopkins University*, vol. 3, pp. 213–216, 1976.

[174] J. Balogh, H. González-Aguilar, and G. Salazar, "Large convex holes in random point sets," *Computational Geometry*, vol. 46, no. 6, pp. 725–733, 2013.

[175] P. Erdős and G. Szekeres, "A combinatorial problem in geometry," *Compositio Mathematica*, vol. 2, pp. 463–470, 1935.

[176] P. ERD& and G. Szekeres, "On some extremum problems in elementary geometry," in *Annales Univ. Sci. Budapest*, pp. 3–4, 53–62, 1960/1961.

[177] G. Tóth and P. Valtr, "The erdős–szekeres theorem, upper bounds and generalizations," *Discrete and Computational GeometryPapers from the MSRI Special Program, Cambridge University Press, Cambridge*, vol. 52, pp. 553–564, 2005.

[178] P. Erdős, "Some more problems on elementary geometry," *Austral. Math. Soc. Gaz*, vol. 5, no. 2, pp. 52–54, 1978.

[179] H. Harborth, "Konvexe fünfecke in ebenen punktmengen.," *Elemente der Mathematik*, vol. 33, pp. 116–118, 1978.

[180] J. D. Horton, "Sets with no empty convex 7-gons," *Canad. Math. Bull*, vol. 26, no. 4, pp. 482–484, 1983.

[181] G. Barequet and S. Har-Peled, "Efficiently approximating the minimum-volume bounding box of a point set in three dimensions," *Journal of Algorithms*, vol. 38, no. 1, pp. 91–109, 2001.

[182] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, "An optimal algorithm for finding minimal enclosing triangles," *Journal of Algorithms*, vol. 7, no. 2, pp. 258–269, 1986.

[183] M. Segal and K. Kedem, "Enclosing k points in the smallest axis parallel rectangle," *Information Processing Letters*, vol. 65, no. 2, pp. 95–99, 1998.

[184] J. O'Rourke, "Finding minimal enclosing boxes," *International Journal of Computer & Information Sciences*, vol. 14, no. 3, pp. 183–199, 1985.

[185] A. Dumitrescu, S. Har-Peled, and C. D. Toth, "Minimum convex partitions and maximum empty polytopes," *Journal of Computational Geometry*, vol. 5, no. 1, pp. 86–103, 2014.

[186] S. C. Nandy and B. B. Bhattacharya, "A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids," *Computers & Mathematics with Applications*, vol. 29, no. 8, pp. 45–61, 1995.

[187] E. M. Hendrix, I. García, J. Plaza, and A. Plaza, "On the minimum volume simplex enclosure problem for estimating a linear mixing model," *Journal of Global Optimization*, pp. 1–14, 2013.

[188] E. M. Hendrix, G. Boglárka, *et al.*, *Introduction to nonlinear and global optimization*. Springer New York, 2010.

[189] D. Chaudhuri and A. Samal, "A simple method for fitting of bounding rectangle to closed regions," *Pattern Recognition*, vol. 40, no. 7, pp. 1981–1989, 2007.

[190] A. Aggarwal, H. Imai, N. Katoh, and S. Suri, "Finding k points with minimum diameter and related problems," *Journal of Algorithms*, vol. 12, no. 1, pp. 38–56, 1991.

[191] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid, "Static and dynamic algorithms for k-point clustering problems," in *Workshop on Algorithms and Data Structures*, pp. 265–276, Springer, 1993.

[192] D. Eppstein and J. Erickson, "Iterated nearest neighbors and finding minimal polytopes," *Discrete & Computational Geometry*, vol. 11, no. 3, pp. 321–350, 1994.

[193] D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "Quantile approximation for robust statistical estimation and k-enclosing problems," *International Journal of Computational Geometry & Applications*, vol. 10, no. 06, pp. 593–608, 2000.

[194] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura, "Computing optimal islands," *Operations Research Letters*, vol. 39, no. 4, pp. 246–251, 2011.

[195] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger, "Finding minimum areakgons," *Discrete & Computational Geometry*, vol. 7, no. 1, pp. 45–58, 1992.

[196] P. Fischer, "Sequential and parallel algorithms for finding a maximum convex polygon," *Computational Geometry*, vol. 7, no. 3, pp. 187–200, 1997.

[197] J. Zhen and D. Den Hertog, "Computing the maximum volume inscribed ellipsoid of a polytopic projection," vol. 2015-004, CentER Discussion Paper Series, 2015.

[198] B. Hajek, "Hitting-time and occupation-time bounds implied by drift analysis with applications," *Adv. Appl. Prob*, vol. 14, no. 3, pp. 502–525, 1982.

[199] D. B. Fogel, *System identification through simulated evolution: A Machine Learning Approach to Modeling*. Ginn Press, Needham Heights, MA, 1991.

[200] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, pp. 85–103, Springer, 1972.

[201] P. S. Oliveto and C. Witt, "Simplified drift analysis for proving lower bounds in evolutionary computation," in *International Conference on Parallel Problem Solving from Nature, Dortmund, Germany*, pp. 82–91, Springer, 2008.

[202] J. Garnier, L. Kallel, and M. Schoenauer, "Rigorous hitting times for binary mutations," *Evolutionary Computation*, vol. 7, no. 2, pp. 173–203, 1999.

[203] J. Suzuki, "A markov chain analysis on simple genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 4, pp. 655–659, 1995.

[204] J. Suzuki, "A further result on the markov chain model of genetic algorithms and its application to a simulated annealing-like strategy," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 1, pp. 95–102, 1998.

[205] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms. San Matteo in Genetic Algorithms*, vol. 1, pp. 69–93, 1991.

[206] D. Thierens, D. E. Goldberg, and A. G. Pereira, "Domino convergence, drift, and the temporal-salience structure of problems," in *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 535–540, IEEE, 1998.

[207] W. M. Rudnick, "Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks," PhD thesis, Oregon Graduate Institute of Science and Technology, 1992.

[208] F. G. Lobo, D. E. Goldberg, and M. Pelikan, "Time complexity of genetic algorithms on exponentially scaled problems," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, pp. 151–158, Morgan Kaufmann Publishers Inc., 2000.

[209] J. L. Balcázar, J. Dıaz, and J. Gabarró, "Structural complexity theory I," EATCS Monographs on Theoretical Computer Science, Springer-Verag, vol. 11, 1988.

Publications

- R. Abo-Alsabeh and A. Salhi " An Evolutionary Approach to Constructing the Minimum Volume Ellipsoid Containing a Set of Points and the Maximum Volume Ellipsoid Embedded in a Set of Points", Submitted to The Journal of Algorithms and Computational Technology, 14-August-2017.
- R. Abo-Alsabeh and A. Salhi "A heuristic approach to the C1S problem", Submitted to The Journal of Algorithms and Computational Technology, 18-Septembers-2017.

Appendix A

Finding Maximum Volume Inscribed Ellipsoid (MVIE) For a System of Linear Inequalities

A.1 Introduction

The issue of the maximum inscribed ellipsoid attracted many authors. In the operation research and theoretical computer science computing dimensional polytope described by a set of linear constraints regards an outface problem. The problem of the MVIE has attracted a lot of interest because of its applications in the interior-point method. A polynomial on the complexity of approximating the maximal inscribed ellipsoid for a polytope is described by Khachiyan and Todd [144]. Zhang and Gao [143] studied practical solution methods for finding the maximal ellipsoid inscribing a full dimensional polytope in R^n defined by a system of affine inequalities. Zhen and Den Hertog [197] provided a method of computing the maximum volume inscribed ellipsoid and k-ball of a projected polytope. This problem is related to the problem of computing the Maximum Inscribed Sphere (MaxIS). For solving the latter there is one way by reducing it to a LP problem with one more dimension then solving it by using one of the LP algorithms. Due to that solving this problem could be costly. Another way to approximate polyhedron by ellipsoid could be by finding the box containing it. Our method depends on solving $2n$ LP problem using the simplex algorithm to find the upper and lower bounds of the polyhedron. The problem of MVIE is investigated also. In the next two subsections we clarify some well known information about linear programming and simplex method.

A.2 The Maximum Volume Inscribed Ellipsoid Problem

We now suggest another problem relates to the MVEE is the MVIE. For bounded and nonempty convex set C, the problem of inscribing the ellipsoid of maximum volume inside C and how it can be formulated when the ellipsoid parameterizes under an affine transformation as the image of the unit ball.

$$E = \{Dv + c \,|\, ||v||_2 \leq 1\} \tag{A.2.1}$$

By solving the following convex optimization problem, we can find the maximum volume ellipsoid inside C, where the volume of the ellipsoid is proportional to $det\, D$, as $D \in S_+^n$.

$$\begin{aligned} max \quad & log\ (det\, D) \\ s.t \quad & sup_{\|v\|_2 \leq 1}\ I_C(Dv + c) \leq 0 \end{aligned} \tag{A.2.2}$$

in the variables $D \in S^n$ and $c \in R^n$, with implicit constraint $D \succ 0$.

A.2.1 Maximum Ellipsoid in a Polyhedron

The problem of finding the ellipsoid of maximum volume in a polyhedron C, which is described by a set of linear inequalities:

$$C = \{x | a_i^T x \leq b_i,\ i = 1, ...m\}, \tag{A.2.3}$$

needs at first the constraints to demonstrate in a more suitable form as

$$\begin{aligned} \sup_{\|v\|_2 \leq 1} I_C\,(Dv + c) \leq 0 \ \Leftrightarrow\ & \sup_{\|v\|_2 \leq 1} a_i^T(Dv + c) \leq b_i\ , i = 1, ..., m. \\ & \Leftrightarrow ||Da_i||_2 + a_i^T c \leq\ b_i\ , i = 1, ..., m. \end{aligned} \tag{A.2.4}$$

(5.12) can be written as a convex optimization problem in the variables D and c:

$$\begin{aligned} min \quad & log\ det\, D^{-1} \\ s.t \quad & ||Da_i||_2 + a_i^T c \leq\ b_i\ , i = 1, ..., m. \end{aligned} \tag{A.2.5}$$

Centers of Minimum and Maximum Ellipsoids
As the two ellipsoids above represent two different problems, we will try to prove that their centers are different.

Theorem A.2.1 *For a given polytope P, the Löwner-John's minimum and maximum ellipsoids are not centric.*

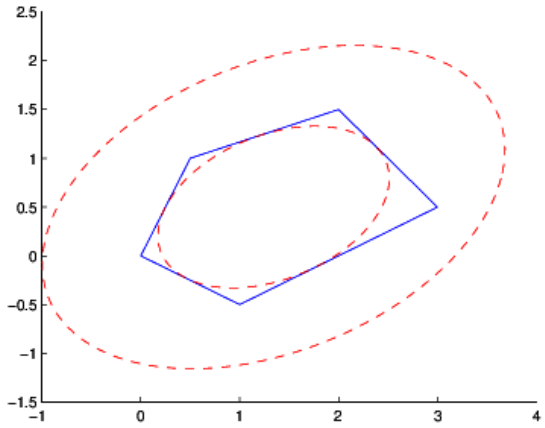

Figure A.1: *MVIE in a polyhedron*

proof:
assume that

$$\|c_1 - c_2\| = 0 \tag{A.2.6}$$

where c_1, c_2 are the centres of two ellipsoids,then for a vertex v belong to P,

$$\|c_1 - c_2\| = \|c_1 + v - v - c_2\| \leq \|c_1 - v\| + \|c_2 - v\| \tag{A.2.7}$$

$\because$ for a point $p \in P$, $\|P\| \leq n$, where n is the dimension of $P \subseteq R^n$
then

$$\leq |\|c_1\| - n| + |\|c_2\| - n| \tag{A.2.8}$$

$$\leq \|c_1| + \|c_2\| + 2n \tag{A.2.9}$$

$\because$ c_1, c_2 are the centers for the maximum and minimum ellipsoids, then $\|c_1 - c_2\| = 4n$

$$\therefore \quad c_1 \neq c_2 \Rightarrow \|c_1 - v\| \neq \|c_2 - v\|. \qquad \blacksquare$$

Solving the two problems to join the ellipsoids in one Figure is shown in Figure A.2.

A.3 Linear programming

A linear program is an optimization tool, determining the optimal value of a given objective function which is a linear of unknown decision variables and subject to a system of linear inequality and linear equality constraints. Linear programming problems are hard to solve. They may consist of many variables and constraints. Some of the main constraints may be inequalities and some are equalities, some variables could be unconstrained and

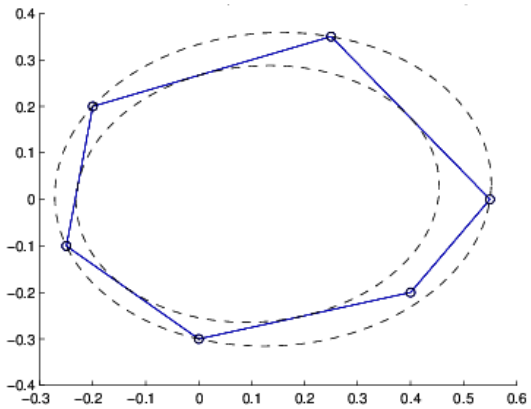

Figure A.2: *The MVEE and the MVIE in one figure*

others constrained to be nonnegative. There are two classes of LP problems; the standard maximum and minimum problems where all the main constraints are inequalities, and all variables are constrained to be nonnegative.

A.3.1 Simplex method

Linear programming problems with two variables can be solved graphically. Unfortunately, this method is not appropriate to solve real-life LPs with more than two variables. Simplex algorithm regards an efficient tool that can solve LPs with thousands of variables and constraints. It is required to convert the LP into the standard form before applying the simplex algorithm.

A.4 Finding the upper and lower bounds of LP via special objective function

The calculation of the upper bounds and lower bounds on the given polyhedron requires the solution of $2n$ LP problems when we are in $n-$dimensions. The issue is to see how expensive this calculation since all the problems have a special objective function with only one non-zero element. If all the calculations can be done in a cheap way, then it is possible to get an idea of where the polyhedron must lie and therefore approximate it using an ellipsoid. This, however, may prove not all that accurate. If the cost is too high anyway, it may not be worthwhile at all. We checked the cost of solving $2n$ LP problem using the simplex algorithm. One simple question to answer here: For the same A and b, how

expensive is it to solve a LP problem

$$\begin{aligned} max \quad & c^T x \\ s.t \quad & Ax \leq b, \\ & x \geq 0,\ x \in R_+^n \end{aligned}$$

where c^T has n nonzeros, compered to solving a similar problem with c^T having only one nonzero, let we say $c^T = (1, 0, 0, ..., 0)$? such that.

$$\begin{aligned} max \quad & x_1 \\ s.t \quad & Ax \leq b, \\ & x \geq 0,\ x \in R_+^n \end{aligned}$$

The quick answer for few experiments is that if the coefficients of the pivot's row are all positive then the second problem will need just one iteration to reach the optimality while the original one may need more than that. In case the the pivot's row contains negative and nonnegative coefficients then the two problems can be about the same.

Solving the last LP problem for maximizing or minimizing the objective function x_1 may not need to repeat running the simplex algorithm many times subject to the number of the variables. We applied this idea for many examples in different dimensions and chose Klee-Minty example as special case. We may solve it for the first nonzero variable then using the sensitivity analysis strategy from the final tableau to check the optimal solution for the second one in two dimensions or to the second and the third variables in three dimensions and so on for high dimensions.

Using sensitivity analysis allows us studying how the optimal solution of LP problem depends on its parameters, thats by changing in the LP's parameters. For the optimal tableau, let BV be the set of basic variables. We want to check whether BV stays optimal by changing the coefficients from zero to one, that means the BV may no longer optimal because row 0 has negative coefficient. The BV is turn to be suboptimal basis. As a result we need to find the optimal solution by continuing the simplex algorithm from final tableau. In this case the objective function will get better value than before.

Changing single cost coefficients for a nonbasic variable

For the objective function

$$z = \sum_{j=1}^{n} c_j x_j \tag{A.4.10}$$

Let x_j be a nonbasic variable with the objective function price c_j where $(c_j = 0)$ for all coefficients except $(c_1 = 1)$ for the variable x_1 as we assumed. The first tableau will be as following

1	-1	0	...	0	c_j	0	...	0	z_{n+1}
0	a_{11}	a_{12}	...		a_{1j}		...	a_{1n}	b_1
0	a_{21}	a_{22}	...		a_{2j}		...	a_{2n}	b_2
0					.				.
0					.				.
0					.				.
0	a_{m1}	a_{m2}	...		a_{mj}		...	a_{mn}	b_m

All the coefficients in row 0 in the final tableau will be zeros. Suppose we change any of these coefficients say c_j to $c_j + \Delta$ where $(\Delta = 1)$ and let $\bar{c}_j$ be the coefficient of x_j in the final tableau. We calculate the coefficient $\bar{c}_j$ by the formula

$$\bar{c}_j = \mathbf{c_{BV}}\mathbf{B^{-1}}\mathbf{a}_j - (c_j + \Delta) \tag{A.4.11}$$

where $\mathbf{a}_j$ is the column of the coefficients of the constraints that belongs to x_j. The $\mathbf{B^{-1}}$ and $\mathbf{b}$ (where $\mathbf{B}$ represents the matrix of the basis and $\mathbf{b}$ represents the the right-hand side) are not change, so $\mathbf{B^{-1}b}$ will not change. The BV remains feasible and $c_{\mathbf{BV}}$ the row of coefficients of the basis variables will not change also, only x_j is changed. Thus BV will stay optimal if the coefficient $\bar{c}_j$ of x_j in the final tableau is nonnegative. If $\bar{c}_j < 0$ then BV will no longer optimal and we need to enter x_j into the basis in the new optimal solution. In general, this may take more than one pivot to get the new optimal.

Changing single cost coefficients for a basic variable

Changing the coefficient for a basic variable is unlike nonbasic variable, it will affect the optimal solution of the Lp's problem. Changing this coefficient will change the BV tableau. As long as the ($\mathbf{B}$ and $\mathbf{b}$) are not changed, the BV will stay feasible, and the right-hand side will not change. But if $\mathbf{c_{BV}}$ is changed then $\mathbf{c_{BV}B^{-1}}$ will change and this may lead to change many coefficients in row 0. If all the coefficients in row 0 still nonnegative then the BV stays optimal, otherwise its now not optimal and we need to calculate the new row 0.

After changing the coefficient c_j of the basic variable x_j from zero to one such that $c_j = c_j + 1$, the new coefficient $\bar{c}_j$ in final tableau can be computed by the equation above. Also we need to calculate the coefficients of row 0 and see if all the coefficients are nonnegative then the BV stays optimal, otherwise we need to pivot to reach the optimality.

With respect to klee-minty example, Victor Klee and George Minty provided a class of linear programs. example number n for n nonnegative variables and n constraints is

$$\begin{aligned} max \quad & \sum_{j=1}^{n} 10^{n-j} x_j \\ s.t \quad & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1} \text{ for } 1 \leq i \leq n \\ & \text{all } x_j \geq 0 \end{aligned}$$

This example needs $2^n - 1$ pivots to reach an optimal solution using the minimal negative entry pivoting rule. We rewrote this example again subject to our suggestion as follow

$$\begin{aligned} max \quad & x_1 \\ s.t \quad & 2\sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1} \text{ for } 1 \leq i \leq n \\ & \text{all } x_j \geq 0 \end{aligned}$$

The only nonzero coefficient here is c_1. For this case the optimality is reached in one iteration. Using the sensitivity analysis technique for other variables by changing the coefficients from zero to one for each variable and setting the reminder to zero give a new problems. Changing any coefficient in the decision variables in the original formula to nonnegative turn the optimal basis BV to suboptimal basis. This needs two pivots to find the new optimal solution. That is because the first slack variable in final tableau is nonbasic variable with reduce cost coefficient 1. After the first iteration this value changes to negative and we need to change it again to zero and enter the variable into the basis variables. So to reach the optimality we need another iteration.

For other LP problems with only coefficient $c_1 = 1$, it takes one iteration to reach the optimal if the pivot row is positive, otherwise it may take more. If any of the c_j is changed from 0 to 1, then it is the same, we need 1 pivot to reach the optimality if the pivot row is positive,otherwise it needs more iterations.

Appendix B

Applying Bender's Decomposition

B.1 Introduction

The purpose of this work is to apply Benders Decomposition to the MILP problem after transforming a ILP problem with a (0, 1)−matrix into a MILP problem. We describe Bender's decomposition approach to optimally solve the ILP. This decomposition depends on the fact that ILP problems with C1P have totally unimodular coefficient matrices. In Chapter 3 we converted the ILP problem with (0, 1)−matrix into MILP using many algorithms. We separated matrix A into two parts $A = (A_1, A_2)$ where A_1 is the integer part and A_2 is the one with the C1P.

B.2 Benders' Decomposition Applied to MILP

We start with rewriting the MILP of Equation 4.8 at the page 66.

$$(MISC): \begin{cases} min & c_1x + c_2y \\ s.t & A_1x + A_2y \geq 1, \\ & x \in \{0, 1\}^p, \quad y \in R^{n-p} \end{cases} \tag{B.2.1}$$

Now we apply Benders procedures to this form, we want to illustrate how this problem is equivalent to the master problem MP. First eliminate the vector y by writing the problem as

$$\begin{cases} min\ z \\ \quad z \geq \quad \min c_2 y \\ \qquad\quad A_2 y \geq b - A_1 x, \\ \qquad\quad x \in \{0,1\}^p,\ \ y \in R^{n-p} \end{cases} \tag{B.2.2}$$

The dual of the problem in y

$$\begin{cases} min\ z \\ \quad z \geq \quad \max v(b - A_1 x) \\ \qquad\quad vA_2 \leq\ c_2 \\ \qquad\quad v \geq 0 \\ \qquad\quad x \in \{0,1\}^p \end{cases}$$

As the cost vector c_2 is non-negative in the set covering problem, our polyhedron

$$\{v\,|vA_2 \leq\ c_2,\ v \geq 0\}$$

is non-empty and has a finite number of vertices ($v^{(r)}$), $r \in R$. The constraint will be

$$z \geq \max\{v(b - A_1 x)\,|\,vA_2 \leq\ c_2,\ v \geq 0\}$$

that is equivalent to

$$z \geq v(b - A_1 x)\ \ for\ all\ v\ such\ that\ vA_2 \leq c_2, v \geq 0$$

which can be replaced by

$$z \geq v^{(r)}(b - A_1 x),\ r \in R$$

guarantee the feasibility of the *LP* is the second step.

$$(P): \begin{cases} min\ c_2 y \\ A_2 y \geq b - A_1 x \\ y \geq 0 \end{cases}$$

We need to avoid the infeasibility that is caused by the values of x. Problem P is infeasible if and only if the following linear program with a non-negative artificial variable β has an optimal value $\beta > 0$.

$$\begin{aligned} &min\ \beta \\ &A_2 y + 1\beta \geq b - A_1 x \\ &y \geq 0,\ \beta \geq 0 \end{aligned}$$

In the dual problem, P is infeasible if and only if the value of the following LP is positive.

$$\begin{aligned} &max\ w(b - A_1 x) \\ &wA_2 \leq 0 \\ &w1 \leq 1 \\ &w \geq 0 \end{aligned}$$

the following polyhedron has a finite number of vertices $w^{(t)}$, $t \in T$.

$$\{w \,|wA_2 \leq 0,\ w1 \leq 1,\ w \geq 0\}$$

thus P is feasible if and only if

$$\max\{w(b - A_1 x) \,|\, wA_2 \leq 0,\ w1 \leq 1,\ w \geq 0\} \leq 0$$

which equivalent to

$$w(b - A_1 x) \leq 0,\ for\ all\ w\ such\ that\ wA_2 \leq 0,\ w1 \leq 1,\ w \geq 0$$

and it can be replaced by

$$w^{(t)}(b - A_1 x) \leq 0,\ t \in T$$

Lastly, the problem MILP in (B.2.1, 4.8) is equivalent to following master problem $*$

$$(MP): \begin{cases} min\ z \\ z \geq v^{(r)}(b - A_1 x),\ r \in R \\ w^{(t)}(b - A_1 x) \leq 0,\ t \in T \\ x \in \{0,\ 1\}^p \end{cases} \tag{B.2.3}$$

Benders procedures for special case:
Suppose we convert a ILP problem from set covering into a MILP one and we get the following matrix.

$$M = \begin{array}{c} \quad\quad A_1 \quad\quad\quad\quad A_2 \\ \begin{pmatrix} c_1 & c_2 & c_3 & d_1 & d_2 & d_3 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

The submatrix A_1 is regarded as the integer part, and A_2 is the continuous part in the new MILP problem which is equivalent to the original one. Although, matrix A_2 has the C1P, it has two zero rows that prevent as to solve it as a LP problem. This rise the need to deal with it as a special case. The idea underlying the transformation of this matrix into another one with a full rank continuous part is simply by changing one 0 element into 1 in each zero row. With this changing A_2 still has the C1P. This process demands in adding the corresponding variable in the equation. Moreover, an equivalent value should be subtracted to maintain the feasibility. Before going any farther lets deal with the pervious matrix as an example.

$$\begin{array}{lll} \text{Min} & x_1 + x_2 + x_3 + y_1 + y_2 + y_3 & \\ & x_1 \quad + x_3 \quad\quad\quad\quad\quad\quad & \geqslant 1 \\ & \quad x_2 \; + x_3 \quad\quad + y_2 + y_3 & \geqslant 1 \\ & \quad\quad\quad + x_3 \quad\quad\quad\quad\quad\quad & \geqslant 1 \\ & x_1 + x_2 \quad\quad + y_1 + y_2 \quad\quad & \geqslant 1 \\ & x_1, x_2, x_3 \in \{0,1\}^p, \; y_1, y_2, y_3 \in R^{n-p} & \end{array} \tag{B.2.4}$$

By replacing the third zero in the first and the third constraints with the variable y_3 and subtract this value say x_4, this needs adding zero cost to the original one setting in the position of the new columns. Thats with the addition of the coupling constraints $y_3 = x_4$. By associating an integer variable to every replacing, the MILP is converted to the following

mixed integer problem.

$$\begin{aligned}
\min \quad & x_1 + x_2 + x_3 + 0 + y_1 + y_2 + y_3 & \\
& x_1 \quad + x_3 \; -x_4 \quad + y_3 & \geqslant 1 \\
& \quad x_2 \; + x_3 \quad +y_2 \; + y_3 & \geqslant 1 \\
& \quad + x_3 \quad + y_3 & \geqslant 1 \\
& x_1 + x_2 \quad + y_1 + y_2 & \geqslant 1 \\
& \quad -x_4 \quad + y_3 & = 0 \\
& x_1, x_2, x_3, x_4 \in \{-1, 0, 1\}^p, \; y_1, y_2, y_3 \in R^{n-p}
\end{aligned} \tag{B.2.5}$$

The two problems B.2.2, B.2.3 are equivalent, and the new will be matrix with size $m + 1 \times n + 1$ where 1 refers to the equality and the column that are added to the system B.2.2. The system can be written as following.

$$(MISC): \begin{cases} \min & c_1 x + c_2 y \\ \text{s.t} & A_1 x + A_2 y \geq 1 \\ & -x + \; y = 0 \\ & x \in \{-1, 0, 1\}, \;\; y \in R \end{cases} \tag{B.2.6}$$

The new cost is $(c_1, 0, c_2)$ with adding zeros cost, but we refer to $(c_1, 0)$ as c_1 for simplicity. The matrix $A_1 \in \{-1, 0, 1\}^{m \times (p+1)}$ refers to the integer inequalities part and $x \in \{-1, 0\}^{1 \times (p+1)}$ the integer equality part. The matrix $A_2 \in R^{m \times (n-p-1)}$ refers to the real inequality part and $y \in R^{1 \times (n-p-1)}$ is the real equality part. This problem is in the same form of problem (B2.1, 4.8). To eliminate y from the problem, the MISC can be written as

$$\begin{cases} \min z & \\ \quad z \geq & \min c_2 y \\ & A_2 y \geq 1 - A_1 x \\ & y = x \\ & x \in \{-1, 0, 1\}, \;\; y \in R \end{cases} \tag{B.2.7}$$

The dual of the problem in **y**

$$\begin{cases} \min z \\ \quad z \geq \quad \max\ v_1(1 - A_1 x) + v_2 x \\ \qquad\quad v_1 A_2 + v_2 \leq\ c_2 \\ \qquad\quad v_1, v_2 \geq 0 \\ \qquad\quad x \in \{-1, 0, 1\} \end{cases} \tag{B.2.8}$$

continuing the computation to directly adapt the following master problem which is equivalent to Equation B.2.1. The Mater Problem Set Covering (MPSC) is written as.

$$(MPSC): \begin{cases} \min z \\ z + (v_1^{(r)} A_1 - v_2^{(r)})\, x \geq v_1^{(r)} 1,\ r \in R \\ (w_1^{(t)} A_1 - w_2^{(t)})\, x \quad \geq w_1^{(t)} 1,\ t \in T \\ x \in \{-1, 0, 1\} \end{cases} \tag{B.2.9}$$

where $(v_1^{(r)}, v_2^{(r)})$, $r \in R$ and $(w_1^{(r)}, w_2^{(r)})$, $t \in T$ are respectively the $|R|$ vertices and the $|T|$ extreme rays of the polyhedron

$$\{(v_1, v_2) \,|\, v_1 A_2 + v_2 \leq\ c_2, v_1, v_2 \geq 0\}$$

The MPSC should have an optimal solution as long as its equivalent to the SCP. The obstacle is that the master problem has a large number of constraints where solving it regards a difficult task. So instead of solving the whole problem MPSC, the constraints can be generated iteratively in a Reduced Master Problem RMP via the iterative cutting plane algorithm.

The problem RMP has no constraints in the beginning of the iteration and they are added iteratively until satisfies the stop criteria. The dual of the problem in Equation B.2.9 which is corresponding to the problem MILP defined in Equation B.2.6, is called subproblem

$$(SP): \begin{cases} \max\ v_1(1 - A_1 x) + v_2 x \\ v_1 A_2 + v_2 \leq\ c_2 \\ v_1, v_2 \geq 0 \end{cases}$$

Given $x \in \{-1, 0, 1\}$, since $c_2 > 0$ then the SP problem has always a feasible solution. So, either it is unbounded or it has an optimal solution. If we start with the trivial cover when $y = 1$, we get $y = x = 0$, which guarantee the optimality of the SP problem. For the rest

of the iterations, suppose the subproblem has an optimal solution (v_1, v_2) and z is optimal value of the problem RMP, z (resp. the optimum solution of the subproblem) is a lower (resp. upper) bound on the value of problem MISC, and so of SCP, so the sequence of $z's$ is non-decreasing. Therefore, either

$$v_1(1 - A_1x) + v_2(x) \;=\; z$$

and it is done, or

$$v_1(1 - A_1x) + v_2(x) \;>\; z$$

so a cut is added

$$\begin{aligned} v_1(1 - A_1x) + v_2(x) \;&\leq\; z \\ v_1 1 \;&\leq\; z - (-v_1A_1 + v_2)\, x \end{aligned}$$

In case the subproblem is unbounded then we have

$$w_1(1 - A_1x) + w_2x \;>\; 0$$

along some extreme ray (v_1, v_2), which demands adding the cut

$$\begin{aligned} w_1(1 - A_1x) + w_2x \;&\leq\; 0 \\ w_1 1 \;&\leq\; (w_1A_1 - w_2)\, x \end{aligned}$$

Unfortunately, applying this procedure did not produce accurate results for many matrices. It seems that large randomly generated (0, 1)−matrices do not contain sufficiently large C1S. Therefore the suggested decomposition can not be justified. A crucial point is to prove some upper bound on the number of columns of any C1S that can be extracted from a (0, 1)−matrix. If a C1S submatrix can produce a strong upper bound, then definitely the result will be equal to that from applying other MILP methods.

Algorithm 12: Bender's Decomposition

(0) Perform the preprocessing phase () $R \leftarrow \emptyset, T \leftarrow \emptyset, UB \leftarrow, i \leftarrow 0$
Begin with $A_1 x = 0 = yA_2$
(1) Solve the LP

$$(SP) \quad \begin{cases} \max & \omega = v_1(1 - A_1 x) + v_2 x & \\ & v_1 A_2 + v_2 & \leq c_2 \\ & v_1, v_2 & \geq 0 \end{cases}$$

If (SP) has an optimal solution (v_1^*, v_2^*) with value ω^* then
    $R \rightarrow R \cup \{(v_1^*, v_2^*)\}$
    If $UB > \omega^*$ then $UB = \omega^*$
    Go to (2)
If (SP) is unbounded (let (w_1^*, w_2^*) be the extreme ray) then
    $T \leftarrow T \cup \{(w_1^*, w_2^*)\}$
(2) Solve the MILP

$$(RMP): \quad \begin{cases} \min z & \\ z + (v_1^{(r)} A_1 - v_2^{(r)})\, x \geq v_1^{(r)} 1, & v \in R \\ (w_1^{(t)} A_1 - w_2^{(t)})\, x \geq w_1^{(t)} 1, & w \in T \\ x \in \{-1, 0, 1\} & \end{cases}$$

Let x be an optimal solution with value z^*
$LB \leftarrow z^*$
If $LB = UB$ then STOP else go to (1)

Appendix C

On GA complexity

C.1 Some interesting general results

Most of the time complexity in the previous work on EAs concentrated on simple cases or only suggested (1 + 1)−EAs without crossover. Here some general results are worth mentioning.

• He and Yao [121] suggested a general EA that employs a finite population, crossover, mutation, selection. For a problem the cost which is at least exponential time (in problem size n), they proposed some drift conditions such that an EA will cost no more than polynomial time (in problem size n) to solve the problem.

Instead of working on a specific EA, they provided a general theory for a class of EAs. Their theory has been improved via drift analysis which is a useful procedure for analyzing random sequences [198]. This theory can find the first hitting time by estimating the drift of a random sequence which is simpler than estimating the first hitting time directly. Their simple proposal starts with modelling the evolution of an EA population as a random sequence, like a Markov chain. The EA consists of a population of multiple chromosomes with the crossover and mutation operators. The drift of the sequence is analyzed to and from the optimum value. Subject to various drift conditions, different bounds are found on the first hitting time. Some of these conditions lead the random sequence to drift in the direction of the optimum, whereas other conditions drift the sequence away from the optimum.

The result of the theory is applied to some classical combinatorial optimization problems such as (the subset sum problem). A simple description of the EAs and drift analysis with some results can be as follows: Find $max\{f(y);\ y \in V\}$ for a function $f(y)$ and a finite state space V, $y \in V$. Suppose $f_{max} = f(y^*)$ where y^* is one state with the optimum value, the EA

used for solving the problem is as follows:

1. Generate heuristically or randomly an initial population of $2N$ chromosomes, represented by $\lambda_0 = (y_1, ..., y_{2N})$. Let $i \leftarrow 0$ and N an integer number, $N > 0$. Define $f(\lambda_i) = max\{f(y_j); y_j \in \lambda_i\}$ for any population λ_i.

2. Generate a new population of offspring using the crossover and mutation operators or any other operators and denote it $\lambda_{i+1/2}$.

3. Select and reproduce $2N$ chromosomes from populations $\lambda_{i+1/2}$ and λ_i and find a new intermediate one λ_{i+V}.

4. Stop if $f(\lambda_{i+V}) = f_{max}$; else let $\lambda_{i+1} = \lambda_{i+V}$ and $i \leftarrow i + 1$, and go to second step 2.

This EA is nearer to evolutionary programming [199] and the evolution strategies [73] than to GAs [118] with respect to applying the crossover and/or mutation before the selection operator. The drift analysis is described as follows: Let $d(y, y^*)$ be the distance between a point y and the optimum y^*. In case there are more than one optimal value then the distance between the chromosome y and the optimal set V^* is suggested as $d(y, V^*) = min\{d(y, y^*) : y^* \in V^*\}$, simply noted as $d(y)$. The distance from a given population $Y = \{y_1, ..., y_{2N}\}$ to the optimum is

$$d(Y) = min\{d(y) : y \in Y\} \tag{C.1.1}$$

The EA generates a random sequence $\{d(\lambda_i); i = 0, 1, 2, ...\}$ which can be modelled by a homogeneous Markov chain. The drift of this sequence at time i is described as

$$\Delta(d(\lambda_i)) = d(\lambda_{i+1}) - d(\lambda_i)$$

The time of the EA to stop is defined as $t = min\{i : d(\lambda_i) = 0\}$ which represents the first hitting time on the optimum value. Their work tested the relationship between the problem size n and the predicted first hitting time t. In fact they estimated the first hitting time $E(t)$ with respect to different conditions; for more details see [121]. To this end, suppose a deterministic algorithm is used to solve an optimization problem with the distance d between the optimum and the starting solution. We require at most d/Δ time iterations to reach the optimum when the drift towards the optimum is greater than Δ at each iteration. The following theorems show the results for some problems with polynomial time.

The subset sum problem:
This problem is NP-complete where the partition problem can be polynomially transformed

to it [200]. To clarify the problem, suggest a set $H_n = \{h_1, ..., h_n\}$ of n integers and large integer D, the problem is finding a subset of V of H such that summation the elements in V does not exceed the largest number D. The objective function of the problem is $\sum_{j=1}^{n} h_k v_k$ where a feasible solution can be represented by an individual $y = (v_1, ..., v_n)$, $v_n \in \{0, 1\}$ such that $\sum_{k=1}^{n} h_k v_k \leq D$. The EA that is used here is suggested with the mating, mutating, and selection operators.

Theorem C.1.1 *[121] Given the family of subset sum problems and the EA to solve them. For any initial population Υ with $d(\Upsilon) > 0$, $E[\, t \,|\, \lambda_0 = \Upsilon] \leq h(n)$, where $h(n)$ is a polynomial of problem size n, $h(n) = O(n^2)$.*

For the following problems, a $(2N + 2N)-$EA with selection and mutation and without crossover is employed. Also, the individuals are represented as $V = \{(v_1, ...v_n),\ v_j \in \{0, 1\}\}$.

linear functions:
A function $f : V \to R$ is linear if $f(y) = u_0 + \sum_{j=1}^{n} u_j v_j$ where $u_j \in R$.

Theorem C.1.2 *[121] For the linear function with positive coefficients $u_1 > u_2 > ... > u_n > 0$, the EA with mutation probability $p_m = 1/n$ requires an average $O(n \ln n)$ steps to reach the optimum.*

Pesudo-modular functions:
An example of this function is

$$f(y) = \sum_{j=1}^{n} \prod_{k=1}^{j} v_k. \tag{C.1.2}$$

Theorem C.1.3 *[121] The expected first hitting time of the EA for the fitness function (4.17) is $E[\, t \,] \leq n^2(e - 1)$ when mutation rate $p_m = 1/n$.*

Unimax functions:
A function $f : V \to R$ is unimax if the $y^* \in V$ is the only locally maximal point. One of the known unimax problem is the long path problem. The path L_n for an odd number n is defined by recursion with $L_1 = \{0, 1\}$ as the base path. The length of the paths is set out by the recurrence equations $|L_1| = 2$, $|L_{n+2}| = 2|L_n| + 1$, and its solution is $|L_n| = 3.2^{(n-1)/2} - 1$ for odd $n \gg 1$.

Given y as a point on the path $|L_n|$, the fitness function

$$f(y) = -(3.2^{(n-1)/2} - 2) + \begin{cases} Pos(y), & if\ Pos(y) \geq 0, \\ -\sum_{j=1}^{n} v_j, & othewise. \end{cases} \tag{C.1.3}$$

where $Pos(y)$ is the location of y on the path that is numbered from 0 to $3.2^{(n-1)/2} - 2$ and $Pos(y)$ is equal -1 for a point not belonging to the path.

Theorem C.1.4 *[121] For the unimax function (4.18), starting from the bottom of the increasing path, the expected hitting time of the EA is $E[\,t\,] = O(n^3)$ when mutation rate $p_m = 1/n$.*

Almost positive functions
Let $f : V \rightarrow R$ be a function, it says to be almost positive if the coefficients of all nonlinear terms are non-negative [114]. An example of the function where the distance function can be defined as $d(y) = \sum_{j=1}^{n} |v_j - 1|$ is

$$f(y) = n - \sum_{j=1}^{n} v_j + (n+1) \prod_{j=1}^{n} v_j \tag{C.1.4}$$

Theorem C.1.5 *[121] The expected first-hitting time of the EA for the almost positive function (4.19) is $E\,[\,t\,] = \Omega(n^n)$ when mutation rate $p_m = 1/n$ and the EA starts from $d(\lambda_0) = n$.*

On the other hand, some optimization problems can be solved by EA in exponential average time in the problem size n. He and Yao analysis is relied on Hajek's previous work [198]. In the case that the EA may only find approximate solution rather than an exact optimum, the time t to stop the EA is defined as $t = min\{i : d(\lambda_i) \leq d_p\}$ where $d_p \geq 0$. The following two conditions show that the population on average will not drift in the direction of the optimum.

Condition 1 [121]. For any population Y with $d_p < d(Y) < d_l$, where $d_p \geq 0$ and $d_l > 0$,

$$E[\,e^{-(d(\lambda_{i+1}) - d(\lambda_i))} | \lambda_i = Y,\ d_p < d(\lambda_i) < d_l] \leq \sigma < 1, \tag{C.1.5}$$

where $\sigma > 0$ is a constant.
this shows that searching the interval (d_p, d_l) is so hard, that the population on average drift far from the optimum.

Condition 2 [121]. For any population Y with $d(Y) \geq d_l$, $d_l > 0$,

$$E[\, e^{-(d(\lambda_{i+1})-d_l)} | \lambda_i = Y,\, d(\lambda_i) \geq d_l] \leq K, \tag{C.1.6}$$

where $K \geq 1$ is a constant.

This shows that the offspring population on average will not drift too much in the direction of the optimum. The following theorem includes the general conditions 1, 2 where the EA requires at least exponential time on average to reach the optimum.

Theorem C.1.6 *[121]Suppose the two previous conditions hold, If* $d(\lambda_0 \geq d_l$, $K \geq 1$ *and* $\sigma < 1$, *then there exist some* $\mu_1 > 0$ *and* $\mu_2 > 0$ *such that*

$$E[\, t \,| d(\lambda_0) \geq d_l] \geq \mu_1 e^{\mu_2 (d_l - d_p)} \tag{C.1.7}$$

For a subset sum problem revisited, the EA that is used to solve it can cost on average at least an exponential time to reach the optimum.

Theorem C.1.7 *[121]For the random sequence,* $\{d(\lambda_i);\, i = 1, 2, ...\}$, *defined by the family of subset sum problems and the EA in this section, if* $d(\lambda_0) \geq d_l$, *then there exist two constants* $\mu_1 > 0$ *and* $\mu_2 > 0$ *such that*

$$E[\, t \,| d(\lambda_0) \geq d_l] \geq \mu_1 e^{(\mu_2 n)} \tag{C.1.8}$$

• He and Yao [115] again estimated the complexity time of the EAs drift analysis. They discussed drift conditions that are employed to find the lower and upper bounds of the first hitting times. Also, a new general classification of easy and hard problems for EAs relied on this analysis are provided. Their EA is proposed to solve maximization problems. Let E be the set of all populations, and let a random variable λ_ι that its values from E be the ι^{th} population of generation. Let E^* is the set of populations with the optimum, to test the distance of the population say $\mathbf{y}$ to E^* a distance function is used. The distance is defined in many ways like Hamming distance, or $G(\mathbf{y}) = min\{|f(\mathbf{y}) - f(\mathbf{z})|; \mathbf{z} \in E^*\}$ or $G(\mathbf{y}) = 0$ if $\mathbf{y} \in E^*$ and $G(\mathbf{y}) = 1$ if $\mathbf{y} \notin E^*$, where $\mathbf{y}$ and $\mathbf{z}$ are any population in E.

To find the EAs computation time, Markov chain and supermartingale are provided as mathematical models for the EAs. Markov chain is used to model the sequence of random variables $\{\lambda_\iota;\, \iota = 0, 1, 2, ...\}$. To find the gain of a population towards the optimal solution, the one-step mean drift at ι^{th} generation for a given distance function is needed. It is defined as

$$\mathbf{E}\,[\, G(\lambda_\iota) - G(\lambda_{\iota+1}) \,|\, \lambda_\iota = \mathbf{y}] := G(\mathbf{y}) - \sum_{\mathbf{z} \in E} P(\mathbf{y}, \mathbf{z}; \iota)\, G(\mathbf{z})$$

where $P(\mathbf{y}, \mathbf{z}; \iota)$ is the transition probability and $\mathbf{y}, \mathbf{z} \in$E are any population.

$$P(\mathbf{y}, \mathbf{z}; \iota) := P(\lambda_{\iota+1} = \mathbf{z} \,|\, \lambda_\iota = \mathbf{y})$$

If this drift is positive then the population converges to the optimal solution and if it is negative then it diverges far from it. For the EA, number of generations to reach optimum (first hitting time), is defined as $t := min\{\iota \geq 0; \lambda_\iota \in E^*\}$. The drift analysis that is applied for specific EAs is showed in the following two cases.

Analysis of a (1+1)-EA for linear functions

This problem can be solved by a (1+1)-EA with mutation and selection with two different distance functions, see [115] for more details. The first function with a binary string $y = (v_1, ..., v_n)$ which is used in the following theorem is

$$d(y) = 4(n-1)^2(\sum_{j=1}^{n} |v_j - 1|) \tag{C.1.9}$$

Theorem C.1.8 *[115] Let t be the mean number of generations for the* $(1+1)-EA$ *to find an optimal solution, then*

$$\boldsymbol{E}[\, t \,|\lambda_0] = O(n^3). \tag{C.1.10}$$

This result can be tightened further by using the second distance

$$d(y) = n \ln (1 + \sum_{j=1}^{n/2} s\,|v_j - 1| + \sum_{j=n/2+1}^{n} |v_j - 1|), \tag{C.1.11}$$

where s $(1 < s \leq 2)$.

Theorem C.1.9 *[115] Let t be the mean number of generations for the* $(1+1)-EA$ *to find an optimal solution, then*

$$\boldsymbol{E}[\, t \,|\lambda_0] = O(n \ln n). \tag{C.1.12}$$

Analysis of a (n+n)-EA for ONEMAX problem

This problem can be solved by $(1+1)-$EA in time $O(n \ln n)$ [124], He and Yao solved by $(n+n)-$EA with mutation and selection only, where n is the length of the string.

Theorem C.1.10 *[115] Let t be the mean number of generations for the* $(n+n)-EA$ *to find an optimal solution, then*

$$\boldsymbol{E}[\, t \,|\lambda_0] = O(n \ln n). \tag{C.1.13}$$

This result can be changed into $\mathbf{E}[\, t\,|\lambda_0] = O(n)$ with a different distance function; Hamming distance between an chromosome and the optimum value.

He and Yao separated the optimization problems into two types relied on the average number of generations required to find the solution.

1. Easy class; given EA, the average number of generations required by the EA to solve a problem is polynomial in the problem size. The sufficient and necessary conditions for this class as follows.

   Theorem C.1.11 *[115] Given an EA which can be modelled by a homogeneous absorbing Markov chain, a problem belongs to the easy class iff there exists a distance function $G(\boldsymbol{y})$ such that; for a polynomial $h_1(n)$ in the problem size n, $G(\boldsymbol{y}) \leq h_1(n)$, and for any population λ_ι at generation ι with $G(\lambda_\iota) > 0$, the one-step mean drift satisfies*

   $$\boldsymbol{E}[G(\lambda_\iota) - G(\lambda_{\iota+1})|\lambda_\iota] \geq b_{low}, \tag{C.1.14}$$

   where $b_{low} > 0$ is a lower bound constant.

2. Hard class; given EA, the average number of generations required by the EA to solve a problem is exponential in the problem size.

   Theorem C.1.12 *[115] Given an EA which can be modelled by a homogeneous absorbing Markov chain, a problem belongs to the hard class iff there exists a distance function $G(\boldsymbol{y})$ such that; for some population $\boldsymbol{y} \in E$ and an exponential $h_2(n)$ in the problem size n, $G(\boldsymbol{y})$ satisfies $G(y) \geq h_2(n)$, and for any population λ_ι at generation τ with $G(\lambda_\iota) > 0$, the one-step mean drift satisfies*

   $$\boldsymbol{E}[G(\lambda_\iota) - G(\lambda_{\iota+1})|\lambda_\iota] \leq b_{up}, \tag{C.1.15}$$

   where b_{up} is a positive upper bound constant.

• Chen et al [126], analyzed the time complexity of the population-based evolutionary algorithms on unimodal problems by improving a new general method relying on EAs models and some common[well-known] concepts, like the supermartinle [115, 121, 198, 201], the Markov chain model [117, 202–204], and the takeover time which provided by Goldberg and Deb [205]. They joined the idea of the overtake time to EAs and drift analysis to prove

that the $(N + N)-$EA with truncation selection (or two-tournament selection) and the bitwise mutation requires $O(n \ln \ln n + n \ln n/N)$ and $O(n \ln n + n^2/N)$ generations to obtain the global optimum of the ONEMAX and LEADINGONES problems, consequently, where N is number of individuals and n is the length of the string. Instead of using only a selection operator, they generalized the takeover time for the EAs with mutation operators.

• Thierens et al, [206] analyzed of the time complexity of convergence the BinInt problem which provided by Rudnick [207]. It is called the sequential convergence (*domino convergence*) because it is similar to a falling row of domino stones. They provided that the time complexity of domino convergence is exponential $O(2^n)$ for proportionate selection and linear in number of building blocks $O(n)$ for selection algorithms with constant selection intensity like (tournament or truncation selection). These results were compared with the previous results for ONEMAX problem, where n is the chromosome length. Their convergence behavior differ from the ONEMAX or Bit-Counting problem which is the reason of choosing it as a prototypical example. Their analysis provided that the ONEMAX problem is of order $O(n \ln n)$ for proportionate selection and $O(\sqrt{n})$ for selection algorithms with constant selection intensity.

• Fernando et al, [208] introduced experimental and theoretical analysis of the GAs complexity time on problems exponentially scaled building blocks. This work relies on the previous one of Thierens et al, but here for the building blocks instead of the case of single genes. They found an overall quadratic time complexity in terms of the evolution of the fitness function under idealized situations as they suggested perfect building block mixing. For the case of the building blocks uniformly scaling, where the time complexity of the GA with perfect mixing is $O(m)$ where the population size and convergence time grow linearly with $\sqrt{m}$ and for the building blocks exponential scaling is $O(m^2)$. Integer m is number of building blocks.

• Rylander, [103] illustrated the Minimum Chromosome Length (MCL) method to compute genetic algorithm time complexity of problems. Using this approach shows the possibility of finding time complexity of problems based GAs. This approach is relied on the search space growth rate as a function of the input problem size. They proved two particular cases empirically and defined a new complexity class NPG (the class of problems that can be solved them by GA with cost more polynomial time). The worst case complexity of the problem for a GA can be bounded by the MCL growth rate. The problem belongs to the class PO (the optimization equivalents of P) if the MCL grows slowly enough. Conversely, it belongs to the class of NPO (the optimization equivalents of NP) as the MCL growth

rate will be no more than linear where searching the space does not grow faster than exponentially [209].

Appendix D

Matlab code

D.1 Code of C1S problem

The problem of C1S.

```
function [ ] = Solving_C1P_Problem( )
% We solve the C1S problem using the genetic algorithm. This file is main one
% that calls other files.
% ************************************************************************
clc; close all; clear all;
[R]=SetCoveringCost();
[m, n]=size(R);
% Apply the code of COLAMD
amdperm=colamd(R);
R=R(:, amdperm);
% applying colamd ones a gain on R to use the permutation as affective individual
amdperm1=colamd(R);
R=R(:, amdperm1);
CountCBM(R);
% ************************************************************************
clear bestSofar;
clear totalFitness;
populationSize=100;
populationSize=4*ceil(populationSize/4);
corssOverPercentage=0.8;
noOfElites=10;
noOfGeneration=100;
```

```
noOfCrossOvers=corssOverPercentage*populationSize;
noOFMutaion=10;
%****************************************************************
% Generating population of individuals
for k=1:populationSize
    x(k, :)=randperm(size(R, 2));
end
%****************************************************************
% Start the loop of generations z=0;
for l=1: noOfGeneration
    if l <0.7* noOfGeneration
        noOFMutaion;
        noOfCrossOvers;
    else
        noOFMutaion=20;
        noOfCrossOvers=0;
    end
y=x;
% Evaluate fitness value of the individuals
FitnessValue=[ ];
for i=1:populationSize
    A=R(:, y(i, :));
% Counting number blocks in the matrix A using FF5      S1=[ ];
    for e=1:size(A, 1)
        q = diff([0 A(e, :) 0] == 1);
        v = find(q == -1) - find(q == 1);
        row=size(v, 2);
        S1=[S1, row];
    end
    S1;
    nbOfBlocks=sum(S1);
    FitnessVal =1/nbOfBlocks ;
    FitnessValue = [FitnessValue, FitnessVal];
    end
%*****************************************************************
    xFit= FitnessValue';
    fitCumSum=cumsum(xFit);
```

```
    rouletteWheel=fitCumSum/fitCumSum(end);
    y=[y, xFit];
    [j, k]=sort(y(:, end), 'descend');
% sort rows according to last column
    y=y(k, :);
    y=y(:, 1: end-1);
% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    for crindex=1: 2: noOfCrossOvers
    ran=rand;
    for i=1: populationSize-1
       if (ran>rouletteWheel(i))
              parent1=x(i+1, :);
       end
    end
    ran=rand;
    for i=1: populationSize-1
       if (ran>rouletteWheel(i))
          parent2=x(i+1, :);
       end
    end
% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
% Crossover
    children=OXcrossover(parent1, parent2, 0.2);
    y(crindex+noOfElites, :)=children(1, :);
    y(crindex+noOfElites+1, :)=children(2, :);
    end
% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    for tm=1:noOFMutaion
    randPos=round(populationSize*rand);
       if randPos==0
          randPos=populationSize;
       end
    mutated=mutateSwap(y(randPos, :));
    y(noOfCrossOvers+noOfElites+tm, :)=mutated;
    end
% * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    x=y;
```

```
    sortedFit=sort(xFit, 'descend');
    bestSofar(l)=sortedFit(1);
    if l > 1 & bestSofar(l) > bestSofar(l-1)
        z=0;
    else
        z=z+1;
    end
    if  z==500; break;
    end
    end
% Best solution
    finalSolution=x(1, :);
    A=R(:, finalSolution);
    save('Afile.txt', 'A', ' -ascii');
    ExtractPropertyC1P(n, A);
    Inserting_Column_In_C1P( );
    CountCBM(A);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This file is generating matrices from set covering
function [Matrix]=Set_Covering_Cost( )
m=Dimension of matrix;
B=[ ];B1=[ ];
for t=1:2
    xtreps = 100*rand(m, 1);
    ytreps = 100*rand(m, 1);
% The radius of the matrix
    r=6;
    nt=length(xtreps);
    for i=1: nt
        for j=1: nt
            a(i, j)=sqrt(( ytreps(j)- ytreps(i)) ^ 2+ ( xtreps(j)- xtreps(i)) ^ 2);
        end
    end
    for i=1: nt
        for j=1: n t
            if a(i, j)<=r
```

```
                a(i, j)=1;
            else
                a(i, j)=0;
            end
        end
    end
    Matrix = a;
    P = Matrix(:, 1: 2: end);
    C = Matrix(:, 2: 2: end);
    B = [B, P];
    B1 = [B1, C];
end
Matrix =[ B(:, 1: m/2), B1(:, m/2+1: end)];
char(Matrix+ '0');
save('Matrixfile.txt', 'Matrix', '-ascii');
n1 = nnz(Matrix);
density = nnz(Matrix) / numel(Matrix);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [solution ]= mutateSwap(solution)
% Mutation File, we swap here only two numbers
rand1= randi([1, length(solution)]);
rand2= randi([1, length(solution)]);
temp= solution(rand1);
solution(rand1)= solution(rand2);
solution(rand2)= temp;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [children]= OXcrossover(parent1, parent2, pc)
% OX Order Crossover, we choose any segment and any two crossover points then instead
% of repairing the chromosome we rearrange the rest of the genes to give a legal tour.
p1=parent1;
p2=parent2;
nVar=numel(p1);
c=randsample(nVar-1, 2);
c1=min(c);
c2=max(c);
```

```
ch1=p1(c1+1: c2);
[~, loc]=ismember(ch1, p2);
loc=sort(loc);
y1=[p1(1:c1) p2(loc) p1(c2+1: end)];
ch2=p2(c1+1:c2);
[~, loc1]=ismember(ch2, p1);
loc1=sort(loc1);
y2=[p2(1: c1) p1(loc1) p2(c2+1: end)];
children=[y1; y2];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ ]=ExtractPropertyC1P(n, A)
% This code extracts the C1P columns after performing the GA
A=load('Afile.txt');
A1=A;
[m, n]=size(A);
% Scan column for C1P
for l=1: n
% Matrix A after transfers the first row to the end
    r1=A(:, 2: end);
    r2=A(:, 1);
    A=[r1, r2];
    S=[ ];
% Scan for the next block of ones
    for e=1:m
        idx=find(A(e, :)==1);
        for j=1:size(idx, 2)-1
            if idx(j+1)-idx(j)> 1
                k2= j+1;
                k=idx(:, k2);
                S=[S, k];
            break;
            else
            end
        end
    end
    t=min(S);
```

```
    numberColum=max(sort(t))-1;
    SubmatrixC1P=A(:, 1: numberColum);
    aa{ l }=size(SubmatrixC1P, 2);
end
w1= min(cellfun(@(x) x, aa ));
if w1==0
    MaxSubmatrixC1P=A;
    ('whole matrix is satisfy property C1P');
%*****************************************************************
% If the matrix is continuous then Solve it as LP problem, we need to change the signs
according to the problem
    i=size(A, 2);
    c=[2: i+1];
    b=-ones(size(A, 1), 1);
    A=-A;
    lb=zeros(i, 1);
    ub=ones(i, 1);
    Aeq=[ ];
    beq=[ ];
    [x, fval] = linprog(c, A, b, Aeq, beq, lb, ub);
%*****************************************************************
else
% If not then solve it as MILD by separating it into continuous and discrete parts
    [SizeMaxSubmatrixC1P, ListMaxSubmatrixC1P]=max(cellfun(@(x) x, aa ));
    ExtractMaxSubmatrixC1P= aa{ListMaxSubmatrixC1P}
% In case the maximum C1S matrix list is final matrix
    if ListMaxSubmatrixC1P==n
% Rearrange the parts in the matrix after consecutive
        A1=[A1(:, 1:ListMaxSubmatrixC1P-1), A1(:, ListMaxSubmatrixC1P: end)];
% Continuous part
        MaxSubmatrixC1P=A1(:, 1:SizeMaxSubmatrixC1P);
% Discrete part
        ReminderBigMatrix=A1(:, SizeMaxSubmatrixC1P+1: end);
    else
% In case any maximum C1S matrix list before final
        A1=[A1(:, ListMaxSubmatrixC1P+1: end), A1(:, 1: ListMaxSubmatrixC1P)];
        MaxSubmatrixC1P=A1(:, 1: SizeMaxSubmatrixC1P);
```

```
        ReminderBigMatrix=A1(:, SizeMaxSubmatrixC1P+1: end);
    end
    sizeMaxSub=size(MaxSubmatrixC1P, 2);
    Remisize=size(ReminderBigMatrix, 2);
    save('MaxSubmatrixC1Pfile.txt', 'MaxSubmatrixC1P', '-ascii');
    save('ReminderBigMatrixfile.txt', 'ReminderBigMatrix', '-ascii');
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ReminderBigMatrix, MaxSubmatrixC1P] = Inserting_Column_In_C1P( )
% This is the column insertion algorithm which calls matrices from EtractPropertyC1P
MaxSubmatrixC1P=load('MaxSubmatrixC1Pfile.txt');
ReminderBigMatrix=load('ReminderBigMatrixfile.txt');
%**********************************************************************
m=size(MaxSubmatrixC1P, 1);
MaxSubmatrixC1P2=MaxSubmatrixC1P;
p=size(ReminderBigMatrix, 2);
KeepReminder=[ ];
if p > 0
    for j=1: p
% We update the matrix that is achieved improvement here to insert the second column
j=2 after inserting j=1.
        for i=1: size(MaxSubmatrixC1P, 2)-1
            a=MaxSubmatrixC1P(:, i);
            b=ReminderBigMatrix(:, j);
            if i< size(MaxSubmatrixC1P, 2)
                c=MaxSubmatrixC1P(:, i+1);
            else
                c=MaxSubmatrixC1P(:, end);
            end
% counting the consecutive rows
            for e=1:m
                if (a(e)==0 && b(e)==1 && c(e)==0)&&(sum( MaxSubmatrixC1P(e, :)))~ =0
                    count=0;
                    break;
                elseif (a(e)==1 && b(e)==0 && c(e)==1)
                    count=0;
```

```
                break;
            else
                count=e;
            end
        end
% insert the fitting column
        if count==e
            ('whole matrix is satisfy property C1P');
            MaxSubmatrixC1P2=[MaxSubmatrixC1P(:, 1: i)
ReminderBigMatrix(:, j) MaxSubmatrixC1P(:, i+1: end)];
            break;
        end
    end
    if size(MaxSubmatrixC1P2, 2) == size(MaxSubmatrixC1P, 2)
        KeepReminder=[KeepReminder, ReminderBigMatrix(:, j)];
    end
    if size(MaxSubmatrixC1P2, 2) > size(MaxSubmatrixC1P, 2)
        MaxSubmatrixC1P=MaxSubmatrixC1P2;
    end
  end
  ReminderBigMatrix=KeepReminder;
  sizehaddadic1p2=size(MaxSubmatrixC1P, 2);
  Remisize=size(ReminderBigMatrix, 2);
  V=[ReminderBigMatrix, MaxSubmatrixC1P];
  CountCBM(V);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ ]=CountCBM(A)
% Finding the final number of all blocks of ones after the GA
S1=[ ];
for e=1:size(A, 1)
  q = diff([0 A(e, :) 0] == 1);
  v = find(q == -1) - find(q == 1);
  row=size(v, 2);
  S1=[S1, row];
end
S1;
```

```
nbOfBlocks=sum(S1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D.2 Code of MVEE

```
% Simulation file to run the files of MVEE in 2 dimensions from ConvHull file. The result
% depends on the initial ellipses they should cover the points. Also it depends on e.

clc;clear;close all;
m=Dimension; n1=Number of points; n=Number of initial ellipsoids; e=Rate of ϵ;
Pop=Population size; Ge=Generations;
lb=Lower bound of the points; ub=Upper bound of the points;
% Call the file of generating the set of points
[Border_Points]=Generate_Points(m, n1, e, lb, ub);
% Calls the file of generating the initial ellipsoids
[Res_Matrix]=custom_Plot(n, m, n1, e, lb, ub);
% Calling the file of genetic algorithm from matlab toolbox
[z, fval, exitflag, output, population] = optim(5, Pop, Ge, Res_Matrix);
% Calling the file to draw the figure
figure;
PlotEllipse(z);
% Check the points outside the ellipsoid and add them to inside the ellipsoid
for i = 1:n
% Calls parameters
    a=abs(z(1)); b=abs(z(2)); xCenter=z(3); yCenter=z(4); phi=z(5);
    Border_Points=load('Border_Pointsfile.txt');
    P=size(Border_Points,1);
    phi2=phi*pi/180;
% Finding the foci
    c=sqrt((a) ^ 2-(b) ^ 2);
    F1=[xCenter-cos(phi2)*c , yCenter-sin(phi2)*c];
    F2=[xCenter+cos(phi2)*c , yCenter+sin(phi2)*c];
    vol=pi*a*b;
    s=2*a;
```

```
    p= load('pfile.txt');
    W=(s < sqrt((p(:,1)-F1(:,1)). ^ 2+( p(:,2)-F1(:,2)). ^ 2)
          +sqrt((p(:,1)-F2(:,1)). ^ 2+( p(:,2)-F2(:,2)). ^ 2));
    nbPointsOutside=sum(W);
    k=find(W==1);
    k1=p(k,:);
    Border_Points=[Border_Points; k1];
    P1=size(Border_Points,1);
    save('Border_Pointsfile.txt', 'Border_Points', '-asci');
    if nbPointsOutside > 0
        [Res_Matrix]=custom_Plot(n, m, n1, e, lb, ub);
        [z, fval, exitflag, output, population] = optim(5, Pop, Ge, Res_Matrix);
        figure;
        PlotEllipse(z);
    else
        break
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File generating the points then find the border to check inside the ellipsoid.
function[Border_Points]=Generate_Points(m, n1, e, lb, ub)
hold on
p=[ ];
Idx=[ ];
for j=1:m
    x = lb+(ub-lb)* rand(n1, 1);
    x1=min(x); x2=max(x);
    p=[p, x];
% Find index the points in the border
    idx = [((x > x1+e) & (x < x2-e))];
    Idx=[Idx, idx];
end
save('pfile.txt', 'p', '-ascii');
if m==2
    plot(p(:, 1), p(:, 2), '.');
end
if m==3
```

```
    plot3(p(:, 1), p(:, 2), p(:, 3), '.');
end
Idx;
[rows, cols] = size(Idx);
Keeping_rows = false(rows, 1);
for row=1: 1: n1
    all_Ones = all(Idx(row, :));
    if  all_Ones
        Keeping_rows(row) = true;
    end
end
% Final index
Keeping_rows;
% Reminder of X to plot
Border_Points = p(Keeping_rows, :);
P=size(Border_Points, 1);
% Save points of border to check inside MVEE
save('Border_Pointsfile.txt', 'Border_Points', '-ascii');
if m==2
    plot(Border_Points(:, 1), Border_Points(:, 2), 'r*');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generating initial population of ellipsoids
function [Res_Matrix]=custom_Plot(n, m, n1, e, lb, ub)
% Load points from Generate_Points
Border_Points=load('Border_Pointsfile.txt');
Size_Points=size(Border_Points, 1);
% Generating ellipsoids for small interval (20-30)
phi=0;
results= zeros(n, 8);
for i=1: n
    p1=(lb+ub)*rand(1, 2);
    p2=(lb+ub)*rand(1, 2);
    t=linspace(0, 2*pi, 1000);
% Plotting the ellipsoids
    xCenter=(p1(:, 1)+p1(:, 2))/2; yCenter=(p2(:, 1)+p2(:, 2))/2;
% Find the length between points p1, p2
```

```
    d=sqrt((p1(:, 1)-p1(:, 2))^ 2+(p2(:, 1)-p2(:, 2))^ 2);
% Find major axis and minor axis
    a=d/2; b=a/3;
    u(1, :)=a*cos(t); u(2, :)=b*sin(t);
    phi=phi+20;
    phi2=phi*pi/180;
% Rotate Matrix
    Q=[cos(phi2) -sin(phi2); sin(phi2) cos(phi2)];
    u=Q*u;
    u(1, :)=u(1, :)+xCenter; u(2, :)=u(2, :)+yCenter;
% plot(u(1, :), u(2, :), 'b');
% find the foci
    c=sqrt((a) ^ 2-(b) ^ 2);
    F1=[xCenter-cos(phi2)*c, yCenter-sin(phi2)*c];
    F2=[xCenter+cos(phi2)*c, yCenter+sin(phi2)*c];
    f1=[F1(:, 1), F2(:, 1)]; f2=[F1(:, 2), F2(:, 2)];
%Plot(f1, f2, '*r');
    volume=pi*a*b;
    s=2*a;
% Check points on the convexhull
    W=(s >= sqrt((Border_Points(:, 1)-F1(:, 1)). ^ 2+(Border_Points(:, 2)-F1(:, 2)). ^ 2)
            +sqrt((Border_Points(:, 1)-F2(:, 1)). ^ 2+(Border_Points(:, 2)-F2(:, 2)). ^ 2));
    nbPointsInside=sum(W)
    if (i==1)
       minvol=volume;
       if (nbPointsInside==Size_Points)
          optimalVol=minvol;
       end
    end
    if (minvol<volume )
       minvol=volume;
       if(nbPointsInside==Size_Points)
          optimalVol=minvol;
       else
          continue;
       end
    end
```

```
    results(i, 1)=a;
    results(i, 2)=b;
    results(i, 3)=xCenter;
    results(i, 4)=yCenter;
    results(i, 5)=phi;
    results(i, 6)=vol;
    results(i, 7)=nbPointsInside;
    fitnessValue=nbPointsInside-volume;
    results(i, 8)=fitnessValue;
end
% Result
sortedRes= sortrows(results, 8);
Res_Matrix= sortedRes(:, 1:5);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Computing the fitness function of the individuals, z is the individual
function [fitnessValue]=fitnessFct(z)
a=abs(z(1)); b=abs(z(2));
xCenter=z(3); yCenter=z(4);
phi=z(5);
Border_Points=load('Border_Pointsfile.txt');
Size_Points=size(Border_Points, 1);
phi2=phi*pi/180;
c=sqrt((a) ^ 2-(b) ^ 2);
F1=[xCenter-cos(phi2)*c, yCenter-sin(phi2)*c];
F2=[xCenter+cos(phi2)*c, yCenter+sin(phi2)*c];
vol=pi*a*b;
s=2*a;
W=(s >= sqrt((Border_Points(:, 1)-F1(:, 1)). ^ 2+( Border_Points(:, 2)-F1(:, 2)). ^ 2)
         +sqrt((Border_Points(:, 1)-F2(:, 1)). ^ 2+( Border_Points(:, 2)-F2(:, 2)). ^ 2));
nbPointsInside=sum(W);
if (nbPointsInside<P)
    fitnessValue=(Size_Points-nbPointsInside +100+volume) ^ 2 ;
else
    fitnessValue=(volume-nbPointsInside);
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the ellipsoid
function [Res_Matrix]=PlotEllipse(z)
a=abs(z(1)); b=abs(z(2));
xCenter=z(3); yCenter=z(4);
phi=z(5);
hold on;
% p= load('pfile.txt');
% plot( p(:, 1), p(:, 2), '.');
Border_Points=load('Border_Pointsfile.txt');
plot( Border_Points(:, 1), Border_Points(:, 2), '.r');
t=linspace(0, 2*pi, 1000);
% Plotting the ellipsoids
u(1, :)=a*cos(t);
u(2, :)=b*sin(t);
phi2=phi*pi/180;
% Rotation matrix
Q=[cos(phi2) -sin(phi2); sin(phi2) cos(phi2)];
u=Q*u;
u(1, :)=u(1, :)+xCenter; u(2, :)=u(2, :)+yCenter;
plot(u(1, :), u(2, :), 'b');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x, fval, exitflag, output, population] =
optim(nvars, PopulationSize_Data, Generations_Data, InitialPopulation_Data)
% Start with the default options
options = gaoptimset;
% Modify options setting
options = gaoptimset(options, 'PopulationSize', PopulationSize_Data);
options = gaoptimset(options, 'Generations', Generations_Data);
options = gaoptimset(options, 'InitialPopulation', InitialPopulation_Data);
options = gaoptimset(options,'CrossoverFcn', @crossoverintermediate [] );
options = gaoptimset(options,'CrossoverFraction', CrossoverFraction_Data);
options = gaoptimset(options,'MutationFcn', @mutationgaussian [] [] );
options = gaoptimset(options, 'Display', 'off');
% options = gaoptimset(options, 'PlotFcns', { @gaplotbestf });
[x, fval, output, population] = ga(@fitnessFct, nvars, [ ], [ ], [ ], [ ], [ ], [ ], [ ], [ ], options);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The MVEE in high dimensions.
% we should generate appropriate population of ellipsoids
function [MatrixHighEllipsoid]=High_Dim_Ellipsoid(n, m, n1, e, lb, ub)
close all;
PP=load('Border_Pointsfile.txt');
NumPoints=size(PP, 1);
if m==3
    hold on
    view(3);
    plot3(PP(:, 1), PP(:, 2), PP(:, 3), '.r');
    K2 = convhull(PP(:, 1), PP(:, 2), PP(:, 3), 'simplify', true);
    trisurf(K2, PP(:, 1), PP(:, 2), PP(:, 3), 'Facecolor', 'cyan');
end
% Default values
nPhi = 64; nTheta = 32;
result= zeros(n, (m*2)+1);
Phi=0;
% Generate points for the initial ellipsoids
for i=1: n
    Ps=zeros(m, 2);
    centres=zeros(m, 1);
    D=zeros(m, 1);
    for j=1: m
        if m < 16
            s1=(ub-(lb))*rand(1); s2=(2*ub)*rand(1); p=[s1, s2];
            s1=1+(ub-lb)*rand(1); s2=(ub-lb)*rand(1)+2*ub; p=[s1, s2];
        end
        if m> =16
            s1=rand(1)*lb/2-(lb/2); s2=(2*ub)*rand(1)+lb/2; p=[s1, s2];
        end
        Ps(j, :)=p;
        xc=(p(:, 1)+ p(:, 2))/2;
        centres(j, :)=xc;
        d=(p(:, 1)-p(:, 2)) .^ 2;
        D(j, :)=d;
```

```
    end
    C=centres';
    MajorAxis=sqrt(sum(D));
    a=MajorAxis/2;      MinorAxis=zeros(m-1, 1);
    for j=1: m-1
        b=1+rand(1)*2;
        Minor=a/b;
        MinorAxis(j, :)=Minor;
    end
    L=[a, (MinorAxis)'];
% Ellipsoids before rotate
    theta = linspace(0, pi, nTheta+1);
    phi = linspace(0, 2*pi, nPhi+1);
% Writing the parameters of high dimensions
    S=zeros(m-2, nTheta+1);
    for j=1: m-2
        Angl=sin(theta);
        S(j,:)=Angl;
    end
% Parameter for m>=4
    xm = sin(phi')*prod(S);
    x3 = sin(phi')*S(1, :);%y
    x2 = cos(phi')* sin(theta);%x
    x1 = ones(length(phi), 1)*cos(theta);%z
    X=zeros(nPhi+1 ,nTheta+1, m-3);
    Res=[ ];
    for j=1: m-2
        Angl=sin(theta);
        SS(j, :)= Angl;
        if j > =2
           S1= prod(SS);
           X1 = cos(phi')* S1 ;
           X(:, :, j)=X1;
           Res=[Res, X(:, :, j)];
        end
    end
% Scaling matrix
```

```
    L1=[L, 1];
    Sc=zeros(m+1, m+1);
    for j1=1: m+1
        Sc(j1, j1)=L1(:, j1);
    end
% Translate the centre
    tra = createTransBox(m, C);
% General formula for rotating matrix
    Phi=Phi+20;
    for j=1: m
        Angle=(Phi*pi / 180)*rand(2, m);
        Rotate=rotate(Angle);
    end
    Rotate(m+1, m+1)=1;
    trans = tra* Rotate * Sc;
% Transformation the ellipsoids for dim >4
    if m>3
        v=[x1, x2, Res, xm];
        save('vfile.txt', 'v', '-ascii');
        e1=transformPointAny(m, v, trans);
    end
% Volume
    mm=(m/2)+1;
    mm=ceil(mm);
    if m==3
        vol=(4/3)*pi*prod(L);
    else
        vol=((pi ^ (m/2))*prod(L)/factorial(mm));
    end
% transformation the ellipsoids in 3dimensions
    if m==3
        v=[x1, x2, x3];
        e1=transformPointAny(m, v, trans);
        x1=e1(:, 1: nTheta+1);
        x2=e1(:, nTheta+2: (nTheta+1)*2);
        x3=e1(:, ((nTheta+1)*2)+1: (nTheta+1)*3);
    end
```

```
% Check the points inside the ellipsoid
    XY=zeros(NumPoints, m);
    for j=1: m
        xy=((PP(:, j)- C(:, j))/ L(:, j)) . ^ 2;
        XY(:, j)=xy;
    end
    W=(1 >= sqrt(sum(XY')));
    nbPointsInside=sum(W);
    if (i==1)
        minvol=vol;
            if(nbPointsInside==NumPoints)
                optimalVol=minvol;
        end
    end
    if (minvol<vol)
        minvol=vol;
            if(nbPointsInside==NumPoints)
                optimalVol=minvol;
            else
                continue;
            end
    end
    result(i, 1: m)=C;
    result(i, m+1: m4*2)=L;
    result(i, (m*2)+1)=Phi;
    result(i, (m*2)+2)=vol;
    result(i, (m*2)+3)=nbPointsInside;
    fitnessvalueEll=nbPointsInside-vol;
    result(i, (m*2)+4)=fitnessvalueEll;
end
%Result; sortedRes= sortrows(result, (m*2)+4);
MatrixHighEllipsoid=sortedRes(:, 1: (m*2)+1);
MatrixHighEllipsoid(MatrixHighEllipsoid(:, :)==0) = 25;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation file of MVEE in high dimensions
clc; clear; close all;
```

```
% For m=3, n=70, e=0.4, bounds=10-15, Ge=200; for m=15, n=60, e=0.1, bounds=10-15,
Ge 100.
n=Initial ellipsoids; m=Dimensions; n1=Points; e=Rate ϵ; Ge=Generations;
lb=Lower bound; ub=Upper bound;
nvarb=(m*2)+1;
[Border_Points]=Generate_Points(m, n1, e, lb, ub)
[MatrixHighEllipsoid]=HigherDimEllipsoid(n, m, n1, e, lb, ub);
MatrixHighEllipsoid(MatrixHighEllipsoid(:, :)==0)=25;
[z, fval, output, population]=optim(nvarb, n, Ge, MatrixHighEllipsoid);
if m==3
    figure;
    Plot3Dim(z);
end
% Check the points outside the ellipsoid and add them to inside the ellipsoid
for i=1: 2
    C=abs(z(:, 1: m));
    L=abs(z(:, m+1: m*2));
    Border_Points=load('Border_Pointsfile.txt');
    NumPoints=size(Border_Points, 1);      mm=(m/2);
    mm=ceil(mm);
    if m==3
        vol=(4/3)*pi*prod(L)
    else
        vol=((pi ^ (m/2))4*prod(L)/factorial(mm))
    end
    p= load('pfile.txt');
    p1=size(p, 1);
    XY=zeros(p1, m);
    for j=1: m
        xy=((p(:, j)- C(:, j)) / L(:, j)). ^ 2;
        XY(:, j)=xy;
    end
    W1=(1 >= sqrt(sum(XY'))); nbPointsInside=sum(W1);
    W=(1 < sqrt(sum(XY'))); nbPointsOutside=sum(W);
    k=find(W==1);
    k1=p(k, :);
    Border_Points=[Border_Points; k1];
```

```
    PP1=Border_Points;
    save('Border_Pointsfile.txt', 'Border_Points', '-ascii');
    if nbPointsOutside > 1
        if m==3
        Ge=150;
    else        Ge=150;
    end
    [z, fval, output]=optim(nvarb, 70, Ge, MatrixHighEllipsoid);
    if m==3
        figure;
        Plot3Dim(z);
    end
    else
        outsidePoints=size(k1, 1)
        SizeOutsideAndBorder=size(PP1,1)
        break;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fitnessvalueEll]=fitnessHighDim(z)
m=(size(z, 2)-1)/2;
C=abs(z(:, 1: m));
L=abs(z(:, m+1: m*2));
Border_Points=load('Border_Pointsfile.txt');
NumPoints=size(Border_Points, 1);
mm=(m/2);
mm=ceil(mm);
if m==3
    vol=(4/3)*pi*prod(L);
else
    vol=((pi ^ (m/2))*prod(L)/ factorial(mm));
end
XY=zeros(NumPoints, m);
    for j=1: m
        xy=((Border_Points(:, j)- C(:, j))/ L(:, j)) . ^ 2;
        XY(:, j)=xy;
    end
```

```
W=(1 >= sqrt(sum(XY')));
nbPointsInside=sum(W);
if(nbPointsInside<NumPoints)
fitnessvalueEll=(NumPoints-nbPointsInside + 100+ vol) ^ 2 ;
else
fitnessvalueEll=(vol-nbPointsInside);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Trans = createTransBox(m, C)
if (nargin<1)
    for j=1: m
        dx(j)=0;
    end
else
% Translation vector given in a single argument
    var=C;
    for j=1: m
        dx(j)=var(j);
    end
end
% Create the translation matrix
Trans=eye(m+1, m+1);
Trans(1: m ,m+1)=dx;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [e1] = transformPointAny(m, v, trans)
nPhi =64; nTheta =32;
F=[ ];
for j=1: m
    B= v(:, ((nTheta+1)*j)-nTheta: (nTheta+1)*j);
    vv=B(:);
    F=[F, vv];
end
NP = numel(vv);
try
F=[F ones(NP, 1)];
```

```
res =F * trans';
e1=[ ];
for j=1: m
    ee=reshape(res(:, j), nPhi+1, nTheta+1);
    e1=[e1, ee];
end for i = 1:NP
    vv(i); end catch ME
disp(ME.message)
end
% Process output arguments
dim=size(B);
dim(2) = 1;
if nargout <= 1
% Results are stored in a unique array
if length(dim) > 2 && dim(2) > 1
    warning('geom3d:shapeMismatch', ...
    'Shape mismatch: Non-vector xyz input should have multiple x, y, z
    output arguments. Cell {x, y, z} returned instead.')
    for j=1: m
        varargout{ j } = {e1(j)};
    end
else
    varargout{ j } = [e1(j)];
end
elseif nargout == m
    for j=1: m
        varargout{ j } = e1(j);
    end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function result = rotate(Angle)
[m, n] = size(Angle);
result = eye(n);
for i=1: m
    for j=1: n
        if i ~= j && Angle(i, j)  = 0
```

```
            tm = eye(n);
            tm(i, i) = cos(Angle(i, j));
            tm(j, j) = tm(i, i);
            tm(i, j) = -sin(Angle(i, j));
            tm(j, i) = -tm(i, j);
            result = result*tm;
        end
    end
end
```