COMPLEXITY ANALYSIS OF ITERATED LOCAL SEARCH ALGORITHM IN EXPERIMENTAL DOMAIN FOR OPTIMIZING LATIN HYPERCUBE DESIGNS

By

Parimal Mridha

Roll No-1051552

A Thesis submitted in the partial fulfillment of the requirements for the degree of Master of Philosophy in Mathematics

Khulna University of Engineering & Technology
Khulna 9203, Bangladesh

August, 2013

Declaration

This is to certify that the thesis work entitled "*Complexity Analysis of Iterated Local Search Algorithm in Experimental Domain for Optimizing Latin Hypercube Designs* `` has been carried out by ***Parimal Mridha*** in the Department of *Mathematics,* Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of thesis work has not been submitted anywhere for the award of any degree or diploma.

[illegible]

....

Signature of Supervisor

Name: Dr. A. R. M. Jalal Uddin Jamali

Designation: Professor of Mathematics, KUET.

[illegible]

....

Signature of Candidate

Name: Parimal Mridha

Designation: M. Phill. Student of Mathematics, KUET.

Approval

This is to certify that the thesis work submitted by ***Parimal Mridha*** entitled "***Complexity Analysis of Iterated Local Search Algorithm in Experimental Domain for Optimizing Latin Hypercube Designs***" has been approved by the board of examiners for the partial fulfillment of the requirements for the degree of ***Master of Philosophy*** in the Department of *Mathematics*, Khulna University of Engineering & Technology, Khulna, Bangladesh in August, 2013.

Dedication

To my beloved Son

Loknath Mridha

Acknowledgement

First of all I wish to express my devotion and reverence to the Supreme Being, the creator and the ruler of the universe, who enables me to complete the thesis for the fulfillment of the degree of Master of Philosophy in Mathematics.

I would like to express my deepest sense of gratitude and respect to my honorable teacher and Supervisor Dr. A. R. M. Jalal Uddin Jamali, Professor, Department of Mathematics, Khulna University of Engineering & Technology, for his continuous encouragement, sincere and regular advice to me for the research works. Without his kind supervision, I could not complete this thesis.

I wish to express my profound gratitude to authority of the Khulna University of Engineering & Technology for accepting me in M. Phil. program. Its outstanding learning environment helped me lay a firm academic foundation.

I am also grateful with warm appreciation and great indebtedness for the support, guidance and suggestion to me by all of my respected teachers in Mathematics Department, Khulna University of Engineering & Technology, Khulna. They helped me whenever I needed assistance regarding administrative as well as other academic necessities. I would like to give sincere gratitude and proper respect to my teacher Prof. Dr. Mohammad Arif Hossain who helped me in different phases of my M. Phil Course.

I would specially be grateful to my elder brother Mr. Birendro Nath Mridha who always inspires me for my study and always wants to see the word "Dr." before my name.

I give heartfelt thanks to my mother and my wife who gave me mental strength in my frustrations.

Abstract

Computer experiments involve a large numbers of variables, but only a few of them have no negligible influence on the response. As is recognized by several authors, the choice of the design points for computer experiments should fulfill at lest two requirements – space-filling and non-collapsing. Unfortunately, randomly generated Latin Hypercube Designs (LHDs) almost always show poor space-filling properties. On the other hand, maximin distance designs have very well space-filling properties but often show poor projection properties under the Euclidean or the Rectangular distance. To overcome this shortcoming, Morris et al. have suggested to search for maximin LHDs when looking for "optimal" designs. It is shown that the Iterated Local search(ILS) approach not only able to obtain good LHDs in the sense of space-filling property but the correlations among the factors are acceptable i.e. multi-collinearity is not high. Anyway from the point of view of computational complexity the problem is open. When number of factors or number of design points is large then it requires hundreds of hours by the brute-force approach to find out the optimal design. So when numbers of factors as well as number of experimental points are large, the heuristic approaches also require a couple of hours or even more to find out a simulated optimal design. So time complexity is an important issue for a good algorithm. Specially for the need of real time solution, the time complexity of the ILS approaches is analyzed. The inner most view as well as the effect of the parameters of the algorithms have been observed and have been analyzed. After analyzing, the time complexity model of the algorithms for two optimal criterion namely Opt (D_1, J_1) as well as Opt(Φ) has been developed. More over some experiments have been performed for higher dimension namely dimensions $k > 10$. Some new maximin LHDs value are obtained from these experiments, as there are few maximin LHDs value available in the literatures for higher dimension, $k > 10$. From these experiments, multi-collinearity property, maximin LHDs in Rectangular distance, mimimal Φ values, maximum pair-wise distance value of LHDs etc. are represented in this thesis.

Contents

PAGE

Title Page i
Declaration ii
Approval iii
Dedication iv
Acknowledgement v
Abstract vi
Contents vii
List of Tables ix
List of Figures x

CHAPTER 1 Introduction 1

1.1 Background 1
1.2 Literature Review 2
1.3 Goal of Thesis 4
1.4 Structure of the Thesis 6

CHAPTER 2 Overview of Complexity 7

2.1 Introduction 7
2.2 Some Definitions 8
2.3 Time Complexity 9
2.3.1 Constant Time Complexity 10
2.3.2 Types of Time Complexity 11
2.4 Measuring Time Complexity 13
2.5 Big-O Notation 14

CHAPTER 3 Iterated Local Search Approach for Maximin Latin Hypercube Designs 18

3.1 Introduction 18
3.2 Iterated Local Search 18
3.3 Maximin Latin Hypercube Designs 23

3.4 Definition of LHD 23

3.5 Optimality Criteria 25

3.6 ILS heuristic for maximin LHD 27

3.6.1 Initialization (I_S) 28

3.6.2 Local Search Procedure (L_S) 28

3.6.3 Local Moves (L_M) 29

3.6.4 Acceptance Rule 30

3.6.5 Perturbation Move (P_M) 31

3.6.6 Stopping Rule (S_R) 33

CHAPTER 4 Complexity analysis of the ILS Algorithms in Experimental Domain: Consider ILS(D_1) 34

4.1 Introduction 34

CHAPTER 5 Complexity Analysis of the Algorithm in Experimental Domain : Consider ILS(Φ) 46

CHAPTER 6 Computational experiment of ILS of Higher dimensions 56

REFERENCES 65

LIST OF TABLES

Table No.	Caption of the Table	Page
3.1	Some well know approaches as well as optimal criterion for optimal experimental designs	27
4.1	Pseudo code of the ILS (D_1 ,J_1) algorithm	34
4.2	Analysis of time complexity for ILS (D_1, J_1)	45
5.1	Pseudo code of the ILS(Φ) algorithm	46
5.2	Analysis of time complexity for ILS ($\boldsymbol{\Phi}$)	55
6.1	The setting of number of runs for the ILS	56
6.2(a)	Comparison of maximin LHD values for dimensions $k = 11$	57
6.2(b)	Comparison of maximin LHD values for dimensions $k = 12$ and 13	58
6.2(c)	Comparison of maximin LHD values for dimensions $k = 14$ and 15	59
6.3	Comparison of maximin LHD values and Φ values for dimensions $k = 11$	60
6.4	Experimental results of maximnin L^1 values corresponding to optimized LHD values measured in L^2 measure for $k = 11 - 15$	61
6.5	Experimental results of maximum average coefficient of correlation of the co-factors of the maximin LHDs for $k = 11 - 15$	62
6.6	Experimental results of maximum pair-wise distance value (L_M^1 and L_M^2) of the maximin LHDs for $k = 11 - 15$	64

List of Figures

Figure No	Caption of the Figure	Page
2.1	Graphical representation of some functions in point of time complexity	15
2.2	Schematic view of the field of complexity	17
3.1	Some LHDs and their corresponding (D_1, J_1) values	24
3.2	Illustration of Neighborhood solutions for $LM_{R}p_{D1}$ based local search (LS) procedure	30
4.1	The percentage of pairs involving and not involving critical points	36
4.2	The history of number of critical points for (k,N) = (7, 50) during Local Move	37
4.3	The impact of N on Maximum Critical Points during history of evaluation	37
4.4	The impact of N on average Critical Points during history of evaluation	38
4.5	The history of WL for (k, N) = (7, 20) and (k, N) = (7, 50) during Local Search	38
4.6	The impact of N on (a) Maximum WL (b) Average WL during history of Local Search	39
4.7	The Impact of k on AWL during Local Search	39
4.8	The History of Elapsed time	40
4.9	The values of log(T) plotted against log(N)	41

4.10 The approximate time complexity for LS with respect to *k* 42

4.11 The impact of *N* on the number of perturbations 43

4.12 The impact of *k* on the number of perturbations 44

5.1 The history of WL values for (k, N) = (7, 10) and (k, N) = (7, 50) during Local Search 47

5.2 The impact of *N* on (a) MWL (b) AWL 48

5.3 The Impact of *k* on AWL 49

5.4 The Impact of *k* on execution of AWL during Local Search with FI (First Improve) in Opt(D_I, J_I) 50

5.5 Elapsed time per local search as a function of *N* 51

5.6 Linear regression between log(T) and log(N) 51

5.7 Impact of *k* on *T* 52

5.8 The approximate time complexity of *k* for LS obtained by the experiments 52

5.9 Relation between the number of perturbations and *N* 53

5.10 Impact of *k* on the number of perturbations 54

CHAPTER I

INTRODUCTION

1.1 Background

The design of computer experiments has much recent interest and this is likely to grow as more and more simulation models are used to carry out research and also made it clear that many simulation models involve several hundred factors or even more. Computer simulation experiments are used in a wide range of application to learn about the effect of input variables $\boldsymbol{x}$ on, a response of interest, y [Butler (2001)]. In computer experiments, instead of physically doing an experiment on the product, mathematical models, describing the performance of the product, are developed using engineering laws/physical laws and solved on computers through numerical methods [Morris (1991)]. Computer experiments involve a large number of variables, but only a few of them have no negligible influence on the response (sparsity assumption) [Jourdan et al. (2010)]. Since the computer response is deterministic, it is desirable to avoid replication, in particular when projecting the design on to a subset of variables (non-collapsing). As is recognized by several authors, the choice of the design points for computer experiments should fulfill at least two requirements - *space-filling* and *non-collapsing* [Johnson et al. (1990), Morris and Mitchell (1995)]. Space-filling property ensures unbiased selection of the design points whereas non-collapsing property provided influence of individual effect on response variable.

For the design of computer experiments **Latin Hypercube Designs (LHDs)** fulfill the *non-collapsing* property. Such design, firstly introduced in 1979 by McKay and his colleagues has proved to be a popular choice for experiments run on computer simulators [Levy et al. (2010)] and in global sensitivity analysis [Helton and Davis (2000), Steinberg and Dennis (2006)]; Assume that N design points have to be placed and that there are k distinct parameters. It would be done such that the points will uniformly spread when projected along each single parameter axis. It is also assumed that each parameter range is normalized to the interval [0, N-1]. Then, a LHD is made up by N points, each of which has k integer coordinates with values in 0, 1, . . . , N-1 and such that there do not exist two points with one common coordinate value. This allows a non-collapsing design because points are evenly spread when projected along a single parameter axis. Note that the number of possible LHDs is huge: there are $(N!)^k$ possible LHDs (where N is number of design points and k is number of factors). Anyway the main attraction of these designs is the one-dimensional projective property. The one-dimensional projective property ensures that there is little redundancy of design points when some of the factors have a relatively negligible effect (sparsity principle).

Unfortunately, randomly generated LHDs almost always show poor *space-filling* properties or / and the factors are highly correlated. On the other hand, **maximin distance** designs, proposed by [Jonson et al. (1990)] have very good *space-filling* properties but often no good projection properties under the Euclidean or the Rectangular distance. To overcome this shortcoming, Morris and Mitchell (1997) suggested to search for **maximin LHDs** when looking for "optimal" designs. In the literature the optimal criterion for maximin LHD are defined in several ways [Grosso et al. (2008)] but the main objective is identical i.e. searching the LHD with the maximizing the minimum pair-wise distance.

1.2 Literature Review

Different methods have been presented in the literature to detect maximin LHDs. Morris and Mitchell (1995) have been proposed a simulated annealing. Li and Jeff (1997) have been proposed a class of algorithms based on column pair-wise exchange to build supersaturated designs. Ye et al. (2000) have been presented an exchange algorithm for finding approximate maximin LHDs with the further restriction to Symmetric LHDs (SLHDs). In order to reduce the number of simulations needed to achieve the desired accuracy Crombecq et al. (2011) have proposed sequential simulation–based method. General formulae for maximin LHDs with $k = 2$ are given by Dam (2005) for the 1-norm (ℓ^1) and infinite norm (ℓ^∞) distances, while for the Euclidean distance(ℓ^2) maximin LHDs up to $N = 1000$ design points are obtained by (adapted) periodic designs, while, using a branch-and-bound algorithm, exact solutions have been obtained for N up to 70. Inspired by Dam (2005), Husslage et al. (2006) proposed (adapted) periodic designs and simulated annealing to extend the known results and construct approximate maximin Latin Hypercube Designs (LHDs) for k up to 10 and N up to 100. All these designs are available in the website http:// www.spacefillingdesigns.nl. Husslage et al. (2006) has shown that the periodic heuristic tends to work when the number N of design points gets above some threshold which depends on the dimension k of the design (more precisely, such threshold increases with k), Grosso et al. (2009) successfully implemented Iterated Local Search (ILS) approach for finding maximin LHDs for $k = 3, 4, \ldots, 10$, and $N = 3, 5 \ldots\ldots, 100$. Iterated Local Search (ILS) [Lourenco et al. (2002)] is a meta-heuristic designed to embed another, problem-specific, local search as if it was a black box. This allows Iterated Local Search to keep a more general structure than other meta-heuristics currently in practice. This simple type of search has been reinvented numerous times in the literature, with one of its earliest incarnations appearing in Lin and Kernighan (1973). The rationale behind ILS is supported by the proximate optimality principle [Glover et al. (1997)]. This principle assumes that good solutions are similar. This assumption is reasonable for most real-world problems.

A lot of improved values (maximin LHD values) are obtained by ILS approach proposed by Grosso et al. (Grosso et al. (2009). The improved values are available in the well known web portal http:// www.spacefillingdesigns.nl. But the algorithm was implemented in *Sun-Cluster Mainframe* environment. It is worthwhile to mention here that under the project 2010-2011 provided by CASR (Committee of Advanced Studies and Research), KUET, the algorithm (Grosso et al. (2009) is partially implemented the in the PC windows environment. Also authors, in [Jamali et al. (2008) and Aparna D. (2012)], analyzed the muliti-collinearity of the maximin LHD obtained by the ILS approach. In [Jamali et al. (2010)], it is shown that the ILS approach not only able to obtain good LHD in the sense of space-filling property but the correlations among the factors are acceptable. Moreover in this article they also discuss about rectangular (ℓ^1) coordinate measure.

From the above discussion it may conclude that ILS approach is a state-of-art method to find out the maximin LHD based experimental design. From the point of view of computational complexity the problem is, to the authors' knowledge, open (but suspected to be NP-complete) [Grosso et al. (2009)]. So when numbers of factors as well as number of experimental points are large, the algorithm required a couple of hours even more to find out a simulated optimal design. So time complexity is an important issue for a good algorithm; especially when we need a real time solution. Therefore the time complexity of the ILS approach [Grosso et al. (2009)] should be analyzed. But as ILS is a Mata-heuristic approach (several random parameters are implemented), it is impossible to analyze the time complexity of the algorithm theoretically.

It is remarked that the maximin criterion is not the only one used in the literature, but also other criteria like the maximum entropy [Currin et al. (1991, Jin et al. (2005), Park (1994), Shewry and Wynn (1987)], the integrated mean squared error [Sacks et al.(1987), Park (1994), Crary (2002)], the minimum correlation between components [Iman and Helton (1985), Owen (1994)] and a mixed criterion involving both maximin distance and correlation [Joseph and Hung (2008)] are used. Bates et al. (2003) by considering minimization of potential energy which is called Audze–Eglais Uniform Latin Hypercube design. [Fuerle and Sienz (2011)], obtained Optimal Latin Hypercubes design by minimizing the Audze–Eglais potential energy of the points using a permutation genetic algorithm. The book of Santner et al. (2003) and the article of Levy and Steinberg (2010) are referred for more details. Moreover some important optimal objective function (Criterion) as well as methods will be given is chapter 3 in tabular form.

1.3 Goal of Thesis

It has been already mentioned above that the design of computer experiments has much recent interest and this is likely to grow as more and more simulation models are used to carry out research. Morris (1991) and Kleijnen (1997), it is also made clear that many simulation models involve several hundred factors or even more. So, for the presents of combinatorial nature, the number of possible LHDs is very high - $(N!)^k$. Consequently, when number of factors and/ or number of design points are large then it requires hundreds of hours by the brute-force approach to find out the optimal design.

The intrinsic difficulty of the problem and/or the limited availability of computation time for the particular application from which the problem arises (think, e.g., about real-time applications, where solutions are required in very short times) may make computationally infeasible to return an optimal solution by the required time. The importance of high performance algorithms for tackling difficult optimization problems can not be understated, and in many cases the only available methods are meta-heuristics like ILS approach. The main objectives of the project are point out below.

- As mention earlier that ILS approach is a meta- heuristic based algorithm and as the complexity analysis regarding theoretical point of view is almost impossible, here the time complexity of the algorithm will be analyzed extensively in experimental point of view.
- As in ILS approach, has considered two optimal criterion. The time complexity of ILS approach for both optimal criterion are measured separately in experimental points of view.
- As in Grosso et al. (2009) used ILS approach for $k \leq 10$ and obtain remarkable results which is updated in the well known web portal www.spacefillingdesigns.nl. So the performance of the ILS approach for the dimension $k \rangle 10$ is experimentally studied.
- The multi-colliniarity property, optimal Φ value, maximin LHD value in rectangular distance measure and maximum pair-wise distance are presented in the thesis.

The algorithm has two variables name number of factors (k) and number of design points (N). Moreover there are several parameters as well as random variables (parameters) exist in the algorithm. These parameters are vary according to the change of the variables. So extensive experiments will be performed by fixing each variable and counting the number of iterations as well as CPU times. We will try to graphically represent the inner- most view of the algorithm for the changing of variables as well as parameters. After analyzing, it will be tried to develop time complexity model of the algorithms.

Though finding the optimal LHD in brute-force approach is $(N!)^k$, but we expected, in ILS approach, this will be polynomial time with low order. It will be worthwhile to mention here that the solution obtained by the ILS approach must not be guaranteed to be optimal one rather it may be approximately optimal. It is also expected that function based approach also work well. Finally it is expected that structural analysis as well as theoretical analysis may build the method as well as experimental design much more stronger. Besides some complexity analysis, several experiments will be performed for dimension $k \rangle 10$. Then the results will be compared available one in the literature.

1.4 Structure of the Thesis

After the **chapter I** in which the literature review as well as introduction of the research work is presented, the concept of Complexity is briefly discussed in **chapter II**. In **chapter III**, the ILS algorithms are presented. In **chapter IV**, the complexity of the algorithms regarding Opt(D_1,J_1) optimal criterion is discussed from experimental point of view. Also In **chapter V**, the complexity of the algorithms regarding Opt (Φ) optimal criterion is discussed from experimental point of view. Several experiments of higher dimensions are presented in **chapter VI**. In this chapter we have observed multicillinearity property, rectangular distance measure and maximum pair-wise distance of LHD for maximinn LHD. Moreover we also observed the track of LHD for minimization of Φ criterion.

CHAPTER II

OVERVIEW OF COMPLEXITY

2.1 Introduction:

In general usage, **complexity** tends to be used to characterize something with many parts in intricate arrangement. Complexity Theory is concerned with the study of the *intrinsic complexity* of computational tasks. Its ``final" goals include the determination of the complexity of any well-defined task. Additional ``final" goals include obtaining an understanding of the relations between various computational phenomena (e.g., relating one fact regarding computational complexity to another). Indeed, we may say that the former type of goals is concerned with *absolute* answers regarding specific computational phenomena, whereas the latter type is concerned with questions regarding the *relation* between computational phenomena.

Interestingly, the current success of Complexity Theory in coping with the latter type of goals has been more significant. In fact, the failure to resolve questions of the ``absolute" type, led to the flourishing of methods for coping with questions of the "relative" type.

In general, Computational complexity theory is a branch of the theory of computation in theoretical computer science and mathematics that focuses on classifying computational problems according to their inherent difficulty, and relating those classes to each other. A computational problem is understood to be a task that is in principle amenable to being solved by a computer, which is equivalent to stating that the problem may be solved by mechanical application of mathematical steps.

The important aspects of this approach are that [Grassberger et al.(2013)]:

- it applies to models rather than natural systems;
- complexity is distinguished from ignorance;
- it is relative to the modelling language it is expressed in;
- it relative to the identification of components and overall behaviour;
- complexity is a global characteristic of a model;
- we will get different kinds of complexities from different types of difficulty;

- complexity represents the gap between component knowledge and knowledge of global (or emergent) behaviour;
- since difficulty is sometimes comparative, complexity will be also

Anyway there are two type of complexity regarding time and space. Time complexity is concerned with the analysis of the elapsed time of an algorithm; whereas, how much memory required is discussed in space complexity.

2.2 Some Definitions:

Some of these concept and respective are offered below:

(i) Time Complexity: A measure of the amount of time required to execute an algorithms is called time complexity. Later time complexity is discussed elaborately.

(ii) Space Complexity: The (space) complexity of a program (for a given input) is the number of elementary objects that this programs needs to store during its execution. This number is computed with respect to the size n of the input data. We thus make the assumption that each elementary object needs the same amount of space.

(iii) Turing Machine: Turing machines provide a model of digital computational which is more primitive, hence harder to 'program" than random access machines. However, their primitiveness becomes an advantage when they are manipulated for the purpose of proving theorical results.

A *Turing machine* is a hypothetical device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer.

(iv) Worst-case and Average case: Worst case analysis is used to find an upper bound on algorithm performance for large problems (large n). Average case analysis determines the average (or expected) performance.

Worst-case performance analysis and average case performance analysis have some similarities, but in practice usually require different tools and approaches. Determining what *average input* means is difficult, and often that average input has properties which make it difficult to characterise mathematically (consider, for instance, algorithms that are designed to operate on string of text). Similarly, even when a sensible description of a particular "average case" (which will probably only be applicable for some uses of the algorithm) is possible, they tend to result in more difficult to analyze equations.

Worst-case analysis has similar problems- it is typically impossible to determine the exact worst-case scenario. Instead, a scenario is considered such that it is at least as bad as the worst case. For example, when analyzing an algorithm, it may be possible to find the longest possible path through the algorithm (by considering the maximum number of loops for instance) even if it is not possible to determine the exact input that would generate this path (indeed, such an input may not exist). This gives a *safe* analysis (the worst case is never underestimated), but one which is *pessimistic*, since there may be no input that would require this path.

Alternatively, a scenario which is thought to be close to (but not necessarily worse than) the real worst case may be considered. This may lead to an *optimistic* result, meaning that the analysis may actually underestimate the true worst case.

When analyzing algorithms which often take a small time to complete, but periodically require a much larger time amortized analysis can be used to determine the worst-case running time over a (possibly infinite) series of operations. This **amortized worst-case** cost can be much closer to the average case cost, while still providing a guaranteed upper limit on the running time. The worst-case time complexity is usually simpler to work out.

2.3 Time Complexity: Time Complexity comparisons are more interesting than space complexity. The programming language chosen to implement the algorithm should not affect in time complexity analysis. There are some other factors that should not affect in time complexity are-: the quality of the compiler, the speed of the computer on which the algorithm is to be executed.

The objectives of the time complexity analysis are to determine the feasibility of an algorithm by estimating an upper bound on the amount of work performed. Objectives of the time complexity analysis are also to compare different algorithms before deciding on which one to implement.

Time complexity analysis is based on the amount of work done by the algorithm. It expresses the relationship between the size of the input and the run time for the algorithm. Time complexity is usually expressed as proportionality, rather than an exact function.
To simplify analysis, we sometimes ignore work that takes a constant amount of time, independent of the problem input size. When comparing two algorithms that perform the same task, we often just concentrate on the differences between algorithms.
For time Complexity, simplified analysis can be based on:

(i) Number of arithmetic operations performed

(ii) Number of comparisons made

(iii) Number of times through a critical loop

(iv) Number of array elements accessed, etc.

2.3.1 Constant Time Complexity:

Algorithms whose solutions are independent of the size of the problem's inputs are said to have constant complexity. It is denoted as 1(O).

Example:

Suppose that exponentiation is carried out using multiplications. Two ways to evaluate the polynomial

$$P(x) = 4x^4 + 7x^3 - 2x^2 + 3x^1 + 6$$

In Brute force method:

$$P(x) = 4 * x * x * x * x + 7 * x * x * x - 2 * x * x + 3 * x + 6$$

In Horner's method:

$$P(x) = (((4 * x + 7) * x - 2) * x + 3) * x + 6$$

Method of Analysis:

(i) Basic arithmetic operations are multiplication, addition, and subtraction.

(ii) We'll only consider the number of multiplications, since the number of addition and subtractions are the same in each solution.

(iii) We'll examine the general form of a polynomial of degree n, and express our result in terms of n.

(iv) We'll look at the worst case (max number of multiplications) to get an upper bound on the work.

2.3.2 Types of Time Complexity:

There are many different types of complexity involved in actual examples of scientific modelling. Conflation of these into a single "complexity" of scientifically modelling a certain system will generally result in confusion.

There might be:

- The complexity of the data: the difficulty of encoding of a data model compactly given a coding language;
- The complexity of the informal (mental) model: the difficulty in making an informal prediction from the model given hypothetical conditions;
- The complexity of using the formal model to predict aspects of the system under study given some conditions;
- The complexity of using the formal model to explain aspects of the system under study given some conditions.

Each of these will be relative to the framework it is being considered in (although this and the type of difficulty may be implicit).

Many important complexity classes can be defined by bounding the time or space used by the algorithm. Some important complexity classes of decision problems defined in this manner are the following:

Complexity class	Model of computation	Resource constraint
DTIME($f(n)$)	Deterministic Turing machine	Time $f(n)$
P	Deterministic Turing machine	Time poly(n)
EXPTIME	Deterministic Turing machine	Time $2^{\text{poly}(n)}$
NTIME($f(n)$)	Non-deterministic Turing machine	Time $f(n)$
NP	Non-deterministic Turing machine	Time poly(n)
NEXPTIME	Non-deterministic Turing machine	Time $2^{\text{poly}(n)}$
DSPACE($f(n)$)	Deterministic Turing machine	Space $f(n)$
L	Deterministic Turing machine	Space O(log n)
PSPACE	Deterministic Turing machine	Space poly(n)
EXPSPACE	Deterministic Turing machine	Space $2^{\text{poly}(n)}$
NSPACE($f(n)$)	Non-deterministic Turing machine	Space $f(n)$
NL	Non-deterministic Turing machine	Space O(log n)
NPSPACE	Non-deterministic Turing machine	Space poly(n)
NEXPSPACE	Non-deterministic Turing machine	Space $2^{\text{poly}(n)}$

We will discuss following three of them:

(i). Polynomial type Complexity (P type)

(ii). NonPolynomial type complexity (NP type)

(iii). NP-Complete type Complexity.

(i) Polynomial Type Complexity (P type):

P is the class of algorithms whose complexity is a polynomial function of the problems size. Examples include minimum spanning tree algorithms, finding an Eulerian cycle through a graph and bubble sort. In fact most useful algorithms have degree 3 or less.

(ii) NP type Complexity:

NP means non-deterministic polynomial. Suppose a computer program could "guess" a solution to a problem and then could check if the guessed "solution" actually solved the problem and this check could be done in polynomial time, then the program is said to be in the class NP. Non-deterministic is another word for guessing. Most problems are in this class. NP includes P i.e $P \subseteq NP$. Some well known problems in NP are:

- find a Hamiltonian circuit thru a graph
- find all subsets of a set
- travelling salesman problem
- knapsack problem – find the most valuable subset of n items of positive integer weights and values which fit into a knapsack of a given positive integer capacity.
- partition problem – give n positive integers, determine if it is possible to partition them into two disjoint subsets of which have equal sum.
- graph colouring – for a given graph find the smallest number of colours that need to be assigned to the graphs vertices so that no two adjacent vertices share the same colour – called the chromatic number of the graph.

It is an open question in computer science whether problems in NP are in P also i.e. NP = P? If so, there would be a polynomial time algorithm for finding a solution to each problem in

NP. For example is there a polynomial time algorithm that solves travelling salesman problem? No one has been able to prove that there is or is not although it is widely believed that there is not. Believed that $P \subset NP$.

(iii) Class NP-complete:

A problem X is NP-complete if it is in NP and if every other problem in NP both known and unknown can be transformed into X. So by finding a solution to X, we find one to all other problems in NP. The transformation should take polynomial time. The problems listed above in NP are NP-complete. If any of them had polynomial time algorithm as a solution, then so would all the others.

2.4 Measuring Time Complexity:

The worst-case time complexity of an algorithm is expressed as a function

$$T : N \rightarrow N$$

Where T(n) is the maximum number of "steps" in any execution of the algorithm on inputs of "size" n. Intuitively, the amount of time an algorithm takes depends on how large is the input on which the algorithm must operate: Sorting large lists takes longer than sorting short lists; multiplying huge matrices takes longer than multiplying small ones. The dependence of the time needed to the size of the input is not necessarily linear: sorting twice the number of elements takes quite a bit more than just twice as much time; searching (using binary search) through a sorted list twice as long, takes a lot less than twice as much time. The time complexity function expresses that dependence. Note that an algorithm might take different amounts of time on inputs of the same size. We have defined the worst-case time complexity, which means that we count the maximum number of steps that any input of a particular size could take. For example, if the time complexity of an algorithm is $3n^2$, it means that on inputs of size n the algorithm requires up to $3n^2$ steps. To make this precise, we must clarify what we mean by "input size" and "step".

(i) Input Size: We can define the size of an input in a general way as the number of bits required to store the input. This definition is general but it is sometimes inconvenient because it is too low-level. More usefully we define the size of the input in a way that is problem-dependent. For example, when we are dealing with sorting algorithms, it may be more convenient to use the number of elements we want to sort as the measure of the input size. This measure ignores the size of the individual elements that are to be sorted.

Sometimes there may be several reasonable choices for the size of input. For instance, if we are dealing with algorithms for multiplying square matrices, we may express the input size as the dimension of the matrix (i.e., the number of columns or rows), or we may express the input size as the number of entries in the matrix. In this case the two measures are related to each other (the latter is the square of the former). One conclusion from this discussion is that in order to properly interpret the function that describes the time complexity of an algorithm we must be clear about how exactly we measure the size of inputs[Nicolas, (2007)].

(ii) Step: A step of the algorithm can be defined precisely if we fix a particular machine on which the algorithm is to be run. For instance, if we are using a machine with a Pentium processor, we might define a step to be one Pentium instruction. This is not the only reasonable choice: different instructions take different amounts of time, so a more refined definition might be that a step is one cycle of the processor's clock. In general, however, we want to analyze the time complexity of an algorithm without restricting ourselves to some particular machine. We can do this by adopting a more flexible notion of what constitutes a step. In general, we will consider a step to be anything that we can reasonably expect a computer to do in a fixed amount of time. Typical examples are performing an arithmetic operation, comparing two numbers, or assigning a value to a variable.

2.5 Big-O Notation:

Big O notation (with a capital letter O, not a zero), also called **Landau's symbol**, is a symbolism used in complexity theory, computer science, and mathematics to describe the asymptotic behavior of functions. Basically, it tells us how fast a function grows or declines. *Landau's symbol* comes from the name of the German number theoretician Edmund Landau who proposed the notation. The letter O is used because the rate of growth of a function is also called its *order*.

For example, when analyzing some algorithm, one might find that the time (or the number of steps) it takes to complete a problem of size n is given by $T(n) = 4n^2 - 2n + 2$. If we ignore constants (which makes sense because those depend on the particular hardware the program is run on) and slower growing terms, we could say "$T(n)$ grows at the *order* of n^2 and write:

$T(n) = O(n^2)$. In mathematics, it is often important to get a handle on the error term of an approximation. For instance, people will write

$$e^x = 1 + x + x^2/2 + O(x)$$

to express the fact that the error is smaller in absolute value than some constant times x^3 if x is close enough to 0.

For the formal definition, suppose $f(x)$ and $g(x)$ are two functions defined on some subset of the real numbers. We write

$$f(x) = O(g(x))$$

(or $f(x) = O(g(x))$ for $x \to \infty$ to be more precise) if and only if there exist constants N and C such that

$$|f(x)| \le C|g(x)| \quad \text{For all } x>N.$$

Intuitively, this means that f does not grow faster than g.

If a is some real number, we write

$$f(x) = O(g(x)) \text{ for } x \to a$$

if and only if there exist constants $d > 0$ and C such that

$$|f(x)| \le C|g(x)| \quad \text{for all } x \text{ with } |x-a| < d.$$

The first definition is the only one used in computer science (where typically only positive functions with a natural number n as argument are considered; the absolute values can then be ignored), while both usages appear in mathematics.

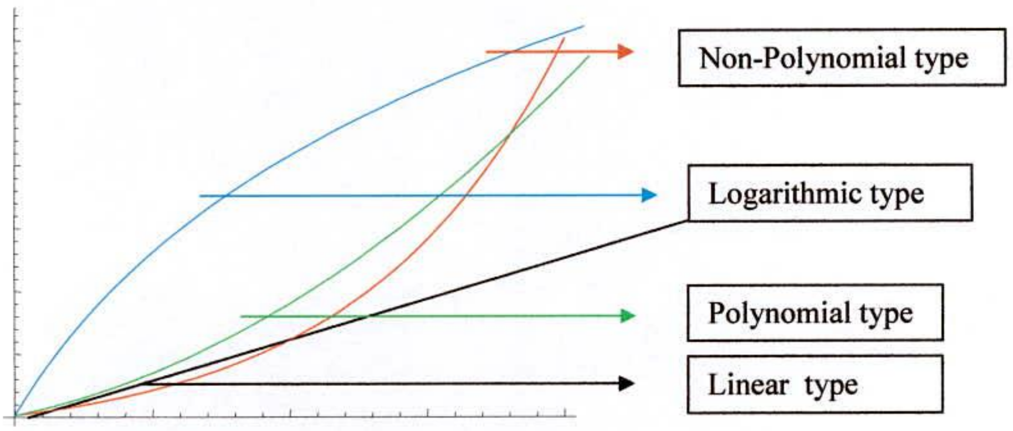

Figure 2.1: Graphical representation of some functions in point of time complexity.

Here is a list of classes of functions that are commonly encountered when analyzing algorithms. The slower growing functions are listed first. *C* is some arbitrary constant.

notation	**name**
$O(1)$	constant
$O(\log(n))$	logarithmic
$O((\log(n)^c)$	polylogarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial
$O(C^n)$	exponential

Note that $O(n^c)$ and $O(C^n)$ are very different. The latter grows much, much faster, no matter how big the constant *C* is. A function that grows faster than any power of *n* is called *superpolynomial*. One that grows slower than an exponential function of the form C^n is called *subexponential*. An algorithm can require time that is both super-polynomial and subexponential; examples of this include the fastest algorithms known for integer.

01. factorization. Note, too, that $O(\log n)$ is exactly the same as $O(\log(n^c)$. The logarithms differ only by a constant factor, and the big O notation ignores that. Similarly, logs with different constant bases are equivalent. The above list is useful because of the following fact: if a function $f(n)$ is a sum of functions, one of which grows faster than the others, then the faster growing one determines the order of $f(n)$. (www.wikipedia.org/w/wiki.phtm?tittle=Big.)

Example: If $f(n) = 10\log(n) + 5(\log(n))^3 + 7n + 3n^2 + 6n^3$ then $f(n) = O(n^3)$

One caveat here: the number of summands has to be constant and may not depend on *n*.

This notation can also be used with multiple variables and with other expressions on the right side of the equal sign. The notation:

$f(n,m) = n^2 + m^2 + O(n + m)$ represents the statement:

$$\exists\, C \ni N \forall n, m > N : f(n,m) \le n^2 + m^3 + C(n+m)$$

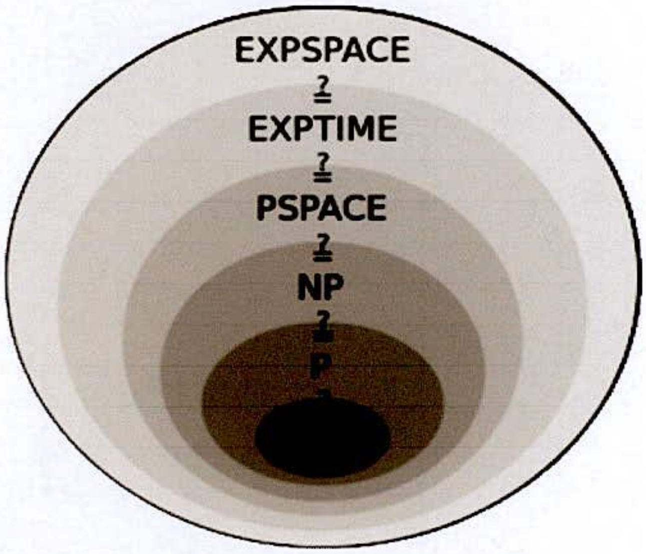

Figure 2.2: Schematic view of the field of complexity

CHAPTER III

ITERATED LOCAL SEARCH APPROACH FOR MAXIMIN LATIN HYPERCUBE DESIGNS

3.1 Introduction

The Latin hypercube design is a popular choice of experimental design when computer simulation is used to study a physical process. These designs guarantee uniform samples for the marginal distribution of each single input. A number of methods have been proposed [Lournce et al.(2002), Martin and Otto(1996) for extending the uniform sampling to higher dimensions. We show how to construct Latin hypercube designs in which all main effects are orthogonal. Our method can also be used to construct Latin hypercube designs with low correlation of first-order and second-order terms. Our method generates orthogonal Latin hypercube designs that can include many more factors than those proposed by Ye [Ye (1998)].

3.2 Iterated Local Search

The importance of high performance algorithms for tackling difficult optimization problems cannot be understated, and in many cases the only available methods are metaheuristics. The word metaheuristics contains all heuristics methods that show evidence of achieving good quality solutions for the problem of interest within an acceptable time. Metaheuristic techniques have become more and more competitive. When designing a metaheuristic, it is preferable that it be simple, both conceptually and in practice. Naturally, it also must be effective, and if possible, general purpose. The main advantage of this approach is the ease of implementation and the quickness.

As metaheuristics have become more and more sophisticated, this ideal case has been pushed aside in the quest for greater performance. As a consequence, problem-specific knowledge (in addition to that built into the heuristic being guided) must now be incorporated into metaheuristics in order to reach the state of the art level. Unfortunately, this makes the boundary between heuristics and metaheuristics fuzzy, and we run the risk of loosing both simplicity and generality.

Here a well known metaheuristics approaches, namely general Iterated Local Search (ILS)has been discussed. Iterated Local Search is a metaheuristic designed to embed another, problem specific, local search as if it were a black box. This allows Iterated Local Search to keep a more general structure than other metaheuristics currently in practice.

The essence of metaheuristic - the iterated local search - can be given in a nut-shell: one iteratively builds a sequence of solutions generated by the embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. This simple idea [Baxter (1981)] has a long history, and its rediscovery by many authors has lead to many different names for iterated local search like *iterated descent* [Baum (1986a), Baum (1986b)], *large-step Markov chains* [Martin et al. (1991)], *iterated Lin-Kernighan* [Johnson (1990)], *chained local optimization* [Martin and Otto (1996)], or *combinations of these* [Applegate et al. (1999)]. There are two main points that make an algorithm an iterated local search: (i) there must be a single chain that is being followed (this then excludes population-based algorithms); (ii) the search for better solutions occurs in a reduced space defined by the output of a black box heuristic. In practice, local search has been the most frequently used embedded heuristic, but in fact any optimizer can be used, be-it deterministic or not.

The purpose of this review is to give a detailed description of iterated local search and to show where it stands in terms of performance. So far, in spite of its conceptual simplicity, it has lead to a number of state-of-the art results without the use of too much problem-specific knowledge; perhaps this is because iterated local search is very malleable, many implementation choices being left to the developer. In what follows we will give a formal description of ILS and comment on its main components.

```
Procedure Iterated Local Search
   $s_0$ = Generate Initial Solution
   $s^*$ = Local Search($s_0$)
 repeat
   $s'$ = Perturbation($s^*$)
   $s^{*\prime}$ = Local Search($s'$)
   $s^*$ = Acceptance Criterion ($s^*$, $s^{*\prime}$)
 until termination condition met
end
```

ILS involves four main components:

1. Creating an initial solution;
2. A black-box heuristic that acts as a local search on the set S;
3. The perturbation operator, which modifies a local solution;
4. The acceptance criterion, which determines whether or not a perturbed solution will become the starting point of the next iteration.

Local search applied to the initial solution s_0 gives the starting point s^* of the walk in the set S^*. Starting with a good s^* can be important if high-quality solutions are to be reached as fast as possible. The initial solution s_0 used in the ILS is typically found one of two ways: a random starting solution is generated or a greedy construction heuristic is applied. A "random restart" approach with independent samplings is sometimes a useful strategy (in particular when all other options fail), it breaks down as the instance size grows because in that time the tail of the distribution of costs collapses. A greedy initial solution s_0 has two main advantages over random starting solutions: (i) when combined with local search, greedy initial solutions often result in better quality solutions s^*; (ii) a local search from greedy solutions takes, on average, less improvement steps and therefore the local search requires less CPU time.

The current s^*, we first apply a change or perturbation that leads to an intermediate state s' (which belongs to S where S is set of all local optimum). Then Local Search is applied to s' and we reach a solution $s^{*\prime}$ in S^*. If $s^{*\prime}$ passes an acceptance test, it becomes the next element of the walk in S^*; otherwise, one returns to s^*. The resulting walk is a case of a stochastic search in S^*, but where neighborhoods are never explicitly introduced. This iterated local search procedure should lead to good biased sampling as long as the perturbations are neither too small nor too large. If they are too small, one will often fall back to s^* and few new solutions of S^* will be explored. If on the contrary the perturbations are too large, s' will be random, there will be no bias in the sampling, and we will recover a random restart type algorithm will be recovered.

In practice, much of the potential complexity of ILS is hidden in the history dependence. If there happens to be no such dependence, the walk has no memory: the perturbation and acceptance criterion do not depend on any of the solutions visited previously during the walk, and one accepts or not $s^{*\prime}$ with a fixed rule. This leads to random walk dynamics on S^* that are "Markovian", the probability of making a particular step from s_1^* to s_2^* depending only on s_1^* and s_2^*. Most of the work using ILS has been of this type, though the studies show unambiguously that incorporating memory enhances performance [Stutzle (1998)].

The main drawback of any local search algorithm is that, by definition, it gets trapped in local optima that might be significantly worse than the global optimum. The strategy employed by ILS to escape from local optima is represented by perturbations to the current local optima. The perturbation scheme takes a locally optimal solution, s^*, and produces another solution from which a local search is started at the next iteration. Hopefully, the perturbation will return a solution outside the basins of attraction of previously visited local minima. That is, it will be "near" a previously unvisited local optimum. Choice of the correct perturbation scheme is of primary importance, because it has a great influence on the intensification/diversification characteristics of the overall algorithm. Generally, the local search should not be able to undo the perturbation; otherwise one will fall back into the local optimum just visited. Perturbation schemes are commonly referred to as "strong" and "weak", depending on how much they affect the solution that they change. A perturbation scheme that is too strong has too much diversity and will reduce the ILS to an iterated random restart heuristic. A perturbation scheme that is too weak has too little diversity and will result in the ILS not searching enough of the search space. The perturbation scheme should be chosen in such a way that it is as weak as possible while still maintaining the following condition: the likelihood of revisiting the perturbed solution on the next execution of Local Search should be low [Lourenco et al. (2002)]. The strength should remain as low as possible to speed up execution time. The desired perturbation scheme will return a solution near a locally optimal value. If this is the case, the local search algorithm should take less time to reach the next locally optimal value. Components from other meta-heuristics can sometimes be incorporated into the perturbation phase. Battiti and Protasi [Battiti and Protasi (1997)] proposed memory structures to control the perturbation. In doing so, one can force intensification when globally good values are reached and force diversification when the search stagnates in an area of the search space. Borrowing from Simulated Annealing [Kirkpatrick et al. (1983)], temperature controlled techniques have been used to force the perturbation to change in a deterministic manner. Basic variable neighborhood search employs a deterministic perturbation scheme. Just as perturbation can range from too much intensification (no perturbations) to too much diversification (perturb all elements of the solution), acceptance criterion choices affect the search in a similar way. The most dramatic acceptance criterion on the side of diversification is to accept all perturbed solutions. This type of practice can undermine the foundations of ILS, since it encourages a "random-walk" type search. Contrasting with this, the algorithm accepts only solutions that are improvements to the globally optimal value (a sort of greedy strategy). Many implementations of ILS employ this type of acceptance strategy [Rossi-Doria et al. (2002)]. This type of criterion, especially with a weak perturbation scheme, can restrict the search from escaping the current basin of attraction. Moreover, with this type of scheme the probability of reaching the same locally optimal value increases a trait that reduces the algorithm's overall effectiveness. When the search stagnated, the random restart is a good way to ensure some diversification and to counterbalance the (possible) negative effects of too greedy a search. Large perturbations are only useful if they can be accepted. This only occurs if the acceptance criterion is not too biased toward better solutions [Lourenco et al. (2001)]. Stutzle (1998) showed that acceptance criteria that accept some worse solutions outperform their best-only counterparts.

For what concerns the stopping rule, generally the algorithm executes until one of the following conditions is met:

- a fixed number of cycles have finished;
- the best solution has not changed for a predefined number of cycles;
- a solution has been found that is beyond some predefined threshold.

ILS has many of the desirable features of a metaheuristic: it is simple, easy to implement, robust, and highly effective. The essential idea of ILS lies in focusing the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for a given optimization engine. The success of ILS lies in the biased sampling of this set of local optima. How effective this approach turns out to be depends mainly on the choice of the local search, the perturbations, and the acceptance criterion. Interestingly, even when using the most naive implementations of these parts, ILS can do much better than random restart. But with further work so that the different modules are well adapted to the problem at hand, ILS can often become a competitive or even state of the art algorithm. This dichotomy is important because the optimization of the algorithm can be done progressively, and so ILS can be kept at any desired level of simplicity. This, plus the modular nature of iterated local search, leads to short development times and gives ILS an edge over more complex metaheuristics in the world of industrial applications. As an example of this, recall that ILS essentially treats the embedded heuristic as a black box; then upgrading an ILS to take advantage of a new and better local search algorithm is nearly immediate. Because of all these features, we believe that ILS is a promising and powerful algorithm to solve real world complex problems in industry and services, in areas ranging from finance to production management and logistics. Finally, notice that although all of the present review is given in the context of tackling combinatorial optimization problems, in reality much of what is covered can be extended in a straight-forward manner to continuous optimization problems.

3.3 Maximin Latin Hypercube Designs:

We will denote as follows the s-norm distance between two points x_i and $x_j, \forall\ i, j = 1, 2, \cdots, N$:

$$d_{ij} = \| x_i - x_j \|_s \tag{3.1}$$

Unless otherwise mentioned, we will only consider the Euclidean distance measure ($s = 2$). In fact, we will usually consider the squared value of d_{ij} (in brief d), i.e. d^2 (saving the computation of the square root). This has a noticeable effect on the execution speed since the distances d will be evaluated many times.

3.4 Definition of LHD:

A Latin Hypercube Design (LHD) is a statistical design of experiments, which was first defined in 1979 [McKay et al. (1979)]. An LHD of k-factors (dimensions) with N design points, $x_i = (x_{i1}, x_{i2} \cdots x_{ik}) : i = 0, 1, \ldots, N-1$, is given by a $N \times k$- matrix (i.e. a matrix with N rows and k columns) X, where each column of X consists of a permutation of integers $0, 1, \cdots, N-1$ (note that each factor range is normalized to the interval $[0, N-1]$) so that for each dimension j all x_{ij} , $i = 0, 1, \cdots, N-1$ are distinct. We will refer to each row of X as a (discrete) design point and each column of X as a factor (parameter) of the design points.

We can represent $\boldsymbol{X}$ as follows

$$\boldsymbol{X}=\begin{pmatrix} x_0 \\ \vdots \\ x_{N-1} \end{pmatrix}=\begin{pmatrix} x_{01} & \cdots & x_{0k} \\ \vdots & \cdots & \vdots \\ x_{(N-1)1} & \cdots & x_{(N-1)k} \end{pmatrix} \tag{3.2}$$

such that for each $j \in \{1, 2 \cdots, k\}$ and for all $p, q \in \{0, 1, \cdots, N-1\}$ with $p \neq q$; $x_{pj} \neq x_{qj}$ holds.

Given a LHD X and a distance d, let

$$D = \{d(x_i, x_j) : 1 \leq i < j \leq N\}.$$

Note that $|D| \leq \binom{n}{2}$. We define $D_r(X)$ as the r-th minimum distance in D, and $J_r(X)$ as the number of pairs $\{x_i, x_j\}$ having $d(x_i, x_j) = D_r(X)$ in X.

The maximin LHD problem aims at finding a LHD X^* such that $D_1(X)$ is as large as possible. However, a search which only takes into account the D_1 values is certainly not efficient. Indeed, the landscape defined by the D_1 values is "too flat". For this reason the search should be driven by other optimality criteria, which take into account also other values besides D_1.

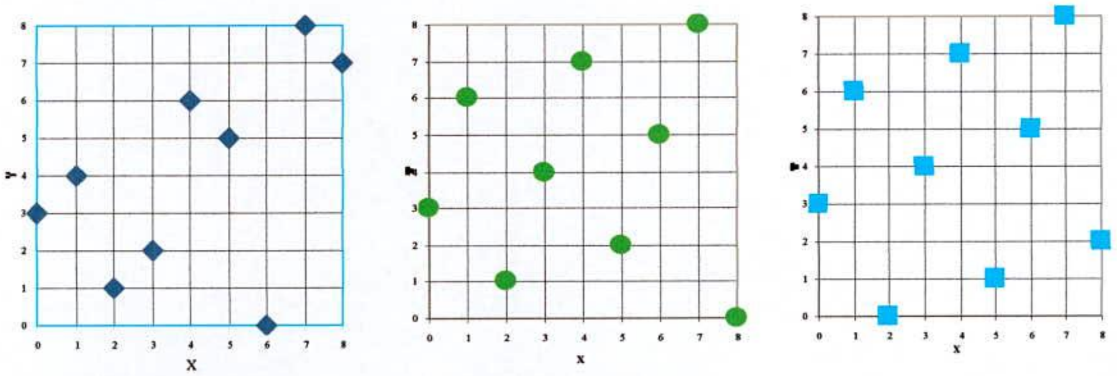

Fig: (a) $D_1(X_r)=2$, $J_1(X_r)=4$ Fig: (b) $D_1(X_{sm})=8$, $J_1(X_{sm})=4$ Fig: (c) $D_1(X_M)=8$, $J_1(X_M)=2$

Figure 3.1: Some LHDs and their corresponding (D_1, J_1) values

3.5 Optimality Criteria

In order to drive the search through LHDs we need some criterion to compare them. Below we will describe some of the criteria employed in the literature.

Opt(D_1, J_1) Optimality Criterion : Under this criterion a LHD Y can be considered better than another one X if a lexicographic ordering holds:

$$\begin{aligned} &D_1(Y) > D_1(X) && \text{or} \\ &D_1(Y) = D_1(X) && \text{and} \quad J_1(Y) < J_1(X). \end{aligned} \tag{3.3}$$

We illustrate this optimality criterion as follows. In Figure 3.1(a) X_r is a randomly generated LHD with $(N, k_r) = (9,2)$ where $D_1(X_r) = 2$ and $J(X_r) = 4$; Figure 3.1 (b) presents an improved configuration X_{sm} where $D_1(X_{sm}) = 8$ with $J(X_{sm}) = 4$. A third LHD X_M is given in Figure 3.1 (c) where $D_1(X_M) = 8$ and $J_1(X_M) = 2$; by the Opt(D_1,J_1) criterion this is the best configuration among the three.

By generalizing this approach, we can consider the problem like a multi-objective problem with priorities: maximize the objective with highest priority D_1; within the set of optimal solutions with respect to D_1, minimize the objective with second highest priority J_1. Note that Johnson et al. [Johnson et al. (1990)] first proposed this optimality criterion.

Opt(φ) Optimality Criterion : As previously remarked, if there exist different LHDs with equal D_1 and J_1 values, i.e. in case there exist at least two LHDs X, Y such that $D_1(X) = D_1(Y) = D_1$ and $J_1(X) = J_1(Y) = J_1$, we could further consider the objective D_2 and maximize $D_2(X)$, the second smallest distance in X, and, if equality still holds, minimize $J_2(X)$, the number of occurrence of $D_2(X)$, and so on. Then an optimal design X sequentially maximizes D_{is} and minimizes J_{is} in the following order: D_1, J_1; D_2, J_2, $\cdots$,D_m, J_m. Morris and Mitchell [Morris and Mitchell (1995)] have used all the above measures to define a family of scalar-valued functions (to be minimized), which can be used to rank competing designs in such a way that a maximin design receives the highest ranking. This family of functions, indexed by p, is given by

$$\phi_p(x) = \sum_{r=1}^{m} \left[\frac{J_r(X)}{(D_r(X))^p} \right]^{1/p} \tag{3.4}$$

where p is a positive integer parameter. Under this criterion, LHD Y is better than X if

$$\phi_P(Y) < \phi_R(X).$$

Note that for large enough p, each term in the sum in (3.4) dominates all subsequent terms. Through p we can control the impact of the different D_r distances: as p increases, the impact of distance D_1 becomes more and more relevant. In the form (3.4), the evaluation of ϕ_p would be computationally costly. However, it has a computationally cheaper form (see [Jin et al. (2005)]). Indeed, (3.4) can be simplified as

$$\phi_P(X) = \left[\sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{1}{d_{ij}^{P}} \right]^{\frac{1}{P}} \tag{3.5}$$

which can be computed without the need of detecting and ordering all the D_i values.

An apparent drawback of the Opt(ϕ) criterion, if we are interested in maximin values (maximum D_1 value), is that LHDs with smaller (better) ϕ_p can have a worse(smaller) D_1, i.e. we can have X and Y such that $\phi_P(X) < \phi_P(Y)$ and $D_1(X) < D_1(Y)$. This phenomenon has been frequently observed in our computational experiments. Nevertheless, a profitable choice is to work in order to minimize the ϕ_p function but, at the same time, keep track of the best (D_1, J_1) values observed during such minimization. This way the search in the solution space is guided by a kind of heuristic function. Such mixed approach might appear strange but, as we will demonstrate experimentally, it can be extremely effective.

While the two criteria above are strictly related to maximin values and they will be widely employed in the definition of approaches for detecting maximin solutions, for the sake of completeness, we also mention that also other optimality criteria, not necessarily related with maximin values, are available in the literature. We present a couple of them as well as the approaches for constructing the optimal Latin hypercube design in Table 3.1.

Table 3.1: Some well know approaches as well as optimal criterion for optimal experimental designs

Researchers	Year	Algorithm	Objective functions
Audze and Eglajs	1977	Coordinates Exchange Algorithm	Potential Energy
Park	1994	A 2-stage(exchange-and Newton-type) algorithm	Integrated mean squared error and entropy criteria
Morris and Mitchell	1995	Simulated annealing	ϕ_p criterion
Ye et al.	2000	Columnwise-pairwise	ϕ_p and entropy criteria
Fang et al.	2002	Threshold accepting algorithm	Centered L_2-discrepancy
Bates et al.	2004	Genetic algorithm	Potential energy
Jin et al.	2005	Enhanced stochastic evolutionary algorithms	ϕ_p criteria, entropy and L_2 discrepancy
Liefvendahl and Stocki	2006	Columnwise-pairwise and genetic algorithms	Minimum distance and Audze-Eglajs function
Van Dam et al.	2007	Branch-and-bound algorithm	1-norm and infinite norm distances
Grosso et al.	2008	Iterated local search and simulated annealing algorithms	ϕ_p criterion

3.6 ILS Heuristic for Maximin LHD

In Section 4.1 we have discussed a general scheme for ILS-based algorithms. Now we present the ILS based procedure for maximin Latin hypercube design. As we have stated earlier, the main components of ILS heuristic approaches are Initialization (I_S), LocalSearch (L_M), Perturbation Move (P_M), and the Stopping Rule (S_R)

The pseudo-code of the proposed ILS heuristic for maximin LHD problems is given bellow:

```
Step 1. **Initialization :** $X = I_S(\{0, 1, . . . , N - 1\}))$
Step 2. **Local Search :** $X^* = L_M(X)$
 while $S_R$ not satisfied do
   Step 3. **Perturbation Move :** $X' = P_M(X)$
   Step 4. **Local Search :** $X^* = L_M(X')$
   Step 5. **Improvement test :** if $X^*$ is better than $X$,
   set $X = X^*$
end while
Return $X$
```

Below we detail the components in order to fully specify our algorithm.

3.6.1 Initialization (I_S)

The initialization (I_S) procedure embedded in our algorithm is extremely simple: the first initial solution is randomly generated. In particular, the first initial solution generation is built as follows. For each component $h \in \{1, . . . , k\}$ a random permutation $v_0, . . . , v_{N-1}$ of the integers $0, 1, . . . ,N - 1$ is generated and we set

$$x_{rh} = v_r \quad \text{for all } r \in \{0, . . . , N - 1\}.$$

Although more aggressive procedures could be designed, we chose random generation because it is fast and unbiased.

3.6.2 Local Search Procedure (L_S)

In order to define a local search procedure (L_S), we need to define a concept of neighborhood of a solution. Given a LHD $X = (x_1, . . . , x_N)$, its neighborhood is made of all other LHDs obtained by applying local moves to X. Before introducing some local moves, we first introduce the notion of critical point.

Critical point: We say that x_i is a critical point for X, if

$$\min_{j \neq i} d(x_i, x_j) = D_1(X),$$

i.e., the minimum distance from $\mathbf{x}_i$ to all other points is also the minimum one among all the distances in $\boldsymbol{X}$. We denote by $I(X)$ $\{1, . . . ,N\}$ the set of indices of the critical points in X.

3.6.3 Local Moves (L_M): A local move is an operator that applies some form of slight perturbation to a solution X, in order to obtain a different solution. Different local moves define different neighborhoods for local search. In the literature two different local moves are available: Rowwise-Pairwise (RP) exchange [Park (1994)] and Columnwise-Pairwise (CP) exchange [Morris and Mitchell (1995)]. In Park's algorithm [Park (1994)] some active pairs (pairs of critical points, in our terminology) are selected. Then, for each chosen pair of two active rows, say i_1 and i_2, the RP exchange algorithm considers all the possible exchanges of corresponding elements as follows:

$$x_{i1,p} \leftrightarrow x_{i2,q} \ \forall \ p, q = 1, 2, \ldots, k : p \neq q,$$

and finds the best exchange among them. The CP algorithm proposed by Morris and Mithchell [Morris and Mitchell (1995)] exchanges two randomly selected elements within a randomly chosen column. But in [Li and Wu (1997)], Li and Wu defined the CP algorithm in a bit different way: they randomly choose a column and replace it by its random permutations if a better LHD is obtained.

It is observed that the effect of CP based local search and RP based local search is not significance [Jamali (2009)]. So, here, RP based local move is considered as defined in [Jamali (2009)] which is a bit different than that of [Park (1994)]. For optimal criteria we consider Opt(ϕ) optimal criteria.

The definition of Rowwise-Pairwise Critical Local Moves (we call it LM_{RpD1}) as follows. The algorithm sequentially chooses two points (rows) such that at least one of them is a critical point, then exchanges two corresponding elements (factors) of the selected pair. If $i \in I(X)$, $r, j \in \{1, \ldots, N\}$, $h, \ell \in \{1, \ldots, k\}$, swapping the ℓ-th component gives the neighbor Y defined by

$$y_{rh} = \begin{cases} x_{rh} & \text{if } r \neq i \text{ or } h \neq \ell \\ x_{ih} & \text{if } r = j \text{ and } h = \ell \\ x_{jh} & \text{if } r = i \text{ and } h = \ell \end{cases} \tag{3.8}$$

It is remarked that, if Opt(D_1, J_1) be the optimality criterion, it perfectly makes sense to avoid considering pairs x_i and x_j such that $I(X) \cap \{x_i, x_j\} = \varnothing$ since any swap involving two non-critical points cannot improve the D_1 value of the current LHD.

When Opt(ϕ) is adopted as optimality criterion, any exchange can, in general, lead to an improved value of ϕ. The RP local move for Opt(ϕ) optimality criterion is denoted by $LM_{RP}\phi$ and is also defined as Eq. (4.8), the only difference being that we drop the requirement that at least one point must be critical.

We now illustrate the RP based local moves by considering a randomly generated initial design A : $(N,k) = (7,2)$ (see Figure 4.2(a)). Then a neighborhood solution of A, by considering points (0,2), (4,4) (here both are critical points), is LHD B, obtained after swapping the second coordinate of the points (0, 2) and (4,4) (See Figure 4.2 (b)).

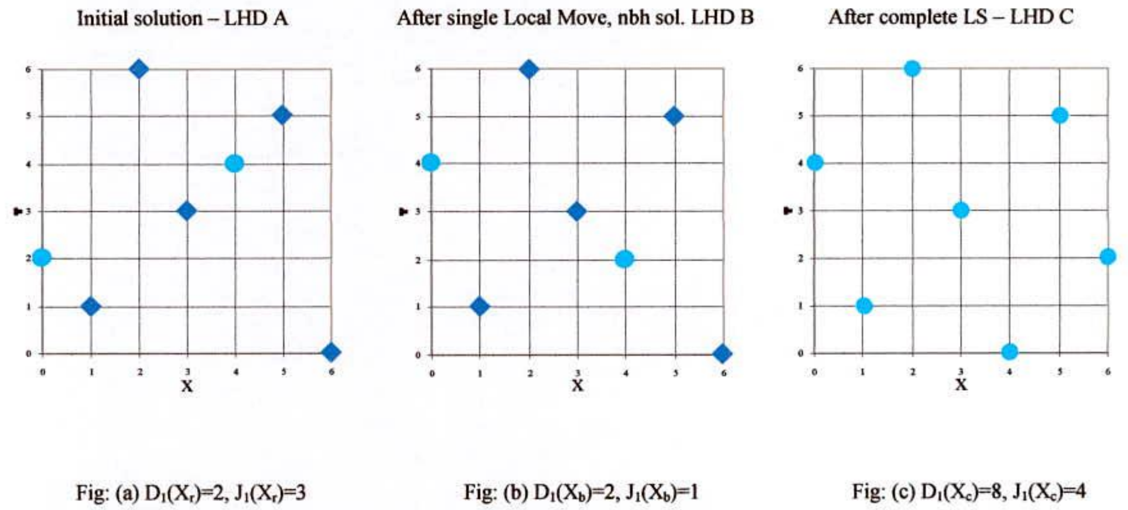

Figure 3.2: Illustration of Neighborhood solutions for $LM_{RP_{D1}}$ based local search (LS) procedure

Also note that LHD B is an improving neighbor of LHD A, since $(D_1, J_1)(B) = (2,1)$ whereas $(D_1, J_1)(A) = (2,3)$. Finally Figure 4.2 (c) shows the maximin LHD produced by the Local search procedure.

3.6.4 Acceptance Rule: Among the two type of local moves [Jamali (2009)], we considered Best Improve (BI) acceptance rule as there are no significant difference regarding output (see [Jamali (2009)]). For the BI acceptance rule, the whole neighborhood of the current solution is searched for the best improving neighbor. We warn again the reader that the meaning of "Y is better than X" can be defined accordingly with the Opt(D_1, J_l) or Opt(ϕ) optimality criterion. So for the Opt(D_1, J_1) optimality criterion: "Y is better than X" if

$$D_1(Y) > D_1(X) \text{ or } (D_1(X) = D_1(Y) \text{ and } J_1(X) > J_1(Y)).$$

On the other hand for Opt(ϕ) optimality criterion : "Y is better than X" if

$$\phi_p(Y) < \phi_p(X),$$

where ϕ_p is defined by (5).

3.6.5 Perturbation Move (P_M)

Perturbation is the key operator in ILS, allowing the algorithm to explore the search space by jumping from one local optimum to another. Basically, a perturbation is similar to a local move, but it must be somehow less local, or, more precisely, it is a move within a neighborhood larger than the one employed in the local search. Actually the perturbation operator produces the initial solutions for all the local searches after the first one. Among the two types of perturbation operators, say, (i) Cyclic Order Exchange (COE) and (ii) Pairwise Crossover (PC) proposed in [Jamali (2009)], we consider COE.

1. Cyclic Order Exchange (COE): Our first perturbation move procedure is Cyclic Order Exchange (COE). The operator COE produce a cyclic order exchange upon a randomly selected single component (column) of a randomly selected portion of the design points (rows). Among the three variant of COE perturbation move techniques: Single Cyclic Order Exchange (SCOE) perturbation operation, Multiple Components Cyclic Order Exchange (MCCOE), and Multiple Single Cyclic Order Exchange (MSCOE) [Jamali (2009)], we consider here only SCOE technique.

Single Cyclic Order Exchange (SCOE): For SCOE, we randomly choose two different rows (points), say x_i and x_j , such that $i < j$ and $j - i \geq 2$, in the current LHD X^*. Then, we randomly choose a column (component), say ℓ. Finally, we swap in cyclic order the value of component ℓ from point x_i to point $\mathbf{x}_j$. The pseudo-code structure for SCOE is the following.

The pseudo-code structure for SCOE is the following.

```
Step 1: randomly select two different points $x_i$ and $x_j$
        such that $i < j$ and $j - i \geq 2$
Step 2: Randomly choose a component $\ell$
Step 3a: set temporarily $x'^{t}_{j}\ell = x_j\ell$
        for $t = j, j - 1, \ldots, i + 1$ do
Step 3b: Replace the component $x(t)\ell$ by $x(t-1)\ell$
        end for
Step 3c: and replace $x_i\ell$ by $x'^{t}_{j}\ell$
```

Note that we require $j - i \geq 2$ because otherwise the perturbation would be a special case of the local move employed in the local search procedure. We illustrate the SCOE perturbation by an example. Assume we have the current LHD X^* with $N = 6$ and $k = 8$

$$X^* = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0 & 5 & 3 & 1 & 5 & 2 & 4 & 4 \\ 1 & 0 & 4 & 2 & 4 & 3 & 3 & 5 \\ 2 & 1 & 5 & 3 & 3 & 4 & 2 & 0 \\ 3 & 2 & 0 & 4 & 2 & 5 & 1 & 1 \\ 4 & 3 & 1 & 5 & 1 & 0 & 0 & 2 \\ 5 & 4 & 2 & 0 & 0 & 1 & 5 & 3 \end{pmatrix} \quad (3.9)$$

Now we randomly choose two rows (points), say $\boldsymbol{x_2}$ and $\boldsymbol{x_5}$ and we randomly choose the column (component) $\ell = 4$. Then, after the SCOE perturbation we get the following LHD X' (bold faces denote the values modified with respect to X^*),

$$X' = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 0 & 5 & 3 & 1 & 5 & 2 & 4 & 4 \\ 1 & 0 & 4 & \mathbf{5} & 4 & 3 & 3 & 5 \\ 2 & 1 & 5 & \mathbf{2} & 3 & 4 & 2 & 0 \\ 3 & 2 & 0 & \mathbf{3} & 2 & 5 & 1 & 1 \\ 4 & 3 & 1 & \mathbf{4} & 1 & 0 & 0 & 2 \\ 5 & 4 & 2 & 0 & 0 & 1 & 5 & 3 \end{pmatrix} \quad (3.10)$$

Note that SCOE only slightly modifies the current LHD X^* but this exactly follows the spirit of ILS, where the perturbation should keep unchanged large portions of the current solution and should not completely disrupt it structure.

CHAPTER IV

COMPLEXITY ANALYSIS OF THE ALGORITHM IN EXPERIMENTAL DOMAIN: CONSIDER ILS (D_1, J_1)

4.1 Introduction

As we discussed in chapter II that there are mainly two kind of complexity – *time complexity* and *space complexity*. It is noted that among the above two complexities, time complexity is most important for analysis any algorithm. Moreover, in any CPU,

Table 4.1: Pseudo code of the ILS (D_1, J_1) algorithm.

```
While do
    Set NonImpIteration = 0
    Whiledo
      for  $i = 1,\ldots\ldots, N$ do
         for $j = i + 1,\ldots.., N$ do
             Step 1= If $\{i,j\} \cap S_{Cpt}(X) \neq \phi$
             then: let $D_1 = D_1^{(S)}$, $k' = 0$
             for $l = 1,\ldots., k$ do
                  Step 2: Swap ($X_{il}$, $X_{jl}$)
                  Step 3: Compute $d_{i',u}^{(X+1)}$ until $d_{i',u}^{(X+1)} \geq D_1'$
                  $\forall i' = i\ .j: u = 0\ldots\ldots\ldots N-1; u \neq i'$
                  else break
                  Step 4 :Set $k' = k$ and $D_1' = \min\ d_{IJ}^{(X+1)}$
              end for
             Step 5:  Upload best LHD  if any
             else continue
         end for
      end for
    Step 6: Repeat the three loops if there has been at least an improvement
    Otherwise STOP
    Return $X'$
    Step 7: if $X'$ is better then set $X = X'$ and NonimpIteration = 0
   Otherwise increase NonimpIteration by one
   Step 8: if MaxNonImp > NonimpIteration
 $PM : X' = \in(X)$ and Repeat the loops
Otherwise STOP
Return X
```

are available spaces for running any algorithm. Also, now-a-days, the crisis of space for running an algorithm is almost solved by the presence of high memory based computer. Therefore, our main attention is to analyze the time complexity of the algorithms of ILS approach.

Before analysis the time complexity of the ILS (D_1, J_1) algorithm, It is worthwhile to present it in a Pseudo code. The Pseudo code of the ILS (D_1, J_1) algorithm is displayed in the Table (4.1).

As the aim of this section is to experimentally assess the computational cost of the proposed ILS(D_1) algorithm, we will first derive the number of operations required by a single local search, and then those for a single run of ILS (from now on in this section we will give as understood that we are discussing the ILS(D_1,J_1) version). For these experiments we consider k = 3, 5, 7, 10 and N = 10i: i = 1, 2, . . ., 10. We use the following parameter setting: Acceptance criteria= Best Improve (BI), Local Search=RP; Stopping criteria MaxNonImp=1000, Perturbation moves=SCOE and number of trials is one if otherwise not defined.

Assume that we are at iteration s of a local search and that the current value is $D_1^{(s)}, J_1^{(s)}$. The basic operation in a local search is the swap one between two points i and j. In order to compare the new candidate solution with the current one, we need to evaluate $D_1^{(s+1)}$ and $J_1^{(s+1)}$. Such operation does not require computing from scratch all the distances within the candidate solution. Indeed, only those involving points i and j are changed with respect to the current solution. Therefore, with a proper implementation we should only compute O(N) new distances, each of which requires a number O(1) of operations (indeed, we do not need to compute the distance from scratch but only update the part corresponding to the single coordinate whose value has been changed). In fact we do not always need to compute all the new distances: as soon as we compute a distance lower than $D_1^{(s)}$ we can stop, since the candidate solution is certainly worse than the current one. Therefore, each swap operation requires at most $O(N)$ operations.

The number of swap operation is not known in advance. Indeed, swap moves are restricted only to those involving at least a critical point. In Figure 4.1 the x-axis reports N and the y-axis reports the percentage of actually analyzed swap moves (those involving at least one critical point) over the total number of possible swaps in each run (those involving all possible pairs of points), for k = 3, 5, 7, 10. The black curve represents the percentage of analyzed swaps, the red curve represents the percentage of "avoided" swaps, i.e. those not involving critical points. We observe that for very small N (N <14) most of the possible swaps are to be considered, but as N grows the percentage of swaps to be considered drops dramatically, quickly falling below 10% for $N > 30$.

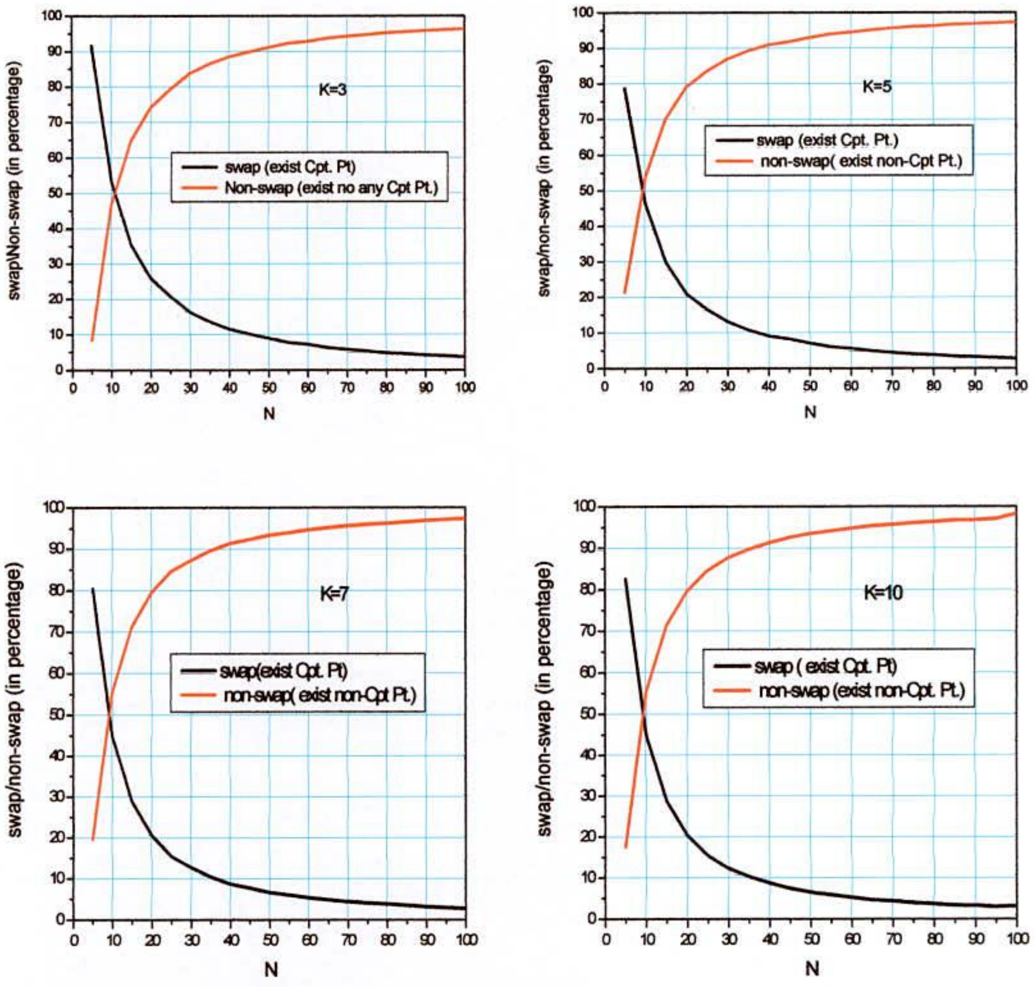

Figure 4.1: The percentage of pairs involving and not involving critical points

Figure 4.2 shows the history of the number of critical points during the local searches for the case (k, N) = (7, 50). We observe that most of the times the number of critical points is 1 or 2, and only occasionally is greater than 6. Figure 4.3 shows with a bar diagram the maximum number of critical points (MCP) obtained during the run of the algorithms for each (k, N). Apparently, we cannot observe any significant impact of N and k on the values of MCP. Indeed such values are always below 20, and most of the time they are near 10. In Figure 4.4 we report the average number of critical points in each neighborhood, rounded to the largest integer; this number is always stuck at 2, for all $k = 3, 5, 10$. So we can confidently claim that the size of the problem has practically no impact on the number of critical points in each visited configuration.

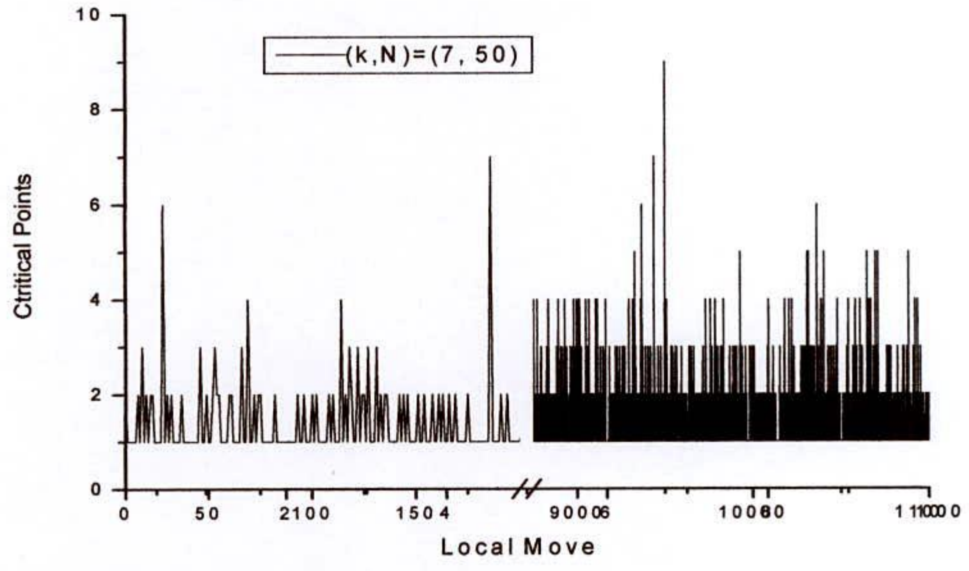

Figure 4.2: The history of number of critical points for $(k,N) = (7, 50)$ during local Move

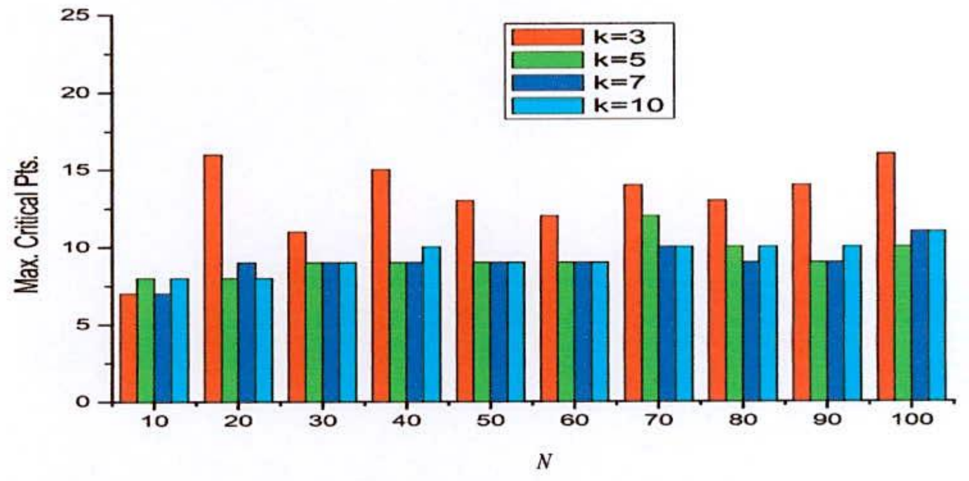

Figure 4.3: The impact of N on Maximum Critical Points during history of evaluation

Since we need to consider all the swap moves involving at least one critical point, the above considerations lead us to conclude that the total number of swap moves that we need to perform at a given iteration is simply $O(kN)$ (factor N is due to the number of pairs involving at least a critical point, factor *k* is due to the fact that, given a pair, we have a swap operation for each possible coordinate).

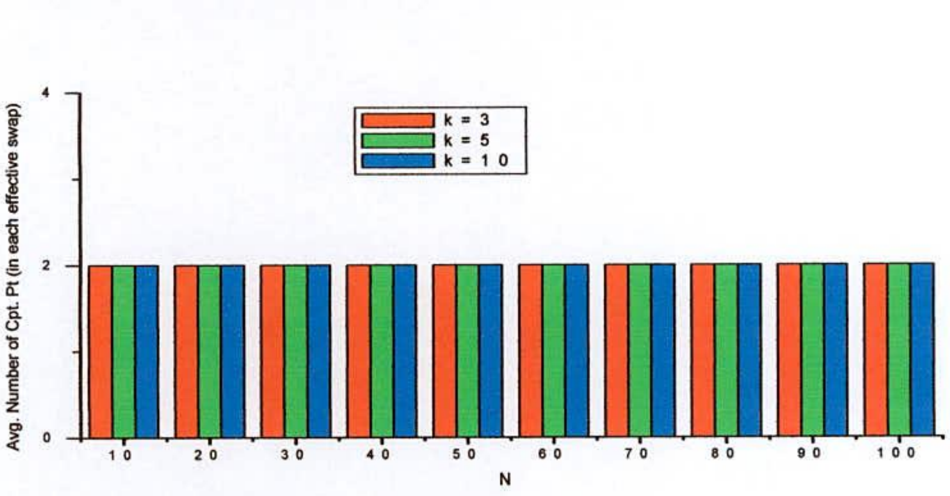

Figure 4.4: The impact of N on average Critical Points during history of evaluation

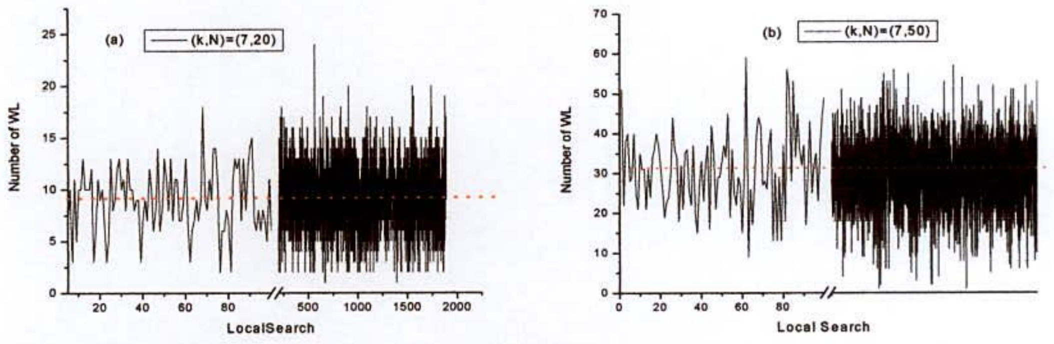

Figure 4.5: The history of WL for $(k, N) = (7, 20)$ and $(k, N) = (7, 50)$ during Local Search

The last thing, we need to consider, in order to evaluate the number of operations required by a local search, is the number of times the While-Loop is executed, i.e. the number of times an improving solution is observed during a local search. We will denote this number by WL. We will perform some experiments in order to find the impact of *N* as Figure 4.5 shows the history of WL during Local Search for (a) $(k, N) = (7, 20)$) and (b) $(k, N) = (7, 50)$. We observe in Figure 4.5 (a) that most of the time WL lies near 10 and the largest value observed in the figure is less than 25. In Figure (b) we notice that most of the time the number WL lies near 30 and the maximum value of WL (MWL) is near 80. That is WL (average as well as maximum value) increases together with N. Figure 4.6 shows a cleaner representation of the impact of *N* on the number MWL. Apparently there exists a linear relation between WL and *N*. We can also observe an impact of the dimension *k* on WL. In order to investigate the dependence on *k*, we performed another series of experiments, for $k = 5i : i = 1, 2, \ldots , 10$ and $N = 10, 25, 50, 100$. Note that for finding out the impact on the local search phase, WL is averaged over the corresponding number of performed perturbations. We observe in Figure 4.7 that there is a significant impact of k for all $N : N = 10, 25, 50, 100$ on the average number of WL (AWL).

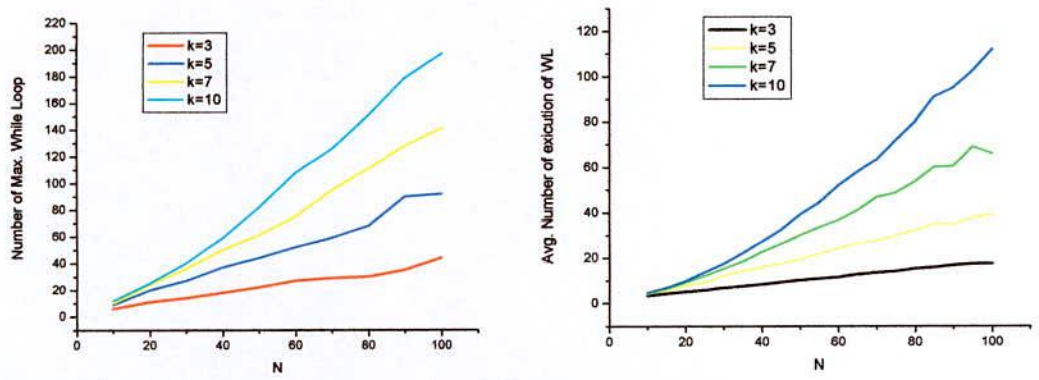

Figure 4.6: The impact of N on (a) Maximum WL (b) Average WL during history of Local Search

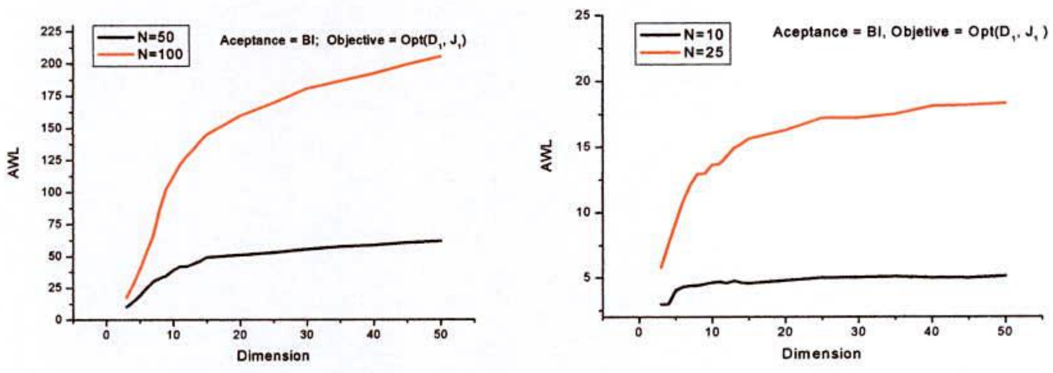

Figure 4.7: The Impact of k on AWL during Local Search

We observe that the trend shown by WL is roughly

$$T \approx k^c$$

where $0 \langle c \langle 1$, i.e. a fractional power functional dependence of WL on k. In conclusion, it has been experimentally seen that the number WL is $O(N\,k^c)$.

Now, if we sum up the time required by a single swap (at most $O(N)$), the total number of swaps per iteration $O(Nk)$, and the total number of iterations WL $O(Nk^c)$, we conclude that a local search requires at most $O(k^r N^q)$ for some r and q (in particular, we might conjecture that q is close to 3 and r ranges between 1 and 2). In order to validate this result we performed some experiments whose outcome is shown in Figure 4.8. In such figure we report the average computation time per local search as a function of N for the three different values $k = 3, 7, 10$ (the time is the average per local search over 10 runs of ILS).

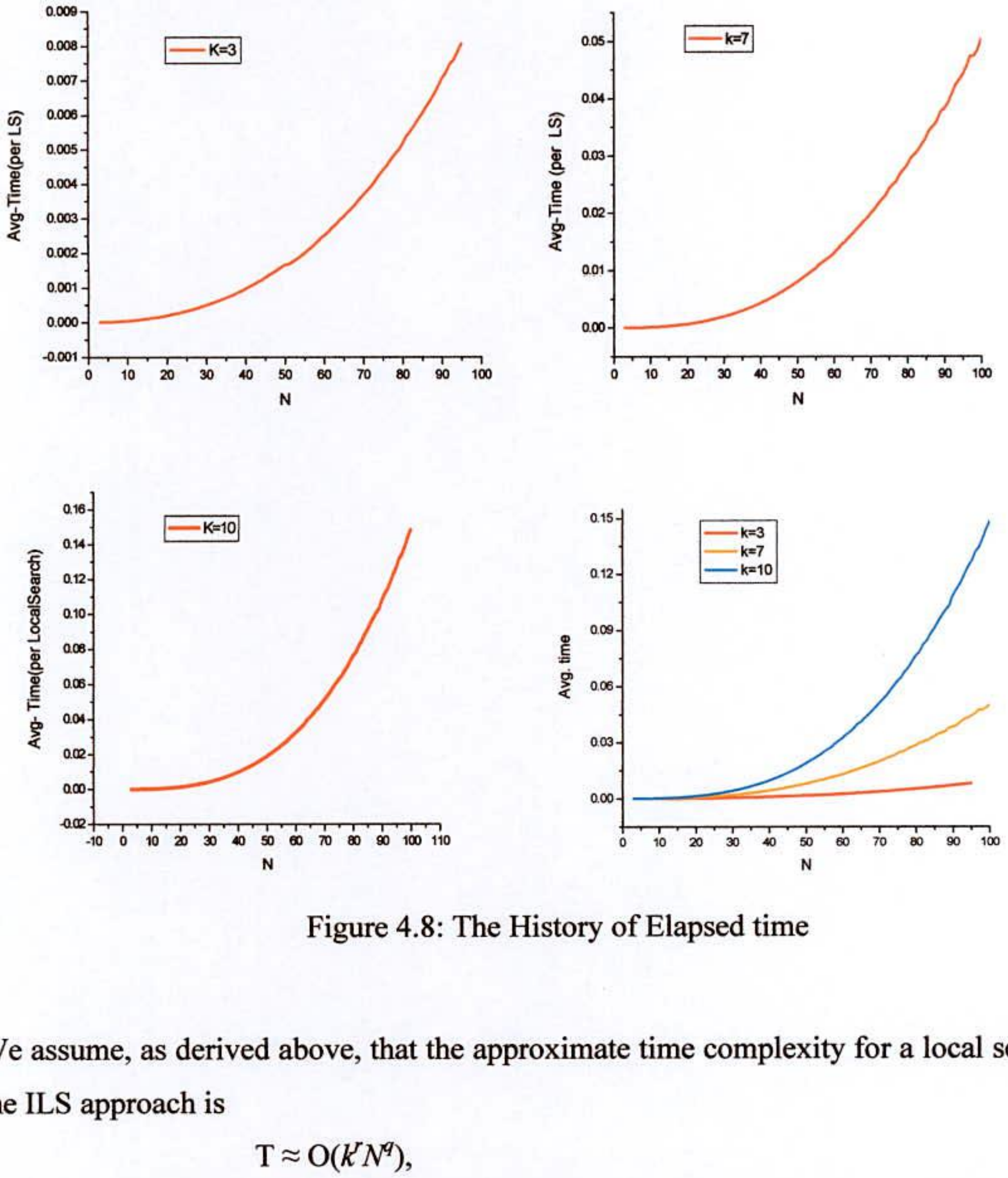

Figure 4.8: The History of Elapsed time

We assume, as derived above, that the approximate time complexity for a local search of the ILS approach is

$$T \approx O(k^r N^q),$$

and will try to determine practical values for r and q experimentally. In Figure 4.8, we observe that, for each k, the curves of elapsed times grow non-linearly with respect to the increasing of N. To find out the approximate value of q we fitted the data in a linear regression for each k as ,

$$\log(T) = q \log(N) + r \log(k),$$

where T = Average elapsed time in each LS. We observe from Figure 4.9 that the range of q is $2.5 \langle\ q \leq 3$. In particular, as k increases it seems that q tends to 3, which is the value previously conjectured.

To find out the approximate value of r we performed some experiments by considering LHDs $k = 5i : i = 1, 2, \ldots, 10$ with $N = 10, 25, 50, 100$. Figure 4.10 shows the impact of k on average elapsed time in each LS. In the figure we observe that the average elapsed time T increases slightly more than linearly with respect to k. In order to find out the approximate value of r we have fitted the data (see figure 4.10). We notice that the range of r is $1\ \langle\ r \leq 2$, which is again in accordance with what was previously conjectured.

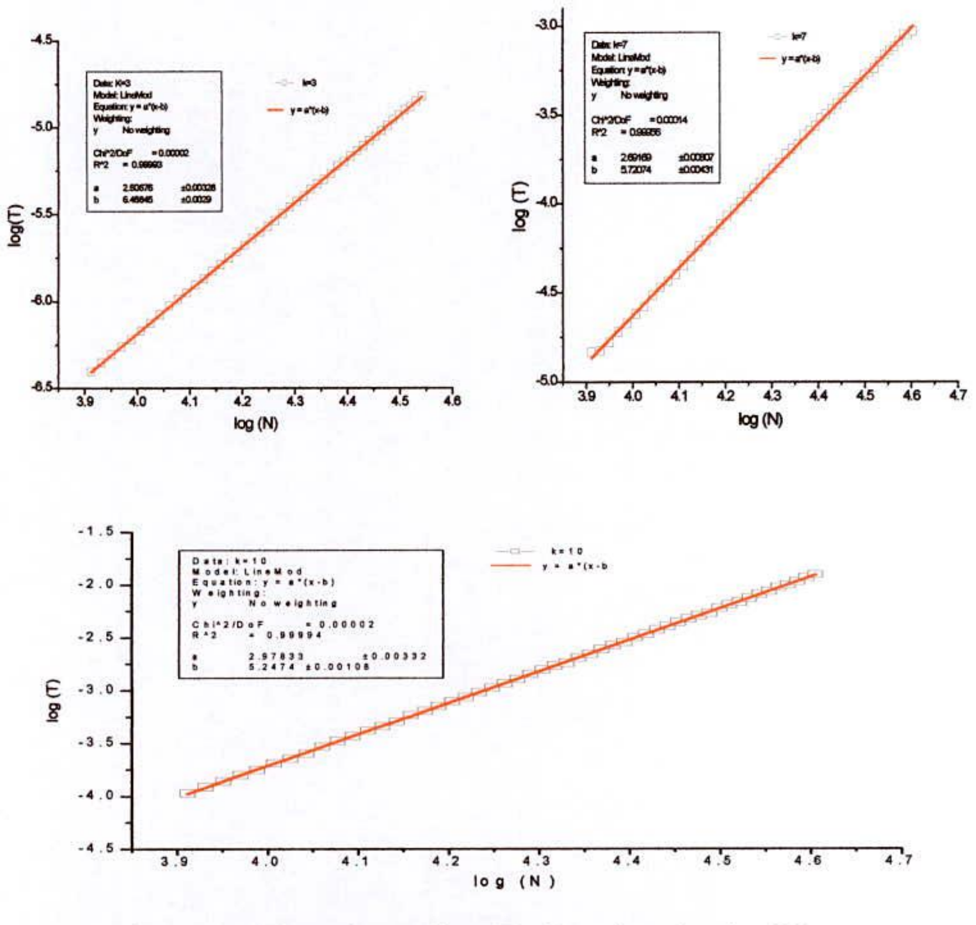

Figure 4.9: The values of log(T) plotted against log(N)

We remind the reader that this practical time complexity ≈ $O(k^r N^q)$, with $r \in (1, 2)$ and $q \in (2, 3)$, for LS has been estimated in the environment of ILS instead of evaluating a stand-alone local search. According to our observations, the local search usually performs less iterations in the ILS environment, due to the "partially optimal" structure preserved by the perturbations.

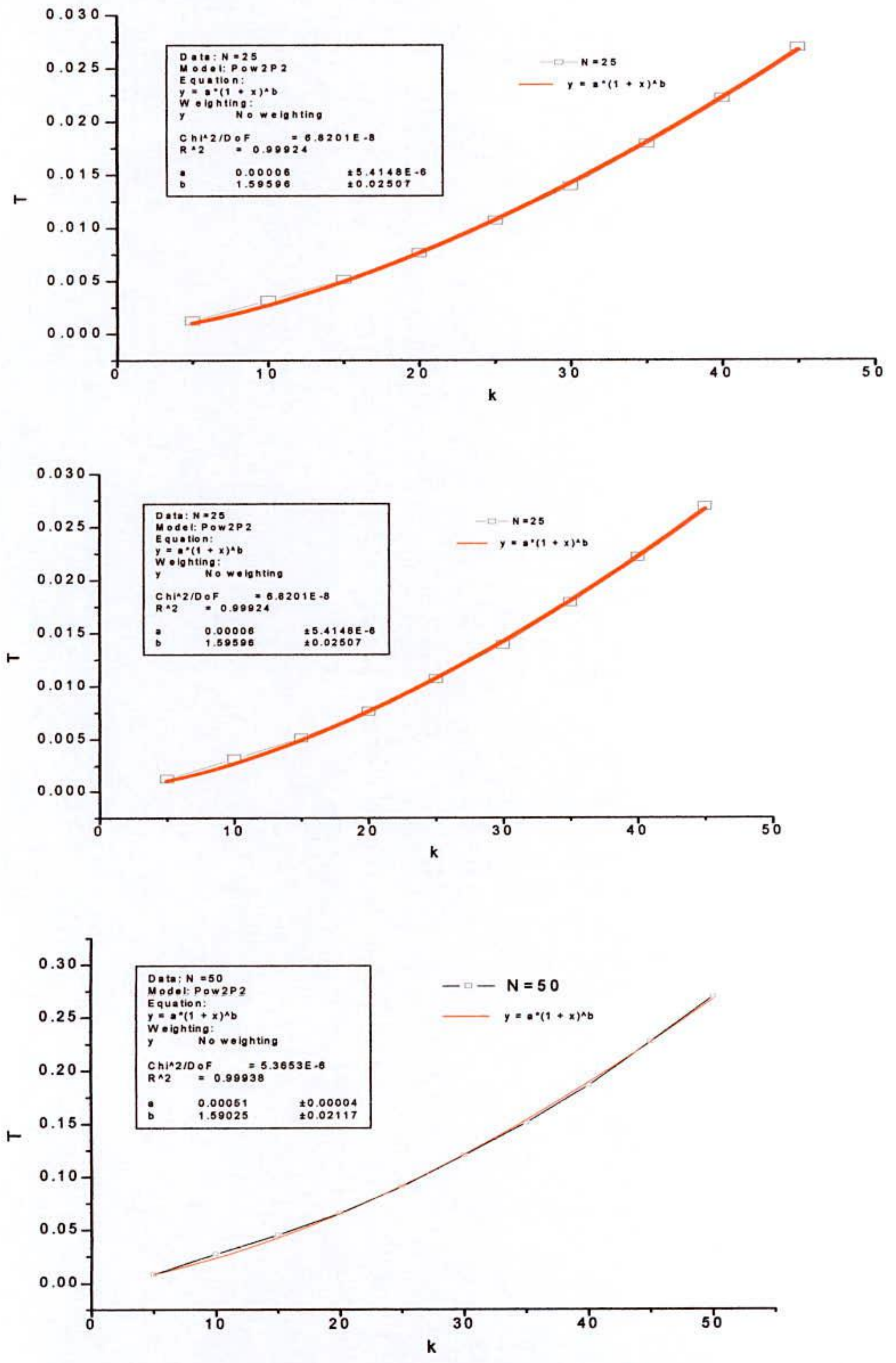

Figure 4.10: The approximate time complexity for LS with respect to k

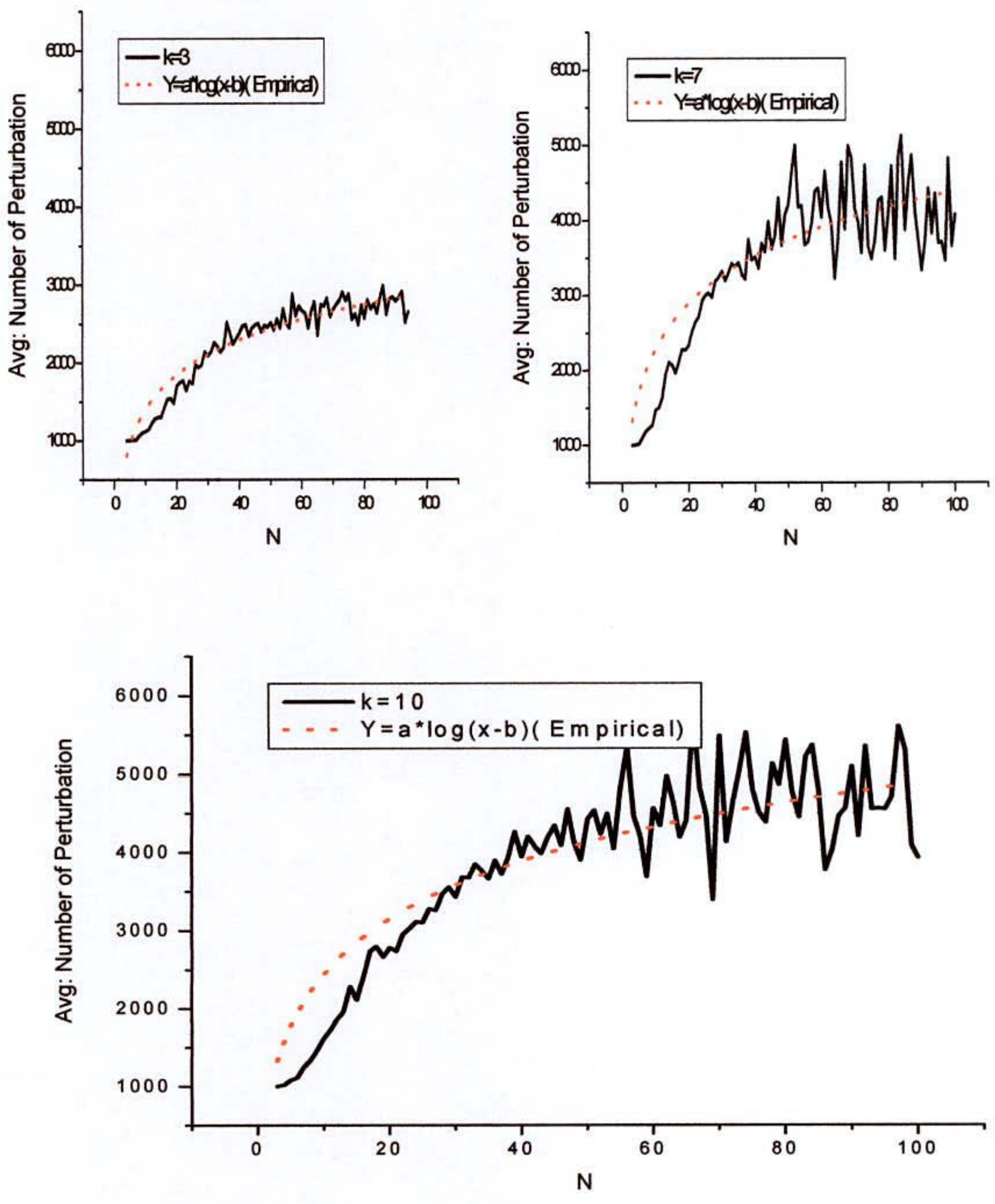

Figure 4.11: The impact of *N* on the number of perturbations

In order to get to an empirical evaluation of the number of operations required by an ILS run, we still need to evaluate the number of perturbations (and, thus, of local searches) performed during an ILS run. Then, we would like to find out the impact of *N* as well as *k* on the number of perturbations during each ILS run. For these experiments we performed ten runs of ILS and considered the average number of perturbations per run.

For these experiments we considered LHDs with $k = 3, 7, 10$ and $N = 3, 4, \ldots, 100$. From the experiments (see Figure 4.11) we notice that there is a significant impact of N on the number of perturbation invoked for all k considered. It seems that the number of invoked perturbations is somewhat logarithmic with respect to N (see the dot curve in Figure 4.11).

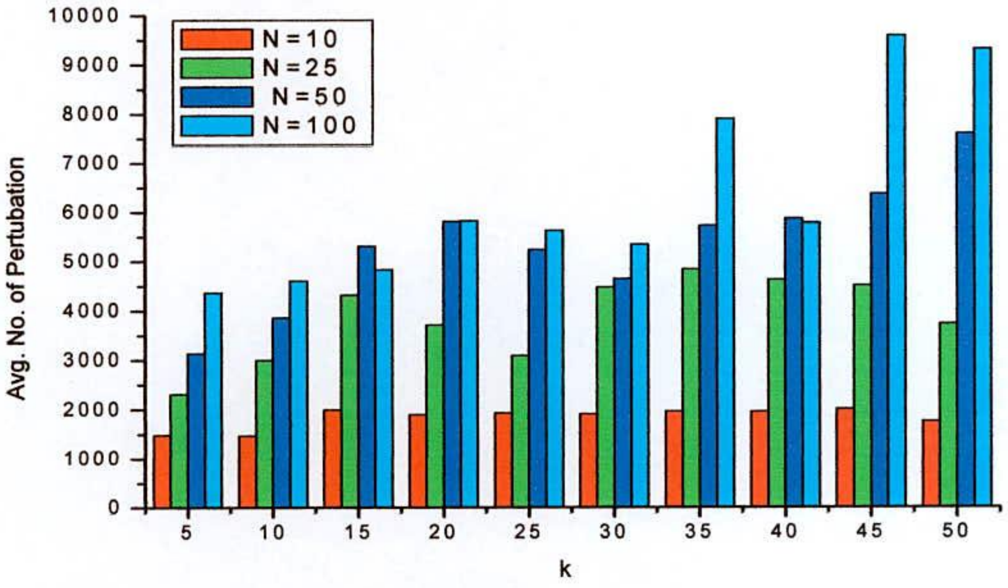

Figure 4.12: The impact of k on the number of perturbations

In order to find the impact of k on the number of perturbations, we considered LHDs: for each $N : N = 10, 25, 50, 100$; $k = 5i : i = 1, \ldots, 10$. From the experiments we remark that there is no significant impact of k on the perturbation invoked during the run (see the bar diagram in Figure 4.12). Now, if we put together the observation that the overall number of perturbations/ local searches per ILS run is $O(\log(N))$, and the previous one about the $O(k^r N^q)$ about the complexity of local searches, we can conclude that a bound on the overall time required by a single ILS run is $O(k^r N^q \log(N))$, a fact that it is also experimentally confirmed by the analysis of the elapsed times per ILS run.

Finally, the analysis of time complexity for ILS (D_1, J_1) is given in a tabular form along with the pseudo code of the algorithm. Table 4.2 represents the analysis of the time complexity for the ILS (D_1, J_1).

Table 4.2: Analysis of time complexity for ILS (D_1, J_1)

(*i*) Compute $d_{i',u}^{(s+1)} \xrightarrow{Operation} O(1)$

(comp. only swapped cord.)

(*ii*) Step 3 (in worst case) $\xrightarrow{Operation} O(N)$

(since compu. stop as soon as $d_{i',u}^{(s+1)} \le D_1'$)

(*iii*) Inner most for loop $\xrightarrow{Operation} O(k)$

(in BI local move)

(*v*) WL (in local search) $\xrightarrow{Operation} O(Nk^c)$

$(0 < c < 1)$, experimentally computed

(*iv*) outer two for loops $\xrightarrow{Operation} O(N)$

(in worse case theoretically $O(N^2)$)

(but experimentally swap $O(N)$)

(*v*) WL (in local search) $\xrightarrow{Operation} O(Nk^c)$

$(0 < c < 1)$, experimentally computed

Total cost of a single LS

$O(1).O(N).O(k).O(N).O(Nk^c) \approx O(k^r N^q)$

$:(1 < r < 2)$ & $(2 < q \le 3)$

Cost of a single ILS: Opt (D_1, J_1)

Perturbation (for fixed MIN))

O (log(N))

Total Cost:

$O(k^r N^q).O(\log(N)) \approx O(k^r N^q \log(N))$

$:(1 < r < 2)$ & $(2 < q \le 3)$

```
While do
  Set NonImpIteration = 0
    Whiledo
    for  $i = 1,\ldots\ldots, N$ do
      for $j = i + 1,\ldots.., N$ do
      Step 1= If $\{i,j\} \cap S_{Cpt}(X) \neq \phi$
        then: let $D_1 = D_1^{(S)}$, $K' = 0$
          for $l = 1,\ldots., k$ do
                Step 2: Swap $(X_i, X_{jl})$
                Step 3:
                Step 3: Compute $d_{i',u}^{(X+1)}$
                until $d_{i',u}^{(X+1)} \ge D_1'$
              $\forall i' = i\ .j: u = 0\ldots\ldots\ldots N-1;$
              $u \neq i'$
                else break
              Step 4 :Set $k' = k$ and
                $D_1' = \min\ d_{IJ}^{(X+1)}$
          end for
          Step 5:  Upload best LHD
                   if any
          else continue
      end for
    end for
    Step 6: Repeat the three loops if
        there has been at least an
        improvement
  Otherwise STOP
  Return X'
  Step 7: if X' is better then
 set X= X' and NonimpIteration = 0
   Otherwise increase
NonimpIteration by on
 Step 8: if MaxNonImp >
NonimpIteration
PM : X' = ∈ (X) and Repeat the
loops
Otherwise STOP
Return X
```

CHAPTER V

COMPLEXITY ANALYSIS OF THE ALGORITHM IN EXPERIMENTAL DOMAIN: CONSIDER ILS(Φ)

In this section we will perform some experiments to derive a formula connecting the computation times of ILS(ϕ) with N and k. The analysis will be similar to the one previously done for ILS(D_1). For these experiments we consider ILS(ϕ) with the following setting: Local Search: acceptance criterion=First Improve(FI), local move $= LM_{Rp\phi}$; stopping criterion: MaxNonImp =100; Perturbation Technique = SCOE.

Table 5.1: Pseudo code of the ILS(Φ) algorithm

```
While do
  Set NonImpIteration = 0
  Whiledo
    for  $i = 1,\ldots\ldots, N$ do
      for $j = i + 1,\ldots\ldots, N$ do
        Step 1= let $D_1 = D_1^{(S)}$, $k' = 0$
          for $l = 1,\ldots., k$ do
            Step 2: Swap $(X_{il}, X_{jl})$
              Step 3: Compute $d_{i',u}^{(X+1)}$ until $d_{i',u}^{(X+1)} \geq D_1'$
              $\forall i' = i\ .j$: u = 0.........N-1; $u \neq i'$
              else break
              Step 4 :Set $k' = k$ and $D_1' = \min d_{IJ}^{(X+1)}$
          end for
        Step 5:  Upload best LHD  if any
        else continue
      end for
    end for
   Step 6: Repeat the three loops if  there has been at least an improvement
   Otherwise STOP
   Return $X'$
   Step 7: if $X'$  is better then set $X = X'$ and NonimpIteration = 0
  Otherwise increase NonimpIteration by one
  Step 8: if MaxNonImp > NonimpIteration
 $PM: X' = \in(X)$ and Repeat the loops
Otherwise STOP
Return X
```

Again before analysis the time complexity of the ILS(ϕ) algorithm, It is worthwhile to present it in a Pseudo code. The Pseudo code of the ILS(ϕ) algorithm is displayed in the Table (5.1).

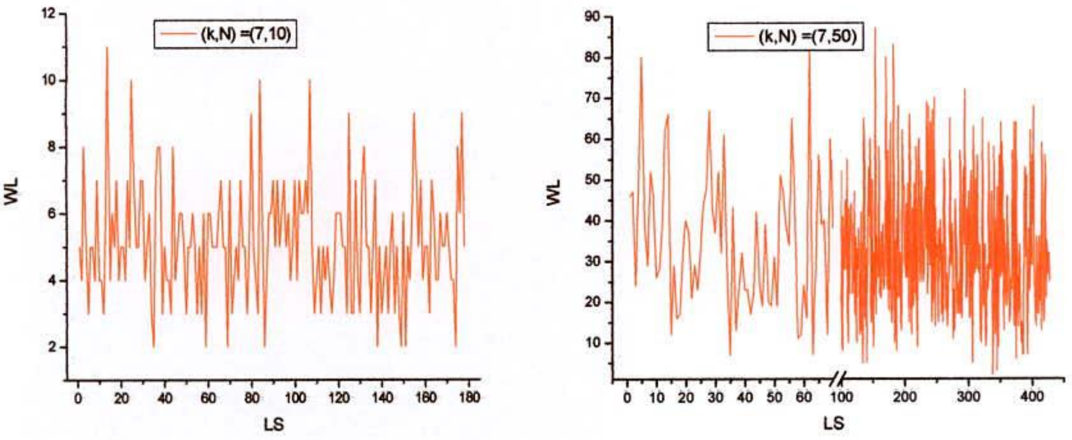

Figure 5.1: The history of WL values for $(k,N) = (7, 10)$ and $(k,N) = (7, 50)$ during Local Search

We will first discuss the time required by a local search. We do not discuss the time required by each swap move: this is the same as in ILS(D_1) and is at most O(N). However, the number of swap moves which have to be attempted at each iteration is now different. Indeed, since we are considering the $LM_{R\rho\varphi}$ local move, we have to consider all possible pairs of points (also those not involving critical points). Therefore, the number of swap operations is O(kN^2). Note that this is an upper bound: since we are employing the FI acceptance criterion, we perform swap operations only until an improvement is observed.

Next, we need to derive some formula for the number of times the While-Loop is executed during a local search, i.e. for the number of iterations performed by a local search. As before, we will denote this number with WL. Note that with respect to ILS(D_l) we made a change in the local search, adopting the FI acceptance criterion rather than the BI one. Figure 5.1 shows the history of WL values during different local searches for (a) $(k,N) = (7, 10)$) and (b) $(k,N) = (7, 50)$. We observe in Figure 5.1(a) that most of the time WL lies near

5 and never exceeds 12, whereas in (b) we notice that most of the time WL lies near 35 and the maximum value of WL is near 90. Therefore, it seems that WL increases together with N both for what concerns Average WL (AWL) values and Maximum WL (MWL) values. Figure 5.2 shows more clearly the relation between N and MWL as well as AWL. We observe that there is a linear impact of N. In order to establish the dependency of WL on k, we perform other experiments with $k = 2i : i = 1, 2\ldots\ldots,40$ and $N = 10, 25, 50, 75$. The relation between WL and k is not quite clear. Indeed, we have observed in Figure 4.7 that WL is increasing with k for $k < 10$ but after that it decreases and finally tends to get stable around a constant value. It seems that by enlarging k the local search is able to reach a local minimum in quite few iterations with respect to lower values of k. This might be due to the fact that by increasing k we also enlarge the size of the neighborhood explored at each iteration of a local search.

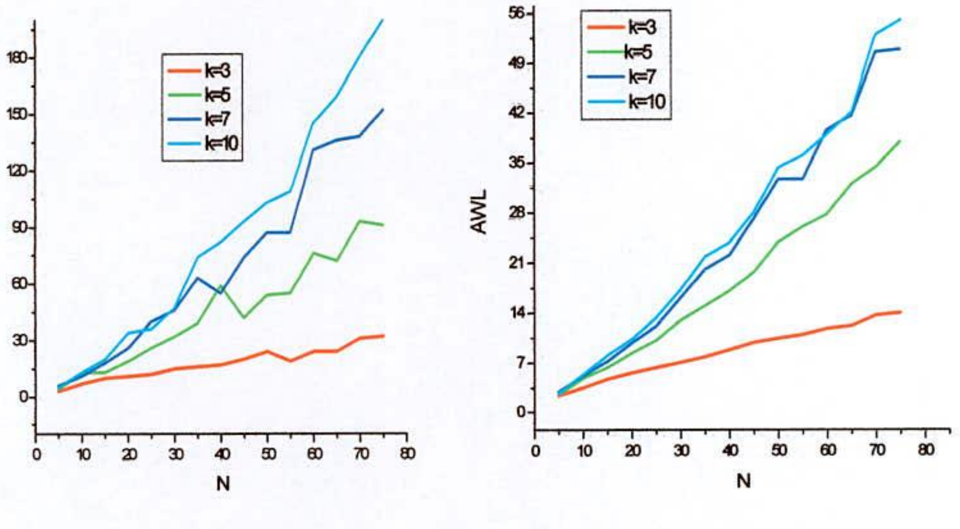

Figure 5.2: The impact of N on (a) MWL (b) AWL

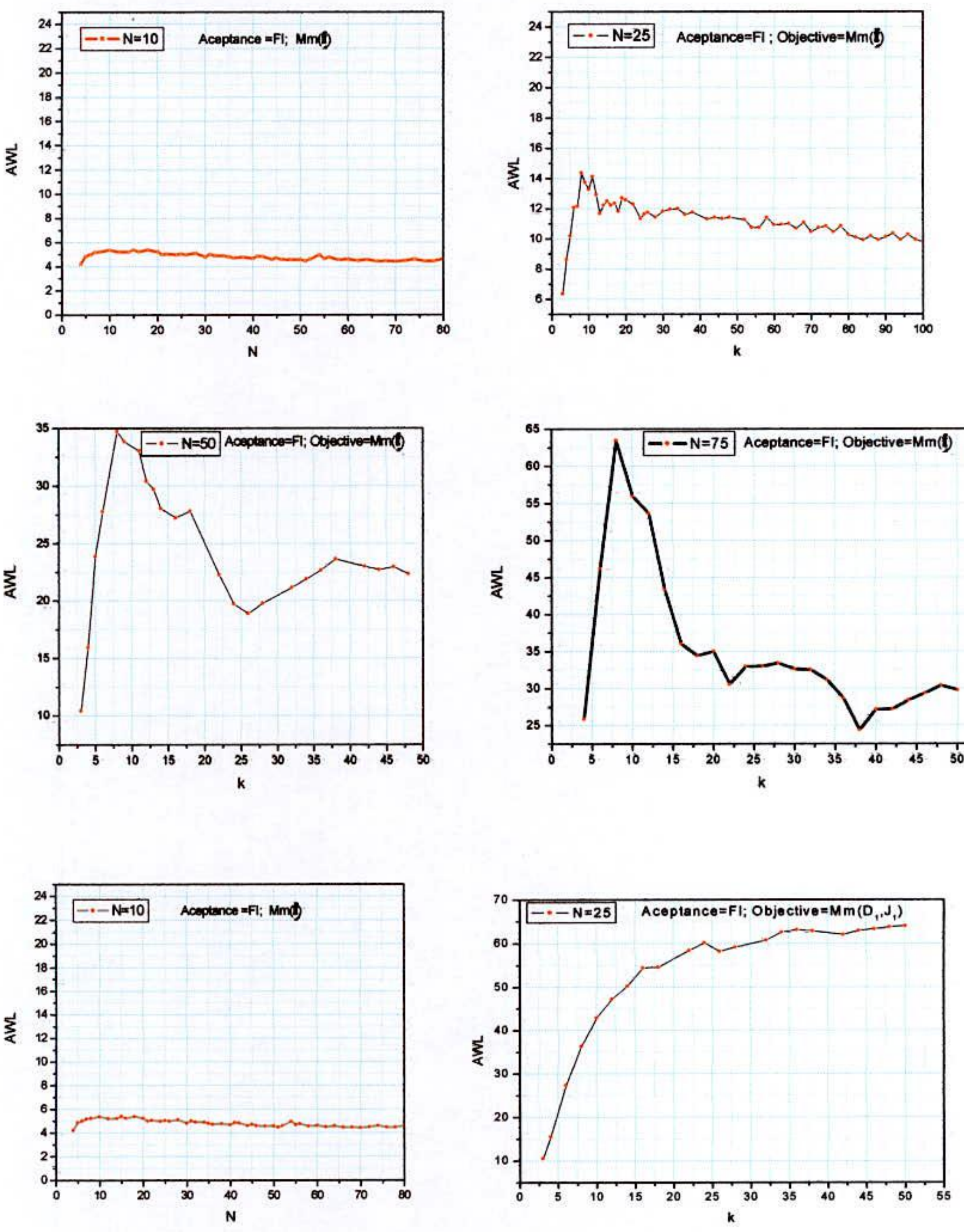

Figure 5.3: The Impact of *k* on AWL

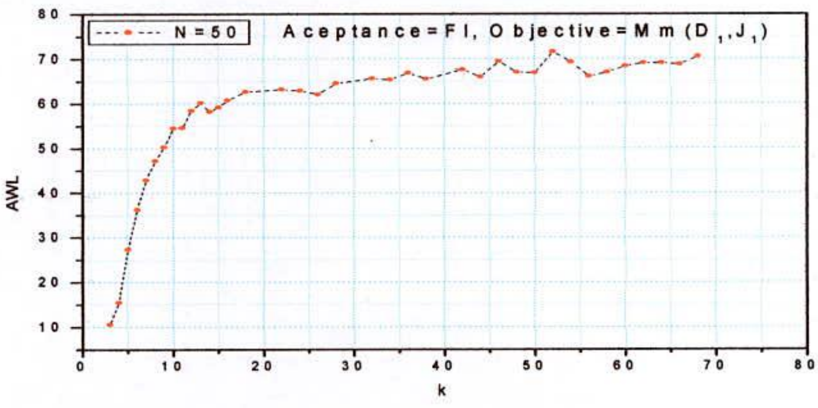

Figure 5.4: The Impact of k on execution of AWL during Local Search with FI(First

In what follows we will neglect the dependency of WL on k and only consider WL as O(N), but we should keep in mind that at small k values a dependency of WL on k is in fact present. Since when testing ILS(D_1) we employed the BI acceptance criterion, we would like to check, for completeness, if such phenomenon, i.e. the non dependency of WL on k at large k values, is somehow connected to the fact that we have considered the FI acceptance criterion. For this reason, we have performed another experiment with ILS(D_1) but with the FI acceptance criterion. We considered LHDs: $k = 2i : i = 1, 2, ...,40$ with $N = 10, 50$. We observe in Figure 5.4 that WL increases quickly at small k values, while at large k values WL still increases, though more slowly. Such behavior is quite similar to the one observed in ILS(D_1) with the BI acceptance criterion. If we put together the expected times for all the components of a local search, we can conclude that the approximate time required by a local search is

$$T \approx O(k^r N^q),$$

where we expect that the q value is close to 4, while the r value could range between 1 and 2. In order to find out the values of q and p experimentally, we performed the following experiments. At first we perform experiments to find the approximate value of q. For these experiments we considered $k = 5$ and $N = 20, 21,........ 80$ and run ILS(φ) ten times for each LHD. In Figure 5.5, we plotted the average execution time per local search as a function of

N. We observe that the increase with N is non linear. Therefore, we applied the logarithmic transformation

$$\log T = q(\log N - b),$$

where T denotes the average elapsed time, and then fitted the data in a linear regression. According to the data, we have that the value of q is 3.92 (see Figure 5.6), thus very close to the expected one, 4.

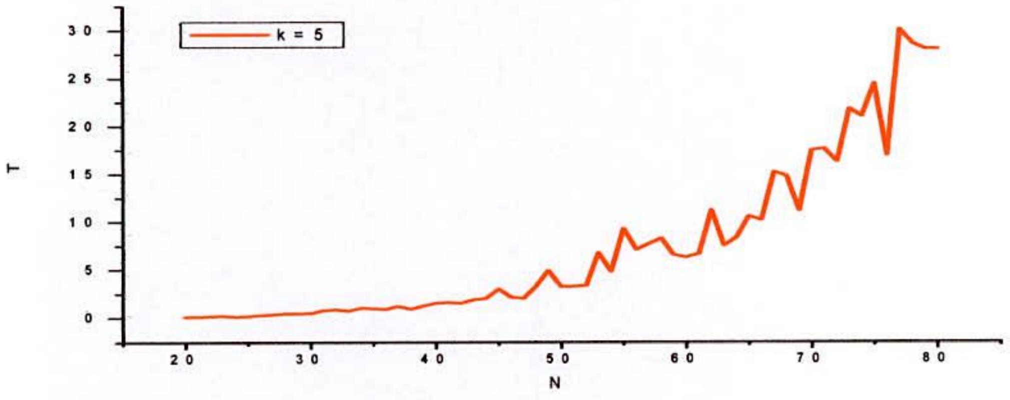

Figure 5.5: Elapsed time per local search as a function of

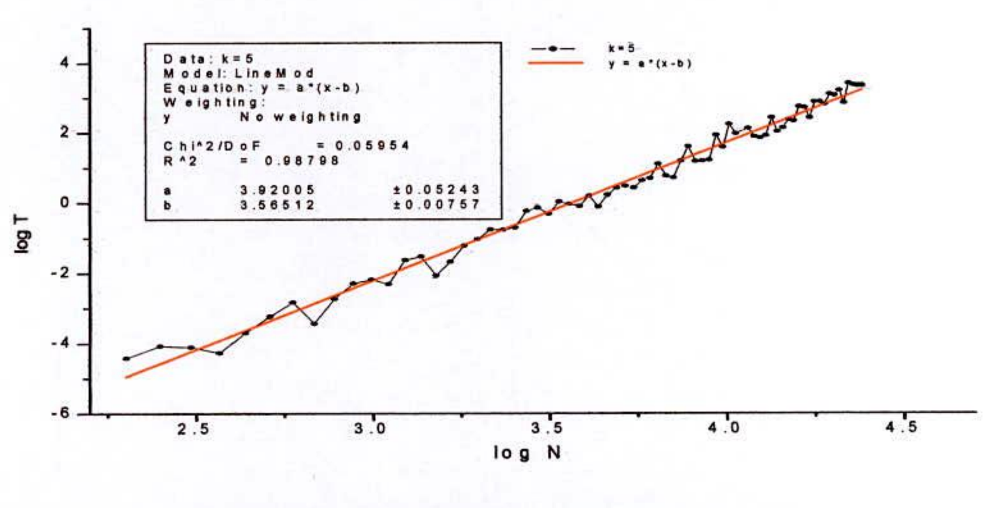

Figure 5.6: Linear regression between log(T) and log(N)

Now to find out the approximate value of r we performed experiments by considering LHDs with $k = 2i : i = 1, 2, \ldots, 25$ with $N = 50$. Figure 5.7 shows the impact of k on the average elapsed time per local search. In the figure we observe that T increases somewhat linearly with the increase of k. In order to find out the approximate time complexity with respect to k, we have fitted the data (see Figure 5.8) and detected a value of r approximately equal to 1.13, again in accordance with what previously derived .

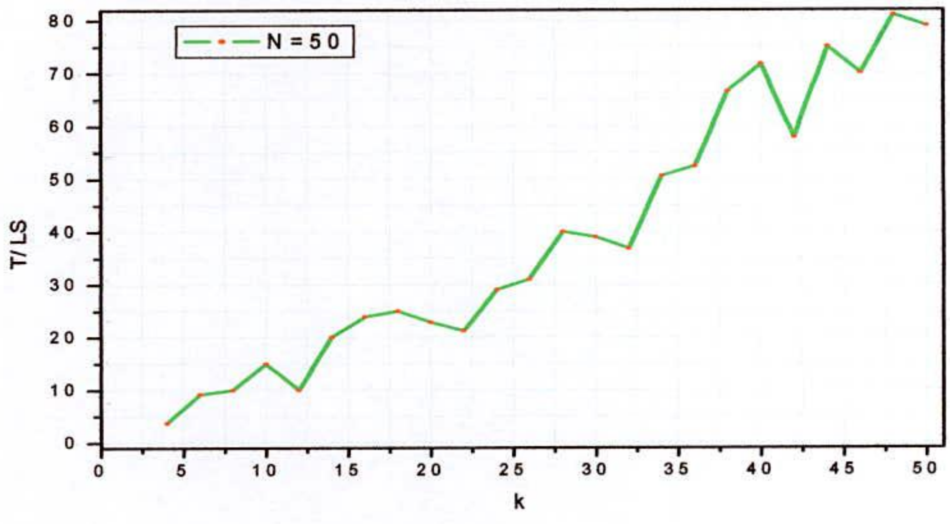

Figure 5.7: Impact of k on T

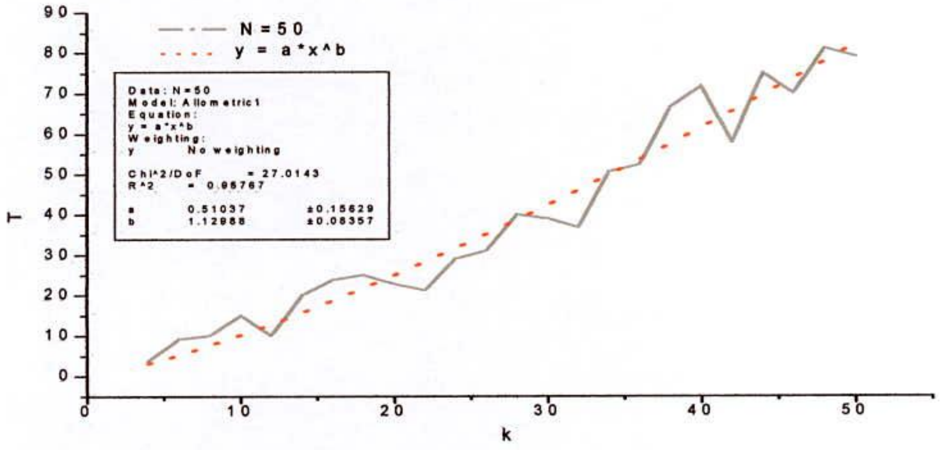

Figure 5.8: The approximate time complexity of k for LS obtained by the experiments

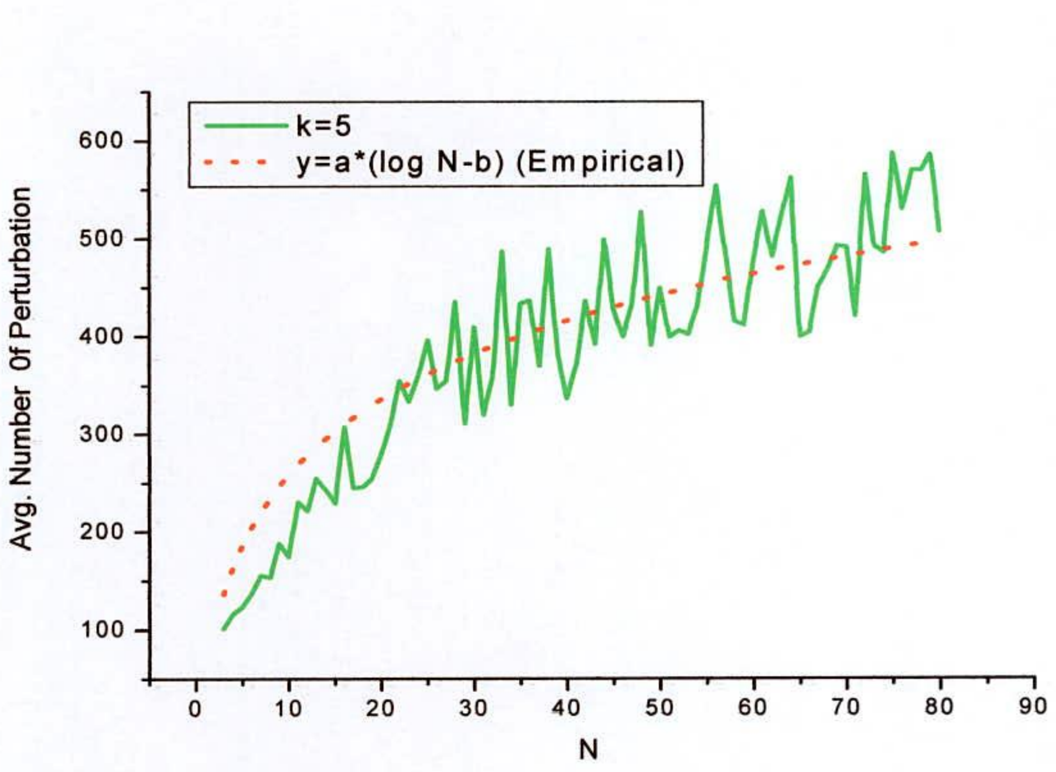

Figure 5.9: Relation between the number of perturbations and N

In order to derive the overall time complexity of ILS(ϕ) we still need to derive a formula for the number of perturbations (i.e. the number of local searches) performed during each ILS run. To find out the impact of N on such number we considered LHDs with $k = 5$ and $N = 3, 4, \ldots, 80$ performing ten runs for each LHD. From the experiments (see Figure 5.9) we notice that there is a significant impact of N on the number of perturbations. We also try to establish a functional relation between N and the number of perturbations. Similarly to what already observed for ILS(D_1), the relation appears to be a logarithmic one with respect to N (see the dot curve in Figure 5.9). We point out that in both cases such logarithmic behavior is probably due to the fact that a fixed value for MaxNonImp (100 for ILS(ϕ), 1000 for ILS(D_1)) has been employed in all these tests, so that the total number of perturbations tends to get stable as we increase N.

To find out the impact of k on the number of perturbations, we considered LHDs with $N = 50$ and $k = 2i : i = 1, \ldots, 10$. From the experiments we notice that, in spite of a peak at $k = 6$, there is no significant impact of k on the number of perturbations invoked during a run (see the bar diagram in Figure 5.10).

In conclusion, summing up all the previous observations, we have that the time required for a single ILS(ϕ) run appears to be $O(k^r N^q \log(N))$, with r slightly larger than 1 and q slightly lower than 4.

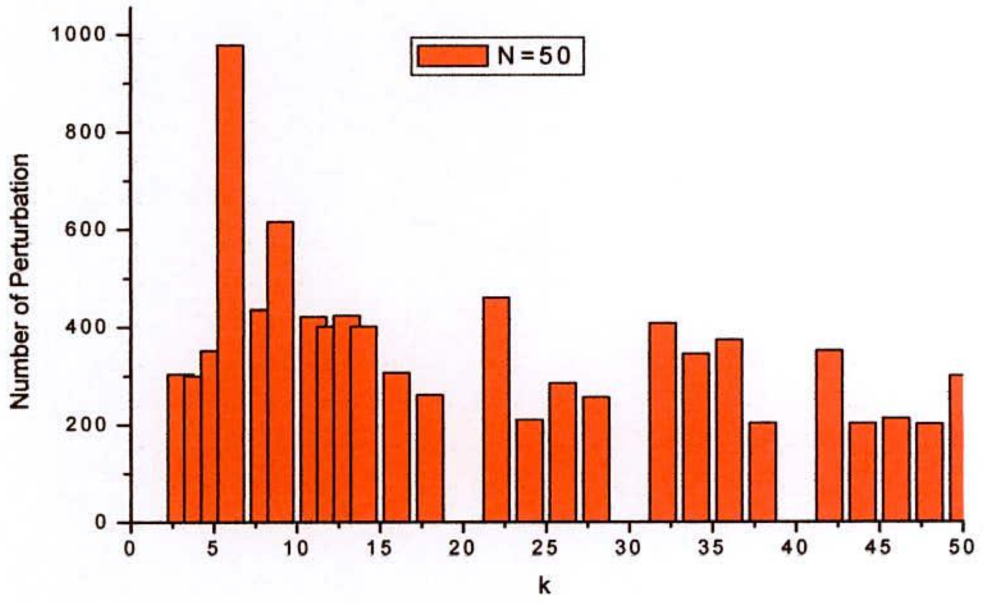

Figure 5.10: Impact of k on the number of perturbations

Finally, the analysis of time complexity for ILS (Φ) is given in a tabular form along with the pseudo code of the algorithm. Table 5.2 represents the analysis of the time complexity for the ILS (Φ).

Table 5.2: Analysis of time complexity for ILS (**Φ**)

(*i*) Compute $d_{i',u}^{(s+1)} \xrightarrow{Operation} O(1)$

(comp.only swapped cord.)

(*ii*) Step 3 (in worst case) $\xrightarrow{Operation}$ $O(N$

(since compu. stop as soon as $d_{i',u}^{(s+1)} \leq D_1')$

(*iii*) Inner most for loop $\xrightarrow{Operation} O(k)$

(in BI local move)

(*iv*) outer two for loops $\xrightarrow{Operation}$ $O(N$

(in worse case theoretic ally $O(N^2)$)

(but experiment ally swap $O(N)$)

(*v*) WL (in local search) $\xrightarrow{Operation} O(Nk^c)$

$(0 < c < 1$), experimentally computed

Total cost of a single LS

$O(1).O(N).O(k).O(N).O(Nk^c) \approx O(k^r N^q)$

$:(1 < r < 2$) & $(2 < q \leq 4$)

Cost of a single ILS (Φ)

Perturbation (for fixed MIN))

O (log(N))

Total Cost:

$O(k^r N^q).O(\log(N)) \approx O(k^r N^q \log(N))$

$:(1 < r < 2$) & $(2 < q \leq 4$)

```
While do
  Set NonImpIteration = 0
  Whiledo
    for  $i = 1,\ldots\ldots, N$ do
      for $j = i + 1,\ldots\ldots, N$ do
        Step 1= $D_1 = D_1^{(S)}$, $k' = 0$
          for $l = 1,\ldots., k$ do
            Step 2: Swap ($X_{il}, X_{jl}$)
            Step 3: Compute $d_{i',u}^{(X+1)}$
            until  $d_{i',u}^{(X+1)} \geq D_1'$
            $\forall i' =$ i .j:
            u = 0.........N-1;
            u $\neq i'$
            else break
            Step 4 :Set  $k' = k$
            and  $D_1' =$ min  $d_{IJ}^{(X+1)}$
          end for
        Step 5: Upload best LHD if any
              else continue
      end for
    end for
   Step 6: Repeat the three loops if
   there has been at least an
   improvement
  Otherwise STOP
  Return  X'
  Step 7: if X'  is better then set X= X'
and NonimpIteration = 0
   Otherwise increase
NonimpIteration by one
   Step 8: if MaxNonImp >
NonimpIteration
 $PM : X' = \in (X)$ and Repeat the loops
Otherwise STOP
Return X
```

CHAPTER VI

COMPUTATIONAL EXPERIMENT OF ILS FOR HIGHER DIMENSIONS

In chapter IV and V we have performed several experiments to find out the time complexity in experimental domain. Grosso et al. (2008) have shown that, for finding maximin LHDs, ILS approach outperforms when number of dimensions (k) are less than 11. The outperforms results are available in the well-known website http://www.spacefillingdesigns.nl/. But they could not performed experiments higher than 10 dimension. It is also noted that few results (in the cases of dimensions greater than 10) are available in the literatures which are reported in the website http://www.spacefillingdesigns.nl/ . Here we will perform several experiments for higher dimensions typically k greater than 10 regarding ILS approaches. Also the results are compared with available one in the literature i.e. in the above mentioned website.

Table 6.1: The setting of number of runs for the ILS

k	N	R
11 – 15	3-25	2

For ILS approach, we set RP local moves with BI acceptance rule in local search, SPC. perturbation and MaxNonImp=100. We also consider the Opt(Φ) optimality criterion i.e. we consider ILS(Φ) approach. the value of p is equal to 20 as well. In what follows the approach will be simply denoted as ILS(Φ). For what concerns the number of runs for each LHD, we considered is given in Table 6.1. In the table R denotes the number of runs (trials) for each experiment(k, N): $k = 11, 12, \cdots, 15$; $N = 3, 4, \cdots, 25$;. It is noted that for dimension $k = 11$, besides $R = 2$ we have also considered $R = 5$ and 10. The experimental results are reported in the Table 6.2(a), Table 6.2(b) and Table 6.2(c) where the distances are measured in Euclidian measure and the values are the best maximin squared distance rather than actual distances each for (k,N). It is noted that the 2nd column of the tables 6.2(a), 6.2(b) and 6.2(c) denotes the maximin LHDs values available in the website

Table 6.2 (a): Comparison of maximin LHD values for dimensions $k = 11$

Points(N)	**Web Value**	**ILS Value**		
		Trial 2	**Trial 5**	**Trial 10**
3		20	20	20
4	35	34	34	35
5	54	52	54	54
6	74	73	74	74
7		98	98	98
8		125	125	126
9		156	157	158
10		190	190	191
11		227	229	229
12		270	270	272
13		306	306	309
14		350	350	350
15		393	397	397
16		445	445	448
17		501	501	501
18		560	560	561
19		619	620	623
20		685	691	692
21		760	767	767
22		844	851	857
23		848	857	857
24		907	925	925
25		967	974	984

http://www.spacefillingdesigns.nl/. In the tables we have observed that few web values (maximin LHDs values available in the web) are available for higher dimensions. It is observed that the ILS values are identical with Web values. It worthwhile to mention here that these values are analytically global optimal rather than approximate optimal. Moreover The ILS approaches able to obtain some new maximin LHDs value for $6<N<26$ which are reported in the tables 6.2(a), 6.2(b) and 6.2(c). There is another observed in the table 6.2(a) that the increasing of number of trials do not significantly increasing the maximum LHDs values. From this observation it may conclude that for higher dimensions, few trials of ILS approach able to find approximate optimal LHDs.

Table 6.2 (b): Comparison of maximin LHD values for dimensions $k = 12$ and 13

$k = 12$			$k = 13$		
Point(N)	Web Value	ILS Value	Point(N)	Web Value	ILS Value
3		24	3		25
4	40	40	4	41	41
5	60	60	5	64	64
6	82	82	6	89	89
7		107	7		117
8		137	8		150
9		172	9		187
10		209	10		229
11		251	11		274
12		296	12		322
13		347	13		376
14		351	14		434
15		438	15		605
16		495	16		669
17		550	17		487
18		611	18		540
19		684	19		745
20		758	20		823
21		830	21		910
22		908	22		997
23		999	23		1083
24		1105	24		1179
25		1230	25		1290

Table 6.2 (c): Comparison of maximin LHD values for dimensions $k = 14$ and 15

$k = 14$			$k = 15$		
Point(N)	Web Value	ILS Value	Point(N)	Web Value	ILS Value
3		26	3		30
4	46	46	4	48	48
5	70	70	5	74	74
6		95	6		103
7		127	7		136
8		162	8		173
9		202	9		217
10		249	10		266
11		293	11		319
12		349	12		374
13		404	13		436
14		464	14		504
15		537	15		574
16		595	16		653
17		660	17		721
18		731	18		797
19		815	19		883
20		897	20		969
21		987	21		1064
22		1079	22		1063
23		1183	23		1271
24		1284	24		1376
25		1397	25		1502

It is also noted that the ILS (Φ) approach proposed by Grosso et al. (2008) does not consider the LHD with corresponding optimal (minimum) Φ value but tracking the optimal maximin LHD (whose minimum pair-wise distance is maximum) during minimizing Φ value. Here several experiment are performed to analyze the above discussion. For these experiments we consider $k = 11$, $N = 3, 4, \ldots, 25$ and number of trials $R = 2, 5, 10$. Other parameter setting are same. The experimental results displayed in table 6.3. It is observed form the table that though increasing of number of trials causes monotonic increasing of maximin LHDs values but corresponding Φ values do not necessarily monotonic decreasing. In the table we also observed that some time lower trial may better results. This implies that initial solution may affect the final output.

Table 6.3: Comparison of maximin LHD values and Φ values for dimensions $k = 11$

N	LHD values				Φ values		
	R=2	R=5	R=10		R=2	R=5	R=10
3	20	20	20		0.456287	0.456287	0.456287
4	34	35	35		0.546794	0.513469	0.513469
5	54	54	54		0.565956	0.60572	0.60572
6	74	74	74		0.653762	0.653762	0.653762
7	97	97	99		0.690905	0.690839	0.690505
8	126	125	125		0.721167	0.721102	0.721102
9	154	154	156		0.747052	0.746939	0.746679
10	189	193	193		0.768803	0.768375	0.768375
11	226	225	225		0.787288	0.787281	0.787281
12	272	272	272		0.803762	0.803513	0.803513
13	309	309	309		0.824438	0.824438	0.824438
14	341	341	341		0.842359	0.842359	0.842359
15	383	383	390		0.857904	0.857904	0.857816
16	443	443	436		0.871477	0.871477	0.871336
17	495	495	504		0.882531	0.882531	0.882233
18	557	557	554		0.893106	0.893106	0.892659
19	625	625	625		0.901792	0.901792	0.901792
20	683	682	684		0.910757	0.910681	0.910578
21	750	768	748		0.918572	**0.918509**	*0.918449*
22	848	848	848		0.924703	0.924703	0.924703
23	850	850	850		0.942046	0.942046	0.942046
24	901	908	908		0.952664	0.95243	0.95243
25	958	958	950		0.965299	0.965299	0.965077

It is worthwhile to mention here that for all experiments performed earlier all the distances measured in L^2 measure. But when maximize the minimum pair-wise distance in L^2 measure might causes L^1 measure too. That is in any experimental design when the minimum pair-wise distance is increasing in L^2 measure, then the minimum pair-wise distance in L^1 measure should be increase but not necessarily monotonic. The experimental results are reputed in the table 6.4. It is noted that the L^1 value reputed in the table 6.4 are measured from the LHDs which are maximin LHD in L^2 measure by the ILS approach but not maximin LHDs in L^1 measure. Since there are no any value available in the literatures, so we could not compare the results.

Table 6.4: Experimental results of maximin L^1 values corresponding to optimized LHD values measured in L^2 measure for $k = 11$ - 15

N	$k = 11$	$k = 12$	$k = 13$	$k = 14$	$k = 15$
3	14	16	17	18	20
4	17	20	21	23	24
5	22	24	26	28	30
6	23	27	28	31	34
7	27	29	32	34	36
8	30	32	36	39	42
9	32	37	39	41	44
10	36	41	41	47	51
11	40	42	46	50	55
12	43	45	49	56	59
13	44	50	55	59	61
14	45	54	57	59	66
15	47	55	60	64	70
16	51	56	64	68	74
17	56	58	68	72	76
18	60	62	71	78	80
19	59	67	74	82	85
20	61	70	76	84	88
21	68	73	80	87	96
22	66	77	85	89	96
23	71	80	87	96	103
24	76	86	86	95	106
25	76	83	96	97	105

Table 6.5: Experimental results of maximum average coefficient of correlation of the co-factors of the maximin LHDs for $k = 11$ - 15

N	$k = 11$	$k = 12$	$k = 13$	$k = 14$	$k = 15$
3	0.953463	0.953463	0.960769	0.963624	0.963624
4	0.738549	0.738549	0.748331	0.751263	0.758445
5	0.592376	0.603023	0.612896	0.620174	0.627163
6	0.490927	0.505309	0.517086	0.526916	0.535017
7	0.409534	0.427894	0.441798	0.453797	0.463749
8	0.340129	0.362158	0.379442	0.393168	0.404955
9	0.27707	0.30364	0.324707	0.341046	0.35451
10	0.215443	0.249506	0.273677	0.294027	0.309932
11	0.14771	0.193683	0.226577	0.250427	0.268747
12	0.038924	0.134783	0.177042	0.207415	0.229611
13	0.135165	0.036809	0.124208	0.163625	0.191376
14	0.154653	0.121373	0.030303	0.113747	0.151578
15	0.162267	0.148942	0.114768	0.029532	0.104976
16	0.157082	0.154097	0.134213	0.105412	0.026695
17	0.149606	0.148493	0.144009	0.127823	0.097976
18	0.136881	0.139549	0.14477	0.135268	0.121348
19	0.118659	0.134324	0.134428	0.137081	0.131929
20	0.096469	0.125651	0.137371	0.142735	0.133814
21	0.07354	0.10854	0.127346	0.1344	0.13852
22	0.018811	0.090604	0.114545	0.12536	0.135415
23	0.046038	0.068007	0.10028	0.116373	0.129211
24	0.075818	0.018534	0.082387	0.107147	0.117659
25	0.089355	0.044584	0.060996	0.093726	0.109148

It is also remarks that multicollinearity is another important properties of an experimental design. A good experimental design should minimum multicollinearity among the factors along with other two properties. Then the measure the multicollinearity among the factors can be defined by the following measure of average pair-wise correlations

$$\rho^2 = \frac{\sum_{i=2}^{k}\sum_{j=1}^{i-1}\rho_{ij}^2}{k(k-1)/2} \tag{6.1}$$

Where ρ_{ij} denotes the product-moment correlation between the i-th and j-th factors. Note that this definition is frequently used in literature [Fang et al. (2000b), Joseph and Hung (2008)]. Whereas the definition of maximum pair-wise correlation is given below:

$$\rho_{\max} = \max_{1\le i,j\le k} \rho_{ij} \tag{6.2}$$

Now another experiment is performed by considering the same setting as considered above. The experimental results are plotted in the table 6.5. It is observed that when the number of design points (i.e N) is small the maximum coefficient of correlation is high. Whereas the coefficient of correlation is negligible as well as decreasing when increasing the N values.

It is noted that in any experimental design when the minimum pair-wise distance is increasing, then the maximum pair-wise distance should be decreasing but not necessarily monotonic. So minimizing the maximum pair-wise distance may be the another optimal criterion for optimal the experimental design. So another experiments is performed by considering the same setting as considered above. The experimental results are plotted in the table 6.5. In the table 6.6, L_M^1 and L_M^2 denote the maximum pair-wise distance of the maximin LHDs measured in L^1 and L^2 distance measure respectively. As there are no value available in the literature, so we could not compare the experimental results.

Table 6.6: Experimental results of maximum pair-wise distance value (L_M^1 and L_M^2) of the maximin LHDs for $k = 11$ - 15

N	L_M^1					L_M^2				
	$k = 11$	$k = 12$	$k = 13$	$k = 14$	$k = 15$	$k = 11$	$k = 12$	$k = 13$	$k = 14$	$k = 15$
3	15	16	18	19	20	23	24	28	29	30
4	20	20	24	24	27	42	40	48	47	55
5	22	24	26	28	30	56	60	66	70	76
6	27	29	32	34	36	80	87	96	101	108
7	32	35	37	39	43	108	118	125	137	148
8	36	39	42	46	48	146	152	162	176	188
9	40	44	47	50	55	174	192	208	222	234
10	45	48	53	56	60	218	238	248	270	290
11	48	53	57	60	67	256	279	307	327	347
12	52	58	63	66	70	305	332	359	380	408
13	85	62	67	72	79	695	395	416	461	482
14	87	94	74	78	82	787	866	482	526	553
15	89	99	107	82	87	887	991	1075	604	628
16	94	104	110	125	93	968	1103	1194	1297	721
17	100	111	120	129	146	1088	1204	1348	1451	1572
18	105	118	127	134	143	1221	1356	1481	1612	1747
19	109	117	134	144	153	1339	1463	1615	1783	1940
20	117	129	137	149	160	1481	1619	1781	1920	2109
21	122	135	140	155	168	1650	1781	1937	2081	2241
22	127	145	151	167	177	1795	1962	2131	2291	2466
23	134	146	158	169	187	1948	2126	2328	2511	2700
24	143	154	169	181	195	2011	2348	2512	2717	2914
25	144	160	169	190	203	2139	2497	2743	2969	3152

REFERENCES

01. Aparna D., 2012 , "Iterated Local search Approaches For Maximin Latin Hypercube Designs", M. Phil thesis paper, Department of Mathematics, Khulna university of Engineering & Tecnology, Khulna.

02. Applegate D., W. Cook and A. Rohe, 1999, "Chained Lin-Kernighan for large traveling salesman problems", Technical Report No. 99887, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany.

03. Arora S., Barak, B., Brunnermeier M., and Ge. R., 2011(May), "Computational Complexity and Information Asymmetry in Financial Products", Communication of the ACM, Vol. 54, pp. 101-106.

04. Audze P., and V. Eglais, 1997, "New approach to planning out of experiments, problems of dynamics and strength", Vol. 35, pp. 104-107.

05. Bates S. J., Sienz J. and Langley D.S., 2003, "Formulation of the Audze-Eglais Uniform Latin Hypercube design of experiments", Advanced in Engineering Software, Vol. 34, Issue 8, pp. 493-506.

06. Battiti R., and Protasi M., 1997, "Reactive search, a history-based heuristic for the MAX-SAT", ACM Journal of Experiments Algorithmic, Vol. 2.

07. Baum, E. B., 1986(b), "Iterated descent: A better algorithm for local search in combinatorial optimization problems", Technical report, Caltech, Pasadena, CA Manuscript.

08. Baum, E. B.,1986(a), "Towards practical "neural" computation for combinatorial optimization problems", In J. Denker, editor, Neural Networks for Computing, pp. 53–64, AIP conference proceedings.

09. Blondel V. D., Tsitsiklis J. N., 2000, "A survey of computational complexity results in systems and control", Vol. 36, pp. 1249-1274, www.elsevier.com/locate/automatic.

10. Butler N. A., 2001 "Optimal and orthogonal Latin Hypercube designs for computer experiments", Biometrika, Vol. 88(3), pp. 847-857.

11. Baxter, J., 1981, "Local optima avoidance in depot location", Journal of the Operational Research Society, Vol. 32, pp. 815–819.

12. Crary S. B., Cousseau P., Armstrong D., Woodcock D. M., Mok E. H., Dubochet O., Lerch P., and Renaud P., 2000, "Optimal design of computer experiments for metamodel generation using I-OPTTM", Computer Modeling in Engineering & Sciences, Vol. 1(1), pp. 127-139.

13. Crombecq K., Laermans E., and Dhaene T., 2011, "Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling", *European Journal of Operational Research*, Vol. 214(3), pp. 683-696.

14. Currin C., Mitchell, T., Morris, M. D., and D. Ylvisaker, 1991, " Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments", Journal of the American Statistical Association, Vol. 86, pp. 953-963.

15. Fang, K. T., D. K. J. Lin, P. Winkler, and Y. Zhang (2000b), "Uniform design: theory and application", Technometrics, Vol. 42, pp. 237–248.

16. Fang K. T., R. Li, and A. Sudjianto, 2006, "Design and Modeling for Computer Experiments", CRC Press, New York.

17. Felipe A.C. Viana, Venter G., 2009(Oct), "An Algorithm for Fast Optimal latin Hypercube Design of Experiments", pp.1-4, DoI: 10.1002/nme.2750

18. Fuerle F. and Sienz J., 2011, "Formulation of the Audze-Eglais Uniform Latin Hypercube design of experiments for constrained design spaces", Advanced in Engineering Software,Vol. 42, pp. 680-689.

19. Grosso A., Jamali A. R. M. J. U. and Locatelli M., 2008, " Iterated Local Search Approaches to Maximin Latin Hypercube Designs", Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering, Springer Netherlands, pp. 52-56.

20. Glover F., and Laguna M. ,1993, "Modern heuristic techniques for combinatorial problems", Oxford: Blackwell, pp. 70150

21. Grosso A., Jamali A. R. J. U. and Locatelli M., 2009, " Finding Maximin Latin Hypercube Designs by Iterated Local Search Heuristics", *European Journal of Operations Research, Elsevier,* Vol. 197, pp. 541-547.

22. Grassberger P., 1997, "Pruned-enriched Rosenbluth method: Simulations of θ polymers of chain length up to 1000000", Phys. Rev., Vol. 56(3), pp. 3682-3693

23. Hadzilacos, V., "Time complexity of algorithms", http://www.cs.toronto.edu/~vassos/teaching/c73/handouts/brief-complexity.pdf

24. Helton J. C. & Davis F. J. 2000, "Sampling-based methods, in Sensitivity Analysis", Ed. A. Saltelli, K. Chan and E. M. Scott, Chichester: John Wiley & Sons.
25. Heylighen F., (1990): "Relational Closure: a mathematical concept for distinction-making and complexity analysis", in: Cybernetics and Systems '90 , R. Trappl (ed.), (World Science Publishers), pp. 335-342.
26. Husslage B., E. R. van Dam, and D. den Hertog, 2005, "Nested maximin latin hypercube designs in two dimensions", CentER Discussion Paper No. 200579.
27. Husslage B., G. Rennen, E. R. van Dam, and D. den Hertog, 2006, "Space-Filling Latin Hypercube Designs for Computer Experiments", CentER Discussion Paper No.2006-18.
28. Husslage B., Rennen G., van Dam E. R., and Hertog D. den, 2006, " Space–Filling Latin Hypercube Designs for Computer Experiments", CentER Discussion Paper No. 2006-18
29. Husslage B., Rennen G, Van Dam E. R., and Hertog D, 2006 "Space-Filling Latin Hypercube Designs for Computer Experiments", CentER Discussion Paper No. 2006-18
30. Iman R. L., and Conover W. J., 1982(a), "A distribution-free approach to inducing rank correlation among input variables", Comm. Stat. Part B – Simulation Comput., Vol. 11, pp. 311-334.
31. Iman, R. L. and Conover W. J. (1982b), "Small-sample sensitivity analysis techniques for computer models, with an application to risk assessment. Communications in Statistics – Part A", Theory and Methods 17, 1749–1842.
32. Johnson M. E., Moore L. M., and Ylvisaker D., 1990, " Minimax and maximin distance designs", Journal of Statistical planning and inference, Vol. 26, pp.131-148.
33. Jin R., W. Chen, and A. Sudjianto, 2005, "An efficient algorithm for constructing optimal design of computer experiments", Journal of Statistical Planning and Inference, Vol. 134(1), pp. 268-287.
34. Joseph V. R. , and Hung Y., 2008, " Orthogonal-Maximin Latin Hypercube Designs", Statistica Sinica, Vol. 18, pp. 171-186.
35. Jourdan A. and Franco J., 2010, " Optimal Latin Hypercube designs for the Kullback-Leibler Criterion", AStA Advances in Statistical Analysis, Springer-Verlag, Vol. 94, pp. 341-351, DOI: 10.1007/s10182-010-0145-y.
36. Kirkpatrick S., C. D. Gelatt Jr., and M. P. Vecchi, 1983, "Optimization by Simulated Annealing , Science, Vol. 220, pp. 671-680.

37. Kleijnen, J. P. C., 1997, " Sensitivility analysis and related analysis: a review of some statistical techniques", J. Statist. Comp. Simul., Vol. 57, pp. 111- 42.

38. Levy S., Steinberg D. M., 2010, Computer experiments, Adv. Stat. Anal. Vol. 94(4), pp. 311-324.

39. Li W. W., and C. F. J. Wu, 1997, "Columnwise-Pairwise Algorithms With Applications to the Construction of Supersaturated Designs", Technometrics, American Society for Quality Control and American Statistical Association, Alexandria, Va, USA, Vol. 39(2), pp. 171-179.

40. Liefvendahl M., and R. Stocki, 2006(1 september), "A study on algorithms for optimization of Latin hypercubes, Journal of Statistical Planning and Inference", Vol. 136 (9), pp. 3231-3247.

41. Lin D. K. J., and D. M. Steinberg, 2006, "A Construction Method for Orthogonal Latin Hypercube Designs", Biometrika, Oxford University Press, Vol. 93(2), pp. 279 -288.

42. Lourenco H. R. et al., 2002, " In Iterated Local Search Handbook of Metaheuristics", ISORMS 57(Eds.: Glover F., and G. Kochenberger,), Kluwer, pp. 321-353.

43. Lourenco H. R., O. Marting, and T. Stutzl, 2001, "A beginner's introduction to Iterated Local Search", In Proceedings of MIC'2001-Meta-heuristics International Conference, Porto-Portugal, Vol. 1, pp. 1-6.

44. Martin O. and S. W. Otto., 1996, "Combining simulated annealing with local search heuristics", Annals of Operations Research, Vol. 63, pp. 57–75.

45. Martin O., S. W. Otto, and E. W. Felten, 1991, " Large-step Markov chains for the traveling salesman problem", Complex Systems, Vol. 5(3), pp. 299–326.

46. McKay M. D., Beckman, R. J., and W. J. Conover W. J., 1979, " A comparison of three methods for selecting values of input variables in the analysis of output from a computer code", Technometrics, vol. 21, pp. 239-245.

47. Morris M. D., 1991, "Factorial plans for preliminary computational experiments", Technometers, Vf'urol. 33, pp.161-174.

48. Morris M. D., and Mitchell T. J., 1995, "Exploratory designs for computer experiments", Journal of Statistical Planning and Inference, Vol. 43, pp. 381-402.

49. Nicolas S., 2006-2007(Nov), "Algorithms & Complexity-Introduction", nstropa@computing.dcu.ie, CA313@Dubai City University.

50. Oliveto P. S., He J., Yao X., 2007, "Time Complexity of Evolutionary algorithms for Combanatories Optimization: A Decade of Results", International Journal of Automation and Computing, Dol: 10.1007/s11633-007-0281-3, Vol. 04(1), pp. 281-293.
51. Owen, A. B., 1994, " Controlling correlations in Latin hypercube samples", Journal of the American Statistical Association,Vol. 89, pp. 1571–1522.
52. Park J. S., 1994, "Optimal Latin hypercube designs for computer experiments", Journal of Statistical Planning and Inference, Vol. 39, pp. 95-111.
53. Rossi-Doria O., M. Samples, M. Birattari, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquete, B. Paechter, and T. Stutzle, 2002, " A Comparison of the performance of different metaheuristics on the timetabling problem", In Proceedings of PATAT 2002, The 4th international conference on the Practice and Theory of Automated Timetabling, Gent, Belgium, pp. 115-119.
54. Sacks, J. and Ylvisaker, D. (1985), "Model robust design in regression: Bayes theory". In Proc. of the Berkeley Conference in Honor of Jerzy Neyman and Jack Kiefer (L. M. Le Cam and R. A. Olshen, eds.), Vol. 2, pp. 667-679, Wadsworth, Monterey, Calif.
55. Santner T. J., Williams B. J., and Notz W. I., 2003, "The design and analysis of computer experiments", Springer Series in Statistics, Springer-Verlag, New York.
56. Shewry M., and Wynn H.,1987, "Maximum entropy design" Journal of Applied Statistics, Vol.14, pp. 165-170.
57. Steinberg G. D. M, and K. J. N. Dennis, 2006, " A construction method for orthogonal Latin hypercube designs", Biometrika, Vol. 93 (2), pp. 279-288.
58. St¨utzle, T., 1998, " Local Search Algorithms for Combinatorial Problems — Analysis", Improvements, and New Applications. PhD thesis, Darmstadt University of Technology, Department of Computer Science.
59. www.wikipedia.org/w/wiki.phtm?tittle=Big O notation
60. www.cs.toronto.edu/~vassos/teaching/c73/handouts/brief-com
61. www.spacefillingdesigns.nl
62. Ye, K. Q., 1998, "Orthogonal column Latin hypercubes and their application in computer experiments" Journal of the American Statistical Association, Vol. 93, pp.1430-1439.

63.Ye, K. Q., W. Li, and A. Sudjainto, 2000, “Algorithmic construction of optimal symmetric Latin hypercube designs”, Journal of Statistical Planning and Inference, Vol. 90, pp. 145-159.