Global Scientific JOURNALS

GSJ: Volume 9, Issue 2, February 2021, Online: ISSN 2320-9186
www.globalscientificjournal.com

COMPLEXITY ANALYSIS OF ITERATED LOCAL SEARCH ALGORITHM IN EXPERIMENTAL DOMAIN: CONSIDER ILS (D_1, J_1) AND OPT (Φ).

Parimal Mridha, Lecturer, Mathematics Department,
Military Collegiate School Khulna (MCSK), Bangladesh.
parimalmridha@yahoo.com

Abstract: *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. 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.*

Keywords: *Computational complexity, Local search, Opt (D_1, J_1), Opt(Φ), Multi-collinearity.*

1. INTRODUCTION

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. 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.

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.

2. Iterated Local Search

Iterated Local Search (ILS) is a meta-heuristic designed to embed another, problem specific, local search as if it were a black box. This allows ILS 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)]. This simple idea [Baxter et al. (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.et al. (1986)], large-step Markov chains [Martin et al. (1991)], iterated Lin-Kernighan [Johnson D. S. (1990)], chained local optimization [Martin Otto (1996)], or combinations of these [Applegate et al. (1999)]. ILS has many of the desirable features of a meta-heuristic: it is simple, easy to implement, robust and highly effective. The essence of the iterated local search meta-heuristic 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. 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 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.

2.1 Experimental Designs:

A computer experiment is modeled as a realization of a stochastic process, often in the presence of nonlinearity and high dimensional inputs [Sacks et al. (1989a)]. In order to perform efficient data analysis and prediction and in order to determine the best settings for a number of design parameters that have an impact on the response variable(s) of interest and which influence the critical quality characteristics of the product or process, it is often necessary to set a good design as well as to optimize the product or process design. In computer experiments, instead of physically doing an experiment on the product, mathematical models describing the performance of the product are developed using engineering/physics laws. Then the mathematical models are solved on computers through numerical methods such as the

finite element method. A computer simulation of the mathematical models is usually time consuming and there is a great variety of possible input combinations. For these reasons, meta-models [Barthelemy and Haftka (1993); Sobieski and Haftka (1997)] that model the quality characteristics as explicit functions of the design parameters are constructed. Such a meta-model, also called a (global) approximation model or surrogate model, is obtained by simulating a number of design points. Since a meta-model evaluation is much faster than a simulation run, in practice such a meta-model is used, instead of the simulation model, to gain insight into the characteristics of the product or process and to optimize it. Fang [Fang et al. (2000a); Fang et al. (2000b)] defined a uniform design as a design that allocates experimental points uniformly scattered on the domain. Uniform designs do not require orthogonal. They consider projection uniformity over all sub dimensions. In [Fang et al. (2000b)] they classify uniform designs as space-filling designs.
Lee and Jung (2000) proposed maximin Eigen value sampling, that maximizes minimum Eigen value, for Kriging model where maximin Eigen value sampling uses Eigen values of the correlation matrix. The Kriging model is obtained from sampled points generated by the proposed method. Note that the Kriging model [Krige (1951)] is used to compare the characteristics of proposed sampling design with those of maximum entropy sampling.
The maximin design problem has also been studied in location theory. In this area of research, the problem is usually referred to as the max-min facility dispersion problem [Erkut (1990)]; facilities are placed such that the minimal distance to any other facility is maximal. Again, the resulting solution is certainly space-filling, but not necessarily noncollapsing.
In statistical environments Latin Hypercube sampling is often used. In such an approach, points on the grid are sampled without replacement, thereby deriving a random permutation for each dimension ([McKay et al. (1979)]).
Giunta [Giunta et al. (2003)] gives an overview of pseudo- and quasi-Monte Carlo sampling, Latin hypercube sampling, orthogonal array sampling, and Hammersley sequence sampling

3. Maximin Latin Hypercube Designs:

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

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

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.

3.1 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

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

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.

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

As we know 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,……, N do
        for j = i + 1,….., N do
          Step 1= If  {i,j} ⋂ S_Cpt(X) ≠ ϕ
          then:  let D1 = D1^(S), k/ = 0
          for l = 1,…., k do
              Step 2: Swap (X_il, X_jl)
              Step 3: Compute d_(i/,u)^(X+1)  until  d_(i/,u)^(X+1) ≥ D1/
              ∀ i/ = i .j: u = 0……N-1; u ≠ i/
               else break
              Step 4 :Set  k/= k and  D1/ = min d_(I J)^(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/ = ∈(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.

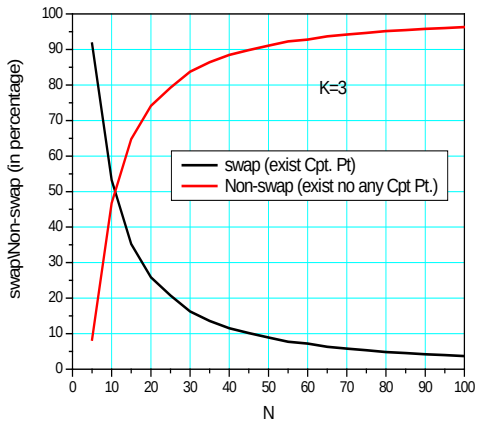
Before analysis the time complexity of the ILS (D_1, J_1) algorithm, It is worthwhile to present it in a Pseudo code. The Pseudo code of the ILS (D_1, J_1) algorithm is displayed in the Table (4.1).

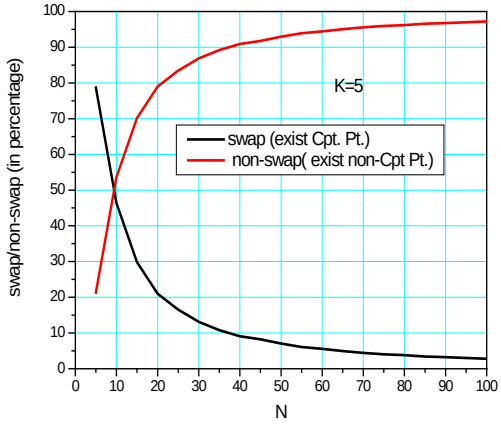
As the aim of this section is to experimentally assess the computational cost of the proposed ILS(D_1) algorithm, we will first derive the number of operations required by a single local search, and then those for a single run of ILS (from now on in this section we will give as understood that we are discussing the ILS(D_1,J_1) version). For these experiments we consider k =

3, 5, 7, 10 and $N = 10i$: $i = 1, 2, \ldots, 10$. 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.





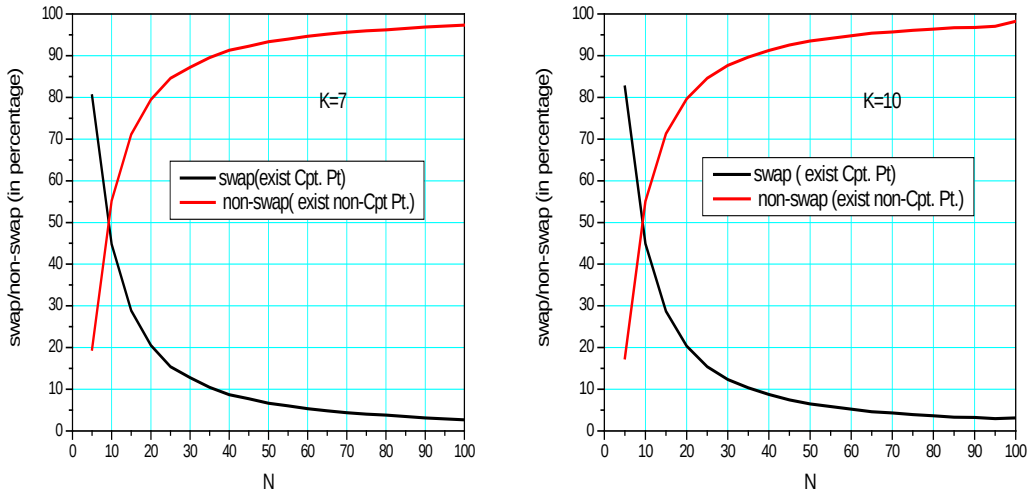


Figure 4.1: The percentage of pairs involving and not involving critical points

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.

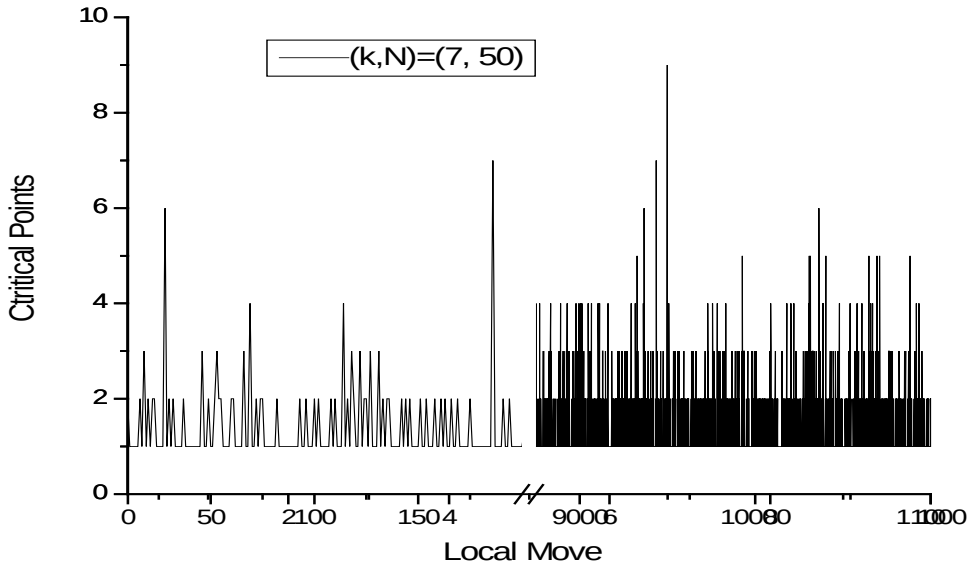


Figure 4.2: The history of number of critical points for (k,N) = (7, 50) during local Move

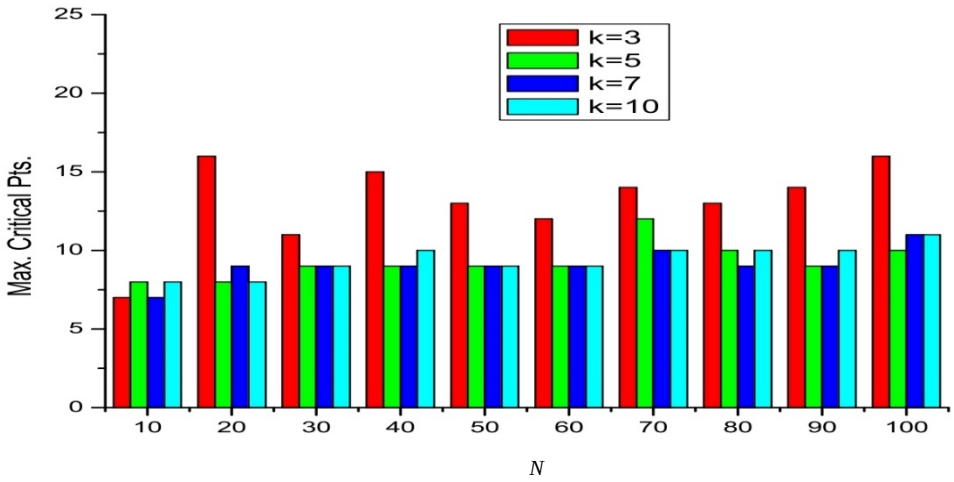


Figure 4.3: The impact of N on Maximum Critical Points during history of evaluation

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).

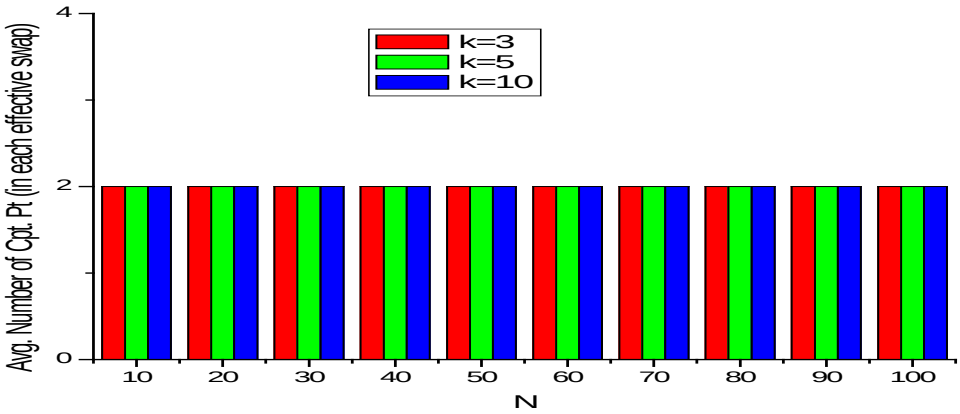


Figure 4.4: The impact of N on average Critical Points during history of evaluation

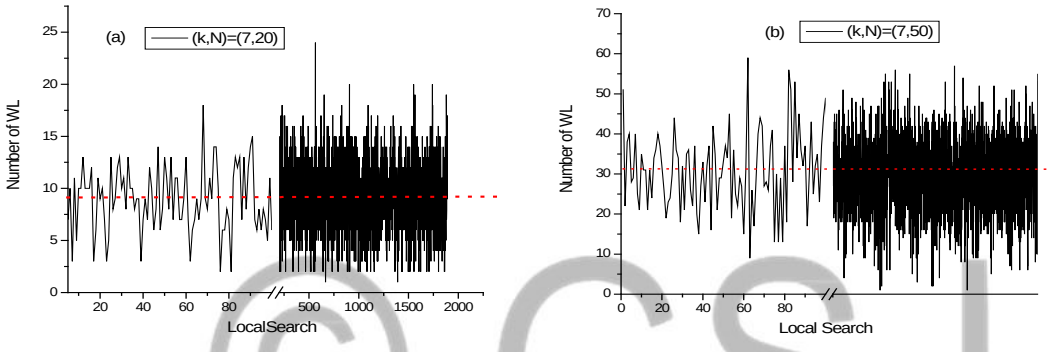


Figure 4.5: The history of WL for (k, N) = (7, 20) and (k, N) = (7, 50) during Local Search

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).

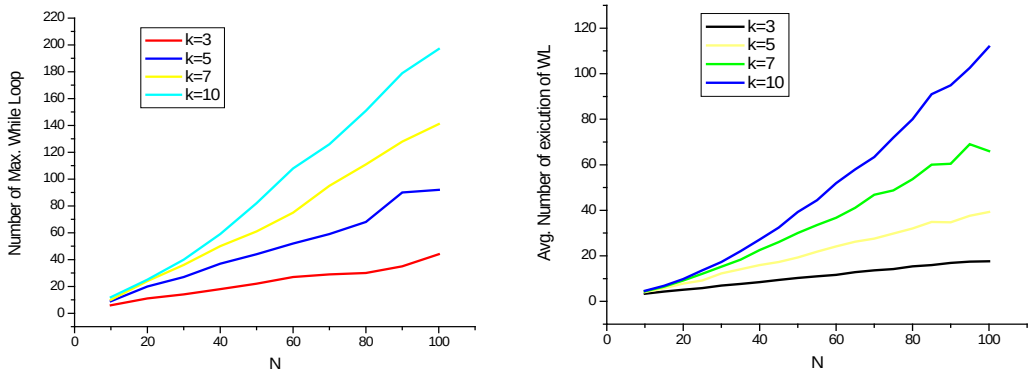


Figure 4.6: The impact of N on (a) Maximum WL (b) Average WL during history of Local Search

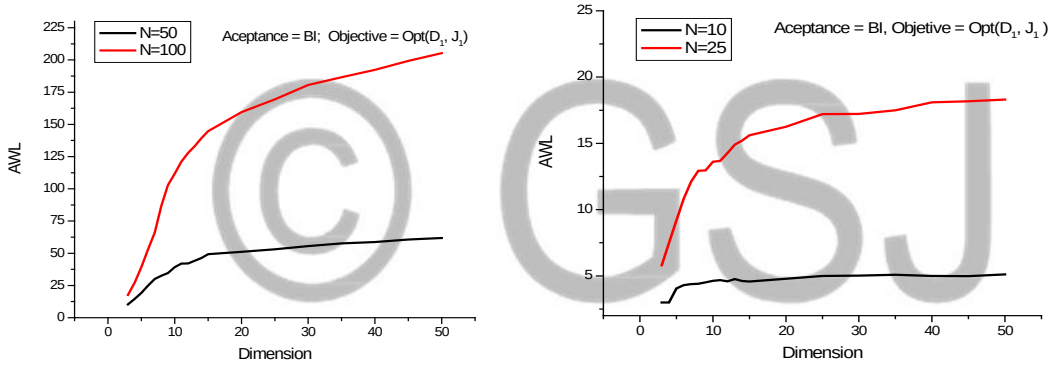


Figure 4.7: The Impact of k on AWL during Local Search

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).

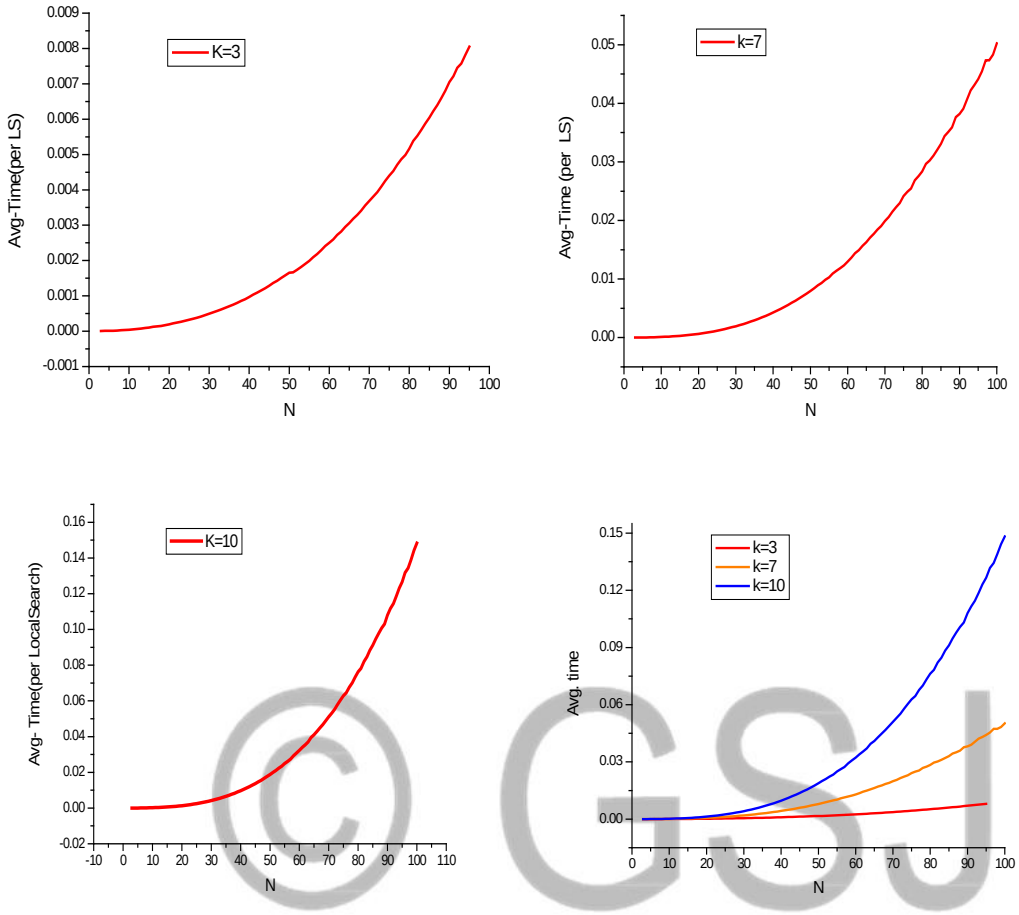


Figure 4.8: The History of Elapsed time

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 $2.5 < q \leq 3$. 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 $1 < r \leq 2$, which is again in accordance with what was previously conjectured.

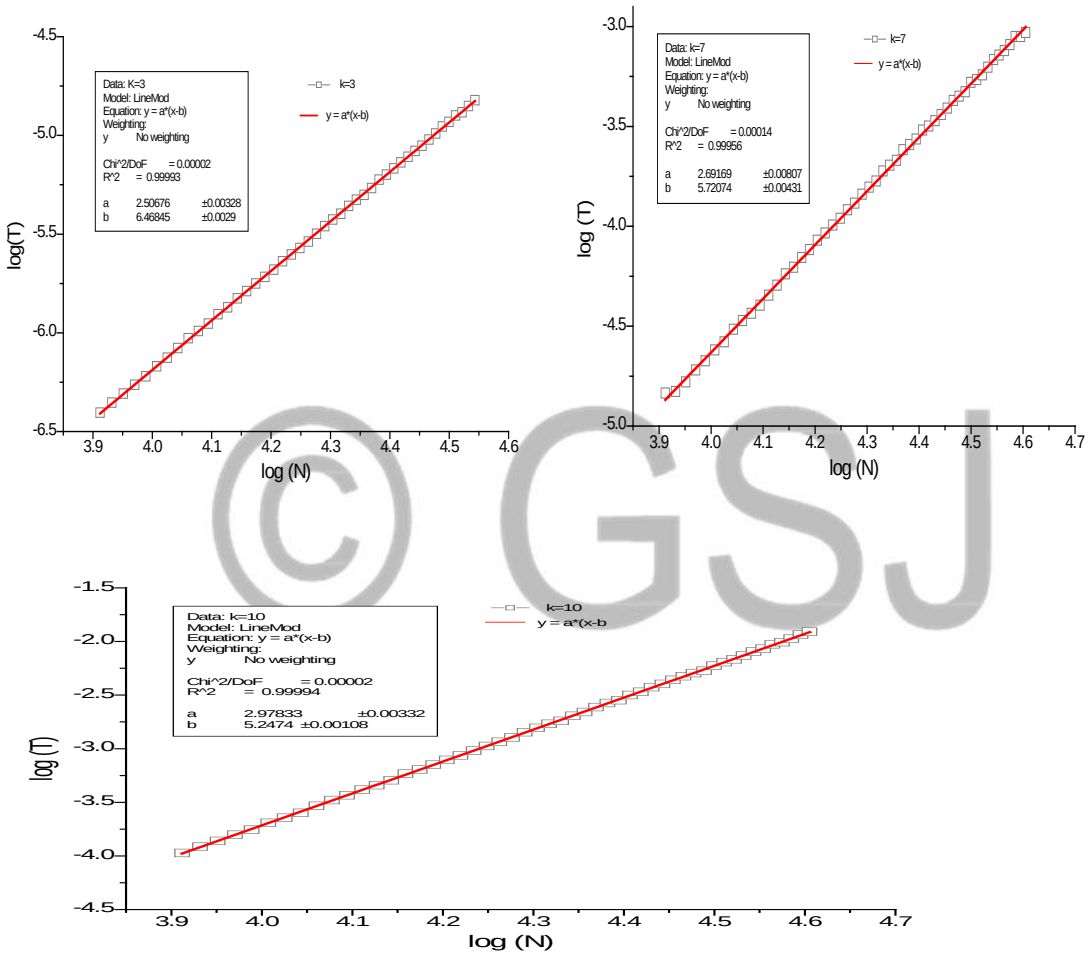


Figure 4.9: The values of log(T) plotted against log(N)

We remind the reader that this practical time complexity $\approx O(k^r N^q)$, 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 perturbation

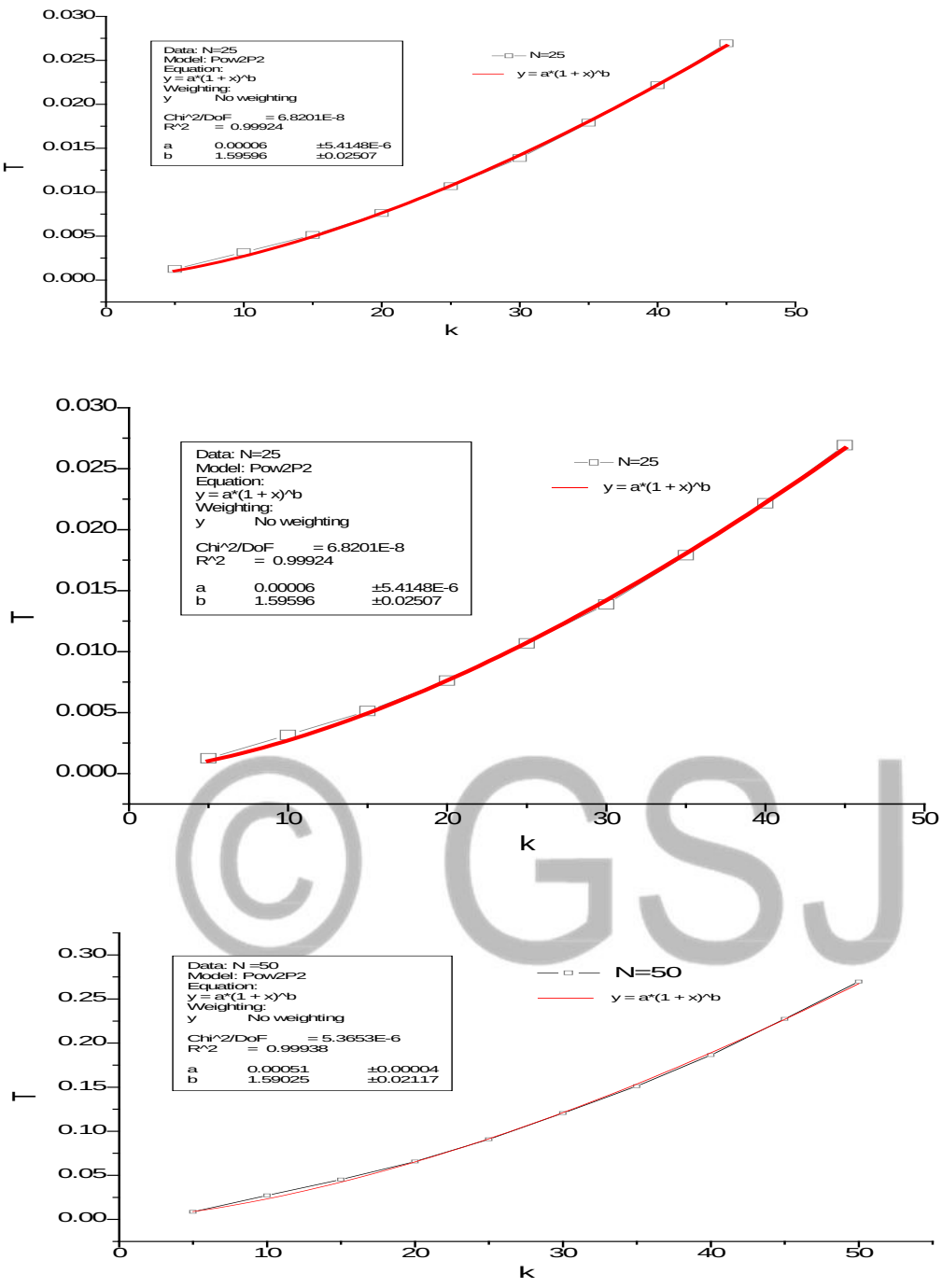


Figure 4.10: The approximate time complexity for LS with respect to *k*

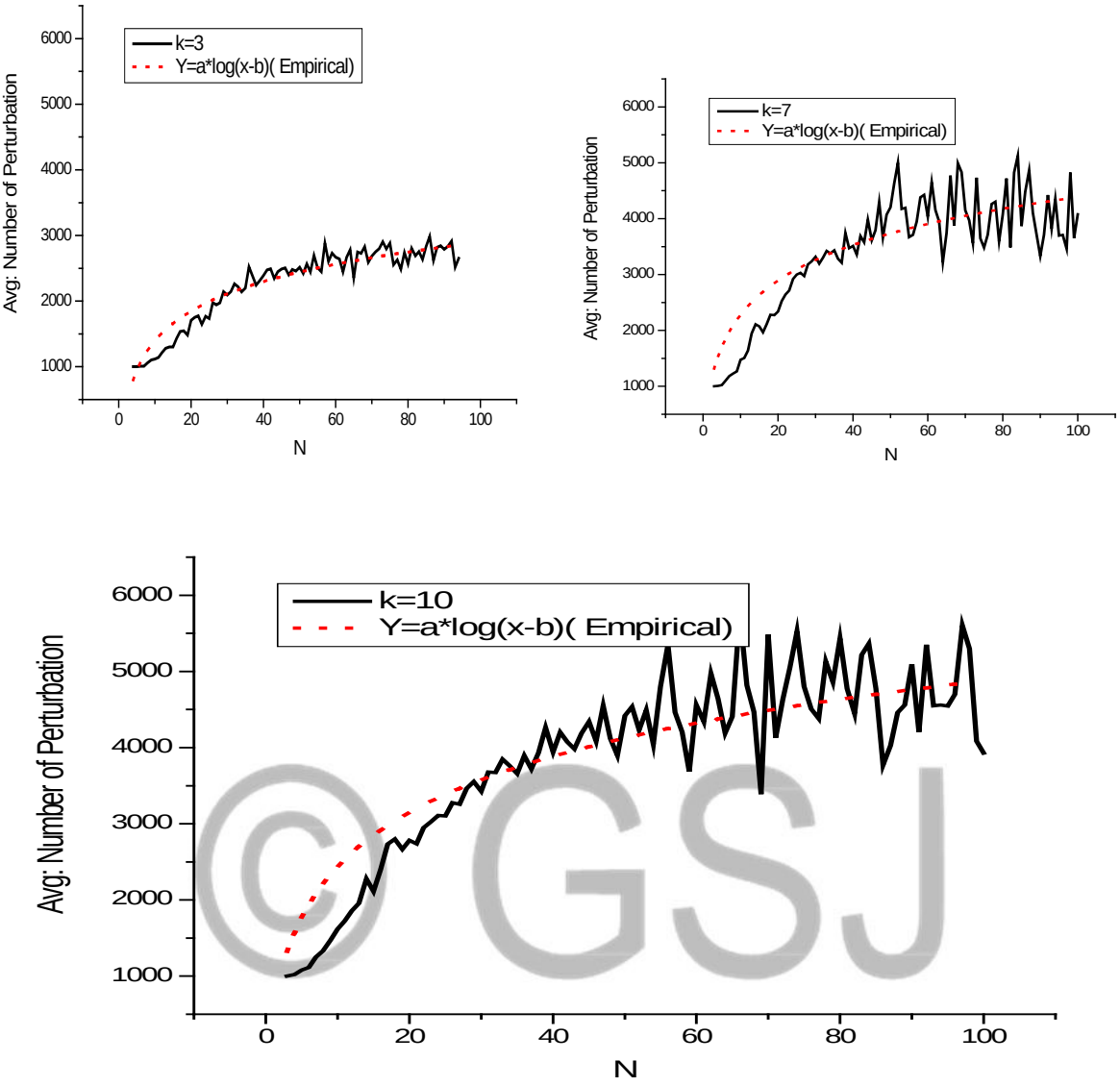


Figure 4.11: The impact of N on the number of perturbations

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).

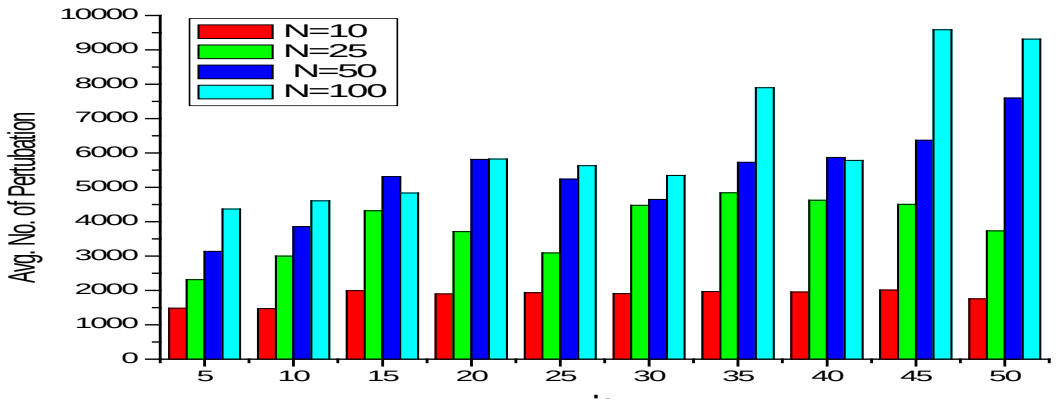


Figure 4.12: The impact of k on the number of perturbations

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……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 : $X^{/} = \in(X)$ and Repeat the loops
	Otherwise STOP
	Return X

5. 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,……, N do
         for j = i + 1,….., N do
            Step 1=  let D1 = D1(S), k/ = 0
             for l = 1,…., k do
                 Step 2: Swap (Xil, Xjl)
                  Step 3: Compute d(X+1) i/,u  until  d(X+1) i/,u ≥ D1/
                  ∀ i/ = i .j: u = 0……N-1; u ≠ i/
                  else break
                  Step 4 :Set  k/ = k and  D1/ = min  d(X+1) I J
              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/ = ∈(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).

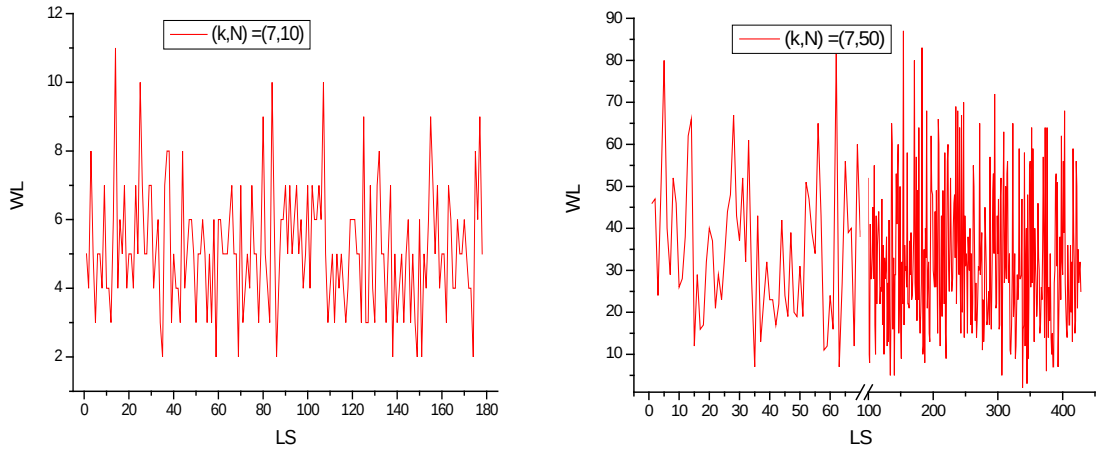


Figure 5.1: The history of WL values for (k,N) = (7, 10) and (k,N) = (7, 50) during Local Search

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 $LM_{Rp\varphi}$ 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_1) 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……,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.

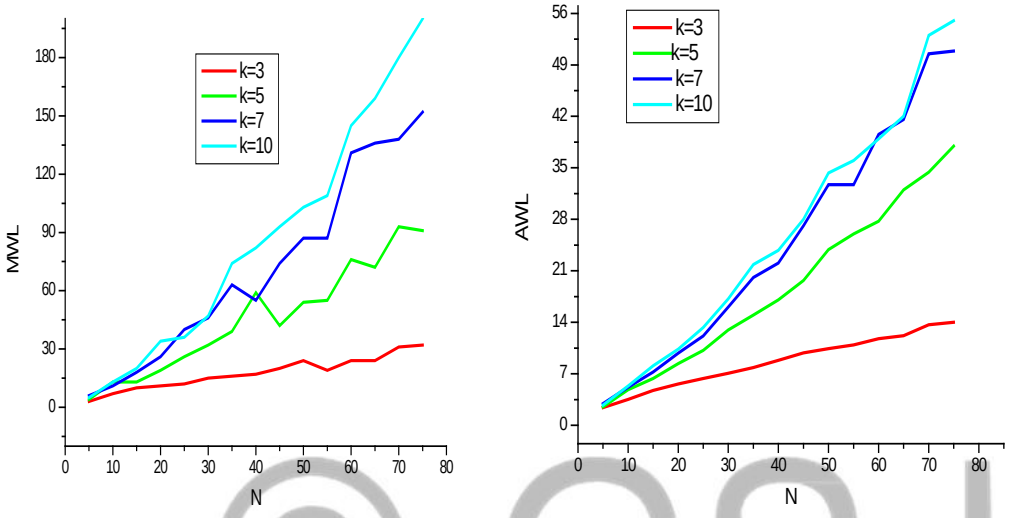
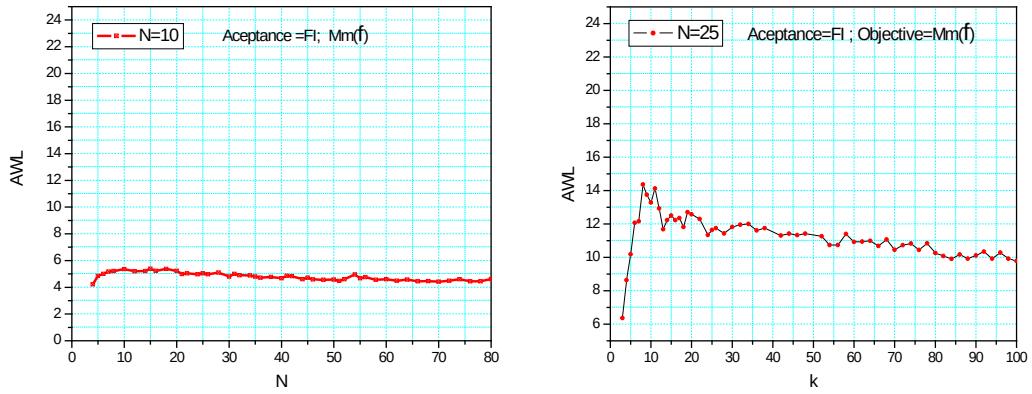


Figure 5.2: The impact of N on (a) MWL (b) AWL



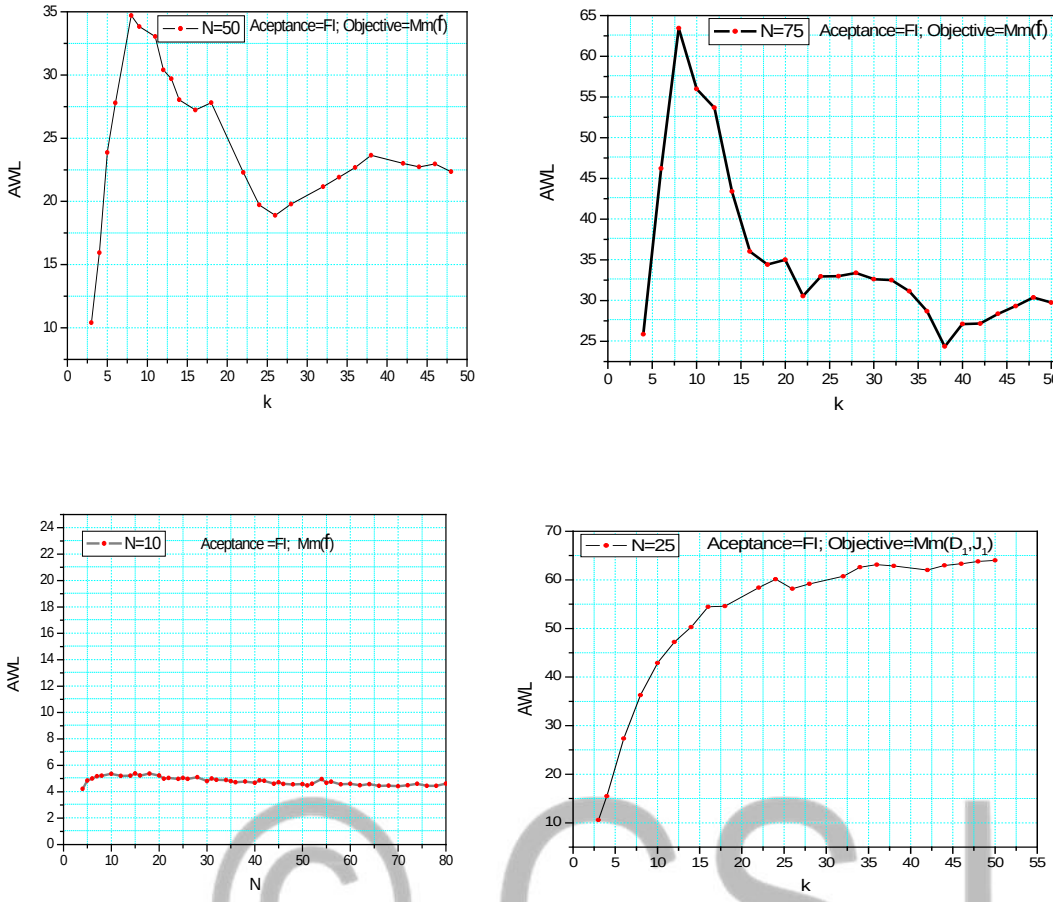


Figure 5.3: The Impact of *k* on AWL

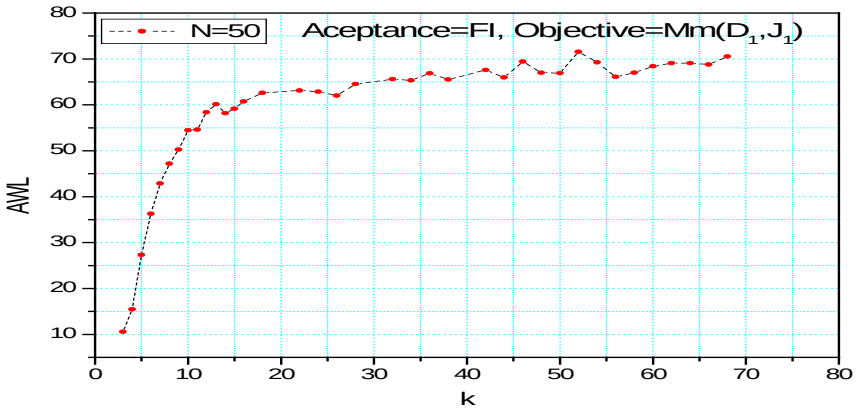


Figure 5.4: The Impact of *k* on execution of AWL during Local Search with FI(First Improve) in Opt(D_1, J_1)

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: $k = 2i : i = 1, 2, \ldots\ldots, 40$ 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 $N = 20, 21, \ldots\ldots.. 80$ 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.

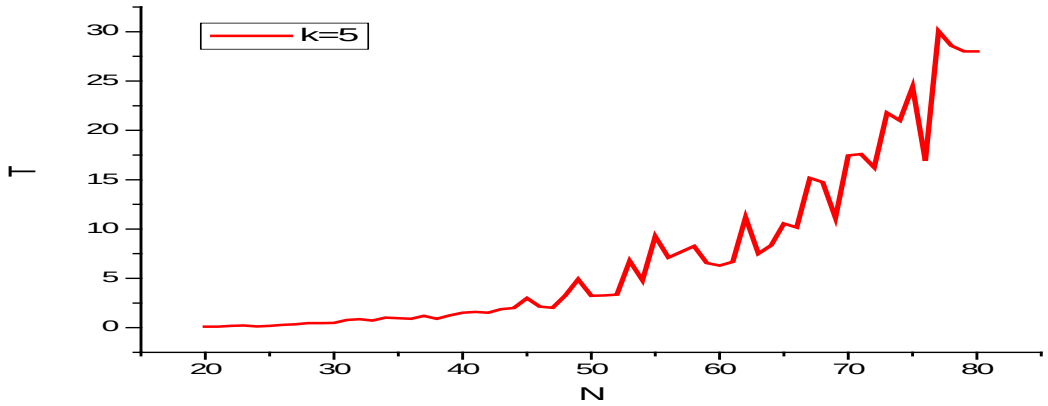


Figure 5.5: Elapsed time per local search as a function of

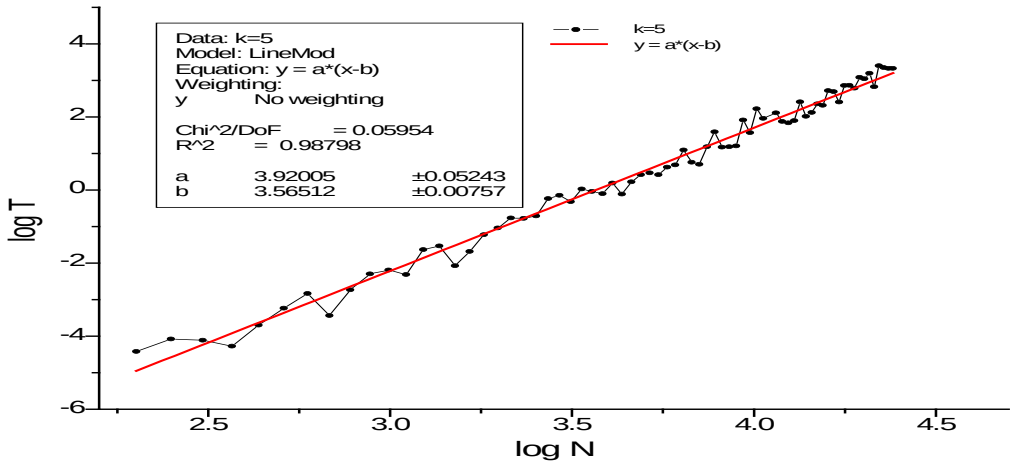


Figure 5.6: Linear regression between log(T) and log(N)

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 .

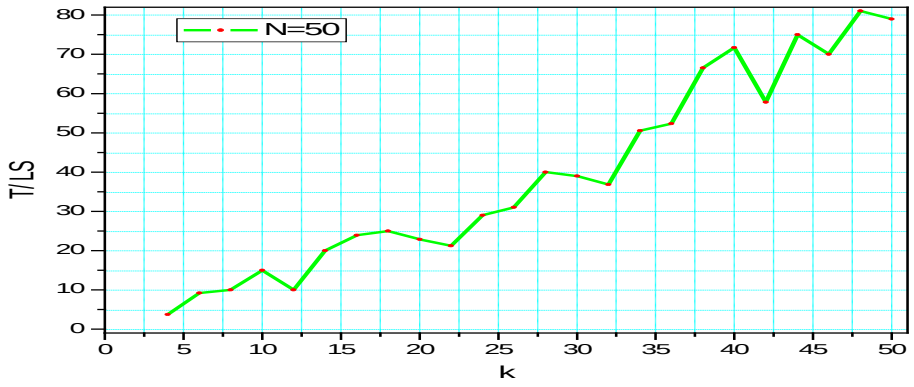


Figure 5.7: Impact of k on T

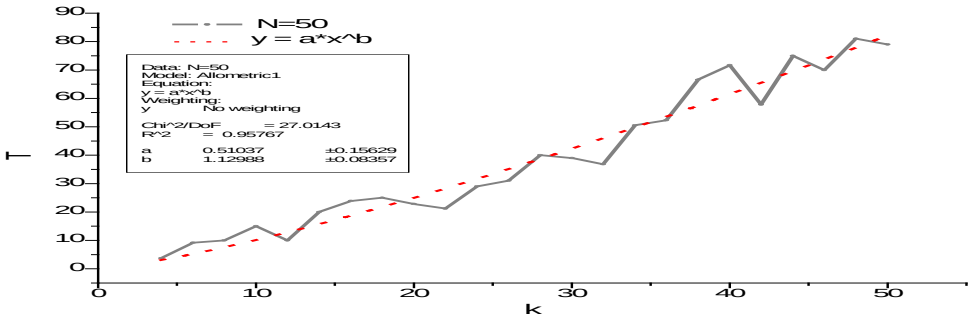


Figure 5.8: The approximate time complexity of k for LS obtained by the experiments

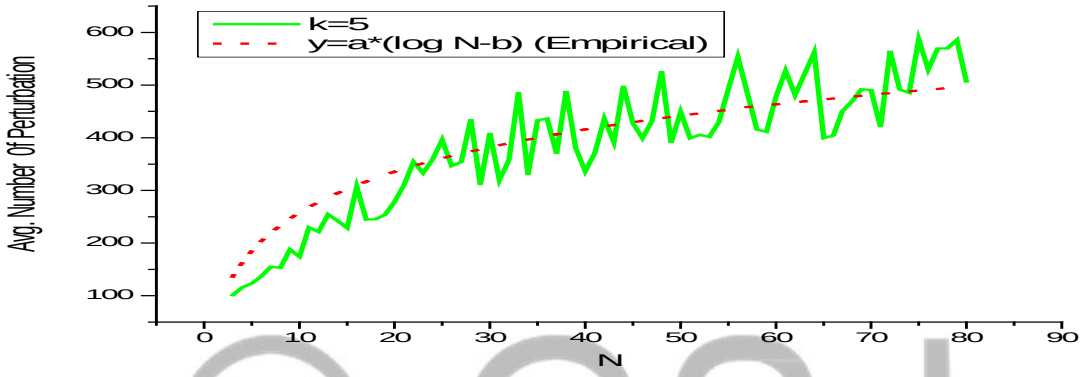


Figure 5.9: Relation between the number of perturbations and N

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.

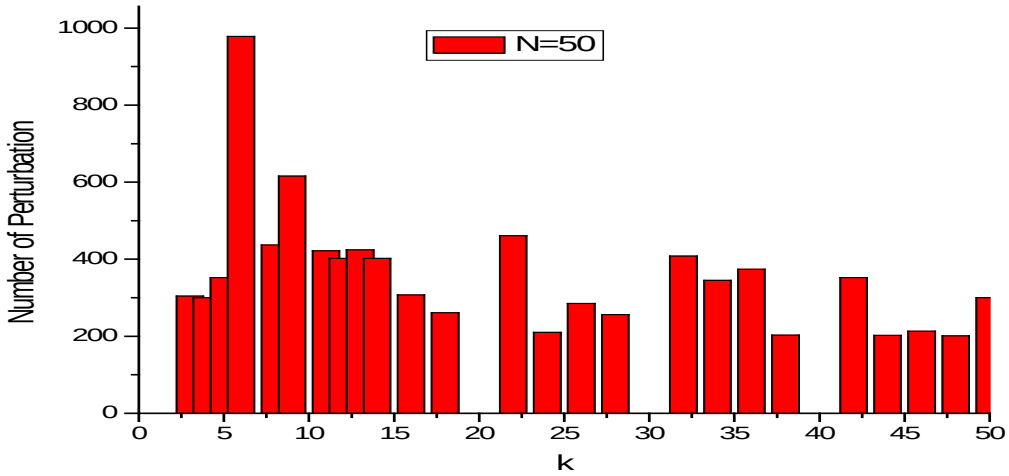


Figure 5.10: Impact of *k* on the number of perturbations

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 $d_{i',u}^{(s+1)} \le D_1'$)
 	
(*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 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
 	
$O(1).O(N).O(k).O(N).O(Nk^c) \approx O(k^r N^q)$
$:(1 < r < 2)$ & $(2 < q \le 4)$
 	
Cost of a single ILS (Φ)
 	
Perturbation (for fixed MIN))
O (log(N))
 	
Total Cost:
$O(k^r N^q).O(\log(N)) \approx O(k^r N^q \log(N))$
$:(1 < r < 2)$ & $(2 < q \le 4)$
	While do
	Set NonImpIteration = 0
	Whiledo
	for $i = 1,\ldots\ldots, N$ do
	for $j = i + 1,\ldots.., N$ 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 $d_{i',u}^{(X+1)} \ge D_1'$
	$\forall i' = i .j:$
	u = 0……N-1;
	$u \ne i'$
	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

REFERENCES

01. 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.
02. Baum, E. B., 1986(b), "Iterated descent: A better algorithm for local search in combinatorial optimization problems", Technical report, Caltech, Pasadena, CA Manuscript.
03. 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.
04. Baxter, J., 1981, "Local optima avoidance in depot location", Journal of the Operational Research Society, Vol. 32, pp. 815–819.
05. Fang, K. T., D. K. J. Lin, P. Winkler, and Y. Zhang (2000b), "Uniform design: theory and application", Technometrics, Vol. 42, pp. 237–248.
06. Fang K. T., R. Li, and A. Sudjianto, 2006, "Design and Modeling for Computer Experiments", CRC Press, New York.
07. 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.
08. 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.
09. Kirkpatrick S., C. D. Gelatt Jr., and M. P. Vecchi, 1983, "Optimization by Simulated Annealing , Science, Vol. 220, pp. 671-680.
10. 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.
11. Martin O. and S. W. Otto., 1996, "Combining simulated annealing with local search heuristics", Annals of Operations Research, Vol. 63, pp. 57–75.
12. 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.
13. 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.

14. 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.
15. www.spacefillingdesigns.nl