Solving Correlation Matrix Completion Problems using Parallel Differential Evolution

by

Srujan Kumar Enaganti

B. Tech, Indian Institute of Technology Guwahati, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

November 2010

Abstract

Matrix Completion problems have been receiving increased attention due to their varied applicability in different domains. Correlation matrices arise often in studying multiple streams of time series data like technical analyses of stock market data. Often some of the values in the matrix are unknown and some reasonable replacements have to be found at the earliest opportunity to avert an unwanted consequence or keep up the pace in the business. After looking to background research related to solving this problem, we propose a new parallel technique that can solve general correlation matrix completion problems over a set of computers connected to a high speed network. We present some of our results where we could reduce the execution time.

Table of Contents

List of Tables

List of Figures

Acknowledgments

This thesis would have never taken this shape without the exemplary supervision of Dr. Alan Wagner throughout my thesis period. He made himself available whenever needed and was extremely helpful with his suggestions steering me in the right direction and helping me progress and complete this thesis. I cannot express my gratitude to Dr. Ed Knorr in any simple terms for all the useful informal discussions giving me insights and finally for his scrupulous reviews as a second reader which helped me refine the thesis. Special thanks to Cody Brown for his awesome technical support in porting the programs on to the Cyclops cluster and troubleshooting any problems that have arisen. Last but not the least, I would like to thank all the people in NSS lab and my fellow grad friends for their cordiality and moral support throughout my stay.

Chapter 1

Introduction

If I have seen farther it is by standing on the shoulders of Giants.
— Sir Isaac Newton

1.1 Motivation

The Matrix Completion problem is the problem of finding unknown entries in a matrix by using some known properties of the expected completed matrix [1]. Real world applications involve the collection of data from different sources and processing it to yield valuable information. Often some of the data may be unreliable or unavailable for a host of reasons. It becomes imperative that such data be reclaimed. Sometimes such missing data is needed at the earliest convenience to either avert an unwanted consequence or to keep up with the competition among the ones who need similar data. Although it is impossible to extract the exact data, it is possible to get the lost data which may follow certain properties that can help to predict the missing values.

Numerical data from the real world is typically represented in a rectangular arrangement such as a matrix. Recovering data entries would amount to recovering certain unknown entries in the matrix. The Matrix Completion problem arises precisely when there is a need to recover the data entries in such an incomplete

matrix. The solution to the problem comes from the fact that certain known constraints will be valid in a completed matrix. For example, correlation matrices are formed by noting correlations of each pair of different streams of data. Such matrices are known to have certain properties including being positive semidefinite. When some of the values are unknown, we exploit the known properties to predict the unknown entries.

The Matrix Completion problem is receiving increased attention for its applications in a variety of areas including:

1. Finding Missing Correlation Values of Streaming Time Series Data
2. Collaborative Filtering in Recommender Systems
3. Global Positioning Systems
4. Remote Sensing

1.1.1 Finding Missing Correlation Values of Time Series Data

A useful function is to predict unknown correlation values among real time streaming data like in stock market data where some values may be unreliable or unknown. Numerous applications that perform technical analysis to compute such data would need reasonable data to work upon when the real data is either unavailable or unknown. Consider the correlation matrix that is formed by taking pairwise correlations of all possible pairs of observed stock data. There is a need to estimate the values that are missing in such a way that maximum risk is minimized.

1.1.2 Collaborative Filtering

Recommender systems are applications that can help someone to make an appropriate choice among a set of alternatives. Collaborative filtering is the most important functional aspect of the same [2].

Typically, major companies host surveys wherein their customers are asked to rate certain objects based on various criteria. These ratings are used by the companies to understand consumer behavior. But it may happen that most of the users may not rate all the objects for all the criteria. In this case, it becomes vital to extract the possible ratings of those objects that are not rated by the users to serve the business end of the company. This is done based on a ratified hypothesis that there are only few criteria/factors that drive a user to make her ratings and choices of objects. This problem can be modeled as a Matrix Completion problem by using Matrix Factorization technique where, corresponding to users and items, factors are given which are inferred from item rating patterns [3] [4]. A higher correspondence between a user and an item will lead to the recommendation of the item for the user. Since not all entries are available at the beginning, it is a matrix completion problem.

Netflix is a subscription service that streams movies and TV episodes over the Internet and sends DVDs by mail. They conduct online surveys from their users of TV series/movies on several criteria, but typical users may not fill in all the survey entries completely. In order to capture users' interests in all fields of all possible movies/serials, they have built an in-house recommender system to predict the unknown entries. In 2007 they conducted a competition called the *NetFlix Challenge* to find a better algorithm than they have that can predict users' unknown ratings by using their known historical data. A minimum performance improvement of 10% over the predictive ability of their existing algorithms was required. Out of a large number of submissions, an algorithm proposed by Yehuda Koren from AT&T labs called the BellKor solution [5] won the first prize in the competition. The other optimal solutions proposed were [6] and [7].

1.1.3 Global Positioning Systems

The global positioning system works by receiving satellite signals from different points to compute an object's location on earth. Often the values can be corrupted due to noise in transmission or reception and this can lead to errors in the calculation of positions of various nodes that the system is supposed to work upon. An

extensive research article addressed this problem [8].

1.2 Terms and Definitions

Before stating the exact problem that I am solving, I give the definitions that will be useful in understanding the context of the same.

Definition Matrix

A matrix is a rectangular arrangement of numbers.

Definition Square Matrix

A square matrix is a matrix which has the same number of rows and columns. An *n-by-n matrix is known as a square matrix of order n.*

Definition Positive Semidefiniteness

A square $n \times n$ matrix A is said to be *positive semidefinite (PSD)*
iff $\forall x \in \Re^n, x^T Ax \geq 0$. Every PSD matrix will have all its eigenvalues as non-negative.

Definition Positive Definiteness

A square $n \times n$ matrix A is said to be *positive definite (PD)*
iff $\forall x \in \Re^n, x^T Ax > 0$. Every PD matrix will have all its eigenvalues as positive.

Definition Real Partial Matrix

A real partial matrix A is one in which some entries are specified as real numbers and the remainder are unspecified, i.e., free variables over the set of real numbers.

Definition Partial Positive Definite Matrix

A partial positive definite matrix is a partial symmetric matrix each of whose specified principal submatrices is positive definite.

Definition Pattern of a Square Matrix

A pattern for an $n \times n$ matrix is a list of positions of $\{1,2,...,n\} \times \{1,2,...,n\}$. A partial matrix specifies the pattern if its specified entries are exactly those listed in the pattern. A pattern Q is called symmetric if $(i, j) \in Q$ implies $(j,i) \in Q$.

1.2.1 Graph of a Partial Matrix

A *partial symmetric matrix* can be represented by an undirected graph as given below.

Let A be a *partial positive definite* $n \times n$ *matrix*. The undirected graph $G = (N, E)$ of A has node set $N = \{1, 2, ..., n\}$ and an edge $\{i, j\} \in E, i \neq j$, iff entry a_{ij} is specified.

For example, suppose we have the following *partial symmetric matrix* as our A,

$$\begin{pmatrix} 5 & 2 & ? & 2 \\ 2 & 4 & 1 & ? \\ ? & 1 & 3 & ? \\ 2 & ? & ? & 6 \end{pmatrix}$$

The resultant graph G of the above matrix A is shown in **figure 1.1**.

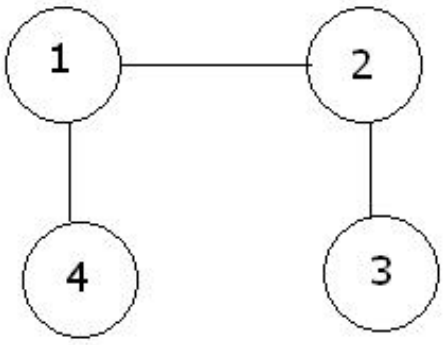

Figure 1.1: Graph of the partial matrix A

1.3 The Problem

The various real world situations given in the motivation section are examples where the matrix completion problem has been found to be useful. A type of the

matrix completion problem is the correlation matrix completion problem. In this thesis, we discuss approaches to solve correlation matrix completion problems that are found particularly in the area of finance where certain streaming values may be unknown or known to be spurious. A formal definition for the correlation matrix completion problem can be given as follows:

Definition Complete Correlation Matrix Problem (CCOMAT)

The problem is to find missing entries in a partial symmetric matrix where all the given entries and the unknown entries fall within -1 and 1 such that the resultant complete matrix is positive semi-definite. All correlation matrices have all 1's on their diagonal and the property that they are positive semi-definite. The problem here will be referred to as CCOMAT in this thesis.

1.4 Our Approach

The problem is being solved by parallelizing an optimization method called Differential Evolution. It is an evolutionary strategy which can be used for solving optimization problems where the cost function may be non-differentiable or non-continuous in the domain over which it is optimized. It works by having a randomly chosen sample of initial potential candidates and then evolving them incrementally through iterations and improving them at every step. This simple parallel method has been improved to a version which can dynamically adapt itself by employing the concept of simultaneous independent evolution on separate islands and the migration of element(s) between them.

The thesis is organized as follows. Chapter 1 has focussed on the problem that is being solved and the motivation behind choosing to solve it that way. In Chapter 2 we look at a background survey of various approaches that have been taken to solve either the CCOMAT problem or similar problems that can lead to solving the same. Chapter 3 gives insights to the algorithmic approach proposed in this thesis with the backdrop of the sequential algorithm that has been used as a base. Chapter 4 gives details regarding the different tools used and the experimental results of executing the algorithms in parallel on a cluster of computers. Chapter 5 gives the

general conclusions of the thesis, as well as future works that could provide a more robust approach towards solving the CCOMAT problem more efficiently.

Chapter 2

Background

A good short-story writer has an instinct for sketching in just enough background to ground the specific story. — Lynn Abbey

2.1 Introduction

In order to solve the correlation matrix completion problem in parallel, it was pertinent to look at the progress that has been made in solving the same problem in a serial way. Since the correlation matrix completion problem is a special case of the PSD completion problem with an additional constraint that all unknown entries are correlation values (and hence between -1 and 1), some results in the latter are directly applicable.

2.1.1 PSD Completion to Chordality

Definition Chordality of a graph

A graph is said to be chordal if it does not have any simple cycle of length more than 3 nodes.

A vital result in relating PSD completion to the structure of a graph corresponding to an incomplete graph was obtained by Grone et al. [9]:

Theorem 2.1.1 *Every partial positive definite (semidefinite) matrix with graph G has a positive definite (semidefinite) completion if and only if G is chordal.*

Proof This was originally proved by Grone et al. [9] by using complex analytic techniques. It has also been recently proved by Smith [10] using only matrix/graph theoretic tools. ∎

2.2 Correlation Matrix Completion for Special Structures

Some theoretical structures were especially treated in the literature probably because of their simplistic nature and occasional occurrence of such structures in solving practical problems.

2.2.1 The Simple Cycle Graph

When the graph of a partial matrix is a simple cycle, the correlation matrix completion problem can be deterministically solved using the following result [11]. Since the known entries of a correlation matrix are within [-1,1], it is possible to represent them as cosine values of a particular value.

Theorem 2.2.1 *Suppose* $n \geq 4$, *let* $N = \{1,2,...,n\}$, *and* $0 \leq \theta_1, \theta_2, ..., \theta_n \leq \pi$. *Then the matrix*

$$C = \begin{bmatrix} 1 & \cos\theta_1 & & & & \cos\theta_n \\ \cos\theta_1 & 1 & \cos\theta_2 & & ? & \\ & \cos\theta_2 & 1 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & ? & & \ddots & \ddots & \cos\theta_{n-1} \\ \cos\theta_n & & & & \cos\theta_{n-1} & 1 \end{bmatrix}$$

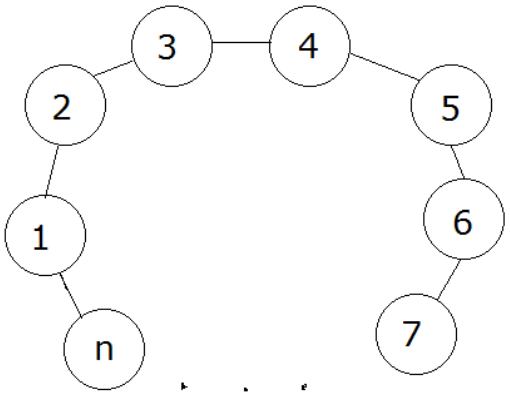

Figure 2.1: A Ring Graph

has a positive semidefinite completion if and only if for each $S \subseteq N$ with $|S|$ odd,

$$\sum_{i \in S} \theta_i \leq (|S|-1)\pi + \sum_{i \in S^c} \theta_i. \tag{2.1}$$

Proof The theorem has been proved in [11] ∎

Definition The C_n^k pattern

Let $n/2 > k \geq 1$, then the symmetric pattern

$$Q = \{(1,k+1),(2,k+2),...,(n-k,n),(n-k+1,1),...,(n,k)\} \tag{2.2}$$

is called a C_n^k pattern.

Definition Symmetric Toeplitz Matrix

A symmetric $n \times n$ matrix $A = (a_{ij})$ is called a symmetric Toeplitz matrix if $a_{i,j} = r_{|i-j|}$ for all $i, j = 1,2,...,n$. A partial Toeplitz matrix is a partial symmetric matrix and if an entry in position (i, j) is specified then all entries in positions $(i+l, j+l)$ $(\text{mod } n)$ are also specified and these entries are equal.

Cycle graphs which are generated by a C_n^k pattern are a generalization of ring graphs. We can see that the C_n^k pattern with $k = 1$ is a ring (See Figure 2.1).

2.2.2 PSD Completion of C_n^k Pattern Matrix

When does a partial matrix specified as a C_n^k pattern have a correlation completion?

$$C = \begin{bmatrix} 1 & & \cos\theta_1 & & \cos\theta_{n-k+1} & & ? \\ & 1 & & \cos\theta_2 & ? & \ddots & \\ \cos\theta_1 & & 1 & & \ddots & & \cos\theta_n \\ & \cos\theta_2 & ? & 1 & & \ddots & \\ \cos\theta_{n-k+1} & ? & \ddots & & \ddots & & \cos\theta_{n-k} \\ & \ddots & & \ddots & & \ddots & \\ ? & & \cos\theta_n & & \cos\theta_{n-k} & & 1 \end{bmatrix} \tag{2.3}$$

Theorem 2.2.2 *Let $n \geq 4, 1 \leq k < n/2, d = \gcd(n,k)$ and $t = n/d$. Then we have:*

1. *If $t = 3$, then the partial PSD matrix (2.3) has a PSD completion.*
2. *If $t \geq 4$, then the partial PSD matrix (2.3) has a PSD completion if and only if for each $S_i \subset N_i \equiv \{i, \overline{i+k}, \overline{i+2k}, ..., \overline{i+(t-1)k}\}$ with $|S_i|$ odd,*

$$\sum_{j \in S_i} \theta_j \leq (|S_i| - 1)\pi + \sum_{j \in N_i S_i} \theta_j, \tag{2.4}$$

$i = 1, 2, ..., d$.

Proof The proof can be seen in [12] ∎

The theoretical treatments of the above structures discussed so far have all been very elegant and directly applicable to solving CCOMAT in a deterministic way. However, the fact that they are applicable to only the structures specified restricts their applicability to those that can arise in practical scenarios. All these approaches have made inroads towards coming up with theoretically elegant solutions to special cases of CCOMAT. The general solution which can be applied to

all possible cases that can arise in a CCOMAT problem is apparently not treated in a theoretical deterministic way so far.

2.3 More Generic Correlation Matrix Completion Approaches

In this section, we present approaches that have been developed to solve the problem of CCOMAT in a general way, i.e., there are fewer assumptions on the structure of unknown entries. Some approaches to solve similar problems to CCOMAT will be examined here to see if they would provide insight to solving the CCOMAT problem.

2.3.1 Generic PSD Completion & Determinant Maximization

An iterative method was proposed by Glunt et al. for solving the positive semidefinite completion of a general matrix [13]. The authors proved that their iterative algorithm does converge after a finite number of iterations and gives a unique matrix that is positive definite which has the maximum possible value for the determinant. Even though it is generically applicable, the problem of the correlation matrix completion is not contained within that because the entries filling in the incomplete entries would not necessarily stay within the absolute value of 1. Hence, we concluded that this algorithm cannot be directly applied to solving the correlation completion problem.

2.3.2 Nearest Correlation Matrix

A similar problem to CCOMAT is to find the nearest correlation matrix, and it has been studied by Nicholas Higham [14]. The solution proposed is through semidefinite programming where a variant of Newton's method is used. The completion of the correlation matrix may use concepts from the nearest correlation matrix solution but are not directly applicable as some of the entries have to remain fixed in the former.

2.3.3 Correlation Matrix Completion using Gaussian Elimination

Kahl and Gunther presented an algorithm for correlation matrix completion when the matrix is a multi-dimensional stochastic volatility model [15]. They explain the specific structure of the incomplete correlation matrices that they are working upon. They make use of properties of the resultant graph representation of the incomplete matrix and use Gaussian elimination to arrive at a completion having the maximum possible determinant. The correlation completion solution, although very efficient, is only restricted to a certain type of incomplete correlation matrix thereby limiting its application in diverse real world situations.

2.3.4 Completion of a Correlation Matrix of Limited Size

Budden et al. introduced a deterministic solution to complete correlation matrices up to order 4 [16]. The algorithm for a correlation matrix of 4 variables assumes that correlations of one of the variables with three others are given as known entries and then deterministically predicts the possible ranges of each of the missing correlation values one after the other. That is, it can give the range of the first unknown correlation value and when that is fixed, it can give the range for the next one, and so on to finally complete the matrix. They have made an improvement on prior solutions that could work up to matrices of order 3. The authors showed that their proposed algorithm runs much more efficiently than an optimization problem. However, the authors state towards the end that extending it to any $n \times n$ for $n > 4$ would be very difficult.

2.3.5 Correlation Matrix Completion using Sequential Differential Evolution

An approach to solve the correlation matrix completion problem using the optimization method called Differential Evolution (DE) was proposed by Mishra [17]. The Differential Evolution procedure of global optimization was originally proposed by Storn and Price [18]. As part of using the method, a random population of elements (consisting of numbers between -1 and 1) that fit the holes in a given in-

complete correlation matrix are generated. Every element of the population would hence correspond to a completion of the matrix. By calculating its eigenvalues, it is possible to determine if it is indeed a desired completion. The author formulated a heuristic measure to give the extent of negativity of the negative eigenvalues or in other words, how far the matrix is from being positive-semidefinite. Using a heuristic way of mixing, a new generation of elements is generated from the given population. The child of an element would replace its parent only if it is stronger than the parent, that is it has a smaller value for the function. The process is repeated for a number of iterations (which is specified by the user) until the desired result is achieved.

I have chosen this method for developing a parallel algorithm for the following reasons.

1. The method does not assume any kind of specific pattern of the unknown entries in the matrix.

2. The method has inherent parallelization that can be exploited.

3. The author claims that the Differential Evolution method is perhaps the fastest evolutionary computational procedure yielding the most accurate solutions to continuous global optimization problems.

Chapter 3

The Algorithms

In fact, there was general agreement that minds can exist on nonbiological substrates and that algorithms are of central importance to the existence of minds. — Vernor Vinge

3.1 Introduction

This chapter discusses the algorithms involved in the approaches to solve the correlation matrix completion problem. Firstly, Evolutionary Strategies (ES) which are part of evolutionary approaches to solve optimization problems are introduced and then an algorithm which makes use of this approach to solve the CCOMAT completion problem is given. The later sections introduce the proposed approach to solve CCOMAT completion in a parallelized way. Finally, an advanced parallel algorithm is proposed.

3.2 Evolution Strategies and Differential Evolution

A global optimization method is a technique which can give best element(s) among all possible alternative elements. Mathematically, it could be a problem to find the point(s) at which the objective value/cost associated with the problem has its optimal (minimal/maximal) value among all others in the known domain. The

minimization problem is a special case of the optimization problem where only the minimal value is looked for. It may be formally defined as below:

Definition Minimization Problem

For an objective function $f : X \subseteq \mathbb{R}^D \rightarrow \mathbb{R}$ where the feasible set $X \neq \phi$, the minimization problem is to

$$\text{Find } x^* \in X \text{ such that } f(x^*) \leq f(x) \qquad \forall x \in X \text{ where } f(x^*) \neq -\infty$$

Evolution Strategies are black-box optimization techniques which are developed based upon nature-based processes of adaptation and evolution. Here by "black-box" technique, we mean that no knowledge of the derivative of the function to be optimized is used and that the evaluations of the objective function that needs to be optimized at various points in the domain are sufficient for finding the optimal value.

Differential Evolution is an Evolutionary Strategy method which is used for global optimization over real continuous spaces. It was first proposed by Stone and Price [19] to solve Chebyshev polynomial fitting. The method was designed not only for being able to use non-differentiable nonlinear cost functions but also having features such as ease of use (with minimal parameters), consistent convergence properties, and parallelizability.

The basic algorithm is given in **Algorithm 1**.

The algorithm takes the three control parameters as below:

- NP - The total number of agents in the population
- CR - Cross-Over Ratio $\in [0,1]$
- F - Recombination Constant $\in [0,2]$

The algorithm also has a function called $CostFunc(Agentx)$ which is a function value associated with each agent and the final goal of the algorithm is to find agent(s) whose $CostFunc$ is almost equal to zero.

Algorithm 1 BasicDiffEvol

Require: NP, CR, F

1: Initialize NP agents of population
2: **for** each agent x_i in the population **do**
3: pick three other distinct agents a_i, b_i and c_i
4: Generate r_i as Normal[0,1] {The mutation and recombination of the algorithm takes place}
5: **if** $r_i < CR$ **then**
6: $y_i := a_i + F(b_i - c_i)$
7: **end if**
8: **if** $CostFunc(Y) < CostFunc(X)$ **then**
9: $x_i := y_i$
10: **end if**
11: **end for**

A Differential Evolution algorithm then can be broken down to the following steps:

1. **Initialization:** Step one of the algorithm performs this where all the agents are initialized to certain random values. Each agent is a value randomly picked out of the available pool of all values.

2. **Recombination and Mutation:** Steps 3-7 show the recombination and mutation. It starts by picking three other distinct agents randomly from the population. This automatically puts the lower limit for NP as four. The values of CR and F come into use here to make a decision for mutation and the way recombination is done. The child is produced by adding one of the agents to the differential weight $(b_i - c_i)$ multiplied by the Recombination constant F.

3. **Selection:** This phase is implemented in Steps 8-10. If the child is fitter than its corresponding parent, it is retained by the algorithm. Otherwise, the parent will prevail in the population of the next generation.

The loop defined between Steps 2 and 11 takes care of applying the steps for all the agents in the population. In a typical practical scenario, these steps may be

repeated for a predefined number of iterations, as will be seen in the next section. In the sections below, the acronym DE is used to refer to Differential Evolution.

3.3 The Sequential Algorithm for CCOMAT Problem

A DE algorithm that can solve the correlation matrix completion problem is given here. The method works for any general real partial symmetric matrix, that is. without assuming any particular pattern of unknown entries within the matrix.

The inputs of the algorithm are

- the partial matrix that needs to be completed
- MAX_ITER which is the maximum number of iterations the algorithm will go through
- $EPS \approx 0$ used as a cutoff to terminate the algorithm if a sufficiently close solution is achieved

The algorithm is an application of the DE algorithm in a more extended form in order to solve the correlation matrix completion problem. A detailed analysis of the steps of Algorithm 2 are given below.

- Step 1 is the same as the one in BasicDiffEvol where the potential candidates are filled in.
- The loop enclosed within steps 2 and 17 takes care of the maximum number of iterations that the algorithm can take.
- In Steps 3-5, values for ReachFunc() are calculated for each of the available agents in the population at the moment, and are stored in a buffer array FV.
- In Step 6, the minimum among the FV arrays is chosen as FV_{min}.
- In Steps 7-9, an early termination criteria is set where if the FV_{min} is sufficiently close to zero, the corresponding mutant may be taken as a solution and the loop is terminated.

- The loop defined within Steps 10 and 16 is the corresponding loop between Steps 2 and 11 in BasicDiffEvol wherein each agent is separately processed.
- In Step 11, IR[1..3] are generated which is analogous to selecting three distinct agents a_i, b_i, c_i different from x_i.
- Step 12 encompasses the recombination and mutation steps in a go. The random number generation and normal distribution are all part of it.
- In Steps 13-15, the selection of tougher agents is done. At the end of the step, better agents are selected.

Algorithm 2 Complete Correlation Matrix - Sequential

Require: Partial Matrix M, MAX_ITER, EPS, CR, F

1: Generate N vectors of size m consisting of real numbers in $[-1, 1]$
2: **for** MAX_ITER iterations **do**
3: **for** $i = 1$ to N **do**
4: $FV[i] = ReachFunc(P_i)$
5: **end for**
6: $FV_{min} = Min(FV[1...N])$
7: **if** $FV_{min} < EPS$ **then**
8: **break**
9: **end if**
10: **for** $i = 1$ to N **do**
11: Generate distinct $IR[0], IR[1] \& IR[2]$ where each of them $\neq i$
12: $Child_i$ = Evolution_Strategy$(P_i, P_{IR[0]}, P_{IR[1]}, P_{IR[2]}, CR, F)$
13: **if** $ReachFunc(Child_i) < ReachFunc(P_i)$ **then**
14: Replace P_i with $Child_i$
15: **end if**
16: **end for**
17: **end for**

The algorithm starts with a population of N elements where each element is a vector of real numbers which is a potentially fillable set of entries into the holes of the partial matrix. It uses a random sample of umpteen number of starting elements. Then the idea is to modify them and create children which are at least as good

as they are so that one of the elements would reach being the target. Once the algorithm reaches the target, it terminates.

This algorithm has the function ReachFunc(Matrix) which calculates a numerical real value for any given completed matrix. Its value indicates the degree of how far the matrix is from being positive semidefinite. For an already PSD matrix, its value will be zero. At the end of the execution of the algorithm, which is determined by the number of iterations, the program hopefully gives us at least one element among N which is desirable.

Pseudocode for computing the function ReachFunc is given in **Algorithm 3**.

Algorithm 3 ReachFunc(Matrix A)

Require: A square symmetric real matrix A

1: $F \leftarrow 0$
2: $sumW \leftarrow 0$
3: $prodW \leftarrow 1$
4: $eig[1..morder] \leftarrow Eigen\_values(A)$
5: **for** $i = 1$ to $morder$ **do**
6: **if** $eig[i] < 0$ **then**
7: $sumW \leftarrow sumW + |eig[i]|$
8: $F \leftarrow F + eig[i] * eig[i]$
9: $prodW \leftarrow prodW * eig[i]$
10: **end if**
11: **end for**
12: **if** $((prodW < 0)||(prodW > 1))$ **then**
13: $F \leftarrow (F + sumW + prodW * prodW)^2$
14: **end if**
15: return F

The algorithm takes a square matrix as its input and calculates its eigenvalues; and depending upon them, it returns a floating point value F.

- In Steps 1-3, three variables F, $sumW$ and $prodW$ are initialized.
- In Step 4, eigenvalues are calculated by a user provided function.
- In Steps 5-11, $sumW$, $prodW$ and F are calculated as the sum, product and sum of squares respectively of all the negative eigenvalues of matrix A.

- In Steps 12-14, depending upon the value of $prodW$, the value of F is adjusted.
- In Step 15, F is returned.

The way random mutation of the elements is done and the way the cost function is defined are such that there is a gravity towards having elements that gives a cost function of zero. In other words, these elements give a completion which is positive semidefinite.

3.4 Basic Parallel Approach for Solving the CCOMAT Problem

In this section, a parallelized version of Algorithm 3 is presented. The parallelization is done by distributing the computation of each of the members of the population in a separate thread of execution. The computationally intensive part of the sequential algorithm is the multiple executions of $ReachFunc()$. The parallel version makes sure that each parallel thread of execution has calls to $ReachFunc$ about the same number of times. This is the way in which the computation is fairly divided among all the processes running in parallel. The pseudocode of the algorithm is as given in Algorithm 4.

The algorithm takes the following inputs:

- Partial matrix M with m unknown entries
- The size of the population n
- Maximum number of iterations MAX_ITER
- $EPS \approx 0$

The algorithm is an application of the DE algorithm in a more extended form in order to solve the correlation matrix problem. A detailed analysis of steps of the algorithm is given below.

- Step 1 initiates a set of n processes where each process takes responsibility for one agent in the population.
- Step 2 happens simultaneously in all processes where an agent is created with a random value in [-1,1].
- The loop enclosed within steps 3 and 25 takes care of the maximum number of iterations the algorithm will execute.
- In Step 4, values for ReachFunc() are calculated for each of the available agents in the population simultaneously and are stored in FV array.

Algorithm 4 Complete Correlation Matrix - Parallel

Require: Partial Matrix M, n, MAX_ITER, EPS, F, CR, MAX_AGE

1: Start n processes which is the same as the size of population :
2: **In Parallel** Generate a random vector of size m with each entry in $[-1,1]$
3: **for** MAX_ITER iterations **do**
4: **In Parallel** Calculate for each process i, $FV[i] = ReachFunc(P_i)$
5: Broadcast FV[i] to every other process
6: **At root**, calculate $FV_{min} \leftarrow Min(FV[1...N])$
7: **if** FV_{min} **then**
8: **Break the execution of all parallel processes**
9: **end if**
10: **In Parallel**
11: **for** each Process i **do**
12: Generate distinct $IR[0]$,$IR[1]$ and $IR[2]$ each of which $\in Int0, n-1 \neq i$
13: $Child_i$ = Evolutionary_Strategy$(F, CR, P_i, P_{IR[0]}, P_{IR[1]}, P_{IR[2]})$
14: **if** $ReachFunc(Child_i) < ReachFunc(P_i)$ **then**
15: Replace P_i with $Child_i$
16: $Age(P_i) \leftarrow 0$
17: **else**
18: $Age(P_i) \leftarrow Age(P_i) + 1$
19: **if** $Age(P_i) > MAX\_AGE$ **then**
20: Generate a random agent and replace P_i with it
21: **end if**
22: **end if**
23: **end for**
24: Broadcast all of the P_i's to all the processes
25: **end for**

- All the values of FV array are broadcasted to all the processes in Step 5.
- Step 6 happens only at the root process where the minimum among the FV arrays is calculated as FV_{min}.
- In Steps 7-9, an early termination criteria is set where if the FV_{min} is sufficiently close to zero, the corresponding agent would be taken as a solution and the loop is terminated.
- The loop defined within Steps 11 and 23 is the corresponding loop between Steps 2 and 11 in BasicDiffEvol wherein each agent is separately processed.

- Step 10 specifies that all the indented steps below from Step 11 to 23 happen simultaneously in parallel in all the processes.
- In Step 14, IR[1..3] are generated which is selecting three distinct agents a_i, b_i, c_i different from x_i from the population.
- Step 12 encompasses the recombination and mutation steps in a go. The random number generation and normal distribution are all part of it
- In Steps 14-22, the selection of tougher agents is done. At the end of the step, better agents among parents and their respective children are selected.
- In Step 16, the values of P_i's are broadcasted to all the processes present in the population.

The algorithm has a potential bottleneck of broadcasting the values after every iteration which could cause a lot of communication cost. The suitability of this algorithm for practically reducing the time taken to solve the problem hence becomes conspicuous when the nodes are not connected with a very high speed and reliable network.

Another problem that could arise in this algorithm is the high number of processes that may be spawned while running it. Typically the operating system restricts the maximum number of processes it allows a user to spawn.

3.5 Improved Parallel DE Approach for Solving the CCOMAT Problem

Based on the basic parallel algorithm presented in the previous section, we propose an improved version of the parallel DE algorithm with features that can make it potentially better. The feature we have added is to pack agents among processes rather than have a process for every single agent. This could optimize the algorithm because interprocess communication is in general way more expensive than intraprocess communication. This could lead to communication happening in somewhat bigger bulk but it lessens the number of such communications.

In Algorithm 5:

- Step 1 initiates a set of NUM_CORE processes where each process has one full core to execute upon. Therefore, NUM_CORE should be less than the total number of physical cores available throughout the cluster.
- Step 2 happens simultaneously in all processes where each agent is assigned a random value in the available domain.
- The loop enclosed within Steps 4 and 28 takes care of the maximum number of iterations the algorithm will execute.
- In Step 5, values for ReachFunc() are calculated for each of the available agents in the population simultaneously and are stored in FV[].
- In Step 6, all the values of FV array are broadcasted to all the processes.
- Step 7 happens only at the root process where the minimum among the FV arrays is calculated as FV_{min}.
- In Steps 8-10, an early termination criteria is set where if the FV_{min} is sufficiently close to zero, the corresponding agent would be taken as a solution and all the processes are terminated.
- The loop defined within Steps 11 and 27 is the corresponding loop between Steps 2 and 11 in BasicDiffEvol where each agent is separately processed.

Algorithm 5 Complete Correlation Matrix - Parallel

Require: M, NUM_PROCS, MAX_ITER, EPS, F, CR, MAX_AGE

Start NUM_PROCS processes
In Parallel for each process i, Generate N_A/N_P number of random vector of size m with each entry in $[-1,1]$
In Parallel for each process i, Calculate the range of agents as $range(i)$
for MAX_ITER iterations **do**
  In Parallel for each process i, Calculate $FV[i] = ReachFunc(P_i)$
  Broadcast FV[i] to every other process
  At root: calculate $FV_{min} \leftarrow Min(FV[1...N])$
  if $FV_{min} < EPS$ **then**
    Break the execution of all parallel processes
  end if
  In Parallel
  for each process i **do**
    for $j \in range(i)$ **do**
      Generate distinct $IR[0]$,$IR[1]$ and $IR[2]$ each of which $\neq i$
      $Child_i$ = Evolutionary_Strategy$(F, CR, P_j, P_{IR[0]}, P_{IR[1]}, P_{IR[2]})$
      if $ReachFunc(Child_j) < ReachFunc(P_j)$ **then**
        Replace P_j with $Child_j$
        $Age(P_i) \leftarrow 0$
      else
        $Age(P_i) \leftarrow Age(P_i) + 1$
        if $Age(P_i) > MAX\_AGE$ **then**
          Generate a random agent and replace P_i with it
        end if
      end if
    end for
    Broadcast all of the P_i's to all the processes
  end for
end for

- Steps 11 and 12 specify that all the indented Steps below from Step 13 to 25 happen in parallel in all the processes.
- In Step 14, IR[1..3] are generated which is analogous to selecting three distinct agents a_i, b_i, c_i different from x_i.
- Step 15 encompasses the recombination and mutation Steps in a go. The random number generation and normal distribution are all part of it.
- In Steps 16-24, the selection of tougher agents is done. An agent may survive if it is at least as good as its child. Otherwise, the child will replace it. Also, we implemented aging in these Steps where an agent will be replaced by a freshly generated one if it does not produce a capable child for as many iterations as MAX_ITER.
- In Step 26, the values of the P_i's are broadcasted to all the processes present in the population.

Chapter 4

Experimental Results

It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.
— Richard Feynman

4.1 Introduction

In this chapter, we discuss the experimental verification of our proposed algorithms. In Sections 4.2.1 and 4.2.2, we give a description of the entire setup for conducting our experiments. That includes the physical cluster we used, and the different libraries/tools primarily for Scientific Computing. In Sections 4.2.3 and 4.2.4, we explain how we generated our test cases and give an example of a typical test case that we used with an incomplete matrix. Then, as a possible solution we fill up the unknown entries to make it a correlation matrix. In further sections, we describe the results for the test cases and explain our observations and give our analysis of the findings.

4.2 The Setup

4.2.1 The Physical Cluster

The cluster over which experiments were carried out consisted of a set of nodes from the Cyclops high performance distributed memory system [20]. It consists of an IBM iDataPlex dx360 M2 system running Ubuntu Server 10.04. Each node on the cluster consists of two quad-core Xeon x5550 2.67 GHz Intel processors which effectively makes 8 cores and has access to 12 GB of memory. All the nodes are connected to a 10 Gb dual-port Ethernet high-speed network.

4.2.2 The Libraries/Tools

The algorithms are implemented in the C programming language assuming a distributed memory model where each node has its own memory space and the nodes communicate with each other through passing messages via the Message Passing Interface (MPI) model. MPI is a widely used standard for implementing portable parallel distributed applications [21]. Throughout the algorithm, for generating newer agents, Mersenne Twister random number generation is used. For evaluation of eigenvalues of matrices which are used in the evaluation of $ReachFunc()$, eigen-solvers from SLEPc have been used. SLEPc works over data structures available in PETSc which is a suite of data structures such as matrices that facilitate the eigenvalue computations. Brief descriptions of these can be seen in the paragraphs below.

Mersenne Twister random number generator has been used to generate the scalar values in the agents generated. Every agent is a vector of scalar values which are as many as the number of unknown entries in the initial partial matrix. Mersenne Twister generates random integers with a period of $(2^{19937} - 1)$ [22]. We normalize every integer thus generated to a unique corresponding real value between -1 and +1 and then used it as a scalar value in an agent.

Message Passing Interface (MPI) is a specification for an Application Programming Interface that has functions facilitating several processes of a program which can run independently on multiple computers and can communicate with each other by passing messages [23]. When the different MPI processes are executed, they all have separate process spaces and run completely independent of each other except during communications. The user can instantiate as many processes as she wants and start all of them simultaneously and independently. An MPI process has to run within a communicator and different processes within a communicator are recognized with their respective IDs. The root process will have the ID '0'. The rest of the processes will be assigned IDs from 1 to the value of (total number of processes - 1). We have used MPICH2 which is a widely used implementation of the Message Passing Interface (MPI).

PETSc stands for **P**ortable **E**xtensible **T**oolkit for **S**cientific **C**omputation and is a suite of data structures and routines that can be used for coding large-scale scientific applications running on parallel computers [24]. It provides solutions by modeling problems as partial differential equations. The data structures it provides includes parallel matrices, i.e. matrices that may be spread across different nodes. The parallel processes using PETSc communicate with each other using MPI routines. Details of usage can be found in the user reference manual [25].

SLEPc stands for **S**calable **L**ibrary for **E**igenvalue **P**roblem **C**omputations [26]. It is a package developed for solving large sparse eigenvalue problems on parallel computers and particularly targeted to solving matrix problems having low ranks. SLEPc works on the top of PETSc and hence it is necessary to install PETSc before installing SLEPc. In a way it is an extension of PETSc with functionality to do eigenvalue computations [27].

4.2.3 Test Cases

The generation of test cases is done by extracting historical time series data from stock market data for some of the known stock indices. We construct a com-

pleted correlation matrix by calculating the correlation between every pair of indices. Then we mask some of the values in a random manner and take them as unknowns. We have masked varied numbers of entries in increments of 10% from 10% of all the non-diagonal entries up to 90% of them.

4.2.4 An Example with Implementation Details

Here given below is a partial matrix M which is one of the typical inputs the algorithm takes. The symbol "?" is used to denote an unknown value.

$$M = \begin{bmatrix} 1.000 & 0.454 & ? & ? & ? & ? & ? & ? \\ 0.455 & 1.000 & 0.278 & ? & ? & ? & 0.728 & 0.552 \\ ? & 0.278 & 1.000 & ? & 0.634 & ? & 0.553 & 0.270 \\ ? & ? & ? & 1.000 & ? & 0.117 & 0.627 & ? \\ ? & ? & 0.634 & ? & 1.000 & ? & 0.259 & 0.004 \\ ? & ? & ? & 0.117 & ? & 1.000 & 0.612 & 0.711 \\ ? & 0.729 & 0.553 & 0.627 & 0.259 & 0.612 & 1.000 & 0.534 \\ ? & 0.552 & 0.269 & ? & 0.004 & 0.710 & 0.534 & 1.000 \end{bmatrix}$$

Since the matrix is a symmetric one, we can restrict our attention to one half, i.e. the upper triangular matrix or the lower triangular one. The final values for unknown entries in either of the halves will be the same due to symmetry. In the given example we see 14 unknown entries in the upper triangle. This fixes the vector size of an agent to 14. Every agent will be a vector of 14 scalars in the range of [-1,1] corresponding to each unknown location of M. Our algorithm will generate as many agents as specified. At each step, it applies the DE strategy and recombination step to improve agents, i.e., obtain agents with as small value of $ReachFunc()$ as possible. Once it finds an agent/a set of agents it would terminate and using those agent(s), a completed matrix such as A may be formed. We have set the maximum number of iterations as 100,000 although in most typical cases that converge, the convergence happens within few hundred iterations.

In the completed matrix A, the new entries have been highlighted in bold so that they are conspicuous. This is one of the possible completions of the partial matrix M given that we want to preserve the positive semidefinite property intact. The number of such possible completions become limited when we induce more properties to be preserved.

$$A = \begin{bmatrix} 1.000 & 0.454 & \mathbf{0.531} & \mathbf{0.723} & \mathbf{0.404} & \mathbf{0.475} & \mathbf{0.583} & \mathbf{0.489} \\ 0.455 & 1.000 & 0.278 & \mathbf{0.537} & \mathbf{0.315} & \mathbf{0.459} & 0.729 & 0.552 \\ \mathbf{0.531} & 0.278 & 1.000 & \mathbf{0.272} & 0.634 & \mathbf{0.689} & 0.553 & 0.270 \\ \mathbf{0.723} & \mathbf{0.537} & \mathbf{0.272} & 1.000 & \mathbf{0.451} & 0.117 & 0.627 & \mathbf{0.383} \\ \mathbf{0.404} & \mathbf{0.315} & 0.634 & \mathbf{0.451} & 1.000 & \mathbf{0.322} & 0.259 & 0.004 \\ \mathbf{0.475} & \mathbf{0.459} & \mathbf{0.689} & 0.117 & \mathbf{0.322} & 1.000 & 0.612 & 0.711 \\ \mathbf{0.583} & 0.729 & 0.553 & 0.627 & 0.259 & 0.612 & 1.000 & 0.534 \\ \mathbf{0.489} & 0.552 & 0.269 & \mathbf{0.383} & 0.004 & 0.711 & 0.534 & 1.000 \end{bmatrix}$$

In the current implementation of the parallel algorithms, the inputs/outputs are completely handled by the process with rank zero. The root process will input the partial matrix from a file and broadcasts it to remaining processes in the communicator.

4.3 The Experimental Parameters

The basic DE parameters of NP, F and CR used in the parallel algorithms were mostly influenced by the original sequential algorithm. NP is taken depending upon the vector size of agents which in our case is the number of holes in the original partial matrix. We have tested for values of NP that are multiples of the number of holes. For example, in the tables below, 5X refers to the fact that the population size NP is 5 times the corresponding size of number of holes. In the case of 8 x 8 matrices, the number of non-diagonal entries will be 56. Considering only the upper triangular matrix, it will be 28. 10% of it is approximately 3. So, when the number of holes is 3 the size of NP corresponding to 5X is 15 and when

corresponding to 10X will be 30 and so on. The value of CR has been set to 0.9 and F has been set to 1. The EPS value of 10^{-11} is chosen for testing proximity to zero. As explained in the previous section, we have used the test cases where the number of holes varies between 10% of all the non-diagonal entries to 90% of all non-diagonal entries so that we get to study the performance of our algorithms on all of them.

4.4 Results for the Algorithms

We have taken 8x8 matrices as our test cases and carried out a set of experiments on our cluster. We divide the section into following subsections.

- Evaluation of Basic Parallel DE Algorithm
- Evaluation of Improved Parallel DE Algorithm

4.4.1 Evaluation of Basic Parallel DE Algorithm

We have used the sequential algorithm of SK Mishra as our benchmark to evaluate our basic parallel algorithm. We have run the sequential FORTRAN program given by SKMishra [17] on a single node of the cluster. Effectively, the sequential program runs over a single core of a CPU as it is a single thread of execution.

We have deployed our implementation of the Basic Parallel algorithm on the cluster and run it on 4 nodes and 16 nodes of the cluster and measured the execution times.

We report our initial set of results in Table 4.1. We have run the programs for 10 different random test cases and averaged the execution times of all converged cases. The execution time reported is in the number of seconds taken. Rows represent the number of holes in the initial partial matrix as a percentage with respect to the total number of non-diagonal entries. Columns are used for giving the hardware that is used for execution and the number of agents (per hole) in the population.

Table 4.1: The average execution time (in seconds) of 8×8 matrix completions

Holes %	5X			10X			15X		
	Single	4	16	Single	4	16	Single	4	16
10	3.116	0.053	0.051	6.197	0.057	0.05	9.337	0.0676	0.046
20	6.264	0.217	0.213	12.405	0.249	0.15	18.734	0.5134	0.157
30	8.184	0.249	0.178	16.329	0.756	0.225	24.344	1.1051	0.229
40	11.075	0.524[a]	0.308[a]	22.414	1.721	0.375	33.622	2.3159	0.425
50	14.201	34.303[b]	12.844[c]	28.293	2.384	0.503	42.551	3.6266	0.661
60	16.995	2.044[c]	0.632[c]	33.933	5.69	1.098	51.007	8.4510	1.154
70	20.203	1.170[c]	0.336[c]	40.226	3.657	1.553	60.315	8.5990	6.241
80	21.968	68.997[d]	15.322[d]	43.904	3.487	0.605	65.990	8.5355	2.123
90	24.771	0.478	0.22	47.631	1.982	0.368	71.359	4.5480	1.276

[a] An underestimate average time as two out of the ten cases did not converge.
[b] The bulky average time is due to an outlier and is still an underestimate as one of the ten cases did not converge.
[c] An underestimate average time as one out of the ten cases did not converge.
[d] The bulky average time is due to presence of one outlier.

In order to explain the average behavior more robustly, we present the median results along with the average results of the algorithm for execution on 16 nodes in Table 4.2.

We have plotted the average execution times when the algorithm is run over 4 nodes in Figure 4.1. We have removed some outliers to make a more elegant looking graph which is Figure 4.2. Correspondingly, we have plotted the average execution times over 16 nodes in Figure 4.4 and Figure 4.5.

Discussion

We make the following observations:

- The time of execution of the sequential program increases steadily with an increase in the number of agents/holes as seen in Figure 4.3.
- The parallel execution time is minimal when the number of holes is minimal

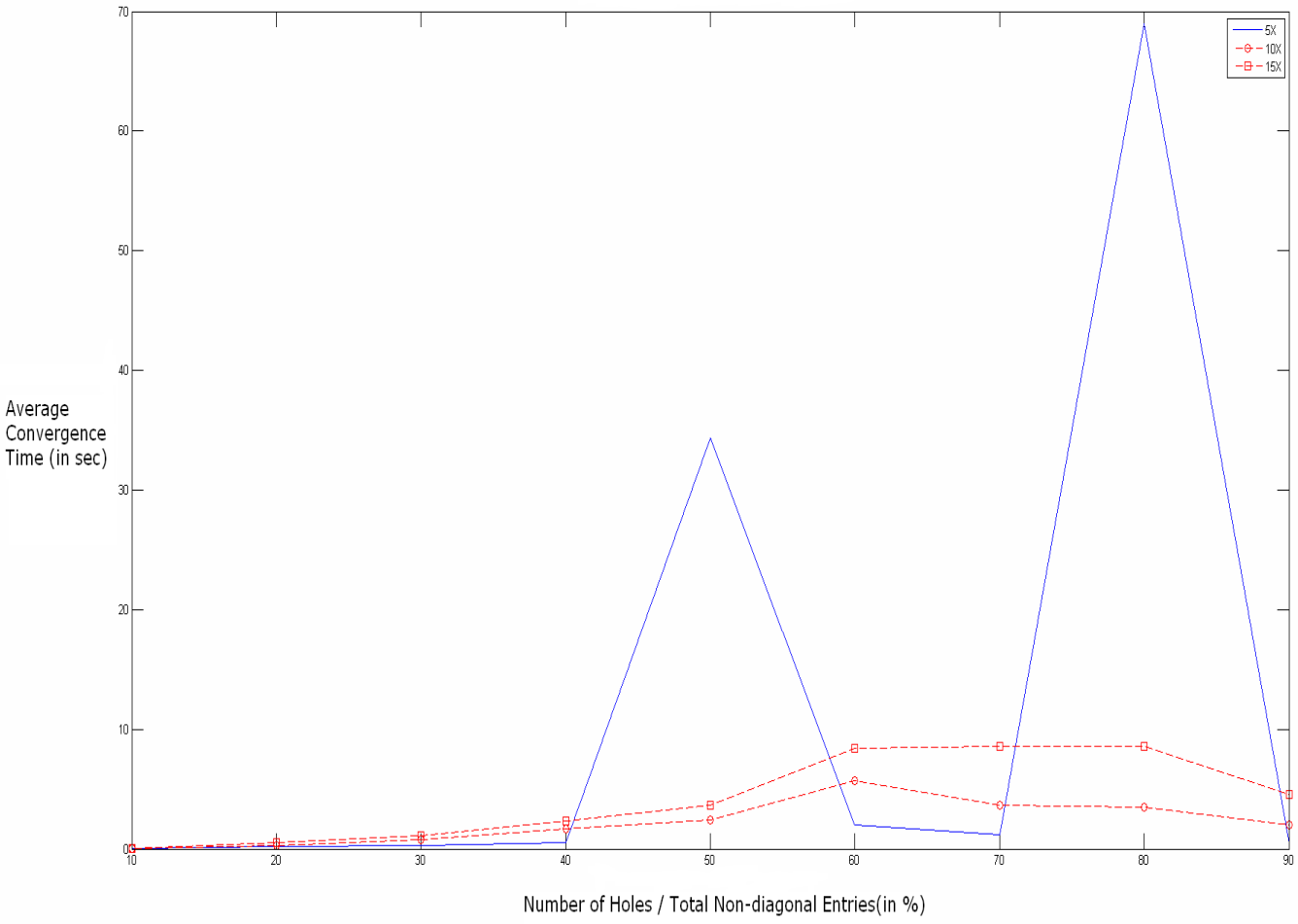

Figure 4.1: Average Execution Times for Basic DE Algorithm Run over 4 Nodes

and it increases with an increase in the number of holes up to certain point and then starts decreasing.

- The average times of execution of certain 5X cases is extremely high due to the presence of certain outliers.

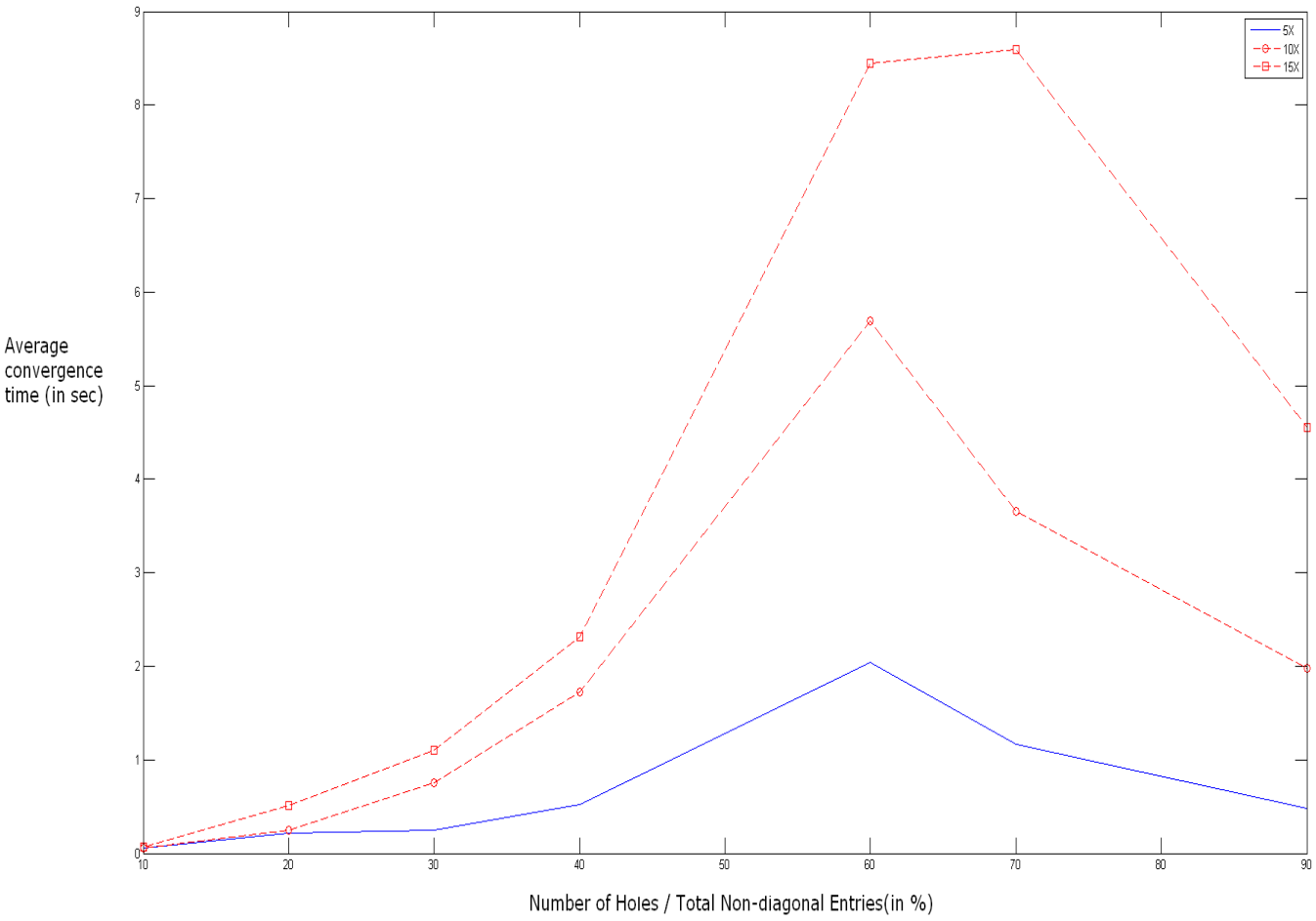

Figure 4.2: Average Execution Times for Basic DE Algorithm over 4 Nodes without Outliers

We can explain the increase in sequential algorithm execution times by the fact that the execution time there depends almost exclusively on the number of fitness tests, i.e., the number of function calls to $ReachFunc()$. Due to differences in implementation details such as the way random numbers are generated, the running time of the sequential program has slightly different characteristics to that of the

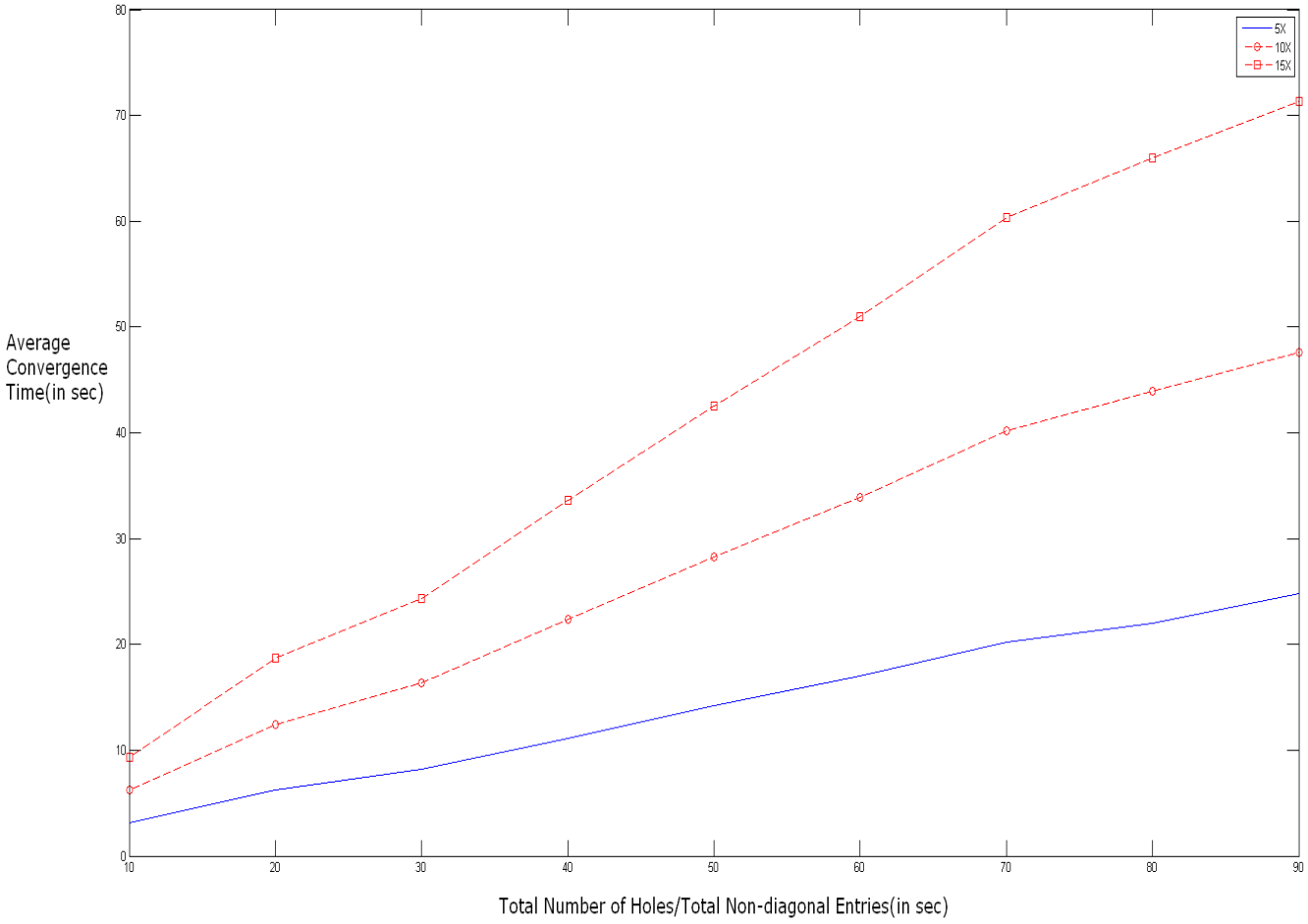

Figure 4.3: Execution Times for Sequential FORTRAN Program

corresponding parallel implementation. That's one of the reasons we also do not see outlier data. When there are more agents in the population, there are more fitness-tests for each iteration and this increases the execution time. When the number of holes increases, we have more agents correspondingly (as tabulated) and this results in a greater number of fitness tests.

Table 4.2: The average execution time(in seconds) of 8×8 matrix completions

Holes %	5X		10X		15X	
	Avg	Med	Avg	Med	Avg	Med
10	0.051	0.050	0.05	0.076	0.046	0.030
20	0.213	0.136	0.15	0.118	0.157	0.304
30	0.178	0.120	0.225	0.174	0.229	0.291
40	0.308[a]	0.307[a]	0.375	0.272	0.425	0.455
50	12.844[b]	0.166[b]	0.503	0.282	0.661	0.318
60	0.632[b]	0.561[b]	1.098	0.619	1.154	1.168
70	0.336[b]	0.120[b]	1.553	0.619	6.241	2.161
80	15.322	0.211	0.605	0.400	2.123	1.941
90	0.22	0.053	0.368	0.102	1.276	0.919

[a] An underestimate average time as two out of the ten cases did not converge.

[b] An underestimate average time as one out of the ten cases did not converge.

In the case of parallel execution, the execution times increase steadily up to a certain point which can be explained in the same way as the sequential one. But we see that it decreases beyond a point. This is due to the fact that the execution time depends so much upon the communication that happens between processes after every iteration. The number of iterations needed to converge decreases when the number of unknowns is very high. This eventually decreases the execution time, although we notice that it is still higher than the execution time when we had very few holes and few nodes to work with.

In Table 4.1, we have footmarked some cases where we have seen bulky execution times due to the presence of an outlier. In the case of 50% of non-diagonal entries being holes, the times of execution when 4 nodes are used are 0.127878, **113.945515**, 0.171923, 0.055251, 0.593854, 0.079021, 0.120626, 0.279233, 0.220109 and a case of 399.700662 seconds which did not converge. We see that among the nine convergent cases, one of them is very high. The corresponding number of

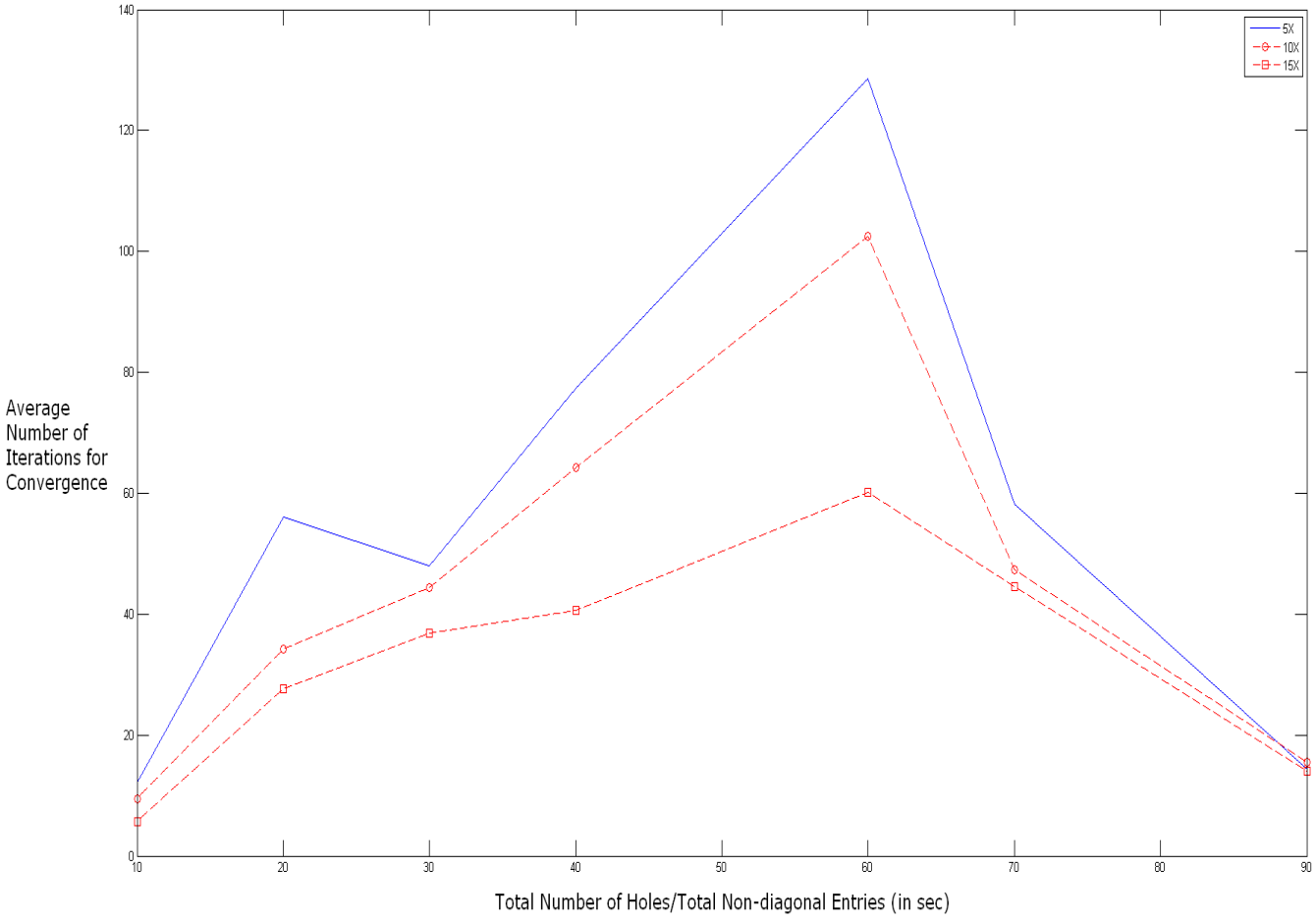

Figure 4.4: Average Number of Iterations Needed for Basic DE Algorithm over 16 Nodes without Outliers

iterations for convergence is reported as 27848 which is pretty high compared to the rest of the converged cases whose highest is 139. This outlying condition can be best attributed to the randomness of the algorithm. Also we note that the same problem occurs when we use 10X the number of nodes, and here the problem converges in 131 iterations. This clearly shows that it takes a long while before we

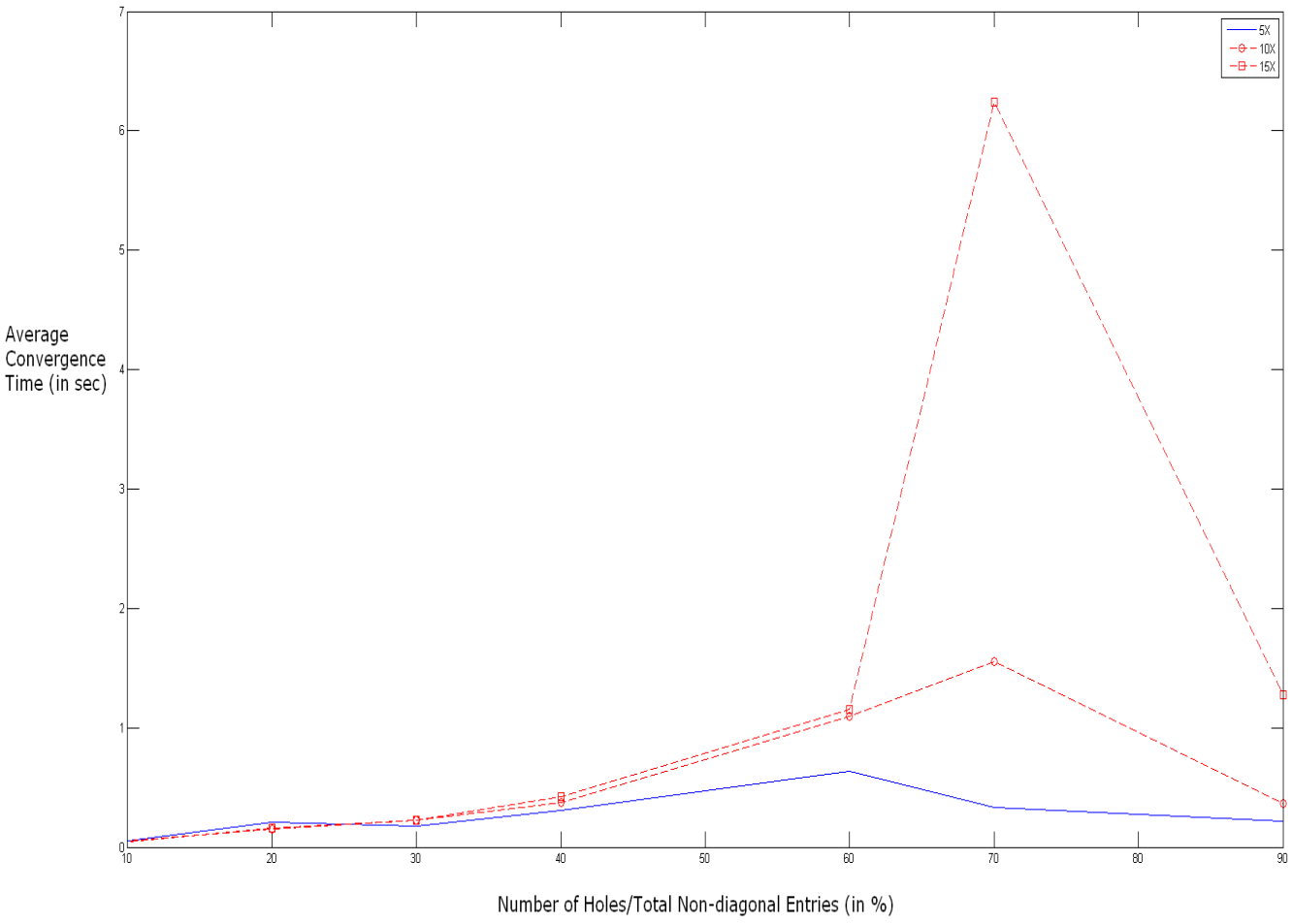

Figure 4.5: Average Number of Iterations Needed for Basic DE Algorithm over 16 Nodes without Outliers

get a desired agent in the case of lesser nodes. This could be attributed to lesser choices of recombination or lesser randomness in the initial population that makes it difficult to reach the goal state in certain problems.

Figure 4.1 clearly demonstrates that barring the outliers, algorithms converge quickly when they use a smaller number of agents such as 5x. This shows that even though on average the running time of 5x is small, it can be very large in some cases depending upon the particular problem at hand.

4.4.2 Evaluation of Improved Parallel DE Algorithm

In this section, we present the results we have obtained after running the Improved Parallel DE algorithm which is presented in the previous chapter as Algorithm 5. We have used the same 10 cases which we used for testing the Basic DE algorithm and recorded the total number of iterations needed and the execution time taken for convergence. In Table 4.3 we tabulate the average number of iterations required for convergence for the Basic Parallel DE algorithm and the Improved Parallel DE algorithm. We have plotted the average iterations required for convergence for Improved DE Algorithm in Figure 4.6.

Table 4.3: Comparisons of Average Number of Iterations between Basic Parallel DE Algorithm and Improved Parallel DE Algorithm

Holes %	5x		10x		15x	
	Basic	Improved	Basic	Improved	Basic	Improved
10	12.1	12.1	9.5	6.9	5.7	4.8
20	56.1	56.1	34.3	32.1	27.6	25.9
30	48.0	48.0	44.4	44.4	36.9	38.4
40	77.38[a]	77.375[a]	64.2	64.2	40.6	41.2
50	3129.9[b]	3129.9[b]	61.4	50.6	41.5	49.3
60	128.7[b]	128.7[b]	102.5	49.3	60.2	49.3
70	58.11[b]	66.0[b]	47.3	43.5	44.5	43.5
80	2749.7	2749.6	36.4	37.6	33.7	33.2
90	14.3	15.7	15.5	13.8	14.0	10.6

[a] An underestimate average time as two out of the ten cases did not converge.

[b] An underestimate average time as one out of the ten cases did not converge.

We now tabulate the comparisons of average execution times of Basic and Improved DE algorithms in Table 4.4. We have plotted the comparison of execution times for each of the algorithms in each case of the number of agents separately. The plots in Figure 4.7, Figure 4.8 and Figure 4.9 display the comparisons of the times for the cases of 5x, 10x and 15x respectively.

Table 4.4: Average Execution Time for Different Algorithms

Holes %	5x		10x		15x	
	Basic	Improved	Basic	Improved	Basic	Improved
10	0.051	0.047	0.05	0.042	0.046	0.042
20	0.213	0.175	0.15	0.148	0.157	0.187
30	0.178	0.171	0.225	0.227	0.229	0.283
40	0.308[a]	0.295[a]	0.375	0.377	0.425	0.476
50	12.844[b]	12.70[b]	0.503	0.335	0.661	0.502
60	0.632[b]	0.622[b]	1.098	0.615	1.154	0.580
70	0.336[b]	0.358[b]	1.553	0.522	6.241	0.518
80	15.322	15.656	0.605	0.454	2.123	0.621
90	0.22	0.119	0.368	0.183	1.276	0.205

[a] An underestimate average time as two out of the ten cases did not converge.

[b] An underestimate average time as one out of the ten cases did not converge.

Some of the observations include the following:

- The average number of iterations needed for convergence for Improved Parallel DE is less than or equal to that required for Basic Parallel DE algorithm.
- The graph of average number of iterations required for convergence is similar to a normal graph with the average around 50-60% of the total number of holes.
- The average time of execution has improved from Basic Parallel DE as compared to Improved Parallel DE.
- The outliers are very moderate for the Improved Parallel DE algorithm.

Discussion

The execution time of the parallel algorithm will involve the computation cost at each of the nodes and the communication cost which is the cost of sending and receiving data elements through messages. We can see that the communication cost of parallel execution time is directly related to the number of iterations. This is because in every iteration, we broadcast certain data within processes.

In Table 4.3, we see how the number of iterations required for convergence varies with the number of holes for different cases. We observe that when the number of holes is neither high nor low, the iterations are at a maximum. Iterations are at a minimum whenever the number of holes is too high or too low. This clearly shows that the parallel time is to a great extent affected by the number of iterations as this will result in the increase in the communication cost. One can see from the plotted data in Figure 4.4 that the number of iterations is at a maximum in the middle when the number of known and unknown entries in the matrix is nearly equal. We have removed a couple of outliers where the number of iterations for 5X is very high so that we have an elegant look at the general trend.

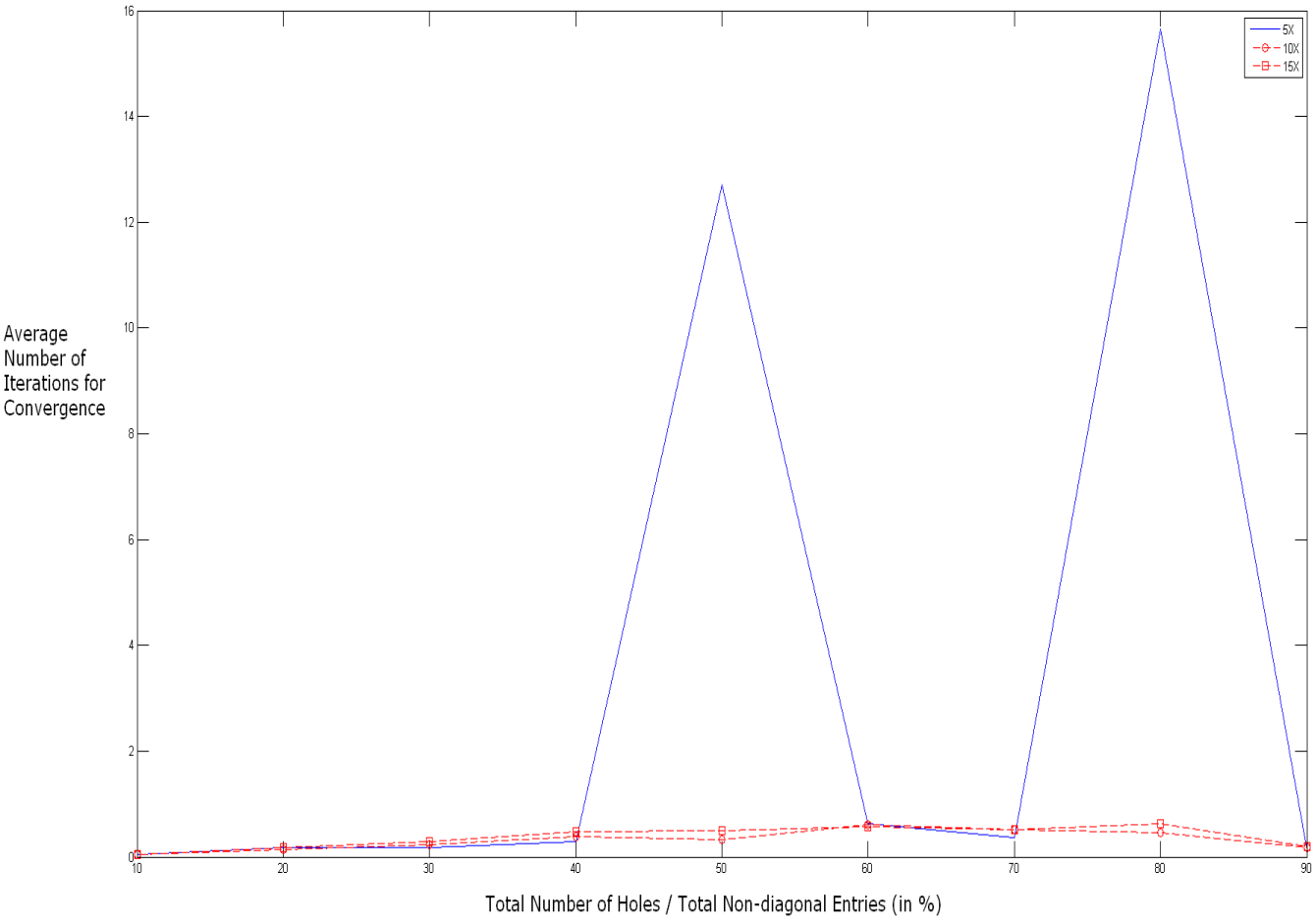

Figure 4.6: Number of Iterations Needed for Improved Algorithm over 16 Nodes

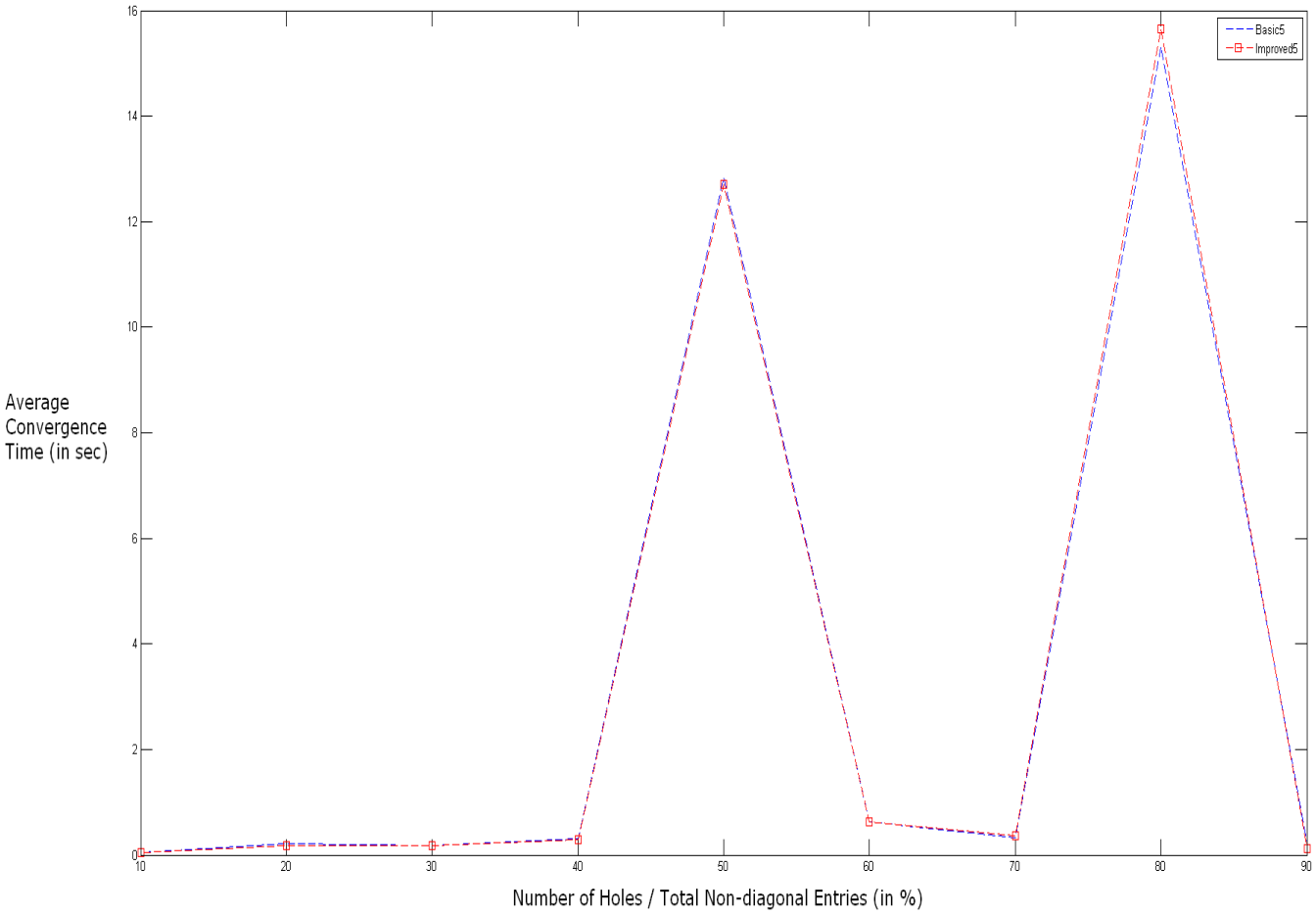

Figure 4.7: Average Execution Times of Improved and Basic Algorithms over 16 Nodes Using 5x Agents

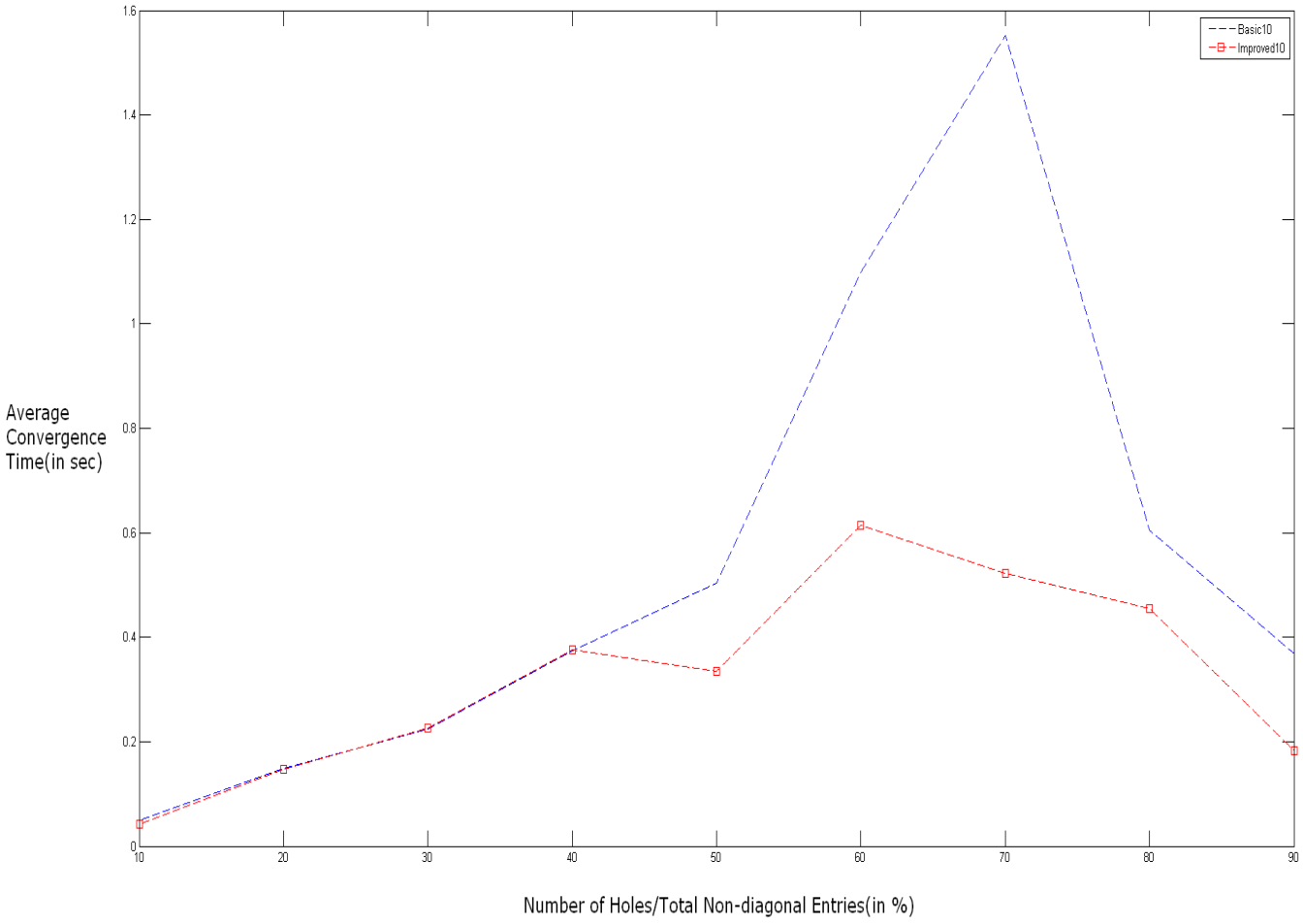

Figure 4.8: Average Execution Times of Improved and Basic Algorithms over 16 Nodes Using 10x Agents

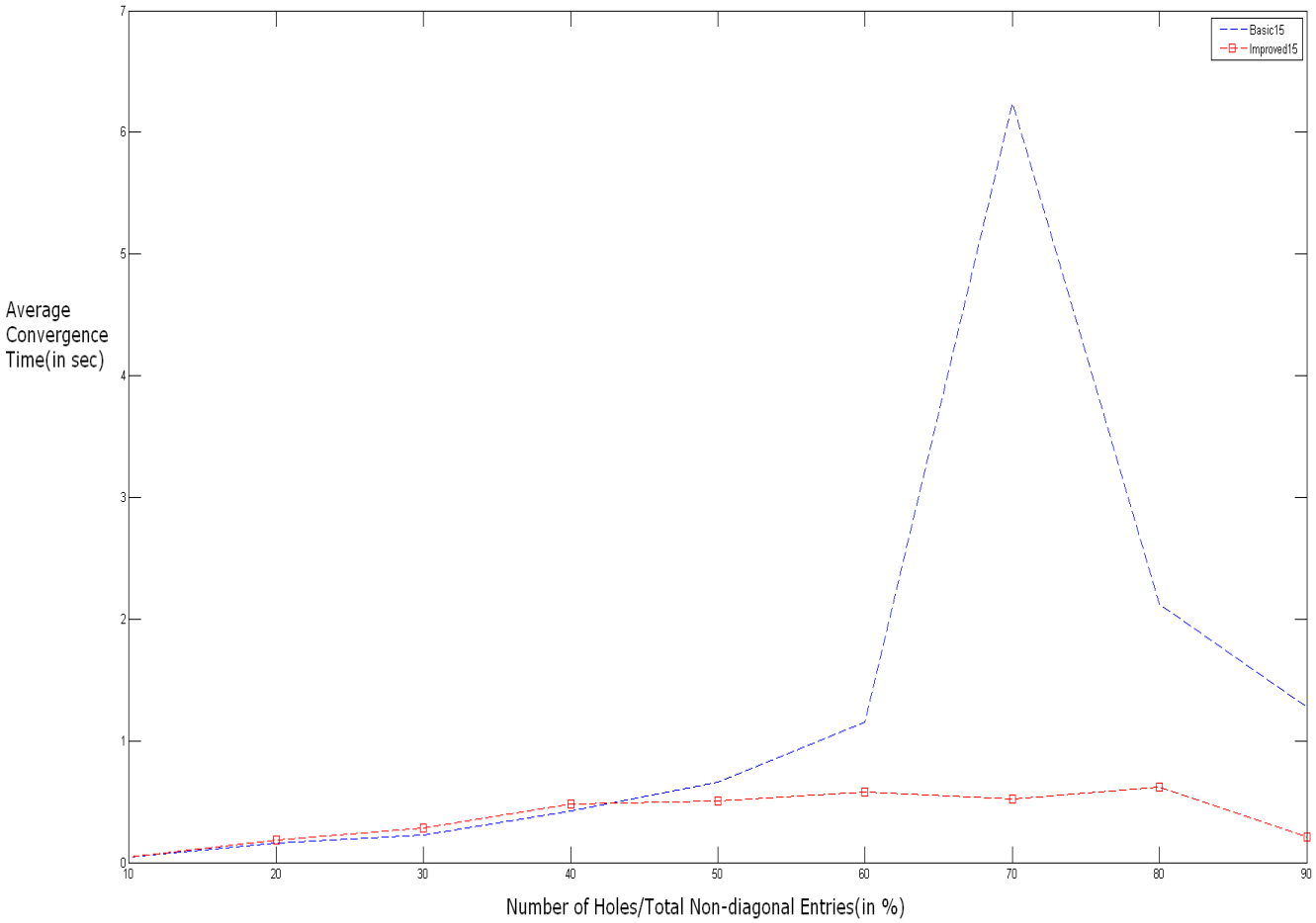

Figure 4.9: Average Execution Times of Improved and Basic Algorithms with 16 Nodes Using 15x Agents

Chapter 5

Conclusions

A conclusion is the place where you got tired of thinking. — Arthur Bloch

5.1 Introduction

In this chapter, we discuss our conclusions and future work that can be done.

5.2 Conclusions

We have verified that parallelization benefits the method of Differential Evolution in solving CCOMAT problems. In the algorithms we have seen, fitness test function $ReachFunc()$ is the most computationally intensive one and in our parallel implementations we could distribute it evenly to different cores in the cluster. We could deduce that communication cost dominates while executing the parallel algorithm due to the increase in the execution time and the increase in the number of iterations. When the method converges in a lesser number of iterations, the running time is significantly reduced due to a decrease in communication cost. We observed that the number of agents used for increases, the corresponding number of iterations that may be needed for convergence decreases and hence total cost of the algorithm decreases since the communication cost is the one dominating factor.

In summary, we would recommend the Parallel DE approach for solving CCO-MAT problems when the cluster of computers available is connected with a very high speed network so that the communication cost is kept to a minimum.

5.3 Future Work

We would hope to see a better heuristic that could reduce the number of iterations as this would have a reduction in communication cost which is a major component of the total execution time of the parallel algorithm.

In Chapter 2, we have seen some specific theoretical structures which would easily be solved for a correlation completion. We could have an add-on to our algorithm to pre-check if a given partial matrix pertains to a known theoretical structure that has already been studied so that handling those cases is straightforward. The evaluation of the theoretical formula may be done in parallel with greater efficiency.

An important problem could be completing a correlation matrix with a certain definite property such as lowest rank or maximum determinant, etc. If someone could come up with a variation in the Evolutionary Strategy step of DE and the way that $ReachFunc()$ is evaluated, one can directly incorporate such a development into our algorithms to work using parallel computers. We look forward for such a contribution from the Heuristic Algorithmic/Scientific Computing community.

One could extend the algorithm by including an island migration strategy as described by Tasoulis et al. while describing their proposed parallel Differential Evolution approach [28]. Every island has an independent population which evolves separately and migration of elements takes place periodically through a ring or a more sophisticated topology. This could give improved results in terms of convergence.

Bibliography

[1] Anthony Austin and Jose Garcia and Stephen Jong and Gilberto Hernandez. Matrix Completion: An Overview. URL http://cnx.org/content/m33136/latest/. → pages 1

[2] Hannes Werthner, Hans Robert Hansen, and Franceso Ricci. Recommender systems. *Hawaii International Conference on System Sciences*, 0:167, 2007. ISSN 1530-1605. doi:http://doi.ieeecomputersociety.org/10.1109/HICSS.2007.459. → pages 2

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi:10.1109/MC.2009.263. URL http://dx.doi.org/10.1109/MC.2009.263. → pages 3

[4] Raghunandan H. Keshavan and Sewoong Oh and Andrea Montanari. Matrix Completion from a Few Entries. *CoRR*, abs/0901.3150, 2009. → pages 3

[5] Y Koren. The BellKor Solution to the NetFlix Grand Prize. URL http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf. → pages 3

[6] A Töscher and M Jahrer and R Bell. The BigChaos Solution to the Netflix Grand Prize. URL http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf. → pages 3

[7] Martin Piotte and Martin Chabbert. The Pragmatic Theory solution to the Netflix Grand Prize. URL http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf. → pages 3

[8] Amit Singer. A remark on global positioning from local distances. *Proceedings of the National Academy of Sciences*, 105(28):9507–9511, 2008. doi:10.1073/pnas.0709842104. URL http://www.pnas.org/content/105/28/9507.abstract. → pages 4

[9] Robert Grone, Charles R. Johnson, Eduardo M. Sá, and Henry Wolkowicz. Positive definite completions of partial hermitian matrices. *Linear Algebra and its Applications*, 58:109 – 124, 1984. ISSN 0024-3795. doi:DOI:10.1016/0024-3795(84)90207-6. URL http://www.sciencedirect.com/science/Marticle/B6V0R-45GWNPR-S/2/e6cf9f07533aa113b7456d76130cb138. → pages 8, 9

[10] Ronald L. Smith. The positive definite completion problem revisited. *Linear Algebra and its Applications*, 429(7):1442 – 1452, 2008. ISSN 0024-3795. doi:DOI:10.1016/j.laa.2008.04.020. URL http://www.sciencedirect.com/science/article/B6V0R-4SPSHN6-2/2/56d6cbe1694dc7fcef1a8f4d007ff52c. → pages 9

[11] Wayne Barrett, Charles R. Johnson, and Pablo Tarazaga. The real positive definite completion problem for a simple cycle. *Linear Algebra and its Applications*, 192:3 – 31, 1993. ISSN 0024-3795. doi:DOI:10.1016/0024-3795(93)90234-F. URL http://www.sciencedirect.com/science/article/B6V0R-45F5BKX-44/2/6b5156e0896414f4cabbb4d7ce6b9297. → pages 9, 10

[12] He Ming and Michael K. Ng. Toeplitz and positive semidefinite completion problem for cycle graph. *Numerical Mathematics, A Journal of Chinese Universities*, 14(1), Feb 2005. URL http://math.nju.edu.cn/CiNM/pdf/2005067.pdf. → pages 11

[13] W. Glunt, T. L. Hayden, Charles R. Johnson, and P. Tarazaga. Positive definite completions and determinant maximization. *Linear Algebra and its Applications*, 288:1 – 10, 1999. ISSN 0024-3795. doi:DOI:10.1016/S0024-3795(98)10211-2. URL http://www.sciencedirect.com/science/article/B6V0R-3W6M0B4-1/2/b782d1a82af8970d873e6e95e0ae4400. → pages 12

[14] Higham, Nicholas J. Computing the nearest correlation matrix - A problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002. doi:10.1093/imanum/22.3.329. URL http://imajna.oxfordjournals.org/content/22/3/329.abstract. → pages 12

[15] C. Kahl and M. Gunther. Complete the correlation matrix. In *From Nano to Space*, pages 229–244. Springer, 2008. URL http://www.springerlink.com/content/t3005r255003w68v. → pages 13

[16] Mark Budden, Paul Hadavas, Lorrie Hoffman, and Chris Pretz. Generating valid 4×4 correlation matrices. *Applied Mathematics E-notes*, 7:53–59, 2007. URL http://www.emis.de/journals/AMEN/2007/060311-1.pdf. → pages 13

[17] SK Mishra. Completing correlation matrices of arbitrary order by Differential Evolution method of global optimization: A Fortran program. MPRA Paper 2000, University Library of Munich, Germany, March 2007. URL http://ideas.repec.org/p/pra/mprapa/2000.html. → pages 13, 33

[18] Rainer Storn and Kenneth Price. Differential Evolution - A Simple and Efficient adaptive scheme for Global Optimization over Continuous Spaces. Technical report, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.5398&rep=rep1&type=pdf. → pages 13

[19] Rainer Storn and Kenneth Price. Differential Evolution A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341 – 359, 1997. URL http://www.springerlink.com/content/X555692233083677. → pages 16

[20] Cyclops Home Page. URL http://cyclops.cs.ubc.ca/. → pages 29

[21] Lyndon Clarke and Ian Glendinning and Rolf Hempel. The MPI Message Passing Interface Standard. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.6740. → pages 29

[22] Mersenne Twister Home Page. URL http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. → pages 29

[23] MCSANL. Message Passing Interface. URL http://www.mcs.anl.gov/research/projects/mpi/. → pages 30

[24] Satish Balay and Kris Buschelman and William D. Gropp and Dinesh Kaushik and Matthew G. Knepley and Lois Curfman McInnes and Barry F. Smith and Hong Zhang. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc. → pages 30

[25] Satish Balay and Kris Buschelman and Victor Eijkhout and William D. Gropp and Dinesh Kaushik and Matthew G. Knepley and Lois Curfman McInnes and Barry F. Smith and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008. → pages 30

[26] Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, September 2005. → pages 30

[27] J. E. Roman, E. Romero, and A. Tomas. SLEPc Users Manual. Technical Report DSIC-II/24/02 - Revision 3.1, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2010. → pages 30

[28] D. K. Tasoulis and N.G. Pavlidis and V. P. Plagianakos and M. N. Vrahatis. Parallel Differential Evolution. In *IEEE Congress on Evolutionary Computation (CEC)*, 2004. → pages 49