Parallel Sparse Factor Analysis for Student Modeling at Scale

Alireza Farasat, Alexander G. Nikolaev

Department of Industrial and Systems Engineering, University at Buffalo (SUNY)

December 9, 2016

Abstract

This paper enables student modeling analyses at scale, by presenting a parallel algorithm for performing SPARse Factor Analysis (SPARFA). SPARFA is accepted as a state-of-the-art machine learning approach for estimating student knowledge levels and predicting their performance in not-yet-taken educational tasks [12]. However, with the Likelihood Maximization (LM) formulation behind it resulting in a non-convex optimization problem with many local minima, the scalability has been a challenge for SPARFA. While the original method employs an alternating optimization approach to approximately solve the SPARFA LM problem, this paper employs a Parallel Coordinate Descent (PCD) algorithm parallelized in the shared memory setting. The performance of PCD is evaluated on varied problem instances with synthetic data and is found to successfully solve instance with up to 7.5 Million variables, which makes it suitable to make inference from the data of any real-world Massive Open Online Course. The experimental results illustrate the superiority of the PCD over the sequential SPARFA with up to 7x speedup particularly in more realistic situations, and suggest that shared memory systems are sufficient for handling any realistic student modeling tasks.

Keywords: parallel coordinate descent, parallel non-convex optimization, sparse matrix factorization, intelligent tutoring systems

1 Introduction

Educational systems have witnessed a substantial transition from traditional educational methods mainly using text books, lectures, etc. to newly developed systems which are artificial intelligent-based systems and personally tailored to the learners. The resulting popularity of online educational platforms with large numbers of students and tasks/problems to solve motivates the development of highly efficient algorithms to personalize the learning experiences [12]. Personalized Learning Systems (PLSs) and Intelligent Tutoring Systems (ITSs) are two areas of educational data mining that take on this challenge. Taking into account the individual progress of learners, PLSs customize the learning experience to the learners' abilities/needs [6]. In computerized learning environments, ITSs model and infer student's knowledge learning states [5, 16, 20]. For a while, Latent Factor

Modeling and Bayesian Knowledge Tracing have been the primary student modeling ITS tools [9]. These approaches encompass computational models, conceived in different scientific disciplines including cognitive and learning sciences, education, computational linguistics, artificial intelligence, operations research, and other fields [5, 12, 14, 16].

More recently, Lan *et al.* developed a new machine learning-based model that combines *learning analytics*, which approximates students knowledge of subject matter concepts underlying a domain, and *content analytics*, which estimates the relationships among a collection of questions and those concepts [12]. This model can be used to calculate the probability that a learner provides a correct response to a question in terms of three factors: their understanding of a set of underlying concepts, the concepts involved in each question, and each questions intrinsic difficulty [12]. Lan *et al.* proposed a bi-convex Likelihood Maximization approach for SPARse Factor Analysis (SPARFA), and exploited an alternating optimization approach to approximately solve the LM problem.

A SPARFA solver has to iterate between two sub-problems, until each sub-problem is optimally solved (due to the bi-convexity of the likelihood function). However, in order to run SPARFA analyses with large numbers of questions and students, i.e., to make it useful for MOOCs of realistic sizes, its scalability needs to be improved. To this end, this paper presents a parallel coordinate descent (PCD) algorithm that uses shared memory. The idea behind PCD is to parallelize the optimization task over computing resources (i.e., cores or processors) so that the cores work on different parts of the optimization problem (in terms of blocks of variables) simultaneously. At each iteration of PCD, a number of variables (i.e. from each block of variables) depending on the number of threads are selected randomly. Each thread individually works on a smaller set of variables and update them based on the gradient information.

This paper contributes to the educational data mining domain with the resulting algorithm that can handle a very large number of variables – up to 7.5 Million variables – in a given SPARFA instance. The computational experiments with synthetic data exhibit that PCD produces up to 7.4x speedup, with the highest speedups achieved with the largest instances.

The rest of this paper is organized as follows. Section 2 reviews the original SPARFA model and provides an overview of parallel optimization algorithms proposed for MF applications. Section 3 gives a detailed description of the new PCD algorithm. Section 4 reports experimental results on synthetic data of varied sizes. Section 5 concludes the paper and discusses future research directions.

2 Challenges of Sparse Factor Analysis at Scale

Matrix Factorization (MF) is a key component of machine learning-based systems that work, e.g., to recommend items to users, estimate missing data values, model gene expressions. More recently, MF has been found useful for predicting students' performance on given activities, e.g., problem solving. Indeed, with the advent of online learning platforms and their massive accessibility, much data are becoming available about students' knowledge states to assist educators in monitoring and accelerating learning. The developments in this paper address the scalability of MF performed for

large educational systems, where thousands of students in physical or virtual classes work on the same set of problems referred to as problem bank. This section overviews and outlines the computational challenges of SPARFA as a state-of-the-art machine learning framework for MF in student modeling. SPARFA utilizes an alternating optimization approach to tune the parameters of its model, however, this approach is computationally expensive, especially in large-scale problems. This section argues in favor of an alternative algorithm to use computational resources more effectively via efficient parallelization.

Student modeling is the cornerstone ITS task that reveals how the knowledge acquisition process can be decomposed as progress in multiple dimensions, typically referred to as knowledge components or skills [9,13]. In order to strategically attack the problem of assigning educational tasks to students in an personalized manner, one needs to both infer each student's skills and determine which sets of skills are required to handle the tasks not yet undertaken by the student. Factor analysis is a well accepted approach to student modeling based on the student performance in quizzes and homeworks [10,11]. SPARFA is a machine learning-based model that connects learning analytics and content analytics to students' performance. It infers the probability that a student correctly answers a question or solves a problem based on their understanding of a set of underlying course concepts, on the concepts involved in each question, and each question's intrinsic difficulty [12]. Given an incomplete matrix of students' performances, Maximum Likelihood Estimation (MLE) is performed in SPARFA to make the inference about students and questions, simultaneously.

Let $\mathbf{Y}$ denote a binary-valued data set of performance of N students on Q questions; hence, $\mathbf{Y}$ is a matrix of size $Q \times N$ with entry $Y_{ij} = 1(0)$ if student j has answered question i correctly. Matrix $\mathbf{Y}$ is typically very sparse. One way to estimate the missing values in $\mathbf{Y}$ is to factorize $\mathbf{Y}$ into matrices $\mathbf{W}, \mathbf{C}$ and $\mathbf{M}$ such that the function $\mathbf{WC+M}$ returns the estimates for the missing values in $\mathbf{Y}$. It is assumed that the collection of questions is related to a small number of abstract concepts represented by $\mathbf{W}$, where the weight W_{ik} ($\forall i = 1, \ldots, Q \; and \; k = 1, \ldots, K$) quantifies the degree to which question i involves concept k, with K being a total assumed number of such latent abstract concepts. Let C_{kj} ($\forall k = 1, \ldots, K \; and \; j = 1, \ldots, N$) denote student j's knowledge of concept k ($\mathbf{C}$ is the matrix version of C_{kj}). $\mathbf{M}$ is an $Q \times N$ matrix reflecting the intrinsic question difficulty. It is assumed that $K \ll Q, N$ so $\mathbf{W}$ becomes a tall, narrow $Q \times K$ matrix and $\mathbf{C}$ a short, wide $K \times N$ matrix. The model for the binary valued observations $Y_{ij} \in \{0, 1\}$ is then expressed as

$$Z_{ij} = \mathbf{w}_i^T \mathbf{c}_j + \mu_i \quad \forall \quad i = 1, \ldots, Q \;\; j = 1, \ldots, N,$$

$$Y_{ij} \sim Ber(\Phi(Z_{ij})),$$

where $\mathbf{w}_i$ and $\mathbf{c}_j$ denote the ith row and jth column of $\mathbf{W}$ and $\mathbf{C}$, respectively, μ_i is the difficulty level of question i, with $Ber(p)$ denoting a Bernoulli distribution with success probability p and $\Phi(x)$ defined as

$$\Phi(x) = \frac{1}{1 + e^{-x}}.$$

In order to estimate $\mathbf{W}$, $\mathbf{C}$ and μ, Likelihood Maximization (LM) of the observed data Ω_{obs} is

performed,

$$(P1): \max_{\mathbf{W},\mathbf{C}} \sum_{(i,j)\in\Omega_{obs}} log\big(p(Y_{ij}|\mathbf{w}_i,\mathbf{c}_j)\big)$$

$$s.t.$$

$$\|\mathbf{w}_i\|_0 \leq \delta \;\forall\; i \tag{1}$$

$$\|\mathbf{w}_i\|_2 \leq \beta \;\forall\; i \tag{2}$$

$$\|\mathbf{C}\|_F = \xi \tag{3}$$

$$W_{ik} \geq 0 \;\forall\; i,k, \tag{4}$$

where $p(Y_{ij}|\mathbf{w}_i,\mathbf{c}_j) = \Phi\big(\mathbf{w}_i^T\mathbf{c}_j\big)^{Y_{ij}}\big[1-\Phi\big(\mathbf{w}_i^T\mathbf{c}_j\big)\big]^{1-Y_{ij}}$. The Constraint Sets (1) through (4) guarantee that the solution to $(P1)$ satisfies the identifiability property of matrix factorization. Constraints (1) impose the sparsity restriction on matrix $\mathbf{W}$, $\|a\|_0$ counts the number of non-zero entries in vector a and $\|\mathbf{C}\|_F$ denotes the Frobenius norm. Since the optimization under Constraint Set (1) requires a combinatorial search, this set of constraints can be replaced by the following sets of constraints (l-1 norm):

$$\|\mathbf{w}_i\|_1 \leq \delta,\; \forall\, i.$$

The constrained optimization problem $(P1)$ can be transformed into an unconstrained optimization problem using Lagrange multipliers:

$$(P2): \min_{\mathbf{W},\mathbf{C},W_{ik}\geq 0\;\forall i,k} \sum_{(i,j)\in\Omega_{obs}} -log\big(p(Y_{ij}|\mathbf{w}_i,\mathbf{c}_j)\big) + \lambda\sum_i \|\mathbf{w}_i\|_1 + \frac{\mu}{2}\sum_i \|\mathbf{w}_i\|_2^2 + \frac{\gamma}{2}\|\mathbf{C}\|_F^2. \tag{5}$$

The resulting optimization problem $(P2)$ is biconvex in $\mathbf{W}$ and $\mathbf{C}$ [12]. *Lan et al.* presented SPARFA-M algorithm for solving this problem [12]. The idea of that algorithm is based on an alternating approach that splits the problem into two sub-problems. Each sub-problem is optimaly solved in a cyclic manner at each iteration. In general, the alternating approaches perform well in bi-convex optimization, however, when the number of questions, Q as well as the number of students, N increase, the sequential alternating approaches becomes slow, creating a challenge, and hence, an opportunity for exploring parallel implementations.

We now turn to the topic of algorithm parallelization. In many applications, and in particular in student modeling, parallelization pursues the objective of efficiently utilizing computational resources – CPU and memory [19]. In general, for large-scale machine learning problems, the classical optimization approaches including Newton, Interior-Point and other gradient-based methods become prohibitive due to high computational effort spent in gradient evaluation at each iteration. Several algorithm alternatives exist that manage to mitigate such efficiency challenges. One of these is Stochastic Gradient Descent (SGD) [24]. The idea behind SGD is to select a subset of problem variables at random, and estimate the gradient of the objective function with respect to the chosen

variables [22], and then, based on this (partial) estimate, update all the variables. Another viable alternative to the classical full gradient-based algorithms is Coordinate Descent (CD), where at each iteration, the objective function is minimized only in some variables, while all the others are kept constant [22]. In both cases, instead of solving a complex problem requiring much memory, a sequence of less computationally demanding problems is solved.

CD has been successfully employed to tackle a variety of large-scale machine learning problems including Support Vector Machines applications [7], Entropy Maximization [7], and Non-negative Matrix Factorization [2], among others. Different variations of CD are distinguished, with its two key components being the update rules and criteria of the selection of sequences in which variables are to be updated [23]. Randomized CD algorithms is a special case of SG methods where the estimation of a gradient is done with respect to only the randomly selected variables.

Although both SGD and CD are popular, they typically struggle handling large-scale data, available for analysis in recommendation systems and intelligent tutoring systems research. However, the inherent sequential and iterative processes allow for exploiting such computational resources as GPU, multi-core or many-core CPUs. In fact, several parallel SGD and CD approaches have been proposed, although mainly for solving convex, smooth optimization problems [1, 21].

Among the existing parallel-SGD methods, particularly among those used for matrix factorization, some are designed for shared-memory systems where all the variables stored in memory are global, i.e., accessible by all the cores [24]. For instance, Mini-Batching enables parallelization across samples: here, SGD updates are done by separately working with samples of the data – averaging the gradients obtained for all the samples [3]. Hogwild [17] describes an asynchronous, nowadays popular method for parallelizing SGD, which utilizes the "data pass" approach by taking a sample from data and performing an update to the global model without any inter-core communication (for details, see [21]. However, CD-based algorithms have the advantage over SGD-based ones in that the objective function is guaranteed to decrease at every iteration [22]. Moreover, in CD-based algorithms, as convergence occurs, the gradient shrinks to zero, while SGD algorithms obtain non-zero gradient estimates even at the optimum [22].

Parallel CD algorithms versions vary depending on the implementation and application-at-hand; they may be made to work in a synchronous or asynchronous manner [24]. Synchronous algorithms divide the computing tasks/operations into smaller subsets that can be executed in parallel on a multi-core machine; however, the processors need to be synchronized frequently across all the cores, to guarantee the consistency of task (re)assignment in each iteration. Asynchronous implementations, typically preferred in practice, relax this requirement. Convergence analysis of asynchronous methods is more complicated than that of synchronous methods [4], but once properly tuned up, these methods excel in real-world big data analysis applications [8, 15, 18, 22]. In this paper, an asynchronous parallel coordinate descent algorithm using shared memory is presented that can be used to accelerate nontrivial matrix factorization problems in general and SPARFA in application to student modeling in particular.

3 The Parallel Coordinate Descent Algorithm for SPARFA

The dimensionality of the students' performance prediction problem increases quickly with the number of students and questions in online courses. However, sequential algorithms become slow and stall even for trivial non-constrained convex optimization problems. To scale SPARFA to larger problems, a parallel version of Coordinate Descent algorithm is implemented. Since many variants of CD are possible, selecting the variant of the sequential CD is the first step in designing PCD. As discussed in detail in [22], the problem variables to be updated can be selected in a cyclic manner, or alternatively, they can be selected randomly at each iteration.

Certain designs of CD require the problem's objective function to satisfy such proprieties as smoothness and convexity. The optimization problem for SPARFA, introduced in (5), is both non-convex and non-smooth (because the regularization function $\|\mathbf{w}_i\|_1 \; \forall i = \{1, \ldots, N\}$ is non-smooth), and its objective function is block-separable. In the sequential CD, at each iteration t, a subproblem is created by making a linear approximation to (5) along the selected coordinate direction at each iteration. In this implementation, two coordinates, w_{ik} and c_{kj}, are selected at each iteration randomly, and then, the corresponding subproblem is solved. Algorithm 1 details the randomized coordinate descent algorithm for SPARFA (it is a sequential algorithm).

Algorithm 1 Coordinate Descent Algorithm

Input: Matrix $\mathbf{Y}_{Q\times N}$, K (number of hidden variables, $\lambda, \mu, \gamma, \beta$.
Output: Matrices $\mathbf{W}^*_{Q\times K}$ and $\mathbf{C}^*_{K\times N}$.

1. Initialize randomly $\mathbf{W}$ and $\mathbf{C}$.
2. **while** (stopping criteria) **do**
3. randomly select $i \in \{1, \ldots, Q\}$, $j \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, K\}$
4. calculate $\nabla F_{W_{ik}}$ and $\nabla F_{C_{kj}}$
5. update $W_{ik}^{t+1} \leftarrow \max\{0, W_{ik}^{t+1} - \beta \nabla F_{W_{ik}}\}$
6. update $C_{kj}^{t+1} \leftarrow \frac{1}{1+\beta\gamma}\left(C_{kj}^{-} \beta \nabla F_{C_{kj}}\right)$
7. calculate Objective Function using (5)
8. update $\beta \leftarrow \frac{t}{t+2}$
9. $t \leftarrow t + 1$
10. **end while**

Selection of step sizes is a crucial element of the algorithm's design that affects the convergence rate; the step size selection rules may or may not be problem-specific. One approach is to select a large value for the step-size (close to one) and adaptively shrink it at each iteration.

3.1 Variable Selection Strategy

In Parallel Coordinate Descent algorithm, P coordinates are selected randomly and independently from each other, to be updated in parallel, with P set equal to the number of available threads or cores. The simplest version of PCD in the one where each coordinate, W_{ik}, $i \in \{1, \ldots, N\}$, $k \in \{1, \ldots, K\}$ has an equal chance of being selected, independently from the selection done at previous iterations – one can think of this strategy as a sampling with replacement. For convex objective functions, one can prove the algorithm's convergence under the random selection strategy. However, a better approach – the one used in this paper – is to do sampling without replacement. This scheme provides an opportunity for all the coordinates (decision variables) to be updated in sync. Though the sampling without replacement is computationally more expensive, it achieves a desirable balances between the exploration (of the solution space) and exploitation (of the computational resources). Let Ω_t denote a set of variables that have not been selected up till iteration t, with $\|\Omega_t\|$ denoting the cardinality of set Ω_t. The probability of selecting a variable is then given by $\frac{1}{\|\Omega_t\|}$. Once Ω_t becomes empty, it is re-populated again with all the coordinates.

3.2 Variable Updating Strategy

At each iteration of PCD, p pairs of variables selected from $\mathbf{W}$ and $\mathbf{C}$ are being updated. The update for variable W_{ik}, $i \in \{1, \ldots, N\}$, $k \in \{1, \ldots, K\}$, is executed as follows:

$$W_{ik}^{t+1} = \max\{0, W_{ik}^{t+1} - \beta \nabla F_{W_{ik}}\}, \tag{6}$$

where $\nabla F_{W_{ik}}$ is the gradient of the objective function in $(P2)$ with respect to variable W_{ik},

$$\nabla F_{W_{ik}} = -\sum_{j=1}^{N} C_{kj}^{t} \Big(Y_{ij} - \frac{1}{1 + e^{-\sum_{k=1}^{K} W_{ik}^{t} C_{kj}^{t}}} \Big) + \mu \frac{W_{ik}^{t}}{\|\mathbf{W}\|_F}.$$

Equation (6) guarantees that W_{ik} remains non-negative after every update. This is a projection operator that helps to deal with non-smoothness of ℓ_1-regularization. Similarly, the gradient of the likelihood function with respect to C_{kj}, $\nabla F_{C_{kj}}$, is expressed as

$$\nabla F_{C_{kj}} = -\sum_{i=1}^{Q} W_{ik}^{t} \Big(Y_{ij} - \frac{1}{1 + e^{-\sum_{k=1}^{K} W_{ik}^{t} C_{kj}^{t}}} \Big) + \gamma \frac{C_{ik}^{t}}{\|\mathbf{C}\|_F}. \tag{7}$$

The following projection operator enables a faster convergence and limits the growth of C_{kj},

$$C_{kj}^{t+1} = \frac{1}{1 + \beta\gamma} \Big(C_{kj}^{-} \beta \nabla F_{C_{kj}} \Big). \tag{8}$$

The PCD algorithm is summarized in the form of a pseudo code in Algorithm 2.

Algorithm 2 Parallel Coordinate Descent Algorithm

Input: Matrix $\mathbf{Y}_{Q\times N}$, K (number of hidden variables, P (the number of processes), $\lambda, \mu, \gamma, \beta$.
Output: Matrices $\mathbf{W}^*_{Q\times K}$ and $\mathbf{C}^*_{K\times N}$.

1. Initialize randomly $\mathbf{W}$ and $\mathbf{C}$.
2. Calculate Objective Function using (5)
3. **while** (stopping criteria) **do**
4. **In Parallel** on P processes
5. randomly select $i \in \{1, \ldots, Q\}$, $j \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, K\}$ with probability $\frac{1}{\|\Omega_t\|}$
6. **In Parallel** on P processes
7. calculate $\nabla F_{W_{ik}}$ and $\nabla F_{C_{kj}}$
8. update $W_{ik}^{t+1} \leftarrow \max\{0, W_{ik}^{t+1} - \beta \nabla F_{W_{ik}}\}$
9. update $C_{kj}^{t+1} \leftarrow \frac{1}{1+\beta\gamma}\left(C_{kj}^{-} \beta \nabla F_{C_{kj}}\right)$
10. **In Parallel** on P processes
11. Calculate Objective Function using (5)
12. update $\beta \leftarrow \frac{t}{t+2}$
13. update $\|\Omega_t\|$
14. $t \leftarrow t + 1$
15. **end while**

4 Computational Results

This section explores the performance of PCD and CD on small-, medium- and large-sized instances of problem ($P2$). In order to evaluate the performance of PCD, the presented algorithm is tested on 12 synthetically generated data sets. The synthetic data generation for student modeling is not straightforward, in general, since one needs to come up with a realistic model incorporating the differences between the students' abilities and questions' difficulty levels. In this paper, however, it is only natural to use the original SPARFA model, because the object of our analysis is algorithm scalability. Consistent with the SPARFA model assumptions, $\mathbf{W}$ is taken as a non-negative sparse matrix and $\mathbf{C}$ as an arbitrary matrix with bounded elements. The probability that student j answers question i correctly is computed per formula (2). The difficulty of questions is assumed to be normally distributed with mean μ_D and σ_D.

4.1 Experimental Setup

In order to evaluate the performance of the PCD algorithm, multiple synthetic data sets of different sizes were generated (the corresponding optimization problems have $(Q+N)\times K$ decision variables). The resulting problem instances are categorized into three classes: (1) Small-sized problems, (2) Medium-sized problems, and (3) Large-sized problems. This distinction is due to the fact that the PCD performance in terms of the delivered speedup and convergence changes with an increasing problem size. The largest problem instance generated featured 25,000 students and 600,000 questions (its complexity is equal to that of any other instance with the same number of variables, e.g., one with 600,000 students and 25,000 questions). This instance is large enough to test the scalability of the presented Parallelized SPARFA to any realistic setting; indeed, no MOOC generates as much data. Table 1 overviews the designed experimental setup with the 12 data sets.

Table 1: Experiments setup for testing Parallel Coordinate Descent algorithm.

Category	Experiment ID	Q	N	K	Variables	CD Iterations	Num of Run	Time Limit (s)
Small	1	5	4	2	18	2000	30	0.1
	2	50	20	4	280	2000	30	0.5
	3	500	200	6	4,200	2000	30	5
	4	1,000	300	6	7,800	2000	30	20
Medium	5	5,000	1,000	8	48,000	1500	25	50
	6	20,000	5,000	8	200,000	1200	25	100
	7	30,000	8,000	9	342,000	1000	25	150
	8	50,000	10,000	9	540,000	500	25	200
Large	9	100,000	12,000	10	1,120,000	500	20	500
	10	250,000	15,000	10	2,650,000	500	20	1000
	11	500,000	20,000	10	5,200,000	200	20	1500
	12	600,000	25,000	12	7,500,000	200	20	2000

4.2 Implementation

Some notes regarding the implementation of the PCD are summarized next.

Software: We used a custom code written in C++ with OpenMP, and compiled it with the Intel® C++ Compiler 16.0 with -o3 optimization and -std=c++11. Double precision values were used for matrices $\mathbf{W}$ and $\mathbf{C}$. The observation matrix $\mathbf{Y}$ is binary and highly sparse, so the compressed column and row representations were employed.

Hardware: Different hardware was used for difference SPRAFA problem instances. For experiments 1 through 4, we used a four-sockect Intel(R) Xeon(R) CPU E7- 4830 @ 2.13GHz with 8 cores per socket and 256 GB memory. For experiments 5 through 10, Dell machines with 12x2.40GHz Intel Xeon E5645 Processor Cores and 12 cores per node with 48 GB memory and Linux (RedHat Enterprise Linux 6.1 2.6.32 Kernel) operating systems were used. For the rest of the experiments, we used High Memory Compute (AMD CPUs) with 2.20GHz Processors, 32 cores per node (8 nodes), and 256 GB RAM.

Datasets: The PCD algorithm is run on synthetic data. To generate synthetic data, we generate sparse matrix $\mathbf{W}$ and non-sparse matrix $\mathbf{C}$. Based on the model introduced in Section 2, the observation matrix $\mathbf{Y}$ is generated. Since the observation matrix is highly sparse, a Markov process (namely, random walk) was employed to remove many of its elements to achieve the sparsity of $\mathbf{Y}$. Taking $\mathbf{Y}$ as the data, $\mathbf{W}^*_{Q\times K}$ and $\mathbf{C}^*_{K\times N}$ are then inferred by the PCD algorithm.

Reporting: The main challenge in dealing with high dimension non-convex optimization problems is a large number of local optima, causing the majority of iteration-based algorithms to get stuck without converging to a global optimum. Another issue with such large-scale optimization problems is that the true optimum is usually unknown in real-world applications. Consequently, defining a stopping criterion is problematic. In this paper, we consider two stopping criteria, one based on runtime and the other based on the number of executed iterations. According to the first criterion, the PCD algorithm traverses the solution space till a pre-set given time, at which point the current best solution is returned and its objective function is reported; this objective function, defined in (5), is the negative log-likelihood function that includes regularization terms. According to the second criterion, the PCD executes a pre-selected fixed number of iterations and then returns the best found solution. To ensure that the randomness in the initial conditions does not have a significant effect on the solution quality, each experiment is run several times as reported in Table 1. In addition, the time from the algorithm's initialization till its termination is noted and reported. The speedup (S_p) and efficiency (E_p) are also reported for each experiment. The speedup is defined

$$S_p = \frac{T_s}{T_p}, \tag{9}$$

where T_s is the execution time of the sequential algorithm and T_p is the execution time of the parallel algorithm with p processes. Note that the ideal (best possible) speedup is $S_p = p$, referred to as a linear speedup. The efficiency is defined as

$$E_p = \frac{S_p}{p}. \tag{10}$$

4.3 Results for Small-sized Problem Instances

The first set of the experiments had four instances with 18, 280, 4,200 and 7,800 variables, respectively. The PCD performance as a function of the number of cores was evaluated under both stopping criteria. Figure 1 illustrates how the PCD algorithm convergence times grow as the number of coordinates (i.e., cores) increases. It shows that when more coordinates are updated at once, the

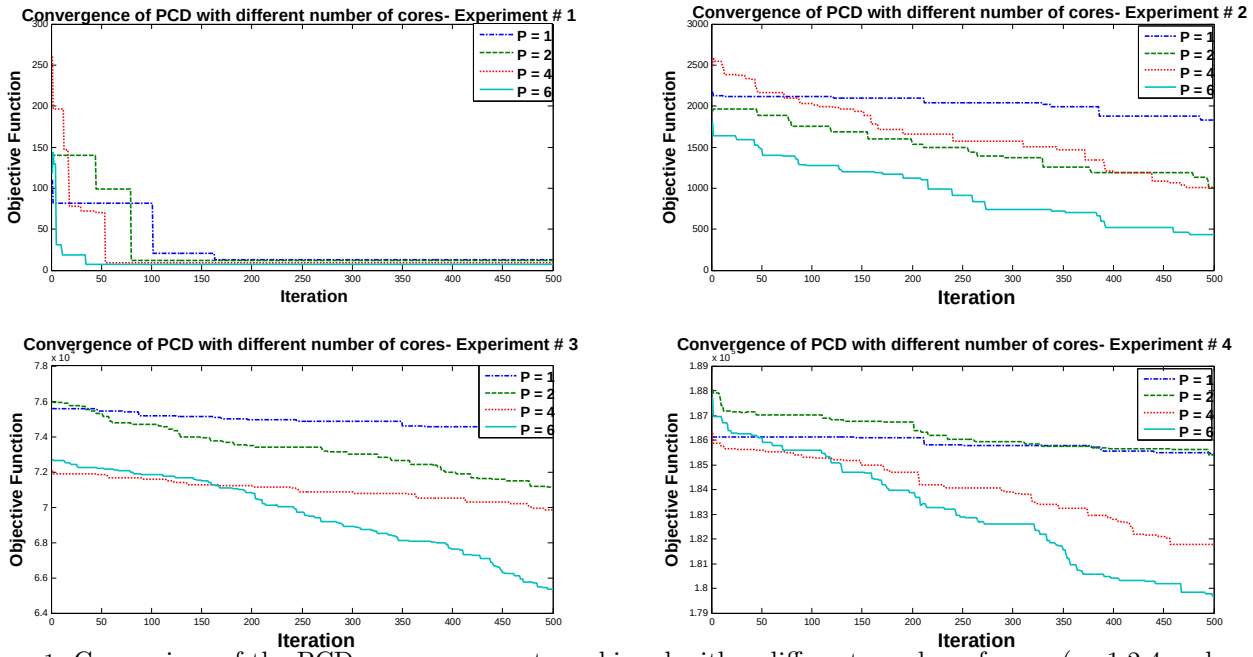

Figure 1: Comparison of the PCD convergence rates achieved with a different number of cores (p=1,2,4 and 6). Using more cores results in a faster convergence.

algorithm converges faster, particularly as the problem size increases. Figure 2 compares the averages of the objective function obtained from 30 runs for each small-sized problem instance. The results show that by increasing the number of cores (i.e. threads generated in OpenMP), the performance of PCD improves, especially if a time limit (e.g. two minutes) is considered as the stopping criterion. To explore how the addition of computational resources affects the run time, particularly when the algorithm is run with a fixed number of iterations, the speedup values are reported in Figure (3). For the small-sizes problem instances, the overhead cost of inter-core communication turns out to be high to reap the substantial benefits from running the CD algorithm in parallel. This phenomenon is particularly true for synchronous CD because of an unbalanced load distribution over the cores. Table 2 summarizes the speedup and efficiency values obtained by running the experiments on a different number of cores. Note that the value of T_p, used to calculate S_p, is taken as the average over 30 runs.

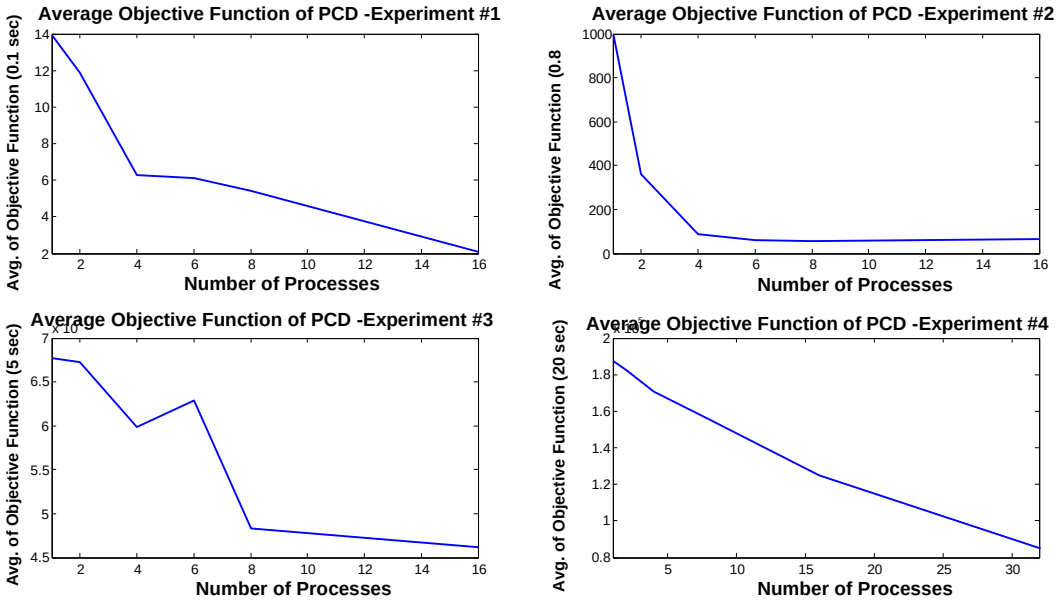

Figure 2: The averages of the objective function for the small-sized instances over 30 runs.

Table 2: Speedup and efficiency for instances 1 through 4 (NR: Not Reported).

Number of Cores	Experiment 1		Experiment 2		Experiment 3		Experiment 4	
	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$
1	1	100	1	100	1	100	1	100
2	0.242	12.1	0.548	27.4	1.711	85.6	1.770	88.5
4	0.086	2.2	0.307	7.7	1.984	49.6	1.778	44.5
6	0.052	0.9	0.196	3.3	2.670	44.5	1.880	31.3
8	0.043	0.5	0.150	1.9	1.076	13.4	2.839	35.5
12	NR	NR	NR	NR	1.145	9.5	1.613	13.4

4.4 Results for Medium-sized Problem Instances

The medium-sized experiments include four instances with 48,000 to 540,000 variables. The performance of PCD algorithm was tested on these instances with a different number of cores. Figure 4 depicts the averages of the objective function over 25 runs for four instances with random initialization. Adding more cores results in improving the average of the objective function. As discussed before, the speedup defined in (9) indicates the impact of parallelization on runtimes. The speedup averages for the medium-sized instances are compared with respect to a linear speedup (the ideal speedup) in Figure 5. The speedup values obtained for medium-sized problem instances

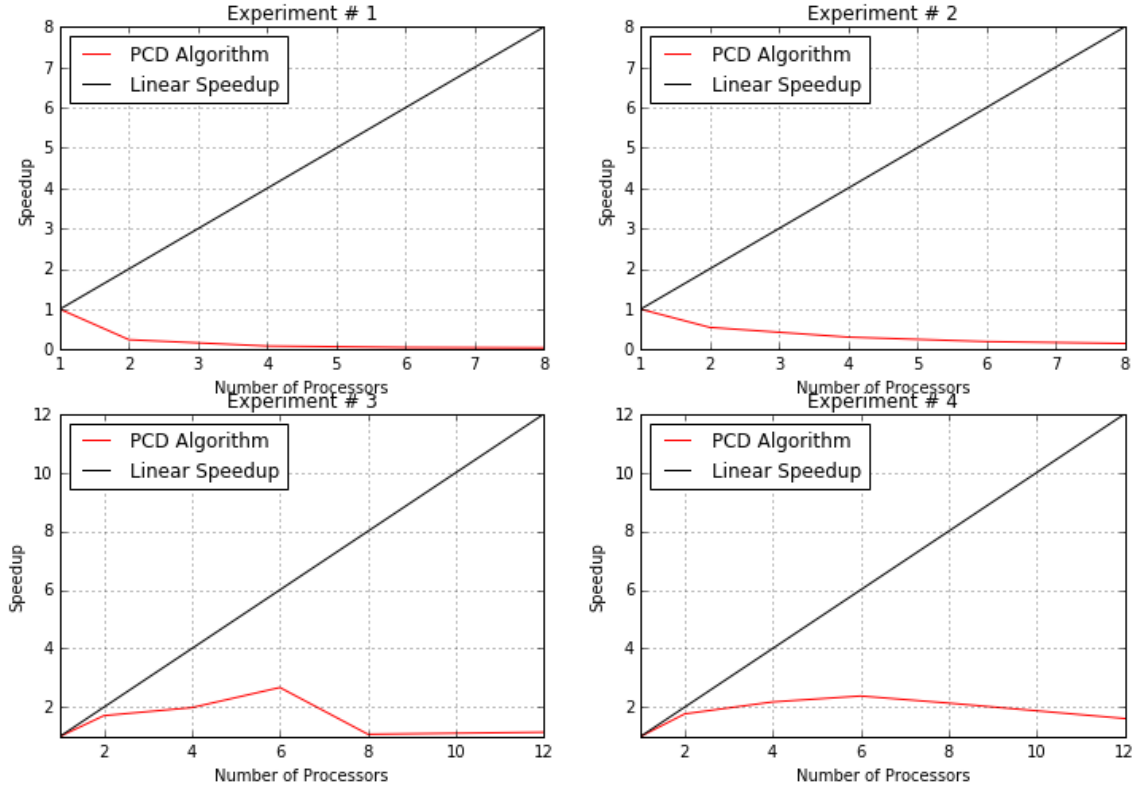

Figure 3: The averages of speedup for the small-sized instances.

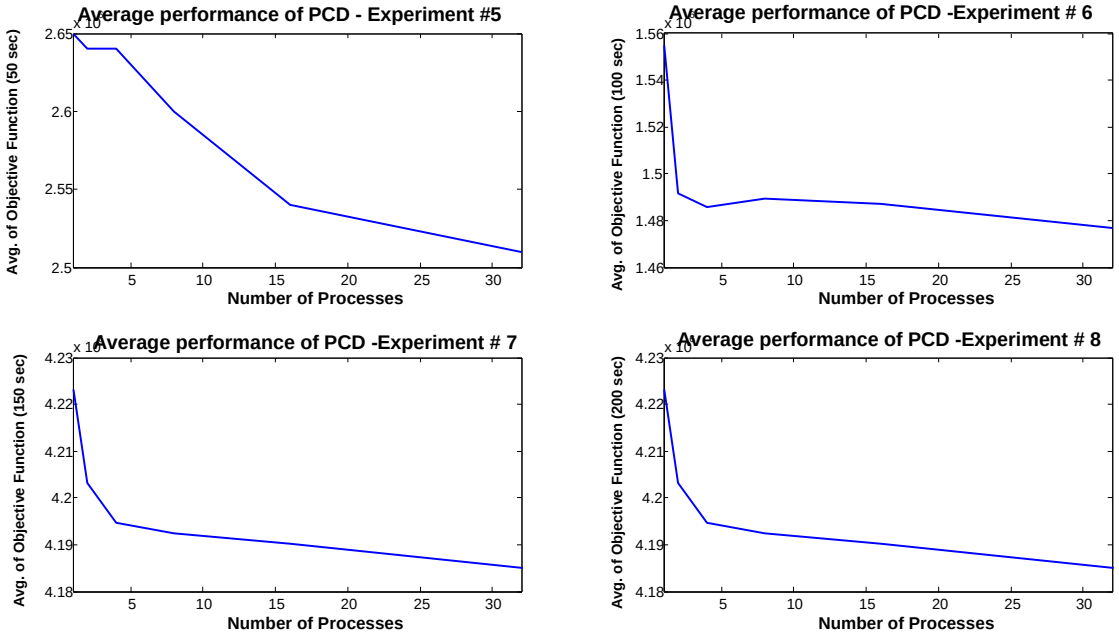

Figure 4: The averages of the objective function for the medium-sized inctances improve over 25 runs.

lie below the line for the linear speedup. However, the speedup improves with a growing problem

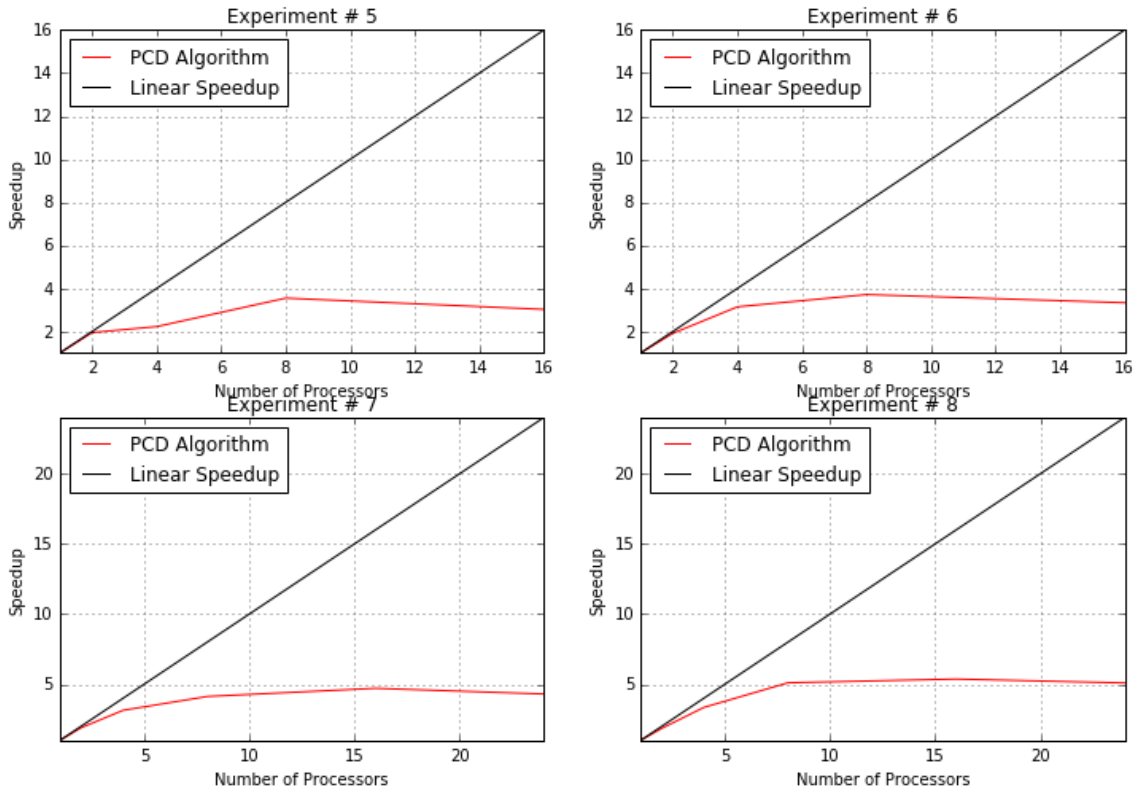

Figure 5: The averages of speedup for the medium-sized instances.

size. For instances 5 and 6, the maximum speedup is reached with eight cores while using 16 cores gives the maximum speedup in instances 7 and 8. Table 3 reports the speedup and efficiency values from running the PCD algorithm on instances 5 to 8 with a different number of cores. In the larger instances, adding more computational resources (i.e. cores) improves the speedup.

Table 3: Speedup and Efficiency for experiments 5 through 8 (NR: Not Reported).

Number of Cores	Experiment 5		Experiment 6		Experiment 7		Experiment 8	
	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$
1	1	100	1	100	1	100	1	100
2	1.947	97.4	1.913	95.7	1.894	94.7	1.854	92.7
4	2.223	55.6	3.142	78.6	3.143	78.6	3.348	83.7
8	3.546	44.3	3.712	46.4	4.128	51.6	5.103	63.8
16	3.023	18.9	3.333	20.8	4.711	29.4	5.370	33.6
24	NR	NR	NR	NR	4.305	17.9	5.092	21.2

4.5 Results for Large-sized Problem Instances

The large-sized experiments include the instances with more than one million variables. Figure 6 illustrates the averages of the SPARFA objective function with a different number of processes (cores) used exploited for the gradient-driven updates. Again, adding more cores improves the

objective function value, on average, up to a certain threshold. Figure 7 shows the speedup values

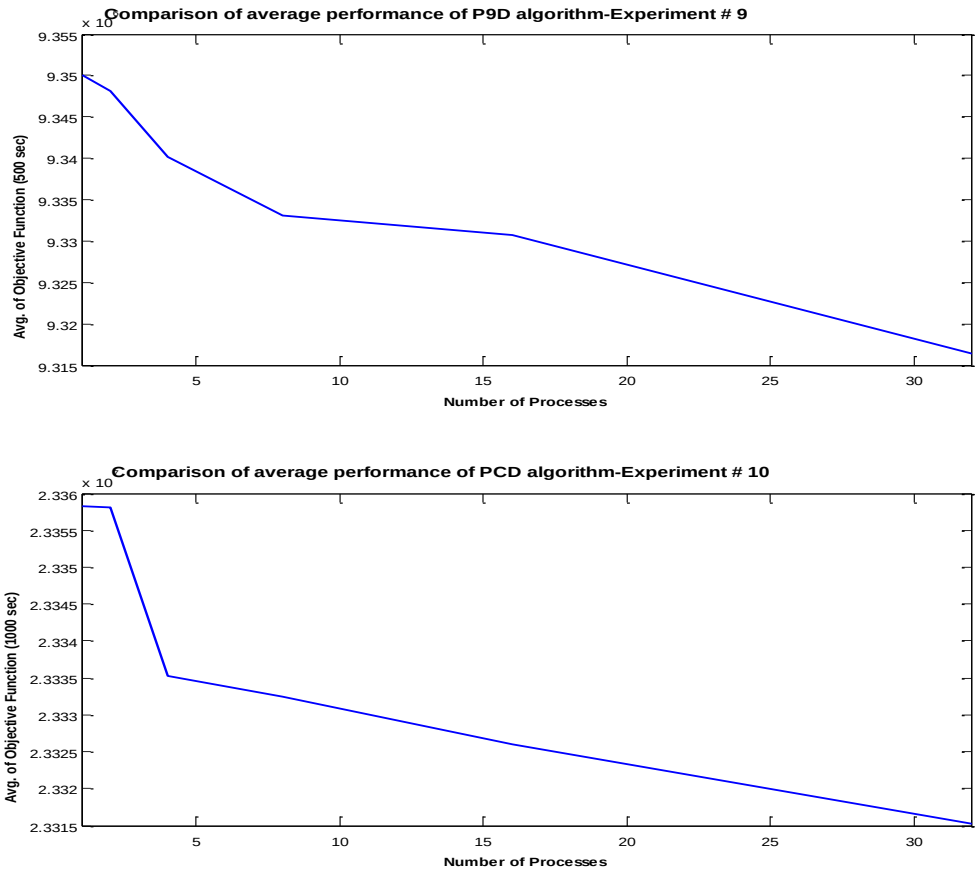

Figure 6: The averages of objective function for the large-sized instances over 25 runs.

and the level of parallelism that PCD achieves over large-sized problem instances. In this set of experiments, the achieved speedup is greater than 7x. This level of speedup indicates that running the PCD algorithm on a large optimization problem, which would otherwise take a week to solve, will take just one day. Table 4 presents the achieved speedup and efficiency values for the large-sized instances. Running the PCD algorithm on eight cores results in more than 70% efficiency. Note that any larger efficiency in a shared memory system is unlikely, due to the overhead computing costs, unless one uses all cache levels (i.e., L1, L2, L3, etc.) in a very efficient way.

An additional study is performed to establish the value of the parallel implementation of SPARFA over its non-parallel counterpart. To this end, consider a practical scenario where SPARFA analysis is to be performed regularly, within a limited time window. In a real-world setting, one can assume that the inference results are to be updated overnight as students take on more questions, and hence, more data become available. Then, any SPARFA run or runs should not take more than eight hours. Indeed, taking into account the fact that the model contains a non-convex optimization component, it is highly recommended to run the optimization algorithm multiple times, with randomly selected values of $\mathbf{W}$ and $\mathbf{C}$. To conduct an analysis for this case, 15 experiments including 5, 10 and 20 sets of 72-, 36- and 18-minute runs, respectively, were performed, with each set taking exactly 8

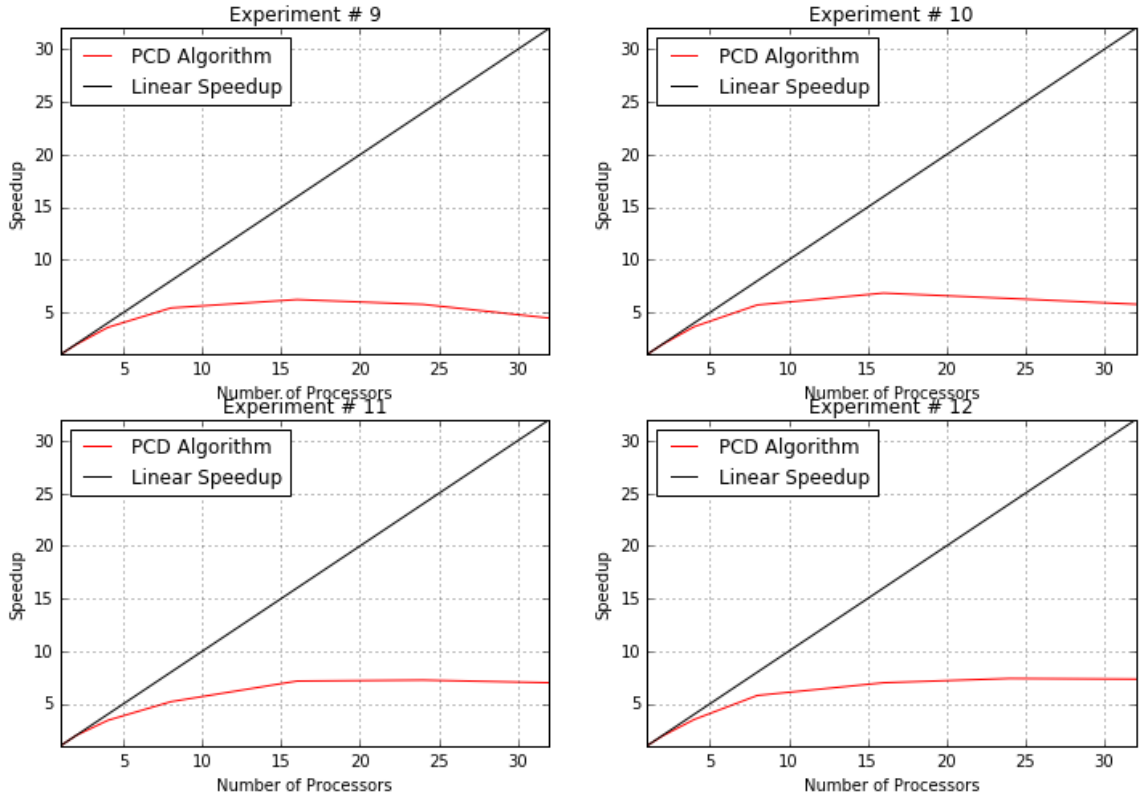

Figure 7: The average speedup of large-size problems.

Table 4: Speedup and efficiency of experiments 9 to 12.

Number of Cores	Experiment 9		Experiment 10		Experiment 11		Experiment 12	
	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$	S_p	$E_p(\%)$
1	1	100	1	100	1	100	1	100
2	1.951	97.6	1.973	98.6	1.992	99.6	1.947	97.4
4	3.578	89.4	3.645	91.1	3.421	85.5	3.506	87.7
8	5.424	67.8	5.710	71.4	5.194	64.9	5.787	72.3
16	6.221	38.9	6.838	42.7	7.155	44.7	6.995	43.7
24	5.763	24.0	6.326	26.4	7.247	30.2	7.403	30.8
32	4.462	13.9	5.779	18.1	6.998	21.9	7.343	22.9

hours in total (e.g., 20×18 minutes $= 8$ hours), using 1, 4, 8, 16 and 32 cores.

Figure 8 shows the results for both the sequential and parallel CD algorithms on the largest problems instance. The sequential SPARFA algorithm (CD with one core) is compared against its parallel versions utilizing 4, 8, 16 and 32 cores. Because in any run, the algorithm might get stuck in a local minimum, it was run with multiple restarts performed within an 8-hour time slot, and the average and the best obtained objective function value (calculated using (5)) was reported. The results show that the parallel SPARFA runs return significantly better objective function values. This observation indicates that parallelizing SPARFA is a worthy undertaking.

In addition, the performance of the parallel SPARFA (with 32 cores) is compared against its

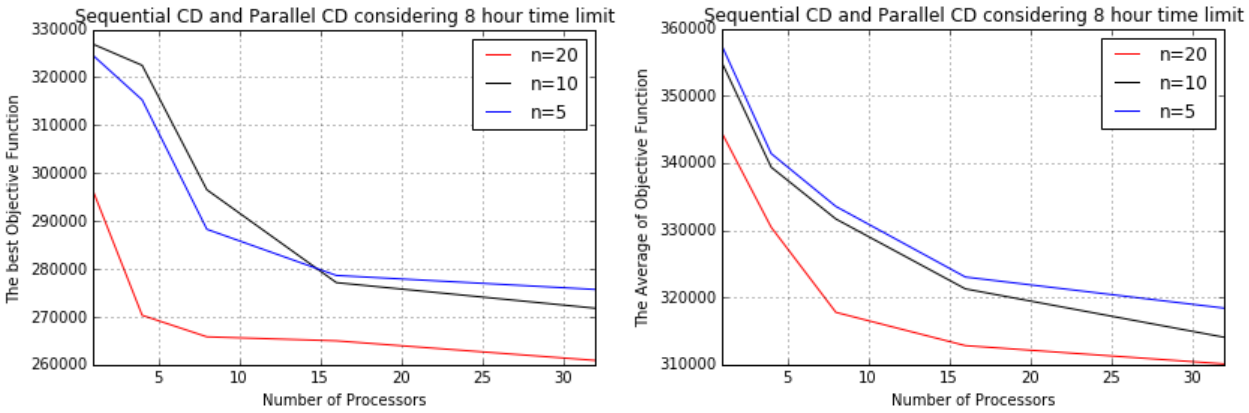

Figure 8: The best objective function (left) and average of objective function (right) obtained in an 8-hour run with 5, 10 and 20 random initialization utilizing 1,4, 8, 16 and 32 cores.

sequential counterpart, iteration by iteration. Since the performance of both sequential and parallel algorithms rely upon starting points, this analysis tracks the improvement achieved during the same number of iterations in the runs with the same initial solutions. In this experiment, each algorithm is given 18 minutes (which amounts to 34 iterations per run) to optimize the objective function of the LM problem, for ten randomly selected initial solutions. This setting was selected because multiple 18-minute runs produced the best overall inference results over an 8-hour time period in the previous set of experiments. Figure 9 depicts the observed objective function value dynamics over time. Indeed, it confirms that the parallel SPARFA consistently achieves lower objective function values, offering a 5-10% improvement; this improvement is substantial even with multiple algorithm runs. Figure 9 (right) details the improvement observed in LM optimization achieved with the best initial solution.

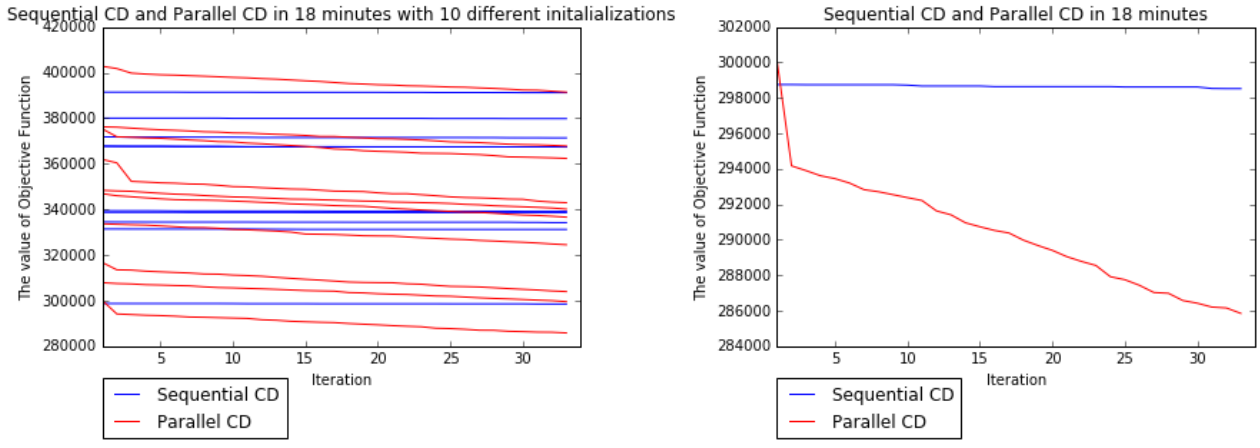

Figure 9: Parallel and Sequential SPARFA in 10 runs with randomly initialized $\mathbf{C}$ and $\mathbf{W}$ (left) Comparing parallel and sequential SPARFA with the best initial candidate.

5 Conclusion

This paper presents a parallel algorithm for SPARFA – a machine learning based approach to student modeling that allows for predicting students' performance on a set of questions they have not yet taken. To this end, a parameteric model is fit to the available data of the observed student question-answering outcomes using MLE. This MLE task results in a non-convex, non-smooth optimization problem, which the originally proposed algorithm solves using an alternating gradient approach, with two sub-problems solved iteratively till the convergence of both.

This paper attacked the scalability challenge in SPARFA, and offered a parallel version of CD algorithm to solve it; the algorithm runs seven times faster, on average, than the sequential approach, with higher speedups observed on larger problem instances. At each iteration, the algorithm selects p variables without replacement to be updated in parallel in a shared memory setting. The performance of the PCD algorithm was tested on several synthetic data sets, with the largest one featuring hundreds of thousands of students and questions, thus being representative of any realistically large MOOC.

The experimental results confirm that the parallelization significantly increases the efficiency of use of computational resources (i.e., more than 70% efficiency with running the PCD algorithms on eight cores in parallel). Note that due to inter-core communication, adding more computational resources beyond some threshold does not lead to further speedup or efficiency improvements; the threshold depends on the problem size.

References

[1] Joseph K Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin, *Parallel coordinate descent for l1-regularized loss minimization*, arXiv preprint arXiv:1105.5379 (2011).

[2] Andrzej Cichocki and PHAN Anh-Huy, *Fast local algorithms for large scale nonnegative matrix and tensor factorizations*, IEICE transactions on fundamentals of electronics, communications and computer sciences **92** (2009), no. 3, 708–721.

[3] Andrew Cotter, Ohad Shamir, Nati Srebro, and Karthik Sridharan, *Better mini-batch algorithms via accelerated gradient methods*, Advances in neural information processing systems, 2011, pp. 1647–1655.

[4] Brent Edmunds, Zhimin Peng, and Wotao Yin, *Tmac: A toolbox of modern async-parallel, coordinate, splitting, and stochastic methods*, arXiv preprint arXiv:1606.04551 (2016).

[5] Arthur C Graesser, Mark W Conley, and Andrew Olney, *Intelligent tutoring systems.*, (2012).

[6] Sabine Graf et al., *Personalized learning systems*, Encyclopedia of the Sciences of Learning, Springer, 2012, pp. 2594–2596.

[7] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan, *A dual coordinate descent method for large-scale linear svm*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 408–415.

[8] Cho-Jui Hsieh and Inderjit S Dhillon, *Fast coordinate descent methods with variable selection for non-negative matrix factorization*, Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2011, pp. 1064–1072.

[9] M Khajah, Rowan M Wing, Robert V Lindsey, and Michael C Mozer, *Integrating latent-factor and knowledge-tracing models to predict individual differences in learning*, Proceedings of the Seventh International Conference on Educational Data Mining, 2014.

[10] Andrew S Lan, Christoph Studer, and Richard G Baraniuk, *Quantized matrix completion for personalized learning*, arXiv preprint arXiv:1412.5968 (2014).

[11] Andrew S Lan, Christoph Studer, Andrew E Waters, and Richard G Baraniuk, *Tag-aware ordinal sparse factor analysis for learning and content analytics*, arXiv preprint arXiv:1412.5967 (2014).

[12] Andrew S Lan, Andrew E Waters, Christoph Studer, and Richard G Baraniuk, *Sparse factor analysis for learning and content analytics*, The Journal of Machine Learning Research **15** (2014), no. 1, 1959–2008.

[13] Nan Li, William Cohen, Kenneth R Koedinger, and Noboru Matsuda, *A machine learning approach for automatic student model discovery*, Educational Data Mining 2011, 2010.

[14] Nan Li, William W Cohen, Kenneth R Koedinger, Noboru Matsuda, and Carnegie Mellon, *A machine learning approach for automatic student model discovery.*, EDM, 2011, pp. 31–40.

[15] Ion Necoara and Dragos Clipici, *Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed mpc*, Journal of Process Control **23** (2013), no. 3, 243–253.

[16] Anna N Rafferty, Emma Brunskill, Thomas L Griffiths, and Patrick Shafto, *Faster teaching by pomdp planning*, Artificial intelligence in education, Springer, 2011, pp. 280–287.

[17] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu, *Hogwild: A lock-free approach to parallelizing stochastic gradient descent*, Advances in Neural Information Processing Systems, 2011, pp. 693–701.

[18] Peter Richtárik and Martin Takáč, *Parallel coordinate descent methods for big data optimization*, Mathematical Programming (2012), 1–52.

[19] ———, *Parallel coordinate descent methods for big data optimization*, Mathematical Programming **156** (2016), no. 1-2, 433–484.

[20] Shaghayegh Sahebi, Yun Huang, and Peter Brusilovsky, *Predicting student performance in solving parameterized exercises*, Intelligent Tutoring Systems, Springer, 2014, pp. 496–503.

[21] Scott Sallinen, Nadathur Satish, Mikhail Smelyanskiy, Samantika S Sury, and Christopher Ré, *High performance parallel stochastic gradient descent in shared memory.*

[22] Stephen J Wright, *Coordinate descent algorithms*, Mathematical Programming **151** (2015), no. 1, 3–34.

[23] Hsiang-Fu Yu, Cho-Jui Hsieh, I Dhillon, et al., *Scalable coordinate descent approaches to parallel matrix factorization for recommender systems*, Data Mining (ICDM), 2012 IEEE 12th International Conference on, IEEE, 2012, pp. 765–774.

[24] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin, *A fast parallel sgd for matrix factorization in shared memory systems*, Proceedings of the 7th ACM conference on Recommender systems, ACM, 2013, pp. 249–256.