Parallelizing Bayesian Knowledge Tracing Tool For Large-scale Online Learning Analytics

Yanjun Pu
State Key Laboratory of Software Development Environment
Beihang University
Beijing, China
puyanjun@nlsde.buaa.edu.cn

Wenjun Wu, Yong Han, Dengbo Chen
State Key Laboratory of Software Development Environment
Beihang University
Beijing, China
{wwj,hanyong,cdb}@nlsde.buaa.edu.cn

Abstract-With the advent of Massive Online Open Courses (MOOCs), the data scale of student learning behavior and knowledge mastery has significantly increased. In order to effectively and efficiently analyze these datasets and present on-the-fly intelligent tutoring to online learners, it is necessary to improve existing learning analytics tools in a parallel and automatic way. One of the most common tools is Bayesian Knowledge Tracing (BKT) that can model temporal progress of online learners and evaluate their mastery of course knowledge. Current implementation of BKT is mostly based on single machine, which leads to slow execution performance during its Expectation Maximization(EM) algorithm for parameter fitting. Although there are a few parallel implementations for EM algorithm, they don't support automatic initial BKT parameter tuning to ensure the correct convergence of the EM iteration. Therefore, this paper presents a new parallel BKT open source tool based on the Spark computational framework with the method of automatic tuning of initial parameters. This tool improves traditional knowledge tracing systems using a parallel EM algorithm with the capabilty of automatically choosing initial parameters. Experimental result demonstrates that our tool can achieve fast execution speed and greatly improve the accuracy of training parameters on both different sizes of simulated data and real educational data sets.

Keywords− Knowledge Tracing; Parameter sweeping; Parallel EM Algorithm, Learning analytics

I. INTRODUCTION

Over the past decade, online education has greatly increased in scale and quality. Especially, more and more students get enrolled in MOOCs courses, which generates large scale learning behavior data. Data-driven learning analytics tools are widely adopted to estimate the students' knowledge state based on their historical data in online educational systems. In such a learning analytics tool, each student's answer sequence is usually analyzed by the Bayesian Knowledge Tracing Model (BKT)[1] , which aims to model students' changing knowledge state during their skill acquisition process. Based on the BKT model, developers can build intelligent tutoring services in online educational systems so that they could provide students instant feedbacks or recommendations for their study, and generate assessment reports daily to enable class instructors to quickly evaluate study process. Given the significant increase in the number of students and dynamics of in-class enrollment, it is important to implement an efficient and robust BKT-based intelligent tutoring algorithm for online educational systems.

The BKT model is a special case of Hidden Markov Models(HMM)[2], consisting of observed and latent variables[3]. Usually the Expectation Maximization[4] algorithm combined with the Baum-Welch Algorithm[5, 6] , which is a classical inference algorithm for HMM, is used to train the parameters in Bayesian Knowledge Tracing. When we apply the Baum-Welch algorithm to the BKT training process we need to set up the initial parameters and execute several rounds of iterations in order to get optimal parameters.

For each cohort of students who come to study the new knowledge unit of a MOOC course, the intelligent tutoring service of a MOOC system needs to train a BKT model for them to track their progress in the learning process and to generate necessary feedbacks or reports. Therefore, the issue about the performance of BKT arises when online educational systems apply the BKT model with EM algorithm to process the learning data on-the-fly. When a large number of students simultaneously join in the session of an online course, the system often has to process the relatively long question-answering sequences generated by hundreds or even thousands of students in order to perform the BKT model training process.

There are two main problems regarding the BKT parameter tuning and execution efficiency. The first one is how to configure the initial parameters. Generally, the training result of parameters are different for disparate answer sequence data set. Due to the nature of the EM algorithm, its convergence process may get trapped in a local optimum point with an inappropriate choice of initial parameters. So it's necessary to find the suitable initial parameters for different data sets. The second problem is how to improve the execution efficiency. The increase in the amount of data often results in longer fitting time for the BKT model. When the data scale reaches tens of thousands, the training time can last for several hours for only one data set. It's intolerable for the on-the-fly learning analytics in most online education platforms which receive a large number of learning data sets on daily basis.

Previous research[7, 8] has proposed general parallel EM algorithms and applied their parallel EM algorithms in different computational frameworks. However, there is no practical parallel implementation for improving the execution efficiency for the BKT model with the Baum-welch algorithm which uses the EM algorithm. In addition, few previous efforts tackle with the problem of EM parameter initialization.

In this paper, we propose a Spark-based automatic parallel

Bayesian knowledge tracing open source tool and implement the tool on the Spark framework. To the best of my knowledge, There are no open source parallel BKT tools so far. The major contributions of our paper include:

(1) Automatic parallel initial parameter sweeping to avoid inappropriate local optimal points for different data sets.

(2) A Parallel Implementation of open source BKT tool based on the Spark framework.

(3) Optimization of the EM algorithm convergence condition.

The paper is organized as follows. Section II describes the BKT model and the related research on parallel EM algorithm. Section III introduces the automatic parallel knowledge tracing process in detail. In Section IV and Section V, we present the method of the generating simulation data and show experimental results in both the simulation data set and real data set. Finally, Sections VI gives the conclusion and discusses future work.

II. RELATED WORK

A. Knowledge Tracing

In the Intelligent Tutoring System (ITS)[9], there are a variety of methods to model students learning outcomes. The Knowledge Tracing Model shown in Fig. 1 has been widely used. The standard KT model has five parameters that are learned from data for each given skill. These five parameters indicate the model's deduced probability whether the student has mastered the skill given his chronological sequence of correct and incorrect responses to a particular set of questions. The P(G) and P(S) are two parameters that determine the student's performance on a question based on his current mastery of this knowledge. The P(G) is the guessing parameter that a student will answer correctly even if he doesn't know this skill to the question. The P(S) is the slipping parameter that a student will answer incorrectly when he already knows this skill. The P(T) and P(F) parameters dictate the change between the states of the student knowledge nodes. P(T) is the learning parameter indicating the probability of transmitting from not mastering to mastering the skill. P(F) is the forgetting rate describing the probability of forgetting this skill transmitting from knowing to not knowing. P(L_0) is referred as the prior probability of the initial knowledge node where a student knows the skill before answering the first question.

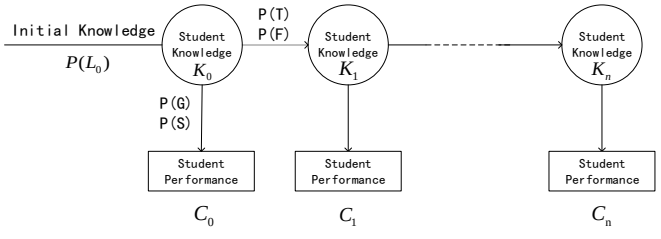

Fig. 1 The Standard Knowledge Tracing Model

B. Parallel Expectation-Maximization algorithm

Expectation-Maximization algorithm is an iterative algorithm used for maximum likelihood estimation containing hidden variables. Given a large dataset, it takes a long time for EM algorithm to finish its training process. López-de-Teruel et. al [7] proposed a generic parallel implementation of the EM algorithm for computer vision in parallel distributed memory environments. Kumar presented a fast parallel implementation of EM on NVIDIA GPUs using CUDA [10] and achieved a good acceleration effect. Based on the work of Pedro, Cui [8], applied the parallel algorithm in three distributed frameworks including Graph Lab[11], Piccolo and Spark[12]. His paper summarizes the features of each framework and their respective parallel strategies and compares the acceleration effect in different data size and machine scale. Altinigneli et al. [13] based on Kumar's work, proposed a parallel algorithm for Expectation Maximization on graphics processors which adopts the idea of asynchronous model updates respecting the memory hierarchy of the device. Hunter implemented a large-scale online Expectation Maximization with Spark Streaming for low-latency applications [14, 15]. Davier et.al introduced the parallel EM algorithm without improvement for Generalized Latent Variable Models and evaluated the overall gain in different CPU environments[16]. Masegosa et al.[17] implemented a toolbox which is a software for scalable probabilistic machine learning with a special focus on massive streaming data.

To the best of our knowledge, none of the above publications target the area of large-scale learning analytics and implemented the parallel Baum-Welch EM algorithm for BKT model. These researchers didn't present good solution to the problem of setting up the initial parameters of the BKT model with large number of data sets.

III. AUTOMATIC PARALLEL KNOWLEDGE TRACING

The total training process is divided into three stages including sequence extraction, automatic parameter initialization and iterative EM calculation. The original learning sequence data come from the online data collection system regularly. The original sequence usually contains useless information and missing question-answering items for every student. These data need to be removed in the first step.

Step1: The first step is to extract the common answer sequence from the original sequence data for the knowledge tracing program. There are two extraction strategies:(1) row first strategy, this extraction method preserves the student number as many as possible in order to model the changing knowledge state for more students. (2) column first strategy, this extraction method preserves the answer sequences as many as possible in order to accurately train the parameters by using more comprehensive answer patterns.

Step2: After getting the common answer sequences, we train the BKT model with these sequences by using the parallel knowledge tracing program. Due to the iterative and local-optimal nature of the EM process, it is important to select the proper initial values for the parameters. A good choice of initial values can not only ensure the optimal and correct training results but also shorten the iteration time of the entire EM process. We design an automatic parameter sweeping method to avoid the inappropriate initial situation. The value space of the initial parameters is divided into several uniform

subintervals[18]. The left endpoint in these subintervals for every parameter composes of the candidate set. The concrete division number is related to the data size, the parallel system size and the precision of the convergence condition. Each combination of parameter intervals can be used as a group of initial parameters to calculate the log-likelihood value in parallel because every group parameter is independent. The parameter values that lead to the optimal training result will be assigned as the initial parameters.

Step3: In the training stage, the parallel EM algorithm mainly involves three steps: the parallel E-step, the parallel M-step and the parallel iteration step. For the E-step, the computation is completely executed in a data parallel way and the M-step is run as a simple reduce process. The iteration stops when the EM calculation reaches the convergence conditions which is suitable for various data sets[19].

Given the large scale of student enrollment in our eLearning system, a single machine's memory size is not enough to execute the whole EM calculation process for the large learning datasets. We need to implement the parallel program on Spark, which is a fast and general engine for large-scale data processing. Spark has an advanced DAG execution engine that supports acyclic data flow and in-memory computing. At an abstract level, the parallel EM algorithm for Knowledge Tracing can be described as follows:

Parallel Knowledge Tracing algorithm

Input:

The array of N input student answer sequences **Seq[N],** and the length of every sequence is **T**

Output:

The initial knowledge state matrix $\boldsymbol{Pi_{1\times L}}$, the knowledge state transition matrix $\boldsymbol{A_{L\times L}}$ and The observation probability matrix $\boldsymbol{B_{L\times M}}$**, L=2(**knowledge state number**), M=2(**observation number**)**

Initialization

Input:

The parameter interval of A, B, Pi, and the partition value k

Output:

The initial parameter of A, B, Pi

Sweeping

Divide the parameter intervals in A, B, Pi into k subintervals and the left endpoint of every subinterval compose the candidate set.

Assume the set number of parameter is n, we get the k^n parameter combinations.

Calculate the log-likelihood by using the k^n combinations in parallel and choose the one that obtains the maximum log-likelihood as the initial parameters.

Repeat

Parallel E-Step

For every parameter in parameter groups (in parallel)**:**

The formal definition for

$\alpha_{xt}(i)$: Forward probability ,$\beta_{xt}(i)$: Backward probability

$\gamma_{xt}(i)$: 1-state probability, $\xi_{xt}(i,j)$: 2-state probability

are omitted and can be found in Baum-Welch Algorithm[5].

For x in Seq[N], calculate the $\alpha_{xt}(i)$ and the log-likelihood by using the forward algorithm in parallel.

For x in Seq[N], calculate the $\beta_{xt}(i)$ by using the backward algorithm in parallel.

For x in $\alpha_{xt}(i)$,$\beta_{xt}(i)$ calculate the $\gamma_{xt}(i)$ and the $\xi_{xt}(i,j)$ in parallel.

Keep the parameter with the maximum log-likelihood

Parallel M-Step

Collect the ξ and γ then calculate matrix $A_{L\times L}$ $B_{L\times M}$, $Pi_{1\times L}$ in parallel by using reduce operation.

$$a_{ij} = \frac{\sum_{x=1}^{N}\sum_{t=1}^{T-1}\xi_{xt}(i,j)}{\sum_{x=1}^{N}\sum_{t=1}^{T-1}\gamma_{xt}(i)} \quad (1)$$

$$b_{jk} = \frac{\sum_{x=1}^{N}\sum_{t=1,o_t=v_k}^{T-1}\gamma_{xt}(j)}{\sum_{x=1}^{N}\sum_{t=1}^{T}\gamma_{xt}(j)} \quad (2)$$

$$Pi_i = \sum_{x=1}^{N}\gamma_{x1}(i) \quad (3)$$

Until

Reach the convergence condition.

A. Extract the common answer sequence

The original sequence is a matrix $\Theta_{M\times N}$ filled with 1,0 and null values. We'd like to extract an inner matrix without null values and make it as large as possible. We define the threshold as

$$threshold = \frac{\sum_i^m\sum_j^n \Theta_{ij} \neq null}{2\times M} + \frac{\sum_i^m\sum_j^n \Theta_{ij} \neq null}{2\times N} \quad (4)$$

. There are two choices to implement the matrix calculation.

1) *Row First Strategy*

This strategy prefers to save more rows, which means more student records could be saved. The algorithm firstly calculates the threshold by Eq(4) and then deletes the columns where non-null values take less ratio than the threshold. Fig. 2 presents an example of extracting the common sequence according to Row-first Strategy.

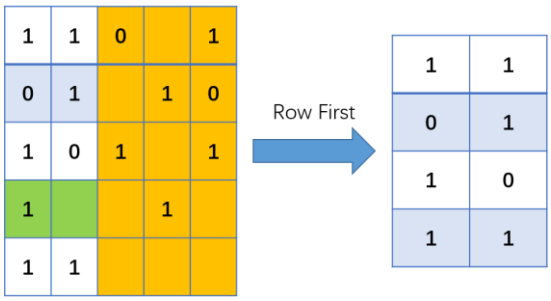

Fig. 2 Extract the sequence with row first

2) *Column First Strategy*

This strategy prefers to save more columns, which means more problem observations will be saved. The algorithm firstly calculates the threshold by Eq(4) and then deletes the row where not null values take less ratio than the threshold. Fig. 3 presents an example of Row-first Strategy, where the same input sequence as Fig.2 is filtered into a new sequence with a shorter list of question-answer results and more student rows.

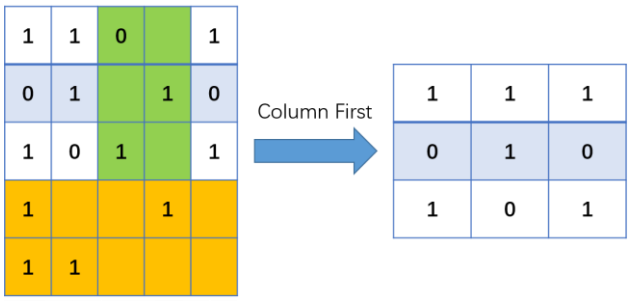

Fig. 3 Extract the sequence with column first

After choosing one of the choices above, we build a bipartite graph G (U, V, E), in which the row indexes are represented by U and column indexes are represented by V. We regard every null value as an edge in E, and denote it as (u, v), where u stands for its row index and v stands for its column index, and note that an edge doesn't have a direction, so (u, v) = (v, u). After that, delete the rows and columns represented by the points in the minimum vertex cover(MVC)[20]. The purpose of MVC is to find the minimum set of points that cover all null values of the matrix. So we can obtain a full matrix by reducing minimum sum of rows and columns.

Based the Kőnig's theorem[21], the steps to find the MVC of G are as follows. Initially we'll find a matching M. A matching is given by,

$$M = \left\{(u,v)\middle|\begin{matrix}(u,v)\in E,\ (u\in U\wedge v\in V), and\\ \forall(u_1,v_1),(u_2,v_2)\in M, \{u_1,v_1\}\cap\{u_2,v_2\}=\emptyset\end{matrix}\right\} \quad (5)$$

Then we try to find some alternating paths. An alternating path P is given by,

$$P = < e_1, e_2, \dots, e_n > and\ \forall e_i, e_{i+1}\in P, \left(e_i\in M\wedge e_{i+1}\in (E\backslash M)\right) \quad or\ (e_i\in (E\backslash M)\wedge e_{i+1}\in M) \quad (6)$$

Besides, the two endpoints of P must not belong to points covered by M. After getting a path, we change every edge's state in the path (which means from matching edge to non-matching edge or from non-matching edge to matching edge). Repeat the steps above and we'll get the maximum matching of the graph. Then we get

$$S = \{u|u\in U\wedge(\forall(u,v)\in E, v\in V, (u,v)\notin M)\} \quad (7)$$

After that we add all points that are reachable by some points in S via an alternating path to S, and finally we define Z

$$Z = (U\backslash S)\cup(V\cap S) \quad (8)$$

And Z is the MVC. By deleting rows and columns of G corresponding to the points in Z, we obtain the maximum common sequences.

B. Initial parallel parameter sweeping

The purpose of this stage is to avoid the problematic situation when the initial parameters deviate far away from the real parameter. In the first iteration, the BKT program executes the initial parallel parameter sweeping. The partition value k is calculated by,

$$k = \frac{\ln(n_{exec}/\log_{10} N)}{n_{para}} \quad (9)$$

,where the N is the number of the input sequences and the n_{para} is the number of parameters. The symbol n_{exec} is the number of Spark executors that run BKT parallel instances on computing nodes. Eq (9) tries to avoid excessive parameter sweeping time. So the more input sequences and the parameter number becomes, the less parameter interval blocks we get.

In the Parallel Knowledge Tracing algorithm, the matrix A has two parameters: The P(T) and the P(F). The value of P(F) representing the forgetting probability always be set as zero, which means that the matrix A actually has only one parameter's interval to be divided. The matrix B has two parameters: The P(G) and P(S). The matrix B actually has only one parameter's interval that needs to be divided too because the value of P(S) stays often within a very narrow range, which doesn't need the parameter sweeping stage to scan. The intervals of P(G) and P(T) are defined as $I_{P_G} = (0,1)$ and $I_{P_T} = (0,1)$. We get the S_{P_G} and S_{P_T} as two candidate parameter sets for P(G) and P(T) by,

$$S_{P_G} = \left\{P_G\middle|P_G = {}^{x}/_{k}, x\in N, y\in I_{P_G}\right\} \quad (10)$$

$$S_{P_T} = \{P_T|P_T = {}^{x}/_{k}, x\in N, y\in I_{P_T}\} \quad (11)$$

Then we generate the sets of matrix A, B, and Pi by,

$$S_A = \{(1-P_T, P_T, P_F, 1-P_F)|\ P_T\in S_{P_T}\wedge P_F = 0\} \quad (12)$$

$$S_B = \{(1-P_G, P_G, P_S, 1-P_S)|\ P_G\in S_{P_G}\wedge P_S = 0.25\} \quad (13)$$

$$S_{Pi} = \{(0.5,0.5)\} \quad (14)$$

Finally, we get the initial parameter set S_c for parameter sweeping by,

$$S_c = \{(A, B, Pi) | A \in S_A \wedge B \in S_B \wedge Pi \in S_{pi}\} \quad (15)$$

The parameter group of A, B and Pi in S_c which get the optimal log-likelihood will be retained as the initial parameters.

C. Parallel Expectation-Maximization algorithm

1) E-step and M-step

Both stages mainly calculate the log-likelihood and maximize the parameters. Fig. 4 displays the core Python code written with the Spark python API for running the E-step and M-step in parallel.

```
def E_Parallel(Seq, flag):
    alpha, prob=Forward(Seq)
    beta=Backward(Seq)
    Gamma=ComputeGamma(alpha, beta)
    Xi=ComputeGamma(alpha, beta, Gamma)
    return (flag,(alpha,prob,Gamma,Xi))
def M_reduce(x, y):
    for i, j in x, y:
        for k in range(len(i)):
            z[k]=i[k]+j[k]
    return z
SeqRDD = sc.parallelize (Seq[N])
MultiparaSeqRDD = SeqRDD.flatMap(lambda
x: [(y, x) for y in para])
E_resultRDD =
MultiparaSeqRDD.map(E_Parallel)
M_resultRDD =
E_resultRDD.reduceByKey(M_reduce)
Parameter_result=M_resultRDD.collect()
for i in range(N):
    Pi[i]=computePi()
    for j in range(N):
        A[i, j]=ComputeA(i, j)
    for k in range(M):
        B[i, k]=ComputeB(i, k)
```

Fig. 4.The core Python code for running E-step and M-step in Spark

In this section, the parameter group means the combination of A, B and Pi. The code firstly generates a RDD[12] by the parallelize function. Resilient distributed dataset (RDD) is a fault-tolerant collection of elements that can be operated on in parallel. After the parallelize operation, every sequence in Seq[N] becomes an element in the RDD collection. Every element of the parameter sets will be combined with the SeqRDD by the flatmap operation, which maps one element in the original RDDx to multiple elements of the flattened RDDz. Fig. 5 shows the process of the RDD flatmap calculation. Every element in RDDz executes the E_parallel in parallel by using the map function. The reduceBykey function aggregates the elements of the E_resultRDD which have the same key by using the M_reduce in parallel. The key of the E_resultRDD is the flag of the parameter group. So every input parameter group get the corresponding result parameter group by the E-M steps in parallel. Finally, the algorithm chooses the parameter group with the maximum log-likelihood and finishes the E-step and M-step in parallel.

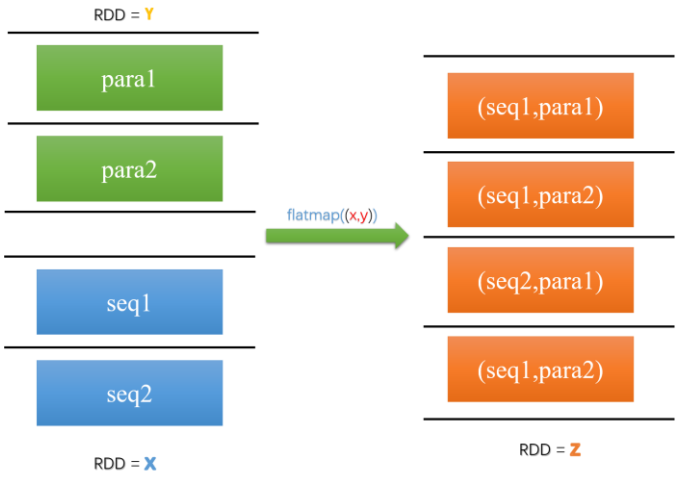

Fig. 5. The flatmap example

2) Optimization of the convergence condition

The Baum-Welch algorithm of BKT aims to find the maximum log-likelihood through the iterations. To ensure the proper outcome of the algorithm process, we specify two convergence conditions in our algorithm.

a) The D-value of log-likelihood is less than the precision P.

The D-value is the difference of the log-likelihood values between adjacent iterations. As the log-likelihood is related to the data sets, the larger data set often has the greater absolute value of log-likelihood. Given a precision of the EM convergence, the larger data scale requires a stricter convergence condition, because the percentage change of log-likelihood will decrease when the EM process reaches the convergence precision. We introduce the logarithm of data quantity to measure the data scale. The base precision is set to $P_b = 0.01$ and the base data scale is set to $S_b = \log_{10} N_b * T_b$ when the dataset contains hundreds of thousands of sequences($N_b = 10^5$) and the length of each sequence is ten($T_b = 10$). The precision P is defined by (16). N is the sequence number and T is the sequence length in practical data set.

$$P = \frac{P_b * S_b}{\log_{10} N * T} \quad (16)$$

b) The ratio of the D-values is less than hundredth

The purpose of condition b is to prevent the EM process from stopping at the scenario where the condition (a) is satisfied but the generated log-likelihood value hasn't reached its maximum yet. Because in some cases, low D-values occur in

the EM process due to its slow convergence pace. This condition makes sure the EM process reaches the optimal point by specifying that the ratio of two consecutive D-values is less than hundredth.

Our empirical study of the EM process suggests the BKT program should adopt the combinations of both conditions as the EM convergence condition.

IV. Data Sets and Implemention

We used a series of simulated date sets and one real educational data set to test our parallel BKT algorithm.

A. simulated data sets

The simulated data sets with different parameters and different data scale are generated as follows:

Simulated answer sequences generation algorithm

Input:

N: the number of sequences

T: the length of every sequence

P(L_0), P(T), P(G), P(S)

N=2, M=2

Output:

The answer sequences matrix $S_{N\times T}$

Initialization

$\mathrm{P}(L_1) = \mathrm{P}(L_0)$

For s **from** 1 to N **do**

For t **from 1 to** T **do**

Generate a random number **R** in (0,1)

$$\mathrm{P}(R_t) = \mathrm{P}(L_t) * (1 - \mathrm{P}(\mathrm{S}) + \big(1 - \mathrm{P}(L_t)\big) * P(G) \quad (17)$$

if R < **P(R_t)**

then $S_{st} = 1$

$$\mathrm{P}(L_t|evidence_t) = \frac{\mathrm{P}(L_t) * \mathrm{P}(\neg \mathrm{S})}{\mathrm{P}(L_t) * \mathrm{P}(\neg \mathrm{S}) + \mathrm{P}(\neg L_t) * \mathrm{P}(\mathrm{G})} \quad (18)$$

else $S_{st} = 0$

$$\mathrm{P}(L_t|evidence_t) = \frac{\mathrm{P}(L_t) * \mathrm{P}(\mathrm{S})}{\mathrm{P}(L_t) * \mathrm{P}(\mathrm{S}) + \mathrm{P}(\neg L_t) * \mathrm{P}(\neg \mathrm{G})} \quad (19)$$

$$\mathrm{P}(L_{t+1}) = \mathrm{P}(L_t|evidence_t) + \mathrm{P}(\neg L_t|evidence_t) * \mathrm{P}(\mathrm{T}) \quad (20)$$

End

End

We simulate a student's exercise response sequence by generating random numbers. $\mathrm{P}(L_t)$ is the probability that the student knows this knowledge at time t. $\mathrm{P}(R_t)$ calculated by Eq(17) represents the probability that the student answers the question correctly at time t. We generate the student answer results R using a random generator between 0 and 1. When the value of R is less than $\mathrm{P}(R_t)$, the student answer will be set to false, otherwise it will be set to true. The posterior probability $\mathrm{P}(L_t|evidence_t)$ can be updated by Eq (18) and Eq (19) after the algorithm calculates the sequence value at time t. Then it updates the prior probability in the next time by Eq (20).

The detailed values of N(the number of sequences), T(the length of every sequence), $\mathrm{P}(L_0)$, P(T), P(G) and P(S) for simulated date sets are given in TABLE I. Based on the pedagogical design of student homework assignments, more problems should be offered to assess the level of student knowledge mastery if knowledge skills related to these problems are more difficult for students to grasp. Therefore, we assume that the P(T) and P(G) should decrease when the length of the sequence increases and $\mathrm{P}(L_0)$ should be set as a constant.

TABLE I

The related parameters of simulated data sets

Data size		Parameter values			
T	**N**	**P(L_0)**	**P(T)**	**P(G)**	**P(S)**
10	10K	0.2	0.2	0.2	0.05
	100K				
	1000K				
20	10K	0.2	0.15	0.15	0.1
	100K				
	1000K				
30	10K	0.2	0.1	0.1	0.15
	100K				
	1000K				

B. Real educational data set

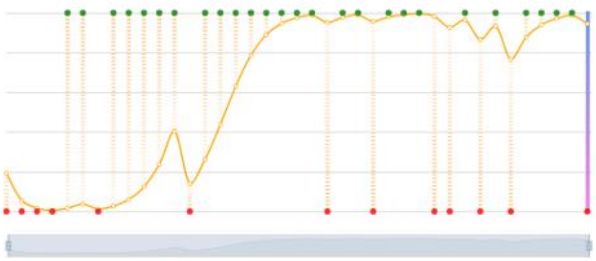

Fig. 6 A student's exercise response sequence and his learning curve of knowledge mastery

We implemented a large-scale online learning analytics system for an eLearning platform, which provides algebra courses for elementary students. Our system collects problem-answer sequences generated by thousands of students on daily basis and visualizes the learning process of individual student after analyzing the dataset through the BKT model.

Fig. 6 displays such a curve plotted by our system. The correct and wrong responses from a student are separated into the two horizontal lines on the top and bottom. The inner dots

TABLE II

Performance in different data sizes and different execution cases

		Sequence length:20				Sequence length:30			
Data size	Case	Seconds	Speed up	Log-likelihood	Iterations	Seconds	Speed up	Log-likelihood	Iterations
10K	A	516	1	-87758.9106098	57	849	1	-138742.480005	62
	B	60	8.6	-87758.9106098	57	71	11.9577464	-138742.480005	62
	C	86	6	-87758.9104524	60	95	8.93684210	-138742.479573	64
100K	A	7379	1	-875154.082758	81	9244	1	-1397999.13286	68
	B	314	23.5	-875154.08276	81	371	24.9164420	-1397999.13285	68
	C	381	19.3674540	-875154.083059	83	463	19.9654427	-1397999.13299	69
1000K	A	126642	1	-8758451.00017	134	178759	1	-13969179.7958	114
	B	3537	35.804919	-8758450.99724	134	4621	41.9523586	-13969179.7572	114
	C	4050	31.269630	-8758450.99696	140	4838	36.9489458	-13969179.7575	114

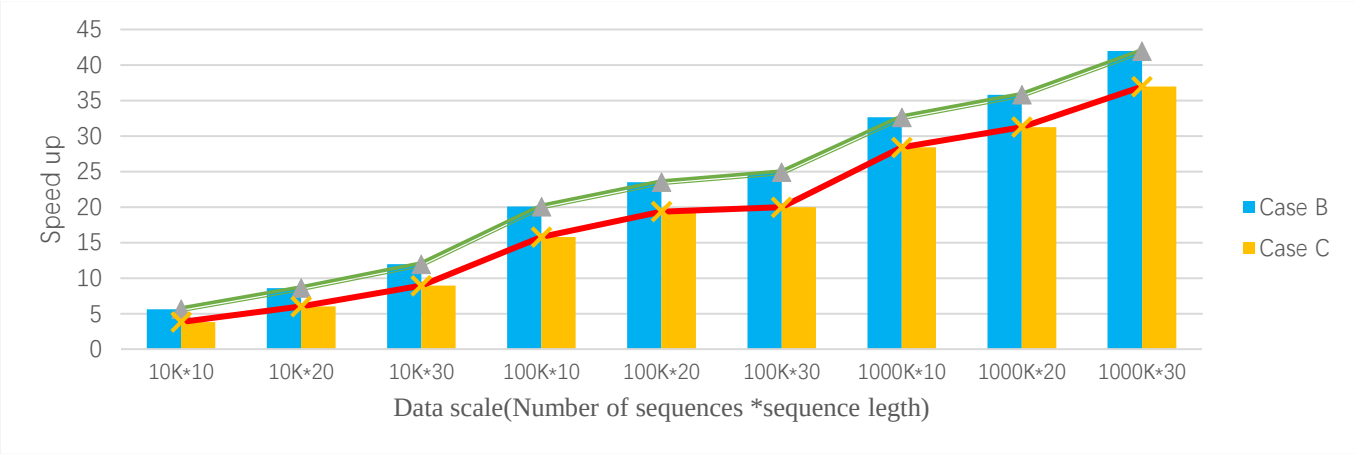

Fig. 7 The trend of Speed up in different data scale with 48 Spark executors in two parallel BKT cases.

along the learning curve represent the probability of correct answer, which are inferred by the BKT model. Our system keeps updating the estimation of the probability timely as the new student response sequences come into the system. This analytic visualization can facilitate instructors and educational science researchers to evaluate and predict knowledge mastery of every student.

In order to evaluate the performance of our BKT implementation, we select 56680 students' answer sequences as the real dataset. The total length of every answer sequence is 40, which includes 20 verbal exercises and 20 calculation exercises. These exercises are designed for the second-grade elementary students, which requires mathematic skills on addition and subtraction calculation. The overall correct rate of this data set is 82.095%.

We implemented the parallelizing BKT Tool based pyspark package and our code can be found on GitHub. The link is https://github.com/buaapyj/Parallelizing-Bayesian-Knowledge-Tracing-Tool

V. Experimental Results

Our experimental environment is a Spark cluster with 10 machines, 60 cores and 120G memory. The sequential BKT model is executed in a single machine. The detailed single machine parameters are in TABLE III:

There are three cases in the experiment:

Case A: sequential BKT model

Case B: parallel BKT model without parameter sweeping

Case C: parallel BKT model with parameter sweeping

TABLE III

the specification of each computing node in the Spark cluster

CPU	Processor Speed	Cores	Processors	Memory
E3-1246 v3	3.5GHz	4	4*2	16GB

A. The results on real eudcatinal data set

We trained this data set both on sequential BKT model and different parallel BKT models.

TABLE IV

Performance in different execution policy

Data size	Case	Seconds	Speed up	Log-likelihood	Iterations
56680	A	3710	1	-1031112.7095	82
	B	389	9.54	-1031112.7095	82
	C	597	6.21	-1031112.7095	104

The default number of Spark executors in parallel models is set to 48. TABLE IV manifests the performance of different cases in the real data set. The parallel BKT models achieve

TABLE V

Parameter error rate (sequence length:10)

Data size	Parameters			
	P(G)	*P(S)*	*P(T)*	$P(L_0)$
10K	12.09%	2.34%	6.34%	12.17%
100K	12.39%	1.40%	4.51%	13.15%
1000K	13.35%	2.23%	5.07%	13.17%

TABLE VI

Parameter error rate (sequence length:20)

Data size	Parameters			
	P(G)	*P(S)*	*P(T)*	$P(L_0)$
10K	5.15%	0.52%	2.41%	9.16%
100K	5.97%	0.06%	3.00%	9.92%
1000K	6.36%	0.20%	2.62%	9.87%

TABLE VII

Parameter error rate (sequence length:30)

Data size	Parameters			
	P(G)	*P(S)*	*P(T)*	$P(L_0)$
10K	4.73%	1.20%	1.87%	6.43%
100K	2.41%	0.17%	1.25%	5.33%
1000K	2.30%	0.07%	1.28%	5.33%

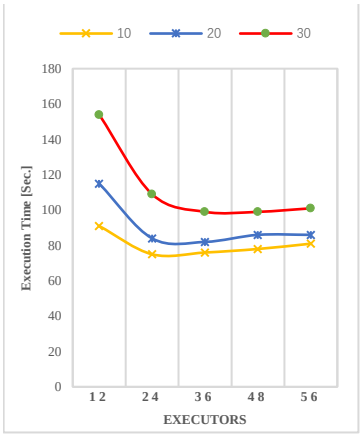

Fig. 8 The execution time with various executors in different data sets, the data size is 10K

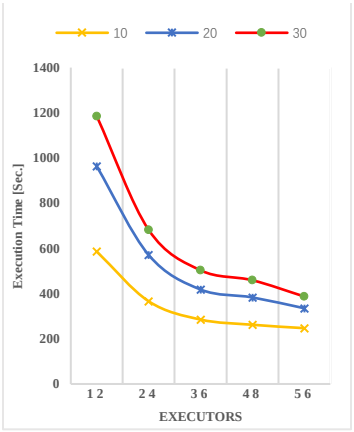

Fig. 9 The execution time with various executors in different data sets, the data size is 100K

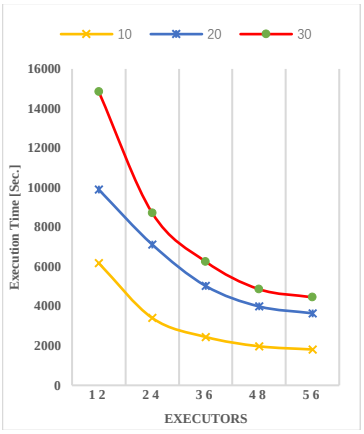

Fig. 10 The execution time with various executors in different data sets, the data size is 1000K

several times acceleration effect. Due to the limited amount of the real dataset, it is inconvenient to fully evaluate the speed-up performance of our parallel algorithm. Thus, we use simulated dataset to run the algorithm in different cases.

B. The results on simulated data sets

1) *Experiments in different data sets*

We apply the three cases A, B, C in different data scales and get the results on TABLE II. The result parameters of P(G), P(S), P(T), $P(L_0)$ are omitted and will be discussed later in the subsection of parameter error rate. We notice that when the data scale rises up to million level, it takes a long time to execute the sequential BKT model even up to two days. In TABLE II and Fig. 7, one can find the tendency that the speed-up increases in both case B and case C with the increase in the data scale. These observations confirm the efficiency improvement made by our parallel algorithms. The execution time of 10K data size is 516 seconds, which seems acceptable for batch-mode educational data analytics but not fast enough for real-time intelligent tutoring. When the number of students under test increases to the campus-level or district level, the system often has to deal with tens of thousands exercise sequences for different subjects and grades, thus demanding the parallel implementation of a BKT system.

The acceleration effect of case C that has extra execution time for parameter sweeping seems lower than that of Case B. But the further experiments indicate that the parameter sweeping is essential to ensure the correctness of the training results.

We designed 81 initial parameter groups in which P(G) and P(T) are separately set from 0.1 to 0.9 with the interval of 0.1. We experimented these parameter groups in the data set with 10K sequence number and the length of every sequence is 10. The result shows that there are 14 initial parameter groups to ensure the correctness of training whereas the result of the case C absolutely converge to the setting parameters of the simulated data sets.

2) *Result parameter error rate in different data sets*

The error rate is calculated by,

$$\mathrm{Error}_p = \frac{|\mathrm{Result}_p - \mathrm{Set}_p|}{\mathrm{Set}_p} \tag{21}$$

. Set_p is the parameter used to generate the simulated date sets including P(G), P(S), P(T) and $P(L_0)$ and the specific values can be found in TABLE I. The data sets with an equal sequence length have the same parameter values. Result_p is the result parameter trained by the BKT model. TABLE VI, TABLE VII and TABLE VIII exhibit that the whole trend of

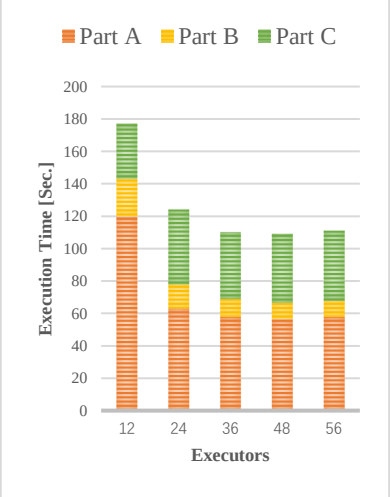

Fig. 11 The detailed execution time of three parts with different executors, The data size is 10K and the sequence length is 30.

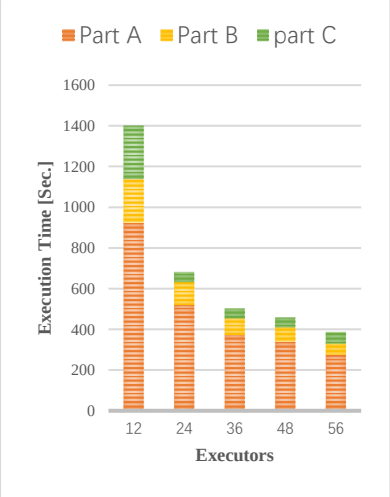

Fig. 12 The detailed execution time of three parts with different executors, The data size is 100K and the sequence length is 30.

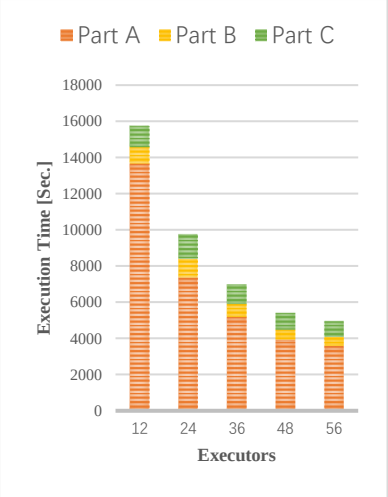

Fig. 13 The detailed execution time of three parts with different executors, The data size is 1000K and the sequence length is 30.

the error rate descends with the increase in both data scale and sequence length. This trend also illustrates that increasing data scale is helpful to improve the parameter accuracy.

3) *Experiments with executors*

Fig. 8, Fig.9 and Fig. 10 demonstrate the execution time in different data sets in case C. The legend of 10, 20 and 30 represent that the sequence length is 10, 20 and 30. Fig. 8 indicates that the excessive executors are not necessary for improving the performance of our algorithm for relative small datasets. Fig. shows that the increase in the scale of datasets demands more computational nodes to run our parallel algorithm. We also notice that there is no linear correlation between the number of Spark executors and parallel execution time of the BKT algorithm. Besides, the execution time even slightly increases with more Spark executors in Fig. 8. To investigate the sublinear speed-up problem, we divide the execution time of the whole BKT training process into three parts: Part A - the time of E-M step in the whole iterations, Part B - the time of initial parameter sweeping, and Part C - the other time in the training process. We counted the detailed time (in seconds) of the three parts in the experiments with executors.

TABLE VIII

Data size	Executors	Total time	Part A	Part B	Part C	Average time
10K	12	154	120	23	34	1.935
	24	109	63.1	15	45.9	1.017
	36	99	57.9	11	41.1	0.933
	48	99	56.5	10	42.5	0.911
100K	12	1185	922	216	263	13.76
	24	682	519	114	49	7.746
	36	503	372	84	47	5.552
	48	459	339	72	48	5.059
1000K	12	14845	13666	900	1178	120.9
	24	8716	7374	1020	1342	65.25
	36	6255	5185	720	1070	45.88
	48	4860	3930	552	930	34.77

TABLE VIII shows the three part time of BKT training process for three data sets with four different number of Spark executors. The last column labelled as average time represents the average execution time of the E-M step in one iteration. The average time reflect the duration of parallel E-M step in one iteration

Note that the average time for parallel E-M execution in one iteration can't be infinitely shortened with the increasing numbers of Spark executors because of the inherent cost introduced by distributed coordination and communication in the Spark system. According to our measurement, the minimum execution time for the EM step approximately costs 0.9 second in the case of 10K data size and 30 sequence length. The Part C usually includes sequential execution time and Spark initialization time.

Fig. 11, Fig. 12 and Fig. 13 visualize the change of proportions among these different parts with the increasing numbers of executors for analyzing three data sets. Fig. and Fig. 13 illustrate that the entire BKT execution time decreases rapidly due to the significant decrease in the Part-A time but the improvement in speedup performance gradually become slowed down. There are two factors for this sublinear speedup: (1) the proportion of the inherent sequential operation and coordination cost in parallel E-M steps increases. (2) the amount of Part-A and Part-B time remains nearly constant. So the scalability of the parallel BKT system could be further improved if these two cost factors are reduced

VI. Conclusion and Future Work

In this paper, we purpose an automatic parallel Bayesian Knowledge Tracing(BKT) open source tool in order to improve the training efficiency and automate the training process for large-scale learning analytics. We aim to address the two major challenges in our application scenarios: (1) how to ensure the

correct convergence result of the BKT model training process;(2) how to improve the execution performance for large-scale data sets. We introduce the automatic parallel parameter sweeping method and design a parallel implementation of Baum-Welch EM algorithm based on the Spark computational framework.

Experimental results confirm that our system achieves performance improvements up to 42 times maximum than the original implementation of the sequential BKT model. Further experiments also demonstrate the scalability of our algorithm running on varied numbers of Spark executors.

Our work can be improved in the following ways. Current implementation of our BKT system assumes that all the students have the same parameters through their learning process. But in fact, students often possess different levels of learning capability, which demands individualized BKT models for every student. Furthermore, our implementation only supports a single skill BKT model that assumes each exercise item only involves single skill without considering latent multiple subskills in student homework and assignments. As future work, we will extend the system to support both individualization and multiple-subskills.

VII. ACKNOWLEDGEMENTS

This work was supported by grant from NSFC (Grant No. 61532004) and State Key Laboratory of Software Development Environment (Funding No. SKLSDE-2017ZX-04). This work is partially funded by Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University.

VIII. REFERENCE

[1] A. T. Corbett and J. R. Anderson, "Knowledge tracing: Modeling the acquisition of procedural knowledge," *User modeling and user-adapted interaction*, vol. 4, no. 4, pp. 253-278, 1994.

[2] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361-365, 1996.

[3] T. Käser, S. Klingler, A. G. Schwing, and M. Gross, "Beyond knowledge tracing: Modeling skill topologies with bayesian networks," in *International Conference on Intelligent Tutoring Systems*, 2014, pp. 188-198: Springer.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1-38, 1977.

[5] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The annals of mathematical statistics*, vol. 41, no. 1, pp. 164-171, 1970.

[6] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, 1989.

[7] P. E. López-de-Teruel, J. M. García, and M. E. Acacio, "The Parallel EM Algorithm and its Applications in Computer Vision," in *PDPTA*, 1999, pp. 571-578.

[8] H. Cui, J. Wei, and W. Dai, "Parallel implementation of expectation-maximization for fast convergence," ed, 2010.

[9] V. A. Goodkovsky, "Intelligent tutoring system," ed: Google Patents, 2004.

[10] N. P. Kumar, S. Satoor, and I. Buck, "Fast parallel expectation maximization for Gaussian mixture models on GPUs using CUDA," in *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*, 2009, pp. 103-109: IEEE.

[11] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," in *OSDI*, 2012, vol. 12, no. 1, p. 2.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[13] M. C. Altinigneli, C. Plant, and C. Böhm, "Massively parallel expectation maximization using graphics processing units," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 838-846: ACM.

[14] T. Hunter, T. Das, M. Zaharia, A. Bayen, and P. Abbeel, "Large-scale online expectation maximization with spark streaming," 2012.

[15] R. D. Datasets, "A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. NSDI*, 2012.

[16] M. von Davier, "High-Performance Psychometrics: The Parallel-E Parallel-M Algorithm for Generalized Latent Variable Models," *ETS Research Report Series*, vol. 2016, no. 2, pp. 1-11, 2016.

[17] A. R. Masegosa *et al.*, "AMIDST: a Java Toolbox for Scalable Probabilistic Machine Learning," *arXiv preprint arXiv:1704.01427*, 2017.

[18] F. Marozzo, D. Talia, and P. Trunfio, "A cloud framework for parameter sweeping data mining applications," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 367-374: IEEE.

[19] C. J. Wu, "On the convergence properties of the EM algorithm," *The Annals of statistics*, pp. 95-103, 1983.

[20] J. Chen, I. A. Kanj, and W. Jia, "Vertex cover: further observations and further improvements," *Journal of Algorithms*, vol. 41, no. 2, pp. 280-301, 2001.

[21] H. König, "A general minimax theorem based on connectedness," *Archiv der Mathematik*, vol. 59, no. 1, pp. 55-64, 1992.