

# Triple Jump Acceleration for the EM Algorithm

Han-Shen Huang    Bou-Ho Yang    Chun-Nan Hsu

Institute of Information Science

Academia Sinica

Nankang, Taipei, Taiwan

{hanshen,ericyang,chunnan}@iis.sinica.edu.tw

## Abstract

*This paper presents the triple jump framework for accelerating the EM algorithm and other bound optimization methods. The idea is to extrapolate the third search point based on the previous two search points found by regular EM. As the convergence rate of regular EM becomes slower, the distance of the triple jump will be longer, and thus provide higher speedup for data sets where EM converges slowly. Experimental results show that the triple jump framework significantly outperforms EM and other acceleration methods of EM for a variety of probabilistic models, especially when the data set is sparse or less structured, which usually slow down EM but are common in real world data sets. The results also show that the triple jump framework is particularly effective for Cluster Models.*

## 1. Introduction

The Expectation-Maximization (EM) algorithm [6] is one of the most popular algorithms for learning probabilistic models from incomplete data. However, when applied to large real world data sets with a large number of parameters to estimate, the EM algorithm is slow to converge. If the data sets also contain a large proportion of missing data or it is required to have a large number of hidden variables in the model to be learned, the convergence can be even slower. Our goal is to develop an approach to accelerating the EM algorithm that requires minimum prior knowledge about the data and no human intervention for tuning. The result is the *triple jump* acceleration framework. The idea is to extrapolate the third search point based on the previous two search points found by regular EM. The extrapolation can reach a point far away from the previous two points, like hop, step and jump in the triple jump. An important feature of the triple jump framework is that, as the convergence rate of regular EM becomes slower, the distance of the jump will be longer, and thus provide higher speedup

for data sets where EM converges slowly. In addition to accelerating EM, the triple jump framework can also be applied to other bound optimization methods [10], including iterative scaling [2], non-negative matrix factorization [7], and convex-concave computational procedure [13]. Experimental results show significant speedup for several different probabilistic models, including Bayesian Networks, Hidden Markov Models, and Mixtures of Gaussians. Experimental results also show that the framework is particularly effective for AUTOCLASS-like Cluster Models [4], for which the accelerated EM can always find the local optimum with one “triple jump,” regardless of the sparsity of data.

The triple jump framework is based on the Aitken acceleration method [3]. Previously, Bauer, Koller and Singer [1] has proposed an Aitken-based method to accelerate EM for Bayesian Networks called *parameterized EM*. Ortiz and Kaelbling [8] has proposed a similar method for Mixtures of Gaussians. Though they showed that their methods can speedup EM in their experiments, the convergence property of EM is no longer guaranteed. Salakhutdinov and Roweis [10] showed that with the learning rate (i.e., the extent of the extrapolation) within a certain interval, parameterized EM is guaranteed to converge. However, since the learning rate in such an interval is too small, the speedup will not be significant. Therefore, they proposed another method called *adaptive overrelaxed EM* [10], which switches back to regular EM during the search if the new data likelihood is not increased. In this way, the data likelihood will monotonically increase and adaptive overrelaxed EM is guaranteed to converge. They also generalized their methods to other bound optimization methods [10]. The triple jump framework accelerates adaptive overrelaxed methods further and is guaranteed to converge as well. A critical difference between the triple jump framework and those previous works is that we use different learning rates for independent sub-vectors in the parameter vectors. In contrast, previous works use one learning rate for all parameters. The derivation in Salakhutdinov et. al. [10] shows that for the one-learning-rate case, the optimal learn-

ing rate increases/decreases the speedup of the parameters that should converge slower/faster, so that they can converge at the same time. By updating sub-vectors with different learning rates, it is likely to achieve more acceleration because each sub-vector has the chance to use the optimal learning rate of itself.

## 2. Bound Optimization Methods

Given an incomplete data set  $\mathcal{D}$ , suppose we want to learn the parameter vector  $\theta$  of a probabilistic model that maximizes the log-likelihood  $L_{\mathcal{D}}(\theta)$  of the data<sup>1</sup>. The EM algorithm solves the problem by iteratively searching for a local optimal solution  $\theta^*$  on the data likelihood surface with the guarantee that the likelihood of estimates increases monotonically. In other words, EM is guaranteed to converge to a local optimum.

The EM algorithm is one of the bound optimization methods [10], which exploits a bound  $G(\theta)$  on the objective function  $L(\theta)$  and proceeds by optimizing  $G(\theta)$ . Bound optimization methods assume that for any  $\theta$  there exists  $G(\theta)$  such that  $G(\theta) \leq L(\theta)$  and it is easy to solve  $\arg \max_{\theta} G(\theta)$ . In each iteration  $t$ , a bound optimization method will find  $\theta^{(t+1)}$  by creating  $G^{(t)}(\theta)$  such that  $G^{(t)}(\theta^{(t)}) = L(\theta^{(t)})$  and then solve this optimization problem:

$$\theta^{(t+1)} = \arg \max_{\theta} G^{(t)}(\theta). \quad (1)$$

Since  $L(\theta^{(t)}) = G^{(t)}(\theta^{(t)}) \leq G^{(t)}(\theta^{(t+1)}) \leq L(\theta^{(t+1)})$ ,  $L(\theta)$  is guaranteed to increase monotonically and converge to a local maximum.

Overrelaxed bound optimization methods [10] accelerate regular bound optimization methods by an extrapolation to the direction of bound optimization:

$$\theta^{(t+1)} = \theta^{(t)} + \eta(\arg \max_{\theta} G^{(t)}(\theta) - \theta^{(t)}), \quad (2)$$

where  $\eta$  is the learning rate and  $G^{(t)}(\theta)$  is the bound as in bound optimization methods. In EM, for example,  $G^{(t)}(\theta)$  is the expected log-likelihood function computed in the E-step. It follows that when  $\eta = 1$ , Equation (2) is reduced to Equation (1) and is just the regular bound optimization. The general algorithm of the overrelaxed bound optimization methods is given in Algorithm 1.

Instantiation of Algorithm 1 for EM with Bayesian Networks is called *parameterized EM* [1], or pEM. It has been shown that parameterized EM converges within the neighborhood of a local maximum when  $0 < \eta < 2$  [11]. However, this learning rate is too small to provide significant speedup. Salakutdinov and Roweis [10] proposed the

<sup>1</sup>We will abbreviate  $L_{\mathcal{D}}(\theta)$  and use  $L(\theta)$  in the context where  $\mathcal{D}$  is obvious.

---

### Algorithm 1 Overrelaxed bound optimization

---

- 1: **Input:**  $\eta$
  - 2: randomly initialize  $\theta^{(0)}$ ,  $t = 0$
  - 3: **while** not converge **do**
  - 4:      $\theta^{(t+1)} \leftarrow$  Equation (2) with  $\eta$
  - 5:      $t \leftarrow t + 1$
  - 6: **end while**
- 

adaptive overrelaxed bound optimization methods to dynamically change the learning rate to provide significant speedup. Algorithm 2 gives the pseudo code of this method.

---

### Algorithm 2 Adaptive overrelaxed bound optimization

---

- 1: randomly initialize  $\theta^{(0)}$ ,  $\eta^{(0)}$ ,  $t = 0$
  - 2: **while** not converge **do**
  - 3:      $\theta^{(t+1)} \leftarrow$  Equation (2) with  $\eta^{(t)}$
  - 4:     **if**  $L(\theta^{(t+1)}) < L(\theta^{(t)})$
  - 5:          $\theta^{(t+1)} \leftarrow$  Equation (1)
  - 6:         determine  $\eta^{(t+1)}$
  - 7:      $t \leftarrow t + 1$
  - 8: **end while**
- 

In Algorithm 2,  $\eta^{(t+1)}$  is updated by multiplying  $\eta^{(t)}$  with a given increasing rate if the condition in line 4 is false. Otherwise,  $\eta^{(t+1)}$  will be reset to one. Adaptive overrelaxed bound optimization methods are guaranteed to converge because the log-likelihood will increase monotonically.

In summary, bound optimization methods are guaranteed to converge. Overrelaxed bound optimization methods accelerate bound optimization methods by extrapolation with a fixed learning rate, but might not converge. Adaptive overrelaxed bound optimization methods dynamically change the learning rate and switch back to bound optimization methods to ensure convergence.

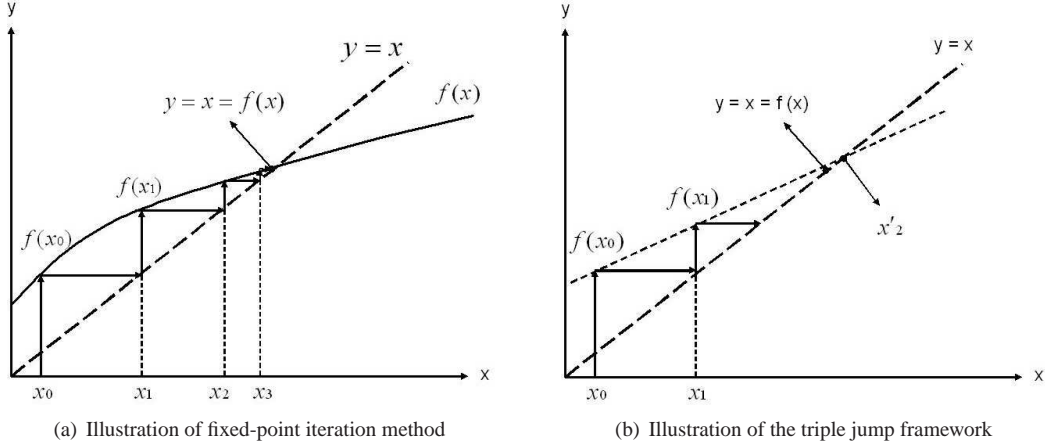
## 3. Accelerating Bound Optimization Methods

Equation 2 can be considered as a fixed-point iteration which has a general form  $x = f(x)$ . Figure 1 illustrates an example of the fixed-point iteration when  $x$  is a scalar. The idea is to find the intersection  $x^*$  of  $y = f(x)$  and  $y = x$  by iteration, as illustrated in Figure 1(a). When  $f'(x) \approx 0$  near the intersection point, the convergence rate will be high, but when  $|f'(x)| \approx 1$  near the intersection point, the convergence will be slow. In that case, Aitken's acceleration suggests compute an extrapolation from  $(x_0, x_1)$  to  $(x_1, x_2)$  to find the next search point. That is, let

$$\gamma = \frac{|x_2 - x_1|}{|x_1 - x_0|},$$

compute a new  $\hat{x}_2$  by extrapolation as follows:

$$\hat{x}_2 = x_1 + \gamma(x_2 - x_1).$$



**Figure 1. Bound optimization methods can be considered as fixed-point iteration, as shown in 1(a). Therefore, bound optimization methods can be accelerated by extrapolation, as shown in 1(b).**

Assuming that  $\gamma < \kappa$  where  $0 < \kappa < 1$  [3], and the errors of  $x_0$ ,  $x_1$  and  $x_2$  with  $x^*$  are in the same direction and reduced at the same rate [3], we can perform the extrapolation along the direction as illustrated in Figure 1(b), by repeatedly applying the extrapolation with the same  $\gamma$ :

$$\begin{aligned} \hat{x}^* &= x_1 + \sum_{s=0}^{\infty} \gamma^s (x_2 - x_1) \\ &= x_1 + \frac{1}{1-\gamma} (x_2 - x_1) \\ &= x_1 + \eta (x_2 - x_1). \end{aligned}$$

If  $f(x)$  is not a linear function, the above equation provides a secant line extrapolation to approximate  $x^*$ .

The above derivation can be generalized to the case where  $x$  is a vector and applied to bound optimization methods including EM to obtain overrelaxed bound optimization methods. However, in both overrelaxed bound optimization and adaptive overrelaxed bound optimization,  $\eta$  is assigned heuristically. With the above derivation, we can assign the learning rate  $\eta$  more aggressively based on  $\gamma$ . For example, suppose  $\gamma = 0.8$ , we have  $\eta = 5$ , which is much higher than 2, the theoretical upper bound that guarantees convergence. This way, we can explain why assigning  $\eta > 2$  can accelerate bound optimization methods. When the convergence is slow, we will have  $f'(x) \approx 1$  and  $\gamma \approx 1$ , too. In that case,  $\eta$  will be very large and provide a large acceleration. Therefore, we can dynamically assign  $\eta$  according to  $\gamma$  such that  $\eta$  will increase when the convergence is slow and decrease otherwise.

## 4. The Triple Jump Framework

The triple jump framework for accelerating bound optimization methods is derived from the analysis described in the last section. The basic idea is to perform two iterations of bound optimization or overrelaxed bound optimization to obtain  $\gamma$ , and then compute the next search point with a large  $\eta$ . This is similar to the hop, step and jump phases in triple jump and explains why we named it this way. To achieve further acceleration, we can even combine the triple jump with adaptive overrelaxed bound optimization methods by dynamically adjusting the learning rate of the hop and step pair.

There are issues that must be resolved to realize this basic idea. First, if an overrelaxed bound optimization method is applied in the first two iterations, they must use the same learning rate. Otherwise, the two iterations will become irrelevant and the ratio obtained will not be reasonable. The second issue is that like other Aitken-based acceleration methods, the extrapolations by the triple jump may not always converge. We can resolve this issue by discarding the third jump and resorting to the regular bound optimization method for the next search point, in a way similar to how adaptive overrelaxed bound optimization methods ensure convergence. At last, the triple jump might lead to an illegal value. We can interpolate the third jump with the second one to bring it back to a legal value.

Algorithm 3 gives the algorithm that computes the third jump in the triple jump framework. The input consists of three parameter vectors — initial, hop and step estimates, computed in advance using Equation 2 with the same learning rate. Note that the parameter vectors are divided into independent sub-vectors and the jump is computed for each

sub-vector independently. For example, in a Mixture of Gaussians model, parameters of different Gaussian components (i.e., mean and variance) are independent and therefore each parameter constitutes an independent sub-vector. But the weights must sum to one and together they constitute a sub-vector. Updating sub-vectors reduces the chance of jumping to an illegal value.

---

### Algorithm 3 Jump

---

```

1: input: initial estimate  $\theta_a$ , hop estimate  $\theta_b$ , and step estimate  $\theta_c$ 
2: output: jump estimate  $\theta_d$ .
3: for each sub vector set  $(\theta'_a, \theta'_b, \theta'_c)$  do
4:    $\gamma \leftarrow \frac{\|\theta'_c - \theta'_b\|}{\|\theta'_b - \theta'_a\|}$ .
5:   if  $(\gamma < \kappa)$ 
6:      $\theta'_d \leftarrow \theta'_b + \frac{1}{1-\gamma}(\theta'_c - \theta'_b)$ .
7:     while  $\theta'_d$  is illegal do  $\theta'_d \leftarrow 0.5\theta'_c + 0.5\theta'_a$ 
8:     else
9:        $\theta'_d \leftarrow \theta'_c$ 
10:    end if
11: end while

```

---

Algorithm 4 gives the pseudo code of the triple jump framework, which can be considered as a greedy state-space search algorithm that searches for a parameter vector  $\theta^*$  that satisfies the condition of convergence. There are three operators for expanding the search space:

- invoking Algorithm 3 to make the third jump,
- extrapolating as in overrelaxed bound optimization with Equation (2), or
- applying an iteration as in regular bound optimization methods using Equation (1).

They represent the most aggressive, moderately aggressive and conservative strategies to search for the next candidate parameter vector.

In each iteration, the candidate parameter vectors produced by the three operators will be tested to see whether they increase the data likelihood and whether they satisfy the constraints as a legal value. The candidate parameter vector computed as the third jump by Algorithm 3 will have the top priority. If it passes the test, it will be selected for the next iteration. Otherwise, the one produced by Equation (2) will be considered. If it still fails the test, the one produced by Equation (1), that is, the most conservative one, will be selected. This is designed to ensure that the search will always converge, with the conservative candidates backing up for the more aggressive candidates.

For conciseness, we define a two-valued function:

$$[L^{(t)}, \hat{\theta}_{\text{BO}}^{(t+1)}] = \mathbf{BO1}(\theta^{(t)}),$$

which returns the log-likelihood  $L^{(t)} = L(\theta^{(t)})$  and  $\hat{\theta}_{\text{BO}}^{(t+1)}$  by executing Equation (1). This function computes one iteration of bound optimization.

In each iteration of Algorithm 4, an array of three candidate parameter vectors  $(\hat{\theta}_1^{(t)}, \dots, \hat{\theta}_3^{(t)})$  will be tested in order as described above to select one as  $\theta^{(t)}$  for the next iteration. Depending on which of them is selected, a new array of three candidate parameter vectors will be computed differently.

- $\hat{\theta}_1^{(t)}$ : In this case, the algorithm performs a triple jump. Another hop and step pair must be prepared before the next triple jump can be made. Therefore, there will be no new  $\hat{\theta}_1^{(t+1)}$  for the next iteration. The algorithm will produce  $\hat{\theta}_3^{(t)}$  by **BO1** and  $\hat{\theta}_2^{(t)}$  by executing Equation (2). If we choose to apply adaptive overrelaxed bound optimization for  $\eta$  when computing  $\hat{\theta}_2^{(t)}$ , the algorithm will increase  $\eta$  here;
- $\hat{\theta}_2^{(t)}$ : If a triple jump attempt fails the test,  $\hat{\theta}_2^{(t)}$  will be considered. And if it is selected, a new triple jump attempt  $\hat{\theta}_1^{(t+1)}$  will be made by invoking Algorithm 3 with input  $\theta^{(t-1)}$ ,  $\theta^{(t)}$ , and  $\hat{\theta}_2^{(t+1)}$ . That is, the last selected candidate, this candidate, and the next extrapolation result. When computing  $\hat{\theta}_2^{(t+1)}$  we will have  $\hat{\theta}_1^{(t+1)}$  computed for the next iteration as well. In this case,  $\eta$  will not be adjusted;
- $\hat{\theta}_3^{(t)}$ : In this case, even a less aggressive extrapolation fails the test. The algorithm will reset  $\eta$  and prepare another triple jump attempt  $\hat{\theta}_1^{(t+1)}$  by invoking Algorithm 3 with input  $\theta^{(t-1)}$ ,  $\theta^{(t)}$ , and  $\hat{\theta}_3^{(t+1)}$ . In this case, the hop and step pair is obtained by regular bound optimization **BO1**. A new  $\hat{\theta}_2^{(t+1)}$  will also be produced here.

---

### Algorithm 4 Triple jump

---

```

1: randomly initialize  $\theta^{(0)}$ ,  $t \leftarrow 0$ 
2:  $[L^{(0)}, \hat{\theta}_3^{(1)}] \leftarrow \mathbf{BO1}(\theta^{(0)})$ ,  $\hat{\theta}_2^{(1)} \leftarrow$  Equation (2) with  $\eta$ 
3: repeat
4:   for  $i \leftarrow 1 \dots 3$  do
5:      $[\hat{L}_i^{(t)}, \hat{\theta}_3^{(t+1)}] \leftarrow \mathbf{BO1}(\hat{\theta}_i^{(t)})$ 
6:     if  $(\hat{L}_i^{(t)} - L^{(t-1)}) > \delta$ 
7:       adjust  $\eta$ ;  $\theta^{(t)} \leftarrow \hat{\theta}_i^{(t)}$ ;  $\hat{\theta}_2^{(t+1)} \leftarrow$  Equation (2) with  $\eta$ 
8:       if  $(i \neq 1)$   $\hat{\theta}_1^{(t)} \leftarrow \text{Jump}(\theta^{(t-1)}, \theta^{(t)}, \hat{\theta}_i^{(t+1)})$  end if
9:     end if
10:   end for
11:    $t \leftarrow t + 1$ 
12: until convergence condition is satisfied

```

---

## 5. Experimental Results

We experimentally evaluated the performance of the EM instantiation of the triple jump framework for different probabilistic models. We will use the following notation to refer to different optimization methods:

- EM: the EM algorithm
- pEM( $\eta$ ): parameterized EM — that is, Algorithm 1 for EM with a fixed learning rate  $\eta$
- pEM: parameterized EM in general
- AEM: the EM instantiation of adaptive overrelaxed bound optimization — that is, Algorithm 2 with EM
- TJEM( $\eta$ ): EM instantiation of the triple jump framework (i.e., Algorithm 4) with a fixed learning rate  $\eta$  for preparing hop and step in a triple jump
- TJEM(AEM): triple jump EM with adaptive learning rate for preparing hop and step
- TJEM: triple jump EM in general

We measured the convergence performance of the different methods by counting the number of times that Equation (2) is executed instead of the number of iterations because Equation (2) may be executed more than once before the next parameter vector  $\theta^{(t+1)}$  is selected in TJEM and AEM. We will simply refer to this measure as *iterations* in the following discussion.

### 5.1. Hidden Markov Models

In our first experiment, we evaluated the acceleration of TJEM with Hidden Markov Models (HMM). We synthesized data sets from a five-state, 20-symbol HMM with randomly assigned state transition probabilities. Each data set contains 500 sequences of length 100. We used the data sets to train HMM with different methods and compared their convergence performance. We compared EM versus TJEM(1.0) and pEM(1.4) versus TJEM(1.4). Note that TJEM(1.0) is TJEM using EM to prepare hop and step. Figure 2 shows the results. TJEM outperforms its counterparts by reaching local maxima faster with much less iterations. We had tried other learning rates for pEM and obtained similar results.

### 5.2. Bayesian Networks

In the next experiment, we evaluated the performance of TJEM with the ALARM model [5], a large real world Bayesian Network with 37 nodes. We randomly assigned conditional probabilities and synthesized data sets with

1,000 examples. In addition, we randomly removed 90% of data from the data set to evaluate the performance of TJEM for sparse data. The presence of missing data yields many local maxima on the log-likelihood surface that slows down EM. Dempster et al. [6] suggested that in the neighborhood of the local optimum, the rate of convergence of EM is the ratio of the missing information (i.e., the entropy of the missing data) to the complete information. Since missing information increases as the proportion of missing data increases, we expect that EM will converge slowly when 90% of data is missing. When EM converges slowly, TJEM will increase the learning rate for the triple jump and achieve large speedup. Figure 3(a) shows the learning curves of EM versus TJEM(1.0) for the data set with 90% of missing data. The curves show that TJEM converges much faster than EM as expected. We can zoom in to the early iterations as shown in Figure 3(b) to observe the change of the log-likelihood obtained in each iteration. The figure shows clear consecutive “triple jumps” — two small hops followed by a far jump, by TJEM.

We also compared the speedup achieved by TJEM, pEM and AEM with different proportions of missing data. We randomly removed 30%, 60% and 90% of data and compared the iterations required for TJEM, pEM and AEM to converge for these data sets. We synthesized ten data sets for each percentage of proportion of missing data. Figure 4 shows the results in scatter plots. TJEM is faster than its counterpart if the spot lies to the left of the diagonal line, close to the top, and far away from the line. The results show that, TJEM(1.4) achieves higher speedup than pEM(1.4) and the improvement is larger for data sets with more missing data. TJEM(AEM) also significantly outperforms AEM, except for four cases that are slightly slower than AEM. We examined those cases and found that in fact, TJEM achieved nearly optimal log-likelihood values faster than AEM, but took more iterations to actually converge because it made several futile triple jump attempts on its way to converge.

### 5.3. Mixture of Gaussians Models

We also performed an experiment to investigate the speedup achieved by TJEM for the Mixture of Gaussians (MoG) model with different level of “well separated” Gaussian components [8, 12]. If the Gaussian components are not well separated, then the data is less structured and has high missing information that will slow down EM. We synthesized data sets for two MoG models. Each model contains five two dimensional Gaussian components with means at  $(0, 0)$ ,  $(0, 1)$ ,  $(1, 0)$ ,  $(0, -1)$ , and  $(-1, 0)$ . The variances in the first model are all 1.0, representing a less structured case, and 0.5 in the second model, representing a structured, well separated case. For each Gaussian com-

ponent, we synthesized 400 data points. That amounts to 2,000 points for each data set. Figure 6 shows the learning curve results comparing TJEM, EM and AEM. The results show that TJEM outperforms its counterparts and achieves higher speedup for the less structured data set. Figure 6(b) also illustrates the situations where TJEM jumps to points where their log-likelihood values are smaller than their previous points. In those situations, TJEM will recover by switching back to AEM.

#### 5.4. Cluster Models

In this experiment, we show that TJEM is particularly effective for Cluster Models, which are AUTOCLASS-like Bayesian Networks [4] consisting of a latent cluster node with independent children as the features. We created a Cluster Model that clusters feature vectors with 50 binary features into 10 groups. We synthesized data sets for this model with each data set containing 1,000 examples. Again, we randomly removed 30%, 60% and 90% of the data in the feature vectors. Then we used TJEM, pEM, AEM and EM to fit the model with the data sets. It turns out that, as expected, pEM, AEM, and EM all requires more iterations to converge for more missing data, but TJEM not only requires less iterations, the number of iterations required to converge nearly stays approximately the same regardless of the proportion of missing data. For example, pEM(1.2) and AEM required averagely 14 and 10 iterations respectively to converge with 60% missing rate, and 49 and 21 iterations respectively with 90% missing rate, but TJEM took only five iterations for almost all cases, regardless of the missing rates. Figure 7 shows the learning curves of two cases with 90% missing rate. The figure shows that it only requires a “triple jump” for TJEM to reached the local maximum.

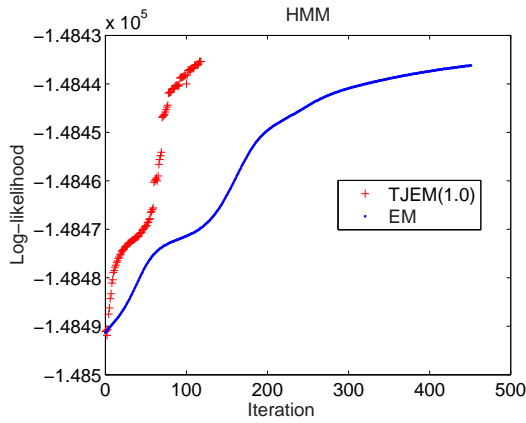
### 6. Conclusion and Future Work

This paper presents a new framework for accelerating the EM algorithm by performing “triple jump” on the surface of log-likelihood of data. This new framework can also be applied to other bound optimization methods. Experimental results show that our method performs well especially when the data set is sparse or less structured, which are common in real world applications but EM is slow in those situations. The results also show that our method is particularly effective for Cluster Models. An explanation of its effectiveness is that the log-likelihood surface for Cluster Models has a shape that exactly matches the assumption made by the triple jump extrapolation. However, it requires further investigation to identify its real cause. The search rarely switches back to regular or parameterized EM because the learning rates of sub-vectors are updated independently.

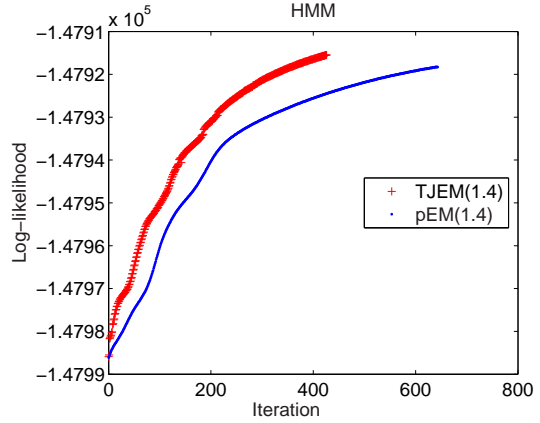
Our future work includes performing experiments for other bound optimization methods with large real world data sets. In this paper, we focus on Aitken-based approach to accelerating EM. Another approach to accelerating EM is gradient-based. Ortiz and Kaelbling [9] showed that for Mixtures of Gaussians, when the Gaussians are not well separated, the conjugate-gradient method can provide more significant acceleration. Salakhutdinov et al. [12] presented a new conjugate gradient method that switches between regular EM according to the missing information. We will empirically compare the convergence rate of our framework with their conjugate-gradient method as another future work.

### References

- [1] E. Bauer, D. Koller, and Y. Singer. Update rules for parameter estimation in Bayesian networks. In *Proc. of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 3–13, 1997.
- [2] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.
- [3] R. L. Burden and D. Faires. *Numerical Analysis*. PWS-KENT Pub Co., 1988.
- [4] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A Bayesian classification system. In *Proceedings of the 5th International Conference on Machine Learning*, pages 54–56, 1988.
- [5] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.
- [6] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal statistical Society*, B39:1–37, 1977.
- [7] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Letters to Nature*, 401:788–791, 1999.
- [8] O. Luis and K. Leslie. Accelerating EM: An empirical study. In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 512–521, 1999.
- [9] L. Ortiz and K. Leslie. Notes on methods based on maximum-likelihood estimation for learning the parameters of the mixture of Gaussians model.
- [10] R. Salakhutdinov and S. Roweis. Adaptive overrelaxed bound optimization methods. In *ICML2003*, pages 664–671, 2003.
- [11] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. On the convergence of bound optimization algorithms. In *UAI2003*, pages 509–516, 2003.
- [12] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *ICML2003*, pages 672–679, 2003.
- [13] A. Yuille and A. Rangarajan. The convex-concave computational procedure (CCCP). In *Advances in Neural Information Processing Systems*, volume 13, 2001.

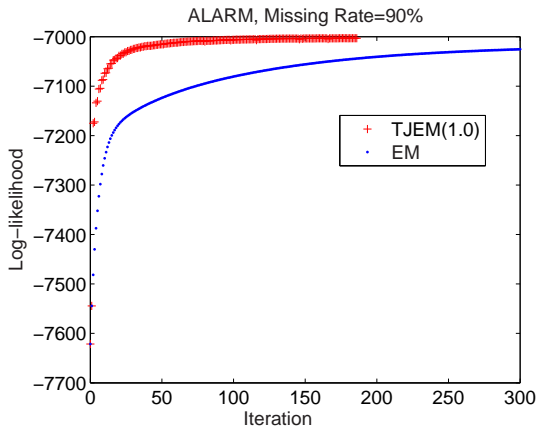


(a) Learning curves of TJEM(1.0) and EM

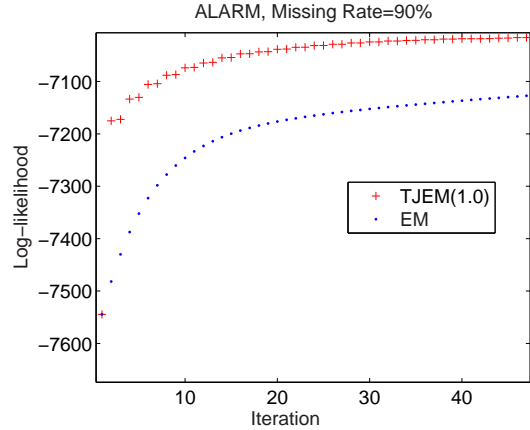


(b) Learning curves of TJEM(1.4) and pEM(1.4)

**Figure 2. Comparing TJEM with EM and pEM(1.4) for training HMMs.**

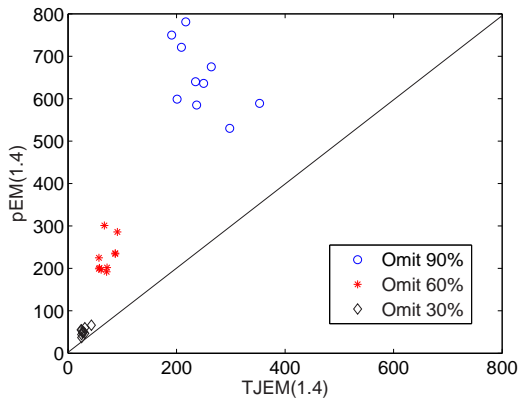


(a) Learning curves of TJEM(1.0) and EM

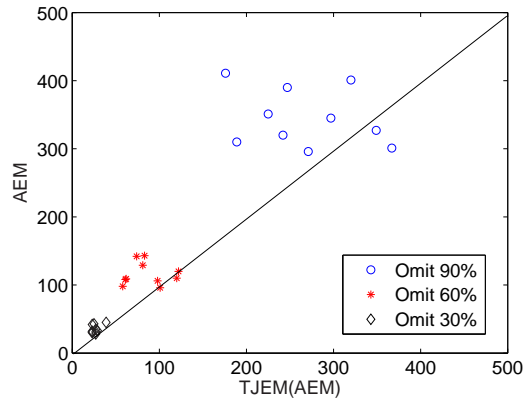


(b) Zoom-in view of early iterations

**Figure 3. Learning curves of TJEM and EM for training a large BN with sparse data.**

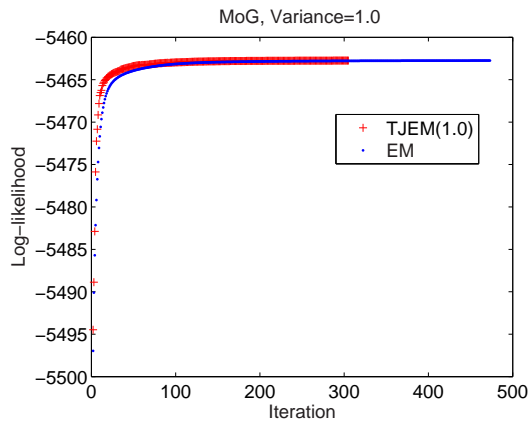


(a) TJEM(1.4) vs. pEM(1.4)

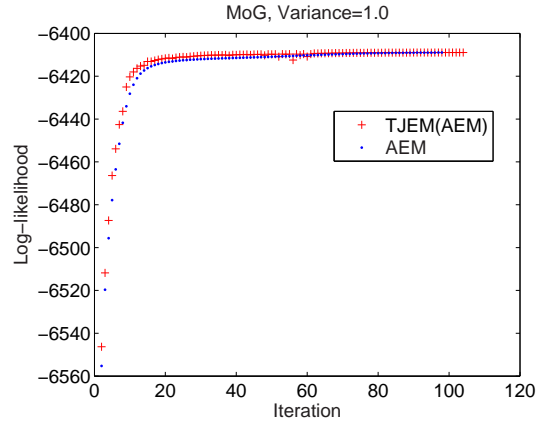


(b) TJEM(AEM) vs. AEM

**Figure 4. Scatter plots that compare the iterations required to converge for TJEM, pEM and AEM for data sets with different proportions of missing data.**

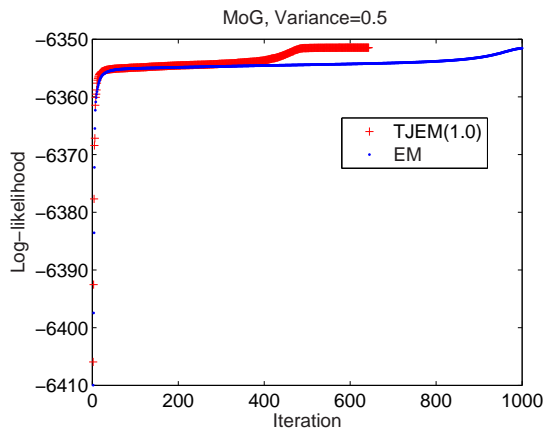


(a) Learning curves of TJEM(1.0) and EM

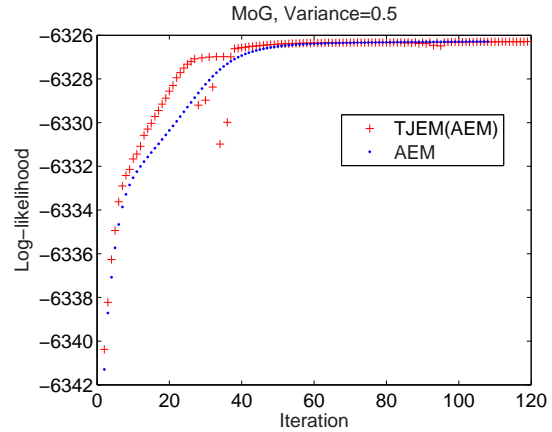


(b) Learning curves of TJEM(AEM) and AEM

**Figure 5. Comparing TJEM to EM and AEM for the Mixtures of Gaussians Model with variance 1.0, representing a structured, well separated case.**

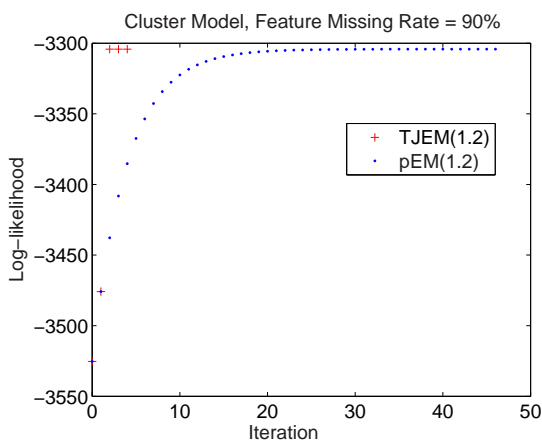


(a) Learning curves of TJEM(1.0) and EM

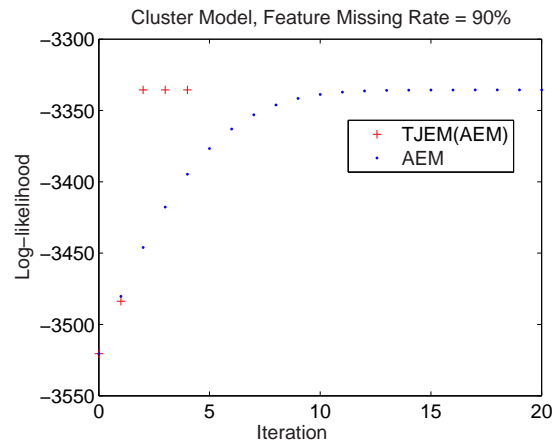


(b) Learning curves of TJEM(AEM) and AEM

**Figure 6. Comparing TJEM to EM and AEM for the Mixtures of Gaussians Model with variance 0.5, representing a less structured case.**



(a) Learning curves of TJEM(1.2) and pEM(1.2)



(b) Learning curves of TJEM(AEM) and AEM

**Figure 7. Learning curves that compare TJEM(1.2) to pEM(1.2) and TJEM(AEM) to AEM for training a Cluster Model with sparse data.**