Mach Learn (2014) 96:155–188
DOI 10.1007/s10994-013-5410-3

A comparison of collapsed Bayesian methods for probabilistic finite automata

Chihiro Shibata · Ryo Yoshinaka

Received: 11 December 2012 / Accepted: 14 August 2013 / Published online: 3 October 2013

Abstract This paper describes several collapsed Bayesian methods, which work by first marginalizing out transition probabilities, for inferring several kinds of probabilistic finite automata. The methods include collapsed Gibbs sampling (CGS) and collapsed variational Bayes, as well as two new methods. Their targets range over general probabilistic finite automata, hidden Markov models, probabilistic deterministic finite automata, and variable-length grams. We implement and compare these algorithms over the data sets from the Probabilistic Automata Learning Competition (PAutomaC), which are generated by various types of automata. We report that the CGS-based algorithm designed to target general probabilistic finite automata performed the best for any types of data.

Keywords Collapsed Gibbs sampling · Variational Bayesian methods · State-merging algorithms

1 Introduction

Since Hidden Markov Models (HMMs) are implemented in many applications, many inference methods for them have thus far been proposed and refined. It is difficult to find transition probabilities that maximize the generation probability of training samples. It is also intractable to marginalize out state transition probabilities and simultaneously sum them with respect to hidden variables. Therefore, some approximation and/or searching-local-optima technique is required. The Expectation Maximization (EM) algorithm, called Baum-Welch, is the most well-known classic method that is used as a statistical method for

Editors: Jeffrey Heinz, Colin de la Higuera, and Tim Oates.

C. Shibata (✉)
School of Computer Science, Tokyo University of Technology, Tokyo, Japan
e-mail: shibatachh@stf.teu.ac.jp

R. Yoshinaka
Graduate School of Informatics, Kyoto University, Kyoto, Japan
e-mail: ry@i.kyoto-u.ac.jp

the HMM learning. Recently, beyond EM, many statistical approaches have been developed and applied for inferring HMMs, such as *Collapsed Gibbs Sampling (CGS)* (Goldwater and Griffiths 2007), *Variational Bayes (VB)* (Beal 2003), and spectral methods (Hsu et al. 2009). CGS is a special form of Gibbs sampling (Bishop 2006), where only hidden variables are sampled after transition probabilities are marginalized. VB approximates all parameters, namely, transition probabilities and probabilities of hidden states, to be independent. CGS is considered to be one of the best choices for the HMM inference as compared empirically to EM and VB (Gao and Johnson 2008).

The recent Bayesian methods have also been applied to various probabilistic models. For instance, Johnson et al. (2007) applied CGS to PCFGs in the Chomsky Normal Form. Liang et al. (2007) applied VB inference for infinite PCFGs, where an infinite number of nonterminal symbols and rules can be modeled by assuming that their priors are represented by a Hierarchical Dirichlet Process (HDP) (Teh et al. 2006a). Pfau et al. (2010) applied the Metropolis-Hastings algorithm (Bishop 2006) for Probabilistic Deterministic Infinite Automata (PDIAs), where graph structures of PDFAs are generated from a variant of HDP (Teh 2006). Their algorithm can be thought of as a method that samples PDFAs by iterating merging and splitting states randomly in the Bayesian manner.

For a probabilistic topic model called Latent Dirichlet Allocation (LDA), which is used in natural language processing, Teh et al. (2006b) proposed a method called *Collapsed Variational Bayes (CVB)*. They used a VB approximation after integrating out transition probabilities, and showed that their method yielded a more accurate result than the standard VB method. CVB has variables, each of which represents a probability that the automaton is in a particular state at a certain time. These variables are assumed to be independent of each other. We update these variables so as to minimize the KL-divergence between the approximated and the true marginal probability. However, since it is still difficult to update variables so as to minimize the KL-divergence exactly, a further approximation is applied to each update. While Teh et al. (2006b) used the second order Taylor approximation when updating the independent approximation of the probability of each hidden variable, Asuncion et al. (2009) found that the zeroth order Taylor approximation (called *CVB0*) is empirically sufficient to achieve good accuracy for LDAs.

In this paper, our targets for learning are the class of probabilistic finite automata (PFAs) and their special cases. We call an inference method *collapsed* if the transition probabilities of the PFAs are integrated out before some approximations or sampling methods are applied. This paper introduces and describes several collapsed inference methods for PFAs and evaluates them. Moreover, we compare collapsed methods that target other subclasses of PFAs, such as HMMs, PDFAs, and variable-length grams (VGrams). We say that a PFA is *fully connected* if one can move from every state to every state using any symbol. In Sects. 2.1–2.4, we discuss how existing techniques of CGS and CVB0 can be applied to the inference of PFAs. We describe and compare the computational cost for CGS, CVB0, and CVB2.

In Sect. 2.5 and Sect. 3, we propose two different approaches, which are modifications of CVB0 and CGS. In Sect. 2.5, we propose a variant of CVB0, which we call GCVB0, for which a convergence property is guaranteed. The standard CVB0 does not have this nice property, since it uses Taylor approximations when updating variables. We modify CVB0 to have the convergence property by defining a global function that approximates the KL-divergence. Variables are updated using existing techniques, such as quasi-Newton methods. In Sect. 3, we introduce a simple generative model for PFAs that are not fully connected, for which a CGS algorithm is presented. In addition to the sequence of hidden states, graph structures of PFAs are also sampled.

Abe and Warmuth (1992) showed that PFAs are KL-PAC learnable from samples of size polynomial in the number of the states and letters and the sample size using maximum-likelihood estimation. They also showed that the actual computational cost must be prohibitively expensive unless RP = NP. Kearns et al. (1994) showed that learning PDFAs even over 2-letter alphabets is as hard as a problem for which no polynomial algorithm is known. On the other hand, Clark and Thollard (2004) proposed an algorithm that PAC learns PDFAs that satisfy μ-*distinguishability* in polynomial time from a polynomial amount of data. Some elaborations of his algorithm have also been proposed (Castro and Gavaldà 2008; Balle et al. 2013). On the other hand, no solid work has been done on the computational cost of techniques generically called MCMC, including Gibbs sampling, which infer the correct posterior distribution in the limit.

Our experimental results are presented in Sect. 4. We compare the inference methods described in the preceding sections as well as other collapsed Bayesian methods for special kinds of PDFAs, including HMMs, PDFAs, and VGrams. Experimental results for PAutomaC data sets[1] showed that CGS performed better than other methods in terms of accuracy. Although GCVB0 is guaranteed to converge to some local optimal point, and thus it is clear at which point its iterations should be stopped, GCVB0 yielded results worse than those of CVB0 and CGS.

PAutomaC data sets were generated by different types of PFAs, including HMMs. CGS-HMM is a modification of our CGS algorithm for PFAs such that it targets HMMs. From the comparison of CGS-PFA and CGS-HMM, it appears that CGS-PFA yields better scores than does CGS-HMM, since CGS-HMM often fails to find appropriate emission probabilities η and state transition probabilities θ that can factorize the transition probability ξ. We also compared CGS-PFA with other collapsed methods for other models, which are actually special cases of PFAs. However, CGS-PFA yields better scores than any of these methods. Therefore, we conclude that CGS-PFA is empirically the best choice among the collapsed methods described in this paper.

A drawback of CGS is its rather high computational cost. In Sect. 4.6, we measure empirically the relation between the computational cost and accuracy of CGS and other classic methods, including a state-merging method based on marginal probability. The computational costs of CGS and the state-merging method have a gap of one to three orders of magnitude. The actual computational cost of CGS depends on the number of iterations where variables are resampled. The sampling process should be repeated until the sampled distribution converges. Our implementation set the iteration number to 20,000 for every problem, which seems unnecessarily large for many problems. However, this number is in fact not too large; we observed that 200 iterations, for example, are too few to make the empirical distribution converge.

2 Collapsed Bayesian approaches for fully connected PFA

2.1 Probabilistic model for fully connected NFAs

A *probabilistic finite automaton (PFA)* is a nondeterministic finite automaton G in which transition probabilities ξ are assigned. We call G *the underlying automaton* and the strings accepted by G *sentences*. A PFA assigns a probability to each sentence according to ξ. A PFA is seen as a machine that generates strings according to these transition probabilities.

[1] PAutomaC (http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/).

We would like to infer a PFA from given sentences $\mathbb{o}_1, \ldots, \mathbb{o}_m$ generated by a PFA of our learning target. For technical convenience, we introduce a special letter, 0, which represents the end of sentences. We assume that the end marker 0 leads the machine to its unique initial state 0 to prepare to generate a next sentence, and that the machine goes into the initial state if and only if one sentence has been generated. This premise allows us to treat the given sentences $\mathbb{o}_1, \ldots, \mathbb{o}_m$ as a single sentence $\mathbb{o}$:

$$\mathbb{o} = \mathbb{o}_1 0 \ldots \mathbb{o}_m 0.$$

The probability that the PFA generates a sentence $\mathbb{o} = a_1 \ldots a_T$ by passing states $\mathbb{z} = z_1 \ldots z_{T+1}$ in this order is given as

$$\Pr(\mathbb{o}, \mathbb{z} \mid \xi, G) = \prod_{1 \le t \le T} \xi(z_t, a_t, z_{t+1}), \tag{1}$$

where each a_t is a letter, z_t is a state, and $\xi(i, a, j)$ is the probability assigned to the transition rule that changes the state from i to j, emitting the letter a. We note that $\mathbb{z}$ contains one more occurrence of 0 than $\mathbb{o}$ does. In particular, $z_1 = z_{T+1} = 0$. We will concisely write ξ_{iaj} for $\xi(i, a, j)$ hereafter. As a probability, ξ must satisfy that $0 \le \xi_{iaj} \le 1$ and $\sum_{a,j} \xi_{iaj} = 1$ for all states i. Moreover, ξ must satisfy that for all i,

$$\xi_{i0j} = 0 \quad \text{for all } j \neq 0,$$
$$\xi_{ia0} = 0 \quad \text{for all } a \neq 0,$$

in accordance with the special roles of the end marker and the initial state.

Without loss of generality, in this section, we assume the underlying automaton G to be a *fully connected* NFA. In a fully connected NFA, every letter a may induce transition from every state i to every state j, except when $a = 0$ or $j = 0$. That is, ξ_{iaj} may have non-zero value for every i, $a(\neq 0)$ and $j(\neq 0)$. Any PFAs whose underlying automata have sparse edges, including DFAs as a special case, can be represented or at least well approximated with the fully connected NFA by assigning 0 or a very small probability to some edges. We will discuss another approach, which infers sparse PFAs, in Sect. 3. We fix the number of states of G to be $N + 1$ and denote the states by natural numbers $0, 1, \ldots, N$. Let A be the size of the letter alphabet including 0. In the sequel, we suppress G.

In addition, we assume that the conjugate prior of ξ is represented as

$$\Pr(\xi) = \frac{1}{R(\beta)} \prod_{i, a \neq 0, j \neq 0} \xi_{iaj}^{\beta - 1} \xi_{i00}^{N\beta - 1}, \quad \text{where } R(\beta) = \left(\frac{\Gamma(\beta)^{N(A-1)} \Gamma(N\beta)}{\Gamma(NA\beta)} \right)^{N+1}, \tag{2}$$

where Γ is the *gamma function*, the extension of the factorial function for real numbers. $\beta(> 0)$ is called a *hyperparameter* of $\Pr(\xi)$, and is often chosen to be smaller than 1.

2.2 Feasible and unfeasible marginalization

It is known to be unfeasible to calculate $\Pr(\mathbb{o}) = \sum_{\mathbb{z}} \int_{\xi} \Pr(\mathbb{o}, \mathbb{z}, \xi)$ based on the definition of $\Pr(\mathbb{o}, \mathbb{z} \mid \xi)$ and $\Pr(\xi)$ given as Eqs. (1) and (2). The following table shows which combination of parameters $\mathbb{o}$, $\mathbb{z}$, and ξ makes it feasible or unfeasible to compute the joint and/or conditional probability.

$\Pr(\xi, \mathbb{z})$ and $\Pr(\mathbb{o}, \xi)$ can be calculated in a dynamic programming manner since Eq. (1) has the form $\prod_t f_t(z_t, a_t, z_{t+1})$. $\Pr(\mathbb{o}, \mathbb{z})$ is calculated as

Definition 1 (delta function)

$$\delta\begin{pmatrix} x_1, \ldots, x_n \\ y_1, \ldots, y_n \end{pmatrix} = \begin{cases} 1 & \text{if } x_i = y_i \text{ for all } i, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2 (counting functions) C is a function mapping $\mathbb{o}, \mathbb{z}$ to natural numbers defined by

$$C(\mathbb{o}, \mathbb{z})_i = \sum_t \delta\begin{pmatrix} z_t \\ i \end{pmatrix}, \qquad C(\mathbb{o}, \mathbb{z})_{iaj} = \sum_t \delta\begin{pmatrix} z_t, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix}.$$

That is, $C(\mathbb{o}, \mathbb{z})_i$ represents how many times the state i is visited and $C(\mathbb{o}, \mathbb{z})_{iaj}$ is the number of times that the machine changes the state from i to j with a. Understanding that the arguments of C are $\mathbb{o}$ and $\mathbb{z}$, let us write C instead of $C(\mathbb{o}, \mathbb{z})$. Using counting functions, we can rewrite Eq. (1) as

$$\Pr(\mathbb{o}, \mathbb{z} \mid \xi) = \prod_i \Bigg(\prod_{a\neq 0, j\neq 0} \xi_{iaj}^{C_{iaj}} \Bigg) \xi_{i00}^{C_{i00}}. \tag{3}$$

By integrating ξ out,

$$\begin{aligned} \Pr(\mathbb{o}, \mathbb{z}) &= \int \Pr(\mathbb{o}, \mathbb{z} \mid \xi) \Pr(\xi) d\xi \\ &= \int \prod_i \Bigg(\prod_{a\neq 0, j\neq 0} \xi_{iaj}^{C_{iaj}+\beta-1} \Bigg) \xi_{i00}^{C_{i00}+N\beta-1} d\xi \, \frac{1}{R(\beta)} \\ &= \prod_i \frac{(\prod_{a\neq 0, j\neq 0} \Gamma(C_{iaj}+\beta))\Gamma(C_{i00}+N\beta)}{\Gamma(C_i + NA\beta)} \frac{1}{R(\beta)}. \end{aligned} \tag{4}$$

However, it appears computationally hard to sum up Eq. (4) with all possible combinations of $\mathbb{o}$ or $\mathbb{z}$ to obtain $\Pr(\mathbb{z})$ or $\Pr(\mathbb{o})$, respectively (Beal 2003).

It is often the case that inferring a PFA from $\mathbb{o}$ is a means of obtaining a probability prediction of future sentences, although inferring a specific PFA is not the only way to fulfill the latter purpose. Since we have fixed the underlying machine to be a fully connected NFA with N states, the inference of a PFA is reduced to that of ξ. According to a Bayesian approach, which this study uses, this amounts to estimating $\Pr(\xi|\mathbb{o})$. The probability of a future sentence $\mathbb{b}$ is represented by $\Pr(\mathbb{b}|\mathbb{o})$, on the other hand. Thus, the computation of $\Pr(\xi|\mathbb{o})$ and $\Pr(\mathbb{b}|\mathbb{o})$ is our central concern, which is, however, unfeasible. The difficulty of computing $\Pr(\xi|\mathbb{o})$ and $\Pr(\mathbb{b}|\mathbb{o})$ can be reduced to the infeasibility of the calculation of $\Pr(\mathbb{o})$. Computing $\Pr(\mathbb{b}|\mathbb{o})$ in general is obviously harder than computing $\Pr(\mathbb{o}) = \Pr(\mathbb{o}|\epsilon)$, where ϵ denotes the empty sequence. One can compute $\Pr(\mathbb{o})$ using $\Pr(\xi|\mathbb{o})$ by $\Pr(\mathbb{o}) = \Pr(\xi, \mathbb{o})/\Pr(\xi|\mathbb{o})$, where $\Pr(\xi, \mathbb{o})$ can easily be obtained by dynamic programming.

Therefore, we necessarily have to use some approximation to achieve the above two purposes. In the following sections, we use two approximations obtained by random algorithms, which are known as *collapsed Gibbs sampling (CGS)* and *collapsed variational Bayesian method (CVB)*. We also give a simple variant method of CVB that converges to a local optimal point, whereas, in general, CVB has no guarantee of convergence to a local optimal point.

2.3 Collapsed Gibbs sampling for the inference of PFAs—CGS-PFA

For a sequence of variables $\mathbb{x} = x_1, \ldots, x_T$, let $\mathbb{x}^{\neg t}$ denote the sequence obtained from $\mathbb{x}$ by removing x_t, i.e., $\mathbb{x}^{\neg t} = x_1 \cdots x_{t-1} x_{t+1} \cdots x_T$. Gibbs sampling and variational Bayes are approximation methods for a difficult-to-compute joint distribution $\Pr(x_1, \ldots, x_k)$, where, on the other hand, the conditional one, $\Pr(x_t \mid \mathbb{x}^{\neg t})$, for each t is easily obtained. Gibbs sampling first arbitrarily initializes $\mathbb{x}^{(0)} = x_1^{(0)} \ldots x_T^{(0)}$, and then, repeats the following procedure for $k = 1, 2, \ldots$, where $x^{(k)}$ is computed from $x^{(k-1)}$.

- For $t = 1, \ldots, T$ in this order, we sequentially sample $x_t^{(k)}$ according to the probability $\Pr(x_t \mid x_1^{(k)}, \ldots, x_{t-1}^{(k)}, x_{t+1}^{(k-1)}, \ldots, x_T^{(k-1)})$.
- We have obtained a sample $\mathbb{x}^{(k)} = x_1^{(k)} \ldots x_T^{(k)}$.

We refer to each repetition computing $\mathbb{x}^{(k)}$ from $\mathbb{x}^{(k-1)}$ as a *single iteration*. Since Gibbs sampling is a Markov chain Monte Carlo method, $\mathbb{x}$, as the random variable, converges to $\Pr(\mathbb{x})$ in the limit, whereas greedy algorithms, such as the Baum-Welch and variational Bayesian methods, may converge to a local optimal point. In an actual implementation, we usually discard many samples in order to make the sample distribution closer to the true probability distribution $\Pr(\mathbb{x})$. First, we discard samples $\mathbb{x}^{(k)}$ from the early iterations for $k = 1, \ldots, b$, which are called the *burn-in periods*, since they strongly depend on the initial value. Second, we take values periodically; for example, we use only every 100th value after the burn-in period to make the respective samples (almost) independent of each other. For details of Gibbs sampling, see, e.g., Chap. 11 of Bishop (2006).

Our approach for obtaining an approximation of $\Pr(\mathbb{b}|\mathbb{a})$ is based on the formula

$$\Pr(\mathbb{b}|\mathbb{a}) = \int \Pr(\mathbb{b}|\xi) \Pr(\xi|\mathbb{a}) d\xi. \tag{5}$$

Instead of integrating $\Pr(\mathbb{b}|\xi) \Pr(\xi|\mathbb{a})$ with respect to ξ, we obtain many concrete values $\widetilde{\xi}^{(1)}, \ldots, \widetilde{\xi}^{(S)}$ of ξ that are "plausible" in terms of $\mathbb{a}$, and approximate the target probability by Rao-Blackwellization:

$$\Pr(\mathbb{b}|\mathbb{a}) = \int \Pr(\mathbb{b}|\xi) \Pr(\xi|\mathbb{a}) d\xi \approx \frac{1}{S} \sum_{s=1}^{S} \Pr\bigl(\mathbb{b}|\widetilde{\xi}^{(s)}\bigr). \tag{6}$$

The computation of $\Pr(\mathbb{b}|\widetilde{\xi}^{(s)})$ is trivial if the value $\widetilde{\xi}^{(s)}$ is obtained. Following Teh et al. (2006b) and others, we first collect different values $\mathbb{z}^{(1)}, \ldots, \mathbb{z}^{(S)}$ by Gibbs sampling. While $\mathbb{z}$ and ξ are dependent, marginalizing out ξ beforehand allows us to treat $\mathbb{z}$ only. This technique is called *Collapsed Gibbs Sampling (CGS)*, and we call our particular algorithm that is applied to PFAs *CGS-PFA*. We then determine values $\widetilde{\xi}^{(1)}, \ldots, \widetilde{\xi}^{(S)}$ simply as the expectation[2]

$$\widetilde{\xi}_{iaj}^{(s)} = \mathrm{E}\bigl[\xi_{iaj} \bigm| \mathbb{a}, \mathbb{z}^{(s)}\bigr] = \frac{C_{iaj} + \beta}{C_i + AN\beta}.$$

Algorithm 1 describes our inference method CGS-PFA.

[2]It would be possible to collect values $\widetilde{\xi}^{(s,1)}, \ldots, \widetilde{\xi}^{(s,S_s)}$ again by Gibbs sampling for each $\mathbb{z}^{(s)}$ instead of taking the expectation $\widetilde{\xi}^{(s)}$; however, this alternative is too computationally expensive.

Algorithm 1: CGS-PFA

Input : $\mathbb{o}$, N, BurnInTime, SamplingNum, Period
Output: $\widetilde{\xi}^{(1)}, \cdots, \widetilde{\xi}^{(\text{SamplingNum})}$

1 Initialize $\mathbb{z}$ randomly (if $a_t = 0$, z_t is fixed to 0.);
2 **foreach** $t \in [0, \text{BurnInTime}]$ **do**
3 `SingleIteration();`
4 **end**
5 **foreach** $s \in [1, \text{SamplingNum}]$ **do**
6 **foreach** $t \in [1, \text{Period}]$ **do**
7 `SingleIteration();`
8 **end**
9 Output $\widetilde{\xi}^{(s)}$ defined by $\widetilde{\xi}^{(s)}_{iaj} = \frac{C_{iaj}+\beta}{C_i+AN\beta}$ for all i, a, j;
10 **end**
11 ;
12 `SingleIteration(){`
13 **foreach** $t \in [0, T]$ **do**
14 **if** $a_t \neq 0$ **then**
15 Let $z_t = k$, where k is randomly chosen by the probability
$\Pr(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t}) = \frac{g_k^{(t)}}{\sum_i g_i^{(t)}}$, where $g_k^{(t)}$ is given by Eq. (9);
16 **end**
17 **end**
18 }

The key of CGS-PFA is the computation of $\Pr(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t})$ (Line 15), whose computational cost is, in fact, very low. Observe that the computation of $\Pr(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t})$ is reduced to that of $\Pr(z_t = i, \mathbb{z}^{\neg t}, \mathbb{o})$ for each i by

$$\Pr\left(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t}\right) = \frac{\Pr(z_t = k, \mathbb{z}^{\neg t}, \mathbb{o})}{\sum_i \Pr(z_t = i, \mathbb{z}^{\neg t}, \mathbb{o})}.$$

The probability $\Pr(z_t = i, \mathbb{z}^{\neg t}, \mathbb{o})$ can be calculated according to Eq. (4). Updating z_t affects only a limited number of counters C_i and C_{iaj}, each of which will be changed by at most 2. The other factors remain unchanged. Due to the nice property of the gamma function, $\Gamma(x) = (x-1)\Gamma(x-1)$, updating Eq. (4) can be done in constant time. Therefore, our algorithm runs efficiently enough. In fact, we do not compute the value $\Pr(z_t = i, \mathbb{z}^{\neg t}, \mathbb{o})$ for each i. As we discuss in detail below, we compute $g_i^{(t)}$ such that $\Pr(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t}) = g_k^{(t)} / \sum_i g_i^{(t)}$.

Let $C^{z_t=k}$ be the counting function, where z_t in $\mathbb{z}$ is assumed to take the state k. That is,

$$C_i^{z_t=k} = C_i^{\neg t} + \delta\begin{pmatrix} k \\ i \end{pmatrix},$$

$$C_{iaj}^{z_t=k} = C_{iaj}^{\neg t} + \delta\begin{pmatrix} k, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix} + \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & k \\ i, & a, & j \end{pmatrix},$$

where

$$C_i^{\neg t} = C_i - \delta\begin{pmatrix} z_t \\ i \end{pmatrix},$$

$$C_{iaj}^{\neg t} = C_{iaj} - \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & z_t \\ i, & a, & j \end{pmatrix} - \delta\begin{pmatrix} z_t, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix}.$$

Let us define a number g_k to satisfy the equation

$$\begin{aligned}\Pr(z_t = k, \mathbb{z}^{\neg t}, \mathbb{o}) &= \prod_i \frac{(\prod_{a\neq 0, j\neq 0} \Gamma(C_{iaj}^{z_t=k} + \beta))\Gamma(C_{i00}^{z_t=k} + N\beta)}{\Gamma(C_i^{z_t=k} + NA\beta)} \frac{1}{R(\beta)} \quad \text{(by Eq. (4))} \\ &= g_k^{(t)} \prod_i \frac{(\prod_{a\neq 0, j\neq 0} \Gamma(C_{iaj}^{\neg t} + \beta))\Gamma(C_{i00}^{\neg t} + N\beta)}{\Gamma(C_i^{\neg t} + NA\beta)} \frac{1}{R(\beta)}. \end{aligned} \tag{7}$$

It should be noted that all the factors, except $g_k^{(t)}$ in Eq. (7), are independent of k, and hence, we have

$$\Pr\left(z_t = k \mid \mathbb{o}, \mathbb{z}^{\neg t}\right) = g_k^{(t)} \Big/ \sum_i g_i^{(t)}. \tag{8}$$

Thus, it is sufficient to calculate g_i for every i. In fact, $g_i^{(t)}$ in Eq. (7) has a simple form as seen below. The differences between $C^{z_t=k}$ and $C^{\neg t}$ are summarized as follows.

$$C_i^{z_t=k} = C_i^{\neg t} + \delta\begin{pmatrix} i \\ k \end{pmatrix}.$$

- If $(i, a, j) = (k, a_t, z_{t+1}) \neq (z_{t-1}, a_{t-1}, k)$,

$$C_{iaj}^{z_t=k} = C_{iaj}^{\neg t} + \delta\begin{pmatrix} k, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix} + \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & k \\ i, & a, & j \end{pmatrix} = C_{iaj}^{\neg t} + 1 = C_{ka_tz_{t+1}}^{\neg t} + 1.$$

- If $(i, a, j) = (z_{t-1}, a_{t-1}, k) \neq (k, a_t, z_{t+1})$,

$$C_{iaj}^{z_t=k} = C_{iaj}^{\neg t} + \delta\begin{pmatrix} k, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix} + \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & k \\ i, & a, & j \end{pmatrix} = C_{iaj}^{\neg t} + 1 = C_{z_{t-1}a_{t-1}k}^{\neg t} + 1.$$

- If $(i, a, j) = (z_{t-1}, a_{t-1}, k) = (k, a_t, z_{t+1})$,

$$C_{iaj}^{z_t=k} = C_{iaj}^{\neg t} + \delta\begin{pmatrix} k, & a_t, & z_{t+1} \\ i, & a, & j \end{pmatrix} + \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & k \\ i, & a, & j \end{pmatrix} = C_{iaj}^{\neg t} + 2 = C_{z_{t-1}a_{t-1}k}^{\neg t} + 2.$$

Since $\Gamma(x) = (x-1)\Gamma(x-1)$, we have

$$\text{if } a_{t-1} = 0, \qquad g_k^{(t)} = \delta\begin{pmatrix} k \\ 0 \end{pmatrix},$$

$$\text{if } a_{t-1} \neq 0 \text{ and } a_t = 0, \qquad g_k^{(t)} = \frac{(C_{k00}^{\neg t} + N\beta)(C_{z_{t-1}a_{t-1}k}^{\neg t} + \beta)}{(C_k^{\neg t} + NA\beta)},$$

$$\text{if } a_{t-1} \neq 0 \text{ and } a_t \neq 0, \quad g_k^{(t)} = \frac{(C^{\neg t}_{ka_t z_{t+1}} + \beta)\left(C^{\neg t}_{z_{t-1}a_{t-1}k} + \delta\begin{pmatrix} z_{t-1}, & a_{t-1}, & k \\ k, & a_t, & z_{t+1}\end{pmatrix} + \beta\right)}{(C^{\neg t}_k + NA\beta)}. \tag{9}$$

Since one can calculate g_k in constant time, the computational cost for a single iteration is just $\mathrm{O}(NT)$.

2.4 Collapsed variational Bayes approximations

Teh et al. (2006b) proposed an approximation method called Collapsed Variational Bayes (CVB) for inferring a probabilistic topic model called *Latent Dirichlet Allocation (LDA)*. In this subsection, we explain how their approach can be applied to the inference of PFAs. In a similar way to that described in Sect. 2.3, we first marginalize ξ out, so that the standard technique of Variational Bayes is applicable to computing $\Pr(\mathbb{z}|\mathbb{o})$. Thus, this approach is called *collapsed*.

A Variational Bayes method approximates $\Pr(\mathbb{z}|\mathbb{o})$ by a probability function $q(\mathbb{z})$ such that all z_t in $\mathbb{z}$ are independent, i.e., the probability function $q(\mathbb{z})$ satisfies that

$$q(\mathbb{z}) = \prod_t q_t(z_t).$$

The probability function q is optimized so that the KL divergence from $\Pr(\mathbb{z}|\mathbb{o})$

$$D_{\mathrm{KL}}\big(q(\mathbb{z}) \| \Pr(\mathbb{z}|\mathbb{o})\big) = -\sum_{\mathbb{z}} q(\mathbb{z}) \log \Pr(\mathbb{z}|\mathbb{o}) + \sum_{\mathbb{z}} q(\mathbb{z}) \log q(\mathbb{z})$$

will be minimum. From the optimum q, we let

$$\widetilde{\xi}_{iaj} = \mathrm{E}_q\left[\frac{C_{iaj} + \beta}{C_i + NA\beta}\right] \approx \frac{\mathrm{E}_q[C_{iaj}] + \beta}{\mathrm{E}_q[C_i] + NA\beta}, \tag{10}$$

based on which we obtain $\Pr(\mathbb{b}|\widetilde{\xi})$. Since $\log \Pr(\mathbb{z}|\mathbb{o}) = \log \Pr(\mathbb{z}, \mathbb{o}) - \log \Pr(\mathbb{o})$ and $\log \Pr(\mathbb{o})$ is constant with respect to q, minimizing the above KL divergence amounts to minimizing

$$D(q) = -\sum_{\mathbb{z}} q(\mathbb{z}) \log \Pr(\mathbb{z}, \mathbb{o}) + \sum_{\mathbb{z}} q(\mathbb{z}) \log q(\mathbb{z}). \tag{11}$$

The probability function q is optimized by updating $q_t(z_t)$ for each $t \in \{1, \ldots, T+1\}$, while all the other $q_s(z_s)$ with $s \neq t$ are fixed. Let $q^{\neg t}(\mathbb{z}^{\neg t}) = \prod_{s \neq t} q_s(z_s)$. We note that the expectation of a function $f(\mathbb{z})$ with a probability function q is given as

$$\mathrm{E}_q\big[f(\mathbb{z})\big] = \sum_{\mathbb{z}} q(\mathbb{z}) f(\mathbb{z})$$

by the independence assumption of q. Eq. (11) can be written as

$$\begin{aligned} D(q) &= -\mathrm{E}_q\big[\log \Pr(\mathbb{z}, \mathbb{o})\big] + \mathrm{E}_q\big[\log q(\mathbb{z})\big] \\ &= -\mathrm{E}_q\big[\log \Pr\big(z_t \mid \mathbb{z}^{\neg t}, \mathbb{o}\big)\big] + \mathrm{E}_q\big[\log q_t(z_t)\big] + \text{const. w.r.t. } q_t(z_t) \\ &= \sum_k \big(-q_t(k) c_k + q_t(k) \log q_t(k)\big) + \text{const.}, \end{aligned}$$

where $c_k = \mathrm{E}_{q^{\neg t}}[\log \Pr(z_t = k \mid \mathbb{z}^{\neg t}, \mathbb{o})]$, which equals $\mathrm{E}_q[\log \Pr(z_t = k \mid \mathbb{z}^{\neg t}, \mathbb{o})]$, since $\Pr(z_t = k \mid \mathbb{z}^{\neg t}, \mathbb{o})$ does not depend on q_t.

From Eq. (8), we have $c_k = \mathrm{E}_q[\log(g_k / \sum_i g_i)] = \mathrm{E}_q[\log(g_k)] - \mathrm{E}_q[\log(\sum_i g_i)]$, where g_i is given by Eq. (9). By Lagrange's multiplier, $q_t(1), \ldots, q_t(N)$, which minimize $D(q)$, are given as

$$q_t(k) = \frac{\exp(c_k)}{\sum_i \exp(c_i)} = \frac{\exp(\mathrm{E}_q[\log(g_k)])}{\sum_i \exp(\mathrm{E}_q[\log(g_i)])}. \tag{12}$$

Hence, it is enough to calculate $\mathrm{E}_q[\log(g_k)]$. If q_t is updated accurately by Eq. (12), $D(q)$ monotonically decreases, and consequently, q converges to some local minimal point, since we always have $D(q) \geq \log \Pr(\mathbb{o})$. This is the basic idea of CVB. The calculation is, however, not easy.

For two functions f and g such that the value of $\mathrm{E}_q[f(g(\mathbb{z}))]$ is difficult but $\mathrm{E}_q[g(\mathbb{z})]$ is easy to calculate, Teh et al. (2006b) proposed approximating $\mathrm{E}_q[f(g(\mathbb{z}))]$ by the Taylor expansion around $\mathrm{E}_q[g(\mathbb{z})]$. The Taylor expansion of $f(g(\mathbb{z}))$ around $\mathrm{E}_q[g(\mathbb{z})]$ is given as

$$\mathrm{E}_q\big[f\big(g(\mathbb{z})\big)\big] = f\big(\mathrm{E}_q\big[g(\mathbb{z})\big]\big) + 0 + f''\big(\mathrm{E}_q\big[g(\mathbb{z})\big]\big)\mathrm{V}_q\big[g(\mathbb{z})\big] + \cdots, \tag{13}$$

where V denotes the variation. The zeroth and first order Taylor approximations result in the identical formula $f(\mathrm{E}_q[g(\mathbb{z})])$. The CVB method with the zeroth or first-order approximation is called CVB0 (Asuncion et al. 2009).[3] By using the 0th order approximation $\mathrm{E}_q[\log(g_k)]$, from Eq. (9), if $a_{t-1} \neq 0$ and $a_t \neq 0$, we have

$$\begin{aligned}
\mathrm{E}_q\big[\log(g_k)\big] &= \mathrm{E}_q\big[\log\big(C^{\neg t}_{ka_t z_{t+1}} + \beta\big)\big] + \mathrm{E}_q\left[\log\left(C^{\neg t}_{z_{t-1}a_{t-1}k} + \delta\begin{pmatrix} k, a_t, z_{t+1} \\ z_{t-1}, a_{t-1}, k \end{pmatrix} + \beta\right)\right] \\
&\quad - \mathrm{E}_q\big[\log\big(C^{\neg t}_k + NA\beta\big)\big] \\
&\approx \log\big(\mathrm{E}_q\big[C^{\neg t}_{ka_t z_{t+1}}\big] + \beta\big) + \log\left(\mathrm{E}_q\left[C^{\neg t}_{z_{t-1}a_{t-1}k} + \delta\begin{pmatrix} k, a_t, z_{t+1} \\ z_{t-1}, a_{t-1}, k \end{pmatrix}\right] + \beta\right) \\
&\quad - \log\big(\mathrm{E}_q[C^{\neg t}_k] + NA\beta\big).
\end{aligned} \tag{14}$$

$\mathrm{E}_q[\log(g_k)]$ can be calculated similarly for the other cases where $a_{t-1} = 0$ and where $a_{t-1} \neq 0$ and $a_t = 0$.

The remaining task is to calculate the three terms on the right hand side of Eq. (14) for PFAs. The existing methods cannot be applied to this task, and therefore, we introduce our own approach.

2.4.1 Treatments of expectations over q

Definition 3 Let $x_1, x_2, y_1, y_2 \in Z \cup H$, where Z is the set of random variables $z_1, \ldots, z_{T+1}$ and H is the set of states. We write $(x_1, y_1) \sim (x_2, y_2)$ iff

$$\{x_1, y_1\} \cap \{x_2, y_2\} \cap Z \neq \varnothing.$$

[3] Whereas Teh et al.'s original work uses the second-order approximation (let us call it CVB2), we use CVB0, since the computational cost of CVB2 for PFA is N times more than that of CVB0 (see Appendix), and it has been reported that CVB0 often outperforms CVB2 with respect to accuracy for Latent Dirichlet Allocation (LDA) (Asuncion et al. 2009).

That is, if (x_1, y_1) and (x_2, y_2) have a common element, which is z_k for some k, we have $(x_1, y_1) \sim (x_2, y_2)$. The relation $\sim$ is obviously symmetric.

Let $x_1, \ldots, x_n, y_1, \ldots, y_n \in Z \cup H$ and $R = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. We define $\delta(R)$, $Z(R)$ and $H(R)$ as

- $\delta(R) = \delta\binom{x_1,\ldots,x_n}{y_1,\ldots,y_n}$,
- $Z(R) = \{x_1, \ldots, x_n, y_1, \ldots, y_n\} \cap Z$,
- $H(R) = \{x_1, \ldots, x_n, y_1, \ldots, y_n\} \cap H$.

Lemma 1 *Let $\sim_R$ be the equivalence closure of $\sim$ in R, which partitions R into disjoint subsets $R_1, \ldots, R_m$. We then have*

$$\mathrm{E}_q\big[\delta(R)\big] = \mathrm{E}_q\big[\delta(R_1)\big] \cdots \mathrm{E}_q\big[\delta(R_m)\big]$$

and for each h,

$$\mathrm{E}_q\big[\delta(R_h)\big] = \sum_i \Bigg(\prod_{z_t \in Z(R_h)} q_t(i) \prod_{k \in H(R_h)} \delta\binom{i}{k} \Bigg).$$

The proof is straightforward. It should be noted that if a function f of $\mathbb{z}$ depends on n elements of $\mathbb{z}$ only, i.e., $f(\mathbb{z}) = f_0(z_{i_1}, \ldots, z_{i_n})$ for some f_0, the expectation of f over q is calculated by the summation of N^n terms:

$$\mathrm{E}_q\big[f(\mathbb{z})\big] = \sum_{z_{i_1},\ldots,z_{i_n}} q_{i_1}(z_{i_1}) \cdots q_{i_n}(z_{i_n}) f_0(z_{i_1}, \ldots, z_{i_n}).$$

First, for all R_h and R_k with $h \neq k$, $Z(R_h) \cap Z(R_k) = \varnothing$ from the definition of $\sim_R$. Thus, $\delta(R_h)$ and $\delta(R_k)$ are independent for each h and k. Second, for each R_h, $\delta(R_h) = 1$ iff all elements of $H(R_h)$ and the values of all elements of $Z(R_h)$ are identical to some element i in H. This proves Lemma 1 holds.

Lemma 1 can easily be generalized to the case where R contains pairs of letters. In that case, respective pairs of letters (a, b) form a singleton equivalence class $R_{a,b}$, where $\mathrm{E}_q[\delta(R_{a,b})] = \delta\binom{a}{b}$.

We are now ready to calculate the approximation of $\mathrm{E}_q[\log(g_k)]$ in Eq. (14). Let us consider $\mathrm{E}_q[C^{-t}_{ka_t z_{t+1}}]$, for example. By definition

$$\mathrm{E}_q\big[C^{-t}_{ka_t z_{t+1}}\big] = \sum_{s \neq t, t-1} \mathrm{E}_q\left[\delta\begin{pmatrix} z_s, & a_s, & z_{s+1} \\ k, & a_t, & z_{t+1} \end{pmatrix}\right].$$

For $s \neq t-1, t, t+1$, we have $z_{t+1} \neq z_s, z_{s+1}$. Lemma 1 implies

$$\mathrm{E}_q\left[\delta\begin{pmatrix} z_s, & a_s, & z_{s+1} \\ k, & a_t, & z_{t+1} \end{pmatrix}\right] = q_s(k)\delta\binom{a_s}{a_t}\left(\sum_i q_{s+1}(i) q_{t+1}(i)\right).$$

For $s = t+1$, we have $(z_s, k) \sim (z_{s+1}, z_{t+1})$. Thus, Lemma 1 implies

$$\mathrm{E}_q\left[\delta\begin{pmatrix} z_s, & a_s, & z_{s+1} \\ k, & a_t, & z_{t+1} \end{pmatrix}\right] = \mathrm{E}_q\left[\delta\begin{pmatrix} z_{t+1}, & a_{t+1}, & z_{t+2} \\ k, & a_t, & z_{t+1} \end{pmatrix}\right] = q_{t+1}(k) q_{t+2}(k) \delta\binom{a_t}{a_{t+1}}.$$

Therefore, we obtain

$$\mathrm{E}_q\left[C^{\neg t}_{ka_tz_{t+1}}\right] = \sum_{s\neq t,t-1,t+1} q_s(k)\delta\begin{pmatrix}a_s\\a_t\end{pmatrix}\left(\sum_i q_{s+1}(i)q_{t+1}(i)\right) + q_{t+1}(k)q_{t+2}(k)\delta\begin{pmatrix}a_t\\a_{t+1}\end{pmatrix}.$$

Similar calculations give

$$\begin{aligned}
\mathrm{E}_q\left[C^{\neg t}_{k}\right] &= \sum_{s\neq t} q_s(k),\\
\mathrm{E}_q\left[C^{\neg t}_{k00}\right] &= \sum_{s\neq t,t-1} \mathrm{E}_q\left[\delta\begin{pmatrix}z_s, & a_s, & z_{s+1}\\ k, & 0, & 0\end{pmatrix}\right]\\
&= \sum_{s\neq t,t-1} q_s(k)q_{s+1}(0)\delta\begin{pmatrix}a_s\\0\end{pmatrix}\\
\mathrm{E}_q\left[C^{\neg t}_{z_{t-1}a_{t-1}k}\right] &= \sum_{s\neq t,t-1} \mathrm{E}_q\left[\delta\begin{pmatrix}z_s, & a_s, & z_{s+1}\\ z_{t-1}, & a_{t-1}, & k\end{pmatrix}\right]\\
&= \sum_{s\neq t,t-1,t-2}\left(\sum_i q_s(i)q_{t-1}(i)\right)\delta\begin{pmatrix}a_s\\a_{t-1}\end{pmatrix}q_{s+1}(k)\\
&\quad + q_{t-2}(k)q_{t-1}(k)\delta\begin{pmatrix}a_{t-2}\\a_{t-1}\end{pmatrix},\\
\mathrm{E}_q\left[\delta\begin{pmatrix}k, a_t, z_{t+1}\\ z_{t-1}, a_{t-1}, k\end{pmatrix}\right] &= q_{t-1}(k)\delta\begin{pmatrix}a_t\\a_{t-1}\end{pmatrix}q_{t+1}(k).
\end{aligned}$$

Using these, $\exp(\mathrm{E}_q[\log g_k])$ is approximated as follows. For the case where $a_{t-1}\neq 0$ and $a_t = 0$,

$$\begin{aligned}
&\exp\left(\mathrm{E}_q[\log g_k]\right)\\
&\approx \left(\sum_{s\neq t} q_s(k) + NA\beta\right)^{-1}\left(\sum_{s\neq t,t-1} q_s(k)q_{s+1}(0)\delta\begin{pmatrix}a_s\\0\end{pmatrix} + N\beta\right)\\
&\quad\times\left(\sum_{s\neq t,t-1,t-2} q_{s+1}(k)\sum_i q_s(i)q_{t-1}(i)\delta\begin{pmatrix}a_s\\a_{t-1}\end{pmatrix} + q_{t-2}(k)q_{t-1}(k)\delta\begin{pmatrix}a_{t-2}\\a_{t-1}\end{pmatrix} + \beta\right);
\end{aligned}\tag{15}$$

for $a_{t-1}\neq 0$ and $a_t\neq 0$,

$$\begin{aligned}
&\exp\left(\mathrm{E}_q[\log g_k]\right)\\
&\approx \left(\sum_{s\neq t} q_s(k) + NA\beta\right)^{-1}\left(\sum_{s\neq t,t-1,t+1} q_s(k)\sum_i q_{s+1}(i)q_{t+1}(i)\delta\begin{pmatrix}a_s\\a_t\end{pmatrix}\right.\\
&\quad\left. + q_{t+1}(k)q_{t+2}(k)\delta\begin{pmatrix}a_t\\a_{t+1}\end{pmatrix} + \beta\right)
\end{aligned}$$

$$\times \left(\sum_{s\neq t,t-1,t-2} q_{s+1}(k) \sum_i q_s(i) q_{t-1}(i) \delta \begin{pmatrix} a_s \\ a_{t-1} \end{pmatrix} \right.$$

$$\left. + q_{t-2}(k) q_{t-1}(k) \delta \begin{pmatrix} a_{t-2} \\ a_{t-1} \end{pmatrix} + q_{t-1}(k) q_{t+1}(k) \delta \begin{pmatrix} a_t \\ a_{t-1} \end{pmatrix} + \beta \right); \tag{16}$$

if $a_{t-1} = 0$, we have $g_k = \delta \binom{k}{0}$.

A naive calculation of Eqs. (15) and (16) takes $\mathrm{O}(NT)$ steps, but in fact one can compute these values in $\mathrm{O}(N)$ by maintaining the values of $\mathrm{E}_q[C_k]$ and $\mathrm{E}_q[C_{jak}]$. To calculate $\sum_{s\neq t} q_s(k)$, for example, it is not necessary to sum up all the T terms $q_s(k)$ every time, since $\sum_{s\neq t} q_s(k) = \mathrm{E}_q[C_k^{-t}] = \mathrm{E}_q[C_k] - q_t(k)$, where $\mathrm{E}_q[C_k]$ is updated by $\mathrm{E}_q[C_k] - q_t^{\mathrm{old}}(k) + q_t^{\mathrm{new}}(k)$, which takes constant time. Noting that updating $\mathrm{E}_q[C_{jak}]$ also takes constant time and

$$\begin{aligned}
&\sum_{s\neq t,t-1,t-2} q_{s+1}(k) \sum_i q_s(i) q_{t-1}(i) \delta \begin{pmatrix} a_s \\ a_{t-1} \end{pmatrix} \\
&= \sum_i q_{t-1}(i) \sum_{s\neq t,t-1,t-2} q_{s+1}(k) q_s(i) \delta \begin{pmatrix} a_s \\ a_{t-1} \end{pmatrix} \\
&= \sum_i q_{t-1}(i) \left(\mathrm{E}_q[C_{ia_{t-1}k}] - q_{t-2}(i) q_{t-1}(k) \delta \begin{pmatrix} a_{t-2} \\ a_{t-1} \end{pmatrix} - q_{t-1}(i) q_t(k) \right. \\
&\quad \left. - q_t(i) q_{t+1}(k) \delta \begin{pmatrix} a_t \\ a_{t-1} \end{pmatrix} \right),
\end{aligned}$$

one can easily see that Eq. (15) can be calculated in $\mathrm{O}(N)$ time. A similar argument applies to the calculation of Eq. (16). Therefore, it takes $\mathrm{O}(N^2 T)$ steps to update $q_t(k)$ for all t and k.

2.5 An approximation for the objective function $D(q)$

In this section, a new approximation approach for minimizing $D(q)$ is discussed. The approach presented in the previous section is based on the updating formula (Eq. (12)), which should lead q to a local minimum convergence point, provided that it is calculated precisely. However, instead, we have used an approximation formula where we have no guarantee of monotonicity or convergence. Moreover, the intractability of the calculation of $D(q)$, even from the approximated q, prevents us from determining a point where we should stop iterating.

Instead, this subsection proposes an approximation $D_0(q)$ of $D(q)$ as an objective function, to which we apply the CVB0 technique. Unlike for the approximation presented in the previous section, it is ensured that the values of q will converge to a local optimal point, and thus one can easily decide when the updating of the values of q should be terminated.

Now, let us return to the definition of $D(q)$:

$$D(q) = -\mathrm{E}_q\big[\log\big(\Pr(\mathbb{z}, \mathbb{o})\big)\big] + \mathrm{E}_q\big[\log\big(q(\mathbb{z})\big)\big].$$

By Eq. (4), $D(q)$ is rewritten as

$$D(q) = -\sum_i \left(\sum_{a\neq 0, j\neq 0} \mathrm{E}_q\big[\log \Gamma(C_{iaj} + \beta)\big] + \mathrm{E}_q\big[\log \Gamma(C_{i00} + N\beta)\big] \right.$$

$$- \mathrm{E}_q\big[\log \Gamma(C_i + NA\beta)\big]\Big)$$
$$+ \mathrm{E}_q\big[\log R(\beta)\big] + \mathrm{E}_q\big[\log\big(q(\mathbb{z})\big)\big].$$

As $\mathrm{E}_q[\log(\Pr(\mathbb{z}, \mathbb{o}))]$ is not tractable, we approximated it by the 0th order Taylor approximation (Eq. (13)). By using $\mathrm{E}_q[\log \Gamma(\cdot)] \approx \log \Gamma(\mathrm{E}_q[\cdot])$, we approximate $D(q)$ by $D_0(q)$ as

$$D_0(q) = -\sum_i \Bigg(\sum_{a\neq 0, j\neq 0} \log \Gamma\big(\mathrm{E}_q[C_{iaj}] + \beta\big) + \log \Gamma\big(\mathrm{E}_q[C_{i00}] + N\beta\big) - \log \Gamma\big(\mathrm{E}_q[C_i] + NA\beta\big)\Bigg) + \mathrm{E}_q\big[\log\big(q(\mathbb{z})\big)\big] + \text{const.},$$

where

$$\mathrm{E}_q[C_i] = \sum_t q_t(i),$$
$$\mathrm{E}_q[C_{iaj}] = \sum_t q_t(i) q_{t+1}(j) \delta\begin{pmatrix} a_t \\ a \end{pmatrix},$$
$$\mathrm{E}_q\big[\log\big(q(\mathbb{z})\big)\big] = \sum_t \sum_i q_t(i) \log q_t(i).$$

We call our method that minimizes $D_0(q)$ *the zeroth-order global approximation of collapsed variational Bayes (GCVB0)*. For $a_{t-1} \neq 0$, which implies $k \neq 0$,

$$\frac{\partial}{\partial q_t(k)} D_0(q)$$
$$= -\sum_{i\neq 0} q_{t+1}(i) \psi\big(\mathrm{E}_q[C_{ka_t i}] + \beta\big)\left(1 - \delta\begin{pmatrix} a_t \\ 0 \end{pmatrix}\right) - \sum_i q_{t-1}(i) \psi\big(\mathrm{E}_q[C_{ia_{t-1}k}] + \beta\big)$$
$$- q_{t+1}(0) \psi\big(\mathrm{E}_q[C_{k00}] + N\beta\big)\delta\begin{pmatrix} a_t \\ 0 \end{pmatrix} + \psi\big(\mathrm{E}_q[C_k] + NA\beta\big) + 1 + \log q_t(k), \quad (17)$$

where $\psi(x) = \frac{d}{dx} \log \Gamma(x)$. It should be noted that $D_0(q)$ is required to be minimized under restrictions that are a result of the fact that $q_t(\cdot)$ is a probability for each t. Optimization is required to treat these restrictions, and therefore, we have to transform variables and/or use numerical optimization methods appropriately. The probability function q has local minimal points, as $D_0(q)$ is also lower-bounded. A theoretical analysis proves the convergence of q to a local optimal point using appropriate numerical optimization methods, such as quasi-Newton methods. *The zeroth-order global approximation of collapsed variational Bayes (GCVB0)*. We note that $D_0(q)$ has many local minimal points and most numerical optimizers just converge to one of these points, as is usual for other methods, such as EM and CVB. The computation time required for the calculation of the derivative in Eq. (17) is $\mathrm{O}(TN^2)$, while we calculate ψ $\mathrm{O}(N^2 A)$ times. Details of the implementation are given in Sect. 4.2.

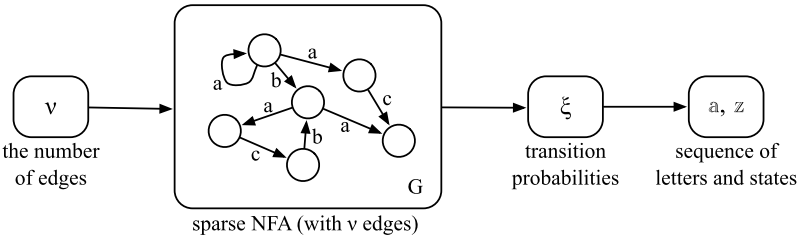


Fig. 1 A simple generative model for sparse NFAs

3 Using graph structures for inferring sparse PFAs

In the algorithms discussed in Sect. 2, if the smoothing parameter β is large, every transition rule tends to have somewhat equally large probabilities, unless the training data are extremely large. Since a smaller β makes the prior of ξ tend to provide fewer edges with higher probabilities, it makes $\Pr(\mathbb{o}, \mathbb{z})$ tend to have a higher probability when many C_{iaj}s equal 0. Hence, to infer sparse PFAs, we must set β to be very small. In this section, we give another CGS-based approach, which we call *CGS-SG*, for sparse NFAs (Fig. 1). CGS-SG samples sparse NFAs as the underlying graph structures of PFAs. We assume that an NFA is generated such that transition rules are added one by one up to some number ν, while N and A are given. We denote the edge from i to j labeled with a by (i, a, j). By identifying an NFA G as a subset of the set of transition edges in the fully connected NFA, we write, e.g., $G \subseteq G'$ if G' has every edge of G.

Let ν_i be the number of outgoing edges from state i. We have $0 \leq \nu_i \leq NA$ and $\nu = \sum_i \nu_i$. The prior of an NFA G before samples $\mathbb{o}$ are drawn is given as

$$\Pr(G) = \sum_{\nu} \Pr(G|\nu) \Pr(\nu).$$

We assume that edges are added uniformly, under which condition all combinations of ν edges are equally likely to be generated. Thus, for each G_k that has k edges we have

$$\Pr(G_k \mid \nu = k) = \frac{1}{\binom{AN^2}{k}}$$

and $\Pr(G_k \mid \nu = h) = 0$ if $k \neq h$. Consequently, for each k in $\{0, \ldots, AN^2 - 1\}$, we have

$$\frac{\Pr(G_{k+1} \mid \nu = k+1)}{\Pr(G_k \mid \nu = k)} = \frac{k+1}{AN^2 - k},$$

for every sequence $G_0 \subset \cdots \subset G_{AN^2}$, where the number of edges of G_k is k.

This type of distribution of G is different from distributions where each edge is independently added. Whereas, in the latter case, ν has a binomial distribution and is centralized at some point, in the former case, the distribution of ν is free to be given as a prior $\Pr(\nu)$.

Suppose that $\mathbb{o}$ and $\mathbb{z}$ are given. By $G_{\min}$, we denote the minimal NFA, which consists of just the edges necessary and sufficient to generate $\mathbb{o}$ and $\mathbb{z}$. That is, $G_{\min} = \{(i, a, j) \mid C_{iaj} > 0\}$. We assume that the hyperparameter of $\Pr(\xi)$ is β for all edges, including transitions to the end marker and states. By integrating out ξ from $\Pr(\mathbb{o}, \mathbb{z} \mid \xi) \Pr(\xi|G)$ (cf. Eq. (3)), we

Algorithm 2: A Single Iteration of Gibbs Sampling for Sparse NFAs

Data: $\mathbb{o}$
input and output: G,$\mathbb{z}$
1 $G_{\min} \leftarrow$ the minimal NFA from $\mathbb{o}$ and $\mathbb{z}$;
2 **foreach** $e = (i, a, j) \in G_{\text{full}} - G_{\min}$ **do**
3 **if** $e \in G$ **then**
4 $G \leftarrow G - \{e\}$;
5 **end**
6 $G \leftarrow G \cup \{e\}$ with probability
  $\Pr(G \cup \{e\} \mid \mathbb{o}, \mathbb{z})/(\Pr(G \mid \mathbb{o}, \mathbb{z}) + \Pr(G \cup \{e\} \mid \mathbb{o}, \mathbb{z}))$;
7 **end**
8 **foreach** $z_t \in \mathbb{z}$ **do**
9 **foreach** *state* $k \in 0, \cdots, N$ **do**
10 Calculate $g_k(G)$;
11 **end**
12 $z_t \leftarrow k$ with probability $g_k(G)/\sum_i g_i(G)$;
13 **end**

have

$$\Pr(\mathbb{o}, \mathbb{z} \mid G) = \frac{\prod_{i,a,j} \Gamma(C_{iaj} + \beta)\Gamma(\beta)^{-1}}{\prod_i \Gamma(C_i + \nu_i\beta)\Gamma(\nu_i\beta)^{-1}}.$$

Thus, $\Pr(G_k, \mathbb{o}, \mathbb{z})$, where $G_{\min} \subset G_k$ is given by

$$\Pr(G_k, \mathbb{o}, \mathbb{z}) = \Pr(\mathbb{o}, \mathbb{z} \mid G_k)\Pr(G_k) = \frac{\prod_{i,a,j} \Gamma(C_{iaj} + \beta)\Gamma(\beta)^{-1}}{\prod_i \Gamma(C_i + \nu_i\beta)\Gamma(\nu_i\beta)^{-1}} \frac{1}{\binom{AN^2}{k}} \Pr(\nu = k). \quad (18)$$

It should be noted that, if $G_{\min} \not\subseteq G_k$, $\Pr(G_k, \mathbb{o}, \mathbb{z}) = 0$.

Let $G_{\min} \subseteq G_k \subset G_{k+1}$ and $G_k \cup \{(i, a, j)\} = G_{k+1}$. From Eq. (18),

$$\frac{\Pr(G_{k+1} \mid \mathbb{o}, \mathbb{z})}{\Pr(G_k \mid \mathbb{o}, \mathbb{z})} = \begin{cases} \dfrac{\Gamma(C_i + \nu_i\beta)\Gamma(\nu_i\beta)^{-1}}{\Gamma(C_i + (\nu_i + 1)\beta)\Gamma((\nu_i + 1)\beta)^{-1}} \dfrac{k+1}{AN^2 - k} \dfrac{\Pr(\nu = k+1)}{\Pr(\nu = k)} \\ \quad \text{if } C_i \neq 0, \\ \dfrac{k+1}{AN^2 - k} \dfrac{\Pr(\nu = k+1)}{\Pr(\nu = k)} \\ \quad \text{otherwise.} \end{cases}$$

Algorithm 2 shows a single iteration of CGS-SG, which samples $\mathbb{z}$ and G in an alternating fashion. It should be noted that G is not equal to $G_{\min}$, but is likely to be larger than $G_{\min}$.

On line 10, g_k is calculated as follows, similarly to Eqs. (7)–(9):

$$\begin{array}{l} \text{if either } (z_{t-1}, a_{t-1}, k) \quad \text{or } (k, a_t, z_{t+1}) \quad \text{is not in } G, \\ \quad g_k = 0, \\ \text{otherwise,} \end{array}$$

$$g_k = \frac{(C^{\neg t}_{ka_t z_{t+1}} + \beta)(C^{\neg t}_{z_{t-1}a_{t-1}k} + \delta(\begin{smallmatrix} z_{t-1}, & a_{t-1}, & k \\ k, & a_t, & z_{t+1} \end{smallmatrix}) + \beta)}{(C^{\neg t}_{k} + \nu_i(G)\beta)},$$

where $\nu_i(G)$ is ν_i of G.

We can locate the update of G (Line 1–7) inside the loop of updating z_t for each t (after Line 12) in order to sample G more frequently. Since the computational cost of updating G is $\mathrm{O}(N^2 A)$, while that of updating z_t for each t is $\mathrm{O}(N)$, the interval of updating G should be more than $\mathrm{O}(NA)$.

We finally predict the generating probability of $\mathbb{b}$ by Eqs. (5) and (6), similarly to CGS-PFA of Sect. 2.3.

4 Experiments

In all the experiments, we used data sets that were offered in the Probabilistic Automata learning Competition (PAutomaC).[4] There are 98 data sets, which were artificially generated from various kinds of PFAs, including general PFAs, HMMs, and PDFAs. The 98 data sets were provided in two phases, PAutomaC I and II, where the respective phases contained 50 and 48 data sets. We refer to the 26th problem of PAutomaC II as Prob. II-26, and so on. Each data set is divided into a training set and a test set. Each test set is constructed such that it contains no duplicate sentences. Competition participants were required to submit an answer that assigns probabilities $\Pr_C(x)$ to the sentences x in each test set TS. The score of the answer is calculated by a perplexity defined as

$$\text{Score} = 2^{-\sum_{x \in \text{TS}} \Pr_T(x) \log \Pr_C(x)}, \tag{19}$$

where $\Pr_T(x)$ is the true probability assigned to the sentence x. Both $\Pr_T$ and $\Pr_C$ are normalized. The power part of Eq. (19) approximates the KL-Divergence (+ constant w.r.t. inferred distributions) between the inferred and the true distribution by limiting its domain to the finite test set. For more details, see Verwer et al. (2012).

Each experiment in this section was run using computation nodes in a grid environment called InTrigger,[5] each node contained 2 CPUs (Xeon E5410/E5330, 2.33/2.3 GHz and 4 cores+HT) with a memory of 32/24 GB. Each execution was done in a single thread, and therefore, essentially, we did not use parallel computation.

4.1 Experimental details of CGS

In our experiments, we sampled $\mathbb{z}$ for every 100 iterations between iterations 10, 100, and 20,000, and hence obtained $S = 100$ samples in total (Period = 100, BurnInTime = 10,000, and SamplingNum = 100 in Algorithm 1). Furthermore, we ran the algorithm 10 times independently.[6] Our final answer to each problem was calculated as the average of the probabilities obtained from 1,000 samples. Figure 2 illustrates how we took sampling points in the whole experiment. Figure 3 shows how the scores vary by changing the number of iterations for Prob. I-19. The scores are for the answers calculated from the last 100 samples of 10

[4] PAutomaC (http://ai.cs.umbc.edu/icgi2012/challenge/Pautomac/).

[5] http://www.intrigger.jp/.

[6] These independent trials were executed in parallel using a parallel computing processing system, GXP Make (http://www.logos.ic.i.u-tokyo.ac.jp/gxp/).

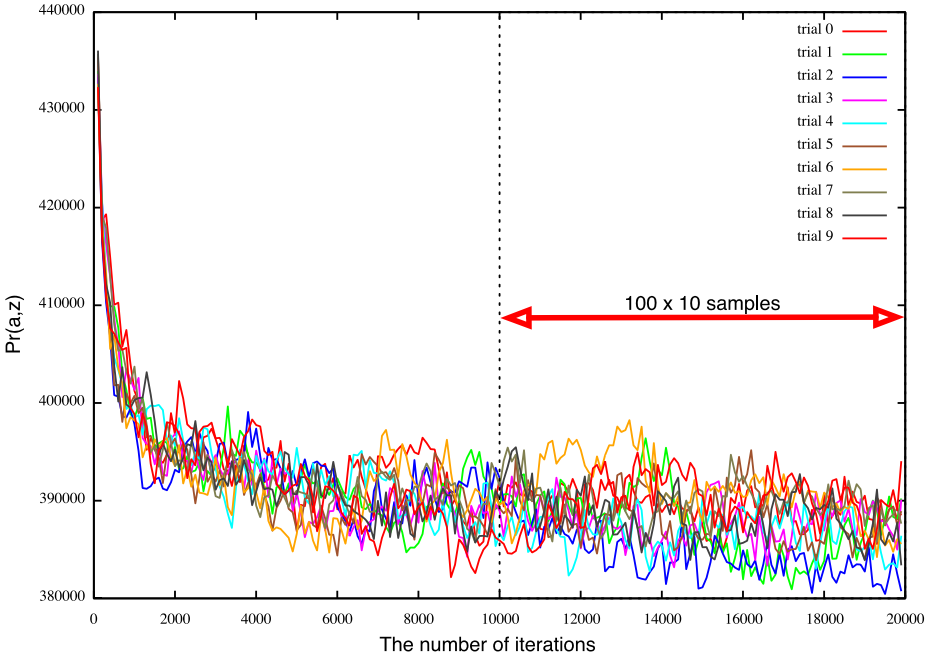


Fig. 2 We took 1,000 sampling points

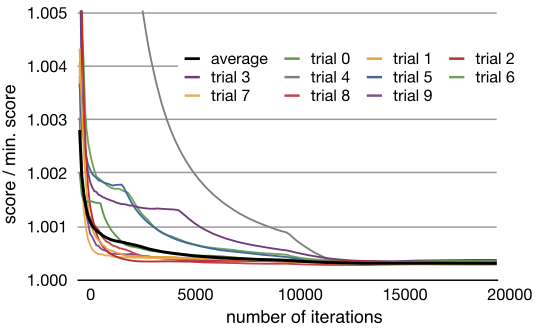


Fig. 3 Scores by CGS with different number of iterations for Prob. I-19 with $N = 10$ and $\beta = 0.02$

trials for a different number of iterations up to 20,000 (thus, the burn-in period was changed while the sampling period was fixed). After the 12,000th iteration, the respective lines seem flat and close to each other. Hence, in fact, 20,000 iterations seem to be sufficient. The black bold line represents the score of the average answer of the 10 answers. Empirically, the score obtained by the average of 10 answers is generally better than the average of their scores.

Before the actual training stage, we conducted two preparatory stages in which N and β were determined. As Fig. 4 shows, the best choice for N depends on the problem. By 10-fold cross-validation (CV), we set N to be the value among $\{10, 20, \ldots, 90\}$ that gives the largest probability, where we used $\beta = 0.5$. After determining N, we selected the best value amongst $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5\}$ for β, again by 10-fold CV. The effectiveness of this process is illustrated in Fig. 5, where the circles indicate the scores achieved by the chosen values. Figure 6 shows the average score ratios obtained using different values for

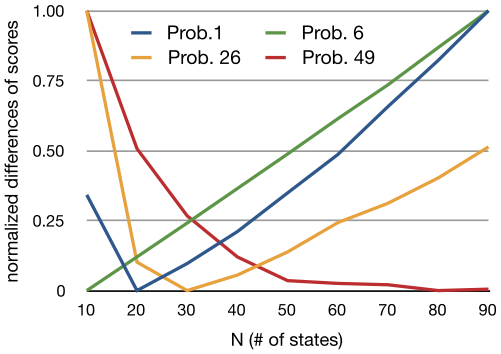


Fig. 4 Variation in scores when changing the number of states for Prob. I-1, between 6, 26, and 49

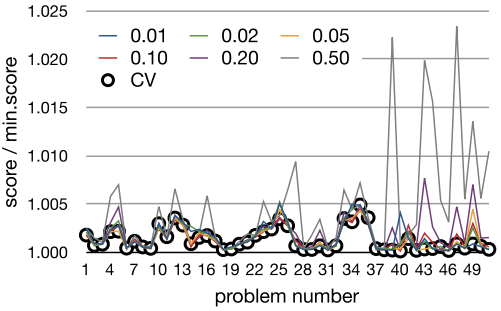


Fig. 5 CGS scores with different β

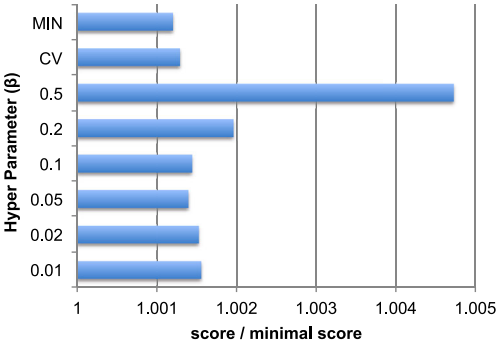


Fig. 6 Average score ratios obtained by different values of β

β. Bars indexed with real values (0.01–0.5) show the average ratio obtained by using fixed respective values. On average, the value of β chosen by CV gives better scores than any other fixed β, and the scores are close to those achieved by the best choice. Figure 7 shows the correlation between the true number and inferred number of states obtained by cross-validation for PAutomaC II data sets. The correlation coefficient is 0.56 and the slope of the regression line is 0.96.

For the respective problems in PAutomaC I, one iteration took approximately 0.2 to 2.0 s, and thus, 400 to 40,000 s for 20,000 iterations. To determine the values of N and β among nine and six candidates by 10-fold cross-validation, respectively, one must run CGS 150 times in total for every problem. Using these determined values, we ran CGS 10 times to obtain the final answer.

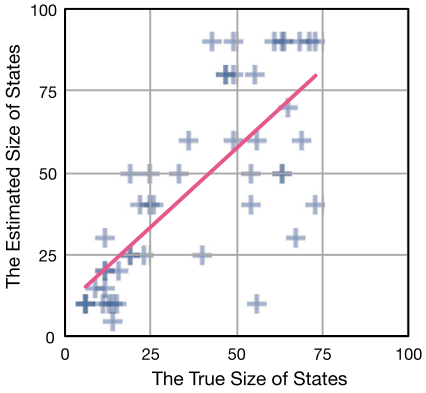


Fig. 7 Correlation between true numbers of states and inferred numbers using CV for PAutomaC II data sets

4.2 Comparison of CVB0, GCVB0, and CGS for PFAs

In this section, we compare CVB0 and GCVB0 for the PFAs described in Sects. 2.4 and 2.5. The numerical optimizer that we used for GCVBO was a limited-memory quasi-Newton method called *L-BFGS-B* (Zhu et al. 1997).[7] L-BFGS-B is able to take lower and upper bounds for each variable and minimizes an objective function within the given bound.

In order to handle the constraints that $\sum_k q_t(k) = 1$ for all t, we use variables $x_t(k)$ such that $q_t(k) = x_t(k)^2 / \sum_i x_t(i)^2$, instead of $q_t(k)$ themselves. The objective function of GCVB0 is also modified as $D_0(q) + \sum_t (1 - \sum_k q_t(k))^2$. By this transformation, each $x_t(k)$ has only one constraint, $x_t(k) \geq 0$.[8] The computational cost for calculating $D_0(q)$ and its derivations is $O(TN^2)$, which is the same as for CVB0. The factor of L-BFGS-B for GCVB0 for convergence[9] is set to 10^7.

Figure 8 shows the experimental results of CVB0 and GCVB0 for the data of PAutomaC I-1, where we set $N = 30$ and $\beta = 0.01$. The thick and thin blue lines in Fig. 8(a) represent the relation between the number of iterations and the value of $D_0(q)$ obtained at each iteration by CVB0 and GCVB0, respectively. It should be noted that the value of $D_0(q)$ is displayed for CVB0 only for the sake of comparison; it was not used in the algorithm CVB0. GCVB0 converges after 680 iterations[10] and is stopped. On the other hand, one cannot determine when CVB0 has converged, since CVB0 has no global function that can be minimized, and the change in $q_t(k)$ did not become smaller, at least during 2000 iterations.

Figure 8(b) shows the scores obtained by estimated transition probabilities ξ according to Eq. (10). The final score of CVB0 (32.609 after 2,000 iterations) was better than that of GCVB0 (32.651). Moreover, the score of CVB0 improved more quickly than that of GCVB0. Both CVB0 and GCVB0 yielded worse scores than CGS (32.569) for the above data and settings of N and β.

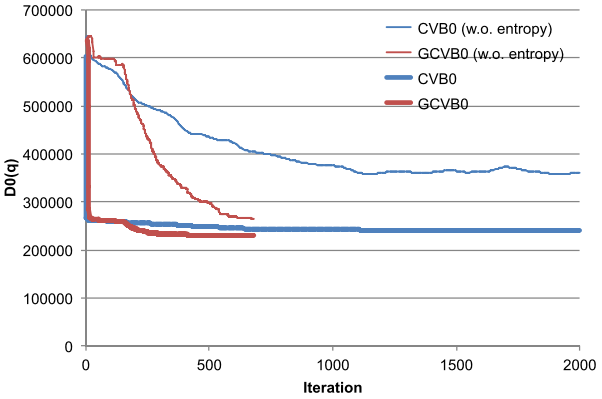
Figure 8(c) shows the number of edges of the obtained PFA. That is, the number of triples (i, a, j) such that $E_q[C_{i,a,j}] > 1$, which indicates the density of the network. As the value

[7] http://users.eecs.northwestern.edu/~nocedal/lbfgsb.html.

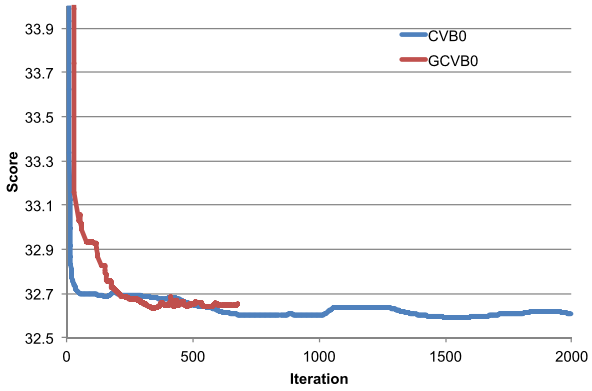
[8] We used $x_t(k) > 10^{-7}$ as the lower bound for L-BFGS-B in practice.

[9] The objective function f is considered converged when $|f(\mathbf{x}^{\text{new}}) - f(\mathbf{x}^{\text{old}})| / |f(\mathbf{x}^{\text{old}})| < 10^7 \cdot \text{EPSMCH}$, where $\text{EPSMCH} = 2.220 \cdot 10^{-16}$ in our environment.

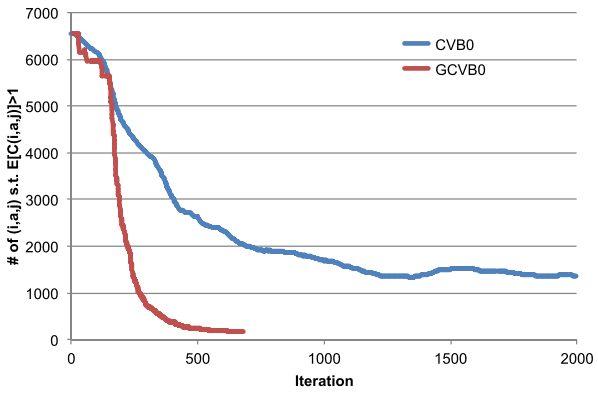
[10] We define a single iteration for GCVB0 as a computation of $D_0(q)$ and its derivation for a single point.



(a) Global approximation function $D_0(q)$ as a function of iterations.



(b) Score as a function of iterations.



(c) The number of (i, a, j) such that $E_q[C_{iaj}] > 1$.

Fig. 8 Comparison of CVB0 and GCVB0 for PAutomaC I-1 data. ($N = 30$, $\beta = 0.01$) (Color figure online)

Table 1 Comparison of CGS for fully-connected PFAs and Algorithm 2 for a sparse PDFA (PAutomaC II-26)

	CGS-PFA				CGS-SG			
β	0.5	0.1	0.05	0.01	0.5	0.1	0.05	0.01
score − min. score	14.442	0.942	0.108	**0.092**	1.800	**0.251**	0.258	0.391
# of valid states i	91	91	88	82	91	91	91	91
# of valid pairs (i, a)	628	355	334	302	440	392	370	322
# of valid edges (i, a, j)	4775	728	511	346	1087	766	708	525
# of possible edges in G	(49231)	(49231)	(49231)	(49231)	1242	1535	2229	5733
inferred determinacy	7.60	2.05	1.53	1.15	2.47	1.95	1.91	1.63

of GCVB0 (117) is much smaller than that of CVB0 (1356), GCVB0 tends to give a more compact PFA than does CVB0. We represent the value of $D_0(q) - \mathrm{E}_q[q(\mathbb{z})]$ by thinner lines in Fig. 8(a), where $\mathrm{E}_q[q(\mathbb{z})]$ shows the entropy. The thick and thin red lines (GCVB0) come quite close to each other, which means that the approximation of $D(q)$ with $D_0(q)$ tends to bias $q_t(k)$ toward 0 or 1 as compared to the approximation technique of CVB0.

4.3 Effects of sampling underlying NFAs for sparse PFAs

An experimental result for CGS-SG, which was introduced in Sect. 3, is shown in this section. In the experiment, we assumed that $\Pr(\nu)$ was uniform, and G was periodically resampled in the loop of updating z_t in Algorithm 2. We implemented and executed CGS-PFA and CGS-SG for Prob. II-26, which was generated by a sparse PDFA. The number of states was set to 90 and the number of iterations to 40,000, where 201 points were taken as samples in the latter half of all iterations.

Table 1 summarizes the results obtained by CGS-PFA and CGS-SG with different values for β. Among them, CGS-PFA with the smallest $\beta = 0.01$ results in the best score. However, with larger values for β, the performance of CGS-PFA decreases sharply. CGS-SG achieved its best score when $\beta = 0.1$. For other values, CGS-SG performed rather stably as compared to CGS-PFA. For the two largest β, 0.1 and 0.5, CGS-SG performed better than CGS-PFA.

We call a state i *valid* if $C_i > 0$, and similarly a pair (i, a) and an edge (i, a, j) are said to be *valid* if $C_{ia} > 0$ and $C_{iaj} > 0$, respectively. It should be noted that the minimal NFA defined in Sect. 3 has all and only valid edges. CGS-PFA with $\beta = 0.01$ had the least number, 346, of valid edges after the last iteration. The number 346 is close enough to the true number, 299, of the edges of the generating PDFA[11] of Prob. II-26. We define the *inferred determinacy* to be the ratio of valid pairs to valid edges. The value should be 1 if the generating PDFA is correctly inferred. The inferred determinacy of CGS-PFA with $\beta = 0.01$ is 1.15, which suggests that CGS-PFA with sufficiently small β can be effective for inferring sparse PDFAs.

CGS-PFA outperformed the other methods when the hyperparameter β was set correctly; nevertheless, with larger values for β, the performance of CGS-PFA decreased quite sharply in terms of both the score and determinacy. On the other hand, CGS-SG worked stably with different values of β.

[11]The number is obtained after transforming the PDFA to satisfy our postulation on the forms of PFAs described in Sect. 2.1.

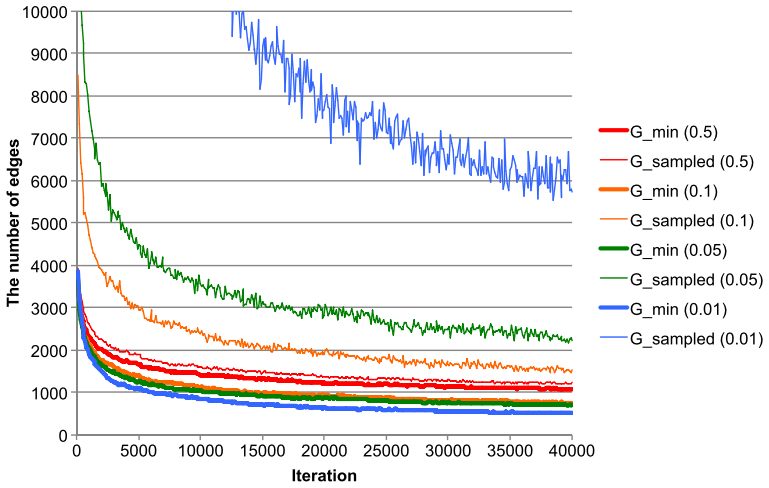


Fig. 9 Number of edges as a function of iterations for $G_{\min}$ and sampled G of Algorithm 2 for PAutomaC II-26. $\beta \in \{0.5, 0.1, 0.05, 0.01\}$

Figure 9 shows a further analysis of the underlying NFA sampled by CGS-SG. The bold and thin lines show the numbers of the edges of the minimal NFAs $G_{\min}$ and sampled NFAs G_{sampled}, respectively, as functions of iterations. Different colors correspond to different values of β. The larger the β, the smaller the gap between the number of edges of $G_{\min}$ and G_{sampled}. If β is not small, while $G_{\min}$s are bounded by sampled Gs and can be easily expanded within sampled Gs, sampled Gs are expected to be not much larger than $G_{\min}$. Thus, CGS-SG succeeded in inferring a relatively sparse PFA, even when β was not sufficiently small.

4.4 Comparison of CGS for PFAs and CGS for HMMs

HMMs are a special type of PFAs. In HMMs, ξ is factorized as $\xi_{iaj} = \eta_{ia}\theta_{ij}$, where η_{ia} is the *emission probability* that letter a is emitted from state i, and θ_{ij} represents the *state transition probability* from state i to state j. It is known that every PFA has an equivalent HMM, but in general the transformation from a PFA to an equivalent HMM squares the number of states.

CGS-HMM, the CGS algorithm for inferring HMMs, is obtained in a way similar to CGS-PFA:

$$\Pr_{\text{HMM}}\left(z_t = k \mid \mathfrak{o}, \mathbb{z}^{\neg t}\right) = \frac{g_k^{\text{HMM}}}{\sum_i g_i^{\text{HMM}}},$$

where g_k^{HMM} is given as follows.

$$\text{if } a_{t-1} = 0, \qquad g_k^{\text{HMM}} = \delta\begin{pmatrix} k \\ 0 \end{pmatrix},$$

$$\text{if } a_{t-1} \neq 0 \text{ and } a_t = 0, \qquad g_k^{\text{HMM}} = \frac{(C_{ka_t}^{\neg t} + \alpha)(C_{z_{t-1}k}^{\neg t} + \beta)}{(C_k^{\neg t} + A\alpha)},$$

$$\text{if } a_{t-1} \neq 0 \text{ and } a_t \neq 0, \quad g_k^{\mathrm{HMM}} = \frac{(C_{ka_t}^{\neg t} + \alpha)(C_{kz_{t+1}}^{\neg t} + \beta)(C_{z_{t-1}k}^{\neg t} + \delta(^{z_{t-1},\ k}_{k,\ z_{t+1}}) + \beta)}{(C_k^{\neg t} + A\alpha)(C_k^{\neg t} - C_{k0}^{\neg t} + N\beta)}.$$

α and β are hyperparameters for the prior of η and θ, respectively. From sampled $\mathbb{z}$, CGS-HMM calculates

$$\tilde{\xi}_{iaj} = \tilde{\eta}_{ia}\tilde{\theta}_{ij} = \frac{C_{ia} + \alpha}{C_i + A\alpha}\frac{C_{ij} + \beta}{C_i + N\beta}, \tag{20}$$

which is inserted in Eq. (6). For technical convenience, for the analysis of CGS-HMM, we further introduce CGS-HMM(*), which calculates $\tilde{\xi}$ using

$$\tilde{\xi}_{iaj} = \frac{C_{iaj} + \beta}{C_i + NA\beta}$$

instead of Eq. (20).

We ran CGS-PFA and CGS-HMM together with CGS-HMM(*) for PAutomaC I data sets, which are classified into three types of data according to the generative model, namely, PFAs, PDFAs, and HMMs. Both hyperparameters α and β in CGS-PFA and CGS-HMM were always set to 0.1. The number of states were searched among $\{10, 15, 20, 30, 40, 50, 70, 90\}$. For each problem, both CGS-PFA and CGS-HMM were run 10 times, where each execution consisted of 10,000 iterations.

As Fig. 10 shows, CGS-HMM performed much worse than CGS-HMM, even when the generating automata were HMMs. The fact that CGS-HMM performed worse than CGS-HMM(*) suggests that CGS-HMM failed to estimate appropriate η and θ values, which can factorize ξ_{iaj} as $\eta_{ia}\theta_{ij}$. Table 2 shows the differences in scores for the data sets that are generated from HMMs. The second row ($\mathtt{HMM} - \mathtt{HMM}(*)$) represents the score of CGS-HMM minus the score of CGS-HMM(*), whose value is negative iff the above factorization of $\tilde{\xi}$ gives a better result. The negative values of the third column ($\min\{\mathtt{HMM}, \mathtt{HMM}(*)\} - \mathtt{PFA}$) show that the scores of CGS-PFA are worse than those of both CGS-HMM and CGS-HMM(*). As Table 2 shows, $\min\{\mathtt{HMM}, \mathtt{HMM}(*)\} - \mathtt{PFA}$ is negative (Nos. 37,39,40) only if $\mathtt{HMM} - \mathtt{HMM}(*)$ is negative (Nos. 37, 38, 39, 40). This implies that the reason why CGS-HMM performs worse than CGS-PFA is that it is difficult for CGS-HMM to estimate the η and θ that can factorize ξ appropriately.

The numbers of states that give the best results for CGS-HMM are shown in the fourth column of Table 2. As the figure shows, CGS-HMM tends to have a small number of states when it gives a good result. Although the number of states may not be sufficient for CGS-HMM, since HMMs with N^2 states can represent any PFAs with N-states, it is preferable to choose CGS-PFA for the following reasons. The computational cost of CGS-PFA is lower than that of CGS-HMM with a larger number of states; and CGS-HMM has the difficulty discussed in the previous paragraph.

4.5 Other collapsed methods for subclasses of PFAs

This section compares CGS-PFA with two other methods: (1) A state-merging algorithms for PDFAs; and (2) an algorithm based on variable-length gram (VGram). As described below, they maximize greedily a probability that is obtained by collapsing transition probabilities.

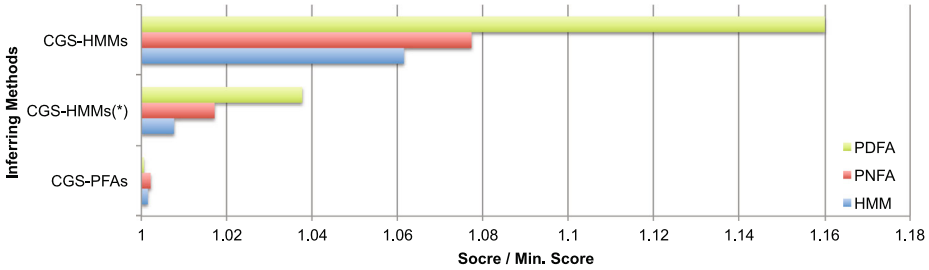


Fig. 10 Comparison of CGS-HMM, CGS-HMM(*) and CGS-PFA for respective generating classes

Table 2 Differences of scores among CGS-HMM, CGS-HMM(*) and CGS-PFA for the subset of PAutomaC I data sets generated by HMMs

No.	$\texttt{HMM} - \texttt{HMM}(*)$	$\min\{\texttt{HMM}, \texttt{HMM}(*)\} - \texttt{PFA}$	N (HMM)	N (PFA)
2	0.6062386	0.0484486	90	40
3	10.6874226	0.2430831	70	15
4	3.6180065	0.3853828	90	15
5	2.0407411	0.1371957	90	50
22	0.2159087	0.0474664	90	30
23	4.0745021	1.0769991	90	15
24	0.0216522	0.0243302	90	50
25	0.4575560	0.3295595	90	70
26	0.1223562	0.1214686	90	50
37	**−0.0039283**	**−0.0006234**	**50**	50
38	**−0.0031304**	0.0040898	**30**	40
39	**−0.0013565**	**−0.0080017**	**50**	50
40	**−0.0000129**	**−0.0002220**	**30**	70
41	0.0221770	0.0005653	70	90

4.5.1 Evidence-based state-merging algorithm for PDFAs (EStateMerge)

ALERGIA (Carrasco and Oncina 1994) is a well-known state-merging algorithm for inferring PDFAs. It constructs a PDFA starting from the probabilistic prefix tree acceptor (PPTA) for the training sample by merging states, the stochastic behaviors of which are similar. A modification of ALERGIA, called MDI (Thollard et al. 2000; Thollard 2001), merges states using another criterion: it merges states if this reduces the size of the automaton, while the *Kullback-Leibler divergence (KLD)* with the initial PPTA is kept small. We propose another state-merging evidence-based algorithm, EStateMerge. The criterion adopted by EStateMerge for merging states is based on the marginal probability obtained by collapsing transition probabilities: We greedily merge states if this increases $\Pr(\mathbb{o}|G)$, where G is the DFA concerned. Since the model is assumed to be a PDFA, the sequence of states $\mathbb{z}$ is uniquely determined by $\mathbb{o}$. Hence, by marginalizing ξ similarly to Eq. (4), we obtain

$$\Pr(\mathbb{o}|G) = \Pr(\mathbb{o}, \mathbb{z}) = \int \Pr(\mathbb{o}, \mathbb{z} \mid \xi) \Pr(\xi) d\xi = \prod_i \frac{\prod_a \Gamma(C_{ia} + \beta)}{\Gamma(C_i + A\beta)} \frac{1}{R(\beta)}, \tag{21}$$

where $C_{ia} = \sum_t \delta\binom{z_t,\ a_t}{i,\ a}$ and $R(\beta) = (\Gamma(\beta)\Gamma(A\beta)^{-1})^{N+1}$.

From the viewpoint of computational cost, one of the advantages of EStateMerge, as well as of other state-merging algorithms, is that both C_i and $C_{i,a}$ are changed only when the state i is merged with other states. Thus, it is enough to recalculate these local parts to update $\Pr(\square|G)$ at each merging step. Since states are aggressively merged whenever $\Pr(\square|G)$ increases, EStateMerge does not have the PAC learnability property for PDFAs that is shown (Clark and Thollard 2004) for state-merging frequency-based algorithms, such as ALERGIA.

4.5.2 Evidence-based VGram (EVGram)

The n-gram model, which assumes the probability of the occurrence of a letter a_i is determined by the preceding $n-1$ letters $a_{i-n+1} \dots a_{i-1}$, is a naive but powerful model for the prediction of sentences. A sequence of letters of length n is called an *n-gram*. A typical elaboration of an n-gram model is a variable-length gram model (VGram), which uses grams of different length. In the literature, a variety of criteria for determining the length of a gram to be used has been proposed. The criterion of our algorithm, Evidence-based VGram (EVGram), is also based on marginal probabilities, which is essentially the same as Eq. (21), since a variable-length gram model can be seen as a special case of a PDFA:

$$\Pr(\square|Q) = \prod_{w \in Q'} \frac{\prod_a \Gamma(C_{wa} + \beta)}{\Gamma(C_w + A\beta)} \frac{1}{R(\beta)},$$

where $Q' = \{w \mid wa \in Q\}$ where Q is the bag of grams. Similarly to EStateMerge, the prefix trie corresponding to Q is expanded at leaves greedily if $\Pr(\square|Q)$ increases.

4.5.3 Comparison of evidence-based VGram and cross-validation-based VGram

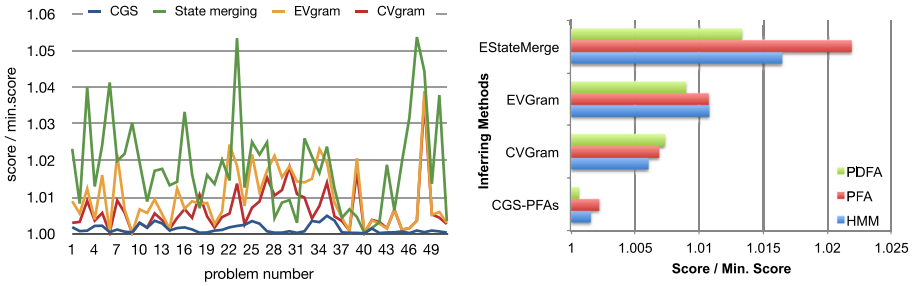
We also implemented a variant of Niesler and Woodland's (1999) VGram algorithm, the criterion of which is based on cross-validation. We call it CVGram and compared it with EVGram.

We used 10-fold CVGram, since it empirically gives a better result than the leave-one-out CVGram, which Niesler and Woodland (1999) used. Generally, the leave-one-out CV decreases the bias of estimated predictive probabilities, although the 10-fold CV is considered sufficient in many cases, as discussed in Kohavi (1995). If the number of blocks of CV is too small, it underestimates the accuracy of a learning method. On the other hand, CV is known to cause over-fitting for model selection due to variance in the estimated predictive probabilities (Cawley and Talbot 2010; Bengio and Grandvalet 2004). Thus, there is a trade-off between underestimating bias and over-fitting due to variance, and such a trade-off is one of the reasons why the results of the 10-fold CVGram are better than those of the leave-one-out CVGram.

As Fig. 11(b) shows, CVGram performed better than EVGram overall. This implies, at least for VGrams, that the model selection based on MP is not necessarily better than others, such as those based on CV.

4.5.4 Comparison of CGS-PFA, EStateMerge, and EVGram

Figure 11(a) shows the scores achieved by the above methods for all the problems of PAutomaC I. For normalizing, they are divided by the minimum scores, which are given by



(a) Scores achieved by each method

(b) Average scores summarized by the source class (PDFA, PFA or HMM)

Fig. 11 Comparison of EStateMerge, EVGram, CGS-PFA, and CVGram for 50 data sets in PAutomaC I

substituting the true probabilities of the test set in Eq.(19). As a result, CGS-PFA performed the best. The average ratio of its scores to the theoretical minimum scores exceeded 1 by 0.00129. CVGram, EVGram, and EStateMerge achieved 0.00642, 0.00992, and 0.0185, respectively. Figure 11(b) summarizes the scores of these methods for three types of generating models, PDFAs, PFAs, and HMMs. For any type of generating model, CGS-PFA yields a higher-level accuracy than the other methods. While the scores of EVGram and CVGram do not show significant differences among different target generating models, CGS-PFA and EStateMerge obtained significantly better scores on the problems generated by PDFAs than did other models.

4.6 Relation between computational costs and scores for different methods

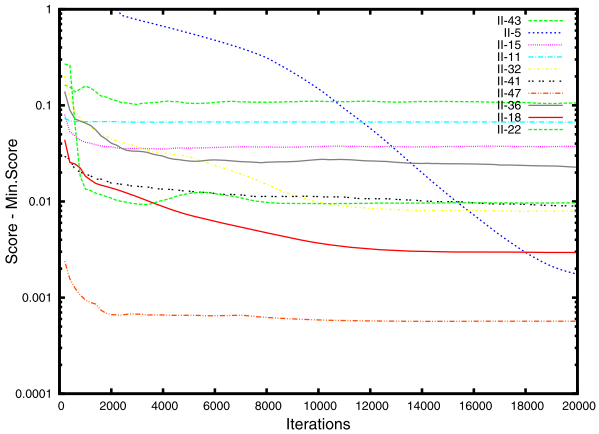
We discuss the relation between the computational costs and the accuracy for CGS, StateMerge, CVGram, and EVGram using a subset of problems of PAutomaC II (Table 3) in this section. The problems were chosen such that we had a sufficient variety of data sizes. The number N of states for each problem used in CGS was chosen beforehand in the preparatory stage described in Sect. 4.1.

4.6.1 How many iterations should suffice in CGS?

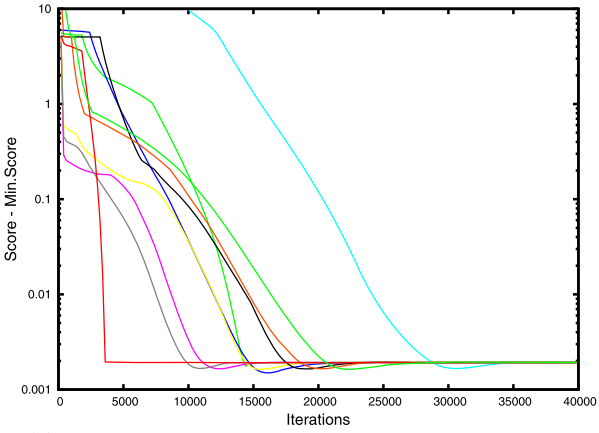
The time complexity of CGS is $O(NTL)$, where N is the number of states, T is the sample size, and L is the number of iterations. We determined N in the preparatory experiment, but we had no particular criterion by which to determine L. Figure 12(a) shows how the scores vary by changing L for different problems from PAutomaC II, where the burn-in period was set at $L/2$. We used the score achieved by a trial of CGS, which may largely have depended on the initial value. Most curves become fairly flat after 10,000 iterations, although some (II-5) continue to decrease until around the 20,000th iteration.

We further investigated the relation between the number of iterations and the score on Prob. II-5 and 43 by running CGS 10 times with different initial values. Figure 12(b) and (c) shows the results for Prob. II-5 and 43, respectively. The shapes of the score curves largely depend on the initial values, particularly for Prob. II-5, but, unlike in Fig. 12(a), not many curves seem to be converged before the 10,000th for this problem. On the other hand, in Fig. 12(c), all the curves gather and are tangled. For Prob. II-43, the choice of the initial value does not seem to be very important.

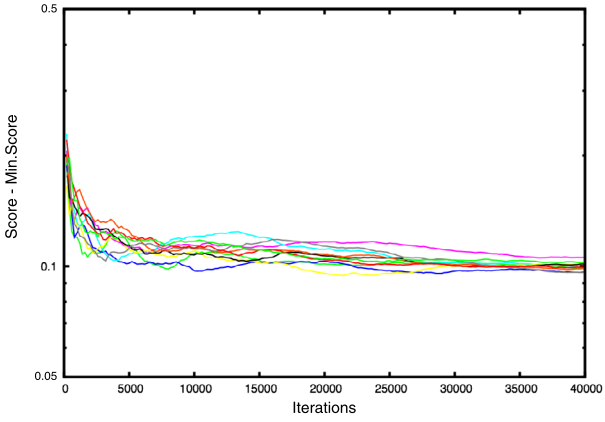
Fig. 12 Relation between iteration numbers and scores (Color figure online)



(a) Relation between the scores and the number of iterations for different problems



(b) Score curves by different initial values on Prob. II-5



(c) Score curves by different initial values on Prob. II-43

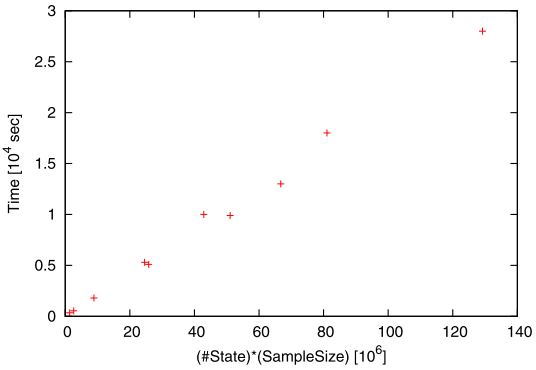


Fig. 13 Computational time of the 10,000 iterations of CGS for each data set as a function of (the number of states) × (sample size)

We conclude that the number of iterations that suffices for convergence depends on the initial value and the problem. At least for the problems of PAutomaC, the number $L = 20{,}000$ seems sufficient, in general, although more iterations might improve the scores for some limited number of problems.

4.6.2 Comparison of computational cost of different methods

Table 3 shows the computation time, the score, and the number of states of different methods, CGS-PFA, EStateMerge, CVGram, and EVGram, for some problems from PAutomaC II. We used a different number for L for CGS where we executed just one trial. Apparently, CGS took much more time, while the scores are much better than those of the other methods, in general. As expected, CGS ran in a time proportional to NTL (Fig. 13), whereas it is not necessarily the case for other methods that the time is proportional to NT. In most problems, CGS with $L = 10{,}000$ received much better scores than the other methods, and these scores are not significantly improved in CGS with $L = 20{,}000$, except for Prob. II-5. As Fig. 12(b) shows, the performance on this problem strongly depends on the initial value. However, most curves get close enough to the convergence point after the 20,000th iteration. We remark that in the actual implementation we ran CGS with different initial values in parallel and used the average. This reduces the effect of unlucky choices of initial values, as discussed in Sect. 4.1.

The scores of CGS in Table 3 for Prob.II-5 are of the trial corresponding to the green line in Fig. 12(b).

Among these methods, StateMerge often achieves scores significantly worse than do the other methods, while it often succeeded in finding a concise automaton as compared to C/EVGram. The number of the states of a VGram tree constructed by EVGram sometimes becomes even bigger than the sample size.

5 Conclusions and future work

In this paper, we compared various collapsed Bayesian methods for PFAs and their variants, including HMMs, PDFAs, and VGrams. For fully connected PFAs, we discussed how existing techniques of Collapsed Gibbs Sampling (CGS) and Collapsed Variable Bayes with the 0th-order Taylor approximation (CVB0) can be applied, and in addition, we proposed a new method called GCVB0 for which the convergence is ensured. While CVB2, the CVB with

Table 3 Comparison of computational time of different methods

Prob.	SampleSize	CGS(10k)			CGS(20k)			ESMerge		
		sec.	score	N	sec.	score	N	sec.	score	N
II-43	87,107	540	0.1105	30	1,100	0.1062	30	5.7	0.9464	13
II-5	136,094	350	0.1792	10	680	0.0019	10	1.4	2.4717	9
II-15	222,182	1,800	0.0371	40	3500	0.0375	40	120	0.5464	174
II-11	307,062	5300	0.0670	80	11,000	0.0670	80	85	0.2934	47
II-32	476,902	10,000	0.0098	90	20,000	0.0079	90	13	1.8411	40
II-41	645,907	5,100	0.0112	40	10,000	0.0089	40	130	0.1500	30
II-47	900,567	18,000	0.0006	90	36,000	0.0006	90	170	0.0538	61
II-36	1,021,088	9,900	0.0273	50	20,000	0.0228	50	670	0.4523	127
II-18	1,335,090	13,000	0.0037	50	25,000	0.0030	50	250	0.9027	24
II-22	1,615,065	28,000	0.0095	80	65,000	0.0097	80	7,600	0.2876	583

Prob.	CVGram			EVGram		
	sec.	score	N	sec.	score	N
II-43	1.6	0.1648	79	0.89	0.1828	43
II-5	1.1	0.0135	99	0.87	0.0135	99
II-15	9.7	0.6289	29,776	14	0.7015	597,556
II-11	80	0.9273	171,886	584	1.0444	7,229,083
II-32	3.4	0.1004	19,616	3.6	0.1323	15,446
II-41	2.0	0.0272	377	2.4	0.0887	105
II-47	18	0.0049	37,377	31	0.0065	644,417
II-36	4.9	0.2848	5,931	7	0.5153	33,411
II-18	8.9	0.0643	8,023	11	0.0587	62,245
II-22	130	0.1279	214,303	260	0.2110	6,039, 155

the second-order approximation, may appear to yield a more accurate probability prediction, its computational cost per iteration is evaluated as $O(N^3T)$, which is not sufficiently efficient unless the size of target automata is restricted to be very small. In contrast, the costs for CGS, CVB0, and GCVB0 are only $O(NT)$, $O(N^2T)$, and $O(N^2T)$, respectively. Hence, these methods can be applied to bigger PFAs. According to the experimental results for PAutomaC data sets, CGS performed better than CVB0 and GCVB0. Although GCVB0 is guaranteed to converge to some local optimal point and thus it is clear when its iterations should be stopped, the results of GCVB0 were worse than those of CVB0 and CGS. For sparse PFAs, by using a simple generative model, an algorithm for sampling graph structures of PFAs is proposed.

We also empirically compared algorithms that targeted different types of PFAs using PAutomaC data sets generated by different types of PFAs. In the comparison of CGS-PFA and CGS-HMM, it appeared that CGS-PFA achieves better scores than CGS-HMM, since CGS-HMM often failed to find appropriate emission probabilities η and state transition probabilities θ that can factorize the transition probability ξ. CGS-PFA gives better scores than EStateMerge, CVGram, and EVGram for every generating model, whereas EStateMerge and EVGram run much faster than CGS-PFA, since they change the graph structures in order to maximize marginal probabilities greedily. Graph structures for PDFAs and

VGram should have been sampled in the Bayesian manner. Our conclusion is that, empirically, CGS-PFA is the best choice among the collapsed methods described in this paper.

Many methods for inferring PFAs still remain to be compared with the methods we described in this paper. For instance, although we fixed the numbers of states based on cross validation in this study, the numbers can be sampled simultaneously with values of $\mathbb{z}$ in nonparametric methods. The comparison of our methods with nonparametric methods, such as HDP (Teh et al. 2006a), on PAutomaC data constitutes future work. In our experiments, EStateMerge did not perform better than CGS in terms of accuracy, even on sample sets generated by PDFAs. There is no guarantee that EStateMerge will PAC-learn PDFAs, since it merges states greedily according to marginal probabilities. Since other state-merging techniques for which PAC learnability is proven might yield more accurate results, we should compare them with the methods examined in this paper using data sets generated from PDFAs.

Acknowledgements We are grateful to the committee of PAutomaC for offering various useful data sets and detailed information on them. We deeply appreciate the valuable comments and suggestions of the anonymous reviewers, which improved the quality of this paper.

Appendix: The second-order CVB for PFAs

In order to apply the second-order Taylor approximation for CVB (CVB2) to PFAs, we have to calculate the variance over q for the terms $C^{\neg t}_{k}$, $C^{\neg t}_{k00}$, $C^{\neg t}_{ka_t z_{t+1}}$, and $C^{\neg t}_{z_{t-1}a_{t-1}z_{t+1}} + \delta\left(\begin{smallmatrix} k, & a_t, & z_{t+1} \\ z_{t-1}, & a_{t-1}, & k \end{smallmatrix}\right)$.

The first term is represented in a simple form, since the variation of the summation of independent variables equals the summation of the variations of independent variables:

$$V_q\left[C^{\neg t}_{k}\right] = \sum_{s \neq t} Ver_q\left[\delta\begin{pmatrix} z_t \\ k \end{pmatrix}\right] = \sum_{s \neq t}\left(E_q\left[\left(\delta\begin{pmatrix} z_t \\ k \end{pmatrix}\right)^2\right] - E_q\left[\delta\begin{pmatrix} z_t \\ k \end{pmatrix}\right]^2\right)$$
$$= \sum_{s \neq t} q_s(k)\big(1 - q_s(k)\big).$$

However, the other terms have complicated forms, since $\delta\left(\begin{smallmatrix} z_s, & a_s, & z_{s+1} \\ k, & a_t, & z_{t+1} \end{smallmatrix}\right)$ is not independent for each s. We calculate the variance of the third term using Lemma 1 and discuss its computational cost below.

$$V_q\left[C^{\neg t}_{ka_t z_{t+1}}\right] = E_q\left[\left(C^{\neg t}_{ka_t z_{t+1}}\right)^2\right] - E_q\left[C^{\neg t}_{ka_t z_{t+1}}\right]^2$$
$$= \sum_{u \neq t, t-1} \sum_{v \neq t, t-1} E_q\left[\delta\begin{pmatrix} z_u, & a_u, & z_{u+1}, & z_v, & a_v, & z_{v+1} \\ k, & a_t, & z_{t+1}, & k, & a_t, & z_{t+1} \end{pmatrix}\right]$$
$$- E_q\left[C^{\neg t}_{ka_t z_{t+1}}\right]^2. \tag{22}$$

Let $e(u, v) = E_q\left[\delta\left(\begin{smallmatrix} z_u, & a_u, & z_{u+1}, & z_v, & a_v, & z_{v+1} \\ k, & a_t, & z_{t+1}, & k, & a_t, & z_{t+1} \end{smallmatrix}\right)\right]$. To apply Lemma 1, $e(u, v)$ should be classified by how $\sim$ partitions them. Ensuring the symmetry of u and v and $\{u, v\} \cap \{t-1, t\} = \varnothing$, we can classify $e(u, v)$ to $e_1(u, v), \ldots, e_5(u, v)$ as

$$e_1(u, v) = E_q\left[\delta\begin{pmatrix} z_u, & a_u, & z_{u+1} \\ k, & a_t, & z_{t+1} \end{pmatrix}\right] \quad \text{if } v = u \text{ and } u \neq t+1,$$

$$e_2(u,v) = E_q\left[\delta\begin{pmatrix} z_{t+1}, & a_{t+1}, & z_{t+2} \\ k, & a_t, & z_{t+1}\end{pmatrix}\right] \quad \text{if } v = u \text{ and } u = t+1,$$

$$e_3(u,v) = E_q\left[\delta\begin{pmatrix} z_u, & a_u, & z_{u+1}, & z_{u+1}, & a_{u+1}, & z_{u+2} \\ k, & a_t, & z_{t+1}, & k, & a_t, & z_{t+1}\end{pmatrix}\right]$$
$$\text{if } v = u+1 \text{ and } u \neq t+1,$$

$$e_4(u,v) = E_q\left[\delta\begin{pmatrix} z_{t+1}, & a_{t+1}, & z_{t+2}, & z_{t+2}, & a_{t+2}, & z_{t+3} \\ k, & a_t, & z_{t+1}, & k, & a_t, & z_{t+1}\end{pmatrix}\right]$$
$$\text{if } v = u+1 \text{ and } u = t+1,$$

$$e_5(u,v) = E_q\left[\delta\begin{pmatrix} z_u, & a_u, & z_{u+1} \\ k, & a_t, & z_{t+1}\end{pmatrix}\right] E_q\left[\delta\begin{pmatrix} z_v, & a_v, & z_{v+1} \\ k, & a_t, & z_{t+1}\end{pmatrix}\right] \quad \text{if } |v-u| > 1.$$

Because $e_5(u,v)$, except for $|v-u|>1$, equals $E_q[C^{\neg t}_{ka_tz_{t+1}}]^2$, which is the second term of Eq. (22), we have to calculate $e_5(u,v)$ when $|v-u| \leq 1$ and subtract them from $e_1(u,v) + \cdots + e_4(u,v)$. By Lemma 1, we have

$$e_1(u,v) = p_u(k)\sum_i p_{u+1}(i)p_{t+1}(i)\delta\begin{pmatrix} a_u \\ a_t\end{pmatrix} \quad \text{if } v = u \quad \text{and} \quad u \neq t+1,$$

$$e_2(u,v) = p_{t+1}(k)p_{t+2}(k)\delta\begin{pmatrix} a_{t+1} \\ a_t\end{pmatrix} \quad \text{if } v = u \text{ and } u = t+1,$$

$$e_3(u,v) = p_u(k)p_{u+1}(k)p_{t+1}(k)p_{u+2}(k)\delta\begin{pmatrix} a_u, & a_{u+1} \\ a_t, & a_t\end{pmatrix} \quad \text{if } v = u+1 \text{ and } u \neq t+1,$$

$$e_4(u,v) = p_{t+1}(k)p_{t+2}(k)p_{t+3}(k)\delta\begin{pmatrix} a_{t+1}, & a_{t+2} \\ a_t, & a_t\end{pmatrix} \quad \text{if } v = u+1 \text{ and } u = t+1.$$

$e_5(u,v)$, where $|v-u| \leq 1$, is again classified according to $\sim$, and therefore, we have

$$e_5(u,v) = \left(p_u(k)\sum_i p_{u+1}(i)p_{t+1}(i)\right)^2\delta\begin{pmatrix} a_u \\ a_t\end{pmatrix} \quad \text{if } v = u \text{ and } u \neq t+1,$$

$$e_5(u,v) = \left(p_{t+1}(k)p_{t+2}(k)\right)^2\delta\begin{pmatrix} a_{t+1} \\ a_t\end{pmatrix} \quad \text{if } v = u \text{ and } u = t+1,$$

$$e_5(u,v) = p_u(k)\left(\sum_i p_{u+1}(i)p_{t+1}(i)\right)p_{u+1}(k)\left(\sum_j p_{u+2}(j)p_{t+1}(j)\right)\delta\begin{pmatrix} a_u, & a_{u+1} \\ a_t, & a_t\end{pmatrix}$$
$$\text{if } v = u+1, u \neq t+1,$$

$$e_5(u,v) = p_{t+1}(k)p_{t+2}(k)p_{t+2}(k)\sum_i p_{t+3}(i)p_{t+1}(i)\delta\begin{pmatrix} a_{t+1}, & a_{t+2} \\ a_t, & a_t\end{pmatrix}$$
$$\text{if } v = u+1, u = t+1.$$

By summing $e_1 + e_2 + e_3 + e_4 - e_5$ with respect to u and v,

$$\mathrm{Var}_q\left[C^{\neg t}_{ka_tz_{t+1}}\right]$$

$$
\begin{aligned}
&= \sum_{u\neq t,t\pm 1} \delta\begin{pmatrix} a_u \\ a_t \end{pmatrix} p_u(k) \sum_i p_{u+1}(i)p_{t+1}(i)\Bigg(1 - p_u(k)\sum_j p_{u+1}(j)p_{t+1}(j)\Bigg) \\
&\quad + \delta\begin{pmatrix} a_{t+1} \\ a_t \end{pmatrix} p_{t+1}(k)p_{t+2}(k)\big(1 - p_{t+1}(k)p_{t+2}(k)\big) \\
&\quad + 2\sum_{u\neq t,t\pm 1} \delta\begin{pmatrix} a_u, & a_{u+1} \\ a_t, & a_t \end{pmatrix} p_u(k)p_{u+1}(k)\Bigg(p_{t+1}(k)p_{u+2}(k) \\
&\quad - \sum_i p_{u+1}(i)p_{t+1}(i)\sum_j p_{u+2}(j)p_{t+1}(j)\Bigg) \\
&\quad + 2\delta\begin{pmatrix} a_{t+1}, & a_{t+2} \\ a_t, & a_t \end{pmatrix} p_{t+1}(k)p_{t+2}(k)\Bigg(p_{t+3}(k) - p_{t+2}(k)\sum_i p_{t+3}(i)p_{t+1}(i)\Bigg).
\end{aligned}
$$

Not only is the above representation complicated, but also its computational cost is high. If we use the technique for reducing the computational cost similar to that of CVB0, we still have to update $O(N^3)$ values in order to keep the following terms:

$$\sum_u \delta\begin{pmatrix} a_u \\ a \end{pmatrix} p_u(k)p_{u+1}(i)p_{u+1}(j),$$

$$\sum_u \delta\begin{pmatrix} a_u, & a_{u+1} \\ a, & a \end{pmatrix} p_u(k)p_{u+1}(k)p_{u+1}(i)p_{u+2}(j).$$

Thus, CVB2 costs $O(N^3T)$.

References

Abe, N., & Warmuth, M. K. (1992). On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, *9*, 205–260.

Asuncion, A. U., Welling, M., Smyth, P., & Teh, Y. W. (2009). On smoothing and inference for topic models. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence* (pp. 27–34).

Balle, B., Castro, J., & Gavaldà, R. (2013). Learning probabilistic automata: a study in state distinguishability. *Theoretical Computer Science*, *473*, 46–60.

Beal, M. J. (2003). *Variational algorithms for approximate Bayesian inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London.

Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of K-fold cross-validation. *Journal of Machine Learning Research*, *5*, 1089–1105.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer. Chap. 11.

Carrasco, R. C., & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the second international colloquium of grammatical inference* (pp. 139–152).

Castro, J., & Gavaldà, R. (2008). Towards feasible pac-learning of probabilistic deterministic finite automata. In *Proceedings of the 9th international colloquium on grammatical inference* (pp. 163–174).

Cawley, G. C., & Talbot, N. L. C. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, *11*, 2079–2107.

Clark, A., & Thollard, F. (2004). PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, *5*, 473–497.

Gao, J., & Johnson, M. (2008). A comparison of Bayesian estimators for unsupervised hidden Markov model POS taggers. In *Proceedings of the 2008 conference on empirical methods in natural language* (pp. 344–352).

Goldwater, S., & Griffiths, T. (2007). A fully Bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of the 45th annual meeting of the association of computational linguistics* (pp. 744–751).

Hsu, D., Kakade, S. M., & Zhang, T. (2009). A spectral algorithm for learning hidden Markov models. In *Proceedings of the 22nd conference on learning theory*.

Johnson, M., Griffiths, T. L., & Goldwater, S. (2007). Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of human language technology conference of the North American chapter of the association of computational linguistics* (pp. 139–146).

Kearns, M. J., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R. E., & Sellie, L. (1994). On the learnability of discrete distributions. In *Proceedings of the 26th annual ACM symposium on theory of computing* (pp. 273–282).

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the fourteenth international joint conference on artificial intelligence* (pp. 1137–1145).

Liang, P., Petrov, S., Jordan, M. I., & Klein, D. (2007). The infinite PCFG using hierarchical Dirichlet processes. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning* (pp. 688–697).

Niesler, T., & Woodland, P. C. (1999). Variable-length category n-gram language models. *Computer Speech & Language*, *13*(1), 99–124.

Pfau, D., Bartlett, N., & Wood, F. (2010). Probabilistic deterministic infinite automata. In *Advances in neural information processing systems 23 (NIPS)* (pp. 1930–1938).

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proceedings of the 44th annual meeting of the association of computational linguistics* (pp. 985–992).

Teh, Y. W., Jordan, M. I., Beal, M. J., & Blei, D. M. (2006a). Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, *101*(476), 1566–1581.

Teh, Y. W., Newman, D., & Welling, M. (2006b). A collapsed variational Bayesian inference algorithm for latent Dirichlet allocation. In *Advances in neural information processing systems 19 (NIPS)* (pp. 1353–1360).

Thollard, F. (2001). Improving probabilistic grammatical inference core algorithms with post-processing techniques. In *Proceedings of the eighteenth international conference on machine learning* (pp. 561–568).

Thollard, F., Dupont, P., & de la Higuera, C. (2000). Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proceedings of the seventeenth international conference on machine learning* (pp. 975–982).

Verwer, S., Eyraud, R., & de la Higuera, C. (2012). Results of the PAutomaC probabilistic automaton learning competition. In *Proceedings of the 11th international conference on grammatical inference* (Vol. 21, pp. 243–248).

Zhu, C., Byrd, R. H., Lu, P., & Nocedal, J. (1997). Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, *23*(4), 550–560.