Probabilistic Size-constrained Microclustering

Arto Klami
Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki, Finland

Aditya Jitta
Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki, Finland

Abstract

Microclustering refers to clustering models that produce small clusters or, equivalently, to models where the size of the clusters grows sublinearly with the number of samples. We formulate probabilistic microclustering models by assigning a prior distribution on the size of the clusters, and in particular consider microclustering models with explicit bounds on the size of the clusters. The combinatorial constraints make full Bayesian inference complicated, but we manage to develop a Gibbs sampling algorithm that can efficiently sample from the joint cluster allocation of all data points. We empirically demonstrate the computational efficiency of the algorithm for problem instances of varying difficulty.

1 INTRODUCTION

Clustering is classically identified as the task of grouping similar objects into a number of clusters. Typical probabilistic formulations are based on mixture models [McLachlan and Peel, 2004], following the intuitive idea that each data point independently chooses a cluster by considering the product of some mixture weights and the likelihood of that cluster producing the data point. Because of these independent decisions, these models have very limited means for controlling the sizes of the clusters. Parametric mixture models are effectively agnostic of the cluster size; they can be tuned to a degree with the prior distribution of the weights, but in the end the likelihood overrides even the strongest priors for large sample sizes. Nonparametric mixture models such as Dirichlet process (DP) mixtures [Antoniak, 1974], Pitman-Yor mixtures [Dubey et al., 2014], and uniform process mixtures [Wallach et al., 2010] in turn induce some characteristic behavior for the size distribution. For example, DP mixtures assume that the number of clusters for a sample size of N grows proportionally to $\log(N)$ and the biggest clusters hold a large proportion of the data points due to the rich-get-richer property, which is not always consistent with reasonable clustering assumptions [Miller and Harrison, 2013]. The uniform process by Wallach et al. [2010] eliminates the rich-get-richer property, but in expectation produces clusters of all sizes with equal probability, resulting in total number of clusters proportional to $\sqrt{N}$.

Both being agnostic about the sizes of the clusters and explicitly assuming the rich-get-richer property are reasonable assumptions for a wide range of clustering tasks, as clearly demonstrated by the wide-spread usage of mixture models. For some clustering tasks, however, it would be useful to have finer control over the possible sizes of the clusters. For example, if the clustering model is used for creating teams of similar individuals [Kim et al., 2015] we would want to find clusters that are approximately of the same size, even as small as two to create working pairs. Another practical example is to use clustering to split a collection of items for further processing for distributed set of workers with limited capacity. We can expect the worker to perform the task more accurately and faster if the items are similar, and the clusters should contain at most a certain pre-given number of instances.

In this work, we consider probabilistic clustering models with explicit constraints on the sizes to address the illustrative scenarios above. Our model that assumes constant maximum size is a specific instance of the more general concept of *microclustering*, defined as clustering models for which the size of the clusters grows sublinearly with respect to the sample size [Miller et al., 2015] – here the size is explicitly forced to be constant, to guarantee small clusters even for extremely large data collections. Our microclustering model builds on standard parametric mixture models, but replaces the independent cluster assignments of individual data points with a joint assignment of all points that needs to satisfy the constraints.

Specifying the model for size-constrained microclustering is straightforward, requiring only a simple change in the prior distribution, and in fact similar models have earlier been discussed outside the probabilistic framework.

Banerjee [2006] considered distance-based clustering models that produce clusters of approximatively the same size, and Zhu et al. [2010] added explicit constraints to K-means to utilize prior knowledge on the cluster sizes. Their solutions correspond effectively to maximum likelihood estimation, which is easy given access to standard constrained optimization tools, whereas the core challenge in our case is in conducting full Bayesian inference over the model.

In this work we discuss the challenge of jointly sampling the cluster assignments of all data points in our microclustering model and present two alternative algorithms for that purpose, a dynamic programming algorithm based on depth-first branch-and-bound, and a simple rejection sampler. We empirically demonstrate that these two alternatives dominate in different conditions, the former being efficient for highly constrained cases and the latter being optimal for loose constraints. We then construct a final sampling algorithm that utilizes both parts to produce a Gibbs sampler that is a direct generalization of the corresponding algorithm for regular mixture models: In the absence of constraints it draws the samples independently with no computational overhead, whereas with increasingly tight constraints it starts using the dynamic programming algorithm for improved efficiency.

Besides introducing the size-constrained clustering model, the main message of this article is to highlight that full Bayesian inference is also feasible for models with combinatorial constraints, encouraging people to explore the opportunities outside independent samples. Even though probabilistic reasoning under combinatorial constraints is generally hard [Roth, 1996], many practical probabilistic models result in sufficiently small problem instances that can today be solved efficiently. Explicit constraints can hence be effective in constraining the model and often the computational overhead needed for finding the solution is small compared to the gain in overall performance. We are currently aware of only limited existing literature in this direction; Chen et al. [2005] and Dobra et al. [2006] presented MCMC algorithms for sampling contingency tables with constrained marginals, Klami [2012, 2013] considered posterior inference over permutations to solve cross-domain object matching problems, and Wang et al. [2015] presented an MCMC algorithm for combinatorial problems related to phylogenetic trees.

2 MICROCLUSTERING

Miller et al. [2015] define microclustering as any clustering model where the size of the clusters grow sublinearly with the total number of data points. Even though their formulation considers non-parametric models, the concept itself is useful also for parametric models where the number or size of the clusters is chosen via other means during the modeling process.

The core requirement for building probabilistic microclustering models is to have control over the sizes of the clusters. Regular mixture models have no control over the size distribution besides the prior weights (that can be dominated by the likelihood) because their independent data point assignment prior

$$p(\{z_n\}_{n=1}^N|\theta) = \prod_{n=1}^{N} p(z_n|\theta) \tag{1}$$

implies that the conditional posterior $p(z_n|\{z_n\}_{-n},\theta) = p(z_n|\theta)$ is independent of the other assignments. Hence the sizes of the clusters cannot influence the decision of the individual sample. Marginalizing the parameters θ out introduces such a dependency, but its nature is completely determined by the prior used on θ and cannot be controlled easily. See Wallach et al. [2010] for both theoretical and empirical analysis of the resulting cluster size distributions for various non-parametric prior processes.

Explicit control over the cluster sizes is conceptually easy to obtain, by replacing the prior in (1) with one that factorizes over the clusters and not the samples:

$$p(\{z_n\}_{n=1}^N|\theta) = \prod_{k=1}^{K} p(s_k|\theta),$$

where s_k is the number of data points assigned to the kth cluster. It is linked to the assignments as $s_k = \sum_{n=1}^N \mathbb{I}(z_n = k)$ where $\mathbb{I}(\cdot)$ evaluates to one if its argument is true and otherwise to zero. In other words, we assume that all joint assignments that result in cluster sizes $\{s_k\}_{k=1}^K$ are equally probable, and their number does not directly influence the probability.

This general formulation leaves open the specific prior given for the sizes. Miller et al. [2015] introduced a non-parametric microclustering model that assumes the cluster sizes follow a negative binomial distribution, whereas we will be using constant priors over a set of legal cluster sizes. One might also imagine other practical choices, such as constant probability for some favoured cluster size with exponential decay for violations from that size. In general, this choice will be application-specific and hence the priors should be subjective, rather unconventionally for mixture modeling in general.

2.1 SIZE-CONSTRAINED MICROCLUSTERING

In this work, we consider microclustering models with explicit hard constraints on the cluster sizes, applicable for scenarios where the (typically maximum) size of a clusters is determined by external channel constraints. These clustering models clearly belong to the family of microclustering models, since the size of the clusters is constant with respect to the total number of data points and hence sublinear. The explicit constraints are easy to formulate but require constrained optimization techniques for inference.

Our model that restricts the sizes between L and U is

$$p(\{x_n\}|\phi,\{z_n\}) = \prod_{n=1}^{N} p(x_n|\phi,z_n) = \prod_{n=1}^{N} g(x_n,\phi_{z_n}),$$
$$p(\{z_n\}) = \prod_{k=1}^{K} p(s_k), \tag{2}$$
$$p(s_k) = \frac{1}{U-L+1}\delta(L \le s_k \le U),$$

where $g(x_n,\phi_{z_n})$ is some likelihood function and the model is coupled with a suitable prior $p(\phi)$ on its parameters. In our experiments, we will use the Gaussian likelihood $\log g(x_n,\phi_k) = C - \frac{1}{2}(x_n-\mu_k)^T\tau(x_n-\mu_k)$ with diagonal precision τ and priors $\mu \sim N(\mu_0,\Sigma_0)$ and $\tau_d \sim \text{Gamma}(\alpha_0,\beta_0)$, but the core inference algorithms for $\{z\}$ works identically for any likelihood that factorizes over the samples. Here the constraints U and L are constant over the clusters, but all of the inference details apply also for cluster-specific constraints if such prior information is available.

3 INFERENCE

We now discuss the full Bayesian inference for the proposed model. The practical algorithmic details are given for a Gibbs sampler that samples the parameters ϕ of the clusters given the assignments and the assignments $\{z_n\}$ given the cluster parameters. The sampling equations for the cluster parameters are exactly as in any standard mixture model, and are not discussed here in any more detail.

The challenging part is sampling the assignments, which needs to be done jointly for all data points due to the prior distribution and constraints (2) defined for the whole collection instead of individual points. We will first briefly explain how the maximum likelihood solution is easy to find, and then proceed to present two alternatives algorithms for producing samples from the posterior distribution.

As a side remark, the microclustering model of Miller et al. [2015] sidesteps the issue of joint sampling by sampling the allocations of individual samples conditional on all others allocations, from $p(z_n|z_{-n})$. This is feasible in their model that does not have hard constraints, but results in long auto-correlation time due to aggressive conditioning. For our model such a sampler would be catastrophic if the constraints are tight; in the extreme case where the cluster sizes are forced to exact values it could never change the allocation since all clusters except the one where this data point was previously allocated at would be full.

3.1 THE MOST LIKELY ASSIGNMENT(S)

An important initial observation is that we can efficiently find the most likely assignment by solving the integer programming problem

$$\max \sum_{n=1}^{N}\sum_{k=1}^{K} \log g(x_n,z_k)\pi_{k,n}, \tag{3}$$
$$s_k = \sum_n \pi_{k,n} \ge L \quad \forall k,$$
$$s_k = \sum_n \pi_{k,n} \le U \quad \forall k,$$
$$\sum_k \pi_{k,n} = 1 \quad \forall n,$$

where π is a binary matrix whose element $\pi_{k,n}$ indicates whether the nth sample is assigned to the kth cluster. Any off-the-shelf linear programming solver will find the optimal solution in reasonable time for problems of practical size. By alternating between assignments obtained by solving (3) and maximum a posteriori choice for the cluster parameters ϕ_k, we would get a probabilistic variant of the size-constrained K-means model by Zhu et al. [2010].

Typical branch-and-bound algorithms used for solving (3) retain a list of solution candidates that are pruned away by comparing upper bounds for their value against a lower bound for the best candidate. They can be easily modified to retain a list of all possible solutions that are close enough to the optimal, to explicitly enumerate all solutions that are sufficiently likely. Given such a list of solutions $\{\pi^i\}$ and their associated log-probabilities $\{c^i\}$ we could easily sample a solution from $p(\pi=\pi^i) = \frac{\exp(c^i)}{\sum_i \exp(c^i)}$.

Unfortunately, explicitly enumerating all of the good solutions is infeasible for all but the smallest problems because their number becomes inordinately large. In the unconstrained case there are N^K possible allocations, all of which would need to be enumerated if the probabilities fall off of too slowly. We do not discuss this approach further since the cases for which it would be efficient are easy to solve by other means as well, but the optimization problem (3) is still important; we will use it for quickly creating an upper bound in our actual sampler, as well as for initializing the sampling chain.

3.2 CLUSTER SIZE ASSIGNMENT SAMPLER

3.2.1 Motivation

A more practical solution to the problem is a dynamic programming algorithm that operates in the space of possible cluster sizes, enumerating only the possible cluster size allocation vectors $r \in [L,U]^K$ instead of the sample allocation vectors $z \in [1,K]^N$. Even in the unconstrained case the maximum number of solutions to be enumerated in the end goes down from N^K to $\binom{N+K-1}{K-1}$, and for the constrained case with uniform maximum size U we have at most $\sum_{q=0}^{\min(K,\lfloor N/(U+1)\rfloor)}(-1)^q\binom{K}{q}\binom{N-q*(U+1)+K-1}{K-1}$ solutions. As a case in point, already for $N=20$, $K=8$ and

$U = 4$ these three numbers would be 2.6×10^{10}, 888.030 and 23.940. With minimum size $L = 2$ the number of possible solutions would further decrease to just 266. Enumerating 2.6×10^{10} solutions would be clearly infeasible, whereas the result set of at most 266 solutions would cause no trouble.

Operating in the space of cluster size allocations does come with a drawback as well, in the form of more difficult bounding of the solution candidates. Furthermore, the number of solution candidates in the intermediate stages is typically considerably higher than the size of the final set, but still orders of magnitude smaller than the number of individual solutions. We will next show how a reasonably efficient dynamic programming algorithm operating in the space of the cluster sizes can be designed.

3.2.2 Basic Concept

The algorithm operates on solution sets $A_a = (R_a, q_a)$, where each set contains a collection of solution candidates $R_a = \{r_a^i, p_a^i\}$ and the total probability of the set $q_a = \sum_i p_a^i$. Each solution i is characterized by a vector of cluster sizes $r_a^i \in \mathbb{N}^K$ and the associated probability p_a^i of that particular solution. Throughout the description of the algorithm, subscripts refer to the sets and superscripts to the individual solution candidates within the set, so that p_a^i means the probability of the ith solution candidate in set A_a.[1]

We build a forward-backward sampling algorithm based on dynamic programming reminiscent of the algorithm used for sampling the state sequence of Bayesian hidden Markov models [Scott, 2002]. Similar to that algorithm we make a forward pass to accumulate total probabilities of solutions and a backward pass to sample given the accumulated probabilities after each sample. For HMMs this algorithm can cover all possibilities since it only needs to keep track of K probabilities at each stage, but since we are keeping track of all possible cluster size allocations we also need to prune out solution candidates that will have negligible probability in the final set.

The overall algorithm is illustrated in Figure 1, which also demonstrates how the practical computations are performed.

3.2.3 Forward Pass

The forward pass starts by constructing N initial sets A_n, each storing the K possible cluster allocations for one sample. The probability of each solution is given by $p_n^k = g(x_n, z_k)$, and the total probability is $q_n = \sum_k p_n^k$.

[1] In practice we naturally store the values in the logarithmic domain for numerical stability, but the presentation below uses actual probabilities to avoid needing to write $\log \sum \exp(\cdot)$ for all cases where we sum up probabilities.

The first iteration of the forward pass picks two of these sets (denoted by i and j) and joins them to create a solution set $A_{i,j}$, containing all possible allocations of the two samples, stored still as the possible cluster size allocation vectors $r_{i,j}^i$ and their probabilities $p_{i,j}^i$. This join is performed by a collection of four basic operations described soon and illustrated in Figure 1.

After joining the two sets we proceed to join the resulting set with another of the initial N sets, denoted by l, this time producing the set $A_{i,j,l}$ that stores the joint allocations of all three samples. The process continues this way for $N - 1$ iterations, until all samples have been joined to the final set $A_{1:N}$. It stores the probabilities of all possible cluster size allocations that satisfy the constraints. Together with all of the intermediate sets it enables drawing a sample from the posterior using the backward pass described in Section 3.2.5.

3.2.4 Set operations

For manipulating the sets the algorithm requires four basic operations:

1. MERGE(A_a,A_b): Takes as input two sets and combines them, to produce a new set that contains all possible combinations of the solutions in the two sets: each solution in A_a is paired with each solution in A_b, so that the cluster size vectors are summed up and the probabilities are multiplied together. Note that this typically results in duplicate solution candidates, since the same sum $r_a^i + r_b^j$ can be reached in multiple ways.

2. COLLAPSE(A_a): Takes as input a solution set A_a with possible duplicates for the cluster size vectors r_a^i and returns the set so that each unique vector is represented only once. The probability of that set is obtained by summing over the probabilities associated with each duplicate: $p_a^i = \sum_j p_a^j \quad \forall r_a^j = r_a^i$.

3. CHECKCONSTRAINTS(A_a, U_a, L_a): Takes as input a solution set and returns a set that excludes all solutions that violate the constraints. That is, we keep only solutions for which $L_a \leq r_a^i \leq U_a$ holds for all K elements.

4. BOUND(A_a, b): Takes as input a solution set and returns a set that excludes all solutions for which the probability is below the bound, $p_a^i < b$.

A single iteration of the forward pass is simply a concatenation of all of the above operations: The initial sets are passed to MERGE, the result of that to COLLAPSE, and then to CHECKCONSTRAINTS and BOUND in either order. Without bounding or checking for the constraints the algorithm would simply proceed to enumerate all possible cluster size allocations, so the efficiency of the algorithm

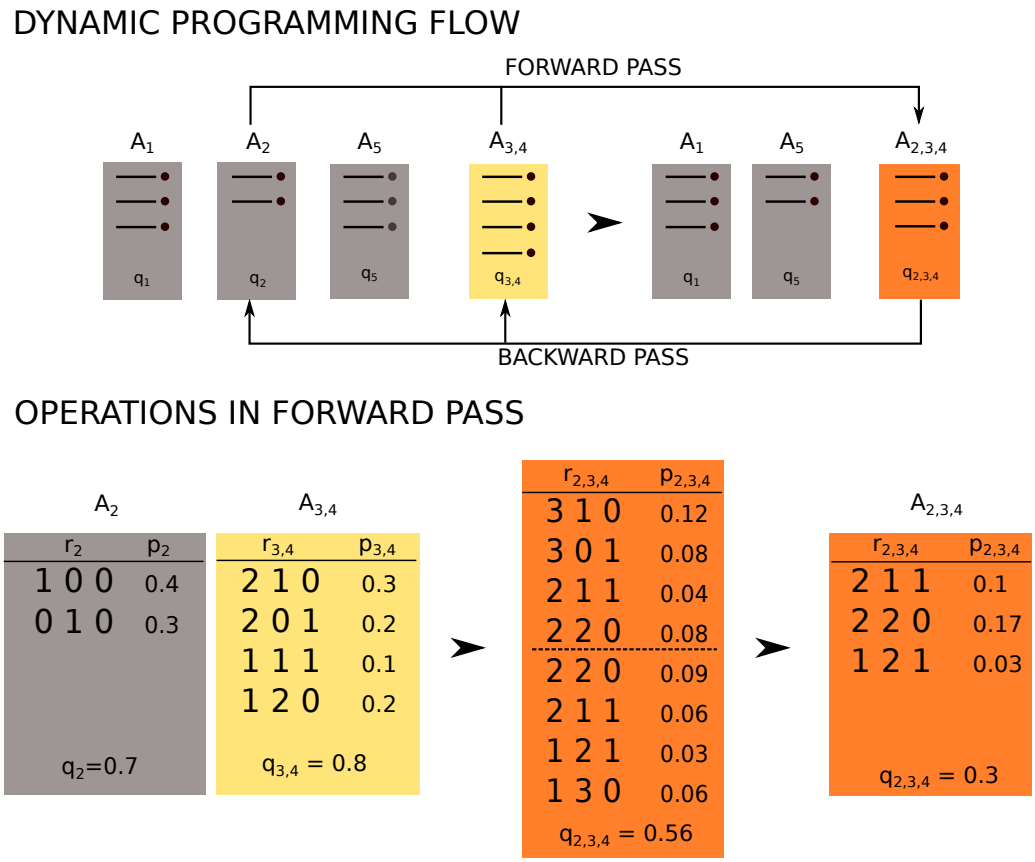

Figure 1: **Top:** Schematic overview of the dynamic programming algorithm. The forward pass computes the probabilities of possible cluster size allocation vectors by joining one sample at a time to the current overall solution set marked in yellow, producing the next overall solution set marked in orange. The backwards step then draws a sample by considering the probabilities in the intermediate sets created during the forward pass. **Bottom:** Example of an individual join. Each set stores all possible cluster size vectors r^i for the samples in that set with the associated probabilities p^i, as well as the total probability q that is their sum. The MERGE operation produces all possible combinations of the cluster size vectors of the two sets, here ordered so that the solutions above the dashed line correspond to r_2^1, and multiplies the probabilities. The COLLAPSE operation then sums up the probabilities for each unique solution, and CHECKCONSTRAINTS cuts out solutions that exceed the constraints; here solutions with cluster sizes above $U = 2$ were pruned out. Note that the total probability in the intermediate stage is simply the product of the probabilities for the incoming sets, but the probability of the final result is often considerably smaller. This allows pruning the other sets again since their upper bounds have decreased, here by $0.56 - 0.3 = 0.26$, resulting in A_5 now having only two valid solutions instead of three.

depends on how well we can cut out solution candidates based on the constraints and the bound on probabilities.

To efficiently bound the solutions in A_a we need to know an upper bound for the probability of still unprocessed samples, denoted by β_a^i, and a global lower bound for the probability of the most likely final solution, denoted by $G = \max_i p_{1:N}^i$. We can then prune out all solution candidates i in set A_a for which

$$p_a^i + \beta_a^i \leq G/\Delta,$$

where Δ is a threshold chosen sufficiently large; we use $\Delta = \exp(12)$ in our experiment, to indicate that we only prune out solutions that are guaranteed to be at least five orders of magnitude less likely than the optimal one. The bound β_a^i is obtained by inspecting the other sets A_m. Since we store with each set the total probability of all possible allocations as q_m, we obtain the bound $\beta_a^i = \sum_{m \neq a} q_m$ that is independent of the solution candidate itself. Note that the quantity G is in general unknown but we can use any lower bound for that instead and retain the validity (at the expense of longer computation time); we will return to the choice of G in Section 3.2.6.

The constraints used for CHECKCONSTRAINTS are not the original constraints, but instead we need to here look at what the remaining feasible solutions for this set are. For each set we can compute the minimum and maxi-

mum sizes of the clusters, denoted by $r_a^l = \min_i r_a^i$ and $r_a^u = \max_i r_a^i$, where the minimum and maximum are taken separately for each dimension. Then the constraints for the set A_a are given by

$$U_a = U - \sum_{m \neq a} r_m^l,$$

$$L_a = \max\left(L - \sum_{m \neq a} r_m^u, 0\right).$$

In other words, we can subtract from the global bound all the counts that we know for sure will be allocated in some future set, and we need to allocate in this set the counts that cannot be anymore allocated in the future sets.

3.2.5 Backward pass

The actual sample is produced by traversing the partial solution sets backwards, starting from the final set $A_{1:N}$. We normalize the probabilities of the possible solutions in that set and randomly sample one of those, denoting it by $r_{1:N}^l$. We then find all possible combinations of the last joined set m and its counterpart A_{-m}, that stores the solutions for all other samples, for which

$$r_m^i + r_{-m}^j = r_{1:N}^l.$$

That is, we find the possible solutions in each set that could have been paired up to create the final solution. For each of these we compute the probability $p_m^i \times p_{-m}^j$ and draw a categorical sample to indicate the cluster assignment for sample m. We then proceed backwards in the table repeating this same procedure, now using r_{-m}^j for the chosen allocation as the target vector, until at the very end we simply pick the only possible allocation for the first sample. This is guaranteed to produce a valid sample, the only error source coming from partial solution sets pruned away because of the upper bound being smaller than G/Δ.

3.2.6 Implementation Remarks

The efficiency of the algorithm depends on the order of the samples being merged into the final set. We use a simple heuristic that attempts to keep the size of the intermediate sets minimal, which proved efficient in our preliminary experiments: we keep track of the expected cluster size vector (obtained by summing simple matrix products for all remaining sets), and always join the sample that is expected to violate the maximum constraints most. If no such samples exist, we merge with the set having the smallest number of remaining solutions.

Another key element is keeping the total probabilities q_a associated with the unprocessed sets updated. Right after the initialization, we can typically exclude considerable number of solution candidates in the singleton sets because already that single assignment would make the full solution too unlikely. Since the bounding is based on the sum over all other sets, including the one that has already accumulated more samples, we should apply BOUND and CHECK-CONSTRAINTS again for all sets after each join.

Finally, we stated earlier that for bounding the candidates we need to know the probability G of the best final solution, so that we can prune out solutions for which the bounded probability is sufficiently smaller than that. Since we are operating in a depth-first manner we do not obtain such a bound with the algorithm itself. Instead, we find a lower bound for it by the following procedure: we first find the most likely individual solution by solving (3) and compute the cluster sizes $\hat{r}$ of that solution. We then solve the forward pass with constraints $L_k = U_k = \hat{r}_k$ and global bound G_0 corresponding to the total probability of all possible assignments without any constraints. This either produces a lower bound for the probability of the solution candidate corresponding to cluster sizes $\hat{r}$ or, if G_0 is too large, fails by producing an empty set. If the process failed we repeat the procedure using smaller G_0, until a valid lower bound is obtained. While this procedure is somewhat inefficient it generally still takes only a fraction of the total computation time, especially for hard instances. In our experiments, the resulting lower bound for G was also typically very close to the actual true maximum probability (seen after running the full forward pass).

Finally, for well-separated clusters it is typically not necessary to solve the whole problem in one go. Instead, we can partition the data set into disjoint subsets of data points that do not compete for the same clusters, using a simple greedy procedure. Then we can apply the algorithm for each subset separately, while still guaranteeing to produce an independent sample. In the empirical experiments we skip this step to keep the results as clear as possible (how often the problem splits into disjoint sets depends heavily on the data), but in practice it should be done since finding the disjoint sets is very light operation.

3.3 REJECTION SAMPLER

A considerably simpler algorithm for solving the same problem can be obtained by a rejection principle. Despite the simplicity, the rejection sampler presented next will still be a practical solution for some problem instances.

Given N samples to be allocated to K clusters, the rejection sampler simply allocates all samples independently, drawing the assignment for each from the normalized likelihoods $p(z_n = k) = \frac{g(x_n, z_k)}{Z}$, where Z sums over the probabilities for the different clusters k. Afterwards, the sampler checks whether the constraints on the cluster sizes are violated. If there are no violations we keep the sample. Otherwise we create another sample and check for the constraints again, continuing until a valid sample is produced.

This sampler is obviously inefficient for cases where the constraints rule out the most likely solutions, but for cases with loose constraints it is a practical tool. Often the very first sample will be accepted and the sampler is so fast that we can typically afford to re-sample quite many times.

4 EVALUATING THE ASSIGNMENT SAMPLERS

In the following we demonstrate the samplers on artificial problems. At this stage we do not consider the sampler as part of a full clustering model, but instead merely look at the process of sampling the cluster assignments given some log-probabilities for the individual assignments. In other words, we simply consider the constrained optimization task of finding all possible solutions to the maximization problem (3) that exceed a certain threshold and drawing a sample from that set.

4.1 PROBLEM INSTANCES

The difficulty of a problem instance can be described crudely along two axes: the optimality gap indicating how close the best individual solution is to the unconstrained optimal allocation, and how quickly the probabilities decay when forced to pick sub-optimal allocations. The former is tightly connected with the tightness of the constraints, whereas the latter related to the tightness and separation of the clusters.

Intuitively, the instances with small (or zero) optimality gap are good for the rejection sampler: Almost all samples produced are within the constraints and hence the sampler is almost as efficient as an unconstrained sampler would be. For the dynamic programming algorithm these instances are the worst possible ones, especially if the probabilities decay slowly; we need to enumerate an excessively large set of solutions. The other extreme of problems with tight constraints and large optimality gap is difficult for the rejection sampler, but easy for the dynamic programming variant as long as the probabilities decay quickly enough.

To study the behavior of the samplers under these characteristics we create random problem instances by sampling the log-probabilities from standard distributions and by controlling the tightness of the constraints. We do this instead of considering actual cluster assignment setups since it allows finer control over the characteristics; for real clustering instances the optimality gap and rate of decay are often correlated in a complicated manner. We return to actual clustering problems in Section 6.

4.2 RESULTS

We created random solution instances with $K \in [6, 15]$ and $N \in [36, 225]$, drawing the entries from normal distribution with zero mean and standard deviation $\sigma \in [4, 40]$. We then solved the problems with varying degree of constraints, so that each cluster size was allowed to differ from the mean by $[0, 3]$ samples. For each problem, we ran both of the above algorithms and stored the running time until a valid solution was found, terminating the samplers if it took more than 20 seconds.

We summarize the results in Figure 2, where we present the computation times as a function of the problem instance characteristics discussed above. The optimality gap is determined by simply comparing the solution of (3) to the unconstrained optimum, and for measuring the probability decay we use a simple proxy: we count for each sample the number of cluster assignments that have probability above $1/\Delta$ of the highest probability, excluding the top candidate itself, and sum them up. This approximates the number of free variables to be considered outside the best allocation, but need not correlate with the original problem size. For producing these plots we always grouped all problem instances satisfying specific conditions into one pool, reporting the quantiles of the computation time for that pool. For studying the effect of the optimality gap we only used instances for which the number of free variables is below 50, and for studying the effect of the free variables we considered cases with logarithmic optimality gap below 7.

The experiment confirms the intuitive expectations of the previous section: for small optimality gap the rejection sampler is optimal but it quickly becomes infeasible when the gap grows. The rejection sampler, meanwhile, is efficient for fast enough probability decay (or, equivalently, small enough effective problem size), but becomes extremely slow if the probabilities do not decay quickly enough. A notable observation is that the running time curves of both algorithms have a sharp curve with respect to one of the measures: The running time is reasonably constant and always manageable until some threshold in gap or probability decay, and after that the running time becomes quickly excessive. In other words, the instances with large optimality gap and small decay of probabilities are problematic for both samplers.

5 HYBRID SAMPLER

In light of the above experiments, we propose as the final sampling solution a hybrid algorithm that uses both the rejection sampler and the dynamic programming algorithm while avoiding the cases that are too slow for both of them.

For a given task of assigning N samples, we first try out with the rejection sampler for some number of tries; if we produce a valid sample we keep it and the process terminates. If we fail to produce a valid sample we proceed to evaluate the difficulty of the problem. If the problem is considered easy enough, we apply the dynamic programming algorithm to produce the sample.

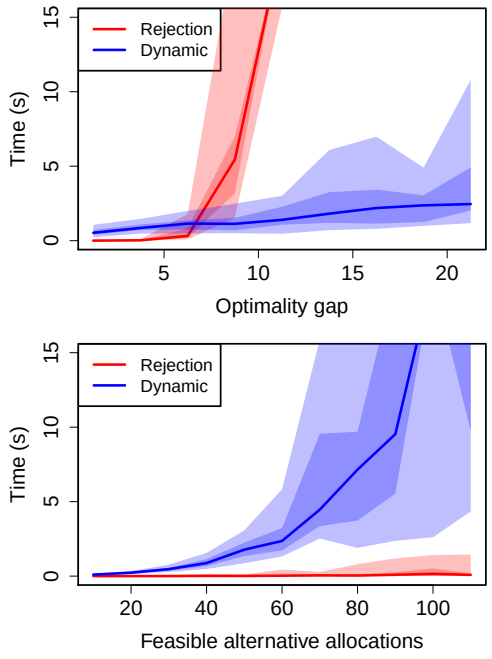

Figure 2: Comparison of running times of the two alternative sampling algorithms in producing a sample from posterior distributions $p(\{z_n\}|\{x_n\}, \phi)$ of varying difficulty. For both figures the dark shaded area indicates times between the 35% and 65% quantiles and the light shaded area between the 20% and 80% quantiles, and the times refer to single-core implementations. **Top:** The rejection sampler breaks down for logarithmic optimality gaps of roughly 7, meaning that the optimal allocation is roughly 1000 times more likely than the best feasible solution, but before that has effectively constant running time. The dynamic programming algorithm is largely insensitive to the optimality gap, but slightly slows down for the harder problems and occasionally takes longer time. **Bottom:** The dynamic programming sampler breaks down when the probabilities decay off too slowly. Here the horizontal axis denotes the count of non-optimal sample allocations with log-probabilities within 6 points of the most likely allocation for that sampler, and we see the algorithm becomes excessively slow around count 60.

If the problem is considered too challenging for the dynamic programming algorithm we split the problem into smaller chunks. We randomly divide the data point into two sets of $N/2$ instances each, and draw the assignments for each half conditional on the current assignments for the other half. For each half we again first try the rejection sampler and then consider the dynamic programming sampler or proceed to further subdivide the problem recursively. Assigning only a subset of the data points at a time naturally introduces auto-correlation in the overall sampling chain, but the time saved in not attempting to solve an overly difficult instance at once allows repeating the process enough times to produce an independent sample.

The exact criteria for when to split the problem into two halves should depend on the cost of sampling the cluster parameters ϕ given the assignments. For models where this stage is efficient, like our Gaussian likelihoods, the increased auto-correlation in sampling the assignments is not an issue and we can sample fairly small sets at once. In the other extreme, such as mixture models where approximative Bayesian computation [Csillery et al., 2010] is needed for sampling the cluster parameters, it pays off to solve the whole problem at once even if it takes a long time.

A full-blown analysis stage should inspect the rate of decline for the probabilities, the optimality gap, the number of solutions within the constraints, and possibly other statistics. In practice, however, we resort to a simpler heuristic to avoid the computation needed for the analysis (finding the optimality gap requires solving (3)); we simply use the same measure of effective problem size used in Section 4.2 and use the dynamic programming algorithm for cases where this number is small enough.

6 EVALUATING THE MICROCLUSTERING MODEL

Next we evaluate the final hybrid algorithm as part of a whole microclustering model to illustrate the balance between the two alternative solutions. As explained above, the algorithm has two parameters: The number of times the rejection sampler is tried before giving up, and the maximum complexity of the problem instance solved with the dynamic programming solver. Both parameters control how many samples can be jointly allocated; bigger values make it more likely that the sampler can allocate large number of data points at once, but at the same time increases the computational time per posterior sample. The relative ratio of these two parameters, in turn, controls how often each of the algorithms is in practice used; high number of tries and small complexity threshold imply that the rejection sampler allocates most samples, and vice versa.

Figure 3 illustrates the effect of the two parameters for an example clustering problem where the input data is uniformly distributed in a two-dimensional rectangle; the data has no natural cluster structure and hence the constraints are crucial in guaranteeing balanced cluster sizes. We show results for both 64 data points being clustered into $K = 8$ clusters and 256 data points being clustered into $K = 16$ clusters, constraining the clusters to be exactly identical in size. For both cases the results are similar: The rejection sampler takes care of the assignments of majority of the samples unless the maximum number of trials is very low, but using the dynamic sampler to solve harder problem instances helps assign more data points at a time. Both indi-

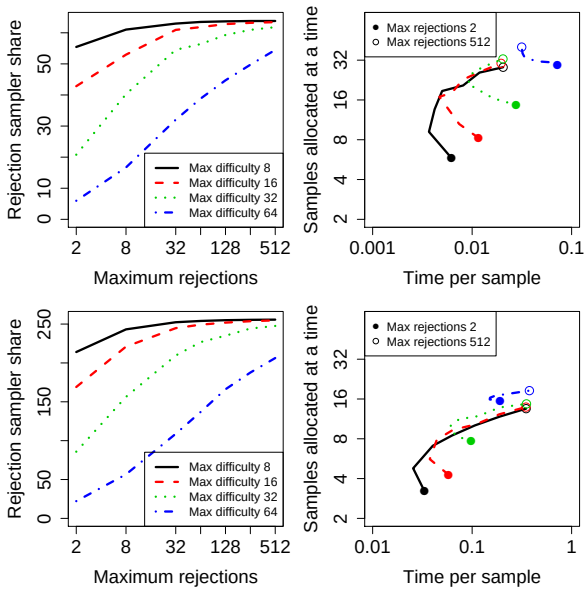

Figure 3: Illustration of the hybrid algorithm on two clustering problems. The top row corresponds to clustering $N = 64$ data points into $K = 8$ clusters and bottom row to clustering $N = 256$ samples into $K = 16$ clusters. For both cases the data points are uniformly distributed on a two-dimensional space, the clusters are forced to be of equal size, and the results are averaged over five randomly created data sets. **The left plots** show how many data points are on average assigned by the rejection sampler; the ratio naturally goes up if we try several times before giving up and down if we solve harder problems using the dynamic programming algorithm. The colored lines corresponding to different maximum difficulty scores, corresponding to the number of free variables in the problem instance as defined in Section 4.2. **The right plots** characterize the overall efficiency of the algorithm, plotting the average number of data points assigned jointly (by either algorithm) versus the computational time required for producing the whole posterior sample. Points closer to the top left corner are here the best, quickly allocating several data points at once, and we see that comparable results are obtained with several combinations for the two parameters as long as the extreme values are avoided. The color-codes in the right plot match the left one; for example, the red line shows how the behavior of the sampler evolves for maximum problem difficulty of 16 when the number of allowed rejections grows from 2 to 512.

vidual algorithms are hence useful for the overall solution. Importantly, a wide range of parameter values gives satisfactory results, suggesting that the overall algorithm is not very sensitive to the choice of the thresholds.

7 DISCUSSION

In this work we introduced a new microclustering model [Miller et al., 2015] for solving clustering tasks with predefined constraints on the sizes of the clusters, motivated by scenarios where the clusters are used, for example, in creating teams of fixed sizes [Kim et al., 2015] or for allocating items for further processing of (manual or automated) workers with limited capacity. To control the sizes we introduced a cluster assignment prior that does not factorize over the samples but instead over the clusters; this formulation is more general and can be used also for models without hard constraints. One straightforward extension would consider priors where the log-probability of the cluster decays linearly when moving away from some preferred size; for such a prior we can still find the most likely assignment easily and hence can generalize the whole sampler.

The model requires the cluster assignments to be drawn jointly for all data points, which increases the computational cost compared to standard mixture models. We discussed two alternative samplers and showed that each is efficient for a subclass of problems, and then proceeded to present a practical algorithm that can draw samples also for large data collections with the possible expense of increased auto-correlation for overly complex problem instances. The algorithm attempts to assign all samples jointly, but in case the problem instance is too difficult it recursively splits the problem into two parts and assigns the samples conditional on the assignments for the other part. Probabilistic treatment of clustering is most useful for fairly small cluster sizes that necessitate explicitly treating the full posterior, and we showed that for such setups we can draw posterior samples in a fraction of a second.

Finally, we want to encourage Bayesian practitioners to consider combinatorial constraints in their models. Even though the problem of finding all solutions that exceed a certain threshold is considerably harder than finding the best one, it is still feasible for problems of moderate size. Sampling-based probabilistic inference also comes with natural solution for splitting the problem into easier and smaller sub-problems, in form of conditioning based on a subset of the assignments. Consequently, we believe that several types of combinatorial constraints can be incorporated into typical latent-variable models and other probabilistic models with fairly low additional overhead. The auto-correlation of the sampling chain increases and the individual sampler steps typically take longer time, but one should not shy away from introducing the constraints if they are important for the model itself.

Acknowledgements

The work was supported by the Academy of Finland, via the Finnish Center of Excellence in Computational Inference Research (COIN) and grant 266969.

References

Charles E. Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics*, 2(6):1152–1174, 1974.

Arindam Banerjee. Scalable clustering algorithms with balancing constraints. *Data mining and knowledge discovery*, 13(3):365–395, 2006.

Yuguo Chen, Persi Diaconis, Susan P. Holmes, and Jun S. Liu. Sequential Monte Carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100(469):109–120, 2005.

Katalin Csillery, Michael G.B. Blum, Oscar E. Gaggiotti, and Olivier Francois. Approximative Bayesian computation (ABC) in practice. *Trends in Ecology & Evolution*, 25(7):410–418, 2010.

Adrian Dobra, Claudia Tebaldi, and Mike West. Data augmentation in multi-way contingency tables with fixed marginal totals. *Journal of Statistical Planning and Inference*, 136(2):355–372, 2006.

Avinava Dubey, Sinead A. Williamson, and Eric P. Xing. Parallel Markov chain Monte Carlo for Pitman-Yor mixture models. In *Proceedings of Uncertainty in Artificial Intelligence*, 2014.

Byoung Wook Kim, Ja Mee Kim, Won Gyu Lee, and Jin Gon Shon. Parallel balanced team formation clustering based on MapReduce. In *Advances in Computer Science and Ubiquitous Computing*, volume 373 of *Lecture Notes in Electrical Engineering*, pages 671–675. Springer, 2015.

Arto Klami. Variational Bayesian matching. In *Proceedings of 4th Asian Conference on Machine Learning*, volume 25 of *JMLR: W&CP*, pages 205–220, 2012.

Arto Klami. Bayesian object matching. *Machine learning*, 92(2):225–250, 2013.

Geoff McLachlan and David Peel. *Finite mixture models*. Wiley, 2004.

Jeffrey Miller and Matthew Harrison. A simple example of Dirichlet process mixture inconsistency for the number of components. In *Advances in Neural Information Processing Systems 26*, pages 199–206, 2013.

Jeffrey Miller, Brenda Betancourt, Abbas Zaidi, Hanna Wallach, and Rebecca C. Steorts. Microclustering: When the cluster sizes grow sublinearly with the size of the data set. *arXiv:1512.00792*, 2015.

Dan Roth. On the hardness of approximative reasoning. *Artificial Intelligence*, 82(1–2):273–302, 1996.

Steven L. Scott. Bayesian methods for hidden Markov models. *Journal of the Americal Statistical Association*, 97(457):337–351, 2002.

Hanna M. Wallach, Shane T. Jensen, Lee Dicker, and Katherine A. Heller. An alternative prior process for nonparametric Bayesian clustering. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *JMLR: W&CP*, 2010.

Liangliang Wang, Alexander Bouchard-Cote, and Arnaud Doucet. Bayesian phylogenetic inference using a combinatorial sequential Monte Carlo method. *Journal of the American Statistical Association*, 110(512):1362–1374, 2015.

Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8): 883–889, 2010.