Scalable Inference on Kingman's Coalescent using Pair Similarity

Dilan Görür, Levi Boyles and **Max Welling**
Bren School of Information and Computer Science
University of California, Irvine
Irvine, CA, USA
{dgorur,lboyles,max.welling}@uci.edu

Abstract

We present a scalable sequential Monte Carlo algorithm and its greedy counterpart for models based on Kingman's coalescent. We utilize fast nearest neighbor algorithms to limit expensive computations to only a subset of data point pairs. For a dataset size of n, the resulting algorithm has $O(n \log n)$ computational complexity. We empirically verify that we achieve a large speedup in computation. When the gain in speed is used for increasing the number of particles, we can often obtain significantly better samples than those of previous algorithms. We apply our algorithm for learning visual taxonomies of birds on 6033 examples, a dataset size for which previous algorithms fail to be feasible.

1 Introduction

Nonparametric Bayesian (NPB) models provide an elegant solution to the problem of inferring structure from data. In particular, as the dataset grows larger, the model adapts its complexity as it discovers finer structure in data. This behavior is reminiscent of humans learning increasingly sophisticated abstract concepts as more of the world is observed. These concepts are often organized hierarchically. For instance, we intuitively understand that there is a hierarchical organization in the sequence of categories: animal→mammal→dogs→terrier→"Max". Clearly, more concepts will form as we perceive more information.

Kingman's coalescent [1] is a NPB prior over binary trees of unbounded size. It was originally developed in population genetics to infer characteristics about a population and it also provides an elegant way to learn a hierarchical clustering of data [2]. As such it is ideally suited to model the growth of concept hierarchies of the type discussed above. Both in population genetics and machine learning applications, we would like to model at least thousands or tens of thousands data points. However the fastest sampling algorithms for the coalescent [2, 3, 4] can currently handle no more than a meager few hundred data-items. It is thus apparent that there is an enormous gap between the scale we would like to apply the coalescent model and what we can achieve in practice. The main goal of this paper is to make a dent in this performance gap by proposing a fast inference algorithm for Kingman's coalescent that can handle thousands of data-items given limited computation time and maintain a similar level of accuracy as alternative methods. We showcase our approach on a challenging problem of learning visual taxonomies of birds using 6033 data points in Figure 1, obtained from a 20-particle run that took 116 hours of CPU time. It is important to note that the alternative algorithms would have taken on the order of months!

There have been many approximate inference techniques developed for coalescent models based on Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC) as well as greedy algorithms. Typically, the agglomerative algorithms start with each data item in its own cluster, and iteratively merge pairs of clusters until a single cluster remains. For the greedy and SMC algorithms in [2] (except `GreedyRate1`), each of the $n-1$ iterations involves costly computations for *each* pair, resulting in $O(n^3)$ computational cost. `GreedyRate1` and `SMC1` [3] follow a similar framework, but use an approach that allows the *reuse* of some computations, leading to a reduced computational complexity of $O(n^2)$. In this paper, we take an alternative approach that *avoids* doing most of the costly computations by using dedicated data-structures (e.g. kd-trees, cover trees) for fast nearest neighbor (NN) search and only performing costly com-

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

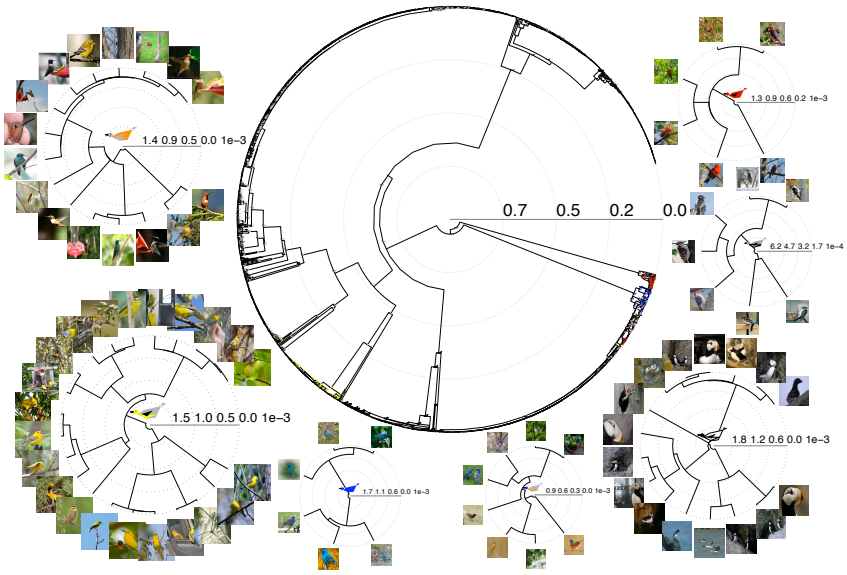

Figure 1: A sample tree from our SMC algorithm run on the Birds200 dataset (**center**), and several subtrees. Bird caricatures are displayed on the nodes of the tree that summarize the color properties of the descendants of that node. (Zoom in for better visibility.) Five out of 15 features are used in the caricatures for interpretability. Light gray indicates "uncertain," and all other colors denote the color of the bird's body, belly, tail, wing, crown, and eyes. The subtrees show the images corresponding to the data at the leaves along with the corresponding common ancestor caricature in the middle.

putations for the closest pairs of points. This results in a scalable algorithm with computational complexity $O(n\log(n))$ assuming the fast NN algorithm maintains its speed as iterations proceed.

In the next section, we set the ground by reviewing Kingman's coalescent and the basis of the SMC algorithms of [2, 3] for inference on this model. The proposed fast SMC inference algorithm and its greedy version are described in Section 3. We compare our method to alternatives as well as demonstrating runs on larger scale datasets which the other inference algorithms cannot cope with in Section 4. We conclude with a discussion in Section 5.

2 Kingman's Coalescent

Kingman's coalescent is a prominent model in population genetics, defining a distribution over genealogical trees of a haploid population [1]. It describes the genealogy of individuals in the current population by going backwards in time, and coalescing pairs of individuals to form the ancestry. In the following, we borrow description and notation from [2] and refer the reader to [2] for details.

Let π be the genealogy of n individuals observed at the present time $t_0 = 0$. There are $n-1$ coalescent events in π, ordered such that $i = 1$ is the most recent one, and $i = n-1$ is the last event when all ancestral lines are coalesced. Let (ρ_{li}, ρ_{ri}) be the ith pair of lineages to coalesce and $\delta_i = t_{i-1} - t_i > 0$ be the duration between adjacent events. Denote with A_0 the initial set of individuals at $t_0 = 0$ and A_i the set of lineages right after coalescent event i. Similarly, denote the set of *pairs* of lineages with $\psi_i = \{(\rho_h, \rho_k) | \rho_h \in A_i, \rho_k \in A_i, h \neq k\}$. Under Kingman's coalescent, every pair of lineages merges independently with exponential rate 1 and lineages are exchangeable. Thus, the ith coalescent event happens after time $\delta_i \sim \text{Exp}\left(\binom{n-i+1}{2}\right)$ and independently the pair (ρ_{li}, ρ_{ri}) is chosen from ψ_{i-1} to form $\rho_i = \rho_{li} \cup \rho_{ri}$. With probability one a random draw from Kingman's coalescent is a binary tree with a single root at $t \to -\infty$ and the n individuals at time $t = 0$. Combining the exponential probabilities of the durations and the uniform choices of lineages we have $p(\pi) = \prod_{i=1}^{n-1} \exp\left(-\binom{n-i+1}{2}\delta_i\right)$.

In addition to its established use in population genetics models [5, 6, 7], Kingman's coalescent has been used for hierarchical clustering [2, 3, 8]. In general, it can be used as a prior over binary trees in a model where we have a tree-structured likelihood function for observations at the leaves. In particular, the data generating process $p(X|\pi)$ is defined as a simple Markov process on the tree which evolves forward in time. Data points at the leaves are generated by applying

the Markov process to transform the root node along the tree branches. Using terminology from population genetics, the latent tree topology together with the times of the mutation and coalescent events and the states of the internal nodes fully describe the genealogy. It is possible to integrate over the mutations and compute the probability of data under a particular tree structure sequentially by using message passing from the leaves towards the root of the tree, as discussed in [2]. Thus, the likelihood can be summarized as a product over functions that depend only on the local messages sent from the coalescing individuals ρ_{l_i} and ρ_{r_i} to their parent ρ_i and the time between consecutive coalescent events δ_i:

$$p(X|\pi) = Z_0(X) \prod_{i=1}^{n-1} Z_{\rho_i}(X, \theta_{i-1}, \delta_i, \rho_{l_i}, \rho_{r_i}), \quad (1)$$

where $\theta_i = \{\delta_j, \rho_{l_j}, \rho_{r_j} \text{ for } j \leq i\}$ denotes the first i coalescent events, $Z_0(X)$ is a normalization constant and $Z_{\rho_i}(X, \theta_i) = Z_{\rho_i}(X, \theta_{i-1}, \delta_i, \rho_{li}, \rho_{ri})$ has the form:

$$Z_{\rho_i}(X, \theta_i) = \int p_0(y_i) \prod_{c=l_i,r_i} \int p(y_c|y_i, \theta_i) M_{\rho_c}(y_c)\, dy_c\, dy_i. \quad (2)$$

In the above equation M_{ρ_c} is the message from child ρ_c to ρ_i, $p(y_c|y_i, \theta_i)$ represents the Markov process from ρ_i to ρ_c and $p_0(y_i)$ is the equilibrium distribution of the Markov process at $t \to -\infty$. The posterior is proportional to the product of the prior and the likelihood:

$$p(\pi \mid X) \propto \prod_{i=1}^{n-1} \exp\left(-\tbinom{n-i+1}{2}\delta_i\right) Z_{\rho_i}(X, \theta_i). \quad (3)$$

2.1 SMC inference on the Coalescent

The SMC algorithms that have been developed for inference on the coalescent can be divided into two classes according to the state space they choose to represent; algorithms that represent the coalescent events and the mutation events, without representing the time of the events [4, 9] and algorithms that represent the coalescent events and the time of coalescence, but do not represent the mutation events [2, 3, 10]. Specifically, the algorithm we propose in the next section is closest to `PostPost` [2] however the idea is generally applicable to the algorithms that use the latter class of representation.

In detail, the state space consists of the lineages, coalescent events between pairs of lineages and the time of these events. We start with n individuals at the leaves of the tree, and pick the pair of lineages to coalesce and their time for coalescence at each iteration. After a pair is merged into a new ancestor $\rho_i = \rho_{li} \cup \rho_{ri}$ the new set of *individuals* A_i is obtained by removing ρ_{li} and ρ_{ri} from A_{i-1} and including ρ_i. Similarly, the set of *pairs* is updated to ψ_i by removing all pairs with ρ_{li} and ρ_{ri} from ψ_{i-1} and adding in pairs with ρ_i. The algorithm terminates after $n-1$ iterations, when all individuals merge into a single common ancestor. Several particles are run in parallel, updating weights after each iteration, and are resampled when weights diverge.

Noting that the posterior distribution in eq.(3) is a product of local terms defined at a current coalescence level and conditioned on the previous, the ith product can be seen as a "local posterior" and it can be used as a proposal distribution at ith iteration,

$$q(\rho_i, \delta_i) \propto \exp\left(-\tbinom{n-i+1}{2}\delta_i\right) Z_{\rho_i}(X, \theta_i). \quad (4)$$

In particular for `PostPost`, sampling from $q(\rho_i, \delta_i)$ is performed by first sampling a pair (ρ_{li}, ρ_{ri}) fom ψ_{i-1} using the probabilities

$$q(\rho^{lr}) \propto \int \exp\left(-\tbinom{n-i+1}{2}\delta\right) \times Z_{\rho_i}(X, \theta_{i-1}, \delta, \rho_{li}, \rho_{ri}) \mathrm{d}\delta, \quad (5)$$

and then sampling the coalescence time δ_i given $\rho_i = \rho_{li} \cup \rho_{ri}$:

$$q(\delta_i|\rho_i) \propto \exp\left(-\tbinom{n-i+1}{2}\delta_i\right) Z_{\rho_i}(X, \theta_i). \quad (6)$$

Note that since the distributions eq.(4)-(6) depend on the iteration number i, a proposal distribution should be reconstructed at every iteration for $O(n^2)$ pairs at that iteration. `SMC1` on the other hand makes use of the memoryless property of the exponential distribution and keeps the rate of the prior exponential distribution fixed to 1:

$$q(\delta_i|\rho_i) \propto \exp\left(-\delta_i\right) Z_{\rho_i}(X, \theta_i). \quad (7)$$

This allows using the same proposal distribution for a particular pair over iterations. `SMC1` samples a coalescence time for each pair in ψ_{i-1} from eq.(7) only once, to be used for all iterations. The coalescing pair is chosen to be the one with the shortest sampled time. At each iteration, $O(n)$ computations is necessary for constructing the proposal distribution for the pairings of the new ancestor, resulting in $O(n^2)$ overall cost.

3 Speeding up Inference using Similarity Information

We propose a novel SMC algorithm that avoids doing costly computations for most pairs by making use of the similarity information between the pairs. Note that the likelihood is defined in terms of a Markov mutation process that acts on the branches of the tree.

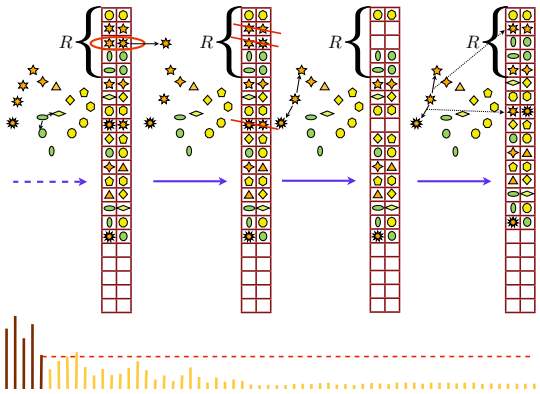

Figure 2: A priority queue of closest pairs is maintained. At each iteration a pair merges and a new point is introduced. The priority queue is updated by removing the pairings of either of the merging points and including the kNN of the new point. The proposal probability of the first R pairs (dark bars) is computed using eq.(5) and the probability of the remaining pairs is set to a constant value. This value is set to the value of the Rth pair for `SMCnn` and zero for `GreedyNN`.

Going backwards in time, nodes that are far away from each other in the state space of mutations are very unlikely to coalesce before closer (or more similar) ones. We can use this observation to gain a speed up in inference and construct a proposal distribution by focusing attention on a small subset of ψ_{i-1} that are more likely to coalesce rather than computing the proposal probability eq. (5) for all pairs in ψ_{i-1}. This results in a great gain in computation time relative to both `PostPost` and `SMC1` as the costly computations for constructing the proposal distribution is limited to a constant number of pairs.

A distance metric in the space of mutations would depend on the Markov transition kernel and may be costly to evaluate. Furthermore, note that the representation we use does not include the state of the internal nodes, but the nodes are represented in terms of messages. The probability eq. (5) of coalescing a pair (ρ_l, ρ_r) at iteration i depends on the messages M_{ρ_l} and M_{ρ_r}, eq. (1). Intuitively, the more similar the messages are, the more likely a pair is to coalesce. We aim to efficiently find the R pairs in ψ_{i-1} whose messages are nearest in some (e.g. Euclidian) sense. This notion of nearness will not necessarily give the pairs that are most likely to coalesce, but the idea is that it will give some proxy to that. Note that we use this approach merely to construct a new proposal distribution. An SMC algorithm using this proposal will still be a correct sampler as long as the new proposal distribution has support over the true posterior density. We ensure this by assigning positive proposal probability to the rest of the pairs.

Algorithm 1 `SMCnn`

1: a. Form a priority queue of sorted pairs of nodes ρ_{lr} according to distance $d(\rho_l, \rho_r)$
   b. For $\rho_{lr} \in \phi_{i-1}^{1:R}$: approximate the "local likelihood" $Z_{\rho_{lr}}$ by $\tilde{Z}_{\rho_{lr}}$,
   c. For $\rho_{lr} \in \psi_{i-1} \setminus \phi_{i-1}^{1:R}$, use $\tilde{Z}_{\psi_{i-1}^R}$
2: a. Compute the probability of merging each pair in $\phi_{i-1}^{1:R}$, eq. (5),
   b. Set the probability of the rest of the pairs to the Rth pair's probability.
3: Sample a pair from the normalized probabilities,
4: Sample a coalescent time for the selected pair using eq.(6),
5: a. Merge pair into a new node, remove merging nodes from representation, include new node.
   b. Remove merging nodes from NN representation, add new node in the representation.
   c. Search for kNN of the new parent among A_i.
   d. Place the new pairs in the sorted list of pairs.
6: Update particle weights using eq. (8), and
7: Resample particles when weights diverge.

3.1 `SMCnn`

We maintain a priority queue $\phi_{i-1} \subset \psi_{i-1}$ of pairs in order to quickly retrieve the closest pairs at each iteration, as pictorially explained in Figure 2. Let ϕ_{i-1}^l denote the lth element in the priority queue. Given a distance metric $d(\cdot, \cdot)$, the pairs are sorted in increasing distance such that $d(\phi_{i-1}^h) \leq d(\phi_{i-1}^l)$ for $h < l$. For the first R pairs $\phi_{i-1}^{1:R}$ we compute eq. (5) to find the proposal probability of coalescing. In order to retain a correct sampler, coalescing one of the remaining pairs in ψ_{i-1} should be given nonzero probability. We use the computed proposal distribution of the Rth pair in the queue to approximate these *distant* pairs. That is, we set the proposal probability of these pairs to be equal to the proposal probability of the Rth pair, $q(l^*, r^*) = q(\phi_{i-1}^R), \forall(\rho_{l^*}, \rho_{r^*}) \in \psi_{i-1} \setminus \phi_{i-1}^{1:R}$. Thus, if one of these distant pairs are chosen to coalesce, we use the Rth pair's coalescence time distribution $q(\delta_i \mid \phi_{i-1}^R)$ to propose a time for the chosen pair.

After each coalescent event, the set of individuals and the set of pairs are updated by adding and removing pairs as in Section 2.1. The priority queue should also be updated accordingly. To save on computational cost, we should choose $R << \binom{n-i+1}{2}$.[1] Therefore, not all new pairs need to be included in the priority queue. For time efficiency, only the nearest neighbors of the new ancestor node are retrieved, and pairings of the new node with these neighbors are included in ϕ_i according to the distance between the individuals mak-

[1] `SMCnn` proposal distribution is equivalent to that of `PostPost` for $R \geq \binom{n-i+1}{2}$.

ing up the pairs. The number of neighbors to retrieve, k, is a parameter of the algorithm, that is interrelated to R.

We make use of efficient NN structures to find the neighboring points for which it is possible to efficiently delete and add points as well as having an efficient query time. Any exact or approximate nearest neighbor data structure can be used for this purpose. In this paper we consider kd-trees [11] and covertrees [12] as examples for analyzing computational complexity.

The algorithm proceeds by sampling the coalescing pair and its coalescence time. In detail, after the probabilities of all pairs are determined, a coalescing pair and their coalescence time are chosen, and the points are merged. An important point to note is that the distant pairs are assigned the proposal distribution as the Rth pair in the priority queue. Therefore, if one of the distant pairs is chosen to be the coalescing pair, its coalescent time should be sampled using the local posterior of the Rth pair.

The SMC weights are updated taking into account the distant pairs sharing the same proposal distribution with the Rth pair

$$w_i = w_{i-1}\frac{Z_{\rho_i}(X,\theta_i)}{\tilde{Z}_{\rho_i}(X,\theta_i)} \times\Big(\big[\tbinom{n-i+1}{2} - R\big]q(\phi_{i-1}^R) + \sum_{h=1}^{R} q(\phi_{i-1}^h)\Big), \tag{8}$$

and the algorithm continues to the next iteration. The algorithm is summarized in Algorithm 1.

In addition to the number of particles M, `SMCnn` has two interrelated parameters; R is the number of pairs to do computations for at each iteration and k is the number of pairings of a new individual to include in the priority queue. `SMCnn` is a correct sampler regardless of the values of R and k, however the efficiency of the algorithm depends on them. The quality of individual samples will improve with increasing R, but the computational cost will also increase. Thus, R should be chosen to compromise between sample quality and speed. Typically, k can be an order of magnitude smaller than R as we only need to ensure there are enough pairs represented in the priority queue, i.e. $nk >> R$. Naturally, knowledge about the structure of the dataset such as the expected number of points in sub-clusters can be used to guide the choice of k.

On a given iteration of SMC, the proposed algorithm computes eq. (5) R times and performs a single NN search of size k. Assuming a kNN query time of $O(k\log n)$, this gives a running time of $O(cRn + kn\log n)$ per particle, where c is the cost of constructing a proposal distribution per pair. However, there may be additional terms depending on the specific NN search algorithm and the resampling frequency. For instance, kd-trees may need to be rebuilt in order to have efficient queries, and covertree operation times depend on the expansion constant of the data, which may be $O(n)$ in the worst case. Furthermore, due to resampling, tree structures may need to be copied (or rebuilt). A more precise expression for running time is $O(cRn + \alpha kn\log n + fr(n))$, where α depends on the query time for the specific NN algorithm, f is the number of iterations of resampling or tree balancing and $r(n)$ is the total cost of updating and rebuilding any NN data structure.[2] The primary gain of our running time is that the algorithm is linear in c, a very large constant. For small to moderate dataset sizes, Rc will be much larger than $k\log(n)$, resulting in the first term dominating the cost. The latter terms will start to dominate as n gets larger. For reference, the cost of different algorithms are provided in Table 1.

	construct proposal	**priorityQ/ search**	**resample/ rebuild**
`SMCnn`	$MRcn$	$Mkn\log n$	$Mfn\log n$
`GreedyNN`	Rcn	$kn\log n$	-
`SMC1`	Mcn^2	$Mn\log n$	Mfn^2
`GreedyR1`	cn^2	$n\log n$	-
`PostPost`	Mcn^3	-	Mfn

Table 1: Computational cost of different algorithms. c is the cost of constructing the proposal distribution for one pair, which is similar for all algorithms.

It is worth pointing out another computational gain our algorithm. Note that at each iteration there are $O(n^2)$ pairs, thus naïvely the pair probability distribution is a discrete distribution of size $O(n^2)$. However, since we are assigning the probability of all but R pairs to a particular value, we can represent this distribution with $R+1$ dimensions. If the last element (representing the pairs in $\psi_{i-1}\setminus\phi_{i-1}^{1:R}$) is sampled, we only need to uniformly randomly choose among these pairs. Thus, we only ever need to deal with R-dimensional distributions.

3.2 `GreedyNN`

When a single good tree suffices for the application, greedy methods can be preferred over sampling methods due to their lower computational cost. The computational cost of greedy methods are typically better than their SMC counterparts as they do not employ several particles. Therefore the cost of the greedy algo-

[2]For kd-trees, $\alpha = 1$ and $r(n) = O(n\log(n))$ the cost of rebuilding the NN structure. For covertrees, $\alpha = \gamma^{12}$ and $r(n) = O(\gamma^6\log(n) + n)$ if we were to copy $O(n)$ trees on every iteration we resample, where γ is the expansion constant of the data. In practice, the third term does not dominate because rebuilding or copying every iteration isn't necessary.

rithms will scale similarly to their single-particle SMC versions.

The `SMCnn` algorithm described above can be modified into a greedy algorithm by replacing the sampling steps with local optimization steps. In detail, rather than sampling a pair in step (3), we choose the pair with maximum probability from among the first R pairs in the priority queue. We assign the coalescence time for this pair to be the mean of the coalescence time distribution for that pair. Note that steps 1.c and 2.b in the `SMCnn` algorithm can be skipped in the greedy algorithm as the pairs in $\psi_{i-1} \setminus \phi_{i-1}^{1:R}$ will not be picked by the greedy algorithm.

3.3 Previous work for speeding up inference

Constructing proposal distributions of the form discussed in Section 2.1 is typically very costly since they involve approximating a nonstandard density function, taking its integral and sampling from it. The `PostPost` algorithm is especially expensive because it requires computing eq. (5) for every pair of points on every iteration, giving an $O(n^3)$ algorithm with a large constant. `SMC1` also requires computing similar integrals but in such a way that allows doing this only once for all iterations for each pair rather than for each pair at each iteration. This brings down the computational cost to $O(n^2)$ with a similarly large constant. [10]'s algorithm is equivalent to the `PriorPost` of [2] and it has a smaller constant for computing the proposal function. In detail, the order of sampling steps are changed such that at each iteration a coalescent time is sampled from the prior and the local posterior of each pair needs to be evaluated at the sampled time, rather than evaluating the integral for each pair. However since the times are sampled from the prior, the proposal is less accurate, making the algorithm less efficient. The algorithm still scales cubicly with the number of data points as the local posteriors need to be evaluated for each pair at every iteration. A common aspect of all three algorithms is that they do costly computations for each possible pair.

Slatkin [10] notes the concern about the running time of the algorithm increasing rapidly with n and proposes to limit the number of pairs to be considered to coalesce at each iteration (step 2), which he calls the *span* of the simulation. The span is set to be the pairs that are chosen at random, giving a choice of compromise between computational cost and sample accuracy. Choosing a small number of pairs speeds up the algorithm but may lead to poor samples due to the naïve way of deciding on the span. Using Slatkin's terminology, our algorithm makes use of the distance between pairs of individuals to decide on the span rather than choosing it randomly.

4 Experiments

The number of particles required for faithfully representing samples from a desired distribution depends highly on the proposal distribution used. Here, we analyze the empirical properties of the proposed algorithms to get a sense of its efficiency in terms of run time as well as the bias and variance of the samples compared to the alternatives. We compare `SMCnn` with `SMC1` [3] and `PostPost` [2] and `GreedyNN` with `GreedyRate1` [2] on several datasets. We did experiments on SpamBase [13], Mushroom [13] and Birds200 [14] datasets. Birds200 consists of *uncertain* Mechanical Turk (MT) annotations of bird images from 200 different categories, with varying degree of similarity between categories and we use uncertainty levels of binarized features [15]. SpamBase were binarized and Mushroom data has categorical values, therefore we use the multinomial likelihood for all datasets.

Data set size vs CPU time

We empirically verify that our algorithm shows significant speedups in terms of computation per particle compared to previous algorithms. For all datasets we analyzed, the theoretical computational complexity of the algorithms is evident in the empirical results, with only occasional deviations. Figure 3 shows the computational cost of several algorithms as a function of n on the the different datasets with the $O(kn \log n), O(n^2)$ and $O(n^3)$ fit to the empirical points. We observe that for small n, `PostPost` is fast but it becomes the slowest as the dataset size increases. `SMCnn` scales linearly with R. It will be slower than `PostPost` for R close n^2 due to the overhead of maintaining the priority queue. The gain in speed up for `SMCnn` gets more pronounced for large n. The plots depicted show runs with $R < n$. Even for datasets as small as 400 datapoints, `SMCnn` is over an order of magnitude faster per particle than `SMC1` and `PostPost`.

Change in performance with time

In [3] `SMC1` is shown to be more efficient than `PostPost` both in terms of computational cost and the quality of samples. As the proposal distribution of `SMCnn` will be equivalent to that of `PostPost` for $R \geq \binom{n-i+1}{2}$, we compare the performance of our algorithm to `SMC1`. We do not expect our algorithm to produce better individual samples than `SMC1`. However, the computational efficiency of `SMCnn` means we can entertain more particles for the given time, which typically leads to better performance. Figure 4 shows the change in approximated log evidence as the CPU time increases. Using a small number of particles, `SMCnn` finds relatively low evidence on all datasets. However, the performance quickly improves as the number of particles is increased. On the other hand, due to more

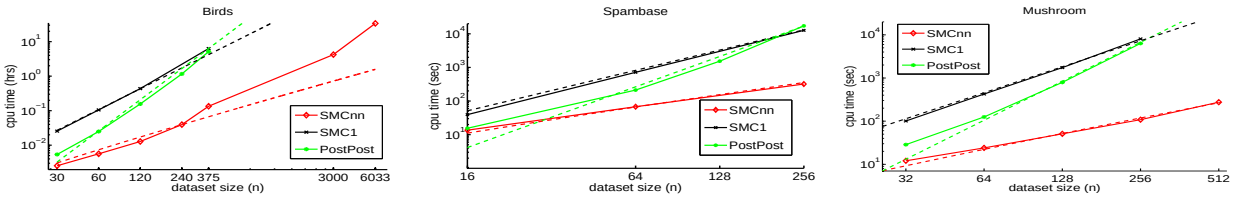

Figure 3: The run time of several algorithms as n increases on Birds (**left**), SpamBase (**center**) and Mushroom (**right**) datasets. Solid lines are linear interpolation of the actual running times, and dashed lines are theoretical running times. All algorithms were run with four particles, resampling when effective sample size falls below two. R was chosen to ensure same level of performance for `SMCnn` as the other algorithms. For the birds dataset this caused a deviation from the $n \log n$ running time (dashed red line) as we needed to increase R with n.

costly computations, `SMC1` can use only a small number of particles given the same computation time. The asymptotic performance of the two algorithms is comparable on the Birds and Mushroom datasets, with `SMCnn` being slightly better for Mushroom. Interestingly, `SMC1` fails to match the performance of `SMCnn` for the SpamBase dataset. We observed that for all datasets we considered, the approximated evidence did not change much for the range of larger M values we considered on a limited CPU time. However, as both algorithms are correct SMC samplers, their samples would converge in the limit of large number of particles. For these experiments, we used both L_1 and Euclidian distances to assess the sensitivity of the algorithm to the different distances used for retrieving the nearest neighbors. The results suggest that for the datasets we considered, both distances performed equally well.

Sample paths and effect of resampling

`SMCnn` can highly benefit from resampling. This is not always the case for `SMC1` as the partial weights are less informative of the quality of the samples. This is due to the fact that `SMC1` defers computing the contribution of a pair on the partial weights until it gets removed from the representation (either as a result of coalescing or losing one of the partners) to save computation time.[3] Sample paths shown in Figure 5 clearly demonstrate this. `SMC1` looks worse than most of the runs for `SMCnn` until the last few iterations on the birds data. On SpamBase and Mushroom datasets, `SMC1` starts promisingly, but fails to achieve the same level of performance as `SMCnn`. We tried runs of `SMC1` both with and without resampling and got very similar performance results.

Performance of `GreedyNN`

To evaluate `GreedyNN` we compare the joint probability of the data and the trees constructed by the different algorithms. We applied `GreedyNN` and `SMCnn` on all 6033 points of Birds200 with $R = 500, K = 25$. The average log joint probability of `SMCnn` with 20 particles was -2.19×10^5, and it took 116 hours. `GreedyNN` resulted in a tree with log joint probability of -1.79×10^5 in 3 hours. Thus, in this case, it seems `GreedyNN` is preferable and more particles will be necessary to cover the posterior space. A sample tree from `SMCnn` and some subtrees are depicted in Figure 1. We do not report results of other algorithms on this dataset size as they would take months of CPU time.

	$-\log P(X, \pi)$	**CPU time (min)**
`GreedyRate1`	11274	4.7
`GreedyNN`	11291	1.3
`SMCnn`, M=5	13565	8.1
`SMCnn`, M=25	11513	37.7
`SMCnn`, M=50	11262	74.9
`SMCnn`, M=100	11078	136.9

Table 2: Negative log joint probability and the CPU time in minutes for the greedy algorithms and `SMCnn` with $R = 100$ and various values of M on the same $n = 375$ subset of Birds200.

`GreedyNN` finds a tree that gives a higher joint probability than `SMCnn` with few particles. However, the performance of the SMC algorithm gets better as we increase the number of particles, as expected. The results on the subset of Birds200 dataset with $n = 375$ is summarized in Table 2. For `GreedyNN` and `SMCnn` we used $R = 100$, $K = 20$. The run time of `GreedyRate1` is three times that of `GreedyNN` [2] for this moderate size example and its joint probability is comparable. On 256 examples of SpamBase, the joint probability achieved (and CPU time used) by `GreedyRate1` and `GreedyNN` were 8931 (84 sec) and 5778 (49 sec), respectively.

5 Conclusion

We have proposed a significantly faster SMC algorithm and its greedy counterpart for generating posterior samples from Kingman's coalescent model. The algorithm makes use of standard NN data structures to

[3] This shortcoming of `SMC1` can be circumvented by including the contribution of every pair in the weights, but this will increase the cost to scale cubically.

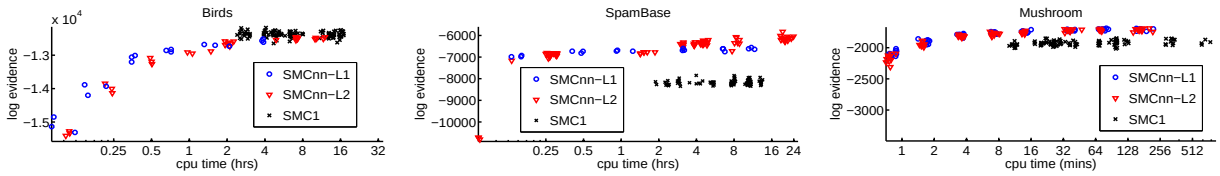

Figure 4: Change in approximated log evidence as a function of CPU time spent by the algorithms on the different datasets. Each point cloud represents several independent runs with the same parameter settings. The fastest `SMC1` runs provided are with $M = 1$, so it is not possible to run `SMC1` at running times faster than shown. Performance of `SMCnn` using both L_1 and L_2 distance are shown. **left:** A subset of Birds data of size 375, with 25 categories. R = 100 , K = 20. **middle:** 256 data points from the SpamBase dataset. R = 100 , K = 20. **right:** 128 data points from Mushrooms dataset. R = 50 , K = 5.

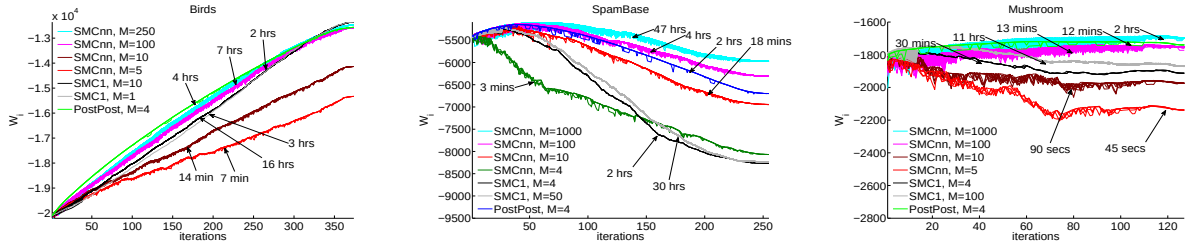

Figure 5: Sample paths, i.e. the w_i's at each iteration through SMC. The intermediate values of weights is of no real importance, however their relative positions are (higher is better). The plots make clear the advantage of using resampling to prune out the bad samples, and the disadvantage of not as informative weights for `SMC1`. **left:** A subset of Birds200 dataset of size $n = 375$ from 25 bird species. **center:** A subset of SpamBase dataset of size $n = 256$. **right:** A subset of the Mushroom dataset of size $n = 128$.

avoid most of the expensive calculations used to formulate a proposal distribution, resulting in orders of magnitude speed up.

The speed of NN algorithms may depend on dataset properties, which could be a problem in applying the algorithm in practice. Our experimental results show that this is not a major issue in practice for the datasets of various dimensions that we tried our model on. Since the NN search is used merely for the proposal distribution, the algorithm does not rely on the returned neighbors to be exact. In this paper we have explored exact nearest neighbor methods but other data structures that might give an even better performance boost could also be explored. Another advantage of the proposed algorithm is that it significantly reduces the memory required because it needs to represent only a subset of pairs.

We approximate the probability of all distant pairs to be the same. A pair-specific approximation that takes into account the similarity of the pairs may result in better proposal distributions. We leave this as future work. Clearly, approximating the posterior too severely may lead to bad proposals and result in particles with very low probability. Thus, there is a fundamental tradeoff in speed versus accuracy which is controlled by M and R. While one could expect the quality of the samples of our SMC algorithm to be poor for $R << \binom{n}{2}$ this effect is often reversed by the fact that we can now entertain many more particles and rely on resampling to prune out bad particles. We presented results showing that when the gain in speed is used for increasing the number of particles, we can often obtain significantly better samples than those of the previous algorithms. We therefore believe that the proposed algorithm presents a significant advance over previous work.

Our greedy algorithm `GreedyNN` is both faster and better than the alternative `GreedyRate1`. It also produces better trees than `SMCnn` with few particles, however `SMCnn` with enough particles produces better samples. The experimental results suggest that the `GreedyNN` can be used as an alternative to `SMCnn` when we require only a single sample tree and when the dataset size is too large to afford a large number of particles. `GreedyNN` may also be used as a tool to guide the choice of R and k to be used for the SMC algorithm with several particles.

The most important contribution of our work lies in the fact that we have managed to speed up inference in the Kingman's coalescent by at least an order of magnitude. This implies that both in population genetics as well as in machine learning this model can now be applied to datasets that were previously out of reach.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0914783 and 0928427. We also thank Pietro Perona for valuable discussions.

References

[1] J. F. C. Kingman. On the genealogy of large populations. *Journal of Applied Probability*, 19:27–43, 1982.

[2] Y. W. Teh, H. Daumé III, and D. M. Roy. Bayesian agglomerative clustering with coalescents. In *Advances in Neural Information Processing Systems*, volume 20, 2008.

[3] D. Görür and Y. W. Teh. An efficient sequential Monte Carlo algorithm for coalescent clustering. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 521–528, 2009.

[4] M. Stephens and P. Donnelly. Inference in molecular population genetics. *Journal of the Royal Statistical Society*, 62:605–655, 2000.

[5] M.K. Kuhner. Coalescent genealogy samplers: windows into population history. *Trends in Ecology & Evolution*, 24(2):86–93, 2009.

[6] M. Nordborg. *Coalescent Theory*. John Wiley & Sons, Ltd, 2004.

[7] RC Griffiths and S. Tavaré. Ancestral inference in population genetics. *Statistical Science*, 9(3):307–319, 1994.

[8] H. Daumé III. Bayesian multitask learning with latent hierarchies. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 135–142. AUAI Press, 2009.

[9] R. C. Griffiths and S. Tavaré. The age of a mutation in a general coalescent tree. *Stochastic Models*, 14:273–295, 1998.

[10] M. Slatkin. A vectorized method of importance sampling with applications to models of mutation and migration. *Theoretical Population Biology*, 62:339–348, 2002.

[11] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[12] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM, 2006.

[13] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[14] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[15] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. *Computer Vision–ECCV 2010*, pages 438–451, 2010.