Thesis!

Ron Visbord

November 3, 2017

1 Calculations Schema

- go over all items in equation 13 and explain how they are calculated
- explain what's calculated in exiting mcmc and what needs new calculations
- explain loci and population independency, enabling simple summing of ln_lds

1.1 Choice of Reference Model

- for post-choosing of reference model, calculate all possible ref models

  TODO: rephrase: start by stating explicitly what we're doing (enabling a single run of mcmc for comparing using multiple ref models)

  When approaching the implementation of mcref, the following considerations occured; By its nature, the use of mcref requires choosing a reference model that is best suited for comparing the competing hypotheses. Every hypothesis being weighted requires a long execution of gphocs. A researcher utilizing mcref may wish to compare between many candidate hypotheses and may wish to obtain comparisons based on many reference models.

  Given these considerations, in an attempt to reduce gphocs runs, we decided each run will calculate and emit sufficient statistics enabling "post-choosing" of the reference model. This is accomplished by having a single gphocs run emit data pertaining to all viable reference models.

- natural trade-off between emitting data and model flexibility

  There's a scaletrade-off between printing and running mcmc. One goal of ours was not having output linear to #loci.

- winning strategy - aggregated stats across all loci

  our solution to the above was the maximal way of printing data $\tilde{\#}$pops and does not restrict post calculations.

2 Calculating sufficient statistics

2.1 Calculating Clade Reference Model

- recursively calculate num coals & coal stats of son clades

  To calculate the genealogy likelihood of the reference model given the locus data, the following data was needed:

  1. Number of coalescence events per population and of migration events per migration band
  2. Coal-stats (needs a descriptive name/explanation)
  3. Reference model theta and tau values and priors

  The construction of the reference model states that the theta and tau priors of the reference and those of the hypothesis models are equal. Because of this, during calculation of the relative-bayes-factor the priors of the hypothesis and reference cancel out. Therefore for the main body of our work, calculation of these priors was skipped. Theta values are required in practice by mcref. They were readily available in existing data structures and were directly emitted to trace files.

- aggregate stats via merge-sort of event-chains of sons and calculate

  Calculation of coal-stats and num-coal of a given clade was done recursively down the population tree, starting from the root population.

  The num-coals of a "leaf-clade" (i.e. a clade containing only a single population leaf) is taken direcly from gphocs' calculation of num-coals. The num-coals of a clade rooted in a higher population is simply the sum of num-coals of clades rooted in it's two child populations plus the num-coals of the top population in the clade.

  The coal-stats of a "leaf-clade" is calculated based on it's chronological event chain. We percieve every event in the event chain as the end of a time interval inside the population where no coalescence took place. The likelihood of this interval is #formula-for-intervals. Using this technique, we run up the event chain, aggregating coal-stats of all intervals. We currently ignore theta, plugging it later (during mcref) into the population genealogy likelihood across all loci.

  The coal-stats of a clade rooted in a higher pop is also based on the clades event-chain, but this event-chain is non-trivial due to populations being "merged" into the current clade. The event-chain for the current clade is obtained by merge-sorting the events of the clades rooted in the two child populations of the current population, and then appending the event-chain of the current population. On this merged event-chain we again calculate coal-stats in the same manner.

2.2 Calculating Comb Reference Model

- partitioning events to below & above comb-age

The main algorithmic difference between the calculation of comb num-coals and coal-stats and of clade num-coals and coal-stats is in the partitioning of events in comb-leaves (i.e. leaf populations descendant from the comb root). Leaf populations under the comb are handled as follows: Events whose occurrence time is below the comb-age are counted towards the leaf coal-stats. Events whose occurrence time is above the comb-age are counted towards the comb coal-stats.

- split border events into intervals below and above comb-age

  During event partitioning we pay careful attention to "border events". These are coalescence or migration events whose interval spans across the comb-age, i.e. the preceding event time is below the comb age and the current event is above the comb age. In this case, we split the event interval into two intervals; An interval from the preceding event time up to the comb-age, which is counted towards the leaf coal-stats, and an interval from the comb-age up to the current event time, which is counted towards the comb coal-stats.

- merge-sort no longer works since we need to repartition events by new comb-age

  Multiple combs rooted in different populations necessarily have different leaf decendants. Since the comb-age is set to the minimal leaf population start time, different combs might always have different comb-ages. For this reason, the event partitioning on leaves takes place a-new for each comb.

  Since the event partitioning differs between combs, the technique used in clade-reference calculation, of reusing event-chains of children clades via merge-sort of children poopulation event chains, no longer works. To calculate comb coal-stats we are forced to recursively recalculate the chronological event-chain for every comb.

2.3 Debugging results

When examining results of gphocs calculations for mcref we were faced with the challenge of validating our results. #explanation-on-why-this-was-needed. We wished to double-check every statistic emitted, with the simple goal of predictably and reliably reaching our intended calculation.

This was accomplished using a variaty of techniques, restricted by the target statistic and reference model and by the tools at our disposal.

- in comb: compare leaves when comb-age:=inf. Compare root comb when comb-age:=0

  To validate our comb coal-stats calculations we permanantly set the comb-age to various values, allowing us to predict results. When setting a high comb-age (essentially infinite), we asserted That the coal-stats of comb-leaves is equal to the leaf population stats calculated by gphocs. When setting comb-age to zero (thus reducing the comb to a clade), we asserted that the coal-stats of the root-comb is equal those of the null reference model (calculated independently by the clade algorithm and by a preexisting gphocs implementation).

- in clade: compare "leaf clade" with leaf. Compare root-clade with gphocs-calculated root-clade
  To assert clade coal-stats, we applied techniques similiar to those used for the comb algorithm. Leaf-rooted clades were compared with leaf stat calculated by gphocs and the root-clade (aka the null reference modle) was compared with independantly calculated stats for the null reference model.

- In tau bounds: monotonouty up pop tree. Assert diff(bound, tau) neg correlated with #loci

  We expected each tau bound to decrease towards its corresponding population tau as the number of loci increases. This due to an increase in the number of coalescence events, leading to an increase in chance of some coalescence event occurring closer to the population tau. We ran a simple set of gphocs experiments using the same sequence data, changing only the number of loci in-use. We observed a negative correlation between the number of loci and the tau bound, as expected.

  Another simple validation we performed was asserting that bounds are monotonously decreasing down the population tree.

3 McRef

- configuration

  When setting up mcref, several parameters are configured. The parameters pertain to standard I/O (e.g. where the trace data files are stored and where to store output), to the phylogenic population models (i.e. the structure of the reference and hypothesis models), to gphocs configuration (e.g. what alpha & beta to use for gamma priora, what print multipliers were applied to trace data when emitted by gphocs etc.), to statistical calculations (e.g. how many bootstrap iterations to run during confidence calculation and how much burn-in and sample-skip to use in genealogy likelihood calculation) and to debugging (e.g. what debug calculations to run and visualizations to emit).

- calculating kingman coalescence and kingman migration for the reference model

  Calculating the reference model genealogy likelihood was done using the standard kingman coalescence model, in the same manner implemented in gphocs. The theta of the actual comb/clade population was set to that of the population at the top of the comb/clade and plugged into #gen_ld_ln-formula.

- calculating tau priors for hypothesis and reference models tau priors for the hypothesis model were calculated using the same gamma-distribution used by gphocs, based on taus emitted during the gphocs run. tau priors for the reference model were calculated using a uniform distribution based on tau-bounds, as described in the chapter about tau bounds.

- estimating variance using bootstrap

  In an attempt to estimate the variability of the genealogy likelihood calculation, bootstrap estimations of the rbf were repeatedly sampled from the trace data.

- optimizing runtime (lazily caching trace files, multiprocess concurrently running comparisons,)

  With the goal of optimizing the practical run-time and usability of mcref, several techniques were employed; trace data files, which are repeatedly read and used, are lazily loaded and cached in each mcref process. Multiple mcref experiments are launched using a single command and are cocurrently run in multiple processes, eventually aggregating summary results to a single log file.

- visualizing results

  To clarify results and to help in the understanding and debugging of mcref runs, several visual outputs were developed. each mcref run emits plots of the genealogy-log-likelihood of the reference model and of the hypothesis model, as well as a plot of the rbf calculation across gphocs iterations and a plot of the harmonic mean of likelihood of the hypothesis model.

- debugging visualizations

  Multiple debug plots are also emitted by mcref. Their goal is to help the researcher assert the experiment was executed as planned. These plots contain the kingman coalescence and kingman migration of every population and migration in both the hypothesis and reference models. They also contain the aggregate coal-stats of the hypothesis and reference model, allowing us to assert that coal-stats of a reference model always exceed those of it's hypothesis #here-goes-an-explanation-of-the-previous-sentence.

4 Tau bounds

4.1 Definitions

4.1.1 Genealogy and Population Phylogeny

A genealogy G is a chronological binary tree whos leaves represent sampled individuals and non-leaf nodes represent coalescence events. each node $e \in G$ has an ocurrence time $t(e)$. TODO: write a more rigurous explanation of a genealogy...

A parameterized population phylogeny P is a chronological binary tree whose nodes represent population-end times $\tau(p)$ (and the birth of two new populations) and whos vertices represent the time span of the population. TODO: Write a better formalization of a population phylogeny (mentioning that Leaf events are permanantly mapped to population they are "born in")...

4.1.2 Some operators, functions and observations

- $leaves(e)$ is the set of all leaf decendants of e
- $father(p/e)$, $leftson(p/e)$ & $rightson(p/e)$ are operators for traversing the population/genealogy tree
- $mrcaPop(e)$ is the most recent population from which all populations in $\{pop(l) | l \in leaves(e)\}$ are descended

- The binary operator $\geq^{p/e}$ between populations/events denotes the natural ancestry partial order relation between populations/events in the population/genealogy tree P/G. We say $p/e_1 \geq^{p/e} p/e_2$ if p/e_1 is ancestral or identical to p/e_2.
  - Note that $p_1 \geq^p p_2 \Rightarrow \tau(p_1) \geq \tau(p_2)$
- Note that if distict subtrees rooted at t_1 and t_2 share decendants then one must be ancestral to the another, i.e. $t_1 >_t t_2 \oplus t_2 >_t t_1$

4.1.3 Embeddability

An embedding of genealogy G in phylogeny P is a mapping $e \mapsto p$ denoted $pop(e)$ of all events onto populations s.t. -

1. $\tau(pop(e)) < t(e) \leq \tau(father(pop(e)))$
2. $pop(e) \geq^p pop(leftson(e))$ and $pop(e) \geq^p pop(rightson(e))$

Intuitively, an embedding is a population assignment to each event, that is consistent with the population phylogeny times (condition 1) and where each edge follows a path of ancestry (condition 2).

Note that if $e_1 \geq_e e_2$ then $pop(e_1) \geq_p pop(e_2)$. TODO: formally state that this is obvious.

4.2 Uniqueness of an Embedding

mini Lemma:

If G has an embedding in P then the embedding is unique

Proof: Assume to the contrary that p_1 & p_2 are two distict populations that meet conditions 1,2,3 of embeddablity for some event e.

p_1 and p_2 have common decendants (as they both meet condition 2), so they must be ordered. Since they are ordered their existence cannot overlap, in contradiction to their coexistence with e (as defined by conditions 1 & 2). Hence at most one population may meet conditions 1 & 2 of an embedding for a specific event, hence if an embedding for G exists, it is unique.

4.3 Existence of an Embedding

Lemma:

$$G \text{ has an embedding in } P \Leftrightarrow \forall e \in G, t(e) \geq \tau(mrcaPop(e))$$

Proof:

$\Rightarrow$ If G has an embedding then $\forall e \in G$,

$$\forall l \in leaves(e), e \geq_e l$$

$$\Downarrow$$

$$\forall l \in leaves(e), pop(e) \geq_p pop(l)$$
$$\Downarrow$$
$$pop(e) \geq_p mrcaPop(e)$$
$$\Downarrow$$
$$t(e) \geq \tau(pop(e)) \geq \tau(mrcaPop(e))$$

$\Leftarrow$ If $\forall e \in G, t(e) \geq \tau(mrcaPop(e))$ then :

Consider the set of populations ancestral to $mrcaPop(e)$. In this set there must be exactly one population coexisting with e; At least one since $t(e) \geq \tau(mrca_p(e))$ (and the time-span of populations is continuous from 0 to ∞) and at most one due to the same cosiderations presented in uniqueness of an embedding. Denote this population p^*. To show that that $p^* \geq_p pop(leftson(e))$ and $p^* \geq_p pop(rightson(e))$ consider w.l.o.g $pop(leftson(e))$. Since the subtrees of p^* and $pop(leftson(e))$ are not disjoint (as they both contain the populations of some of $leaves(leftson(e))$), they must be ordered or identical. Since p^* coexists with e and $pop(leftson(e))$ coexists with $leftson(e)$ and $t(e) > t(leaftson(e))$, population $pop(leftson(e))$ cannot be strictly ancestral to p^*. Therefore $p^* \geq_p pop(leftson(e))$ and $p^* \geq_p pop(rightson(e))$, making it a proper embedding for e.

4.4 Algorithm

Given a genealogy G and a Population tree P without taus, we want to define the set of possible tau ranges s.t. G is embeddable in P.

- For leaf populations - $\tau(p) = 0$
- For all populations - $\tau(p) \leq \tau(father(p))$
- For all genealogy leaves - $t(e) \geq \tau(mrcaPop(e))$

...... more explanation of the actual algorithm......

The following is a python snippet implementing the algorithm. The main function is *find_tau_bounds*. For full implementation, see appendix A:

```
def find_tau_bounds(root_event):
    tau_bounds = defaultdict(lambda: float('Inf'))

    _find_tau_bounds_rec(root_event, tau_bounds)

    return tau_bounds
```

```
def _find_tau_bounds_rec(event, tau_bounds):
    if event.lca_pop is not None:  # event is a leaf
        tau_bounds[event.lca_pop] = event.time
        return

    _find_tau_bounds_rec(event.left, tau_bounds)
    _find_tau_bounds_rec(event.right, tau_bounds)

    event.lca_pop = lca(event.left.lca_pop, event.right.lca_pop)

    for pop in descendants(event.lca_pop):
        tau_bounds[pop] = min(tau_bounds[pop], event.time)
```