Gene genealogies in a haploid population evolving according to sweepstakes reproduction – approximating $\mathbb{E}[R_i(n)]$ for the time-changed δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent

Bjarki Eldon[1,2]

Fix $0 < \alpha < 2$ and $0 < \gamma \le 1$. Let $\{\xi^n\} \equiv \{\xi^n(t) : t \ge 0\}$ be the continuous-time time-changed δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent with time-change function $G(t) = \int_0^t e^{\rho s} ds$. Write $\#A$ for the number of elements in a given finite set A, $L_i(n) \equiv \int_0^{\tau(n)} \#\{\xi \in \xi^n(t) : \#\xi = i\}\, dt$ and $L(n) \equiv \int_0^{\tau(n)} \#\xi^n(t) dt$ and $\tau(n) \equiv \inf\{t \ge 0 : \#\xi^n(t) = 1\}$ for $i \in \{1, 2, \ldots, n-1\}$. We then have $L(n) = L_1(n) + \cdots + L_{n-1}(n)$. Define $R_i(n) \equiv L_i(n)/\sum_j L_j(n)$ for $i = 1, 2, \ldots, n-1$. Interpreting $\{\xi^n\}$ as 'trees' we may view $L_i(n)$ as the random total length of branches supporting $i \in \{1, 2, \ldots, n-1\}$ leaves, with the length measured in coalescent time units, and n sample size. With this C++ code we sample the time-changed δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent with $0 < \alpha < 2$ and exponential growth parameter $\rho \ge 0$ fixed, and approximate $\mathbb{E}[R_i(n)]$

Contents

1 Copyright

Copyright © 2026 Bjarki Eldon

This document and any source code it contains is distributed under the terms of the GNU General Public Licence (version ≥ 3). You should have received a copy of the licence along with this file (see file COPYING).

The source codes described in this document are free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

[1] beldon11@gmail.com

[2] compiled @ 3:37pm on Tuesday 6th January, 2026; using
kernel 6.18.3+deb14-amd64 GNU/Linux
GNU bash, version 5.3.3(1)-release (x86_64-pc-linux-gnu)
CTANGLE 4.12.1 (TeX Live 2025/Debian)
g++ (Debian 15.2.0-12) 15.2.0
GSL 2.8
CWEAVE 4.12.1 (TeX Live 2025/Debian)
This is LuaHBTeX, Version 1.22.0 (TeX Live 2025/Debian) Development id: 7673
SpiX 1.3.0
GNU Awk 5.3.2, API 4.0, PMA Avon 8-g1, (GNU MPFR 4.2.2, GNU MP 6.3.0)
GNU Emacs 30.2

This document and the code it contains is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this file (see COPYING). If not, see http://www.gnu.org/licenses/.

2 compilation and output

This CWEB [KL94] document (the `.w` file) can be compiled with `cweave` to generate a `.tex` file, and with `ctangle` to generate a `.c` [KR88] C++ code file.

Use the shell tool `spix` on the script appearing before the preamble (the lines starting with %$); simply

```
spix /path/to/the/sourcefile
```

where `sourcefile` is the `.w` file

One may also copy the script into a file and run `parallel` [Tan11] :

```
parallel --gnu -j1 :::: /path/to/scriptfile
```

3 intro

Here we consider the continuous-time δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent with the time-change corresponding to exponential population growth where the time-change function $G(t) = \int_0^t e^{\rho s} ds$ for some fixed $\rho \geq 0$.

One samples the δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent by sampling group sizes $2 \leq k_1, \ldots, k_r \leq n$ with $\sum_i k_i \leq n$, $s = n - \sum_i k_i$, according to

$$
\begin{aligned}
\lambda_{n;k;s} &= \frac{1}{C_{\kappa,\alpha,\gamma}} \binom{n}{k} \left(\mathbb{1}_{\{r=1,k_1=2\}} C_\kappa + \frac{c\alpha}{\mathbb{m}^\alpha} B(\gamma, k-\alpha, n-k+\alpha) \right) \\
C_\kappa &= \frac{2}{\mathbb{m}^2} \left(\mathbb{1}_{\{\kappa=2\}} + \mathbb{1}_{\{\kappa>2\}} \frac{c_\kappa}{2^\kappa(\kappa-2)(\kappa-1)} \right) \\
C_{\kappa,\alpha,\gamma} &= C_\kappa + c\alpha \mathbb{m}^{-\alpha} B(\gamma, 2-\alpha, \alpha) \\
\mathbb{m} &= 1 + \frac{1+2^{1-\kappa}}{2(\kappa-1)}
\end{aligned}
\tag{1}
$$

where $2 + \kappa < c_\kappa < \kappa^2$ when $\kappa > 2$, and $B(p, a, b) \equiv \int_0^1 \mathbb{1}_{\{0<t\leq p\}} t^{a-1}(1-t)^{b-1} dt$ for $a, b > 0$ and $0 < p \leq 1$ fixed.

Let $\#A$ denote the number of elements in a finite set A. For a given coalescent $\{\xi^n\}$ write $L_i(n) \equiv \int_0^{\tau(n)} \#\{\xi \in \xi^n(t) : \#\xi = i\} \, dt$ and $L(n) \equiv \int_0^{\tau(n)} \#\xi^n(t) dt$ where $\tau(n) \equiv \inf\{t \geq 0 : \#\xi^n(t) = 1\}$. Then $L(n) = L_1(n) + \cdots + L_{n-1}(n)$. Write $R_i(n) \equiv L_i(n)/L(n)$ for $i = 1, 2, \ldots, n-1$. With this C++ code we use simulations to approximate $\mathbb{E}[R_i(n)]$ when the coalescent is the time-changed continuous-time δ_0-Beta$(\gamma, 2-\alpha, \alpha)$-coalescent with time-change function $G(t) = \int_0^t e^{\rho s} ds$ for a given fixed $\rho \geq 0$. See Figure 1 in § 5 for an example

The code follows in § 4.1–§ 4.5, we conclude in § 5. Comments within the code in **this font and colour**

4 code

4.1 includes

the included libraries

5 ⟨includes 5⟩ ≡

```
#include <iostream>
#include <fstream>
#include <vector>
#include <random>
#include <functional>
#include <memory>
#include <utility>
#include <algorithm>
#include <ctime>
#include <cstdlib>
#include <cmath>
#include <list>
#include <string>
#include <fstream>
#include <chrono>
#include <forward_list>
#include <assert.h>
#include <math.h>
#include <unistd.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_sf.h>
#include <boost/math/special_functions/beta.hpp>
#include "deltanull_incbeta_expgrowth.hpp"
```

This code is used in chunk 11.

4.2 random number generators

the random number generators

6 ⟨rngs 6⟩ ≡

```
std::random_device randomseed;  /* Standard Mersenne twister random number engine */
std::mt19937_64 rng(randomseed());
gsl_rng * rngtype;
void setup_rng(const unsigned long int s)
{
  const gsl_rng_type *T;
  gsl_rng_env_setup();
  T = gsl_rng_default;
  rngtype = gsl_rng_alloc(T);
  gsl_rng_set(rngtype, s);
}
```

This code is used in chunk 11.

4.3 the merging rate

7 $\langle\, \lambda_{n;k;s}\ 7 \,\rangle \equiv$

```
double rate(const double m, const double k) {
    assert(k ≤ m);     /*
        the full (non-normalised) incomplete beta function */
    auto incbeta = [](const double a, const double b, const double x)
    {
      return boost::math::beta(static_cast⟨long double⟩(a), static_cast⟨long
          double⟩(b), static_cast⟨long double⟩(x));
    }
    ;
    const long double lbinom = lgammal(m + 1) − lgammal(k + 1) − lgammal(m − k + 1);
    const long double dnull = (int) k < 3 ? logl((long double)(CKAPPA)) : 0;
    const long double bpart = logl((long double)(CEPS ∗ ALPHA ∗ incbeta(k − ALPHA,
        m + ALPHA − k, GAMMA)/pow(MINF, ALPHA)));
    return (static_cast⟨double⟩(expl(lbinom + dnull + bpart))/CKAG); }
```

This code is used in chunk 11.

¶ **totalrate**
compute the total rate

8 $\langle\, \lambda_{n}\ 8 \,\rangle \equiv$

```
void totalrate ( std::vector < double > &v )
{
  for (double m = 2; m ≤ SAMPLE_SIZE; ++m) {
    for (double j = 2; j ≤ m; ++j) {
      assert(j ≤ m);     /*
          § 4.3 */
      v[m] += rate(m, j);
    }
  }
}
```

This code is used in chunk 11.

4.4 one experiment

9 ⟨ genealogy 9 ⟩ ≡

```
static void genealogy ( const std::vector < double > &vlambdan, std::vector <
    double > &vri ) {
std::vector < int > t(SAMPLE_SIZE, 1);
std::size_t ms
{ }
;
double timi
{ }
;
int newb
{ }
; std::vector < double > vb(SAMPLE_SIZE);
std::size_t q
{ }
;
double otimi
{ }
; auto newtime = [](const double lambdab, const double oldtime)
{
  return (RHO > DBL_EPSILON ? log1p(−(RHO ∗ exp(−RHO ∗ oldtime)/lambdab) ∗
      log(gsl_rng_uniform_pos(rngtype)))/RHO : gsl_ran_exponential(rngtype, 1./lambdab));
}
;
auto getmerger = [&vlambdan](const double m)
{
  const double u = gsl_rng_uniform(rngtype);
  double j = 2;     /*
      § 4.3 */
  double s = rate(m, j);
  while (u > s/vlambdan[static_cast⟨int⟩(m)]) {
    ++j;
    assert(j ≤ m);
    s += rate(m, j);
  }
  return static_cast⟨std::size_t⟩(j);
}
;
while (t.size( ) > 1) {
timi = newtime(vlambdan[t.size( )], otimi);
assert(timi > DBL_EPSILON);
otimi += timi;
std::for_each (t.cbegin( ), t.cend( ), [&timi, &vb](const int t)
{
  assert(t > 0);
  vb[0] += timi;
  vb[t] += timi;
```

```
}
);
ms = getmerger(static_cast⟨double⟩(t.size()));
std::shuffle(t.begin(), t.end(), rng);
newb = std::accumulate(t.rbegin(), t.rbegin() + ms, 0);
q = t.size();
t.resize(q − ms);
t.push_back(newb); }
assert(vb[0] > DBL_EPSILON);
const double d = vb[0];
std::transform (vb.cbegin(), vb.cend(), vri.begin(), vri.begin(), [&d](const auto &x, const
        auto &y)
{
  return y + (static_cast⟨double⟩(x)/d);
}
); }
```

This code is used in chunk 11.

¶ **approximate** $\mathbb{E}[R_i(n)]$.

10 ⟨ go ahead 10 ⟩ ≡

```
void approximate() {
    std::vector < double > v_lambdan(static_cast⟨int⟩(SAMPLE_SIZE) + 1);
    totalrate(v_lambdan);
    std::vector < double > vri(static_cast⟨int⟩(SAMPLE_SIZE));
    int r = EXPERIMENTS + 1;
    while (−−r > 0) {      /*
          § 4.4 */
      genealogy(v_lambdan, vri);
    }
    std::for_each (vri.begin(), vri.end(), [](const auto &x)
    {
      std::cout ≪ x ≪ '\n';
    }
    ); }
```

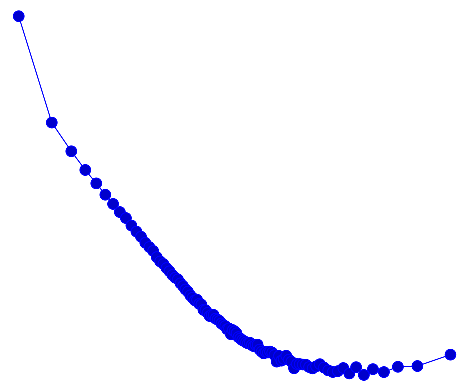
This code is used in chunk 11.

4.5 main

the *main* module

11
```
/*
§ 4.1 */
⟨ includes 5 ⟩  /*
  § 4.2 */
⟨ rngs 6 ⟩  /*
  § 4.3 */
⟨ λ_{n;k;s} 7 ⟩
⟨ λ_n 8 ⟩  /*
  § 4.4 */
⟨ genealogy 9 ⟩
⟨ go ahead 10 ⟩
int main(int argc, const char *argv[])
{  /*
    § 4.2 */
  setup_rng(static_cast⟨long unsigned⟩(atoi(argv[1])));  /*
    § 4.4 */
  approximate();
  gsl_rng_free(rngtype);
  return GSL_SUCCESS;
}
```

5 conclusions and bibliography



$\log(i/n) - \log(1 - i/n)$

Figure 1: *An example approximation of* $\mathbb{E}[R_i(n)]$ *for the given parameter values and graphed as logits against* $\log(i/n) - \log(1 - i/n)$ *for* $i = 1, 2, \ldots, n-1$ *where* n *is sample size*

References

[D2024] Diamantidis, Dimitrios and Fan, Wai-Tong (Louis) and Birkner, Matthias and Wakeley, John. Bursts of coalescence within population pedigrees whenever big families occur. Genetics Volume 227, February 2024.
https://dx.doi.org/10.1093/genetics/iyae030.

[CDEH25] JA Chetwyn-Diggle and Bjarki Eldon. Beta-coalescents when sample size is large. 2026 bioRxiv https://doi.org/10.64898/2025.12.30.697022

[DK99] P Donnelly and T G Kurtz. Particle representations for measure-valued population models. *Ann Probab*, 27:166–205, 1999.
https://dx.doi.org/10.1214/aop/1022677258

[KL94] Donald Ervin Knuth and Silvio Levy. *The CWEB system of structured documentation: version 3.0*. Addison-Wesley Longman Publishing Co., Inc., Reading, Massachusetts, 1994.

[KR88] Brian W Kernighan and Dennis M Ritchie. The C programming language, 1988.

[Pit99] J Pitman. Coalescents with multiple collisions. *Ann Probab*, 27:1870–1902, 1999.
https://dx.doi.org/10.1214/aop/1022874819

[Sag99] S Sagitov. The general coalescent with asynchronous mergers of ancestral lines. *J Appl Probab*, 36:1116–1125, 1999.
https://doi.org/10.1239/jap/1032374759

[Sch03] J Schweinsberg. Coalescent processes obtained from supercritical Galton-Watson processes. *Stoch Proc Appl*, 106:107–139, 2003.
https://doi.org/10.1016/S0304-4149(03)00028-0

[S00] J Schweinsberg. Coalescents with simultaneous multiple collisions. *Electronic Journal of Probability*, 5:1–50, 2000.
https://dx.doi.org/10.1214/EJP.v5-68

[Tan11] O Tange. GNU parallel – the command-line power tool. The USENIX Magazine, 2011.

Index

List of Refinements

⟨ $\lambda_{n;k;s}$ 7 ⟩ Used in chunk 11.
⟨ λ_n 8 ⟩ Used in chunk 11.
⟨ genealogy 9 ⟩ Used in chunk 11.
⟨ go ahead 10 ⟩ Used in chunk 11.
⟨ includes 5 ⟩ Used in chunk 11.
⟨ rngs 6 ⟩ Used in chunk 11.