

Gene genealogies in a haploid population evolving according to sweepstakes reproduction – approximating $\mathbb{E}[R_i(n)]$ for the time-changed δ_0 -Poisson-Dirichlet($\alpha, 0$)-coalescent

BJARKI ELDON¹² 

Fix $0 < \alpha < 2$ and $0 < \gamma \leq 1$. Let $\{\xi^n\} \equiv \{\xi^n(t) : t \geq 0\}$ be the continuous-time time-changed δ_0 -Poisson-Dirichlet($\alpha, 0$)-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 $\tau(n) \equiv \inf\{t \geq 0 : \#\xi^n(t) = 1\}$ for $i \in \{1, 2, \dots, n-1\}$. We then have $L(n) = L_1(n) + \dots + L_{n-1}(n)$. Define $R_i(n) \equiv L_i(n) / \sum_j L_j(n)$ for $i = 1, 2, \dots, n-1$. Interpreting $\{\xi^n\}$ as ‘trees’ we may view $L_i(n)$ as the random total length of branches supporting $i \in \{1, 2, \dots, n-1\}$ leaves, with the length measured in coalescent time units, and n sample size. With this C++ code we sample the time-changed δ_0 -Poisson-Dirichlet($\alpha, 0$)-coalescent with $0 < \alpha < 2$ and exponential growth parameter $\rho \geq 0$ fixed, and approximate $\mathbb{E}[R_i(n)]$

Contents

1	Copyright	1
2	compilation and output	3
3	intro	4
4	code	5
4.1	the coalescent rate	6
4.2	partitions summing to a given number	8
4.3	all partitions summing to a given number	10
4.4	get all partitions	12
4.5	tree modules	13
4.6	one experiment	14
5	conclusions and bibliography	17

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).

¹beldon11@gmail.com

²compiled @ 3:57pm 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

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.

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

We consider the continuous-time time-changed δ_0 -Poisson-Dirichlet($\alpha, 0$)-coalescent, with $A \equiv 1 / \left(\prod_{j=2}^n (\sum_i \mathbb{1}_{\{k_i=j\}})! \right)$, transition rates

$$\lambda_{n;k_1, \dots, k_r; s} = A \binom{n}{k_1 \dots k_r s} \left(\mathbb{1}_{\{r=1, k_1=2\}} \frac{C_\kappa}{C_\kappa + c(1-\alpha)} + \frac{c p_{n;k_1, \dots, k_r; s}}{C_\kappa + c(1-\alpha)} \right) \quad (1)$$

where $0 < \alpha < 1$, $\kappa \geq 2$, $c \geq 0$ all fixed, and

$$\begin{aligned} p_{n;k_1, \dots, k_r; s} &= \frac{\alpha^{r+s-1} \Gamma(r+s)}{\Gamma(n)} \prod_{i=1}^r (k_i - \alpha - 1)_{k_i-1} \\ C_\kappa &= \mathbb{1}_{\{\kappa=2\}} \frac{2}{m_\infty^2} + \mathbb{1}_{\{\kappa>2\}} \frac{c_\kappa}{2\kappa(\kappa-2)(\kappa-1)} \end{aligned} \quad (2)$$

and $\kappa + 2 < c_\kappa < \kappa^2$ for $\kappa > 2$.

4 code

¶ includes.
the included libraries

```
5 <includes 5> ≡  
#include <iostream>  
#include <cstdlib>  
#include <iterator>  
#include <random>  
#include <fstream>  
#include <iomanip>  
#include <vector>  
#include <numeric>  
#include <functional>  
#include <algorithm>  
#include <cmath>  
#include <unordered_map>  
#include <assert.h>  
#include <float.h>  
#include <fenv.h>  
#include <gsl/gsl_rng.h>  
#include <gsl/gsl_randist.h>  
#include <gsl/gsl_math.h>  
#include "headerfile.hpp"
```

This code is used in chunk 17.

¶ the 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;  
static void setup_rng(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 17.

4.1 the coalescent rate

the coalescent rate $\lambda_{n;k_1,\dots,k_r;s}$

```

7   $\langle \lambda_{n;k_1,\dots,k_r;s} \rangle \equiv$ 
    static double lambdanks (const double n, const std::vector < unsigned int > &v_k ) {
        assert(v_k[0] > 1);
        double d
        { }
        ;
        double k
        { }
        ;
        double f
        { 1 };
        const double r = static_cast<double>(v_k.size()); std::unordered_map < unsigned int ,
            unsigned int > counts
        { }
        ;
        const auto descending_factorial = [](const double x, const double m)
        {
            double p = 1;
            for (double i = 0; i < m; ++i) {
                p *= (x - i);
            }
            return p;
        }
        ;
        const auto veldi = [](const double x, const double y)
        {
            feclearexcept(FE_ALL_EXCEPT);
            const double d = pow(x, y);
            return (fetetestexcept(FE_UNDERFLOW) ? 0. : (fetetestexcept(FE_OVERFLOW) ? FLT_MAX : d));
        }
        ;
        for (std::size_t i = 0; i < v_k.size(); ++i) {
            f *= descending_factorial(static_cast<double>(v_k[i]) - 1. - ALPHA,
                static_cast<double>(v_k[i]) - 1); /*
                count occurrence of each merger size */
            ++counts[v_k[i]];
            k += static_cast<double>(v_k[i]);
            d += lgamma(static_cast<double>(v_k[i] + 1));
        }
        assert(k < n + 1);
        const double s = n - k;
        const double p = static_cast<double>( std::accumulate (counts.begin(), counts.end(), 0,
            [](<double a, const auto &x)
            {
                return a + lgamma((double) x.second + 1);
            }
            ) );

```

```

const double l = ((v_k.size() < 2 ? (v_k[0] < 3 ? 1. : 0) : 0) * CKAPPA) + (CEPS * veldi(ALPHA,
    r + s - 1) * tgamma(r + s) * f/tgamma(n));
return (veldi(exp(1),
    (lgamma(n + 1.) - d) - lgamma(n - k + 1) - p) * l/(CKAPPA + (CEPS * (1 - ALPHA)))); }

```

This code is used in chunk 17.

4.2 partitions summing to a given number

generate the partitions of a given size summing to a given number *myInt*

8 $\langle \text{GenPartitions } 8 \rangle \equiv$

```
static double GenPartitions (const unsigned int m, const unsigned int myInt, const
    unsigned int PartitionSize, unsigned int MinVal, unsigned int MaxVal,
    std::vector< std::pair< double, std::vector< unsigned int >>> &v_l_k, std::vector
    < double > &lrates_sorting ) { /*
    m is the given number of blocks; PartitionSize is the size of the partitions; the
    partitions sum to myInt */
double lrate
{}
;
double sumrates
{}
; std::vector< unsigned int > partition(PartitionSize);
unsigned int idx_Last = PartitionSize - 1;
unsigned int idx_Dec = idx_Last;
unsigned int idx_Spill = 0;
unsigned int idx_SpillPrev;
unsigned int LeftRemain = myInt - MaxVal - (idx_Dec - 1) * MinVal;
partition[idx_Dec] = MaxVal + 1;
do {
    unsigned int val_Dec = partition[idx_Dec] - 1;
    partition[idx_Dec] = val_Dec;
    idx_SpillPrev = idx_Spill;
    idx_Spill = idx_Dec - 1;
    while (LeftRemain > val_Dec) {
        partition[idx_Spill++] = val_Dec;
        LeftRemain -= val_Dec - MinVal;
    }
    partition[idx_Spill] = LeftRemain;
    const char a = (idx_Spill) ? ~((-3 >> (LeftRemain - MinVal)) << 2) : 11;
    const char b = (-3 >> (val_Dec - LeftRemain));
    switch (a & b) {
    case 1: case 2: case 3: idx_Dec = idx_Spill;
        LeftRemain = 1 + (idx_Spill - idx_Dec + 1) * MinVal;
        break;
    case 5:
        for (++idx_Dec, LeftRemain = (idx_Dec - idx_Spill) * val_Dec;
            (idx_Dec ≤ idx_Last) ∧ (partition[idx_Dec] ≤ MinVal); idx_Dec++)
            LeftRemain += partition[idx_Dec];
        LeftRemain += 1 + (idx_Spill - idx_Dec + 1) * MinVal;
        break;
    case 6: case 7: case 11: idx_Dec = idx_Spill + 1;
        LeftRemain += 1 + (idx_Spill - idx_Dec + 1) * MinVal;
        break;
    case 9:
        for (++idx_Dec, LeftRemain = idx_Dec * val_Dec;
            (idx_Dec ≤ idx_Last) ∧ (partition[idx_Dec] ≤ (val_Dec + 1));
            idx_Dec++) LeftRemain += partition[idx_Dec];
        LeftRemain += 1 - (idx_Dec - 1) * MinVal;
```



```

    break;
case 10:
    for (LeftRemain += idx_Spill * MinVal + (idx_Dec - idx_Spill) * val_Dec + 1, ++idx_Dec;
        (idx_Dec ≤ idx_Last) ∧ (partition[idx_Dec] ≤ (val_Dec - 1)); idx_Dec++)
        LeftRemain += partition[idx_Dec];
    LeftRemain -= (idx_Dec - 1) * MinVal;
    break;
}
while (idx_Spill > idx_SpillPrev) partition[--idx_Spill] = MinVal;
assert(static_cast<unsigned int>(std::accumulate(partition.begin(), partition.end(),
    0)) ≡ myInt);    /*
    § 4.1 */
lrate = lambdanks(static_cast<double>(m), partition);
assert(lrate ≥ 0);
v_l_k.push_back(std::make_pair(lrate, partition));
rates_sorting.push_back(lrate);
sumrates += lrate;
} while (idx_Dec ≤ idx_Last);
assert(sumrates ≥ 0);
return sumrates; }

```

This code is used in chunk 17.

4.3 all partitions summing to a given number

get all partitions (merger sizes) summing to a given number

9 $\langle \text{allmergers_sum_m } 9 \rangle \equiv$

```
static double allmergers_sum_m (const unsigned int n, const unsigned
    int m, std::vector < std::pair < double , std::vector < unsigned
    int >>> &v__l_k, std::vector < double > &v__lrates_sort ) {    /*
    n is the number of blocks; the partitions sum to m */
const std::vector < unsigned int > v__m
{m};    /*
§ 4.1 */

double sumr = lambdanks(static_cast<double>(n), v__m);
v__l_k.push_back(std::make_pair(sumr, v__m));
v__lrates_sort.push_back(sumr);
if (m > 3) {
    for (unsigned int s = 2; s ≤ m/2; ++s) {
        assert(m > 2 * (s - 1));    /*
        § 4.2 */
        sumr += GenPartitions(n, m, s, 2, m - (2 * (s - 1)), v__l_k, v__lrates_sort);
    }
}
assert(sumr ≥ 0);
return sumr; }
```

This code is used in chunk 17.

¶ write mergers to file.

10 $\langle \text{ratesmergersfile } 10 \rangle \equiv$

```
static void ratesmergersfile (const unsigned int n, const std::vector < unsigned
    int > &v__indx, const std::vector < std::pair < double , std::vector <
    unsigned int >>> &v__l_k, const double s, std::vector < std::vector <
    double >> &a__cmf ) {
    assert(s > 0);
    double cmf
    {}
    ;
    std::ofstream f;
    f.open("gg_" + std::to_string(n) + "_.txt", std::ios::app);
    a__cmf[n].clear();
    for (const auto &i:v__indx) {
        cmf += (v__l_k[i].first)/s;
        assert(cmf ≥ 0);
        a__cmf[n].push_back(cmf);
        assert((v__l_k[i].second).size() > 0);
        for (const auto &x:v__l_k[i].second)
        {
            f << x << ' ';
        }
        f << '\n'; }
    f.close();
```

```
assert(abs(cmf - 1.) < 0.999999); }
```

This code is used in chunk 17.

4.4 get all partitions

```

11 <allmergers_when_n_blocks 11> ≡
    static void allmergers_when_n_blocks (const unsigned int n, std::vector <
        double > &v__lambdan, std::vector < std::vector < double >> &a__cmf ) {
        std::vector < std::pair < double , std::vector < unsigned int >>> vlk
    {}
    ; std::vector < double > ratetosort
    {}
    ;
    ratetosort.clear();
    double lambdan
    {}
    ;
    vlk.clear();
    assert(n > 1);
    for (unsigned int k = 2; k ≤ n; ++k) { /*
        the partition sums to k; the number of blocks is n */
        lambdan += allmergers_sum_m(n, k, vlk, ratetosort);
    } /*
        record the total rate when n blocks */ /*
        use for sampling time */
    assert(lambdan > 0);
    v__lambdan[n] = lambdan;
    std::vector < unsigned int > indx(ratetosort.size());
    std::iota(indx.begin(), indx.end(), 0);
    std::stable_sort (indx.begin(), indx.end(), [&ratetosort](const unsigned int x, const
        unsigned int y)
    {
        return ratetosort[x] > ratetosort[y];
    }
    ); /*
        merger rates sorted in descending order; print the cmf and rates to file */
    ratesmergersfile(n, indx, vlk, v__lambdan[n], a__cmf); }

```

This code is used in chunk 17.

¶ get all partitions.

generate all mergers up to sample size

```

12 <all possible partitions 12> ≡
    static void allmergers ( std::vector < double > &vlmn, std::vector < std::vector <
        double >> &acmf )
    {
        for (unsigned int tmpn = 2; tmpn ≤ SAMPLE_SIZE; ++tmpn) { /*
            § 4.4 */
            allmergers_when_n_blocks(tmpn, vlmn, acmf);
        }
    }

```

This code is used in chunk 17.

4.5 tree modules

13 $\langle \text{updatetree } 13 \rangle \equiv$

```
static void updatetree ( std::vector < unsigned int > &tre, const std::vector < unsigned
    int > &mersersizes ) {
    assert(mersersizes.size() > 0); std::vector < unsigned int > newblocks
    {}
    ;
    newblocks.clear();
    std::shuffle(tre.begin(), tre.end(), rng);
    std::size_t s = tre.size(); for (const auto &k:mersersizes)
    {
        assert(k > 1);
        assert(k ≤ s);
        s -= k; /*
            record the size of the merging blocks */
        newblocks.push_back(std::accumulate(tre.rbegin(), tre.rbegin() + k, 0)); /*
            remove the blocks that merged */
        tre.resize(s);
    }
    assert(newblocks.size() > 0);
    assert(static_cast<unsigned int>(std::accumulate(newblocks.begin(), newblocks.end(),
        0)) ≤ SAMPLE_SIZE);
    tre.insert(tre.end(), newblocks.begin(), newblocks.end());
    assert(static_cast<unsigned int>(std::accumulate(tre.begin(), tre.end(),
        0)) ≡ SAMPLE_SIZE); }
```

This code is used in chunk 17.

¶ read partitions.

14 $\langle \text{readmersersizes } 14 \rangle \equiv$

```
static void readmersersizes (const unsigned int n, const unsigned int j, std::vector <
    unsigned int > &v__mersers ) {
    std::ifstream f("gg_" + std::to_string(n) + "_txt");
    std::string line {}
    ;
    v__mersers.clear();
    for (unsigned int i = 0; std::getline (f, line ) ^ i < j; ++i ) { if (i ≥ j - 1) {
        std::stringstream ss ( line ) ;
        v__mersers = std::vector < unsigned int > ( std::istream_iterator < unsigned int > (ss),
            {} ) ; } }
    assert(v__mersers.size() > 0);
    assert(v__mersers[0] > 1);
    assert(v__mersers.back() > 1);
    f.close(); }
```

This code is used in chunk 17.

4.6 one experiment

15 $\langle \text{onexperiment } 15 \rangle \equiv$

```

static void onexperiment ( std::vector < double > &v__ri, std::vector < double > &vl,
    const std::vector < double > &v__lambda__n, const std::vector <
    std::vector < double >> &a__cmf ) {
    std::vector < unsigned int > v__tre(SAMPLE_SIZE,1);
    std::fill(vl.begin(), vl.end(), 0);
    unsigned int lina
    {}
    ;
    double tau
    {}
    ;
    double new_time
    {}
    ; std::vector < unsigned int > v__merger_sizes(SAMPLE_SIZE/2);
    v__merger_sizes.reserve(SAMPLE_SIZE/2); auto newtime = [](const double
        lambdam, const double tau)
    {
        return (RHO > DBL_EPSILON ? log1p(-(RHO * exp(-RHO * tau)/lambdam) * log(1. -
            gsl_rng_uniform_pos(rngtype)))/RHO : gsl_ran_exponential(rngtype,
            1./lambdam));
    }
    ; auto updatelengths = [&v__tre, &vl](const double t) {
        for (const auto &b:v__tre)
        {
            assert(b > 0);
            assert(b < SAMPLE_SIZE);
            vl[0] += t;
            vl[b] += t;
        }
        ;
        auto samplemerger = [] (const unsigned int n, const std::vector <
            double > &v__cmf )
        {
            unsigned int j
            {}
            ;
            const double u = gsl_rng_uniform(rngtype);
            while (u > v__cmf[j]) {
                ++j;
            }
            return j;
        }
        ;
        while (v__tre.size() > 1) {
            new_time = newtime(v__lambda__n[v__tre.size()], tau);
            tau += new_time;

```

```

        updatelengths(new_time);
        lina = samplemerger(v__tre.size(), a__cmf[v__tre.size()]);
        readmergersizes(v__tre.size(), 1 + lina, v__merger_sizes);    /*
            § 4.5 */
        updatetree(v__tre, v__merger_sizes);
    }
    assert(v__tre.back() == SAMPLE_SIZE);
    assert(vl[0] > DBL_EPSILON);
    const double d = vl[0]; std::transform (vl.cbegin(), vl.cend(), v__ri.begin(),
        v__ri.begin(), [&d](const auto &x, const auto &y)
    {
        return y + (x/d);
    }
    ); }

```

This code is used in chunk 17.

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

```

16 <approximate 16> ≡
    static void approximate() {
        std::vector < double > vri(SAMPLE_SIZE);
        vri.reserve(SAMPLE_SIZE); std::vector < double > v__l(SAMPLE_SIZE);
        v__l.reserve(SAMPLE_SIZE); std::vector < double > v__l_n(SAMPLE_SIZE + 1);
        v__l_n.reserve(SAMPLE_SIZE + 1); std::vector < std::vector < double >> a__cmfs
            (SAMPLE_SIZE + 1, std::vector < double > {} );
        allmergers(v__l_n, a__cmfs);
        int r = EXPERIMENTS + 1;
        while (--r > 0) {    /*
            § 4.6 */
            onexperiment(vri, v__l, v__l_n, a__cmfs);
        }
        double x
        {}
        ;
        for (unsigned int i = 1; i < SAMPLE_SIZE; ++i) {
            x = vri[i]/static_cast<double>(EXPERIMENTS);
            std::cout << log(x) - log1p(-x) << '\n';
        }
    }

```

This code is used in chunk 17.

¶ **main.**
 the *main* module

```

17 <includes 5>

```

```

<rngs 6> /*
    § 4.1 */
< $\lambda_{n;k_1,\dots,k_r;s}$  7> /*
    § 4.2 */
<GenPartitions 8> /*
    § 4.3 */
<allmergers_sum_m 9>
<ratesmergersfile 10> /*
    § 4.4 */
<allmergers_when_n_blocks 11>
<all possible partitions 12> /*
    § 4.5 */
<updatetree 13>
<readmergersizes 14> /*
    § 4.6 */
<onexperiment 15>
<approximate 16>
int main(int argc, const char *argv[])
{
    setup_rng(static_cast<unsigned long>(atoi(argv[1]))); /*
        § 4.6 */
    approximate();
    gsl_rng_free(rngtype);
    return GSL_SUCCESS;
}

```


5 conclusions and bibliography

We approximate $\mathbb{E}[R_i(n)]$ when the coalescent is the time-changed δ_0 -Poisson-Dirichlet($\alpha, 0$)-coalescent. An example in Figure 1

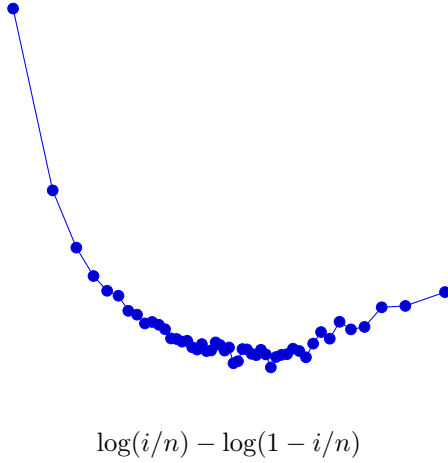


Figure 1: *An example approximation of $\mathbb{E}[R_i(n)]$ for the given parameter values and graphed as logits against $\log(i/n) - \log(1 - i/n)$ for $i = 1, 2, \dots, n - 1$ where n is sample size*

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](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

a: 7, 8.
a__cmf: 10, 11, 15.
a__cmfs: 16.
abs: 10.
accumulate: 7, 8, 13.
acmf: 12.
allmergers: 12, 16.
allmergers_sum_m: 9, 11.
allmergers_when_n_blocks: 11, 12.
ALPHA: 7.
app: 10.
approximate: 16, 17.
argc: 17.
argv: 17.
assert: 7, 8, 9, 10, 11, 13, 14, 15.
atoi: 17.
b: 8, 15.
back: 14, 15.
begin: 7, 8, 11, 13, 15.
cbegin: 15.
cend: 15.
CEPS: 7.
CKAPPA: 7.
clear: 10, 11, 13, 14.
close: 10, 14.
cmf: 10.
counts: 7.
cout: 16.
d: 7, 15.
DBL_EPSILON: 15.
descending_factorial: 7.
double: 7, 15.
end: 7, 8, 11, 13, 15.
exp: 7, 15.
EXPERIMENTS: 16.
f: 7.
FE_ALL_EXCEPT: 7.
FE_OVERFLOW: 7.
FE_UNDERFLOW: 7.
feclearexcept: 7.
fetestexcept: 7.
fill: 15.
first: 10.
FLT_MAX: 7.
GenPartitions: 8, 9.
getline: 14.
gsl_ran_exponential: 15.
gsl_rng: 6.
gsl_rng_alloc: 6.
gsl_rng_default: 6.
gsl_rng_env_setup: 6.
gsl_rng_free: 17.
gsl_rng_set: 6.
gsl_rng_type: 6.
gsl_rng_uniform: 15.
gsl_rng_uniform_pos: 15.
GSL_SUCCESS: 17.
i: 7, 10, 14, 16.
idx_Dec: 8.
idx_Last: 8.
idx_Spill: 8.
idx_SpillPrev: 8.
ifstream: 14.
indx: 11.
insert: 13.
ios: 10.
iota: 11.
istream_iterator: 14.
j: 14, 15.
k: 7, 11, 13.
l: 7.
lambdam: 15.
lambdan: 11.
lambdanks: 7, 8, 9.
LeftRemain: 8.
lgamma: 7.
lina: 15.
log: 15, 16.
log1p: 15, 16.
lrate: 8.
lrates_sorting: 8.
m: 7, 8, 9.
main: 17.
make_pair: 8, 9.
MaxVal: 8.
mergersizes: 13.
MinVal: 8.
mt19937_64: 6.
myInt: 8.
n: 7, 9, 10, 11, 14, 15.
new_time: 15.
newblocks: 13.
newtime: 15.
ofstream: 10.
onexperiment: 15, 16.
open: 10.
p: 7.
pair: 8, 9, 10, 11.
partition: 8.
PartitionSize: 8.
pow: 7.
push_back: 8, 9, 10, 13.
r: 7, 16.
random_device: 6.

randomseed: 6.
ratesmergersfile: 10, 11.
ratetosort: 11.
rbegin: 13.
readmersizes: 14, 15.
reserve: 15, 16.
resize: 13.
RHO: 15.
rng: 6, 13.
rngtype: 6, 15, 17.
s: 6, 7, 9, 10, 13.
SAMPLE_SIZE: 12, 13, 15, 16.
samplemerger: 15.
second: 7, 10.
setup_rng: 6, 17.
shuffle: 13.
size: 7, 10, 11, 13, 14, 15.
ss: 14.
stable_sort: 11.
std: 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16.
string: 14.
stringstream: 14.
sumr: 9.
sumrates: 8.
T: 6.
t: 15.
tau: 15.
tgamma: 7.
tmpn: 12.
to_string: 10, 14.
transform: 15.
tre: 13.
u: 15.
unordered_map: 7.
updatelengths: 15.
updatetree: 13, 15.
v__cmf: 15.
v__indx: 10.
v__l: 16.
v__l_k: 9.
v__lambda__n: 15.
v__lambda_n: 11.
v__m: 9.
v__mergers: 14.
v__ri: 15.
v__tre: 15.
v_k: 7.
v_l_k: 8.
v_l_n: 16.
v_lrates_sort: 9.
v_merger_sizes: 15.
val_Dec: 8.
vector: 7, 8, 9, 10, 11, 12, 13, 14, 15, 16.
veldi: 7.
vl: 15.
vlk: 10, 11.
vlmn: 12.
vri: 16.
x: 7, 10, 11, 15, 16.
y: 7, 11, 15.

List of Refinements

$\langle \lambda_{n;k_1,\dots,k_r;s} \ 7 \rangle$ Used in chunk 17.
 $\langle \text{GenPartitions} \ 8 \rangle$ Used in chunk 17.
 $\langle \text{all possible partitions} \ 12 \rangle$ Used in chunk 17.
 $\langle \text{allmergers_sum_m} \ 9 \rangle$ Used in chunk 17.
 $\langle \text{allmergers_when_n_blocks} \ 11 \rangle$ Used in chunk 17.
 $\langle \text{approximate} \ 16 \rangle$ Used in chunk 17.
 $\langle \text{includes} \ 5 \rangle$ Used in chunk 17.
 $\langle \text{onexperiment} \ 15 \rangle$ Used in chunk 17.
 $\langle \text{ratesmergersfile} \ 10 \rangle$ Used in chunk 17.
 $\langle \text{readmergersizes} \ 14 \rangle$ Used in chunk 17.
 $\langle \text{rngs} \ 6 \rangle$ Used in chunk 17.
 $\langle \text{updatetree} \ 13 \rangle$ Used in chunk 17.