Bioinformatics, 35(6), 2019, 1002–1008
doi: 10.1093/bioinformatics/bty725
Advance Access Publication Date: 28 August 2018
Original Paper

OXFORD

Genetics and population analysis

Efficient computation of the kinship coefficients

Brent Kirkpatrick[1,*], Shufei Ge[2] and Liangliang Wang[2,*]

[1]Intrepid Net Computing, Dillon, MT 59725, USA and [2]Department of Statistics and Actuarial Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada

*To whom correspondence should be addressed.

Associate Editor: Oliver Stegle

Received on October 31, 2017; revised on August 9, 2018; editorial decision on August 19, 2018; accepted on August 22, 2018

Abstract

Motivation: For families, kinship coefficients are quantifications of the amount of genetic sharing between a pair of individuals. These coefficients are critical for understanding the breeding habits and genetic diversity of diploid populations. Historically, computations of the inbreeding coefficient were used to prohibit inbred marriages and prohibit breeding of some pairs of pedigree animals. Such prohibitions foster genetic diversity and help prevent recessive Mendelian disease at a population level.

Results: This paper gives the fastest known algorithms for computing the kinship coefficient of a set of individuals with a known pedigree, especially for large pedigrees. These algorithms outperform existing methods. In addition, the algorithms given here consider the possibility that the founders of the known pedigree may themselves be inbred and compute the appropriate inbreeding-adjusted kinship coefficients, which has not been addressed in literature. The exact kinship algorithm has running-time $O(n^2)$ for an n-individual pedigree. The recursive-cut exact kinship algorithm has running time $O(s^2m)$ where s is the number of individuals in the largest segment of the pedigree and m is the number of cuts. The approximate algorithm has running-time $O(nd)$ for an n-individual pedigree on which to estimate the kinship coefficients of $\sqrt{n}$ individuals of interest from $\sqrt{n}$ founder kinship coefficients and d is the number of samples.

Availability and implementation: The above polynomial-time exact algorithm and the linear-time approximation algorithms are implemented as PedKin in C++ and are available under the GNU GPL v2.0 open source license. The PedKin source code is available at: http://www.intrepidnetcomputing.com/research/code/.

Contact: bbkirk@intrepidnetcomputing.com or lwa68@sfu.ca

1 Introduction

Computing the kinship coefficients is fundamental to understanding breeding relationships and human genealogies (Thompson, 1985). The kinship coefficients have been used for correcting GWAS case–control studies for known relationships and for reducing spurious signals in case–control and quantitative trait association studies (Eu-Ahsunthornwattana *et al.*, 2014; Kang *et al.*, 2010; Rakovski and Stram, 2009; Yu *et al.*, 2006). They have also been used for pedigree case–control disease association (Thornton and McPeek, 2007). The legal permissibility of marriages can be defined using kinship coefficients that prevent inbreeding. When breeding pedigree animals such as dogs, cats, horses, cows or endangered species, using kinship coefficients to prevent inbreeding would foster the genetic diversity. This genetic diversity is important, because it prevents excessive homozygosity and the raise of recessive Mendelian disease in the population as a whole (Woods *et al.*, 2006). There is some confusion as to how quickly kinship coefficients can be computed (Abney, 2009). The kinship coefficients are defined by counting paths of genetic transmission of alleles. The popular recursive algorithm described in Karigl (1981) was implemented in Zheng and Bourgain (2009).

There are two ways to obtain kinship coefficients. One is the empirical kinship coefficient estimated using genetic data without knowing the pedigree graph, and the other is based on defining the

© The Author(s) 2018. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

kinship coefficient as a function of a pedigree graph. The topic of estimation from data is outside the scope of this paper. This paper focuses on computing and approximating the kinship coefficient as a function of a pedigree graph.

While there exists a nice set of recursive equations, the algorithmic properties have needed more treatment in the literature. This paper aims to fill this gap by introducing three fastest known kinship algorithms: two exact algorithms and one approximate algorithm. Our numerical studies show that our proposed algorithms are several orders of magnitude faster than Zheng and Bourgain (2009) and faster than Abney (2009) when the number of individuals per generation is large.

2 Background

The pedigree graph, $P = (V, E)$, is the canonical descriptor of known relationships. This is a directed acyclic graph with individuals as vertices, V, and each vertex having a gender, either male or female. The graph has edges, E, directed from parent to child indicating direct relationship. The graph is acyclic, meaning that no individual can be their own ancestor. The graph has in-degree at most two with one parent of each gender, meaning that each individual has at most two parents, one of each gender. Let $I \subseteq V$ be the *individuals of interest*, or the individuals whose relationships we would like to quantify. Let there be n individuals in the pedigree, meaning that $n = |V|$.

A pedigree graph is a complete description of relationships between individuals, because it includes every parent–child edge. Many interesting quantifications of pedigree relationships are NP-complete to compute. Most crucially, the pedigree likelihood (Kirkpatrick, 2011; Piccolboni and Gusfield, 2003), which is the basis of many other pedigree calculations, is NP-complete. This means that the pedigree and its quantities are inefficient. We now turn our attention to more efficient representations.

There are a variety of ways to summarize the relationships in a pedigree in condensed forms. One can look at a pair-wise pedigree relationships and describe those using the language of family relationships: parent, child, nephew, cousin, etc. One can look at pair-wise genetic relationships and ask whether two alleles are inherited from an identical allele in an ancestor, i.e. whether the alleles are identical-by-descent (IBD). One can look at the k-wise relationships describing the probability of $2k$ alleles being IBD which is *generalized kinship coefficient* (Abney, 2009). One can also summarize a pair-wise relationship by the probability of two random alleles being IBD; this is the kinship coefficient, sometimes called the coefficient of relationship. This last summary of relationships, called the *kinship coefficient*, is the subject of this paper. This paper discusses the kinship coefficient, as it is defined, as a function of a pedigree graph.

Formally, for two individuals in a pedigree, define *identity-by-descent (IBD)* as the event that both individuals inherited an allele from the same ancestor. For two individuals i and j, there are four alleles, two for individual i, a_1 and a_2 and two for individual j, b_1 and b_2. In complete generality, there are 15 ways for these 4 alleles to be IBD (Jacquard, 1972). We draw these 15 possibilities as graphs on 4 alleles, as in Figure 1. There is an edge between every pair of alleles that is IBD. For example, allele a_1 and a_2 can be IBD while none of the other alleles are IBD, see row 3 of Figure 1. As another example, alleles a_1, b_1 and b_2 could be IBD while allele a_2 is not related to the others, see row 7 of Figure 1. Mathematically, the *kinship coefficient* of two individuals i and j in a pedigree is the probability of IBD between two randomly drawn alleles, one from each person. Let the matrix ϕ contain all the pair-wise kinship

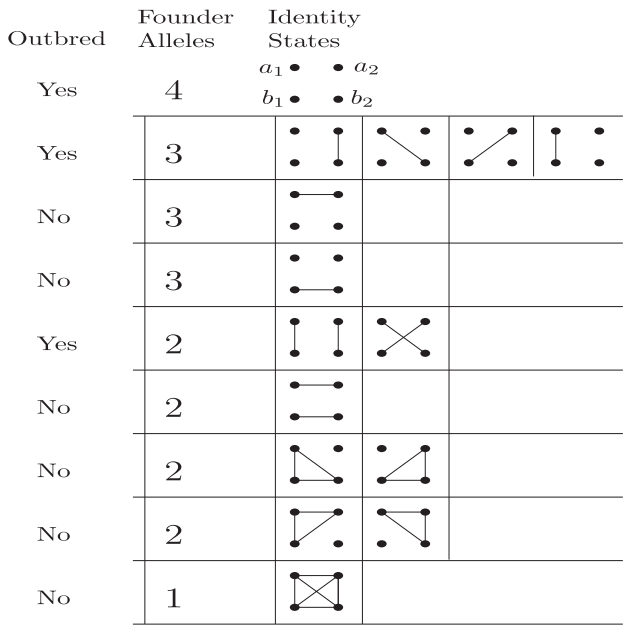

Fig. 1. Identity states. Each identity state is a graph with four alleles of two individuals drawn as the nodes and edges appearing between every pair of IBD alleles. The 15 identity states are grouped so that each row corresponds to 1 of the 9 condensed identity states. The number of founder alleles for each identity state is listed, along with whether the identity state is out-bred

coefficients in a pedigree. Entries ϕ_{ij}, for $i \neq j$, are kinship coefficients, and entries ϕ_{ii} are related to inbreeding coefficients. Label the alleles of individuals i and j with distinct labels: (a_1, a_2) and (b_1, b_2), respectively. By the definition of the kinship coefficient,

$$\phi_{ij} = (1/4)Pr[a_1 \equiv b_1] + (1/4)Pr[a_1 \equiv b_2] + (1/4)Pr[a_2 \equiv b_1] + (1/4)Pr[a_2 \equiv b_2],$$

where this '$\equiv$' operator means IBD. Unlike the IBD probabilities which can apply to specific alleles in data, the kinship coefficients are an expectation over the structure of the pedigree and are independent of the data. The kinship coefficients are often defined as an expectation over all possible inheritance paths (or all possible joint assignments to the segregation indicators of a pedigree), this paper will develop the equivalence with an expectation over the identity states and their probabilities.

Suppose that we consider only out-bred IBD possibilities (defined as having $\phi_{ii} = 1/2$ for all i). The identity states encode all the possibilities for IBD, including inbreeding possibilities, and Figure 1 has three rows indicating they are out-bred (rows 1, 2, 5). These correspond to the a pair of individuals, i and j sharing zero, one or two alleles IBD, respectively. In this case, we can denote the probability of each of these event as f_0^{ij}, f_1^{ij} and f_2^{ij} respectively. In this particular case, the kinship coefficients are simple to compute

$$\phi_{ij} = (2f_2^{ij} + f_1^{ij})/4.$$

Assume that there might be inbreeding in the pedigree. Now, we wish to compute all the kinship coefficients in one computation. The recursive algorithm for computing this was developed in detail in Thompson (1985) and Karigl (1981). This section will only give the recursive equations which are a top-down recursion on the pedigree. For all founders f, the matrix is initialized as

$$\phi_{ff} = 1/2, \text{ and}$$
$$\phi_{fj} = 0, \text{ for any } j \text{ that is not a descendant of } f.$$

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

Let the mother and father of i be denoted m and p. Then use the kinship coefficient between j and i's parents to compute

$$\phi_{ij} = (\phi_{mj} + \phi_{pj})/2,$$

where i is not an ancestor of j and $i \neq j$.

In a similar manner, we compute the kinship coefficient for i from its parents,

$$\phi_{ii} = (1 + \phi_{mp})/2,$$

where this function of ϕ_{mp} accounts for the probability of picking the same allele when uniformly choosing two alleles from the same person.

Some researchers write the kinship matrix as given by the output of the above recursion. Others transform the matrix, so that it contains the inbreeding coefficients for each individual instead of the kinship coefficients. This transformation, applied only to the diagonal, is

$$\begin{aligned}\Phi_{ij} &= \phi_{ij} \text{ for all } i \neq j, \text{ and}\\ \Phi_{ii} &= 2\phi_{ii} - 1.\end{aligned}$$

We find it convenient to represent the inbreeding coefficient on the diagonal, and this is the convention used in this paper.

3 Materials and methods

Before introducing the details of the algorithms, we need to develop some mathematical methods that ease the task of algorithm development. First, the kinship recursion can be initialized in several ways. If the founders are known to be inbred, it is appropriate to initialize the recursion with that information, for more accurate computation of kinship coefficients. Second, the kinship coefficients can be represented in terms of identity state coefficients. This relationship is useful for developing a sampling algorithm that samples identity states.

3.1 Generalized initialization of the kinship recursion

We relate the kinship to inbreeding coefficients estimated for unrelated individuals. The founders of a pedigree have an ancestry that relates them, even if that ancestry is not recorded in the pedigree graph.

Let Ψ be the kinship coefficients for the founders, i.e. a matrix of $F \times F$ where F is the number of founders. We initialize the algorithm using the kinship coefficients of the founders as follows:

$$\begin{aligned}&\phi_{ff} = (1 + \Psi_{ff})/2,\\ &\phi_{fg} = \Psi_{fg} \text{ for founders } f \neq g, \text{ and}\\ &\phi_{fj} = 0, \text{ for non-founder, not a founder child,}\\ &\qquad j \text{ that is not a descendant of } f,\end{aligned} \tag{1}$$

where Ψ_{ff} is the inbreeding coefficient for founder f. Recall that the diagonal elements of Ψ are the inbreeding coefficients. The recursive equations also need to be slightly modified, by the addition of the following case for founder children c not descended from f:

$$\begin{aligned}&\phi_{fc} = \phi_{cf} = (\phi_{m(c),f} + \phi_{p(c),g})/2,\\ &\qquad \text{for founder } m(c) \text{ or } p(c) \text{ and founder } f.\end{aligned}$$

It may be difficult to obtain the kinship coefficients on the founders if their genealogy is unknown. More feasibly, we can estimate the inbreeding coefficient from the homozygosity in Leutenegger *et al.* (2003). We modify the above recursion in Equation (1) to initialize it with the average inbreeding among all the founders using the average inbreeding coefficient for founders h, $\phi_{fg} = 1/F \sum_h \Psi_{hh}$ for founders f and g, $f \neq g$. The change to the recursive equations is still relevant.

If the kinship coefficients of the founders are not properly taken into account, then the kinship will be computed assuming that the founders are out-bred. Thus, it will inaccurately represent the genetic relationships between the individuals whose ancestry contains inbreeding in the founders.

3.2 Relating identity states and the kinship coefficient

To relate the identity coefficients into kinship coefficients, we take an expectation over the identity states. Since there is a deterministic mapping between the condensed identity states and the identity states, the expectation can be written in terms of the condensed identity states. The proof for this approach takes as a first step the expression of the kinship as an expectation over inheritance paths (Kirkpatrick, 2012), which is easily converted into an expectation over identity states.

For an identity state, let $t \in \{aa, ab, bb\}$ denote an *edge type*, for example, ab indicates any edge between the alleles in two different individuals a and b, i.e. and edge between one of the nodes $\{a_1, a_2\}$ and one of the nodes $\{b_1, b_2\}$. Another example, aa, indicates an edge between nodes a_1 and a_2 within the same individual. Recalling that the identity state is a graph on nodes $\{a_1, a_2, b_1, b_2\}$, we let $e(s, t)$ be a function of identity state s and edge type t which gives the number of edges of type t in identity state s. For example, in Figure 1, row 8, column 1, $e(s, aa) = 1, e(s, ab) = 2, e(s, bb) = 0$.

The kinship coefficient $\Phi_{a,b}$ between individuals $a \neq b$ is related to the identity states and their coefficients via the following expectation over the set of 15 possible identity states (a set that we denote by $\mathcal{S}$):

$$\Phi_{a,b} = \sum_{s \in \mathcal{S}} \frac{e(s, ab)}{4} \mathbb{P}[S = s]. \tag{2}$$

In this equation, $e(s, ab)/4$ can be interpreted as the fraction of the 4 possible edges between one node in $\{a_1, a_2\}$ and one node in $\{b_1, b_2\}$ that are present in the identity state s.

The inbreeding coefficient $\Phi_{a,a}$ is computed slightly differently via:

$$\Phi_{a,a} = \sum_{s \in \mathcal{S}} e(s, aa)\, \mathbb{P}[S = s], \tag{3}$$

where $e(s, aa)$ indicates whether the single possible edge between nodes a_1 and a_2 exists.

The transformation to obtain the kinship coefficient from Equation (3) is $\phi_{a,a} = (1 + \Phi_{a,a})/2$. The convention in this paper is to use $\Phi_{a,a}$.

4 Efficient implementations

We will discuss three algorithms for obtaining kinship coefficients which are efficient in different scenarios. Recall that there are n individuals in our pedigree. The first scenario is exact computation in time polynomial to the size of the pedigree (i.e., running-time $O(n^2)$), and the second scenario is approximate computation in linear time [i.e., running-time $O(n)$].

First, the recursions given, above, can be implemented efficiently in an $O(n^2)$-time algorithm. The challenging portion of the recursive equations is the check for which individuals are ancestors of each other. This can be done in constant time, provided that the correct

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

data structure tabulates the ancestry information. The details of this algorithm are given in Section 4.1.

Second, when the individuals of interest are a subset of the individuals in the pedigree, then we can use a divide-and-conquer approach for applying the first exact algorithm. In this approach we recursively cut the pedigree graph to obtain sub-pedigrees on which to apply the full exact algorithm. As long as the individuals of interest all reside in a single sub-pedigree, this approach is more efficient than the first exact algorithm. Details are given in Section 4.2.

Third, when a linear algorithm is desired and the kinship coefficients of a subset of individuals are needed, there is a sampling algorithm that samples identity states. Given sample size d and the kinship of $\sqrt{n}$ founders, the sampling algorithm works in $O(nd)$ time and the kinship of $\sqrt{n}$ of the individuals is computed. A smaller version of this sampling algorithm that estimates the kinship of one pair of individuals has previously been introduced (Sun *et al.*, 2014). That algorithm was evaluated for sampling error, and it was discovered that several thousand identity states are needed for very large pedigrees. Details of this algorithm appear in Section 4.3.

4.1 Efficient, exact algorithm

There is an $O(n^2)$-time implementation of the kinship calculation where n is the number of individuals in the pedigree, and $I = V$. Note that any implementation that touches every cell of the kinship matrix requires running-time at least $O(n^2)$. Any implementation of the kinship that represents the full kinship matrix requires $O(n^2)$ space.

We define A_i, the *ancestor set* for individual i in the pedigree, as the set containing i and all its ancestors. This object can easily be computed in $O(n^2)$ time. The ancestor set allows us to do the ancestry check in the kinship recursion in constant time.

The ancestor sets are computed in a top-down recursion on the pedigree graph. Initialize the recursion with $A_f = \{f\}$, where $f \in V$ is a founder in the pedigree, meaning the individual has no parents.

Now, the remaining individuals' ancestor sets are computed from their parents' using $A_i = \{i\} \cup A_{m(i)} \cup A_{f(i)}$, where $m(i)$ is the mother of i and $f(i)$ is the father of i.

The top-down order is obtained by creating a topological sort of the graph. This is done by creating a queue of individuals, initializing the queue with the founders, and popping an individual from the front of the queue while simultaneously adding their children to the end of the queue. This produces an order such that each individual is considered after all of their parents.

The recursion from Section 2 can now be implemented in $O(n^2)$ time, using the ancestor sets. The ancestor sets can be queried quickly, if they are stored in a look-up table. This makes the recursion a double loop over the individuals in the pedigree, when those individuals are considered in the top-down order. Algorithm 1 gives the details of the recursive loops.

Algorithm 1 $O(n^2)$ Exact Kinship Algorithm

```
Let A be an n × n matrix initialized with all zeros.
for all the founders i ∈ V do
  Let A_ii = 1
end for
for each i ∈ V with parents m(i), f(i) such that their A row is set do
  for each j ∈ p(i) do
    for each v ∈ V do
      A_iv = A_m(i)v AND A_f(i)v
      if v == i then
        A_ii = 1
      end if
    end for
  end for
end for
Let Φ be an n × n matrix initialized with –1.
for every founder f ∈ V do
  Φ_f,f = (1 + Ψ_ff)/2
end for
for every pair of founders f, g ∈ V with f ≠ g do
  Φ_f,g = Ψ_fg
end for
for every founder f and every non-founder j ∈ V, j not founder child do
  if A_jf == 0 then
    Φ_fj = 0
  end if
end for
for every i ∈ V whose parents have been assigned kinship do
  for every j ∈ V whose parents have been assigned kinship do
    if i == j then
      Φ_ii = (1 + Φ_mp)/2
    else
      if A_ji == 0 then
        Φ_ij = (Φ_mj + Φ_pj)/2
      end if
    end if
  end for
end for
for every i ∈ V do
  Φ_ii = (2 * Φ_ii) − 1
end for
```

4.2 Faster, recursive-cut exact algorithm

In the case where we require the kinship coefficients of a subset of the pedigree individuals, $I \subset V$, we can improve the running-time for the exact calculation. Recall from Section 3.1, that a correct kinship computation can be done after initializing the founders of a pedigree with their known kinship coefficients.

Consider a large pedigree, which is a directed acyclic graph with the founders as sources and the leaves as sinks. Informally, we can segment the large graph by taking vertex cuts that run 'horizontally' and create a generation. The individuals in the cut become the leaves of the upper segment and the founders of the lower segment of the pedigree. This splits the pedigree in half at the cut, and allows us to apply the exact algorithm from Section 4.1 to each segment of the pedigree.

Formally, a *vertex cut* is defined as a partition of the vertices of the graph into two sets such that the *cut vertices*, when removed, disrupt every path from any source to any sink. A vertex cut produces two edge-disjoint subgraphs. In our application the sub-pedigrees will share the vertices of the cut set, but will be edge-disjoint. In one sub-pedigree, the cut vertices will be the leaves, and in the other sub-pedigree, the cut vertices will be the founders.

A *pedigree cut* for the purposes of this algorithm will be defined as a set of cut vertices which defines a generation of individuals.

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

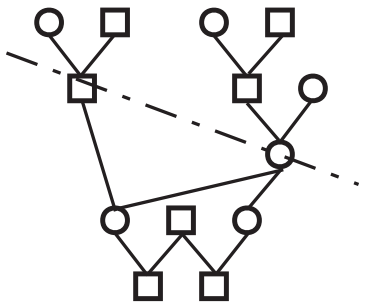

Fig. 2. Recursive cut. The two individuals through which the dash-dotted line passes are in the cut set. The older sub-pedigree has those two individuals as leaves and has two components. The younger pedigree has those two individuals as founders

Any pedigree cut separates the pedigree into two sub-pedigrees, one containing the cut vertices as leafs which we will refer to as the upper, or older, sub-pedigree, and one containing the cut vertices as founders which we will refer to as the lower, or younger, sub-pedigree. In Figure 2, the two individuals through which the dash-dotted line passes are in the cut set. The upper sub-pedigree, by the placement of the pedigree cut, now has new leaf individuals, which are the cut vertices. Let the cut set be set $C^i \subseteq V$. Let the older sub-pedigree be for individuals V^i and the younger sub-pedigree be for individuals V^{i+1} In this case, we know that $C^i \subseteq V^i$ and $C^i \subseteq V^{i+1}$.

We can recursively bipartition the pedigree and sub-pedigrees many times to get a reasonable running time of the exact kinship algorithm on any single sub-pedigree. The finest recursive partitioning will produce generational sub-pedigrees, but with the flexibility of definitions to allow staggered generations. The number of generations puts a lower bound on the number of recursive cuts that are possible. Also, the individuals of interest all need to appear in a single sub-pedigree, if we are to obtain all their pair-wise kinship coefficients from this algorithm.

Now, we apply the exact kinship algorithm for the sub-pedigrees from the top down. The kinship coefficients for the leafs of each sub-pedigree are used to initialize the founder kinship coefficients for the next sub-pedigree. This is done iteratively down the pedigree, until the kinship coefficients of the individuals of interest are obtained.

When applied to a generational pedigree drawn using the diploid Wright-Fisher model, the pedigree has $|C^i| = 2N$ individuals per generation for all generations $0 \leq i \leq G$ and $V^i = C^i$. This recursive-cut kinship algorithm runs on this generational pedigree in $O(N^2G)$ time. More generally if we cut an arbitrary pedigree into m partitions with the maximum partition having $s = |V^i|$ individuals, then the running-time of the recursive-cut kinship algorithm is $O(s^2m)$. When n is the number of individuals in the pedigree, this is an improvement on the $O(n^2)$ running-time of the exact method, since $s \leq n/m$.

The main disadvantage of both these exact algorithms is that they are only polynomial in running time and perhaps may not be fast enough for applications on very large pedigrees. For very large pedigrees, we need a linear-time algorithm for scalability.

4.3 Efficient, approximate algorithm

There is a linear-time approximate algorithm. Recall that n is the number of individuals in the pedigree. The approximate algorithm runs in $O(nd)$-time algorithm for $I \subset V$ where $|I| = O(\sqrt{n})$, $F = O(\sqrt{n})$ for number of founders and d is the sample size.

Algorithm 2 O(nd) Approximate Kinship Algorithm when $|V| = n$, $|I| = O(\sqrt{n})$, $F = O(\sqrt{n})$ and d is the number of samples.

Let Φ be the n × n matrix initialized with zeros.
for s in 1 to d **do**
  for $i \in V$ **do**
    Flip 2 {m, f}-labeled coins, and assign their values to (x_i^m, x_i^f).
end for
  Initialize all $c_i = (0, 0)$; Let counter = 1.
  for $l \in V$ where l is a leaf **do**
    Let $(c_i^m, c_i^f) = (counter, counter + 1)$
    $counter = counter + 2$
  end for
  for each $i \in V$ provided all of i's children have been processed **do**
    for $p \in \{m, f\}$ and $j \in V$ being the appropriate gendered parent **do**
      if $c_j^{x_i^p} < c_i^p$ **then**
        $c_j^{x_i^p} = c_i^p$
      end if
  end for
  end for
for every pair of founders r and q **do**
  flip a coin for edges $(a_1, b_1)(a_2, b_2)$ or $(a_1, b_2)(a_2, b_1)$, a = 0 or a = 1, respectively
  for $x \in \{m, f\}$, and let $y \in \{m, f\} \setminus x$ **do**
    Let $u \in [0, 1]$ be a uniform random variable
    if $u \leq \Psi_{rq}$ **then**
      if $a == 0$ **then**
        $c_r^x = c_q^x$
      else
        $c_r^x = c_q^y$
      end if
    end if
  end for
end for
for each $i \in V$ provided all i's parents have been processed, denote ip as i's parent **do**
    $c_i^p = c_{i_p}^{x_i^p}$
end for
for each $i \in I$ **do**
  for each $j \in I$ **do**
    Create the identity state graph for $(c_i, c_j) = (c_i^m, c_i^f, c_j^m, c_j^f)$
      if $i == j$ **then**
        if $c_i^m == c_i^f$ **then**
          $\Phi_{ii} = \Phi_{ii} + \frac{1}{d}$
        end if
      else
        $\Phi_{ij} = \Phi_{ij} + \frac{e}{4d}$, where e is the number edges between c^i and c^j in the identity state graph.
      end if
    end for
  end for
end for

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

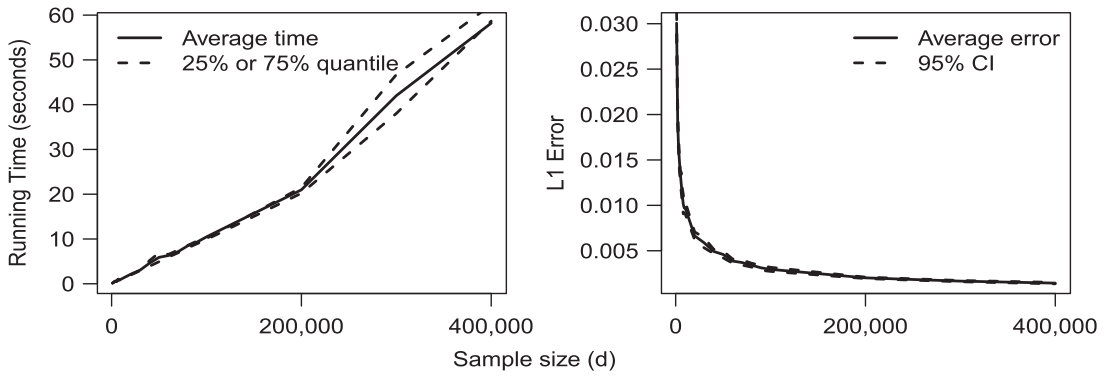

Fig. 3. Simulation results of the approximate algorithm. The left panel represents the relationship between of running time (seconds) and number of iterations and the right panel represents the relationship between L_1 error and number of iterations

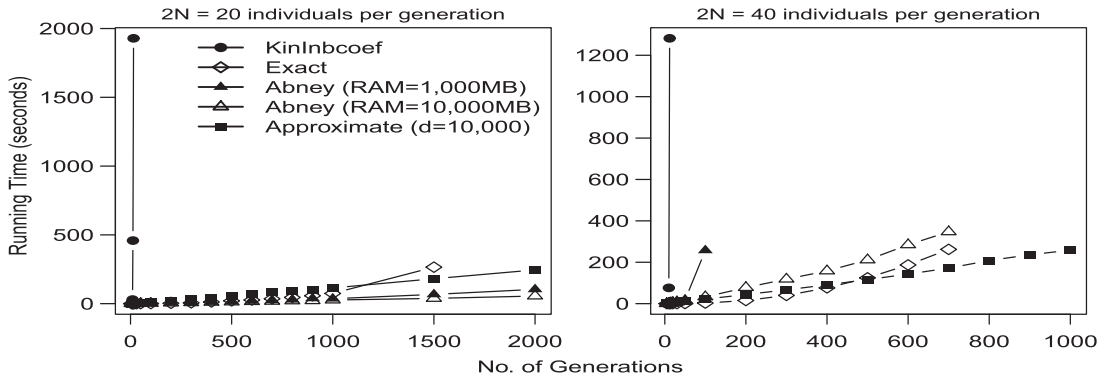

Fig. 4. Comparison of running time. two panels describe the relationship between the running time (in seconds) and the number of generations G

This algorithm quickly estimates the kinship coefficients by sampling identity states and using the expectations in Equations (2) and (3).

Recall that in diploid individuals, each individual has two alleles at every site in the genome. Of these two alleles, one comes from the father, and one from the mother. From each parent, the allele is copied either from the grand-father or from the grand-mother. This binary choice of grand-paternal origin is usually stored in *segregation* indicators, $x_i = (x_i^m, x_i^f)$, for individual i where $x_i^p \in \{m, f\}$. An inheritance path consists of the segregation indicators of all the individuals in the pedigree.

Consider the graph of all the alleles of all the individuals at one site with edges connecting the alleles that are inherited from parent to child (i.e., the edges indicated by the segregation indicators). This is a graph of the inheritance path, and it has connected components. The connected component is the set of inherited copies of a particular ancestral allele, which is the root of the connected component. Let the *Connected Component (CC) membership* of each allele be given by a tuple of integers, $c_i = (c_i^m, c_i^f)$, for individual i.

Our method, Algorithm 2, will sample an inheritance path with one pass through the pedigree. When considering each individual, the algorithm flips a coin to set the segregation indicators for that individual. Later, these indicators will be used to determine which alleles are identical-by-descent for this inheritance path.

Two more passes through the pedigree will be used to set the CC membership with consistent values. The leaf alleles of the pedigree are populated with distinct integers representing their putative CC membership. The first pass proceeds from the bottom of the pedigree to the top, and processes each child before their parents. For each allele in each individual, their CC membership integer is copied to the allelic ancestor given by the segregation indicator. If two children give two different CC memberships to the parent, the tie is broken with the largest integer value. To account for founder inbreeding, we merge founder CC memberships randomly according to the probabilities given by the initialization kinship coefficients. The second pass proceeds from top to bottom, and the tie-broken CC membership integers are simply copied back down the path of allelic inheritance. After these two passes, every allele in a connected component of the inheritance path graph will have the same CC membership, and every pair of alleles from different CC's will have distinct CC membership unless the CC's were randomly merged by founder inbreeding.

Finally, for every pair of individuals of interest, we can use the CC membership to obtain the identity state. This identity state can be used to update the estimated kinship with the appropriate terms from Equations 2 and 3.

The details of these procedures are left to Algorithm 2. We will explore the convergence properties of this sampling algorithm using simulations in Section 4.4.

4.4 Simulation results

The diploid Wright-Fisher model is used to generate pedigree data in our simulation study. To assess the convergence of our approximate kinship algorithm, we simulate a scenario with $2N = 20$ individuals per generation, and $G = 10$ generations in total. The sample size d ranges from 1000 to 400 000. We measure the error of estimates in the L_1 form. The left panel in Figure 3 shows that the running time increases almost linearly with number of iterations. The right panel

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021

in Figure 3 indicates that L_1 form error decreases rapidly before the sample size reaches 3000. The L_1 error is 1.6×10^{-2} when the sample size $d = 3000$ and keeps decreasing with the the sample size d increases.

KinInbcoef (Zheng and Bourgain, 2009) is a C++ program that computes inbreeding and kinship coefficients for general pedigrees using the recursion algorithm mentioned in Section 2, IdCoefs (Abney, 2009) is a C program that computes the generalized kinship coefficients and condensed identity coefficients. Both programs will be used as the competitors in following simulations.

To compare the running time of our exact algorithm and approximate kinship algorithm with KinInbcoef and IdCoefs, we simulate two scenarios of pedigree data: one is 20 individuals per generation and another is 40 individuals per generation. We consider a sequence of numbers of generations for both scenarios. The simulations were carried out on the Grex SGI Altix XE 1300 cluster of Westgrid. The results show both our exact and approximate algorithms can handle large pedigrees efficiently.

We compare our exact algorithm with the two competitors. Figure 4 shows our exact method is several orders of magnitude faster than the KinInbcoef, and it is faster than IdCoefs when the number of individuals per generation is not too small. Shown on the right panel, IdCoefs requires a large RAM size to run efficiently. Note that its performance becomes much worse if we decrease the RAM size from 10 000 to 1000 MB. Therefore, IdCoefs cannot scale to large pedigrees.

The approximate method is not only efficient but also accurate. As shown in Figure 4, the time cost of our approximate method is linear with number of generations. We use the sample size $d = 10\,000$. It runs faster than the exact algorithm when the pedigree is larger, for example, when ($2N = 20$, $G > 1000$) and ($2N = 40$, $G > 400$). The corresponding accuracy of the estimate is to 3 decimal digits.

5 Discussion

The above kinship algorithms are applicable to large pedigrees, since they either have efficient polynomial or linear running-times. This paper gives two exact algorithms and one approximation algorithm. The fastest known exact algorithm for kinship computations is the recursive-cut exact algorithm. All these algorithms are easily run using the source code that is published in tandem with this paper. All of the algorithms in this paper have running times that are parameterized by the number of individuals. Since the kinship coefficients are defined by inheritance paths which correspond to edges in the pedigree graph, it is possible that the number of edges in a pedigree graph provides the true lower-bound on the number of operations needed to compute the exact kinship coefficients. We leave it as an open problem whether there is an efficient algorithm, perhaps along the lines of the recursive-cut algorithm, that has running-time parameterized by the number of edges in the pedigree graph. Another open problem is whether there is a sparse algorithm for computing the kinship. This type of algorithm would only compute the non-zero entries in the kinship matrix. If designed properly, such an algorithm would need far less space, since it would not need to represent the entire kinship matrix. We believe that such an algorithm does exist and has yet to be discovered.

Funding

This work was supported by funding from the Natural Sciences and Engineering Research Council of Canada (Discovery Grant 435713-2013-RGPIN). Research was enabled in part by support provided by Compute Canada (www.computecanada.ca).

Conflict of Interest: none declared.

References

Abney,M. (2009) A graphical algorithm for fast computation of identity coefficients and generalized kinship coefficients. *Bioinformatics*, **25**, 1561–1563.

Eu-Ahsunthornwattana,J. *et al*. (2014) Comparison of methods to account for relatedness in genome-wide association studies with family-based data. *PLoS Genet*., **10**, e1004445.

Jacquard,A. (1972) Genetic information given by a relative. *Biometrics*, **28**, 1101–1114.

Kang,H.M. *et al*. (2010) Variance component model to account for sample structure in genome-wide association studies. *Nat. Genet*., **42**, 348–354.

Karigl,G. (1981) A recursive algorithm for the calculation of identity coefficients. *Ann. Hum. Genet*., **45**, 299.

Kirkpatrick,B. (2011) Haplotype versus genotypes on pedigrees. *Algorithms Mol. Biol*., **6**, 10.

Kirkpatrick,B. (2012) Non-identifiable pedigrees and a bayesian solution. *Int. Symp. Bioinformatics Res. Appl. (ISBRA)*, 139–152.

Leutenegger,A.-L. *et al*. (2003) Estimation of the inbreeding coefficient through use of genomic data. *Am. J. Hum. Genet*., **73**, 516–523.

Piccolboni,A. and Gusfield,D. (2003) On the complexity of fundamental computational problems in pedigree analysis. *J. Comput. Biol*., **10**, 763–773.

Rakovski,C.S. and Stram,D.O. (2009) A kinship-based modification of the armitage trend test to address hidden population structure and small differential genotyping errors. *PLoS ONE*, **4**, e5825.

Sun,D. *et al*. (2014) Kinship accuracy: comparing algorithms for large pedigrees. *Stanford Undergraduate Res. J*., **13**, 97–102.

Thompson,E.A. (1985) *Pedigree Analysis in Human Genetics*. Johns Hopkins University Press, Baltimore.

Thornton,T. and McPeek,M. (2007) Case–control association testing with related individuals: a more powerful quasi-likelihood score test. *Am. J. Hum. Genet*., **81**, 321–337.

Woods,C.G. *et al*. (2006) Quantification of homozygosity in consanguineous individuals with autosomal recessive disease. *Am. J. Hum. Genet*., **78**, 889–896.

Yu,J. *et al*. (2006) A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. *Nat. Genet*., **38**, 203–208.

Zheng,Q. and Bourgain,C. (2009) *KinInbcoef: Calculation of Kinship and Inbreeding Coefficients*. https://www.stat.uchicago.edu/~mcpeek/software/KinInbcoef/index.html.

Downloaded from https://academic.oup.com/bioinformatics/article/35/6/1002/5085372 by Simon Fraser University user on 28 March 2021