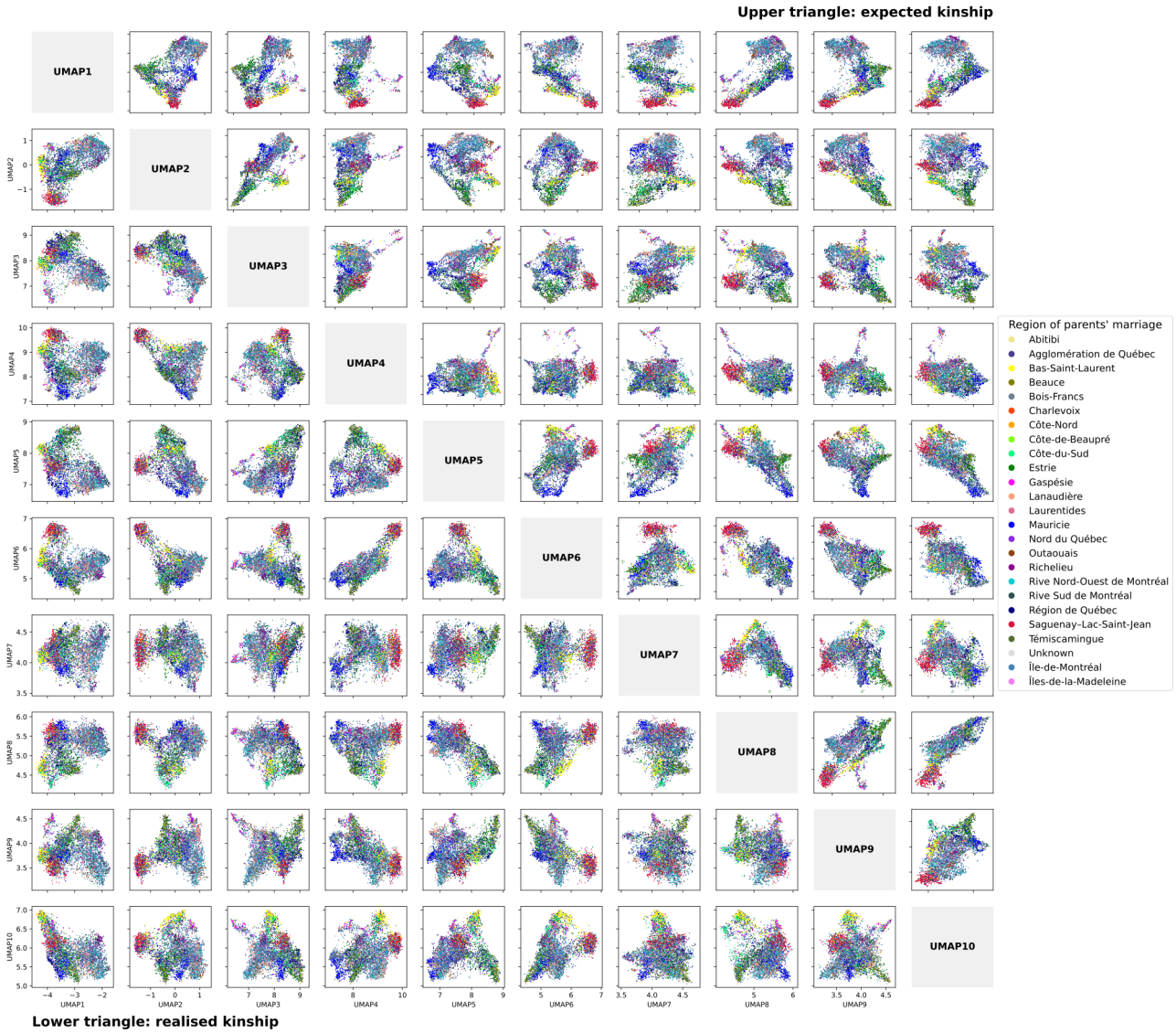
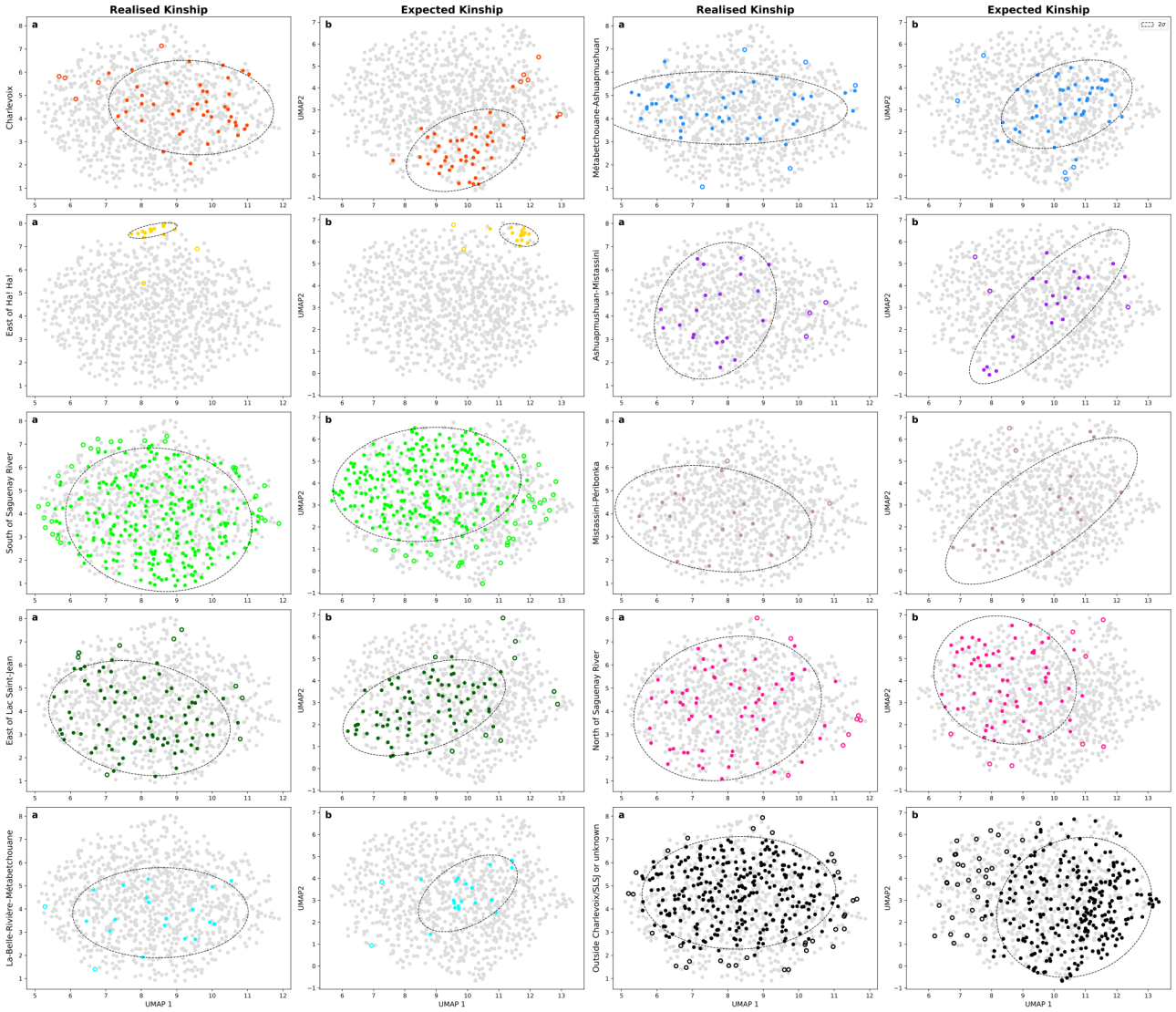
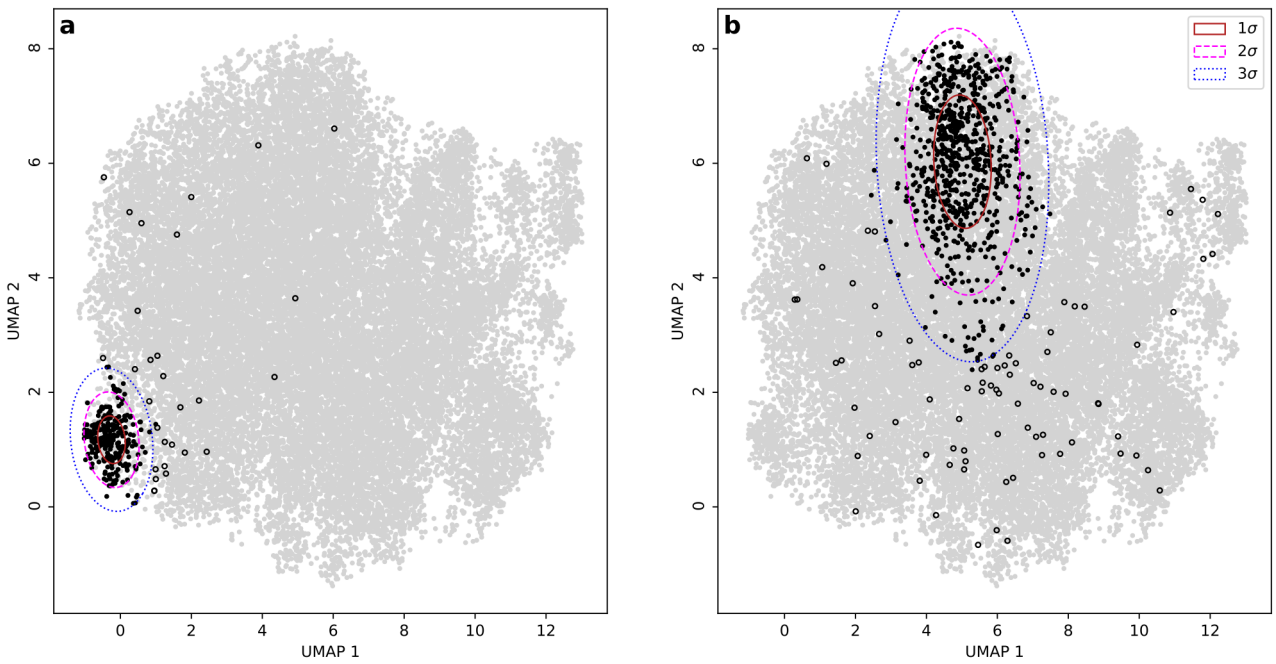
Fine-scale structure of a whole regional population through genetics and genealogies



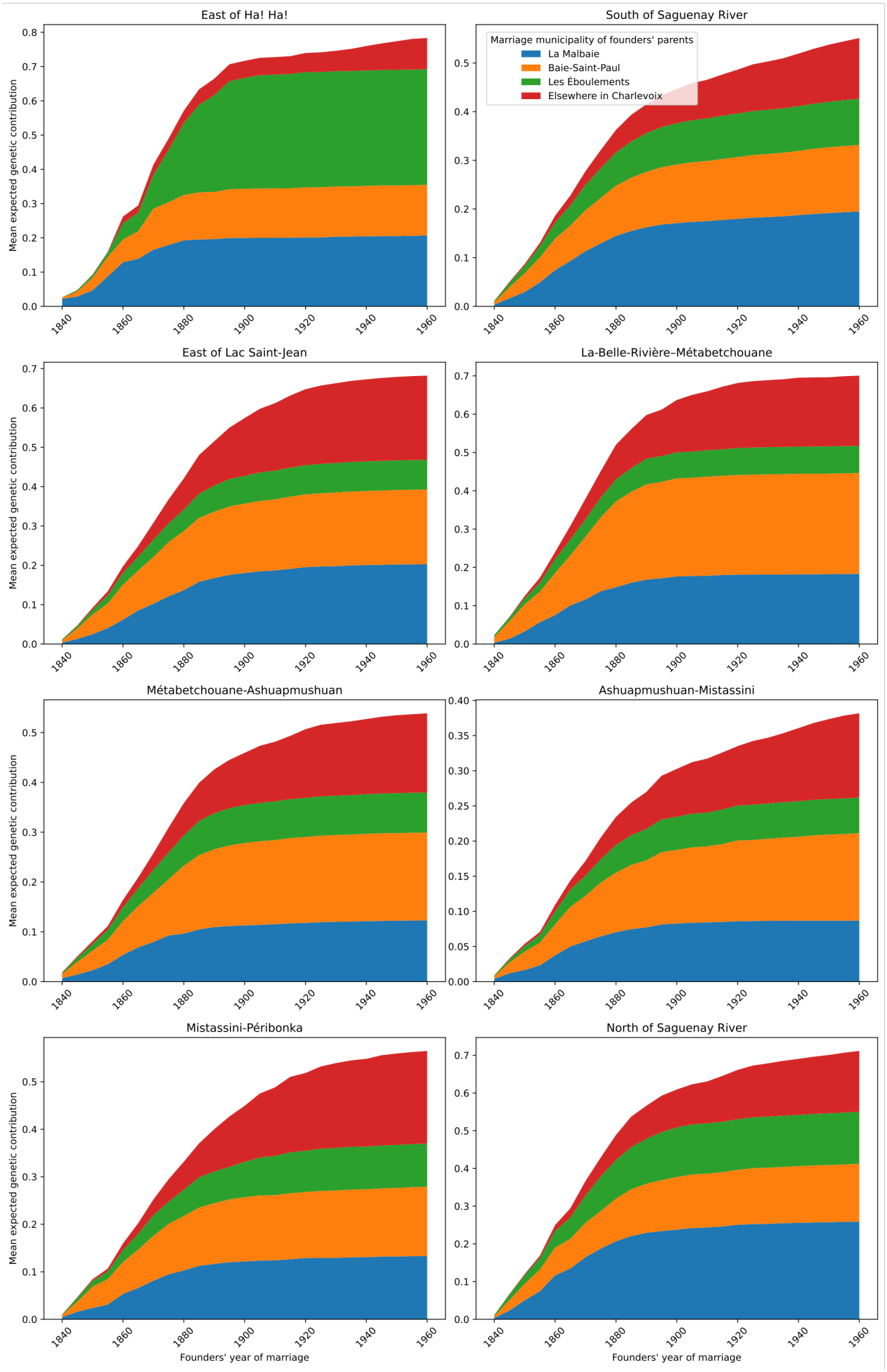
Supplementary Figure 1. **Ten-dimensional UMAP of CARTaGENE individuals based on pairwise realised (lower triangle) and expected (upper triangle) kinship.** Uniform Manifold Approximation and Projection (UMAP) projection of 7,970 individuals from the CARTaGENE cohort, computed from their (lower triangle) realised kinship and (upper triangle) expected kinship transformed as a precomputed distance ($1 - \phi$). The colours represent the region of parents' marriage.



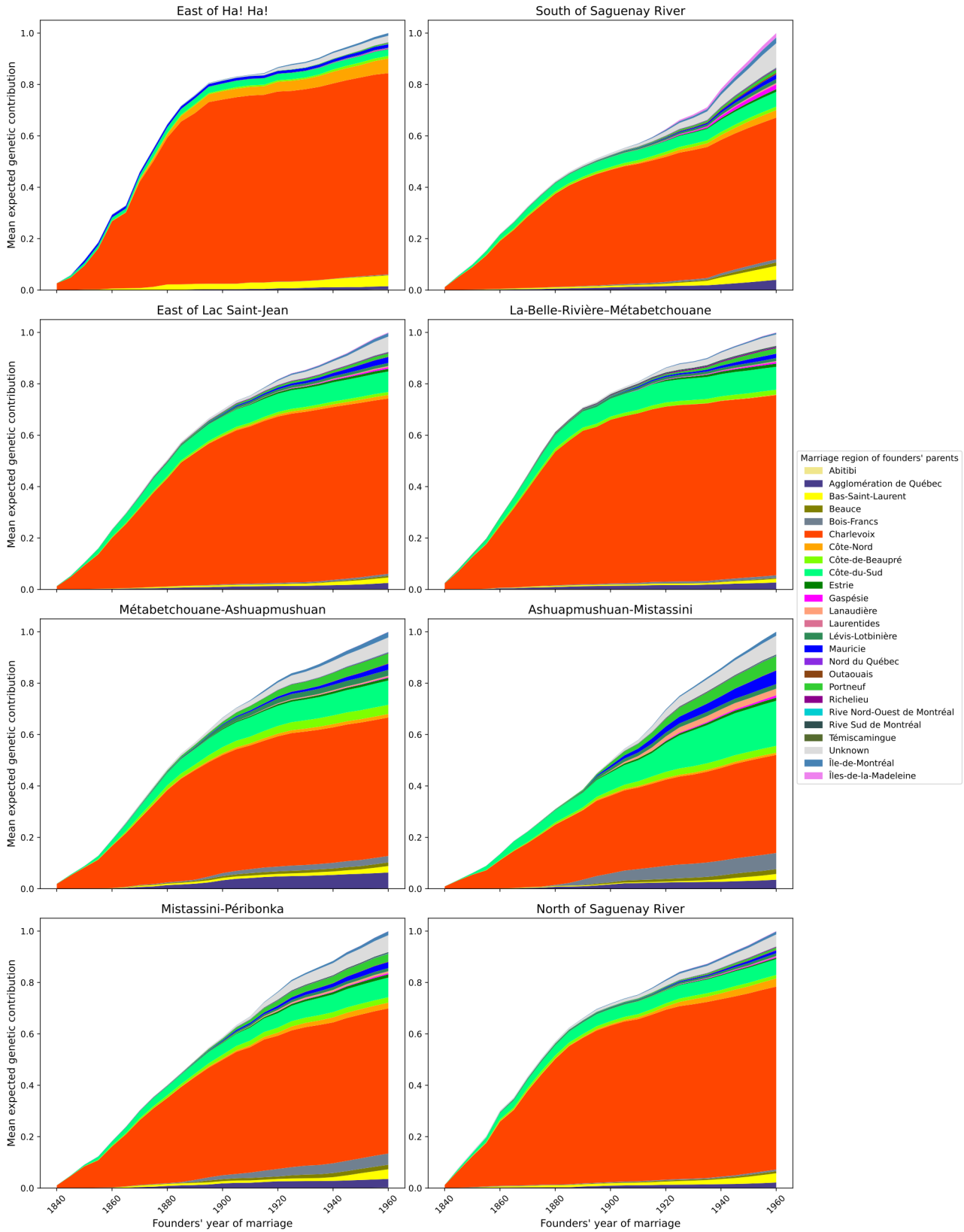
Supplementary Figure 2. **Distribution of CARTaGENE SLSJ individuals whose parents married in each watercourse subdivision.** Uniform Manifold Approximation and Projection (UMAP) projection of 938 individuals from the CARTaGENE cohort most likely originating from Saguenay–Lac-Saint-Jean (SLSJ), computed from their (**a**) realised kinship and (**b**) expected kinship transformed as a precomputed distance ($1 - \phi$). The coloured dots represent individuals whose parents married in the indicated subdivision of SLSJ. Open circles are individuals who were identified as outliers using an ellipse learned from their Gaussian distribution. The confidence ellipses of the remaining inliers have a radius of two standard deviations.



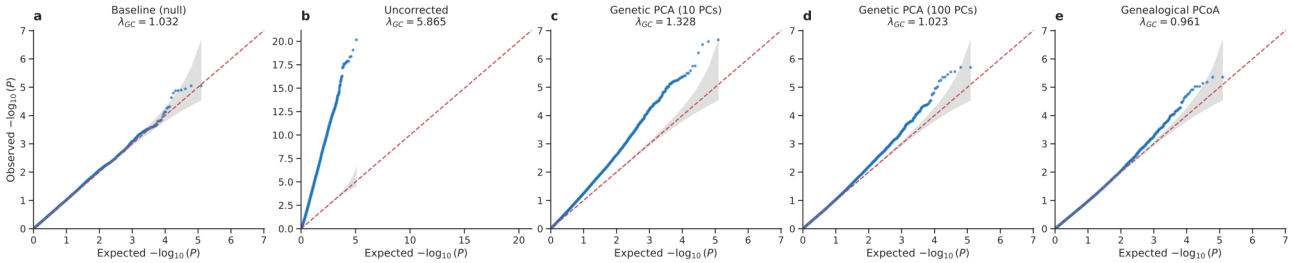
Supplementary Figure 3. **Difference in the spatial dispersion of individuals between a rural (a) and an urban (b) municipality.** Uniform Manifold Approximation and Projection (UMAP) projection of 26,445 non-siblings who married between 1931 and 1960 in Saguenay–Lac-Saint-Jean, computed from their expected kinship (ϕ) transformed as a precomputed distance ($1 - \phi$). Black dots represent individuals whose parents married in **(a)** L'Anse-Saint-Jean (n = 284); and **(b)** Alma (n = 805). Black open circles are individuals who were identified as outliers, whereas the confidence ellipses of the remaining inliers have a radius of one, two, and three standard deviations (σ).


East of Ha! Ha!
South of Saguenay River
East of Lac Saint-Jean
La-Belle-Rivière–Métabetchouane
Métabetchouane-Ashuapmushuan
Ashuapmushuan-Mistassini
Mistassini-Péribonka
North of Saguenay River
Marriage municipality of founders' parents
La Malbaie
Baie-Saint-Paul
Les Éboulements
Elsewhere in Charlevoix
Mean expected genetic contribution
Founders' year of marriage

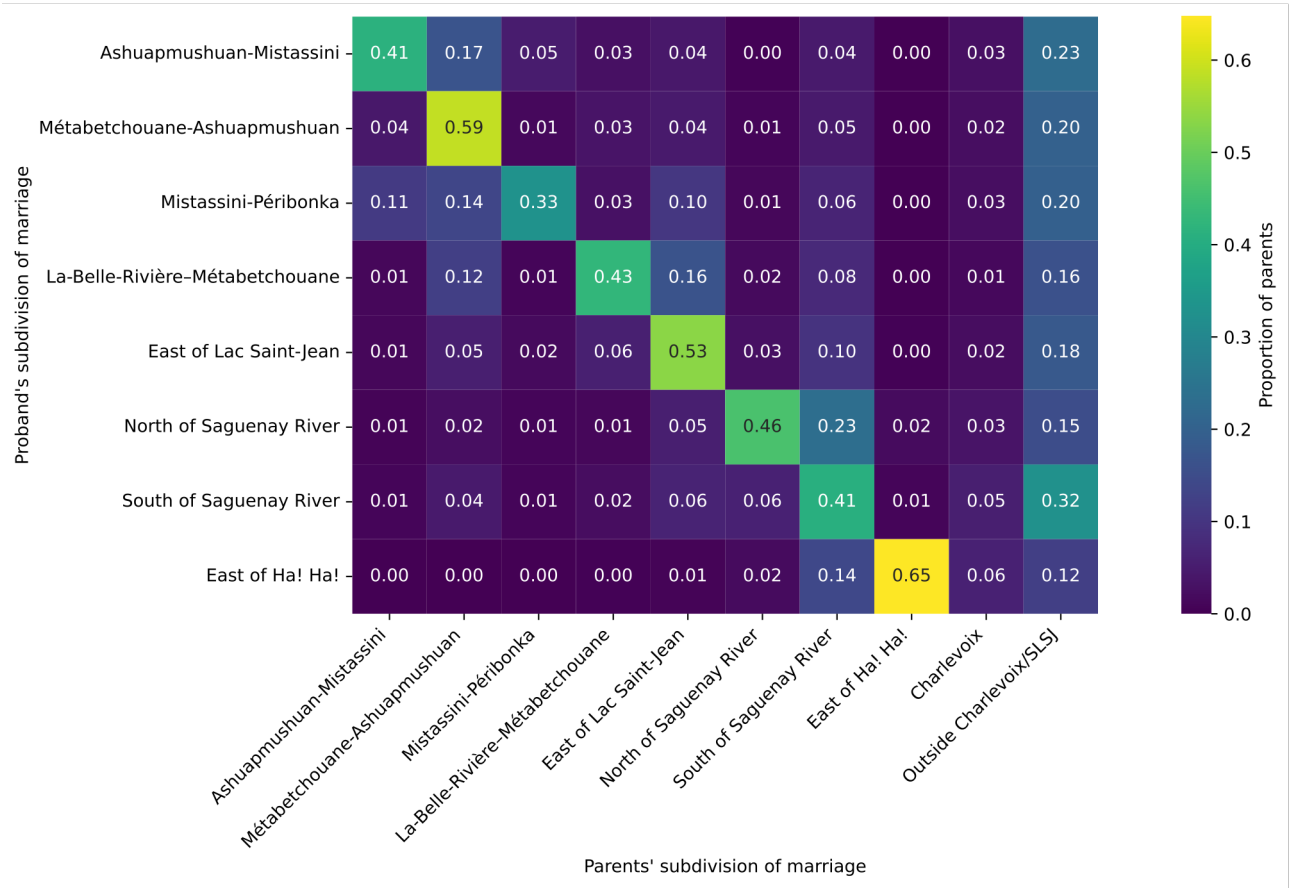
Supplementary Figure 4. **Cumulative genetic contribution of SLSJ founders originating from Charlevoix to probands in each watercourse subdivision per period of arrival.** Cumulative mean expected genetic contribution of Saguenay–Lac-Saint-Jean (SLSJ) founders per municipality of Charlevoix and period of marriage to probands (total 80,348) married in various subdivisions.



Supplementary Figure 5. **Cumulative genetic contribution of SLSJ founders originating from Quebec to probands in each watercourse subdivision per period of arrival.** Cumulative mean expected genetic contribution of Saguenay–Lac-Saint-Jean (SLSJ) founders per Quebec regions and period of marriage to probands (total 80,348) married in various SLSJ subdivisions.



Supplementary Figure 6. **PCA does not adequately control for inflation of GWAS on a simulated phenotype correlated with SLSJ fine-scale structure.** Quantile-quantile plots for simulated data (N = 10,000) across 100 Mb of neutral genomic sequence using `msprime` conditioned on the genealogy. The phenotype was simulated to be partially correlated with the expected genetic contribution (GC, computed on the genealogy) from Charlevoix founders (h^2 = 0.1) since this GC was shown to be a major factor influencing the Saguenay–Lac-Saint-Jean (SLSJ) fine structure. Panels display genome-wide association studies (GWAS) results for: **(a)** a baseline null phenotype (pure noise); **(b)** the correlated phenotype with no correction; **(c–d)** the correlated phenotype corrected using the top 10 and 100 genetic principal components (PCA) of common single nucleotide polymorphisms (minor allele frequency < 0.05); and **(e)** the correlated phenotype corrected using the top 2 coordinates from a principal coordinate analysis (PCoA) derived from the genealogical kinship matrix. The genomic inflation factor (λ_{GC}) is reported for each model to quantify test statistic inflation. The red dashed line represents the null expectation ($y = x$), and the gray shaded region indicates the 95% confidence interval. See Supplementary Methods for details.



Supplementary Figure 7. **Migration within and between subdivisions.** Proportion of parents married in each watercourse subdivision (column) for each Saguenay–Lac-Saint-Jean (SLSJ) probands' (total 26,445) subdivision of marriage (row). The sum of each row is equal to one.

Supplementary Table 1. **Municipalities of watercourse-defined subdivisions of Saguenay–Lac-Saint-Jean (SLSJ).**

Subdivision (No of municipalities)	Municipalities (East to West)
East of Ha! Ha! (4)	Petit-Saguenay, L'Anse-Saint-Jean, Rivière-Éternité, Saint-Félix-d'Otis
South of Saguenay River (9)	Ferland-et-Boilleau, La Baie, Chicoutimi, Laterrière, Arvida, Jonquière, Saint-Ambroise, Bégin, Larouche
East of Lac Saint-Jean (13)	Mont-Apica, Notre-Dame-du-Rosaire, Labrecque, Saint-Nazaire, Delisle, Saint-Bruno, Alma, Hébertville-Station, L'Ascension-de-Notre-Seigneur, Hébertville, Saint-Gédéon, Saint-Henri-de-Taillon, Sainte-Monique-de-Honfleur
La-Belle-Rivière–Métabetchouane (4)	Lac-à-la-Croix, Métabetchouan, Desbiens, Saint-André-du-Lac-Saint-Jean
Métabetchouane-Ashuapmushuan (9)	Chambord, Saint-François-de-Sales, Lac-Bouchette, Mashteuiatsh, Roberval, Saint-Prime, Sainte-Hedwidge, Saint-Félicien, La Doré
Ashuapmushuan-Mistassini (7)	Dolbeau, Saint-Méthode, Albanel, Normandin, Girardville, Saint-Edmond, Saint-Thomas-Didyme
Mistassini-Péribonka (10)	Chute-des-Passes, Saint-Ludger-de-Milot, Saint-Augustin-du-Lac-Saint-Jean, Péribonka, Sainte-Élisabeth-de-Proulx, Sainte-Jeanne-d'Arc-du-Lac-Saint-Jean, Saint-Stanislas-du-Lac-Saint-Jean, Mistassini, Saint-Eugène-du-Lac-Saint-Jean, Notre-Dame-de-Lorette
North of Saguenay River (7)	Sainte-Rose-du-Nord, Saint-Fulgence, Chicoutimi-Nord, Saint-Honoré-de-Chicoutimi, Saint-David-de-Falardeau, Shipshaw, Saint-Charles-de-Bourget

Supplementary Methods

Simulating GWAS stratification using the BALSAC SLSJ genealogy. To assess the impact of population stratification driven by a fine structure induced by a regional founder effect on a genome-wide association study (GWAS), we performed a coalescent simulation using 10,000 probands married between 1931–1960 sampled from the BALSAC genealogy of Saguenay–Lac-Saint-Jean (SLSJ). Probands were randomly selected among 16,269 probands related less than first cousins. We generated 100 Mb of neutral genomic sequence using `msprime`[1,2] 1.3.4's `FixedPedigree` model to track ancestry through the known genealogy, recapitated the simulation with the OutOfAfrica_3G09 demographic model[3] via `stdpopsim`[4,5,6] 0.3.0 to provide realistic ancestral diversity, and applied standard human mutation (1.2×10^{-8})[7] and recombination (1.1×10^{-8})[8] rates. A correlated phenotype was constructed based on the expected genetic contribution from founders originating in the Charlevoix region, standardized and mixed with random noise ($h^2 = 0.1$), using the following formula:

$$y \sim \mathrm{GC} \times \sqrt{h^2} + N(0,1) \times \sqrt{1 - h^2}.$$

Following filtering for minor allele frequency (MAF > 0.01), we performed association testing using linear regression under four conditions: uncorrected, corrected with 10 or 100 genetic principal component (PCA) of common single nucleotide polymorphisms (MAF > 0.05) pruned for linkage disequilibrium (window = 50 variants, sliding window = 5 variants, $r^2 = 0.2$), and corrected with 2 genealogical principal coordinates (PCoA) derived from the genealogical expected kinship matrix, evaluating model performance via the genomic inflation factor (λ_{GC})[9] and quantile-quantile plots.

Supplementary Note 1

Pseudocode for hybrid kinship algorithm. In the C++ implementation of this algorithm, *genealogy* is a hash map of a data structure named *individual*, referenced through a unique integer ID. The genealogy is sorted so that any ancestor appears before their offspring. Each *individual* contains notably a reference to two other *individual* structures, a father (if present in the genealogy) and a mother (if present). It also possesses an immutable *rank* from 1 to *n* individuals in the genealogy, which indicates the individual's order in the hash map. Finally, its mutable *founder_index* indicates whether the individual is currently a founder (if non null) and, if that's the case, where their kinship values are located in the current founder kinship matrix. The function compute_kinships(*genealogy, proband_IDs*) is called using the hash map and a list of the probands' integer IDs, and returns a kinship matrix with the probands in the same order as in the list. The function compute_kinship_between_probands may run the nested for loop in parallel, for each generation.

```
FUNCTION get_previous_generation(genealogy, current_individuals):
    CREATE a new empty set called 'parents'
    FOR each ID in 'current_individuals':
        GET the person from 'genealogy' using the ID
        IF person has a father:
            ADD father's ID to 'parents'
        IF person has a mother:
            ADD mother's ID to 'parents'
    RETURN 'previous_generation'

FUNCTION get_generations(genealogy, starting_probands):
    CREATE an empty list of sets called 'generations'
    CREATE an empty set called 'current_generation'
    FOR each ID in starting_probands:
        ADD ID to 'current_generation'
    WHILE 'current_generation' is not empty:
        ADD 'current_generation' to 'generations'
        SET 'current_generation' to the result of calling get_previous_generation(genealogy,
current_generation)
    RETURN 'generations'

FUNCTION copy_bottom_up(generations):
    CREATE an empty list of sets called 'bottom_up'
    CREATE an empty set called 'first_generation'
    FOR each ID in the first set of 'generations':
        ADD ID to 'first_generation'
    ADD 'first_generation' to 'bottom_up'
    FOR each index 'i' from the first to the second-to-last set of 'generations':
        CREATE an empty set called 'combined_generation'
        CREATE a set called 'previous_generation' from the individuals in bottom_up[i]
        CREATE an empty set called 'next_generation'
        FOR each ID in generations[i + 1]:
            ADD ID to 'next_generation'
        PERFORM a set union of 'previous_generation' and 'next_generation', adding the unique
results to 'combined_generation'
        ADD 'combined_generation' to 'bottom_up'
    REVERSE 'bottom_up' to go from oldest to youngest
    RETURN 'bottom_up'

FUNCTION copy_top_down(generations):
    REVERSE the order of 'generations'
    CREATE an empty list of sets called 'top_down'
    CREATE an empty list called 'first_generation'
    FOR each ID in the first set of the (now reversed) 'generations':
        ADD ID to 'first_generation'
    ADD 'first_generation' to 'top_down'
    FOR each index 'i' from the first to the second-to-last set of 'generations':
        CREATE an empty set called 'combined_generation'
        CREATE an empty set called 'previous_generation'
        FOR each ID in top_down[i]:
            ADD ID to 'previous_generation'
```

```
        CREATE an empty set called 'next_generation'
        FOR each ID in generations[i + 1]:
            ADD ID to 'next_generation'
        PERFORM a set union of 'previous_generation' and 'next_generation', adding the unique
results to 'combined_generation'
        ADD 'combined_generation' to 'top_down'
    RETURN 'top_down'

FUNCTION intersect_both_directions(bottom_up, top_down):
    CREATE an empty list of lists called 'vertex_cuts'
    FOR each index 'i' for every list in 'bottom_up':
        CREATE an empty list called 'current_vertex_cut'
        CREATE a set called 'set_from_bottom_up' from the IDs in bottom_up[i]
        CREATE a set called 'set_from_top_down' from the IDs in top_down[i]
        CREATE an empty set called 'common_individuals'
        PERFORM a set intersection of 'set_from_bottom_up' and 'set_from_top_down', adding the
unique results to 'common_individuals'
        FOR each ID in 'common_individuals':
            ADD ID to 'current_vertex_cut'
        ADD 'current_vertex_cut' to 'vertex_cuts'
    RETURN 'vertex_cuts'

FUNCTION cut_vertices(genealogy, proband_IDs):
    CREATE an empty list of sets called 'generations'
    CREATE an empty list of lists called 'vertex_cuts'
    SET 'generations' to the result of calling get_generations(genealogy, proband_IDs)
    CREATE empty lists of sets called 'bottom_up' and 'top_down'
    SET 'bottom_up' to the result of calling copy_bottom_up(generations)
    SET 'top_down' to the result of calling copy_top_down(generations)
    SET 'vertex_cuts' to the result of calling intersect_both_directions(bottom_up, top_down)
    SET the last list in 'vertex_cuts' to 'proband_IDs'
    RETURN 'vertex_cuts'

FUNCTION compute_kinship(ind1, ind2, founder_matrix):
    SET 'kinship' to 0.0
    SET 'founder_index1' to individual ind1's founder_index
    SET 'founder_index2' to individual ind2's founder_index
    IF 'founder_index1' is not null AND 'founder_index2' is not null:
        SET 'kinship' to founder_matrix[founder_index1][founder_index2]
    ELSE IF 'founder_index1' is not null:
        IF ind2 has a father:
            ADD 0.5 * compute_kinship(ind1, ind2's father, founder_matrix) to 'kinship'
        IF ind2 has a mother:
            ADD 0.5 * compute_kinship(ind1, ind2's mother, founder_matrix) to 'kinship'
    ELSE IF 'founder_index2' is not null:
        IF ind1 has a father:
            ADD 0.5 * compute_kinship(ind1's father, ind2, founder_matrix) to 'kinship'
        IF ind1 has a mother:
            ADD 0.5 * compute_kinship(ind1's mother, ind2, founder_matrix) to 'kinship'
    ELSE IF ind1's rank is equal to ind2's rank:
        SET 'kinship' to 0.5
        IF ind1 has a father AND ind2 has a mother:
            ADD 0.5 * compute_kinship(ind1's father, ind2's mother, founder_matrix) to 'kinship'
    ELSE IF ind1's rank is less than ind2's rank:
        IF ind2 has a father:
            ADD 0.5 * compute_kinship(ind1, ind2's father, founder_matrix) to 'kinship'
        IF ind2 has a mother:
            ADD 0.5 * compute_kinship(ind1, ind2's mother, founder_matrix) to 'kinship'
    ELSE:
        IF ind1 has a father:
            ADD 0.5 * compute_kinship(ind1's father, ind2, founder_matrix) to 'kinship'
        IF ind1 has a mother:
            ADD 0.5 * compute_kinship(ind1's mother, ind2, founder_matrix) to 'kinship'
    RETURN 'kinship'

FUNCTION compute_kinship_with_oneself(probands, founder_matrix, proband_matrix):
    FOR each index 'i' for every proband in 'probands':
        SET 'proband' to probands[i]
        SET 'kinship' to 0.5
        IF proband has a father AND proband has a mother:
```

```
            ADD 0.5 * compute_kinship(proband's father, proband's mother, founder_matrix) to
'kinship'
        SET proband_matrix[i][i] to 'kinship'

FUNCTION compute_kinship_between_probands(probands, founder_matrix, proband_matrix):
    FOR each index 'i' for every proband in 'probands':
        SET 'proband1' to probands[i]
        FOR each index 'j' from the first proband up to (but not including) the current index 'i':
            SET 'proband2' to probands[j]
            SET 'kinship' to compute_kinship(proband1, proband2, founder_matrix)
            SET proband_matrix[i][j] to 'kinship'
            SET proband_matrix[j][i] to 'kinship'

FUNCTION compute_kinships(genealogy, proband_IDs):
    SET 'vertex_cuts' to the result of calling cut_vertices(genealogy, proband_IDs)
    CREATE a new matrix called 'current_founder_matrix' with dimensions (size of vertex_cuts[1],
size of vertex_cuts[1]) and filled with zeros
    FOR each index 'i' for every individual in the first vertex cut:
        SET current_founder_matrix[i][i] to 0.5
    FOR each index 'i' from the first to the second-to-last vertex cut:
        SET 'founder_index' to 1
        FOR each person ID in vertex_cuts[i]:
            GET the person from the genealogy using their ID
                SET person's founder index (null by default) to 'founder_index'
            INCREMENT 'founder_index'
        CREATE a new matrix called 'current_proband_matrix' with dimensions (size of vertex_cuts[i +
1], size of vertex_cuts[i + 1])
        CREATE an empty list of persons called 'current_probands'
        FOR each person ID in vertex_cuts[i + 1]:
            ADD 'person' from 'genealogy' using their ID to 'current_probands'
        CALL compute_kinship_with_oneself(current_probands, current_founder_matrix,
current_proband_matrix)
        CALL compute_kinship_between_probands(current_probands, current_founder_matrix,
current_proband_matrix)
        SET 'current_founder_matrix' to 'current_proband_matrix'
    RETURN 'current_founder_matrix'
```

Time complexity of the hybrid kinship algorithm. The preprocessing phase, which identifies vertex cuts through ancestral traversal and set operations, has $O(n)$ time complexity where n is the total number of individuals in the genealogy. The kinship computation phase has $O(\Sigma\, v_i^2)$ time complexity, where v_i represents the number of individuals in vertex cut i and the sum is taken over all g vertex cuts. Since kinship coefficients are computed for all pairs within each vertex cut and memoised in founder matrices, subsequent recursive calls achieve $O(1)$ amortised lookup time. The overall algorithm complexity is dominated by $O(\Sigma\, v_i^2)$, which can be approximated as $O(g \times \bar{v}^2)$ or $O(n \times \bar{v})$ where $\bar{v}$ is the mean vertex cut size.

Supplementary References

1. Baumdicker, F. *et al.* Efficient ancestry and mutation simulation with msprime 1.0. *Genetics* **220**, iyab229 (2022).
2. Kelleher, J., Etheridge, A. M. & McVean, G. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Comput. Biol.* **12**, e1004842 (2016).
3. Gutenkunst, R. N., Hernandez, R. D., Williamson, S. H. & Bustamante, C. D. Inferring the joint demographic history of multiple populations from multidimensional SNP frequency data. *PLoS Genet.* **5**, e1000695 (2009).
4. Adrion, J. R. *et al.* A community-maintained standard library of population genetic models. *eLife* **9**, e54967 (2020).
5. Lauterbur, M. E. *et al.* Expanding the stdpopsim species catalog, and lessons learned for realistic genome simulations. *eLife* **12**, RP84874 (2023).
6. Gower, G. *et al.* Accessible, realistic genome simulation with selection using stdpopsim. *Mol. Biol. Evol.* **42**, msaf236 (2025).
7. Kong, A. *et al.* Rate of de novo mutations and the importance of father's age to disease risk. *Nature* **488**, 471–475 (2012).
8. Kong, A. *et al.* A high-resolution recombination map of the human genome. *Nat. Genet.* **31**, 241–247 (2002).
9. Devlin, B. & Roeder, K. Genomic control for association studies. *Biometrics* **55**, 997–1004 (1999).