Bioinformatics, 2020, 1–3
doi: 10.1093/bioinformatics/btaa988
Advance Access Publication Date: 28 November 2020
Applications Note

Genetics and population analysis

miqoGraph: fitting admixture graphs using mixed-integer quadratic optimization

Julia Yan[1,*], Nick Patterson[2] and Vagheesh M. Narasimhan[2,3,4]

[1]Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139, USA, [2]Department of Genetics, Harvard Medical School, Boston, MA, 02115, USA, [3]Department of Integrative Biology, The University of Texas at Austin and [4]Department of Statistics and Data Science, The University of Texas at Austin

*To whom correspondence should be addressed.

Associate Editor: Russell Schwartz

Received on August 20, 2020; revised on October 25, 2020; editorial decision on November 12, 2020; accepted on November 16, 2020

Abstract

Summary: Admixture graphs represent the genetic relationship between a set of populations through splits, drift and admixture. In this article, we present the Julia package miqoGraph, which uses mixed-integer quadratic optimization to fit topology, drift lengths and admixture proportions simultaneously. Through applications of miqoGraph to both simulated and real data, we show that integer optimization can greatly speed up and automate what is usually an arduous manual process.

Availability and implementation: https://github.com/juliayyan/PhylogeneticTrees.jl.

Contact: jyyan@alum.mit.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

The genetic relationship between a set of populations cannot be described precisely by a simple tree because of the presence of admixture. An admixture graph provides a way to represent the complex relationship between populations, including their separation, subsequent drift and possible merging by using distributions of multiple trees. Several methods exist to build and visualize admixture graphs as well as to infer optimal parameters of drift lengths and admixture edges, such as TreeMix (Pickrell and Pritchard, 2012), AdmixTools (Patterson *et al.*, 2012), MixMapper (Lipson *et al.*, 2013) and admixturegraph (Leppälä *et al.*, 2017). Some of these methods cannot simultaneously infer the optimal topology with the parameters of the graph under that topology. Rather, they require that the topology of a particular graph be pre-specified, and then infer the graph parameters. Other methods such as TreeMix (Pickrell and Pritchard, 2012) and MixMapper (Lipson *et al.*, 2013) search a restricted space of possible admixtures. In Leppälä *et al.* (2017), all possible topologies are enumerated using exhaustive searches; however, such an approach becomes intractable at larger problem sizes. Sridhar *et al.* (2008), Catanzaro *et al.* (2013) and Fortz *et al.* (2017) use mixed-integer linear optimization is used to infer graph topology and drift lengths simultaneously. However, in contrast to our approach, which uses a maximum likelihood objective, these works fit trees of maximum parsimony (or, minimum total drift length). Furthermore, unlike our work, they do not consider the possibility of admixture.

Here, we describe the algorithms in our miqoGraph package, which uses mixed-integer quadratic optimization to infer graph topology, drift lengths and admixture proportions simultaneously. For medium-size graphs in which admixture occurs only at leaf nodes, we can find optimal solutions in seconds and prove optimality (subject to input parameters described in Section 2) in minutes. We test our algorithm on simulated and real datasets comprising several populations and show that even as fewer parameters are specified *a priori*, running times are sped up over competitive algorithms by orders of magnitude.

2 Materials and methods

A typical approach to admixture graph fitting is to first specify a topology, and then compute the graph's fit to genetic data. Drift patterns in the data can be summarized by *f*-statistics (Patterson *et al.*, 2012), and for a given topology it is possible to construct a basis set of expected values of *f*-statistics that define the graph (Pickrell and Pritchard, 2012). We will refer to the vector of empirical *f*-statistics as $\mathbf{f}$, and given a topology $\mathbf{x}$, drift lengths $\mathbf{w}$ and admixture proportions $\boldsymbol{\alpha}$, we will call the vector of the expected *f*-statistics $g(\mathbf{w}, \boldsymbol{\alpha}; \mathbf{x})$. Drift lengths and admixture proportions are then selected to maximize the likelihood as follows:

$$\max_{\mathbf{w}\geq 0, \boldsymbol{\alpha}\in\mathcal{U}} -\left(\mathbf{f}-g(\mathbf{w},\boldsymbol{\alpha};\mathbf{x})\right)' \Sigma^{-1} \left(\mathbf{f}-g(\mathbf{w},\boldsymbol{\alpha};\mathbf{x})\right), \tag{1}$$

where Σ^{-1} is the covariance matrix of the empirical statistics $\mathbf{f}$, and $\mathcal{U}$ represents the set of all valid admixture proportions. This is the approach of qpGraph, developed by Patterson *et al.* (2012).

In our approach, which we call miqoGraph, rather than fixing the topology $\mathbf{x}$ before solving for drift lengths $\mathbf{w}$ and admixture proportions $\boldsymbol{\alpha}$, we optimize over all three *simultaneously*:

$$\max_{\mathbf{x}\in\mathcal{T},\mathbf{w}\geq 0,\boldsymbol{\alpha}\in\mathcal{U}} -\left(\mathbf{f}-g(\mathbf{w},\boldsymbol{\alpha};\mathbf{x})\right)'\Sigma^{-1}\left(\mathbf{f}-g(\mathbf{w},\boldsymbol{\alpha};\mathbf{x})\right), \tag{2}$$

where $\mathcal{T}$ represents the set of all valid topologies. Topologies have previously been explored by enumeration over small graphs (Leppälä *et al.*, 2017), but this approach is intractable for larger graphs. Here, we present a novel formulation of the problem using mixed-integer quadratic optimization (MIQO), where we model the problem of determining a best-fit graph topology as assignment of populations to leaf nodes of a binary tree. Although such problems are difficult in theory, modern solvers such as Gurobi (Gurobi Optimization, Inc., 2016), which is available via academic license, can quickly solve large-scale MIQO problems in practice. For an overview of integer optimization, see Wolsey and Nemhauser (2014). For the specifics of formulation (2) and a list of other optimization solver options, see Supplementary Material.

Our approach requires pre-specification of the following parameters:

1. Tree depth $D \in \mathbb{Z}^+$,
2. Number of admixture events $A \in \{0\} \cup \mathbb{Z}^+$ and
3. Admixture resolution K for $K \in \mathbb{Z}^+$ (only needed if $A > 0$).

If there are no admixture events ($A = 0$), the populations' relationship can be represented using a single binary tree. We model admixture ($A > 0$) by allowing the population assignments to leaf nodes to be between 0 and 100%, and the admixture resolution K allows admixture proportions to be estimated to an accuracy of $\frac{1}{K}$. For example, $K = 10$ allows values of $0\%, 10\%, \ldots, 90\%, 100\%$ (see Supplementary Material).

We then solve optimization problem (2) to find the best-fit tree topology, drift lengths and admixture proportions under the specified parameters. A major benefit of miqoGraph over prior approaches is the flexibility of the parameters, with each specification of parameter values representing numerous potential admixture graphs. As such, although it is computationally intractable to enumerate over all potential topologies for several populations, our algorithm quickly finds well-fit topologies using MIQO. Although it may not be obvious which parameter values are appropriate *a priori*, multiple optimization problems can be solved in parallel on a reasonable range of parameter values. In our experiments, we found that trying one tree depth, several admixture resolutions and a few admixture events were sufficient to find the correct admixture graph topologies.

Although it is not required, prior knowledge can reduce the solution space and speed up the solution time. For example, a user can specify that the path from the root to a particular population does not contain admixture, which we found to be a particularly useful feature in our simulations.

3 Computational results

We first validated our model on simulated data (see Supplementary Section S5) and showed that we can infer the correct topology on several known graphs of increasing complexity. We now apply our model to a six-population dataset of modern and ancient DNA samples from Eurasia and the Americas, in order to infer the phylogeny of populations leading to the Karitiana, a South American population from Brazil.

On this Eurasian and American dataset, even without specifying which population should be admixed and at a coarse admixture granularity of $K = 2$, miqoGraph found a solution within 8 s and verified optimality after 9 s. Most importantly, the graph matched one found using exhaustive searches with qpGraph.

We were able to refine the admixture proportions by running miqoGraph at higher resolutions of $K = 3$ and 4. By leveraging the knowledge that Karatiana should be admixed, learned from the $K = 2$ output, miqoGraph inferred the correct topology almost instantaneously (1 and 3 s, respectively). At $K = 4$, the inferred admixture proportions 25–75% corresponded closely to the values of 28–72% estimated by qpGraph, and the drift lengths were also similar (see Supplementary Material). We also ran further instances of miqoGraph for $K = 5$ through 10, and as expected, saw convergence in topology, and qualitatively similar weights and proportions. Even at the highest granularity of $K = 10$, miqoGraph terminated in under a minute.

The admixture graph inferred by miqoGraph at $K = 4$ is shown inFigure 1. In this topology, Karatiana is admixed between an ancient North Eurasian-related and a present-day East Asian-related source, consistent with previous results examining the initial peopling of the Americas (Raghavan *et al.*, 2014).

4 Limitations

The main limitation of miqoGraph lies in the restriction of admixture events to the leaf nodes of the graph and therefore, the interpretation of its output in the presence of multiple nested admixture events. Suppose a particular population A has admixture from populations B and C, and that B itself is admixed from D and E. The ordering of these events is not captured in our representation of the graph, and it can be challenging to reconstruct the correct sequence of events leading to the true admixture graph. Furthermore, if there is drift post-admixture, we are unable to capture this in our current formulation. To aid interpretability, our framework allows the user to sequentially add new populations while fixing the topology for other populations. The positions of these new populations can vary freely, or they can be tentatively assigned to positions based on the user's best guess, giving the optimizer a 'warm start' to improve upon. A second issue with our approach is that the proportion of admixture inferred is done in discrete values whose granularity is specified *a priori*. It is possible that at low admixture granularities, the best-fit topology may be incorrect. One possible way to mitigate this effect is to use miqoGraph to explore a possible set of graph topologies and then to use continuous optimizers such as that implemented in AdmixTools (Patterson *et al.*, 2012) to fit parameters on these topologies.

5 Conclusion

Using miqoGraph, we are able to simultaneously infer topologies, drift lengths and admixture proportions in seconds to minutes on admixture graphs and we applied our method to several simulated and real world cases. Due to the restriction of admixture events to leaf nodes, our formulation is primarily useful in settings with few nested admixture events. Nonetheless, the use of integer

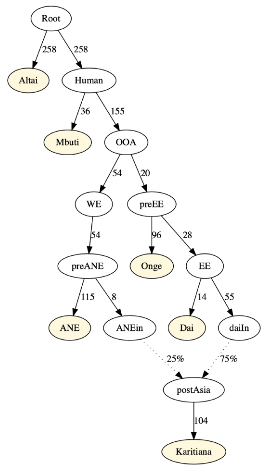


Fig. 1. Topology, drift lengths and admixture proportions inferred on a dataset from Eurasia and the Americas. Populations with real data are colored in beige, while auxiliary nodes are uncolored

optimization to model what was previously a combination of labor-intensive manual enumeration and continuous optimization represents a significant step forward in efficient inference of admixture graphs. Such methods are likely to become increasingly important as dataset sizes grow, and our formulation provides an important starting point for the development of future methodologies.

Acknowledgements

The authors thank Dimitris Bertsimas and members of the Reich laboratory for productive discussions.

Financial Support: none declared.

Conflict of Interest: none declared.

References

Catanzaro,D. *et al.* (2013) A mixed integer linear programming model to reconstruct phylogenies from single nucleotide polymorphism haplotypes under the maximum parsimony criterion. *Algorithms Mol. Biol.*, 8, 2.

Fortz,B. *et al.* (2017) Compact mixed integer linear programming models to the minimum weighted tree reconstruction problem. *Eur. J. Oper. Res.*, **256**, 242–251.

Gurobi Optimization, Inc. (2016) *Gurobi Optimizer Reference Manual.* https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.0/refman.pdf (7 December 2020, date last accessed).

Leppälä,K. *et al.* (2017) admixturegraph: an r package for admixture graph manipulation and fitting. *Bioinformatics*, **33**, 1738–1740.

Lipson,M. *et al.* (2013) Efficient moment-based inference of admixture parameters and sources of gene flow. *Mol. Biol. Evol.*, **30**, 1788–1802.

Patterson,N. *et al.* (2012) Ancient admixture in human history. *Genetics*, **192**, 1065–1093.

Pickrell,J.K. and Pritchard,J.K. (2012) Inference of population splits and mixtures from genome-wide allele frequency data. *PLoS Genet.*, **8**, e1002967.

Raghavan,M. *et al.* (2014) Upper Palaeolithic Siberian genome reveals dual ancestry of Native Americans. *Nature*, **505**, 87–91.

Sridhar,S. *et al.* (2008) Mixed integer linear programming for maximum-parsimony phylogeny inference. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, **5**, 323–331.

Wolsey,L.A. and Nemhauser,G.L. (2014) *Integer and Combinatorial Optimization*. John Wiley & Sons, Hoboken, NJ.

`miqoGraph` Supplementary Material

Julia Yan[1,*], Nick Patterson[2], and Vagheesh Narasimhan[2]

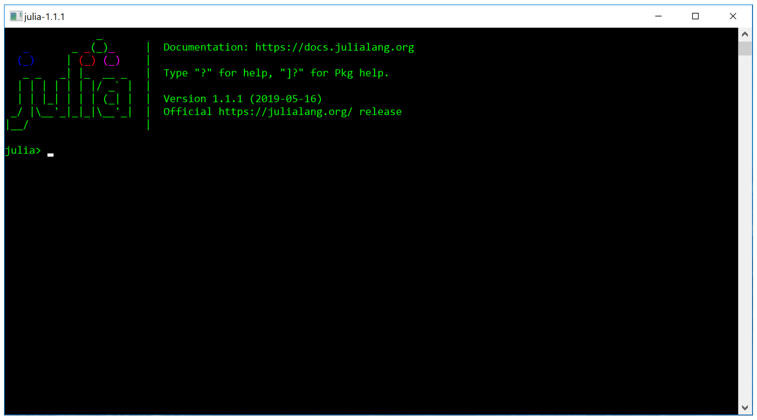
[1] Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA, 02139
[2] Department of Genetics, Harvard Medical School, Boston, MA, 02115

Section 1 provides documentation on installation and running the algorithm. Section 2 reviews some preliminaries of f-statistics and admixture. Section 3 explains the details of the mixed-integer quadratic optimization formulation. Section 4 includes the parameters used for simulating the SimpleMix, UnevenMix, and NestedMix datasets. Section 5 provides detailed computational results on the simulated and real data sets. Section 6 discusses limitations of the algorithm.

1 Documentation

1.1 Installation Instructions

1. Download the **Julia language** from `https://julialang.org/downloads/`. This package was developed using v1.0, but has also been tested on v1.1; **for best results, use one of those versions**. Open Julia and you should see a window that looks similar to Figure 1. Documentation for Julia can be found here: `https://docs.julialang.org/en/v1/index.html`. Directions for running Julia directly from the terminal can be found here: `https://en.wikibooks.org/wiki/Introducing_Julia/Getting_started#Running_directly_from_terminal`.



```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.1.1 (2019-05-16)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 
```

Figure 1: The Julia terminal.

2. If you are on a cluster that has the **Gurobi Optimizer** already installed, you should be able to skip this step and load Gurobi as instructed by the cluster admins (for example, `module load gurobi` on Slurm). Otherwise, download the **Gurobi Optimizer**. We provide a summary of the necessary instructions here; for further issues, consult the quick start guide: `https://www.gurobi.com/documentation/quickstart.html` (**Note:** this is more comprehensive than what you will need. For example, you should not need to follow instructions for interfacing with languages other than Julia).

   (a) **Students, faculty, and staff at degree-granting academic institutions.** You must be on a university network and use your university email address for this step. Register for an academic license here: `https://www.gurobi.com/downloads/end-user-license-agreement-academic/`. Then log in. Go to the downloads page: `https://www.gurobi.com/downloads/`. Download the Gurobi Optimizer. Navigate back to the downloads page, and under **Request a License**, click Academic License. Accept the conditions, and run the `grbgetkey` command given at the bottom of the page under **Installation**. The license needs to be renewed after one year.

   (b) **Other users.** A free trial of Gurobi is available here: `https://www.gurobi.com/free-trial/`. We are working on compatibility with open-source solvers.

   If you installed Gurobi to a non-default location you need to set several environment variables. On a bash shell you can do this by adding the following lines to your .bashrc file. Once these are added, please re-login to your shell to set these or export them in your current session.

```
export GUROBI_HOME="/user/gurobi801/linux64"
export PATH="${PATH}:${GUROBI_HOME}/bin"
export LD_LIBRARY_PATH="${GUROBI_HOME}/lib"
export GRB_LICENSE_FILE="/user/gurobi801/linux64/bin/y/gurobi.lic"
```

Please change these according to the relevant locations for your installation. `GUROBI_HOME` should be set to the location of `gurobi.sh`, and `GRB_LICENSE_FILE` should be set to the location of `gurobi.lic`. You can now test your license by opening the Gurobi Interactive Shell. On Windows, this can be done by opening the Gurobi Interactive Shell application or double clicking on the desktop Gurobi icon. On a Mac or on Linux, this can be done by typing `gurobi.sh` in the Terminal. The shell should display text similar to the following:

```
Gurobi Interactive Shell, Version 8.1.1
Copyright (c) 2019, Gurobi Optimization, LLC
Type "help()" for help
gurobi>
```

More details are available in the quick start guide.

3. Within Julia, open the Julia package manager. This can be done by typing the right bracket (]) key in the Julia terminal shown in Figure 1. You should see you are in the package manager if the text in front of the cursor switches from `julia>` to `(v1.0) pkg>` or `(v1.1) pkg>`. Install `miqoGraph` by running the following command in the package manager:

   ```
   (v1.0) pkg> add https://github.com/juliayyan/PhylogeneticTrees.jl.git
   ```

   (Do not type the colored text; it is just meant to indicate that you should be in the package manager.) For a full reference to the Julia package manager, see `https://julialang.github.io/Pkg.jl/v1/`. The code for PhylogeneticTrees should be contained in ` /.julia/packages/PhylogeneticTrees/xx` where `xx` is a unique hash code.

4. To get access to solver-specific parameters such as time limits and output flags, you will want to install the Gurobi package separately. It will also be helpful to install the JuMP, CSV, and DataFrames packages separately. Run the following command in the package manager:

   ```
   (v1.0) pkg> add Gurobi
   (v1.0) pkg> add JuMP@0.18.5
   (v1.0) pkg> add CSV@0.4.3
   (v1.0) pkg> add DataFrames@0.17.1
   ```

   (You can generally add packages using the package name instead of the git url. Since our package is not registered in the wider Julia package ecosystem, we use the git url for installation.)

5. Within the Julia package manager, test the package by running the following command:

   ```
   (v1.0) pkg> test PhylogeneticTrees
   ```

   This will take a few minutes and print a lot of output. The final message should say `Testing PhylogeneticTrees tests passed`. You can exit the package manager by pressing backspace.

6. **(For future reference)** If at a future point you need to update the package, you can run the following command:

   ```
   (v1.0) pkg> update PhylogeneticTrees
   ```

Installation troubleshooting. If you are unable to run `test PhylogeneticTrees`, you may have an issue with the package dependencies. Here are some common issues:

- **Build errors.** If you see an error message saying something like `Please run Pkg.build("X")` (where X is some package name), then within the Julia package manager, run the following:
  ```
  (v1.0) pkg> build X
  ```
  (Do not type the colored text; it is just meant to indicate that you should be in the package manager.) Note that there are no quotation marks around the package name X in the command executed within the package manager. Quit the Julia application and try again.

- **Older or newer package versions.** Within the Julia package manager, type the following command:
  ```
  (v1.0) pkg> status
  ```
  This package was developed under JuMP v0.18.5, CSV v0.4.3, and DataFrames v0.17.1. If you see other versions listed, you can switch to these versions with the following commands:
  ```
  (v1.0) pkg> add JuMP@0.18.5
  (v1.0) pkg> add CSV@0.4.3
  (v1.0) pkg> add DataFrames@0.17.1
  ```
  Note that if you are using Julia 1.2 or above, you will need JuMP v0.18.6 instead of JuMP v0.18.5.

- **Solver access.** Gurobi is available under both academic and commercial licenses. For those who do not have access to such licenses, a variety of free, open-sources solvers that can solve MIQO problems are available. Among open-source JuMP-compatible solvers, there are mixed-integer nonlinear solvers such as Bonmin [3], Couenne [2], and SCIP [1]; and the mixed-integer convex solvers Juniper [9] and Pajarito [5]. The ECOS_BB [6] and MIOSQP solvers [13] are options in Python. Although the code in our package is not solver-agnostic, the JuMP-compatible solvers could be substituted for Gurobi by replacing all instances of Gurobi in the package with the chosen solver name.

1.2 Usage

Some code and data examples are provided in the `test/` directory of the GitHub repository. Here, we give more detail on the required input data and code.

1.2.1 Input Data

f_3-statistics file. The f_3-statistics are provided as a CSV. The CSV must follow the following header format:

```
Outgroup,A,B,f3
```

where the first column is the outgroup, the second and third columns are populations, and the fourth column is the value of the statistic $F_3(\texttt{Outgroup}; \texttt{A}, \texttt{B})$. Each row will be a different f_3-statistic, but it is not necessary to include both permutations of the `A` and `B` columns. All population names must be `String`s, i.e., 7 is not a valid population name. They also should not include spaces.

The first few rows of a sample f_3-statistics CSV are shown below as an example. The remainder of the file can be found in `test/testdata/f3.Europe6.csv`.

```
Outgroup,A,B,f3
Mbuti,Altai,Altai,482.47
Mbuti,Altai,WEHG,34.748
```

Covariance file. The covariance matrix is also provided as a CSV. The CSV must have the following header format:

```
A1,B1,A2,B2,covariance
```

where the first four columns are populations, and the fifth column is the value of the covariance for $F_3(\texttt{Outgroup}; \texttt{A1}, \texttt{B1})$ and $F_3(\texttt{Outgroup}; \texttt{A2}, \texttt{B2})$. Each row will be a different element of the covariance matrix, but it is not necessary to include all permutations of the columns. All population names must match those in the f_3-statistics file.

The covariance matrix can be obtained by running qpGraph (version 6.0) with the following parameter settings:

```
fstatsname:  filename.txt
doanalysis:  NO
```

The first few rows of a sample covariance CSV are shown below as an example. The remainder of the file can be found in `test/testdata/f3.Europe6-covariance.csv`.

```
A1,B1,A2,B2,covariance
Altai,Altai,Altai,Altai,4.531
Altai,Altai,Altai,WEHG,1.535
```

1.2.2 A Default Wrapper

For user convenience, a default wrapper is contained in `example/default.jl`, along with a default parameters file in `example/params-miqo.csv`.

The `params-miqo.csv` file contains the following fields:

- `mean_file` The name of the file containing the f_3-statistics,
- `cov_file` The name of the file containing the covariance matrix,
- `output_file` The name of the file that PhylogeneticTrees should write output (the topology) to,
- `log_file` The name of the file that PhylogeneticTrees should write logging information to,
- `time_limit` A time limit (in seconds) for Gurobi,
- `warm_start` 1 if the model with admixture should warm-start the problem with a tree without admixture and 0 otherwise,
- `warm_start_time_limit` A time limit (in seconds) for the warm start model,
- `depth` Depth of the tree (a count of the edges from root to leaf),
- `granularity` Admixture granularity ($K \geq 1$),
- `admixture_events` Number of admixture events ($A \geq 0$),
- `unmixed_pops` Any populations (whose names should match those in the mean and covariance files) that should not experience admixture in the model, separated by spaces.

The default wrapper for the Americas example in our paper can be run by navigating to `example/` and running the command `julia default.jl params-miqo.csv`. The wrapper by default looks for a file called `params-miqo.csv`, so the argument can be omitted if this is the parameters file name.

1.2.3 Reading Data

1. If you are still in the Julia package manager, exit the package manager by pressing backspace. You should see the `julia>` text before your cursor.

2. Within Julia, navigate to the directory containing your data files. Open the shell mode by typing the semicolon (;) key in the Julia terminal shown in Figure 1. You should see that you are in shell mode if the text in front of the cursor switches from `julia>` to `shell>`. Once in shell mode, you can use the system shell to execute system commands. Navigate to the directory that is storing your data files with the following command:
   ```
   shell> cd YOUR_DIRECTORY
   ```

3. Exit shell mode by pressing backspace. You should see the `julia>` text before your cursor. Load the `PhylogeneticTrees` and `Gurobi` packages with the following line of code:
   ```
   julia> using PhylogeneticTrees, Gurobi
   ```

4. You can then read data from your files using the following code:
   ```
   julia> pd = PhylogeneticTrees.PopulationData("F3_FILE.csv", "COVARIANCE_FILE.csv")
   ```
   The first argument of the `PhylogeneticTrees.PopulationData()` function is the file name of the f_3-statistics CSV file, and the second argument is the file name of the covariance CSV file. This code stores the population data in a data structure named `pd`.

5. The number of populations, outgroup key, population names, f_3-statistics, and covariance matrix can be accessed through `pd.npop`, `pd.outgroup`, `pd.pops`, `pd.f3`, and `pd.cov`, respectively.

1.2.4 Fitting a Model

Following these steps will allow you to construct a model **without admixture**.

1. Build a binary tree data structure of depth $D = 3$ with the following code:
   ```
   julia> D = 3
   julia> bt = PhylogeneticTrees.BinaryTree(D)
   ```
   This will create a binary tree with $2^{D+1} - 1 = 15$ nodes, $2^D = 8$ of which are leaf nodes.

2. Construct the optimization model with the following code:
   ```
   julia> tp = PhylogeneticTrees.TreeProblem(pd, bt, solver = GurobiSolver(TimeLimit = 60))
   ```
   The first argument of `PhylogeneticTrees.TreeProblem()` is the data structure containing the population data, the second argument is the binary tree, and the third is the solver. The `TimeLimit` flag indicates that the solver will terminate after 60 seconds.

3. To solve the model, use the following code:
   ```
   julia> solve(tp.model)
   ```
   The `solve()` function comes from the JuMP package, which you should have installed separately.

4. To print which populations were assigned to which nodes, you can use the following code:
   ```
   julia> PhylogeneticTrees.printnodes(tp)
   ```
   In this output, the first column is the population name, the second column is the node, and the third column is the proportion of the population that was assigned to that node (always 1.0 if there is no

admixture).

For relatively small trees (depth 4 and below), you can print a visualization of the tree itself using the following code:

```
julia> PhylogeneticTrees.printtree(tp)
```

Admixture events can be added using optional arguments to the `PhylogeneticTrees.TreeProblem()` function:

- `nlevels` (default `1`): the level of granularity K that is desired for admixture proportion estimation. The higher K is, the more precise the estimation, which is in intervals of $\frac{1}{K}$. By default, the granularity level is set to 1, indicating no admixture. We recommend starting with a low level before moving to higher levels to test what your computer can handle. We have tested up to $K = 10$, and do not recommend going significantly further. At $K = 10$, the proportions can take on values $0\%, 10\%, 20\%, \ldots, 90\%, 100\%$.

- `nmixtures` (default `pd.npop`): the maximum number of nodes that can be assigned to, or `pd.npop`$+A$, where A is the number of admixture events. By default, `nmixtures = pd.npop` indicates $A = 0$, meaning that there are no admixture events.

Now, to construct a model **with admixture**, we model the previous procedure as follows:

1. Same as without admixture

2. Construct the optimization model with the following code:
   ```
   julia> tp = PhylogeneticTrees.TreeProblem(pd, bt, solver = GurobiSolver(TimeLimit = 60),
   nlevels = 2, nmixtures = pd.npop + 1)
   ```
   The extra parameters allow one of the populations to be mixed at 50%-50%.

3. The mode can be solved directly as before, but a couple of extra lines of code can help the model solve more quickly.

   (a) A "warm start" can be provided to the solver using the following code:
   ```
   julia> PhylogeneticTrees.warmstartunmixed(tp, timelimit=30)
   ```
   This finds the best possible tree without admixture within the time limit specified by the (optional) `timelimit` parameter (in seconds, with a default of 30 seconds). Then it loads the tree without admixture as a starting solution before attempting to add admixture.

   (b) Populations that should not be admixed can be specified using the following code:
   ```
   julia> PhylogeneticTrees.unmix(tp, "POPULATION")
   ```
   where the name of the population is provided as a `String` in the second argument to `PhylogeneticTrees.unmix()`. This name must match what is in `pd.pops`.

4. Same as without admixture

Model-fitting troubleshooting. Here are some common issues:

- **Data input.** If you see an error message saying something like `Objective Q not PSD`, then you have probably inputted your covariance matrix incorrectly. Check the formatting instructions and try again.

1.2.5 Running Scripts

If you have saved all your previous commands in a script called `script.jl`, you can save yourself some typing by using the following code to execute the commands of the script:

```
julia> include("script.jl")
```

2 Preliminaries

This section introduces notation and discusses properties of f-statistics that will be useful in developing an optimization formulation for phylogenetic inference. For more background on f-statistics, see [10].

2.1 f-Statistics

Let $\mathcal{P}$ represent a set of populations. Let X_p denote the random variable corresponding to the allele frequencies at a single polymorphism in population $p \in \mathcal{P}$. We also have a tree $\mathcal{T}$ that is composed of nodes $\mathcal{V}$ and edges $\mathcal{E}$. In general, we use the indices $o, p, q, r, s, u,$ and v to refer to specific populations.

Allele frequencies are considered to follow a *martingale property*. In particular, if the edge (p, q) is present in the graph, meaning that q is a descendant of p, then we have

$$\mathbb{E}[X_q | X_p = x] = x. \tag{1}$$

The f_2-statistic, also called *branch length*, is the squared drift. For two populations p and q, it is defined as follows:

$$F_2(p, q) = \mathbb{E}[(X_p - X_q)^2]. \tag{2}$$

It is assumed that drifts on distinct edges of the phylogenetic tree are orthogonal. Namely, for two distinct edges (p, q) and (r, s), we have

$$\mathbb{E}[(X_p - X_q)(X_r - X_s)] = 0. \tag{3}$$

This property means that f_2-statistics (branch lengths) are additive. Consider a path $p \to q \to r$ in our tree. We can show the additivity of branch lengths as follows:

$$\begin{aligned}
F_2(r, p) &= \mathbb{E}[(X_r - X_p)^2] \\
&= \mathbb{E}[(X_r - X_q + X_q - X_p)^2] \\
&= \mathbb{E}[(X_r - X_q)^2] + 2\mathbb{E}[(X_r - X_q)(X_q - X_p)] + \mathbb{E}[(X_q - X_p)^2] \\
&= \mathbb{E}[(X_r - X_q)^2] + \mathbb{E}[(X_q - X_p)^2] \\
&= F_2(r, q) + F_2(q, p).
\end{aligned}$$

The rest can be shown by induction.

The f_3-statistic, for three populations o, p, and q, is defined as follows:

$$F_3(o; p, q) = \mathbb{E}[(X_o - X_p)(X_o - X_q)]. \tag{4}$$

The f_3-statistics can also be computed as sums of the f_2 statistics by inspecting the paths from population p to population o in the tree, and similarly for population q to population o (ignoring edge direction).

$$F_3(o; p, q) = \mathbb{E}[(X_o - X_p)(X_o - X_q)]$$

$$
\begin{aligned}
&= \mathbb{E}\left[\left(\sum_{(u,v)\in\mathcal{E}} (X_u - X_v)\mathbb{1}_{\{(u,v)\in\text{path}(p,o)\}}\right)\left(\sum_{(u,v)\in\mathcal{E}} (X_u - X_v)\mathbb{1}_{\{(u,v)\in\text{path}(q,o)\}}\right)\right] \\
&= \mathbb{E}\left[\left(\sum_{(u,v)\in\mathcal{E}} (X_u - X_v)^2\mathbb{1}_{\{(u,v)\in\text{path}(p,o)\cap\text{path}(q,o)\}}\right)\right] \\
&= \sum_{(u,v)\in\mathcal{E}} \mathbb{E}\left[(X_u - X_v)^2\right]\mathbb{1}_{\{(u,v)\in\text{path}(p,o)\cap\text{path}(q,o)\}} \\
&= \sum_{(u,v)\in\mathcal{E}} F_2(u,v)\mathbb{1}_{\{(u,v)\in\text{path}(p,o)\cap\text{path}(q,o)\}}, \qquad (5)
\end{aligned}
$$

where path(p,o) indicates the sequence of edges on the path from the node containing population p to the node containing population o, ignoring edge direction. Namely, the statistic $F_3(o;p,q)$ is the sum of the branch lengths of the overlapping edges of the paths from p to o and q to o. For an illustration of equation (5), see Figure 2. In Figure 2, the edges corresponding to the intersection of path(p,o) and path(q,o) are highlighted bold and blue.

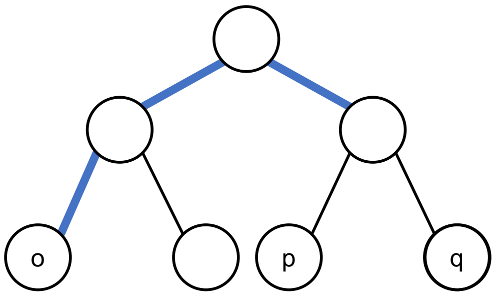


Figure 2: (Color online) An illustration of the computation of the statistic $F_3(o;p,q)$. The bold blue edges are those that contribute to the computation of $F_3(o;p,q)$, as per equation (5).

Note that $F_3(o;p,p) = F_2(o,p)$.

Typically, f_3-statistics are collected with an *outgroup* as the first argument. An outgroup is a population that split up from the ancestral populations of the other populations of study, before the population splits leading to the other populations. We reserve the index o to refer to the outgroup.

2.2 Admixture

If two populations p and q are admixed with mixing proportions α and $1-\alpha$, respectively, then the resulting allele frequency will be

$$\alpha X_p + (1-\alpha)X_q. \qquad (6)$$

Given the mixing relationship (6), we can derive the theoretical formulas for admixed populations. For example, if population p is mixed from populations q and r at frequencies α and $(1-\alpha)$ respectively, and both q and r do not experience admixture, then $F_2(o,p)$ (with outgroup o) can be calculated as follows:

$$
\begin{aligned}
F_2(o,p) &= \mathbb{E}[(\alpha X_q + (1-\alpha)X_r - X_o)^2] \\
&= \mathbb{E}[(\alpha(X_q - X_o) + (1-\alpha)(X_r - X_o))^2]
\end{aligned}
$$

$$
\begin{aligned}
&= \alpha^2\mathbb{E}[(X_q - X_o)^2] + 2\alpha(1-\alpha)\mathbb{E}[(X_q - X_o)(X_r - X_o)] + (1-\alpha)^2\mathbb{E}[(X_r - X_o)^2] \\
&= \alpha^2 F_2(o,q) + 2\alpha(1-\alpha)F_3(o;q,r) + (1-\alpha)^2 F_2(o,r) \\
&= \alpha^2 \sum_{(u',v')\in\text{path}(o,q)} F_2(u',v') \\
&\quad + 2\alpha(1-\alpha) \sum_{(u',v')\in\text{path}(o,q)\cap\text{path}(o,r)} F_2(u',v') \\
&\quad + (1-\alpha)^2 \sum_{(u',v')\in\text{path}(o,r)} F_2(u',v'). \qquad (7)
\end{aligned}
$$

We now provide a general statement on f_3-statistics with admixture. For brevity, we will refer to populations that do not experience admixture in their path to the root as "unmixed." If population p is mixed from unmixed ancestral populations $r \in \mathcal{A}_p$ at levels $\{\alpha_r\}_{r\in\mathcal{A}_p}$ ($\alpha_r \geq 0$ and $\sum_r \alpha_r = 1$) and population q is mixed from unmixed ancestral populations $s \in \mathcal{A}_q$ at levels $\{\beta_s\}_{s\in\mathcal{A}_q}$ ($\beta_s \geq 0$ and $\sum_s \beta_s = 1$), the statistic $F_3(o;p,q)$ can be computed as follows:

$$
\begin{aligned}
F_3(o;p,q) &= \mathbb{E}\left[\left(\sum_{r\in\mathcal{A}_p} \alpha_r X_r - X_o\right)\left(\sum_{s\in\mathcal{A}_s} \beta_s X_s - X_o\right)\right] \\
&= \mathbb{E}\left[\left(\sum_{r\in\mathcal{A}_p} \alpha_r (X_r - X_o)\right)\left(\sum_{s\in\mathcal{A}_s} \beta_s (X_s - X_o)\right)\right] \\
&= \sum_{r\in\mathcal{A}_p}\sum_{s\in\mathcal{A}_s} \alpha_r\beta_s \mathbb{E}\left[(X_r - X_o)(X_s - X_o)\right] \\
&= \sum_{r\in\mathcal{A}_p}\sum_{s\in\mathcal{A}_s} \alpha_r\beta_s \left(\sum_{(u,v)\in\text{path}(r,o)\cap\text{path}(s,o)} F_2(u,v)\right). \qquad (8)
\end{aligned}
$$

Note that if there is no drift, equation (8) can also represent nested layers of two-way admixtures, since any admixture topology can be represented as a mixture of multiple unmixed ancestral populations. For example, as shown in Figure 3, if population q is 50% population p and 50% population r, and population r is in turn 50% population s and 50% population u, then population q is 50% population p, 25% population s, and 25 population u.

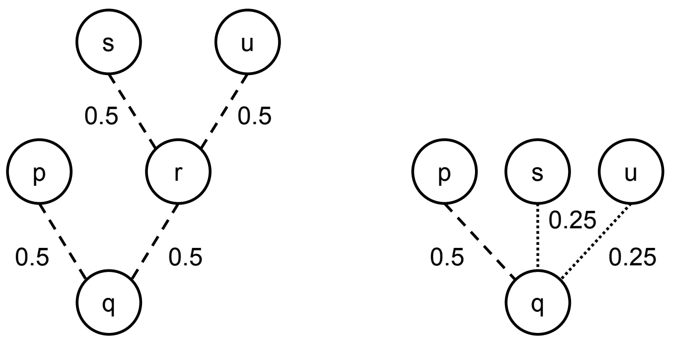


Figure 3: An illustration of nested admixture without intermediate drift. The left and right graphs are equivalent.

In our formulation, it will be more convenient to rearrange the terms of (8). An edge (u, v) is in $\mathrm{path}(r, o) \cap \mathrm{path}(s, o)$ if it satisfies certain conditions. There are two cases. First, if (u, v) *is not* on the path from the root to the outgroup, then there must exist ancestral populations r and s that are both in the subtree rooted at v, which we call $\mathcal{S}_v$. Second, if (u, v) *is* on the path from the root to the outgroup, then there must exist ancestral populations r and s that are both outside of $\mathcal{S}_v$. Therefore, equation (8) is equivalent to the following:

$$F_3(o; p, q) = \sum_{(u,v)\in\mathcal{E}\setminus\mathrm{path}(\mathrm{root},o)} F_2(u, v) \sum_{r\in\mathcal{A}_p\cap\mathcal{S}_v} \sum_{s\in\mathcal{A}_q\cap\mathcal{S}_v} \alpha_r\beta_s + \sum_{(u,v)\in\mathrm{path}(\mathrm{root},o)} F_2(u, v) \sum_{r\in\mathcal{A}_p\setminus\mathcal{S}_v} \sum_{s\in\mathcal{A}_q\setminus\mathcal{S}_v} \alpha_r\beta_s. \tag{9}$$

Note that if p and q each only have one ancestor, equation (9) reduces to equation (5), but rearranged in our more convenient format for our eventual formulation. For an illustration of equation (9), see Figure 4. Figure 4 highlights in bold colors the relevant edges for computation of the f_3 statistic, with green denoting the $(u, v) \in \mathcal{E} \setminus \mathrm{path}(\mathrm{root}, o)$ case, and with blue denoting the $(u, v) \in \mathrm{path}(\mathrm{root}, o)$ case.

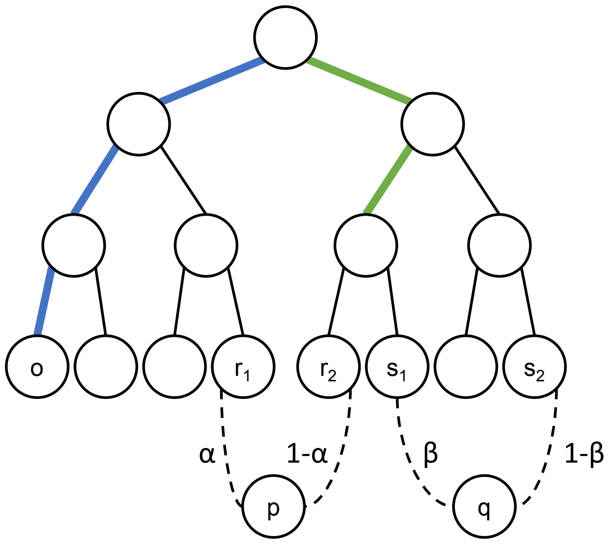


Figure 4: (Color online) An illustration of the computation of the statistic $F_3(o; p, q)$ in the presence of admixture (dashed edges leading to populations p and q). The bold colored edges are those that contribute to the computation of $F_3(o; p, q)$, as per equation (9). Green edges denote the first case, when edges are not on the path from the root to the outgroup. Blue edges denote the second case, when edges are on the path from the root to the outgroup.

3 Integer Optimization Formulation

We can now formulate a mixed-integer quadratic optimization problem to solve for an optimal tree. We first study the case without admixture in Section 3.1, and then generalize to admixed populations in Section 3.2.

3.1 Without Admixture

3.1.1 A Motivating Example

Phylogenetic inference can be framed as the problem of finding a graph topology and set of edge weights that best fit a set of given f_3-statistics. The graph topology itself can be seen as the assignment of each population to leaf nodes of a binary tree.

Given a topology, it is straightforward to compute weights that best fit the given f_3-statistics using a tool such as `qpGraph` [10]. A dedicated practitioner could conceivably propose a topology, then compute weights to fit the f_3-statistics using `qpGraph`, and iterate through all possible topologies to find the best one. However, we can dispense with this iteration by formulating a mathematical model incorporating topology and weights together, and using optimization solvers to efficiently search the space of potential topologies and weights.

Before discussing the formulation in detail, we will provide an illustrative example based on Figure 5, which involves the populations Chimp, Modern African, and Neanderthal.

Our approach will require that we first determine an appropriate depth for a binary tree to hold our populations. A binary tree of depth 2, which is shown on the left hand side of Figure 5, contains four leaf nodes (Nodes 4-7), and so is sufficiently large to hold the three populations. Larger trees could also be considered, but are unnecessary for this example. Note that there is a distinction between *populations* (i.e., Chimp, Modern African, and Neanderthal) and general *nodes* (i.e., Nodes 1-7) in a tree structure. The latter solely indicate positions on trees, and these positions might or might not eventually be associated with populations. We will restrict the assignment of populations to leaf nodes.

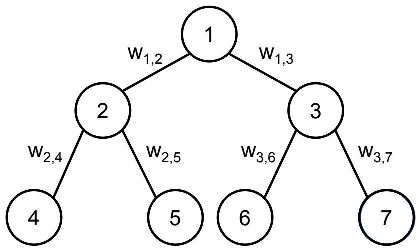
On the right hand table of Figure 5, we use shaded cells to show a set of population-node assignments that matches conventional understanding of these populations, with Chimp in Node 4, Modern African in Node 6, and Neanderthal in Node 7. Node 5 remains empty. This sort of assignment, along with the edge weights $\mathbf{w}$, comprise the output of our optimization model.

Our approach, which decides whether to assign each population to each node, differs from prior works such as [12], [4], and [7], which use path-based formulations that decide whether or not to draw edges between nodes in a graph subject to constraints that the resulting graph must have tree structure. Although the path-based formulations can be solved efficiently, they cannot accommodate admixture, which we will discuss in Section 3.2.

In addition to the pre-specification of the depth of the tree, our approach will also require that the user designate one population as the outgroup, and manually assign the outgroup to a node, so that this information is known before the optimization. The designation of the outgroup is consistent with standard practice [10], as it facilitates computation of the f_3-statistics, which are measured relative to the outgroup. The assignment also breaks symmetries in the optimization problem.

Detailed readers might observe that as a result of the outgroup's pre-assignment, the outgroup is more constrained than the other populations in the tree. This is by design, and the tree's symmetry prevents this choice from influencing the final results. To see this, first suppose the Chimp is taken as the outgroup, and assigned to Node 4. In addition to the Chimp's assignment to Node 4, the Modern African is assigned to

Node 6, and the Neanderthal is assigned to Node 7, as in Figure 5. However, an equivalent permutation of the tree could assign the Chimp to Node 6, the Modern African to Node 4, and the Neanderthal to Node 5. Indeed, regardless of where the outgroup is placed, the resulting tree can be permuted accordingly to produce the original assignment. We will follow a convention of always placing the outgroup in the leftmost leaf node.



	Node 4	Node 5	Node 6	Node 7
Chimp	🟦			
Modern African			🟦	
Neanderthal				🟦

Figure 5: (Color online) A small illustrative example without admixture, involving the assignment of Chimp, Modern African, and Neanderthal populations to the leaf nodes of a binary tree. The shaded cells in the table indicate the node assignments.

3.1.2 Formulation

Consistent with Section 2, we will use the indices o, p and q to refer to specific populations (e.g., Chimp, Modern African, Neanderthal in Figure 5) and their associated statistics. We further use the indices i and j, and n to refer to nodes (e.g., Nodes 1-7 in Figure 5). The population index o is reserved for the pre-specified outgroup population, and the node index n is reserved for the pre-specified outgroup node.

Input Data and Parameters We begin with the following **input data**:

- $\mathcal{P}$: A set of populations
- $o \in \mathcal{P}$: An outgroup
- $\hat{f}_{p,q}$: f_3-statistics $F_3(o; p, q)$ measured relative to the outgroup.

We also specify the following tree parameters:

- $D \in \mathbb{Z}_+$: Desired depth of the tree (which will have nodes $1, 2, \ldots, 2^{D+1} - 1$)
- $n = 2^D$: The node at which to fix the outgroup.

The depth of the tree should be chosen large enough so that the populations $\mathcal{P}$ can comfortably fit in the 2^D leaf nodes. The choice of $n = 2^D$ is arbitrary; any leaf node $2^D, 2^D + 1, \ldots, 2^{D+1} - 1$ would suffice. The set of nodes is denoted by $\mathcal{V} := \{1, 2, \ldots, 2^{D+1} - 1\}$, and the set of all (undirected) edges is denoted by $\mathcal{E} := \{(i, j) \mid i \in \mathcal{V}, j \in \mathcal{V}, j \in \{2i, 2i + 1\}\}$.

Decision Variables Our optimization problem will have the following **main decision variables** that parametrize the tree:

- $x_{p,i} \in \{0, 1\}$: 1 if population p is assigned to node i, 0 otherwise. We only include nodes i that are leaves of the tree, which we will refer to as $\mathcal{V}_D := \{2^D, \ldots, 2^{D+1} - 1\}$. The root node is labeled 1.

- $w_{i,j} \geq 0$: branch length on edge (i,j). This is also the f_2-statistic of equation (5).

Given a particular topology, parametrized by the main variables $\mathbf{x}$ and $\mathbf{w}$, we must compute the associated f_3-statistics. For this task, we use the following **auxiliary decision variables** to determine which edges contribute to computation of the f_3-statistics:

- $y_{p,q,(i,j)} \geq 0$: although not explicitly defined to be binary, will be constrained to take on the value 1 if edge (i,j) contributes to the $F_3(o;p,q)$ statistic, and 0 otherwise.
- $z_{p,q,(i,j)} \geq 0$: the branch length on edge (i,j) if it contributes to $F_3(o;p,q)$, and 0 otherwise.

Finally, the f_3-statistics and errors will be computed using the following **fitting decision variables**:

- $f_{p,q} \geq 0$: the fitted value of $F_3(o;p,q)$
- $\epsilon_{p,q}$: the error associated with fitting $F_3(o;p,q)$.

Constraints We now turn to the constraints necessary to model the problem. First, the outgroup must be assigned to the proper node:

$$x_{o,n} = 1. \tag{10}$$

Recall that o and n are known data and parameters, not optimized.

Each population is assigned to a leaf node using the following constraint:

$$\sum_{i \in \mathcal{V}_D} x_{p,i} = 1 \quad \forall p \in \mathcal{P}. \tag{11}$$

Similarly, each leaf node is assigned at most one population as follows:

$$\sum_{p \in \mathcal{P}} x_{p,i} \leq 1 \quad \forall i \in \mathcal{V}_D. \tag{12}$$

Using $\mathcal{S}_j$ to denote the leaf nodes that are in the subtree rooted at j, and recalling equation (9), we use the following constraints to link the auxiliary $\mathbf{y}$ and main $\mathbf{x}$ variables, so that the population-node assignment induces the correct computation of the f_3-statistics between populations:

$$y_{p,q,(i,j)} \leq \sum_{j' \in \mathcal{S}_j} x_{p,j'} \tag{13a}$$

$$y_{p,q,(i,j)} \leq \sum_{j' \in \mathcal{S}_j} x_{q,j'} \tag{13b}$$

$$y_{p,q,(i,j)} \geq \sum_{j' \in \mathcal{S}_j} (x_{p,j'} + x_{q,j'}) - 1 \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}, (i,j) \in \mathcal{E} \setminus \mathrm{path}(n,1), \tag{13c}$$

for edges not in the path from the outgroup node to the root, and

$$y_{p,q,(i,j)} \leq 1 - \sum_{j' \in \mathcal{S}_j} x_{p,j'} \tag{14a}$$

$$y_{p,q,(i,j)} \leq 1 - \sum_{j' \in \mathcal{S}_j} x_{q,j'} \tag{14b}$$

$$y_{p,q,(i,j)} \geq 1 - \sum_{j' \in \mathcal{S}_j} (x_{p,j'} + x_{q,j'}) \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}, (i,j) \in \text{path}(n, 1). \tag{14c}$$

for edges in the path from the outgroup node to the root. Constraints (13) and (14) are intended to express that for a particular edge (i, j), whether populations p and q are assigned to particular portions of the tree (as represented by $\mathbf{x}$) then determines whether that edge contributes to the calculation of $F_3(o; p, q)$. Note that constraints (13) and (14) are exact even without explicitly enforcing that the $\mathbf{y}$ variables be binary; if the $\mathbf{y}$ variables are constrained to be nonnegative then they will naturally only take 0-1 values.

With the $\mathbf{y}$ variables set to the appropriate values, the theoretical $F_3(o; p, q)$ statistic can be computed as follows:

$$f_{p,q} = \sum_{(i,j) \in \mathcal{E}} w_{i,j} y_{p,q,(i,j)} \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}, \tag{15}$$

which is bilinear. However, because the $\mathbf{y}$ variables only take 0-1 values, we can set auxiliary variables $z_{p,q,(i,j)}$ to take on the values of $w_{i,j} y_{p,q,(i,j)}$ using the McCormick relaxation:

$$z_{p,q,(i,j)} \leq M y_{p,q,(i,j)} \tag{16a}$$

$$z_{p,q,(i,j)} \leq w_{i,j} \tag{16b}$$

$$z_{p,q,(i,j)} \geq w_{i,j} + M y_{p,q,(i,j)} - M \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}, (u,v) \in \mathcal{E}, \tag{16c}$$

where M is an upper bound representing the largest possible edge weight. Note that if $y_{p,q,(i,j)} = 1$, these constraints reduce to $z_{p,q,(i,j)} \leq M, z_{p,q,(i,j)} \leq w_{i,j}, z_{p,q,(i,j)} \geq w_{i,j}$, thereby inducing $z_{p,q,(i,j)}$ to be equal to $w_{p,q,(i,j)}$. By contrast, if $y_{p,q,(i,j)} = 0$, these constraints reduce to $z_{p,q,(i,j)} \leq 0, z_{p,q,(i,j)} \leq w_{i,j}, z_{p,q,(i,j)} \geq w_{i,j} - M$, thereby inducing $z_{p,q,(i,j)}$ to take the value 0. It then remains to pick a suitable upper bound M based on our input data. We choose $M = \Gamma \max_{p,q} \hat{f}_{p,q}$, with $\Gamma \geq 1$. Then, we can replace constraint (15) with the linear constraint

$$f_{p,q} = \sum_{(i,j) \in \mathcal{E}} z_{p,q,(i,j)} \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}. \tag{17}$$

Finally, we can set the error terms using the following constraint:

$$\epsilon_{p,q} = \hat{f}_{p,q} - f_{p,q} \tag{18}$$

We seek to minimize the error (weighted by the inverse covariance matrix):

$$\min_{\mathbf{x},\mathbf{y},\mathbf{w},\mathbf{g},\boldsymbol{\epsilon}} \sum_{p \in \mathcal{P}, q \in \mathcal{P}} \sum_{p' \in \mathcal{P}, q' \in \mathcal{P}} \Sigma^{-1}_{p,q,p',q'} \epsilon_{p,q} \epsilon_{p',q'}, \tag{19}$$

where $\Sigma^{-1}_{p,q,p',q'}$ is the entry in the inverted covariance matrix corresponding to $F_3(o; p, q)$ and $F_3(o; p', q')$.

3.1.3 Example Variable Values and Constraints

To illuminate the formulation in Section 3.1.2, we return to the example in Figure 5. For this example, we provide values for relevant $\mathbf{x}$ and $\mathbf{y}$ variables under the assignment in the figure, and show these values' consistency with a selection of constraints. We will use the abbreviations "CP" for "Chimp", "MA" for

"Modern African", and "NT" for "Neanderthal".

Given the population-node assignments in the table of Figure 5, the optimal $\mathbf{x}$ values are as follows:

- $x_{\mathrm{CP},4} = 1$,
- $x_{\mathrm{MA},6} = 1$,
- $x_{\mathrm{NT},7} = 1$,
- and $x_{p,i} = 0$ for all other $(p, i) \in \mathcal{P} \times \mathcal{V}_D$ not listed.

It is straightforward to verify that these given $\mathbf{x}$ values satisfy constraints (10) through (12), recalling that the Chimp is the outgroup and that we take the convention $n = 2^D = 4$.

We now turn to the $\mathbf{y}$ variables, which are needed for computation of the f_3-statistics. The given $\mathbf{x}$ values, combined with constraints (13) and (14), induce the following $\mathbf{y}$ values:

- Computation of $F_3(\mathrm{CP}; \mathrm{MA}, \mathrm{MA})$:
  - $y_{\mathrm{MA},\mathrm{MA},(3,6)} = 1$,
  - $y_{\mathrm{MA},\mathrm{MA},(1,3)} = 1$,
  - $y_{\mathrm{MA},\mathrm{MA},(1,2)} = 1$,
  - $y_{\mathrm{MA},\mathrm{MA},(2,4)} = 1$,
  - and $y_{\mathrm{MA},\mathrm{MA},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ not listed.
- Computation of $F_3(\mathrm{CP}; \mathrm{MA}, \mathrm{NT})$:
  - $y_{\mathrm{MA},\mathrm{NT},(1,3)} = 1$,
  - $y_{\mathrm{MA},\mathrm{NT},(1,2)} = 1$,
  - $y_{\mathrm{MA},\mathrm{NT},(2,4)} = 1$,
  - and $y_{\mathrm{MA},\mathrm{NT},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ not listed.
- Computation of $F_3(\mathrm{CP}; \mathrm{NT}, \mathrm{NT})$:
  - $y_{\mathrm{NT},\mathrm{NT},(3,7)} = 1$,
  - $y_{\mathrm{NT},\mathrm{NT},(1,3)} = 1$,
  - $y_{\mathrm{NT},\mathrm{NT},(1,2)} = 1$,
  - $y_{\mathrm{NT},\mathrm{NT},(2,4)} = 1$,
  - and $y_{\mathrm{NT},\mathrm{NT},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ not listed.

To see how the above $\mathbf{x}$ and $\mathbf{y}$ values satisfy constraints (13) and (14), we write out the constraints in full for $p = \mathrm{MA}$, $q = \mathrm{NT}$, and $(i, j) \in \{(1, 2), (3, 7)\}$, and leave the remainder as an exercise for the reader.

First, recall that the outgroup was manually assigned to Node 4 (see constraint (10) with $o = \mathrm{CP}$, $n = 2^D = 4$, which were pre-determined parameters and input data rather than optimized indices). An optimized assignment of the other populations to nodes was then computed relative to the outgroup's position. Because the outgroup's location is known before the optimization, we know that edge $(1, 2)$ does lie in the path from the outgroup node to the root, and so constraints (14) apply. Written in full, these constraints are:

$$y_{\mathrm{MA},\mathrm{NT},(1,2)} \leq 1 - x_{\mathrm{MA},4} - x_{\mathrm{MA},5}$$

$$y_{\mathrm{MA,NT},(1,2)} \leq 1 - x_{\mathrm{NT},4} - x_{\mathrm{NT},5}$$
$$y_{\mathrm{MA,NT},(1,2)} \geq 1 - x_{\mathrm{MA},4} - x_{\mathrm{MA},5} - x_{\mathrm{NT},4} - x_{\mathrm{NT},5},$$

and plugging in the given **x** and **y** values satisfies the constraints $(1 \leq 1, 1 \leq 1, 1 \geq 1)$. Consistent with Figure 2, the value of $y_{\mathrm{MA,NT},(1,2)}$ is 1, and the edge $(1,2)$ correctly contributes to the calculation of $F_3(\mathrm{CP};\mathrm{MA},\mathrm{NT})$ in the given solution topology.

For the case $(i,j) = (3,7)$, we apply constraints (13), recalling that the outgroup's location is known before the optimization and that edge $(3,7)$ does not lie in the path from the outgroup node to the root. Written in full, the constraints are:

$$y_{\mathrm{MA,NT},(3,7)} \leq x_{\mathrm{MA},7}$$
$$y_{\mathrm{MA,NT},(3,7)} \leq x_{\mathrm{NT},7}$$
$$y_{\mathrm{MA,NT},(3,7)} \geq x_{\mathrm{MA},7} + x_{\mathrm{NT},7} - 1,$$

and plugging in the given **x** and **y** values satisfies the constraints $(0 \leq 0, 0 \leq 1, 0 \leq 0)$. Consistent with Figure 2, the value of $y_{\mathrm{MA,NT},(3,7)}$ is 0, and the edge $(3,7)$ correctly does not contribute to the calculation of $F_3(\mathrm{CP};\mathrm{MA},\mathrm{NT})$ in the given solution topology.

In summary, we have related the assignment of Chimp to Node 4, Modern African to Node 6, and Neanderthal to Node 7 to the corresponding **x** values, which, in accordance with constraints on the remaining variables, correctly compute f_3-statistics based on the topological relationship between the populations.

3.2 With Admixture

3.2.1 A Motivating Example

We first motivate the problem of admixture with an example. In Figure 5, we show population-node assignments that reflect conventional understanding of the relationships between Neanderthal, Non-African, and Modern African populations. Unlike the example of Figure 5, this new example includes a fractional assignment of the Non-African population across Node 5 and Node 6, indicating that it is produced from admixture between populations in those nodes. The darker shading for Node 6 indicates that the Non-African population is assigned mostly to Node 6, with a smaller contribution from Node 5. The Neanderthal population, which is assigned to Node 4, and the Modern African population, which is assigned to Node 7, are not associated with admixture.

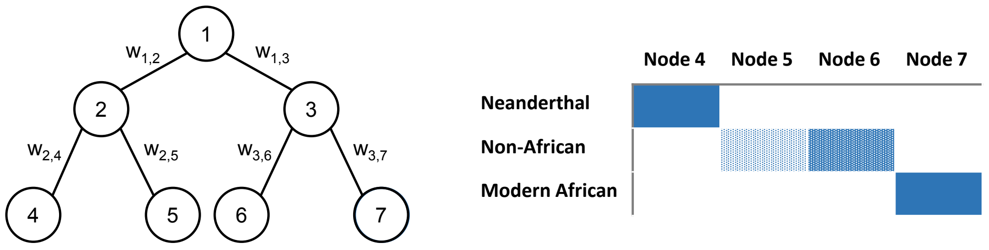


Figure 6: (Color online) A small illustrative example with admixture, involving the assignment of Neanderthal, Non-African, and African populations to the leaf nodes of a binary tree.

3.2.2 Formulation

As outlined in the Figure 6 example, an option for modeling admixture might be to relax the $\mathbf{x}$ variables to be continuous rather than binary. With continuous $\mathbf{x}$ variables, additional auxiliary variables and constraints would be needed to prevent more than one population from being assigned to each node, which is not difficult. However, the real stumbling point is that $\mathbf{y}$ variables could then take on continuous values in $[0, 1]$, and therefore the linearization that produced constraints (13) and (14) would no longer be exact. We have seen computationally that the big-M bounds are not tight enough in the linear relaxation to produce reliable results.

Rather than relax the $\mathbf{x}$ variables to be continuous, we approximate the continuous admixture proportions by choosing a suitably large $K \in \mathbb{Z}_+$, and introducing the following sum of auxiliary binary variables $\boldsymbol{\chi} \in \{0,1\}^{|\mathcal{P}| \times 2^D \times K}$:

$$\text{admixure proportion } (p, i) = \frac{1}{K} \sum_{k=1}^{K} \chi_{p,i}^k. \tag{20}$$

Under this framework, there are K levels at which each population can be assigned to nodes, enabling fractional assignment that can accommodate the example in Figure 5. Each variable $\chi_{p,i}^k$ will indicate whether population p is assigned to node i, for $k = 1, \dots, K$. The summation in equation (20) then represents the proportion of admixture for population p arising from the ancestral population in node i. As such, a population can be assigned to anywhere between one and K nodes, inclusive; assignment to one node corresponds to an admixture proportion of 1, while assignment to K nodes indicates that the population is produced from an equal mixture of the K nodes at a proportion of $\frac{1}{K}$ each. Higher levels of K allow for more granularity in the approximation of the admixture proportions; for example, if $K = 10$, then the admixture proportions can take on values $0, 0.1, \dots, 0.9, 1$. Note that equation (20) is intended as an approximation to the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ terms in equation (9).

This model of admixture assigns each population p to *one or more* leaf nodes, where the level of assignment of population p to node i is given in the definition (20). As such, constraint (11), which had ensured that each population was assigned to *exactly one* leaf node, are no longer valid, because each population may be assigned to up to K leaf nodes. First, the following additional constraint is needed to supplement constraint (10) and ensure that the outgroup is fully assigned to the proper node:

$$\chi_{o,n}^k = 1 \quad \forall k = 1, \dots, K, . \tag{21}$$

Then, constraint (11) is entirely replaced with the following constraints on the new $\boldsymbol{\chi}$ variables:

$$\sum_{i \in \mathcal{V}_D} \chi_{p,i}^k = 1 \quad \forall p \in \mathcal{P},\ k = 1, \dots, K. \tag{22}$$

Constraint (22) ensures that each of the K levels of population p should be assigned to exactly one node.

The $\mathbf{x}$ and $\boldsymbol{\chi}$ variables need to be linked as follows:

$$x_{p,i} \geq \chi_{p,i}^k \quad \forall p \in \mathcal{P},\ i \in \mathcal{V}_D,\ k \in 1, \dots, K, \tag{23}$$

which ensures that if $\chi_{p,i}^k = 1$ for any $k = 1, \dots, K$, then $x_{p,i} = 1$ as well.

Constraint (12), which ensures that each node is assigned at most one population, is still valid. Furthermore, by upper bounding the $\boldsymbol{\chi}$ variables with the $\mathbf{x}$ variables, any equality and upper bound constraints on $\mathbf{x}$ will apply on the relevant $\boldsymbol{\chi}$ variables as well.

Additionally, if we want to restrict the number of admixture events to be at most some number A, we can add the following constraint:

$$\sum_{p\in\mathcal{P}}\sum_{i\in\mathcal{V}_D} x_{p,i} \leq |\mathcal{P}| + A. \tag{24}$$

To understand equation (24), first consider the case $A = 0$. In that case, the number of node assignments, denoted by the summation over the $\mathbf{x}$ variables, should be at most the number of populations. Combined with constraint (23), this means that each population can be assigned to at most one node, and therefore, no populations will be mixed between nodes. The case $A = 1$ will allow at most one population to be assigned to at most two nodes, representing a single admixture event. More generally, the parameter A accommodates admixture by allowing the populations to be spread out over an additional A nodes, in addition to the $|\mathcal{P}|$ nodes that they must occupy in the tree.

Because the new $\boldsymbol{\chi}$ variables are binary, we can replace the auxiliary variables $\mathbf{y}$ with auxiliary variables $\boldsymbol{\psi} \in [0,1]^{|\mathcal{P}|^2\times|\mathcal{E}|\times K^2}$, which will help calculate the $F_3(o;p,q)$ statistics according to equation (9). For each variable $\psi^{k,\ell}_{p,q,(i,j)}$, index k will be associated with population p, and index ℓ will be associated with population q. Then, analogous to constraint (13), for edges not in the path from the outgroup node to the root, we have the following constraints linking $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$:

$$\psi^{k,\ell}_{p,q,(i,j)} \leq \sum_{j'\in\mathcal{S}_j} \chi^k_{p,j'} \tag{25a}$$
$$\psi^{k,\ell}_{p,q,(i,j)} \leq \sum_{j'\in\mathcal{S}_j} \chi^\ell_{q,j'} \tag{25b}$$
$$\psi^{k,\ell}_{p,q,(i,j)} \geq \sum_{j'\in\mathcal{S}_j} \left(\chi^k_{p,j'} + \chi^\ell_{q,j'}\right) - 1 \quad \forall p\in\mathcal{P}, q\in\mathcal{P}, (i,j)\in\mathcal{E}\setminus\text{path}(n,1), k=1,\dots,K, \ell=1,\dots,K. \tag{25c}$$

Similarly, analogous to constraint (14), for edges in the path from the outgroup node to the root, we have the following constraints linking $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$:

$$\psi^{k,\ell}_{p,q,(i,j)} \leq 1 - \sum_{j'\in\mathcal{S}_j} \chi^k_{p,j'} \tag{26a}$$
$$\psi^{k,\ell}_{p,q,(i,j)} \leq 1 - \sum_{j'\in\mathcal{S}_j} \chi^\ell_{q,j'} \tag{26b}$$
$$\psi^{k,\ell}_{p,q,(i,j)} \geq 1 - \sum_{j'\in\mathcal{S}_j} \left(\chi^k_{p,j'} + \chi^\ell_{q,j'}\right) \quad \forall p\in\mathcal{P}, q\in\mathcal{P}, (i,j)\in\text{path}(n,1), k=1\dots,K, \ell=1,\dots,K. \tag{26c}$$

As before, since the $\boldsymbol{\chi}$ variables are binary, the $\boldsymbol{\psi}$ variables will also take only 0-1 values despite not explicitly being constrained to be binary.

Then, substituting the expanded variables into constraint (9), we obtain the following:

$$f_{p,q} = \frac{1}{K^2}\sum_{(i,j)\in\mathcal{E}} w_{i,j}\sum_{k=1}^{K}\sum_{\ell=1}^{K}\psi^{k,\ell}_{p,q,(i,j)}, \tag{27}$$

which is the K-level modification of constraint (15). Note that the summation over $\psi^{k,\ell}_{p,q,(i,j)}/K^2$ terms captures the facets of equation (9) via (i) the K-level approximation of the product of p and q's admixture proportions, thereby producing K^2 terms, (ii) the distinction of whether (i,j) is in the path between the root and the outgroup or not by separating constraints (25) and (26), and (iii) the summations in constraints (25) and (26) over the subtrees rooted at j, $\mathcal{S}_j$.

As before, constraint (27) can be linearized again with a McCormick relaxation, since the $\boldsymbol{\psi}$ variables only take 0-1 values. Analogous to constraints (16), we define auxiliary variables $\zeta^{k,\ell}_{p,q,(i,j)}$ that will take on the values of the product $w_{i,j}\psi^{k,\ell}_{p,q,(i,j)}$ with the addition of the following constraints:

$$\zeta^{k,\ell}_{p,q,(i,j)} \leq M\psi^{k,\ell}_{p,q,(i,j)} \tag{28a}$$

$$\zeta^{k,\ell}_{p,q,(i,j)} \leq w_{i,j} \tag{28b}$$

$$\zeta^{k,\ell}_{p,q,(i,j)} \geq w_{i,j} + M\psi^{k,\ell}_{p,q,(i,j)} - M \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}, (u,v) \in \mathcal{E}, k = 1,\ldots,K, \ell = 1,\ldots,K, \tag{28c}$$

where M is the data-computed upper bound representing the largest possible edge weight. Then, we can replace constraint (27) with the linear constraint

$$f_{p,q} = \frac{1}{K^2} \sum_{(i,j)\in\mathcal{E}} \sum_{k=1}^{K} \sum_{\ell=1}^{K} \zeta^{k,\ell}_{p,q,(i,j)} \qquad \forall p \in \mathcal{P}, q \in \mathcal{P}. \tag{29}$$

The constraint (18) and objective (19) remain the same.

This model has $O(|\mathcal{P}|^2 2^D K^2)$ auxiliary variables, and in the case where $K = 1$, it reduces to the formulation without admixture.

3.2.3 Example Variable Values and Constraints

To illuminate the formulation in Section 3.2.2, we return to the example in Figure 6. For this example, we provide values for relevant $\mathbf{x}$, $\boldsymbol{\chi}$, and $\boldsymbol{\psi}$ variables under the assignment in the figure, and show these values' consistency with a selection of constraints. We will use the abbreviations "NT" for "Neanderthal", "NA" for "Non-African", and "MA" for "Modern African". The Neanderthal is the outgroup.

Given the population-node assignments in the table of Figure 6, the optimal $\mathbf{x}$ values are as follows:

- $x_{\text{NT},4} = 1$,
- $x_{\text{NA},5} = 1$,
- $x_{\text{NA},6} = 1$,
- $x_{\text{MA},7} = 1$,
- and $x_{p,i} = 0$ for all other $(p,i) \in \mathcal{P} \times \mathcal{V}_D$ not listed.

Note that in this example, the Non-African population is assigned to two nodes (Nodes 6 and 7), which is allowed, having removed constraint (11). With the given $\mathbf{x}$ values, the summation on the left-hand side of constraint (24) is equal to four, while the number of populations is three. As such, in order to allow assignment with admixture, the input parameter A would need to be set to be greater than or equal to one.

We now turn to the $\boldsymbol{\chi}$ variables, which are the K-level generalization of the $\mathbf{x}$ variables. Supposing that the first of the K levels of the Non-African population is in Node 5, and the remaining levels 2 through K are in Node 6, we have the following $\boldsymbol{\chi}$ values:

- $\chi^k_{\mathrm{NT},4} = 1$ for $k = 1, \dots, K$,
- $\chi^1_{\mathrm{NA},5} = 1$,
- $\chi^k_{\mathrm{NA},6} = 1$ for $k = 2, \dots, K$,
- $\chi^k_{\mathrm{MA},7} = 1$ for $k = 1, \dots, K$,
- and $\chi^k_{p,i} = 0$ for all other $(p, i) \in \mathcal{P} \times \mathcal{V}_D$ and $k \in \{1, \dots, K\}$ not listed.

First, these $\boldsymbol{\chi}$ values enforce that the Non-African population is made up of $\frac{1}{K}$ of a population represented by Node 5, and $\frac{K-1}{K}$ of a population represented by Node 6. Second, it is straightforward to verify that these $\mathbf{x}$ and $\boldsymbol{\chi}$ values satisfy constraints (10) (recalling that $o = \mathrm{NT}$ and $n = 2^D = 4$), (21), (22), and (23).

We now turn to the $\boldsymbol{\psi}$ variables, which are needed for computation of the f_3-statistics. The given $\boldsymbol{\chi}$ values, combined with constraints (25) and (26), induce the following $\boldsymbol{\chi}$ values:

- Computation of $F_3(\mathrm{NT}; \mathrm{NA}, \mathrm{NA})$:
  - $\psi^{k,\ell}_{\mathrm{NA},\mathrm{NA},(3,6)} = 1$ for $k = 2, \dots, K, \ell = 2, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{NA},(1,3)} = 1$ for $k = 2, \dots, K, \ell = 2, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{NA},(1,2)} = 1$ for $k = 1, \dots, K, \ell = 1, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{NA},(2,4)} = 1$ for $k = 1, \dots, K, \ell = 1, \dots, K$,
  - $\psi^{1,1}_{\mathrm{NA},\mathrm{NA},(2,5)} = 1$,
  - and $y_{\mathrm{NA},\mathrm{NA},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ and $k \in \{1, \dots, K\}$ not listed.
- Computation of $F_3(\mathrm{NT}; \mathrm{NA}, \mathrm{MA})$:
  - $\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(3,7)} = 1$ for $k = 2, \dots, K, \ell = 1, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{MA},(1,3)} = 1$ for $k = 2, \dots, K, \ell = 1, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{MA},(1,2)} = 1$ for $k = 2, \dots, K, \ell = 1, \dots, K$,
  - $y^{k,\ell}_{\mathrm{NA},\mathrm{MA},(2,4)} = 1$ for $k = 1, \dots, K, \ell = 1, \dots, K$,
  - and $y_{\mathrm{NA},\mathrm{MA},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ and $k \in \{1, \dots, K\}$ not listed.
- Computation of $F_3(\mathrm{NT}; \mathrm{MA}, \mathrm{MA})$:
  - $\psi^{k,\ell}_{\mathrm{MA},\mathrm{MA},(3,7)} = 1$ for $k = 1, \dots, K$, $\ell = 1, \dots, K$,
  - $\psi^{k,\ell}_{\mathrm{MA},\mathrm{MA},(1,3)} = 1$ for $k = 1, \dots, K$, $\ell = 1, \dots, K$,
  - $\psi^{k,\ell}_{\mathrm{MA},\mathrm{MA},(1,2)} = 1$ for $k = 1, \dots, K$, $\ell = 1, \dots, K$,
  - $\psi^{k,\ell}_{\mathrm{MA},\mathrm{MA},(2,4)} = 1$ for $k = 1, \dots, K$, $\ell = 1, \dots, K$,
  - and $\psi^{k,\ell}_{\mathrm{MA},\mathrm{MA},(i,j)} = 0$ for all other $(i, j) \in \mathcal{E}$ and $k \in \{1, \dots, K\}$ not listed.

To see how the above $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$ values satisfy constraints (25) and (26), we write out the constraints for $p = \mathrm{NA}$, $q = \mathrm{MA}$, and $(i, j) \in \{(1, 2), (2, 5)\}$, and leave the remainder as an exercise for the reader.

First, recall that the outgroup was manually assigned to Node 4 (see constraint (10) with $o = \mathrm{NT}$, $n = 2^D = 4$, which were pre-determined parameters and input data rather than optimized indices). An optimized assignment of the other populations to nodes was then computed relative to the outgroup's position. Because the outgroup's location is known before the optimization, we know that edge $(1, 2)$ does lie in the path from the outgroup node to the root, and so constraints (26) apply. Written in full, these constraints are:

$$\begin{aligned}
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(1,2)} &\leq 1 - \chi^{k}_{\mathrm{NA},4} - \chi^{k}_{\mathrm{NA},5}\\
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(1,2)} &\leq 1 - \chi^{\ell}_{\mathrm{MA},4} - \chi^{\ell}_{\mathrm{MA},5}\\
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(1,2)} &\geq 1 - \chi^{k}_{\mathrm{NA},4} - \chi^{k}_{\mathrm{NA},5} - \chi^{\ell}_{\mathrm{MA},4} - \chi^{\ell}_{\mathrm{MA},5},
\end{aligned}$$

and plugging in the given $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$ values satisfies the constraints so that edge $(1, 2)$ contributes to the calculation of $F_3(\mathrm{NT}; \mathrm{NA}, \mathrm{MA})$ at fractional value $\frac{K-1}{K}$ in the given solution topology.

For the case $(i, j) = (2, 5)$, we apply constraints (25), recalling that the outgroup's location is known before the optimization and that edge $(2, 5)$ does not lie in the path from the outgroup node to the root. Written in full, the constraints are:

$$\begin{aligned}
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(2,5)} &\leq \chi^{k}_{\mathrm{NA},5}\\
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(2,5)} &\leq \chi^{\ell}_{\mathrm{MA},5}\\
\psi^{k,\ell}_{\mathrm{NA},\mathrm{MA},(2,5)} &\geq \chi^{k}_{\mathrm{NA},5} - \chi^{\ell}_{\mathrm{MA},5} - 1,
\end{aligned}$$

and plugging in the given $\boldsymbol{\chi}$ and $\boldsymbol{\psi}$ values satisfies the constraints so that edge $(2, 5)$ does not contribute to the calculation of $F_3(\mathrm{NT}; \mathrm{NA}, \mathrm{MA})$ in the given solution topology.

In summary, we have related the assignment of Neanderthal to Node 4, Non-African to Node 5 (for $k = 1$) and Node 6 (for $k = 2, \dots, K$), and Modern African to Node 7 to the corresponding $\mathbf{x}$, $\boldsymbol{\chi}$, and $\boldsymbol{\psi}$ values, which, in accordance with constraints on the remaining variables, correctly compute f_3-statistics based on the topological relationship between the populations.

4 Simulated Data

We used the msprime coalescent simulator [8] to simulate chromosomes (excluding the autosomes) at their relevant lengths from a set of eight populations with different topologies. We used a mutation rate of 1.5×10^{-8}, a recombination rate of 1×10^{-8}, and Ne of 500, and sampled 20 individuals from each population. To reflect the analysis performed with real data, we then created 40 haploid chromosomes from these 20 diploid sequences. We then computed empirical f-statistics and their accompanying covariances using a weighted block jackknife [10].

5 Computational Results

We tested `miqoGraph` on three simulated admixture graphs that represent the possible varieties of admixture events to validate its performance. Parameters for the simulation were described in Section 4.

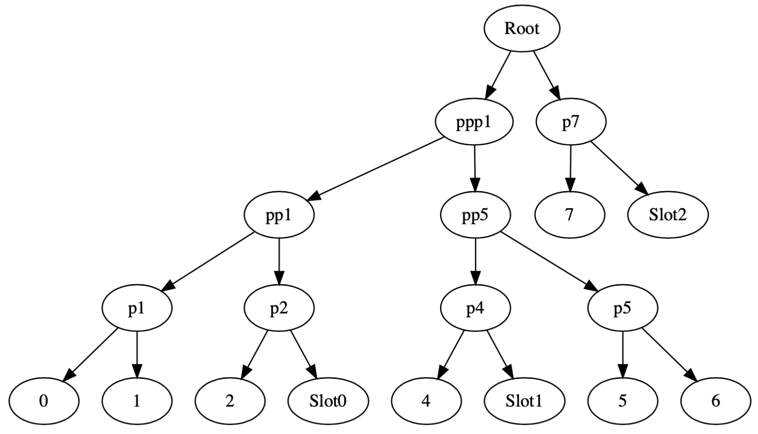
A summary of our simulated graphs is shown in Figure 7. Figure 7a shows the base graph upon which the three simulated graphs are built. The first of these, in Figure 7b, involves a single admixture event where Population 3 is produced from equal mixtures of Populations Slot0 and Slot1. The next graph in Figure 7c is identical to the first except that the admixture proportions are changed to 10% and 90%. A more complex graph in Figure 7d has Population 3 produced from a nested admixture event between Populations Slot0, Slot1, and Slot2. These graphs will be referred to as *SimpleMix*, *UnevenMix*, and *NestedMix*, respectively.

These simulations are by no means exhaustive. Because the underlying optimization model's size scales quadratically with the admixture granularity K, `miqoGraph` may not be appropriate for detecting low admixture proportions; the lowest that we test is 10% in the UnevenMix case. Furthermore, our formulation models admixture only at the leaf nodes: for example, the NestedMix case is represented as Population 3 being assigned 25% to Slot0, 25% to Slot1, and 50% to Slot2. Although the nesting is straightforward in this case, `miqoGraph` may produce less interpretable results for admixture graphs with many interdependent nested admixture events. Nonetheless, `miqoGraph` is a powerful tool on a variety of use cases, as we will show.

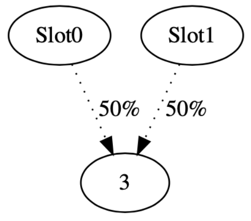
5.1 The SimpleMix example

The first step in recovering the SimpleMix graph was to infer a topology without admixture. The optimal topology was found in merely four seconds, with an objective value of 274.41. The topology without admixture was quite close to the actual topology, with the sole error that population 3 was placed in its ancestral pre32 node due to the graph being unable to capture admixture.

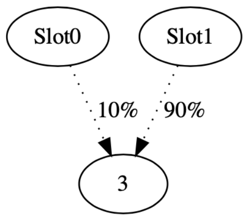
We then fitted a tree with a single admixture event at an admixture resolution of $K = 2$. Our `miqoGraph` algorithm found the optimal solution in 94 seconds with an objective value of 18.99, although it took significantly longer to prove optimality, terminating in 1,040 seconds. The optimization progression is shown in Figure 8, with time on the x-axis on a log scale, and the objective value on the y-axis. The upper bound (solid line) represents incumbent optimization solutions, with each decrease indicating that a better solution has been found. The lower bound (dotted line) represents the solver's progress towards verifying whether a solution is optimal. A solution is proved to be optimal when the upper bound meets the lower bound. Figure 8 shows that the optimization solver makes quick progress towards finding the optimal solution, but takes much longer to prove optimality. As such, common practice is to terminate the solver early.



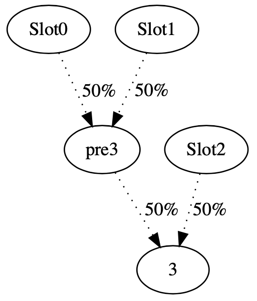
(a) A base graph topology upon which we add admixture events



(b) SimpleMix: Population 3 admixed between populations Slot0 and Slot1 equally.



(c) UnevenMix: Population 3 admixed between Slot0 (10%) and Slot1 (90%).



(d) NestedMix: Population 3 as a mixture of populations Slot0, Slot1, and Slot2.

Figure 7: Simulated admixture graph topologies.

If a guarantee of optimality is desired, we now demonstrate that it can be efficiently obtained through a combination of optimization and grid search. Solution time can speed up dramatically if certain populations are fixed to be unmixed *a priori*. Since we are solving for a tree with a single admixture event in the SimpleMix case, we can fix all populations to be unmixed except for one, and solve the resulting optimization problem to optimality. If we repeat this process for every population, then we can simply choose the tree with the lowest objective value, which will be optimal for a single admixture event.

The results for this grid search are shown in Table 1, and mixing population 3 clearly gives the best result that matches the objective of 18.99, verifying the optimality of the solution in the earlier model that did not have prior knowledge of which population was admixed. This enumeration takes 106 seconds total, a dramatic reduction of the running time of the original proof of optimality. The running time of this procedure can be cut further if prior knowledge is taken into account to reduce the scenarios of the grid search, since it is often the case that there are a limited number of candidates for admixture.

Most importantly, the optimal solution matched the simulated SimpleMix graph perfectly in topology, illustrating the power of `miqoGraph` to quickly recover admixture graphs by optimizing topologies, weights, and mixing proportions jointly.

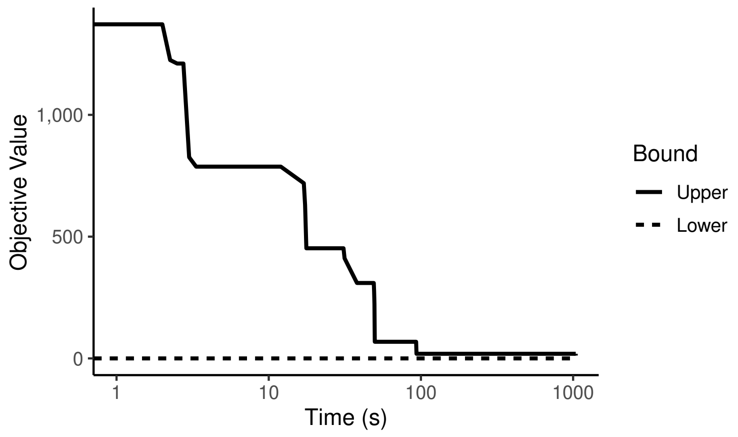


Figure 8: Optimization upper and lower bounds on the SimpleMix data with a single admixture event at a granularity of $K = 2$. Time is shown on a log scale.

Table 1: Objective values and solution times on the SimpleMix data, where a single population was allowed to be admixed and all other populations were constrained to be unmixed.

Admixed Pop.	**Objective**	**Time (s)**
1	274.03	14
2	274.03	21
3	18.99	6
4	274.29	23
5	67.08	13
6	274.29	18
8	248.73	11

5.2 The UnevenMix example

In the SimpleMix case, we were fortunate that the actual graph was admixed at exactly 50% and 50%, allowing for a low admixture granularity of $K = 2$ to capture the correct admixture event. A natural question arises when considering unequal admixture proportions that require a higher level of resolution to capture: what level of resolution is sufficient to infer the correct admixture graph? To answer this question, we turn to the UnevenMix case of Figure 7c, where the admixture proportions are 10% and 90%.

As in the SimpleMix case, for the UnevenMix case we began with solving for a tree without admixture. In this case, `miqoGraph` terminated in three seconds with an objective value of 22.08. We then ran `miqoGraph` varying the admixture resolution from $K = 2, 3, \ldots, 10$, specifying that only population 3 was admixed. The solution objective values for each level of resolution are shown in Figure 9 (solid line). For reference, the objective values for the correct admixed topology with the closest possible admixture weights to 10% and 90% allowed by the admixture granularity are shown as well (dotted line). For admixture resolutions $K \geq 2$, population 3 was forced to be admixed. For example, for the correct mixing proportions of 10% and 90%, the closest possible mixing proportions for a tree with granularity $K = 2$ were 50% and 50%, and for a mixed tree with granularity $K = 3$ they would be 33% and 66%. For the tree without admixture, we simply assigned population 3 to the pre32 node, which was optimal for the problem without admixture.

Because the tree without admixture at $K = 1$ is actually close in topology to the correct tree, its objective value is relatively low. However, at $K = 2$, the estimated and correct topologies differ sharply, as the 50%-50% mixing dictated by a $K = 2$ resolution is quite far from the 10%-90% reality, and as such `miqoGraph` finds another topology that differs from the correct topology but has lower objective value. Ultimately, past $K \geq 7$, the objectives and topologies converge to the correct values. This trajectory indicates that to get the right topology, the resolution need not be set exactly to the level required for the correct graph ($K = 10$), although it should be reasonably close. A reasonable approach might be to run the algorithm at increasing levels of admixture granularities until convergence in the topology is seen. A continuous optimization algorithm such as `qpGraph` can also be run to fine-tune the admixture proportions and weights.

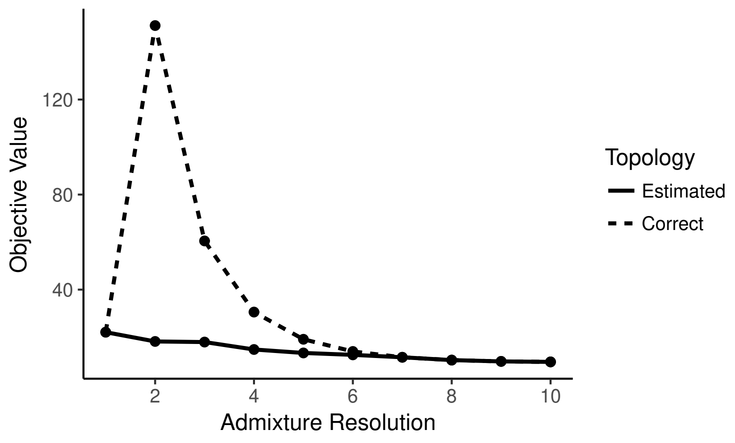


Figure 9: Objective values for increasing levels of admixture resolution on the UnevenMix case.

Formulation size and running times for varying levels of K are shown in Table 2. As K increases, the size of the formulation, as measured by the number of decision variables, increases dramatically. However, part of Gurobi's power is its ability to "presolve" and efficiently identify variables that can be eliminated from the model. As such, the presolved model size grows at a much more reasonable rate.

Past $K \geq 6$, Gurobi is unable to prove optimality within five minutes. However, the solutions are most likely optimal, given that the topology is close to the correct solution at $K = 6$ and is exactly correct for $K \geq 7$. The time required to find the optimal solution generally increases with K, as expected. However, even for $K = 10$, the solution is found in under three minutes.

Table 2: Model sizes and solution times on the UnevenMix data for varying levels of admixture granularity K. The asterix (*) indicates where `miqoGraph` found the correct topology and parameters, but was unable to prove optimality within five minutes.

K	**Num. Variables**		**Running Time (s)**	
	Initial	*Presolved*	*Last Soln.*	*Prove Opt.*
2	9,090	1,337	5	7
3	20,018	2,734	4	40
4	35,266	4,053	6	61
5	54,834	5,667	20	111
6	78,722	7,627	128	*300
7	106,930	9,951	41	*300
8	139,458	12,561	78	*300
9	176,306	15,493	68	*300
10	217,474	16,371	169	*300

5.3 The NestedMix example

Our final simulated case, NestedMix, followed a similar procedure as SimpleMix and UnevenMix, but had an additional complexity: in order to capture this graph, two admixture events were needed. The optimal graph without admixture was found in four seconds with an objective value of 157.16. Admixed graphs were inferred with two admixture events, and only Population 3 was allowed to experience admixture. For both admixture resolutions $K = 3$ and $K = 4$, the inferred topology once again matched the original topology exactly, albeit with different admixture proportions, and these trees were found in 33 and 63 seconds, respectively. The optimization progress for the $K = 4$ granularity is shown in Figure 10, with time shown on a log scale as in Figure 8. Even for this larger model, with the addition of the constraints restricting that only population 3 be admixed, the optimal solution is found within ten seconds, and the remainder of the time is spent proving optimality.

Table 3 shows a summary of the running times of `miqoGraph` for each dataset, compared with the running time of `qpGraph` (with the topology fixed to the correct topology). Our algorithm, `miqoGraph`, is able to find the correct graphs orders of magnitude more quickly than `qpGraph`, and with the exception of the UnevenMix case, we prove optimality more quickly as well. Our algorithm accomplishes these improved running times while also allowing exploration of varied graph topologies, when by comparison, `qpGraph` requires fixing a single graph topology *a priori*.

The three cases SimpleMix, UnevenMix, and NestedMix are toy examples, but they represent a range of common cases. In the following section, we demonstrate the performance of `miqoGraph` on real data.

5.4 The Eurasian-American example

We ran `miqoGraph` on a six-population dataset from Eurasia and the Americas to infer the phylogeny of populations leading to the Karitiana, a South American population from Brazil. An admixture graph

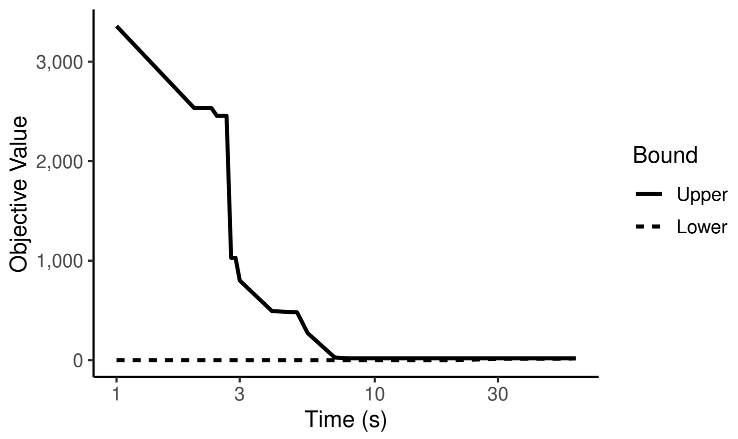


Figure 10: Optimization upper and lower bounds on the NestedMix data with two admixture events at a granularity of $K = 4$. Time is shown on a log scale.

Table 3: A comparison of running times of `miqoGraph` and `qpGraph` on three simulated admixture graphs. The asterix (*) indicates where `miqoGraph` found the correct topology and parameters, but was unable to prove optimality within five minutes.

Example	**Running Time (s)**		
	`miqoGraph` Correct Soln.	`miqoGraph` Prove Opt.	`qpGraph`
SimpleMix ($K = 2$)	2	6	393
UnevenMix ($K = 7$)	42	*300	392
NestedMix ($K = 4$)	8	63	407

created using qpGraph, with the topology pre-specified through manual enumeration, is shown in Figure 11. Karitiana was admixed between an ancient North Eurasian-related and a present-day East Asian-related source, which is consistent with [11].

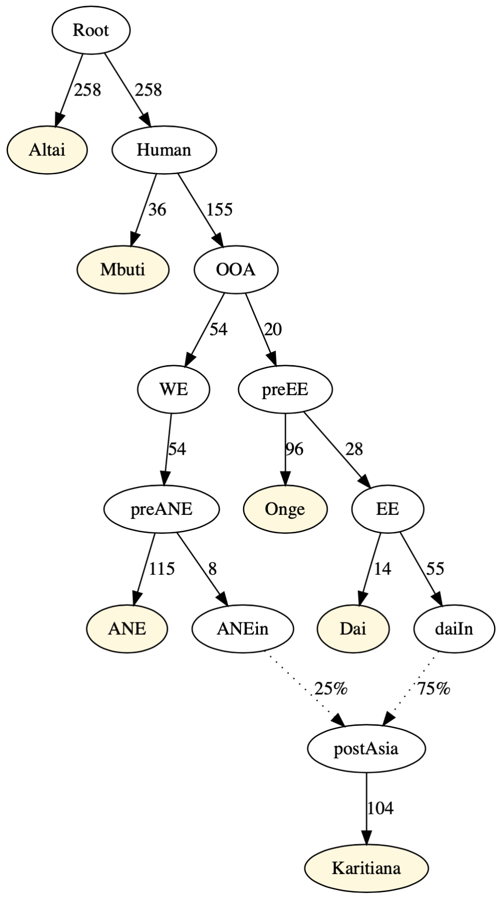


Figure 11: Drift lengths and admixture proportions inferred by qpGraph on the Eurasian-American dataset. The topology was pre-specified. Populations with real data are colored in beige, while auxiliary nodes are uncolored.

The admixture graph created using miqoGraph at $K = 4$ is shown in Figure 12a. Notably, the topology inferred by miqoGraph matches the pre-specified topology in Figure 11 (and therefore [11]), with some permutation that we explain stepwise in Figure 12. Permutation is allowed by miqoGraph because the f_3-statistics do not depend on the direction of the edges. For ease of comparison, in Figures 12b and 12c we show the steps taken to translate the output of Figure 12a into the standard format of Figure 11. The translation maps the non-leaf nodes $1-7$ of Figure 12a to an empty node, OOA, EE, Human, preANE, preEE, and an empty node, respectively. Root and WE from Figure 11 do not appear in Figure 12a, but by examining the drift lengths, we see that they were absorbed into the (4,Altai-1) and (2,5) edges, respectively. The drift of 517 along the edge (4,Altai-1) corresponds to the (Root,Altai) and (Root,Human) drifts of 258 each ($258 + 258 = 516$). Similarly, the drift of 120 along the edge (2,5) corresponds to the OOA-WE and WE-pre-ANE drifts of 54 each ($54 + 54 = 108$).

At $K = 4$, we are unable to capture the full continuity of mixing proportions, and as a result, our drift lengths and admixture proportions do not match those of qpGraph exactly. Nonetheless, they are close: our inference of 25% and 75% are close to the precise values inferred by `qpGraph` of 28% and 72%, and the drift lengths match closely as well. The drift lengths around Karatiana are an exception, but is likely due to the fact that `qpGraph` cannot distinguish between drift that occurs before or after admixture.

We also ran further instances of `miqoGraph` for $K = 5$ through 10, and as expected, saw convergence in topology, and qualitatively similar weights and proportions. Even at the highest granularity of $K = 10$, `miqoGraph` terminated in under a minute.

6 Limitations

The main limitation of `miqoGraph` lies in the restriction of admixture events to the leaf nodes of the graph and therefore, the interpretation of its output in the presence of multiple nested admixture events. Suppose a particular population A has admixture from populations B and C, and that B itself is admixed from D and E. The ordering of these events is not captured in our representation of the graph, and it can be challenging to reconstruct the correct sequence of events leading to the true admixture graph. To aid interpretability, our framework allows the user to sequentially add new populations while fixing the topology for other populations. The positions of these new populations can vary freely, or they can be tentatively assigned to positions based on the user's best guess, giving the optimizer a "warm start" to improve upon. A second issue with our approach is that the proportion of admixture inferred is done in discrete values whose granularity is specified *a priori*. It is possible that at low admixture granularities, the best-fit topology may be incorrect. One possible way to mitigate this effect is to use `miqoGraph` to explore a possible set of graph topologies and then to use continuous optimizers such as that implemented in `AdmixTools` [10] to fit parameters on these topologies.

Nonetheless, `miqoGraph`'s ability to produce admixture graphs in seconds and match both `qpGraph`-enumerated results as well as knowledge from the literature [11] illustrates the power of mixed-integer quadratic optimization in fitting topologies, drifts, and admixture proportions jointly.

References

[1] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] Pietro Belotti. Couenne: a user's manual. Technical report, Technical report, Lehigh University, 2009.

[3] Pierre Bonami and Jon Lee. Bonmin user's manual. *Numer Math*, 4:1–32, 2007.

[4] Daniele Catanzaro, Ravi Ramamoorthi, and Russell Schwartz. A mixed integer linear programming model to reconstruct phylogenies from single nucleotide polymorphism haplotypes under the maximum parsimony criterion. *Algorithms for Molecular Biology*, 8(3):2, 2013.

[5] Chris Coey, Miles Lubin, and Juan Pablo Vielma. Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation*, pp. 1–45, 2020.

[6] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pp. 3071–3076, 2013.

[7] Bernard Fortz, Olga Oliveira, and Cristina Requejo. Compact mixed integer linear programming models to the minimum weighted tree reconstruction problem. *European Journal of Operational Research*, 256(1):242–251, 2017.

[8] Jerome Kelleher, Alison M Etheridge, and Gilean McVean. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Comput Biol*, 12(5):1–22, 05 2016.

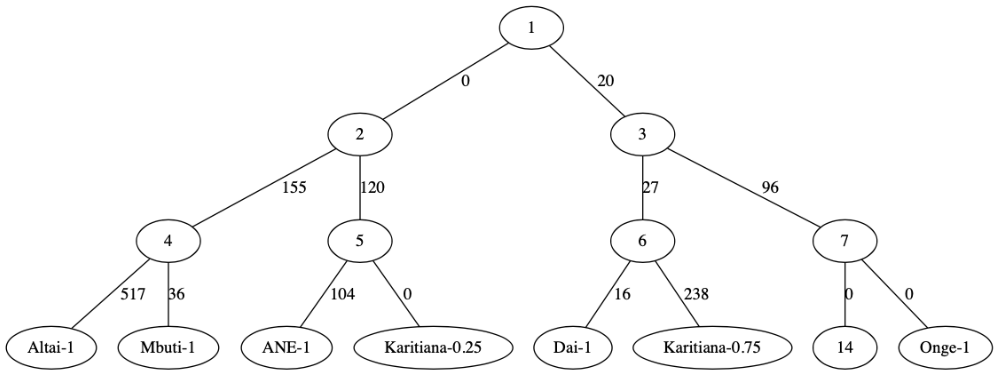
[9] Ole Kröger, Carleton Coffrin, Hassan Hijazi, and Harsha Nagarajan. Juniper: an open-source nonlinear branch-and-bound solver in julia. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 377–386. Springer, 2018.

[10] Nick Patterson, Priya Moorjani, Yontao Luo, Swapan Mallick, Nadin Rohland, Yiping Zhan, Teri Genschoreck, Teresa Webster, and David Reich. Ancient admixture in human history. *Genetics*, 192(3):1065–1093, 2012.

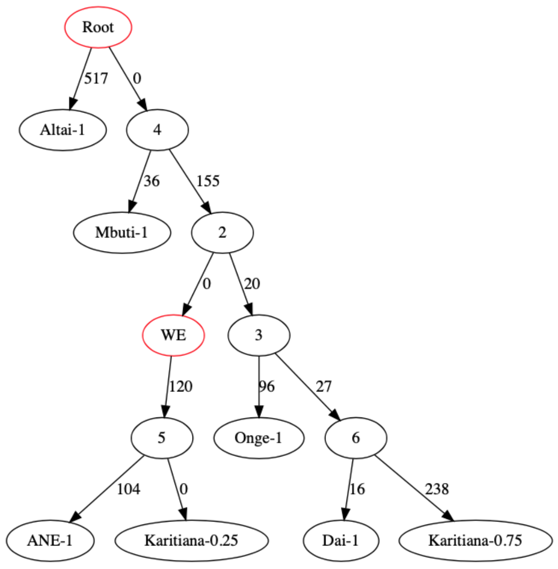
[11] Maanasa Raghavan, Pontus Skoglund, Kelly E Graf, Mait Metspalu, Anders Albrechtsen, Ida Moltke, Simon Rasmussen, Thomas W Stafford Jr, Ludovic Orlando, Ene Metspalu, et al. Upper Palaeolithic Siberian genome reveals dual ancestry of Native Americans. *Nature*, 505(7481):87, 2014.

[12] Srinath Sridhar, Fumei Lam, Guy E Blelloch, Ramamoorthi Ravi, and Russell Schwartz. Mixed integer linear programming for maximum-parsimony phylogeny inference. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):323–331, 2008.

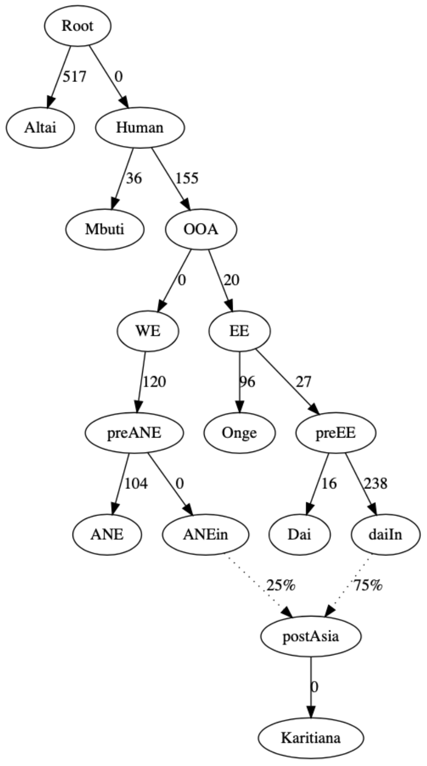
[13] Bartolomeo Stellato, Vihangkumar V Naik, Alberto Bemporad, Paul Goulart, and Stephen Boyd. Embedded mixed-integer quadratic optimization using the osqp solver. In *2018 European Control Conference (ECC)*, pp. 1536–1541. IEEE, 2018.



(a) Topology, drift lengths and admixture proportions inferred by `miqoGraph`. Nodes of the binary tree are labeled with number $1-15$, with the leaves corresponding to nodes $8-15$. The label "Altai-1" in node 8 means that Altai was assigned to node 8 at 100%; the label "Karitiana-0.25" in node 11 means that Karitiana was assigned to node 11 at 25%.



(b) An equivalent permutation of the binary tree in Figure 12a, with directedness added to the edges. Root and WE (red borders) were absorbed into the (4, Altai-1) and (2, 5) edges, respectively. Nodes 1, 7, and 14 do not appear because they are meaningless filler nodes.



(c) The graph in Figure 12b with the ancestral nodes labeled according to Figure 11.

Figure 12: Topology, drift lengths and admixture proportions inferred by `miqoGraph` on the Eurasian-American dataset. Each subfigure represents a step in translating the output of `miqoGraph` to the format of `qpGraph`. Figure 12a shows the immediate binary-tree-formatted output produced by `miqoGraph`, which is permuted in Figure 12b, and then relabeled in Figure 12a. This process illustrates that the output from `miqoGraph` matches that produced by enumeration and `qpGraph`.