Introduction and Purpose

The purpose of this project was to develop pedigree graphs for use in pedigree-linkage analysis projects. More specifically, this project sought to:

- find a source of pedigree and quantitative trait loci (QTL) data that is suitable (i.e.: large enough) for building arbitrary pedigrees
- use the aforementioned data to build Bayesian Network representations of arbitrary pedigrees
  - for this project, segregation networks were used (Allen and Darwiche)
- convert arbitrary Bayesian Network pedigrees into a standardized format (UAI)
- extract meaningful subsets of variables from the aforementioned Bayesian Network pedigrees for use in marginal MAP queries

This paper assumes general familiarity with "segregation networks" (Bayesian Network representations of pedigrees) and some relevant terminology such as allele, genotype, QTL, etc. For more information, see the Segregation Networks section of this paper and *Graphical Models for Genetic Analyses* (Lauritzen and Sheehan).

Generating Pedigree/QTL Data with QMSim

Because of an ostensible scarcity of reliable pedigree/QTL data, I used a pedigree/QTL data simulator (QMSim) to generate all the data necessary for this project.

"QTL and Marker Simulator", or QMSim, is lightweight software that generates pedigree, quantitative trait loci, and genetic marker data for simulated livestock populations. To read more about QMSim, see Sargolzaei's and Schenkel's paper *QMSim: A Large Scale Genome Simulator for Livestock*.

QMSim is a fairly complex piece of software with myriad uses, input parameters, and output parameters, and for this reason can be somewhat daunting to use. Because of the small scope of this project, however, there is a fairly small subset of usage parameters that I found useful. Below, I will describe how to use QMSim in the context of this project, which parameters should be considered in the context of this project, and the effects each parameter has on networks built from data generated by QMSim.

How to Use

Note: Although QMSim comes with a large user manual that contains an exhaustive list of parameter descriptions, many of the use cases and parameters described in the user manual are not useful in the context of this project. The full user manual can be found at: http://www.aps.uoguelph.ca/~msargol/qmsim/QMSim_documentation.pdf.

Setup

There are three version of QMSim: one for each of the three major operating systems (Linux, OS X, Windows). QMSim can be downloaded at: http://www.aps.uoguelph.ca/~msargol/qmsim/. After downloading QMSim, unzip the downloaded directory. The pre-compiled QMSim executable in the unzipped directory is now ready for use from the command line.

Command Line Execution

To execute from a command line shell:

```
./QMSim [path/to/prm/file]
```

Parameters and Parameter Files

Parameter Files

QMSim takes as input parameter files which have the extension ".prm". There are several mandatory parameters that all QMSim parameter files must contain, but to avoid repeating what is already present in the QMSim instruction manual, I'll only discuss the parameters that were of interest to me in generating data for pedigrees. There are several valid `prm` files in the `qmsim/prm` directory that came with this report, most of which were provided as samples with QMSim and which I modified as necessary. When generating new pedigrees, I recommend using these files as templates and modifying them as necessary.

Parameters

Historical Population Parameters

Historical population parameters affect the simulation of the populations that precede the population for which data is generated by QMSim. In other words, if QMSim generates data for generations 0-10, historical population parameters affect the simulation process for all generations < 0. It's especially important to pay attention to historical population data when generating small pedigrees: if the historical population isn't sufficiently large, QMSim will throw QTL variance errors.

Historical population parameters are specified between `hp` tags:

```
begin_hp;
    ...
end_hp;
```

Parameter: `hg_size`

`hg_size` is an optional parameter that affects the number of individuals in each historical generation, as well as the number of historical generations. To use:

```
hg_size = v1 [v2] ... v3 [v4];
```

Where `v1` is the number of individuals in generation `v2` (which should always start at 0), and `v3` is the number of individuals in generation `v4`. Arbitrary size/generation values can be placed between the values for the first and last generations.

Although these parameters don't have a direct effect on generated networks, they are nonetheless very important for correctly generating populations with sufficiently diverse QTL. Although the QMSim manual states that legal values for the population sizes of historical generations (`v1`) can be as small as 2, and that the number of historical generations (`v4`) can be as small as 1, these values, in my experience, must be significantly higher to produce populations with desirable genetic diversity and with sufficiently balanced gender distributions. Anecdotally, population sizes should be > 10 and the number of generations should be > 20. Of course, various parameters interrelate, so some experimentation may be required depending on the situation. For all of my experiments, I used 200 generations with population sizes of 500. In general, more and larger historical generations produce more genetically diverse reported populations (depending on a few other parameters; see the Genome Parameters section).

Population Parameters

Population parameters affect the simulation of the populations for which QMSim actually generates data.

Population parameters are specified between `pop` tags:

```
begin_pop = "str";
    ...
end_pop;
```

where `str` is a string name for the population.

Parameter: `male/female`

The `male` and `female` founder parameters control various aspects of the male and female founder populations and are primarily used, along with `ls`, `ng`, and `pmp` to control the size of the generated pedigree. Although there are several ways in which QMSim allows its users to configure the male/female founder populations, the most "interesting" use for these parameters in the context of this project is controlling the male/female ratio in the founding generation. This ratio in turn affects the sizes of subsequent generations, and therefore the number of variables in a segregation network built from generated data.

For example, a founder population consisting of 10 males and 10 females produces a segregation network with 840 variables. A founder population consisting of 2 males and 18 females produces a segregation network with 1480 variables, albeit with more inbreeding. I also found that a larger male/female ratio increases the induced width of the corresponding segregation graph. For example, using a $\frac{1}{5}$ male/female ($\frac{5}{25}$) generated a graph with an induced width of ≈ 85 (calculated using an average of 5 different networks), whereas using a ratio of $\frac{1}{1}$ ($\frac{15}{15}$) resulted in networks with induced widths of ≈ 110 (also calculated using an average of 5 different networks).

To use:

```
begin_founder
    male  [n = int, pop = "str"];
    female[n = int, pop = "str"];
end_founder
```

where `int` is an integer in the range $[1 - 50000]$ and `str` is the string name of the population for which data is being generated. The male/female ratio can be at most 1.

Parameter: `ng`

This parameter controls the number of generations in the population. To use:

```
ng = int;
```

where `int` is an int in the range $[0 - 2000]$. `ng`, as well as `ls`, `pmp`, and `male/female`, is used to control the size of the generated pedigree.

Based on my observations, `ng` has no measurable effect on network complexity outside of affecting changes in network size. A network created using 20 male founders and 20 female founders, with an `ng` of 10, contained 840 variables , had a pseudotree depth of 80, and an induced width of 54. Doubling the number of generations to 20 almost doubles the number of variables and the pseudotree depth (to 1640 and 144, respectively), but only increased the width to 63. Doubling `ng` once more to 40 about doubled the number of nodes and tree depth once again (to 3240 and 277, respectively), but again caused a modest increase in tree width from 63 to 72.

Parameter: `ls`

This parameter controls the maximum number of offspring produced by each female. `ls`, along with `ng`, `pmp`, and `male/female`, is used to control the size of the generated pedigree.

Parameter: `pmp`

This parameters controls the proportion of male progeny in each litter. `pmp`, along with `ng`, `ls`, and `male/female`, is used to control the size of the generated pedigree.

Population Output Parameters

Population output parameters allow users to specify the data that QMSim generates.

Population output parameters are specified between `popoutput` tags, which are also nested between `pop` tags:

```
begin_pop;
    ...
    begin_popoutput;
        ...
    end_popoutput;
end_pop;
```

Parameter: `data`

Specifying the `data` output parameter causes QMSim to generate family/pedigree data for the specified generations. To use:

```
data \gen0 ... \gen_n;
```

where `\gen_n` refers to an arbitrary generation. If no generation is specified, data for all generations will be reported. See the Output section for a more detailed description of the format of files generated by the `data` tag.

Note: the files generated by `data` are required to build segregation networks.

Parameter: `allele_freq`

Specifying the `allele_freq` output parameter causes QMSim to report allele frequency statistics. To use:

```
allele_freq \gen0 ... \gen_n
```

where `\gen_n` refers to an arbitrary generation. If no generation is specified, data for all generations will be reported. See the Output section for a more detailed description of the format of files generated by the `allele_freq` tag.

Note: `allele_freq` statistics for generation 0 are required to build segregation networks.

Parameter: `genotype`

Specifying the `genotype` output parameter causes QMSim to report allele assignments for individuals in the pedigree. To use:

```
genotype \gen0 ... \gen_n
```

where `\gen_n` refers to an arbitrary generation. If no generation is specified, data for all generations will be reported. See the Output section for a more detailed description of the format of files generated by the `genotype` tag.

Note: the files generated by `genotype` are required to build segregation networks.

Genome Parameters

Genome parameters control the way QMSim simulates genotype/QTL/marker data for generated pedigrees. Genome parameters are specified between `genome` tags. Additionally, all genome parameters relevant to this project are specified between `chr` (chromosome) tags:

```
begin_genome;
    begin_chr = int;
        ...
    end_chr;
end_genome;
```

where `int` is an integer that specifies the number of chromosomes for which to generate data.

QMSim produces both genetic marker and QTL data, but this project uses only QTL data.

Parameter: `nqloci`

This parameter affects the number of quantitative trait loci (QTL) on each chromosome. To use:

```
nqloci = int;
```

where `int` is an integer in the range $[0 - 50,000]$. If `nqloci` is 10, and the number of chromosomes being simulated is 20, QMSim will generate data for $10 \cdot 20 = 200$ QTL.

`nqloci` affects the complexity of segregation networks by allowing users to incorporate more alleles into a network. For examples, creating a pedigree using `male = female = 10`, `ls = 10`, and `ng = 10` produces a network with 840 variables if only one QTL is used to build the network. Adding another QTL to the network increases the size of the network to 1680, and adding another QTL to the network increases its size to 2520. It is easy to see that linearly increasing the number of QTL incorporated into a network results in a linear increase in that network's size (measured in number of variables). It is not quite so easy to describe the increase in network complexity that occurs as a result of the incorporation of more QTL: the increase in network complexity caused by adding more QTL is largely dependent on the number of allelic types that occur at that QTL and the underlying distribution of those types. See Parameter: `nqa` for more information.

Parameter: `qpos`

This parameter affects the positions of QTL on their respective chromosomes. This parameter has several potential values. See the QMSim manual for a full list of possible values. To use:

```
qpos = even;
```

causes QTL to be spaced evenly along their respective chromosomes.

```
qpos = rnd;
```

causes QTL to be spaced randomly along their respective chromosomes.

```
qpos = pd float_1 float_2 ... float_n
```

causes QTL to be positioned at predefined locations specified by the float values `float_1` to `float_n`, where all values v are $0 \leq v \leq 100$. If the `pd` option is used, the number of float position values specified must be the same as the value of `nqloci`.

`qpos` affects the complexity of networks in a manner that is difficult to measure: QTL positions are used to build factor tables for segregation graph cliques that consist of multiple segregation nodes. See the Non-Founder Segregation Node Cliques section for more information.

Parameter: `nqa`

This parameter affects the number of possible allele assignments for each QTL in the first generation. More concretely, this parameter affects the cardinalities of founder nodes (and factor tables for single non-segregation node cliques, see Founder Allele Node Cliques) in segregation networks. This parameter has several potential values. See the QMSim manual for a full list of possible values. To use:

```
nqa = all int;
```

causes all founder QTL to have `int` possible allelic types.

```
nqa = rnd int_1 int_2 ... int_n;
```

causes all founder QTL to have `int_*` possible allelic types.

```
nqa = pd int_1 int_2 ... int_n;
```

causes founder QTL at index *n* to have `int_n` allelic types. If the `pd` option is used, the number of `int` values specified must be the same as the value of `nqloci`.

`nqa` affects the complexity of segregation networks in a manner that is difficult to measure, but, intuitively, more allelic types means more complex networks.

Output

QMSim produces various output files, only some of which are necessary for creating pedigrees and segregation networks.

Note: QMSim places all output files in directories it creates that are named using the following convention: if QMSim generates data based on the parameters in the input file `input.prm`, it will place all output files in a directory named `r_input`.

Data Files

Data files are produced by QMSim when the `data` output tag is specified (see the Population Output Parameter section). These files contain all the information necessary for building pedigrees. These files are made up of rows that have the following format:

```
Progeny ID    Sire ID    Dam ID    Sex   Generation ID
```

Progeny, sire, dam, and generation IDs are all integers. Sex is one of the two strings ‘M’/‘F’.

These files contain more information, but the columns listed above are the only ones necessary for building pedigrees.

Data files are named by QMSim using the following convention:

```
[population_name]_data_[int].txt
```

Allele Frequency Files

Allele frequency files are produced by QMSim when the `allele_freq` output tag is specified (see the Population Output Parameter section). These files contain the empirical distributions of allele assignments at each QTL. These files are used to build the factor tables for cliques made up of single founder nodes in segregation networks (see Founder Allele Node Cliques). Allele frequency files are made up of rows that have the following format:

```
QTL ID    Generation ID    Chromosome ID    Variance    Allele_n:Frequency_n ...
```

If there are n possible allelic types for QTL m, then there will be n `allele:frequency` pairs in the row describing QTL m.

Allele frequency files are named using the following convention:

```
[population_name]_freq_qtl_[int].txt
```

Note: QMSim also produces marker frequency files, which are named using the following convention:

```
[population_name]_freq_mrk_[int].txt
```

This project does not use marker frequency files.

Genotype Files

Genotype files are produced by QMSim when the `genotype` output tag is specified (see the Population Output Parameter section). These files contain allele assignments for progeny reported in data files. These files are used to build UAI evidence files (see Creating UAI Representations... section). Genotype frequency files are made up of rows that have the following format:

```
Progeny ID    Paternal Genotype 0    Maternal Genotype 0    ... P Genotype n ...
```

If there are n QTL and m chromosomes being simulated, then the row describing individual i will have $2nm$ entries describing allele assignments.

Genotype files are named using the following convention:

```
[population_name]_qtl_[int].txt
```

QTL Position Files

QTL position files are produced by QMSim by default. These files describe the positions of individual QTL on their respective chromosomes. These files are used to build factor tables for segregation graph cliques that consist of multiple segregation nodes (see Non-Founder Segregation Node Cliques for more information). QTL position files are made up of rows that have the following format:

```
QTL ID    Chromosome ID    Position
```

Position values are floats that are in the range $[0 - 100]$.

These files contain more information, but the columns listed above are the only ones necessary for building pedigrees and segregation networks.

Genotype files are named using the following convention:

```
lm_qtl_[int].txt
```

Creating Bayesian Network Representations of Pedigrees

In the context of this project, the purpose of generating pedigree and QTL data is to create graphical representations of pedigrees and to convert those graphs into standardized representations that can be used as input to various graph algorithms.

Below, I will describe how the data described above is used to create graphical representations of pedigrees and how to use the Python code provided with this report (in the `uai` directory) that converts raw data into pedigree/segregation networks.

Note: all Python code assumes python3+.

Pedigrees

Note: this report assumes a basic familiarity with pedigrees in the context of pedigree-linkage analysis. For more information, see *Graphical Models for Genetic Analyses* (Lauritzen and Sheehan).

The first step in creating a segregation network is creating a representation of a pedigree that is more useful than the row/column representation created by QMSim. The Python `Pedigree` class in `uai/pedigree.py` achieves this.

`Pedigree` Objects (`pedigree.py`)

`pedigree.py` contains the class `Pedigree`, which is a Python class that is used to create objects that serve two primary functions:

- to parse pedigree data generated by QMSim
- to create graphical representations of pedigrees that are used as input to the Python `QTL` class (see the Segregation Networks section)

Input

`Pedigree` objects only have one mandatory input: the path to a `data` file generated by QMSim that describes progeny data and population structure (see the Data Files section).

How It Works

`Pedigree` objects are rather simple: they simply parse the `data` files generated by QMSim and convert the pedigree data contained therein into three different representations:

1. a dictionary representation that maps parents to their children, sex, and generation
2. a dictionary representation that maps children to their parents
3. a digraph representation that captures the underlying structure of the pedigree contained in the input file

Dependencies

`Pedigree` objects require `matplotlib` and `networkx`.

How to Use

Although `pedigree.py` does not currently support a command line interface, `Pedigree` objects can easily be incorporated into existing code.

To Instantiate

```
from pedigree import Pedigree

pedigree = Pedigree("path/to/qmsim/data/file.txt")
```

To Access Data

Data can be accessed directly from the `Pedigree` objects through the three following instance variables:

1. `pedigree.pedigree`: dictionary representation of pedigree
2. `pedigree.inverted_ped`: dictionary that maps children to parents
3. `pedigree.ped_graph`: `networkx.DiGraph` representation of pedigree

Additional Uses

Viewing/Printing Pedigrees

`Pedigree` objects offer various methods to view pedigrees and their underlying structures. See `pedigree.py` for more detailed descriptions of `Pedigree` methods.

Pruning Pedigrees

In addition to constructing pedigrees from the data produced by QMSim, `Pedigree` objects can also extract subsets of pedigrees defined by that data. If, for instance, a pedigree generated by QMSim is too large to be useful for a particular task, `Pedigree` objects provide the `extract_sub_ped` method to extract subsets of larger pedigrees. See `pedigree.py` for a detailed description of the `extract_sub_ped` method.

Segregation Networks

Although it is assumed that the reader is relatively familiar with segregation networks in the context of pedigree-linkage analysis, I will provide a brief explanation of them and why they were chosen for use in this project. For a full description of segregation networks, refer to *RC_Link: Genetic Linkage Analysis Using Bayesian Networks* (Allen and Darwiche).

Brief Overview of Segregation Networks

Segregation networks are graphical representations (Bayesian Networks) of pedigrees and describe not only the underlying structure of a pedigree, but also the allelic relationships between individuals present in a pedigree. In a segregation network, individuals are not represented by single nodes, as they would be in a pedigree graph, but are represented by either:

1. for founders: a pair of nodes, each representing a paternally/maternally inherited allele
2. for non-founders: three nodes, two representing inherited alleles and one segregation node

Types of Nodes

1. allele nodes: allele nodes can represent either maternally or paternally inherited alleles, and can take on any value that is valid for the QTL to which the given allele corresponds
2. segregation nodes: binary valued nodes that, if 0, indicates that an allele inherited the paternal allele of its parent, and if 1, indicates that an allele inherited the maternal allele of its parent

Network Structure

Nodes representing founder alleles have only outgoing edges.

Nodes representing non-founder maternal alleles have incoming edges from both alleles of the corresponding individual's mother. Nodes representing paternal alleles have incoming edges from both alleles of the corresponding individual's father. Non-founder allele nodes also have incoming edges from segregation nodes.

Segregation nodes always have at least one outgoing edge, but can, in certain circumstances, have either zero or one incoming edge and an additional outgoing edge. Segregation nodes act as "attachment nodes" for additional graph structure that describes "other" QTL. For example, if the first layer of a segregation network describes QTL a (let's call the network SG_a), another identically structured segregation network describing QTL b (SG_b) can be attached to SG_a by creating directed edges from the segregation nodes in SG_a to the corresponding segregation nodes in SG_b. Segregation nodes in SG_b will have one incoming edge and one outgoing edge, and segregation nodes in SG_a will have two outgoing edges. In this way, an arbitrary number of QTL can be described by a segregation network.

Types of Cliques

Cliques must be reported when generating UAI files (see `UAI` Objects section). Segregation networks contain four kinds of cliques.

Founder Allele Node Cliques

Founder cliques contain one allele node. Their factors consist of the empirical distributions of allelic types that are reported in QMSim allele frequency files.

Non-Founder Allele Node Cliques

These cliques consist of three allele nodes (one child node and two parent allele nodes) and one segregation node. A description of the factors of these cliques can be found in *RC_Link: Genetic Linkage Analysis Using Bayesian Networks* (Allen and Darwiche).

Founder Segregation Node Cliques

These cliques consist of one segregation node. Their factors consist of two entries: 0.5 and 0.5, the probability that, without influence from another QTL, an allele will inherit its parent's maternal/paternal allele.

Non-Founder Segregation Node Cliques

These cliques consist of two segregation nodes. Their factors are built using data from QMSim QTL position files because the segregation of an allele can be affected by the segregation of another allele based on the proximity of the two QTL. The probability that an allele at one QTL segregates given the segregation at another QTL can be calculated using a well-know formula that defines the recombination fraction between two QTL:

$$r = \frac{1}{2}(1 - e^{-2\lambda})$$

where λ is the genetic distance between two QTL. See *Graphical Models for Genetic Analyses* (Lauritzen and Sheehan) for more information.

Why Segregation Networks Were Chosen for This Project

The main alternative to segregation networks is a type of Bayesian Network called an allele network, which doesn't use segregation nodes. Ultimately, I decided to use segregation networks because this project sought to create models that included multiple QTL.

`QTL` Objects (`qtl.py`)

`qtl.py` contains the class `QTL`, which is a Python class that is used to create objects that serve four primary functions:

- to parse genotype, QTL position, and allele frequency data generated by QMSim
- to create segregation networks from the data generated by QMSim
- to store allele assignment and allele distribution data
- to provide the data and an interface for creating UAI representations of segregation networks

Input

`QTL` objects have four mandatory inputs and one optional input:

1. `ped`: a well-formed `Pedigree` object
2. `qtl_file`: the path to a genotype file generated by QMSim (see the Genotype Files section)
3. `allele_file`: the path to an allele frequency file generated by QMSim (see the Allele Frequency Files section)
4. `pos_file`: the path to a QTL position file generated by QMSim (see the QTL Position Files section)
5. `num_alleles` (optional): the number of alleles to incorporate into the constructed segregation network

How It Works

Upon construction, `QTL` objects first parse genotype files to build a structure that stores allele assignments. Allele assignments are stored in a dictionary that maps individual progeny IDs to lists of tuples of allele assignments. The length of the aforementioned lists will be equal to the `num_alleles` parameter (provided there are at least `num_alleles` QTL provided in the `qtl_file`). Each tuple in the list corresponds to one QTL, where index 0 of each tuple corresponds to a paternal allele, and index 1 to a maternal allele. This structure is used to create evidence UAI files.

`QTL` objects then parse allele frequency files and store information regarding founder allele distributions. Allele frequency distributions are stored in a doubly nested list. The outer list stores lists that corresponds to individual QTL. The inner lists store floats that indicate the distributions of allelic types for a specific QTL.

`QTL` objects then parse QTL position files and store QTL positions as a list of floats. The index of a QTL in this list is the same the index of the same QTL in the allele frequency list (described above).

Finally, `QTL` objects use `Pedigree` objects to create a `networkx.DiGraph` representation of segregation networks. The most important thing to know about the way this network is built is the convention used to name nodes:

1. allele nodes: named with three integers separated by "_" (underscore) characters. The first integer is the progeny ID of the individual that the node corresponds to. The second integer indicates whether this node refers to a paternally (0) or a maternally (1) inherited allele. The third integer indicates the index of the QTL to which this node corresponds.
2. segregation nodes: named with the character "S" and three integers separated by "_" (underscore) characters. The first integer indicates progeny ID of the relevant parent. The second integer indicates the progeny ID of the relevant child. The third integer indicates the index of the QTL to which this node corresponds.

Dependencies

QTL objects require matplotlib, networkx, numpy, and Pedigree.

How to Use

Although qtl.py does not currently support a command line interface, QTL objects can easily be incorporated into existing code.

To Instantiate

```
from pedigree import Pedigree

qtl_file    = "path/to/qmsim/genotype/file"
allele_file = "path/to/qmsim/allele/freq/file"
pos_file    = "path/to/qmsim/qtl/position/file

pedigree = Pedigree("path/to/qmsim/data/file.txt")

qtl = QTL(pedigree, qtl_file, allele_file, pos_file, num_alleles=1)
```

To Access Data

QTL objects have various useful member variables and methods. Refer to qtl.py for detailed descriptions of them.

Creating UAI Representations of Segregation Networks

Note: This report assumes familiarity with UAI. For more information, refer to: http://www.hlt.utdallas.edu/~vgogate/uai14-competition/queryformat.html.

In this section, I will explain how to use the Python code that converts segregation networks into UAI files.

UAI Objects (uai.py)

uai.py contains the class UAI, which is a Python class that is used to create objects that serve two primary functions:

- to create files that contain UAI representations of segregation networks
- to create UAI evidence files for UAI models

Input

UAI objects only require one input: a well-formed QTL object.

How It Works

UAI objects rely entirely on abstractions provided by the QTL class. The following two sections describe the three most important aspects of UAI objects.

Mapping UAI Variable IDs to Segregation Network Node Names

UAI variable IDs are mapped to variables in QTL segregation network graphs (`QTL.allele_graph`) by a topological sort: after `QTL` objects build segregation network graphs, the nodes of the graphs are topologically sorted, and the indexes of each node in the topological sort are saved to be used as identifiers in UAI files. Because arbitrary graphs can have many topological sorts, UAI IDs may be assigned to nodes differently across multiple uses.

Sorting Variables in Non-Founder Allele Cliques

Non-founder allele cliques consist of four variables. Although not complex, the way I sort the variables in these cliques for reporting in UAI files is somewhat unorthodox, so I'll describe here the rules of the sort: segregation nodes always come first, and all other nodes are ordered according to their indexes in the topological sort. The only reason I follow this protocol is because it makes building factor tables for these cliques much easier.

How Variables Are Chosen for Observation

When creating evidence files, `UAI` objects use a probabilistic method to decide which variables should be observed and incorporated into evidence files. The `UAI.observe` method, which writes model observations, takes a parameter, `prob`, that is used to determine which nodes will be observed. By default, the probability that a node will be observed is a function of `prob`, a node's position in its segregation network's topological sort (i), and the number of nodes in the segregation network (n):

$$observed = rand < prob \cdot (\frac{i}{n})$$

This prevents nodes that are too high in the segregation graph from being observed.

Dependencies

`UAI` objects require `networkx`, `numpy`, and `QTL`.

How To Use

Using from Command Line

`uai.py` supports a command line interface that allows users to create UAI models and evidence files. Please refer to `uai/README.markdown` for usage instructions.

Using In Code

To instantiate:

```
from pedigree import Pedigree
from qtl import QTL

qtl_file    = "path/to/qmsim/genotype/file"
allele_file = "path/to/qmsim/allele/freq/file"
pos_file    = "path/to/qmsim/qtl/position/file

pedigree = Pedigree("path/to/qmsim/data/file.txt")

qtl = QTL(pedigree, qtl_file, allele_file, pos_file, num_alleles=1)
```

```
uai = UAI(qtl)
```

To create model files:

```
with open('path/to/model/output/file', 'w') as f:
    sys.stdout = f
    uai.write()
```

To create evidence files:

```
with open(evid_out_file, 'w') as f:
    sys.stdout = f
    uai.observe(prob=prob_to_observe)
```

Query Objects (`query.py`)

In this section, I will explain how to use the Python code that extracts query variables from segregation networks.

Input

`Query` objects only require one input: a well-formed `QTL` object.

How It Works

`Query` objects support two types of queries:

Allele Queries (`Query.extract_allele_type`)

Allele queries extract all variables in a segregation network that are relevant to a particular QTL. The conventions used to name nodes in `QTL` object segregation networks (see the QTL > How It Works section) make this very simple.

Family Queries (`Query.extract_within_range`)

Family queries extract families around individuals. `Query` objects have two methods that support family queries: `Query.extract_within_range` and `Query.extract_random_person`. Both of these methods do the same thing: they extract all alleles in a segregation network that are within n generations of a specific individual. The difference between these methods is that the former allows users to specify the individual around which variables are extracted, whereas the latter chooses an individual at random. For more information about `Query` methods, see the doc strings in `query.py`.

Dependencies

`Query` objects require `matplotlib`, `networkx`, `numpy`, and `QTL`.

How To Use

Using from Command Line

`query.py` does not support a command line interface, but users can make family queries by using `ini` files and `main.py`. See the End To End Usage section.

Using In Code

To instantiate:

```
from pedigree import Pedigree
from qtl import QTL

qtl_file    = "path/to/qmsim/genotype/file"
allele_file = "path/to/qmsim/allele/freq/file"
pos_file    = "path/to/qmsim/qtl/position/file

pedigree = Pedigree("path/to/qmsim/data/file.txt")
qtl      = QTL(pedigree, qtl_file, allele_file, pos_file, num_alleles=1)
query    = Query(qtl)
```

To make allele queries:

```
allele_index = 0

query_nodes  = query.extract_allele_type(allele_index):
```

To make family queries:

```
progeny_id    = 1000
rgne          = 2
allele_idx    = [0, 1, 2]

query_nodes = query.extract_within_range(progen_id, rnge, allele_idx):
```

End To End Usage

Below, I'll outline the end to end usage instructions for generating pedigree/genotype data, creating segregation networks, and extracting query variables.

Step 1: Create a QMSim `prm` File

The first step is to create a `prm` input file for QMSim. I recommend using one of the files provided with this report in the directory `qmsim/prm` as a template, and changing parameters as necessary.

Step 2: Run QMSim

Let's call the file created in the above step `input.prm`. To generate data, run QMSim from the command line (it's assumed that the cwd is the directory that contains the QMSim executable, as well as `input.prm`):

```
./QMSim input.prm
```

QMSim will create the directory `r_input`, in which it will put all relevant data.

Step 3: Define `ini` File for `main.py`

`main.py` is an executable that encapsulates all the functionality of the Python code described in this report, including `pedigree.py`, `qtl.py`, `uai.py`, and `query.py`. `main.py` allows users to generate UAI models, extract sub-pedigrees, and extract query variables all through one interface. `main.py` takes as input one command line argument that is the path to a valid `ini` file. For descriptions of valid `ini` parameters, refer to `uai/README.markdown`.

Step 4: Run `main.py`

To run `main.py` with a valid `ini` file:

```
python3 `main.py` path/to/ini/file
```

Examples

Refer to `uai/examples` for example QMSim `prm` files, example `ini` files, and the models generated from them.

References

Allen, David, and Adnana Darwiche. "RC_Link: Genetic Linkage Analysis Using Bayesian Networks." *RC_Link: Genetic Linkage Analysis Using Bayesian Networks.* International Journal of Approximate Reasoning, n.d. Web. 09 June 2016.

Sargolzaei, M., and F. S. Schenkel. "Result Filters." *National Center for Biotechnology Information.* U.S. National Library of Medicine, n.d. Web. 09 June 2016.

Sheehan, Nuala A., and Steffen L. Lauritzen. "Graphical Models for Genetic Analyses." *Statistical Science Statist. Sci.* 18.4 (2003): 489- 514. Web.