

Objects

- ped: class Pedigree
 - ped.count: int, #individuals
 - ped.markerCount: int, #markers
 - ped.markerNames: StringArray, marker names
- ped[i]: class Person
 - ped[i].famid/pid/fatid/motid: String, pedigree info
 - ped[i].sex: int, sex
 - ped[i].traits: float*, traits
 - ped[i].markers: class Alleles*
 - ped[i].markers[j][0]/[1]: int, allele
 - ped[i].markers[j].one/two: int, allele, equivalent to item above
 - ped[i].father/mother: class Person*
 - ped[i].sibs: class Person**
 - ped[i].sibCount: int, sib count
- ped.GetMarkerInfo(j): class MarkerInfo
 - ped.GetMarkerInfo(j).CountAlleles(): int, #alleles
 - ped.GetMarkerInfo(j).freq[k]: float, allele frequency for allele k
 - ped.GetMarkerInfo(j).GetAlleleLabel(k): String, allele representation

Methods

- ped.EstimateFrequencies(1, true): method, calculate allele frequency based on founders

Example

CPP

```
#include "Pedigree.h"
#include "MerlinFamily.h"
#include "MerlinHaplotype.h"
#include "MerlinSort.h"
#include <algorithm>
#include <vector>
#include <string>
#include <iterator>
#include <iostream>

void showPed(Pedigree & ped)
{
    printf("Loaded %d individuals\n", ped.count);
    for (int i = 0; i < std::min(ped.count, 10); i++) {
        printf("[%s]: %s, %s, %s, %d\t\t",
            (const char *)ped[i].pid, (const char *)ped[i].famid,
            (const char *)ped[i].fatid, (const char *)ped[i].motid,
            ped[i].sex);
        for (int j = 0; j < ped.markerCount; ++j) {
            printf("%d\t", ped[i].markers[j].one, ped[i].markers[j].two);
        }
        printf("\n");
    }
    //
```

```

printf("Loaded %d markers\n", ped.markerCount);
// Estimate allele frequencies for all markers, verbose mode
ped.EstimateFrequencies(1, false);
// Get genotype statistics for markers
for (int i = 0; i < ped.markerNames.Length(); i++) {
    printf("Statistics for marker [%s]\n", (const char *)ped.markerNames[i]);
    // Allele index starts with 1 not 0
    for (int j = 1; j <= ped.GetMarkerInfo(i)->CountAlleles(); j++) {
        printf("\tFrequency for allele %d: %f\n", j, ped.GetMarkerInfo(i)->freq[j]);
        printf("\tName for allele %d: %s\n", j, (const char *)ped.GetMarkerInfo(i)->GetAlleleLabel(j));
    }
}
return;
}

void readData(Pedigree & ped,
              const char * datfile, const char * pedfile, const char * mapfile)
{
    // The data file contains a description of the contents of the
    // pedigree file, including for example, a list of marker and
    // trait names
    ped.Prepare(datfile);
    // The pedigree file contains a list of individuals, stored one
    // per row, with specific information about each individual as
    // detailed in the data file.
    ped.Load(pedfile);
    SortFamilies(ped);
    ped.LoadMarkerMap(mapfile);
    return;
}

void loadVariants(Pedigree & ped, std::vector<std::string> & marker_ids,
                  std::vector<int> & marker_positions,
                  int chrom = 1)
{
    for (unsigned i = 0; i < marker_ids.size(); ++i) {
        ped.pd.columnHash.Push(ped.GetMarkerID(marker_ids[i].c_str()));
        ped.pd.columns.Push(1);
        ped.pd.columnCount++;
        MarkerInfo * info = ped.GetMarkerInfo(i);
        info->chromosome = chrom;
        info->position = (double)marker_positions[i] * 0.01;
    }
    return;
}

void addPerson(Pedigree & ped, std::vector<std::string> & fam_info,
               std::vector<std::string> & genotypes)
{
    // add person info
    bool failure = false;

    ped.AddPerson(fam_info[0].c_str(), fam_info[1].c_str(),
                  fam_info[2].c_str(), fam_info[3].c_str(),
                  ped.TranslateSexCode(fam_info[4].c_str(), failure));
    // add person genotypes
    for (unsigned i = 0; i < genotypes.size(); ++i) {
        String c1 = genotypes[i].c_str()[0];
        String c2 = genotypes[i].c_str()[1];
        Alleles new_genotype;
        new_genotype[0] = ped.LoadAllele(ped.GetMarkerInfo(i), c1);
        new_genotype[1] = ped.LoadAllele(ped.GetMarkerInfo(i), c2);
        if (new_genotype.isKnown()) ped[ped.count - 1].markers[i] = new_genotype;
    }
}

void loadData(Pedigree & ped, int which = 1)
{
    if (which == 1) {
        //
        // haplo.dat
        //
        std::vector<std::string> marker_ids { "V1", "V2", "V3" };
    }
}

```

```

std::vector<int> marker_positions { 1, 2, 3 };
loadVariants(ped, marker_ids, marker_positions);
//
// haplo.ped
//
std::vector<std::string> fam_info { "1", "1", "0", "0", "1" };
std::vector<std::string> genotypes { "21", "21", "21" };
addPerson(ped, fam_info, genotypes);
fam_info = { "1", "2", "0", "0", "2" };
genotypes = { "11", "11", "11" };
addPerson(ped, fam_info, genotypes);
fam_info = { "1", "3", "1", "2", "1" };
genotypes = { "21", "21", "21" };
addPerson(ped, fam_info, genotypes);
fam_info = { "2", "1", "0", "0", "1" };
genotypes = { "22", "21", "00" };
addPerson(ped, fam_info, genotypes);
fam_info = { "2", "2", "0", "0", "2" };
genotypes = { "11", "11", "11" };
addPerson(ped, fam_info, genotypes);
fam_info = { "2", "3", "1", "2", "1" };
genotypes = { "21", "21", "21" };
addPerson(ped, fam_info, genotypes);
fam_info = { "3", "1", "0", "0", "1" };
genotypes = { "22", "21", "21" };
addPerson(ped, fam_info, genotypes);
fam_info = { "3", "2", "0", "0", "2" };
genotypes = { "11", "11", "21" };
addPerson(ped, fam_info, genotypes);
fam_info = { "3", "3", "1", "2", "1" };
genotypes = { "21", "21", "21" };
addPerson(ped, fam_info, genotypes);
}
//
// sort
//
ped.Sort();
SortFamilies(ped);
}

void haplotyping(Pedigree & ped, String chrom)
{
    // activate these analysis options
    FamilyAnalysis::bestHaplotype = true;
    FamilyAnalysis::zeroRecombination = false;
    MerlinHaplotype::outputHorizontal = true;
    if (chrom == "X") PedigreeGlobals::chromosomeX = true;
    //
    ped.EstimateFrequencies(0, true);
    // recode alleles so more frequent alleles have lower allele numbers internally
    ped.LumpAlleles(0.0);
    // remove uninformative family or individuals
    // !! Do not trim here, because if a family is uninformative we can report as is
    // ped.Trim(true);
    FamilyAnalysis engine(ped);
    engine.SetupGlobals();
    engine.SetupMap(chrom);
    for (int i = 0; i < ped.familyCount; i++)
        if (engine.SelectFamily(ped.families[i])) {
            engine.Analyse();
            for (unsigned i = 0; i < engine.hapOutput.size(); ++i) {
                for (unsigned j = 0; j < engine.hapOutput[i].size(); ++j)
                    std::cout << engine.hapOutput[i][j] << "\t";
                std::cout << std::endl;
            }
        }
    engine.CleanupGlobals();
}

int main(int argc, char ** argv)
{
    if (argc != 3) {
        printf("usage: %s <data source code: 1, 2, 3> <task: 1 or 2>\n", argv[0]);
        return 0;
    }

```

```

}

Pedigree ped;
if (atoi(argv[1]) == 1) readData(ped, "haplo.dat", "haplo.ped", "haplo.map");
else if (atoi(argv[1]) == 2) readData(ped, "gene.dat", "gene.ped", "gene.map");
else if (atoi(argv[1]) == 3) loadData(ped, 1);
else ;
if (atoi(argv[2]) == 1) showPed(ped);
else if (atoi(argv[2]) == 2) haplotyping(ped, "1");
else ;
}

```

Makefile

```

CXX = g++
CFLAGS=-O3 -I./libsrc -I./merlin -I./pdf -I./clusters -D_FILE_OFFSET_BITS=64 -D_ZLIB_AVAILABLE__ -Wall -std=c++11
BINDIR = demo
MERLIN = demo/demo.exe
EXECUTABLES = $(MERLIN)
# MERLIN File Set
MERLINBASE = merlin/AssociationAnalysis merlin/FastAssociation \
merlin/AnalysisTask merlin/Conquer \
merlin/ConquerHaplotyping merlin/DiseaseModel \
merlin/ParametricLikelihood merlin/GenotypeInference \
merlin/Houdini \
merlin/KongAndCox merlin/Manners merlin/MerlinBitSet \
merlin/MerlinCluster merlin/MerlinCore \
merlin/MerlinError merlin/MerlinFamily merlin/MerlinIBD \
merlin/InformationContent merlin/MerlinCache merlin/MerlinModel \
merlin/MerlinKinship merlin/MerlinKinship15 \
merlin/MerlinHaplotype merlin/MerlinMatrix merlin/MerlinParameters \
merlin/MerlinPDF merlin/MerlinSimulator merlin/MerlinSimwalk2 \
merlin/MerlinSort merlin/NPL-ASP merlin/NPL-QTL \
merlin/Magic merlin/Mantra merlin/Parametric merlin/QtlModel \
merlin/Tree \
merlin/TreeBasics merlin/TreeIndex merlin/TreeManager \
merlin/TreeInfo merlin/TreeFlips merlin/VarianceComponents
MERLINHDR = $(MERLINBASE:=.h) merlin/TreeNode.h
MERLINSRC = $(MERLINBASE:=.cpp) demo/LibmerlinDemo.cpp
MERLINOBJ = $(MERLINSRC:.cpp=.o)

# Files for dealing with clustered markers
CLUSTERS = clusters/HaploFamily clusters/HaploGraph \
clusters/HaploSet clusters/HaploTree clusters/Likelihood \
clusters/SparseLikelihood clusters/Unknown
CLUSTERCPP = $(CLUSTERS:=.cpp)
CLUSTERHDR = $(CLUSTERS:=.h)
CLUSTEROBJ = $(CLUSTERS:=.o)
# Utility Library File Set
LIBFILE = libsrc/lib-goncalo.a
LIBMAIN = libsrc/BasicHash libsrc/Error libsrc/FortranFormat \
libsrc/GenotypeLists libsrc/InputFile libsrc/IntArray libsrc/Hash \
libsrc/LongArray libsrc/Kinship libsrc/KinshipX libsrc/MapFunction \
libsrc/MathCholesky libsrc/MathDeriv libsrc/MathFloatVector \
libsrc/MathGenMin libsrc/MathGold libsrc/MathMatrix libsrc/MathStats \
libsrc/MathNormal libsrc/MathSVD libsrc/MathVector \
libsrc/MemoryInfo libsrc/MiniDeflate \
libsrc/Parameters libsrc/Pedigree libsrc/PedigreeAlleleFreq \
libsrc/PedigreeDescription libsrc/PedigreeFamily libsrc/PedigreeGlobals \
libsrc/PedigreePerson libsrc/QuickIndex libsrc/Random libsrc/Sort \
libsrc/StringArray libsrc/StringBasics libsrc/StringMap \
libsrc/StringHash libsrc/TraitTransformations
LIBPED = libsrc/PedigreeLoader libsrc/PedigreeTwin libsrc/PedigreeTrim
LIBSRC = $(LIBMAIN:=.cpp) $(LIBPED:=.cpp)
LIBHDR = $(LIBMAIN:=.h) libsrc/Constant.h \
libsrc/MathConstant.h libsrc/PedigreeAlleles.h libsrc/LongInt.h
LIBOBJ = $(LIBSRC:.cpp=.o)
# PDF Library File Sets
PDFLIB = pdf/libpdf.a
PDFFILES = pdf/PDF pdf/PDFfont pdf/PDFinfo pdf/PDFpage \
pdf/PDFchartbasics pdf/PDFchartbar pdf/PDFlinechart \
pdf/PDFhistogram \
pdf/PDFchartaxis pdf/PDFchartlegend pdf/PDFchartmarker \
pdf/PDFchartline pdf/PDFchartobject
PDFSRC = $(PDFFILES:=.cpp)
PDFHDR = $(PDFFILES:=.h)

```

```

PDFOBJ = $(PDFFILES:=.o)

# make everything
all : $(EXECUTABLES)

$(EXECUTABLES) : $(BINDIR)

$(BINDIR) :
    mkdir -p $(BINDIR)

# dependencies for executables
$(MERLIN) : $(LIBFILE) $(PDFLIB) $(MERLINOBJ) $(CLUSTEROBJ)
    $(CXX) $(CFLAGS) -o $@ $(MERLINOBJ) $(CLUSTEROBJ) $(PDFLIB) $(LIBFILE) -lm -lz

$(LIBFILE) : $(LIBOBJ) $(LIBHDR)
    ar -cr $@ $(LIBOBJ)
    ranlib $@

$(PDFLIB) : $(PDFOBJ)
    ar -cr $@ $(PDFOBJ)
    ranlib $@

$(MERLINOBJ) : $(MERLINHDR) $(CLUSTERHDR) $(LIBHDR)

$(CLUSTEROBJ) : $(CLUSTERHDR) $(MERLINHDR) $(LIBHDR)

$(LIBOBJ) : $(LIBHDR)

$(PDFOBJ) : $(PDFHDR)

clean :
    rm -f */*.a */*.o $(EXECUTABLES)

.c.o :
    $(CXX) $(CFLAGS) -o $@ -c $*.c

#.cpp.X.o :
#    $(CXX) $(CFLAGS) -o $@ -c $*.cpp -D__CHROMOSOME_X__

.cpp.o :
    $(CXX) $(CFLAGS) -o $@ -c $*.cpp

.SUFFIXES : .cpp .c .o .X.o $(SUFFIXES)

```
