



Group 19: Pedigree Analysis

Nicole Wu (22LL20)

Sophie Liang (22WHR)

Tracy Chan (22THB2)

Annika Tran (23LM5)

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

October 31, 2024

Abstract

When given a pedigree chart, our model will track the family members to their previous generations with all possibilities triats. As they are affected by a trait to determine if a trait is: dominant or recessive, and autosomal or X-linked.

A pedigree chart is a diagram that documents how a genetic trait is pass on in a family, from one generation to another.

A trait can be determined as either dominant or recessive:

- Dominant traits only need one affected allele to be expressed.
- Recessives traits are only expressed when both alleles are affected.

Autosomal or x-linked:

- Autosomal traits are located on all the chromosomes except the X or Y chromosome
- X-linked traits are located on the X chromosomes; are more likely to affect males

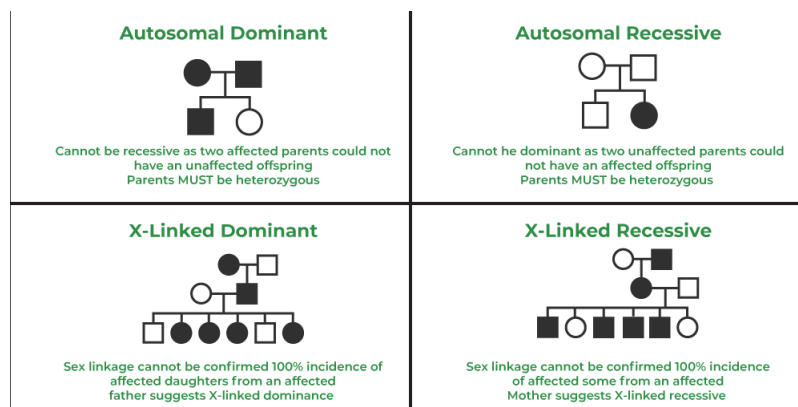


Figure 1: Image from <https://www.geeksforgeeks.org/pedigree-analysis/>

Propositions

- Drafts
 - Characteristics of each family member
 - * $F(i)$, is true if a family member at position i is female
 - * $A(i)$: is true if a family member at position i is affected
 - * $R(i)$: is true if a family member at position i is a blood relative
 - Relationship between family members
 - * $C(i,j,k)$: is true if a family member at position i is the child of family members at position j and k
 - * $S(i,j)$: is true if a family member at position i is the sibling of someone at position j
 - Inheritance Pattern
 - * $M(g)$: is true if more male family members are affected in a generation
 - Inheritance mode of trait passed down
 - * R : is true if a disease is recessive and false if it is not
 - * X : is true if a disease is x linked and false if it is autosomal

Constraints

- Drafts:

- Two family members are siblings if and only if they have the same parents:

$$* C(a, c, d) \wedge C(b, c, d) \iff S(a, b)$$

$$* \neg(\exists b, c C(a, b, c)) \vee R(a) \iff (\exists b, c C(a, b, c))$$

- If both parents of an affected family member are unaffected, then the disease is recessive

$$* (A(a) \wedge C(a, b, c)) \wedge (\neg A(b) \wedge \neg A(c)) \implies R$$

- There must be one male and one female parent

$$* C(i, a, b) \iff (F(a) \wedge \neg F(b)) \vee (\neg F(a) \wedge F(b))$$

- Parents cannot be blood relatives

$$* C(i, a, b) \iff (R(a) \wedge \neg R(b)) \vee (R(b) \wedge \neg R(a))$$

- To do:

- If there are more males in a generation than females the M(g) is true
- If M(g) is true for at least half of the generations then, X is true

Model Exploration

- building pedigree

1. First draft: 2d array based on number of gen and the number of ppl in last gen

```
gen_count = input("number of generations: ")
gen_last_count = input("number of people in generation 3: ")

def create_family_tree(gen_count, gen_last_count):
    family_tree = {}
    current_id = 1

    # Calculate the number of people in each generation, starting from the last generation
    gen_sizes = [gen_last_count]
    for _ in range(gen_count - 3):
        # Ensure each previous generation has enough people to be parents of the next generation
        gen_sizes.insert(0, (gen_sizes[0] + 1) // 2)

    # Generate unique IDs for each generation
    for size in gen_sizes:
        generation = [current_id + i for i in range(size)]
        family_tree.append(generation)
        current_id += size

    return family_tree
```

- Issue with model: too restrictive, limits the amount of generations and how many children the older generations have

2. Second Draft: Recursive 2d dictionary that build up a person, then siblings then parents

```
def create_family_tree(person_id, num_siblings, parents, family_tree, generation=0):
    # Check if the person already exists in the tree
    if person_id not in family_tree:
        # Initialize the person's data
        family_tree[person_id] = {
            "generation": generation,
            "siblings": set(),
            "parents": [],
            "spouse": None
        }

    # Add siblings based on num_siblings count
    siblings = []
    for i in range(num_siblings):
        sibling_id = person_id + i + 1
        if sibling_id not in family_tree:
            create_family_tree(sibling_id, 0, [], family_tree, generation)
        siblings.append(sibling_id)
        family_tree[person_id]["siblings"].add(sibling_id)
        family_tree[sibling_id]["siblings"].add(person_id)

    # Add parents if provided
    if parents:
        parent1, parent2 = parents
        if parent1 not in family_tree:
            # Recursively add each parent and establish spousal relationship
            create_family_tree(parent1, 0, [], family_tree, generation + 1)
        if parent2 not in family_tree:
            create_family_tree(parent2, 0, [], family_tree, generation + 1)

        # Link the parents as spouses and set them as the person's parents
        family_tree[parent1]["spouse"] = parent2
        family_tree[parent2]["spouse"] = parent1
        family_tree[person_id]["parents"] = [parent1, parent2]

    # Assign siblings' parents to the same parents
    for sibling_id in siblings:
        family_tree[sibling_id]["parents"] = [parent1, parent2]

    return family_tree
```

- Issue with model: the function does not follow the format of our propositions and constraints

3. Current draft: a dictionary of individual family members with their IDs as the key. Assign values to their keys by giving them characteristics (affected, gender, blood relation) using Char propositions. Then create a dictionary using for immediate families (siblings+parents) by linking individual family members using Rel propositions, then create a recursive function that builds the pedigree up using the dictionaries

```

PEDIGREE={}
Dictionary to store immediate family (siblings+parents)
FAMILIES={}
Dictionary to store individual family member (id, chars)
PEOPLE={}
Amount of people in the family
FAMNUM=20

Generate a dictionary represent each family member as an unique integer
def create_people(FAMNUM):
    while 1 == FAMNUM:
        id=1
        PEOPLE.update({id: Name})

@roposition()
class Char(object):
    def __init__(self, id, char):
        characteristic={0,0} char[0]: bool for female, char[1]: bool for blood relative, char[2]: bool for affected
        if char=="Female":
            char[0]=1
        elif char=="Blood Relative":
            char[1]=1
        elif char=="Affected":
            char[2]=1
        PEOPLE[id]=characteristic
    def get_person:
        if id in PEOPLE:
            return PEOPLE[id]
    def __prop_name(self):
        return f"Char({self.id}-{self.char}"

@roposition()
class Me(object):
    def __init__(self, id1, id2, id3=None):
        # Initialize a family structure
        ifamily = {"parents": [], "siblings": []}

        # Helper function to add a person to a list without duplicates
        def add_to_list(person_id, list_name):
            person = PEOPLE[person_id]
            if person not in ifamily[list_name]:
                ifamily[list_name].append(person)

        # Two arguments case: add both to siblings
        if id3 is None:
            add_to_list(id1, "siblings")
            add_to_list(id2, "siblings")
        else:
            # Three arguments case: first goes to siblings, second and third go to parents
            add_to_list(id1, "siblings")
            add_to_list(id2, "parents")
            add_to_list(id3, "parents")

        # Add the completed family dictionary to FAMILIES
        FAMILIES.append(ifamily)

```

```

@staticmethod
def create_pedigree():
    PEDIGREE = {} # Dictionary to store generational information

    # Divide PEOPLE into blood relatives and non-blood relatives
    list_BR = [id for id, details in PEOPLE.items() if details[1] == 1] # Blood relatives
    list_S = [id for id in PEOPLE if id not in list_BR] # Non-blood relatives

    # Recursive function to process blood relatives in generations
    def process_generation(current_gen, previous_gen=None):
        next_gen_ids = []

        for person_id in list_BR[:]: # Iterate over a copy of list_BR
            person = PEOPLE[person_id]
            parents = FAMILIES[person_id].get("parents", [])

            # Check for first generation (no parents)
            if not parents and current_gen == 1:
                if "gen 1" not in PEDIGREE:
                    PEDIGREE["gen 1"] = {}
                PEDIGREE["gen 1"][person_id] = person
                list_BR.remove(person_id)

            # Process subsequent generations based on parents in the previous generation
            elif current_gen > 1 and any(parent in PEDIGREE.get(f"gen {current_gen - 1}", {}) for parent in parents):
                gen_key = f"gen {current_gen}"
                if gen_key not in PEDIGREE:
                    PEDIGREE[gen_key] = {}

            # Add person and their siblings
            PEDIGREE[gen_key][person_id] = person
            siblings = FAMILIES[person_id].get("siblings", [])
            for sibling in siblings:
                if sibling in list_BR:
                    PEDIGREE[gen_key][sibling] = PEOPLE[sibling]
                    list_BR.remove(sibling)

            list_BR.remove(person_id) # Remove person from list_BR
            next_gen_ids.extend(parents) # Collect for the next generation

        # Recurse if there are more people to process in list_BR
        if list_BR:
            process_generation(current_gen + 1, next_gen_ids)

    # Start processing from generation 1
    process_generation(1)

    # Process non-blood relatives in list_S once all blood relatives are processed
    def process_siblings(last_gen):
        while list_S:
            for person_id in list_S[:]:
                person = PEOPLE[person_id]
                parents = FAMILIES[person_id].get("parents", [])

                # Find a suitable generation based on parents' generation
                for gen in range(last_gen, 0, -1):
                    if any(parent not in PEDIGREE.get(f"gen {gen - 1}", {}) for parent in parents):
                        gen_key = f"gen {gen - 1}"
                        if gen_key not in PEDIGREE:
                            PEDIGREE[gen_key] = {}
                        PEDIGREE[gen_key][person_id] = person
                        list_S.remove(person_id)
                        break

            # Process remaining siblings after blood relatives
            process_siblings(last_gen)

        return PEDIGREE

```

Jape Proof Ideas

- A person is affected and their parents are not. This must mean the parents are carriers of the trait and the trait is recessive
- In a family, one parent is affected, and one is not. They have more than one kid but not all of them are affected. This must mean either one parent has a dominant gene and the other doesn't have this gene or the parent has two recessive genes and the other is a carrier.

Requested Feedback

- Ideas for how to implement, the constraints in our to-dos (If there are more males in a generation than females the $M(g)$ is true & If $M(g)$ is true for at least half of the generations then, X is true). We are trying to do this recursively but are struggling to count the amount of people and generations with only logic.