**Graphing Patient Relationships to Obtain Medical History**

Taher Ahmed, Jasmine Boyer, Ethan Carpe, Brooke Lampe, Brett Middle

University of Nebraska-Lincoln: Department of Computer Science and Engineering

[taher139010@stemegypt.edu.eg](mailto:taher139010@stemegypt.edu.eg), [jasmineboy2@gmail.com](mailto:Jasmineboy2@gmail.com), [ethanmichaelcarpe@gmail.com](mailto:ethanmichaelcarpe@gmail.com), [blampe@cse.unl.edu](mailto:blampe@cse.unl.edu), [bmiddle@cse.unl.edu](mailto:bmiddle@cse.unl.edu)

**1. Introduction**

In our previous courses, we have been working with OpenMRS, an open-source medical software system. The system maintains the patient’s medical records, in terms of medical conditions, treatments, surgeries, etc., but it does not track the patient’s family medical history. Medical history is crucial in the timely diagnosis of a number of serious diseases, including cancer.

We propose a system that will link the medical records of families across a hospital’s software system. It will determine a patient’s relatives by analyzing the patient’s listed relatives and then accessing the listed relatives’ data to find the relatives listed there. As more connections are found, the medical history becomes more concrete. Thus, the doctors treating the patient have a useful diagnostic tool. This system will be particularly important for patients who list very few family members or have little access to family medical history.

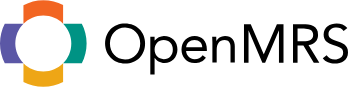
2. Challenges

One of the biggest challenges when building any medical system is to respect HIPAA laws. We should not expose any information that can identify a patient to someone who is not authorized to do so, which means either the user must be authorized through some means (username and password authentication, etc.) or we strip the names out as a whole for anonymization. Since patients might not have a certain relationship to be added in the first place, such as their parents marrying into a new family and them becoming step-children or they’re adopted and have no provided family history, we would not be able to map out their connections accurately.

Since we are required to generate and use random data, this can be a challenge as we need to be able to create families and assign medical conditions as generation proceeds. After generation, we should be able to represent the information that we created through neo4j, which is a new library for all of us in this group. We will need to take the time to learn the library as well as get to know how to pass our data through.

3. Related Work

A majority of the team has had related work concerning OpenMRS due to the coursework of the Software Engineering major. We had been tasked with building our own module which had utilized patient records to assist in their creation. A challenge for some teams required them to ensure that a specifically authorized user (doctor, front-desk receptionist, etc.) was signed in in order to use their module, which is addressed in the previous section of this proposal. We also have experience creating diagnosis software, which takes in medical history and current lab results and runs them through an algorithm that determines whether or not to diagnose a patient with a certain disease.



**Figure 1.** *OpenMRS logo.*

Although, OpenMRS does have the capability to list off relatives for each patient. It does not, however, have the ability to create a user-friendly graph to display the proper relationships.

In a similar vein, there are projects that are able to create and display family trees. DNA analytic websites are able to explain the possibilities of certain hereditary conditions being passed down to future generations.

4. Proposed Approaches

We will be using JavaScript with PHP that connects to a MySQL database. The JavaScript will then be used to create a patient relationship graph and to run queries with regard to medical history of illnesses.

We need to generate the data for our purposes. To start off, we need to generate a patient table that is full of patient information, such as their name, where they live, and so on. This does not need to have any connection to other tables, but it needs to give some sort of information about a specific patient. The next thing we need to do is create patient to parent connections which would require another table. Lastly, we need to grab a handful of potential hereditary diseases and randomly assign them to patients.

When a patient is selected, we would like to find all relatives of this patient. For each relative we will calculate a weight based on how much DNA we estimate is shared between the patient and the relative. We will calculate the weights associated with each disease and find the highest weighted diseases for the patient. We will use the relative, weight, and disease information to generate a graph of the patient’s family and have different colors correspond to different diseases. Perhaps we can use the weight to give the intensity of each node.

5. Expected Results

We expect to have the ability to generate a graph that illustrates the relationships between the patient and their relatives. This will assist in the potential diagnosis of issues that can come up through hereditary means. The graph should be able to list off the diseases that run in the family. This way, doctors may be able to narrow down what the fitting disease is by process of elimination.

The database does not need to be a perfect system: many hospital systems will have missing patient relationships, which ultimately leads to a difficult diagnosis if there is no information about a patient’s family within the database. Taking this into account will allow us to create a more realistic system to emulate what a real-world hospital system would be like. Some patients will not have a single connection; some will have parents, grandparents, maybe even further down the ancestry line; and on rare occasions should we have an extreme amount of people within a patient’s family included in the database.

6. Evaluation Plan

Our main goal is to detect relationships of medical illnesses between different patients on the diagram. To evaluate the data in our system, we will conduct queries to see if a certain patient has a medical history of a condition based on their relationships to other patients. For example, there might be a patient who is ill and whose condition is unknown to the doctors so far. As long as the patient’s mother and father are listed, the system should identify his grandparents through either their father’s and mother’s files. Then, medical professionals will be able to query the system for certain conditions they suspect the patient may have. As a result, they might be able to connect a certain illness or condition to the patient’s family.

**7. Approaches**

The approach we took differed slightly from our proposed approach. The first step in the process was generating a realistic, artificial dataset since we had limitations of what data we were able to access. We began by using a random data generator, Mockaroo, to generate a set of names addresses, phone numbers and other relevant data that is normally found in a patients file in a hospital. Next, we manipulated a selection of the last names to create families of multiple generations, resulting in a file with each person and their mother and father’s name. Then, a script was written in Java to take in this input file, and convert it to a CSV file that could illustrate “child-of” and “parent-of” relationships in Neo4j. After creating all of the person and family information, the next step was to compile a list of medical conditions and assign them to people in the database. We were able to create both a 500 person dataset and a 5,000 person dataset to run queries on. Next, we conducted research and compiled a list of the 100 most common medical conditions. A script was written in Python to randomly assign between 0 and 3 medical conditions to each person in the database, and that generated a CSV file that creates a “has-disease” relationship in Neo4j. The following query was used to import the data into Neo4j:

LOAD CSV WITH HEADERS FROM "[file:///persons.csv](https://d.docs.live.net/persons.csv)" AS line

CREATE (p:Person{id: toInteger(line.id),

first\_name: line.first\_name,

last\_name: line.last\_name});

LOAD CSV WITH HEADERS FROM "[file:///parents.csv](https://d.docs.live.net/parents.csv)" AS line

MATCH (p:Person{id: toInteger(line.person\_id)})

MATCH (f:Person{id: toInteger(line.parent\_id)})

CREATE (p)-[:CHILD {id: line.id}]->(f)

LOAD CSV WITH HEADERS FROM "[file:///diseases.csv](https://d.docs.live.net/diseases.csv)" AS line

CREATE (d:Disease{id: toInteger(line.id), name: line.name});

LOAD CSV WITH HEADERS FROM "[file:///has\_disease.csv](https://d.docs.live.net/has_disease.csv)" AS line

MATCH (p:Person {id: toInteger(line.person\_id)})

MATCH (d:Disease {id: toInteger(line.disease\_id)})

CREATE (p)-[:HAS\_DISEASE {id:line.disease\_relationship\_id}]->(d)p

The next step was to determine the appropriate queries to display the medical conditions of a certain patient and their parents, and grandparents:

START u = node([NODE\_NUMBER])

MATCH (u)-[p\*1..10]->(node) unwind p as r

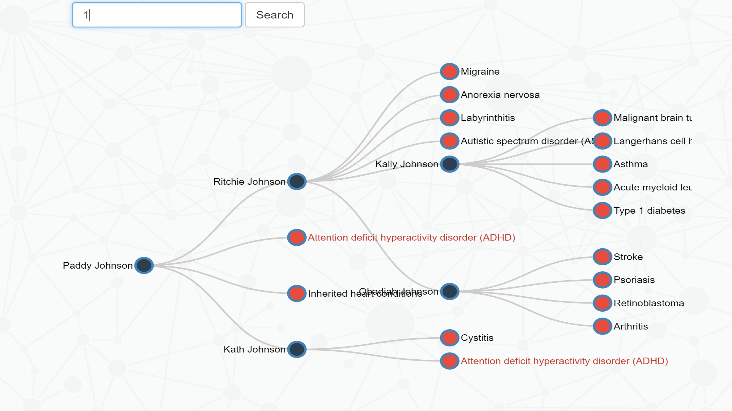
RETURN DISTINCT id(startNode(r)),endNode(r),u

We also were able to figure out a query to display the medical conditions of cousins and siblings of a certain patient in addition to their parents and grandparents:

START j=node([NODE\_NUMBER])  
OPTIONAL MATCH (j)-[:PARENT\*..10]->(k)  
WHERE size((k)-[:PARENT]->()) = 0  
OPTIONAL MATCH (k)-[:CHILD\*..10]->(node)   
OPTIONAL MATCH (j)-[:CHILD\*..10]->(node1)  
OPTIONAL MATCH (node)-[:HAS\_DISEASE]->(node2)  
OPTIONAL MATCH (node1)-[:HAS\_DISEASE]->(node3)  
RETURN DISTINCT k, node, j, node1, node2, node3

After we were able to display that information in Neo4j, the next step in our approach was to create a web interface to be able to visualize this information in a way that might be useful for a medical provider. We were able to do this using the d3 tree node library and JavaScript on the frontend, retrieving data in Neo4j using Ajax HTTP requests. The webpage allows for a user to enter a patient ID, view their family history and the associated conditions with their family they could be at risk for.

**8. Roadblocks**

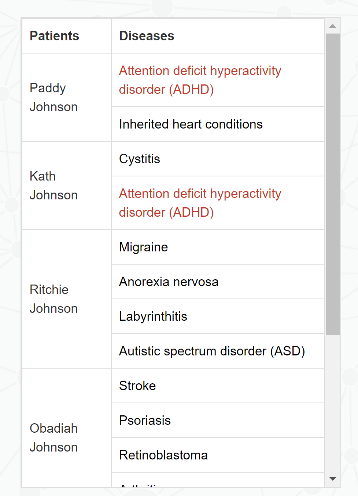
Because real medical data is sensitive, we needed to generate our own medical data. The data generation step was difficult because we had specific expectations about the data. For the data to be useful with our queries, we needed patients who were connected by parent, grandparent, etc. relationships, rather than completely random patients. To overcome this roadblock, we generated random data for each patient that included the patient name, the names of each parent, an address, and a phone number. Then, we copied some of the names from the parent columns and made them into patients with a set of parents who would be grandparents to the original patients. We also sometimes copied both parent names into a new column and generated new names for the patient, so that multiple patients would have the same patients. This created sibling relationships. We wrote out a process detailing how to manually generate children, parents, grandparents, great-grandparents, siblings, and cousins in reasonable quantities. Similarly, we generated patient diseases randomly, although in this case, we used an automated script.

**Figure 2:** Sample run of the web application that shows a family tree of the patient Paddy Johnson.

Medical data contains a lot of information, and even in our self-generated data, we had the relationships between children, parents, grandparents, great-grandparents, siblings, and cousins, along with all the diseases each family member had. We had a lot of information to represent, but that information also led to high complexity. We did not want to overwhelm ourselves as we developed the code, and we did not want to overwhelm our potential users with the information we put into the visualization. To reduce complexity, we decided to focus on parent, grandparent, and great-grandparent relationships, which are cases of direct descent. To reduce the complexity of the graph, we show the patient who was searched, any ancestors or descendants, and each person in the graph’s diseases.

**9. Results**

As mentioned earlier, the primary objective of this project is to effectively manipulate patients’ data in a way that helps doctors diagnose their patients. After generating patients’ data, we utilized modern web technologies to retrieve, process, visualize, and display this data on a web app. That is, we developed a web application that enables doctors to search for a patient, then the website displays the family history tree of this patient along with the diseases that each person in the family has. Not only does the website display the data in a meaningful way represented in the family tree, but it also marks common diseases in that family to indicate potential diseases that the patient could have. Figure 2 shows a sample run of the website displaying the family tree of Paddy Johnson. As indicated with the red font color, ADHD is a common disease in this family.

In addition to displaying the family visualized in a tree structure, the website also‒ as shown in figure 3‒ lists all family members and diseases in a table.

With regard to efficiency and performance, the website showed high performance results in retrieving and processing the data, which is a result of our efficient design in using Neo4j to store the data. Additionally, since scalability was one or our main goals, we made the system scalable so that it can be extended to have more functionalities, such as considering cousins and siblings relationships, finding relationships between a disease and people who have this disease, search by name instead of id, etc.

**Figure 3:** A table representaion of figure 1

**10. Conclusions**

Our overall goal with this project is to build a system that would improve the diagnosis by giving doctors an insight into illnesses that run in the patient’s family. This was done by implementing a graphical solution. The graph links the patient to the patient’s family members and correspondingly, all members to the genetic diseases which they have already been diagnosed with. The solutions presented currently only links patients with their medical conditions and direct ancestors. The end goal would be to have an application where a doctor could enter the patient’s name/patient ID and then that would be used in a query to then display the family of the patient and the possible diseases. We were able to get cypher queries that displayed all family members including siblings and cousins. We decided to keep it simple by not implementing this in the webpage, but this is potentially something that could be added in the future. We could also weight the diseases in likelihood by the percent of DNA shared by the relative. A parent having a given disease is more alarming than a cousin, for example. Also, for researchers, we could display the graph of all patients afflicted by a given disease.

11. References

https://www.bmj.com/rapid-response/2011/10/29/importance-medical-history-medical-education

https://www.nia.nih.gov/health/obtaining-older-patients-medical-history

https://scholarworks.uvm.edu/cgi/viewcontent.cgi?referer=https://www.bing.com/&httpsredir=1&article=1154&context=fmclerk