Primary Care Clinician Match System for Vaden Health Services

CS151 Logic Programming
Tiantian Fang, Farzaan Kaiyom, Filippos Nakas

I. Problem Description

Most Stanford students have paid the Campus Health Service Fee to receive primary care at Vaden Health Center for the diagnosis and treatment of illness, injury, and preventive counseling. Vaden has professional clinicians providing a full range of care, from screening tests to medical emergencies.

Nevertheless, with the lack of a dynamic clinician selection system, many Stanford students struggle to find a satisfactory clinician match when scheduling primary care appointments at Vaden. Especially amid Covid-19, the remote clinical workflow can be unstandardized and inadequate, while manual matching of patients can exacerbate the staffing shortage of medical practitioners and administrators.

Vaden currently implements a static one-size-fits-all questionnaire to help patients roughly filter clinicians who might match their needs. However, this questionnaire is single-choice, unintuitive, and too generic for patients with complex needs and symptoms. Moreover, Vaden only displays the names of available clinicians without any additional tags (like specialty information) for patients to make informed decisions.

II. Proposed Solution

Our solution optimizes for efficiency, transparency, comprehensiveness, and a clean user experience. First, patients are guided to declare relevant symptoms and health information (via checkboxes) prior to their appointments. Those symptoms are mapped to symptom categories to be displayed in the form of a pie chart. This will give patients a rough idea of which clinician specialties they should aim for when scheduling primary care appointments.

Then, patients can choose to display a ranking of Vaden clinicians based on the relevance of their sets of specialties to the patient's indicated symptoms. Our program calculates and displays a relevance score for each clinician (name and profile image). This will enable patients to compare Vaden clinicians and identify those who are most likely to help with their medical needs.

III. Design Description

A. Data

Clinician data is scraped from [Medical Services | Vaden Health]() including the clinician's name, title, professional affiliation, and professional interests. Clinician data is abstracted to specialty categories to inform the matching rules.

B. User Interface

Patients can declare their symptoms and relevant health information using the frontend checkbox components. These checkbox components are synced with the Epilog logical program using rules that map parts of the DOM to logical objects. The number and categories of symptoms checked via the checkboxes are then visualized using a frontend pie chart component that the user has an option to either hide or show. This allows patients to optionally understand the inputs into the matching system.

Patients can choose to display the pie chart and/or the ranking of Vaden clinicians. The clinician recommendations are displayed via clinician names, profile images, and relevance scores, which quantify how good of a fit a given clinician is for the user.

C. Rules

We use primitive predicates to represent our universe of symptoms, treatment categories, and doctors. Medical categories are correlated with symptoms by relations that encapsulate the strength of connections between treatment categories and particular sets of symptoms.

For instance, *dermatology* is suggested by a symptom set containing *itch* and *skin irritation* with a connection strength of 12. We chose this type of modeling to allow greater freedom in defining nonlinear relationships that can correlate symptoms and treatment categories. This allows us to give a connection strength between `"dermatology"` and `[itch, skinirritation]` that is higher than the sum of the strengths of each individual symptom by itself.

```
categorySymptomSet("dermatology",[itch,skinirritation,acne],18)
categorySymptomSet("dermatology",[itch,skinirritation],12)
categorySymptomSet("dermatology",[itch],5)
```

Once the user has selected their symptoms the logic program determines, for each treatment category, the most closely connected symptom set that is a subset of the user's symptoms. The relevance score of that symptom set with the given treatment category defines its overall relevance to the patient's symptoms. These relevance values determine the percentages displayed in the pie chart:

```
hasCategory(X) :- hasCategorySymptomSet(X, Z, N)

patientCategories(L) :- evaluate(setofall(D, hasCategory(D)),K) &
sorted(K,L)

hasCategorySymptomSet(X, Z, N) :- category(X) & categorySymptomSet(X, Z,
N) & allUserSymptoms(S) & subset(Z, S)

hasCategoryDanger(X, M) :- hasCategory(X) & evaluate(maximum(setofall(N,
hasCategorySymptomSet(X, Z, N))),M)
```

Having determined the user's risk/relevance for each medical category the program proceeds to derive the relevance of each doctor to the patient's suspected condition based on this information. Our intuition for calculating this step is that doctors that specialize in the treatment categories that most closely match the patient's symptoms should get the highest relevance scores. The database contains information about each doctor's specializations in the following form:

```
doctorCats("COLLOFF",["preventive"])
doctorCats("CUA",["urgent","women"])
doctorCats("CURRY",["adolescent","preventive"])
```

Since the patient's symptoms are somewhat related to various treatment categories and the doctors specialize in potentially many of them, our solution for this matching problem is to first calculate the intersection of likely treatment methods with the specializations of each doctor. To do this we had to define set theory and its basic operations based on sorted lists:

```
leq(X,Y) :- evaluate(stringmin(X,Y),X)
leqNum(X,Y) :- evaluate(min(X,Y),X)
primitive(X) :- symptom(X)
primitive(X) :- category(X)
list(nil)
list(cons(X,Y)) :- object(X) & list(Y)
object(X) :- primitive(X)

mem(X,X!L) :- list(L)
mem(X,Y!L) :- leq(Y,X) & mem(X,L)

subset(nil,Y) :- list(Y)
subset(X!L,Y) :- mem(X,Y) & subset(L,Y)

sorted([],[])
sorted(X, M!Y) :- listMin(X,M) & del(M, X, XM) & sorted(XM, Y)
```

We also had to define auxiliary operations that would allow us to easily manipulate these lexicographically ordered lists:

```
intersection(nil,Y,nil) :- list(Y)
intersection(X!L,Y,X!Z) :- mem(X,Y) & intersection(L,Y,Z)
intersection(X!L,Y,Z) :- ~mem(X,Y) & intersection(L,Y,Z)

minTemp(C,[])
minTemp(C, M!X) :- evaluate(stringmin(C,M),C) & minTemp(C,X)
listMin(L, X) :- member(X,L) & minTemp(X,L)

del(X,[],[])
del(X,Y!L1,Z!L2) :- distinct(X,Y) & same(Y,Z) & del(X, L1, L2)
del(X,Y!L1,L2) :- same(X,Y) & del(X,L1,L2)
```

To make sure that our input data involving sets are sorted we defined special "adding" actions that enforced this constraint, such as:

```
addSymSet(D, S, N) :: category(D) & list(S) & sorted(S, S2) ==> categorySymptomSet(D,S2, N)
defineDocCategories(D, L) :: doctor(D) & allCategories(T) & subset(L, T) & sorted(L,S) ==>
doctorCats(D,S)
defineDocCategories(D, L) :: doctorCats(D,S2) & doctor(D) & allCategories(T) & subset(L, T)
& sorted(L,S) & distinct(S,S2) ==> ~doctorCats(D,S2)
```

Using set theory, we can finally calculate the desired intersections:

```
patientDoctorOverlap(D, I) :- doctor(D) & doctorCats(D,S) &
patientDiseases(L) & intersection(S,L,I)
```

The natural next step in determining each doctor's relevance is to aggregate by summation the relevance score of all treatment categories that lie in the aforementioned intersection. After calculating the relevance score of each doctor, we use the user's place and time preferences to remove from the final doctor list all doctors that do not satisfy these criteria:

```
doctorRelevance(D, N) :- patientDoctorOverlap(D, I) &
evaluate(sum(setofall(M, member(X,I) & hasCategoryDanger(X, M))), N)

docFullRelevance(F,N) :- doctorFullName(D,F) & doctorRelevance(D, N)

filteredDoctorRelevance(D, N) :- doctorPlace(D,P) & doctorTime(D,T) &
userPlace(P1) & userTime(T1) & same(P,P1) & same(T,T1) &
doctorRelevance(D, N)

filteredDocFullRelevance(F,N) :- doctorFullName(D,F) &
filteredDoctorRelevance(D, N)
```

IV. Evaluation

We evaluate the dynamic clinician match system against the current Vaden Appointment Portal. Our system complements the current portal by allowing patients to declare multiple symptoms instead of just one. Patients can now visualize how different declared symptoms correspond to clinician specialties. This is especially helpful for primary care where many patients have complex symptoms and each clinician often covers a wide range of specialties.

The current Vaden portal assumes a one-to-one correspondence between patient symptoms and clinician specialties, but this is an inaccurate representation of reality. Our system's relevance score helps patients better navigate the complicated clinician matching process. In this way, patients can quantitatively compare the clinicians' abilities to help them with their specific sets of symptoms.

V. Summary

A. Implementation Challenges

The most important part of this project is programming the rules to dynamically match clinicians to patients. We had several different ideas for abstracting and mapping symptoms to specialties (based on severity, infectivity, and even affected organs), and eventually decided on the 'symptom set plus relevance score' idea. This is because we hope to not only utilize logic programming as appropriately as possible but also implement an easy-to-use interface.

We also dedicated a large amount of time in finding a way to normalize the final doctor relevance scores dynamically with respect to the maximum score. More specifically we attempted to scale the scores to a scale of 0 to 5 (see one of the implementations in the Appendix) . Even though our implementation was successful it made the application far too slow given the large number of floating number operations it required. Various rounding tricks were tested to solve this problem without much success.

B. Logic Programming

Our scoring of clinicians dynamically changes in accordance with the checkbox statuses on the frontend, which makes it a good use case of logic programming. Since the matching rules are key to this project, logic programming on worksheets and Sierra made it easy for us to test while developing those rules before deploying them to the actual frontend.

However, we found it challenging to integrate the Epilog backend into the HTML frontend. We had to not only coordinate the naming of predicates but also sync how users interact with the frontend with how our backend computes the relevance between symptoms, symptom sets, and clinician specialties.

C. Looking Forward

We can incorporate scheduling components into the system, which is another good use case of logic programming. This will complement our clinician match system by taking time, location, and type of visit into consideration. Moreover, we can improve on the matching rules to better serve patient needs by allowing patients to indicate the severity of their symptoms and other health information.

Another very fruitful extension would be to create an additional administrative web page that will be used by Vaden administrators to dynamically define symptom sets and determine their relevance to each treatment type. This extension might actually prove to be surprisingly easy to implement given that we have already defined all the necessary actions needed to input new symptom types, medical categories, symptom sets, and most importantly, new connections between symptom types and treatment categories, on the one hand, and doctors and specializations on the other. The intuitive symptom set conceptualization is likely to make this system very intuitive to use for Vaden's administrators.

VI. Appendix

The code is submitted in a separate HTML file on Gradescope, which also contains the Epilog code.

Relevance Score normalization code:

```
maxRelevance(M) :- evaluate(maximum(setofall(N, docFullRelevance(F,N))),M)

multFactor(N) :- maxRelevance(M) & evaluate(quotient(round(times(quotient(5,M),100)),100),
N)

normalizedDocRelevance(F,N) :- multFactor(M) & docFullRelevance(F,K) &
evaluate(round(times(K,M)),N)
```