A RADIOTHERAPY PLAN SELECTOR USING CASE-BASED REASONING

A Thesis

Submitted to

the Temple University Graduate Board

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Engineering

By

Aloysious Zziwa

May 2010

Dr. Brian P. Butz

Thesis Advisor

Dr. Chang-Hee Won

Committee Member

Dr. Iyad Obeid

Committee Member

COPYRIGHT NOTIFICATION

ABSTRACT

Developing a head and neck cancer treatment plan for a candidate of Intensity Modulated Radiation Therapy (IMRT) requires extensive domain knowledge and subjective experience. Therefore, it takes a cancer treatment team at least 2 to 3 days to develop such a plan from scratch. Many times the team may not use a reference plan. Sometimes, to reduce the amount of time taken to generate each treatment plan, these experts recall a patient, whose plan they recently prepared, and who had similar symptoms as the candidate. Using this recalled patient's plan as the starting point, the cancer treatment team modifies it based on the differences in the symptoms of the new candidate and those of the reference patient record. The resultant plan after modification is presented as the new treatment plan for the oncologist to evaluate its suitability for treatment of the candidate.

This approach is heavily dependent on the team's choice of the reference patient record. Choosing a starting treatment plan where the patient's symptoms are not the closest to the new candidate implies that more time will be spent modifying the plan than is necessary and the resultant treatment plan may not be the best achievable under the same circumstances given a better starting plan. Therefore, the team's bias in choosing the starting plan may affect the quality of treatment plan that is finally produced for the candidate.

This thesis proposes a system that behaves like an un-biased radiotherapy expert – following a similar process and standards as the human experts and which searches the entire IMRT patient database and returns the record (with patient symptoms and treatment plan) for a patient whose symptoms are most similar to the candidate's symptoms. It takes in the new candidate's information (from diagnosis, scans of the tumor and interviews with the candidate), searches the database and prints out a patient record showing another patient's treatment plan as the suggested starting point for generating the new plan. The system uses Case-Based Reasoning (CBR) because it mimics the experts' approach since it makes use of previous successes and shuns reasoning that has failed in the past. This occurs by considering only treatment plans that have been implemented successfully on patients in the hospital archive. For this thesis, CBR is applied using fuzzy IF-THEN rules to search the patient database. Fuzzy logic is used because it can handle imprecise expressions commonly used in natural language to determine the appropriate weight of the patient attributes in the search process. Filtering of patient records based on parameter value ranges is also used to reduce the number of records that have to be compared.

The system code developed for this thesis was prepared in Java and C Language Integrated Production System (CLIPS) using the Java Expert System Shell (JESS). This system is part of a bigger expert system that is being prepared by the Intelligent Systems Applications Center (ISAC) for Thomas Jefferson University Hospital, expected to generate a radiotherapy plan for a patient designated for IMRT treatment. Initial results from the developed prototype prove the viability of selecting similar patients using CBR.

It is important to note that the overall objective of the project is to build a system that effectively aids decision support by the IMRT team when generating a new treatment plan and not to replace them. The team is expected to use the generated plan as a starting point in determining a new treatment plan. If the generated plan is sufficient, the oncologist and their team will have to check this plan (in their various capacities) against expected standards for quality control before passing it on for implementation. This will save them time in planning and allow them to focus more on the patient’s needs hence a higher quality of life for the patient after treatment.

ACKNOWLEDGEMENTS

The author would like to thank the following people who have each made this thesis possible:

My thesis supervisor, Dr. Brian P. Butz, for a full year of guidance and support from inception to completion, and for giving me the academic freedom to express my opinion in research findings or differences in approaches.

My thesis committee members, Dr. Chang-Hee Won and Dr. Iyad Obeid for being understanding and considerate even during tight deadlines.

The ISAC senior design team, Mr. Andriy Demkiv, Mr. Griffin Wright, Mr. Jay Patel and Mr. John Ruddy for the feedback and comparing notes to assess ways of improving our program code.

Mrs. Kavya Kollu, for the help and great hints when CLIPS code got tough and help was not understandable or easy to find.

Mr. Tochi T. Nwachukwu, Mr. Clarence Augustin and other members of ISAC team for following up on this project and accepting to take over from where I stopped so that this project is completed.

And of course, my loving wife, Mrs. Alice Zziwa for being understanding, patient and supportive while looking after our son Jethro Zziwa, despite the big distance between us and long months away from home, and without whom, I would not have got the will to go on and complete this thesis and degree.

Thank you all!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Background of IMRT

Intensity modulated radiation therapy (IMRT), is the use of very small beams (beamlets), aimed at a tumor from many angles. During treatment, the radiation intensity of each beamlet is controlled, and the beam shape changes hundreds of times during each treatment [16]. As a result, the radiation dose effectively bends around important healthy tissues in a way that is impossible with other techniques. Because of the complexity of these motions, physicians use special high-speed computers, treatment-planning software, diagnostic imaging and patient-positioning devices (restraints) to plan treatments and control the radiation dose during therapy.

Therefore, the effectiveness of IMRT heavily depends on the accurate definition of the anatomical position of the tumor as well as the surrounding healthy tissues that may be affected by the radiation (also called Organs at Risk – OAR). To make precise definitions, Computed Tomography (CT) and Positron Emission Tomography (PET) scans of the tumor and the surrounding area are made. With the help of Magnetic Resonance Imaging (MRI) necessary three-dimensional anatomical information is obtained from the scans and a simulation of the treatment area obtained. The arrangement and effectiveness of the beamlets is tested on the simulation until the optimum treatment plan is obtained before it is implemented on the cancer patient [17].

Based on the studies that have been carried out to compare IMRT and other cancer treatment methods, IMRT appears to be clinically justifiable for most tumor locations in the head and neck area. This is because of its ability to spare adjacent normal tissues with acceptable target dose uniformity [1]. However, although technically superior, IMRT is relatively expensive compared to other treatment methods. Nevertheless, its effectiveness compared to its cost needs to be re-evaluated as technology evolves because implementation becomes less costly and more widely available.

1.2 Radiotherapy Planning

According to the interviews carried out with the head and neck cancer experts at the Thomas Jefferson University Hospitals, the radiotherapy planning process generally follows the steps below:

a) Consultation and Evaluation: The head and neck cancer patient is initially seen in consultation by the surgeon and radiation oncologist, and often by the medical oncologist. In more holistic treatments, other members of the cancer team such as, a nutritionist, psychologist, dentist and physical therapist are also consulted [2]. This is to evaluate the patient's status, prepare and explain the radiotherapy treatment to them before implementation.

b) Imaging and Simulation: With the patient in the treatment position, the tumor and region around the tumor is scanned to obtain the images that will be used in the treatment planning process. CT and PET scans are commonly obtained from this step. The Organs at Risk (OAR) and their position in relation to the tumor is noted.

c) Treatment Planning and Optimization: After construction of the 3D simulation of the tumor and surrounding areas, the radiation oncologist discusses the case with the Treatment Planner (usually a dosimetrist) communicating important information such as brief clinical findings, location of the primary tumor, high risk regions, adjacent critical structures (OAR), and the suggested minimum dose to the tumor in centi-Grays (cGy) and maximum dose to critical structures (in cGy).

   The primary site of the tumor is identified and used as the reference region when planning. Currently, the Thomas Jefferson, Head and Neck Cancer experts consider six such sites namely; Oropharynx, Hypopharynx, Larynx, Oral cavity, Nasopharynx and Sino-nasal [3]. Taking into consideration the information passed by the oncologist, the treatment planner uses the image simulation of the tumor area to assess the effectiveness of the various beamlet positions, clinical dose limits and algorithm constraints. The tradeoffs to be made for successful treatment are identified taking into consideration the patient's age, health status and personal preferences. The number of times (fractionations) the treatment has to be applied to clear the tumor is also computed and compared.

The plan that would most likely produce the best treatment results is passed onto the oncologist for evaluation as the patient's treatment plan. Depending on the oncologist's opinion, established standards and experience, the plan may be rejected or accepted. If rejected, the Treatment Planner modifies the plan until it meets the requirements and then re-submits it for evaluation. This process is repeated until a satisfactory plan is produced.

To reduce the number of iterations and also time it takes to prepare the initial plan, the Treatment Planner sometimes starts with a treatment plan of a previous patient who had similar symptoms as the new IMRT candidate. The time saved in having a reference plan as a starting point heavily depends on the plan taken as a reference. When the plan is accepted by the oncologist, the radiation therapist goes ahead to prepare the patient for treatment and implement the plan.

1.3 Objective

The goal of this thesis research is to develop a system that behaves like an un-biased radiotherapy expert – following a similar thought process and standards as the human experts, and which searches the entire IMRT patient database and returns the record (with patient symptoms and treatment plan) for a patient whose symptoms are most similar to a new IMRT candidate's symptoms. It is expected to take in the new candidate's information (from diagnosis, scans of the tumor and interviews with the candidate), searches the database and prints out a patient record showing another patient's treatment plan as the suggested starting point for generating the new plan.

1.4 Thesis Outline

This document is organized as follows; A statement of the problem is given in Chapter 2. The expected outcome of the system to solve this problem is also briefly discussed in this chapter. The following section (Chapter 3) discusses a sample of the previous work that has been done in this field of research. Chapter 4, discusses the architecture of the system developed for this thesis and how it works. The preparation, organization and meaning of the code will also be detailed in this chapter. Chapter 5 details the results obtained with screenshots to illustrate them. Conclusions and future work are included in Chapter 6 to enable smooth continuation of the research. References and the system code are appended to this document for use by the reader, where required.

2 STATEMENT OF THE PROBLEM

2.1 The Problem

IMRT planning is heavily subjective and leans on the experience of the cancer treatment team to produce an effective treatment plan. Usually, the cancer treatment team develops the plan from scratch since no two people are exactly the same (even identical twins). When a plan is done from scratch, it takes longer to complete and optimize because a number of experts have to be consulted for their subjective opinion and input to determine the viability of the plan.

Instead of taking 2 to 3 days to prepare a new plan, sometimes the team starts with a previous patient's plan to reduce the time it takes to develop the new IMRT candidate's plan. The patient generally has symptoms similar to the IMRT candidate and the patient's case is recalled by one of the members of the team. The patient's data is then retrieved from the hospital patient database for re-use. By changing the treatment specifications such as minimum dosage to the tumor and maximum dosage to the OAR (through slightly varying the angles of the beamlets) and taking into consideration the special conditions of the candidate, a new treatment plan is generated. The new plan is presented to the oncologist for evaluation and if it passes, it is implemented for the candidate.

The amount of time saved when using this option greatly relies on the plan chosen as the starting point. A patient record which may have similar symptoms but varies on aspects of high importance will require the Treatment Planner to take longer modifying the previous plan to develop the new plan. In addition, given the highly subjective nature of the optimization process, the resulting plan may not be the best achievable for the patient under the same circumstances given a better starting plan. The starting plan may be chosen based on the patients the team has worked on before, the oncologist's preferences or what one member of the team recalls from memory. Therefore, the initial bias of the team when choosing the starting plan greatly affects the quality and effectiveness of the final treatment plan, hence the quality of life of the patient after treatment.

2.2 Specifications

The problem is to be solved by attempting to return the most similar record of a patient from the Head and Neck cancer database to be used as the starting point for generation of a new plan, regardless of the composition of the cancer treatment team. First, the patient data is filtered based on major attributes identified to obtain a shortlist of generally similar patients. Next, fuzzy logic is applied to the patient attributes to determine the weight of each attribute, given the candidate's information. These attribute weights are then used to prioritize the attributes when the comparing the patient records in the shortlist to the candidate's information and hence return the most similar patient record. The determination of the most similar patient is achieved using Case-Based Reasoning (CBR) implemented with fuzzy logic.

According to Dr. Ying Xiao and Dr. Matthew Studenski of Thomas Jefferson Hospitals [3], the following attributes are mainly considered for each patient:

Table 2-1: Attributes considered in generation of a treatment plan for a Head and Neck cancer patient

Patient Attribute	**Possible Values**	**Comments**
Tumor Location[*2]	Oropharynx, Hypopharynx, Larynx, Oral Cavity, Nasopharynx, Sino-Nasal	Used to filter data
Tumor T-stage	1, 2, 3, 4	Used to filter data
Is this a Retreat?	YES, NO	Used to filter data
Tumor Orientation	Midline, Lateral	
N-Stage	1, 2, 3, 4	
Spread	Bilateral, Unilateral	
Was a resection done?	YES, NO	
Patient Age	0-10, 10-20, 20-30, 30-40, 50-60, 60-70, 70-80, 80-90, 90-100	
Patient Weight[*1]	0-20, 20-25, 25-30, 30-100	Classified by Body Mass Index (BMI)
Tumor Volume[*1]	0-1cc, 1-2cc, 2-3cc, 3-4cc, 4-5cc, 5-6cc, 6-7cc, 7-8cc, 8-9cc, 9-10cc	

Table 2-1, continued

Patient Attribute	Possible Values	Comments
Distance to Spinal Cord[*1]	0-1cm, 1-2cm, 2-3cm, 3-4cm, 4-5cm, 5-6cm, 6-7cm, 7-8cm, 8-9cm, 9-10cm	Distance taken from the centroid of the PTV
Distance to OAR[*1]	0-1cm, 1-2cm, 2-3cm, 3-4cm, 4-5cm, 5-6cm, 6-7cm, 7-8cm, 8-9cm, 9-10cm	Distance taken from the centroid of the PTV

*1 – The possible values for these attributes are estimated based on the sample values given in [3].

*2 – Figure 2-1 below illustrates the possible tumor locations in a cross section of the head and neck region.

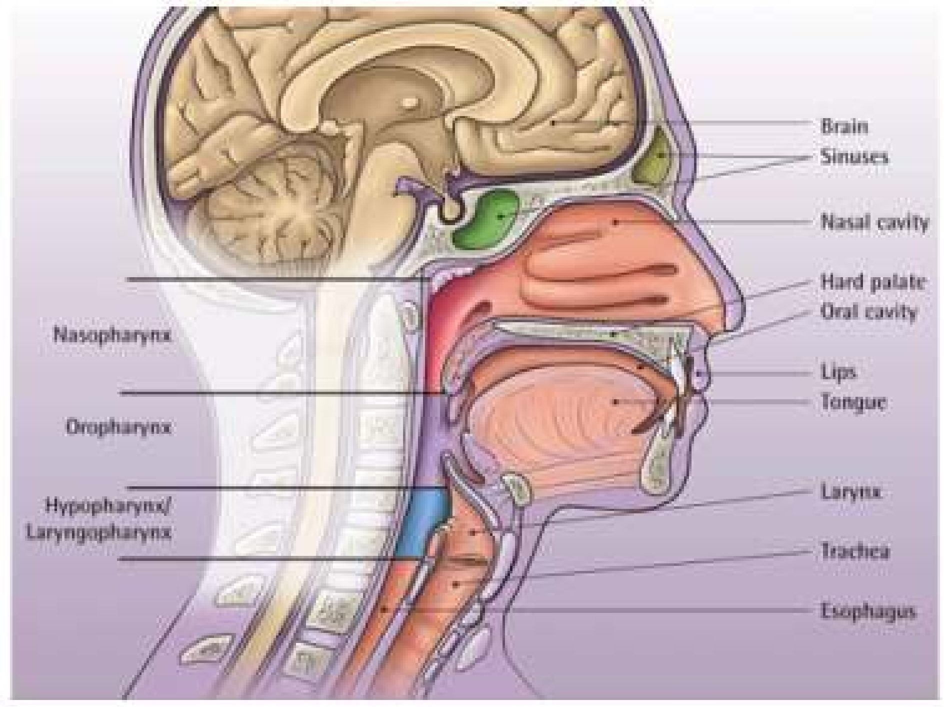


Figure 2-1: Possible tumor locations in the head and neck region [10]

It is expected that each patient record from the database contains the patient symptoms and the treatment plan that was successfully implemented on that patient. The treatment plan in the record may be shown with the Dose Volume Histogram (DVH) for the Planned Target Volume (PTV), Spinal Cord and OAR. A sample record from the database would be in the format shown in table 2-2 below. This record would be obtained from the Thomas Jefferson University Hospital database after extraction and filtering of the raw patients data. The extraction process was not part of the scope for this thesis and therefore dummy data was used awaiting completion of the data extraction system.

Table 2-2: Sample database record

Patient Attribute	**Value**	**Units**
Case Number	JF5488841	
Patient Age	23	years
Retreat	no	
Patient BMI	30	Kg/m^3
Tumor Volume	3.9	cc
Tumor Location	nasopharynx	
T Stage	T3	
N Stage	N2	
Orientation	lateral	
Spread	unilateral	
Resection	yes	
Distance to Cord	2.5	cm
Distance to Left Parotid	1.2	cm
Diagnosis Comments	The patient was down with cough 2 days before treatment	
Treatment Plan Attributes	**Value**	**Units**
Minimum PTV54 Dosage	5024	cGy
Minimum PTV60 Dosage	5635	cGy
Minimum CTV60 Dosage	5910	cGy
Maximum Cord Dosage	3982	cGy

Table 2-2, continued

Patient Attribute	**Value**	**Units**
Spinal Cord DVH[*2]	90% [702], 66% [1020], 50% [1340], 25% [2304], 10% [5903]	
Mandible DVH[*2]	90% [820], 66% [1000], 50% [1500], 25% [2400], 10% [6200]	
Treatment Time	8	
Number of Fractionations	2	
Treatment Comments	Will require restraints	

*2 – The DVH representation is represented in this form. It is assumed that 100% of each DVH receives 0cGy and therefore the beginning point; 100% [0] is always assumed. For example, the spinal cord DVH from the record shown in Table 2-2 above is illustrated in figure 2-2 below as a best-fit graph.

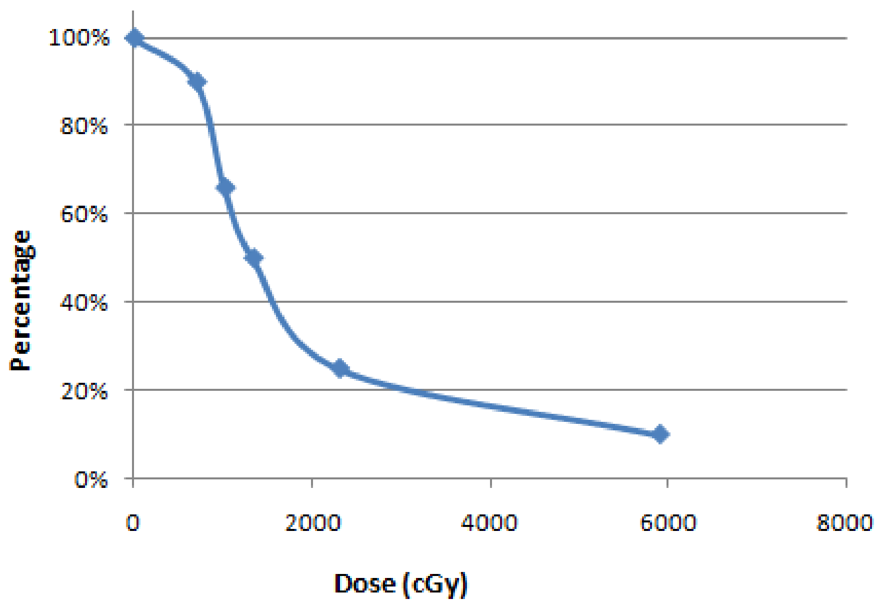


Figure 2-2: Graphical Representation of the Spinal Cord DVH value from Table 2-2

2.3 Expected Outcome

The search process is expected to filter through the data in the database based on preset must-have attributes whose values should be exactly the same. It should then return a shortlist of qualifying records for further ranking based on the other patient attributes and the expert logic. The expert logic is implemented by changing the weights (importance) of attributes based on the IMRT candidate's information. The system is expected to return the record of the most similar patient from the ranking of the shortlist. This record should contain all (but is not restricted to) the attributes outlined in Table 2-2 above.

3 PREVIOUS WORK

3.1 A Case-Based Reasoning Approach to Dose Planning in Radiotherapy

Xueyan Song et al. [4] propose a system for determining a dose plan for treatment of prostate cancer based on analysis of the trade-off between the benefits (cancer control) and risks (side effects to the rectum due to radiation). Case-Based Reasoning (CBR) is applied using fuzzy logic to make use of expert opinion, past experience and industry standards (in the field of prostate cancer treatment) for determining similar patient records.

The most relevant cases (patient records) are thus filtered out using fuzzy set theory by determining the similarity between the new candidate information and the old records while considering the level of radiation exposure of the organs at risk (mainly the rectum). An optimal solution for the new candidate of prostate cancer is then determined from the most similar records (or combination of records). Two or more records are combined using the Dempster-Shafer (DS) theory – which strengthens aspects where the records agree and reduces record conflicts in the result. It is assumed that the final dose plan of the record with the highest belief is the most suitable solution for the new candidate.

3.2 Roentgen: Radiation Therapy and Case-based Reasoning

Berger [5] proposes a case-based system – “Roentgen” that aids a dosimetrist during the planning process by suggesting new treatment plan for a new thoracic cancer patient. Roentgen bases its suggestions on the hospital archive of past patient records which captures the hospital’s experience. It carries out its function in four modules discussed in the steps below:

a) Retrieval: The system gets the patient from the record archive which best matches the tumor location and treatment constraints of the new IMRT candidate. To get the most appropriate location, the system uses the “best-fit ellipse” approach [6] on the “ellipsized” cross-sections of the patient's affected area. Search results are stored in summarized indexes and the DVH for each is computed to get the area-dose distributions. The system ranks the patient records based on "satisfaction scores". The index of the most similar record (with the highest satisfaction score) is stored for future reference in new searches.

b) Adaptation: The system then modifies the plan to fit the treatment constraints of the candidate. The modification usually involves adding offsets to the isocenter of the most similar record to align it with that of the new candidate. This leads to adjustment of the orientation of the beamlets to fit the new location and hence repositioning of the collimator jaws (which control the beamlets).

c) Evaluation: Roentegen then checks the results of implementation of the plan. Important aspects of the plan like whether the prescribed dose is delivered and there is no overdose (hot spots) in areas around the tumor or OAR are evaluated for potential errors.

d) Repair: If any problems are discovered in step (c), Roentegen goes ahead and attempts to modify the plan so that it is fit for implementation.

From the paper, it is not clear whether the system had been implemented by the time of publication. Also, no results showed how the dosimetrist benefited from the outputs of Roentgen.

3.3 Using Experience in Learning and Problem Solving

Phyllis Koton [7], in her doctoral thesis, proposed CASEY - a system that uses case-based reasoning to solve problems of heart failure. It remembers problems it has encountered before and utilizes a causal model in its domain to justify re-using previous solutions to solve unfamiliar problems. Using a model to reason about the behavior of objects in a domain is called "model-based reasoning" and is much slower than "associational reasoning" which most expert systems currently use. CASEY uses both approaches of reasoning. "Association reasoning" is used for familiar problems to save time and "model-based reasoning" is used for cases where the problem is not familiar.

It remembers patient cases it has seen before by storing them in a self-organizing memory system [8]. Such a memory stores similar records in groups called generalizations. Extraction of records from memory is along indexed paths. The most similar case from the generalizations is returned. This is termed as “reminding” of the program. It takes its inputs as a list of parameters that describe a patient (about 40 parameters) and outputs a causal explanation of the patient’s symptoms, a diagnosis and therapy suggestions for the patient. Results are compared with those of the Heart Failure Program [9] to test accuracy.

3.4 Relation of Previous Work to This Thesis

This thesis builds on the research that has been done in this area so far by combining aspects that give the above approaches an edge in the search process and implementing it for a different region of the patient’s body – the head and neck area which has different standard requirements and tradeoffs to consider. From Song’s paper [4], the method of computing similarity and applying CBR and fuzzy logic is adapted. The implementation of ranking similar records based on “satisfaction scores” is adapted from the retrieval module of Roentgen. This application will also serve as a great resource when the ISAC project reaches the evaluation and repair stage in the treatment plan generation process. CASEY’s “associational reasoning” is also adapted and modified to work for this thesis. In addition, CASEY will serve as an important future reference for further improvement in the speed of the search process through indexing patient records.

4 RADIOTHERAPY CASE-BASED REASONING

4.1 System Architecture

A schematic of the overall project is shown in figure 4-1 below. The purpose of this thesis was to develop the search functionality of this schematic. The output of the search functionality is the most similar patient record which will be adapted to generate a new DVH (part of the treatment plan).

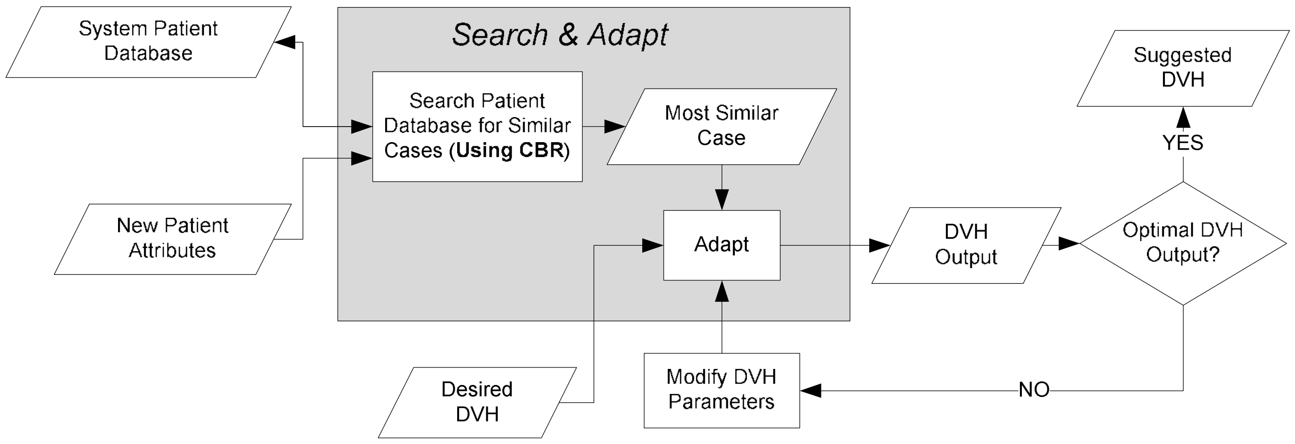


Figure 4-1: Schematic of overall system planned to ultimately generate a new treatment plan based on previous patient data, new patient attributes, computer simulation data on the new patient as well as information about the desired DVH.

A system will be developed to filter and clean the data from the Jefferson Hospital, head and neck cancer database. The processed data is stored in the system patient database. The rules implemented for CBR are extracted from information collected in interviews with the cancer experts. All required data and rules are assumed already stored in the system knowledge base when a new search is run. An example of rules that could be in the knowledge base are:

Rule 1:

```
IF     the patient [age] is YOUNG
       AND the distance to the spinal cord is VERY SMALL
THEN   the value of the weight of the distance to the spinal cord
       in search process is VERY LARGE.
```

Rule 2:

```
IF     the patient [age] is VERY OLD
       AND the distance to the spinal cord is VERY SMALL
THEN   the value of the weight of the distance to the spinal cord
       in the search process is AVERAGE.
```

Rule 3:

```
IF     the distance to the left parotid is VERY SMALL
       AND the tumor volume is VERY LARGE
THEN   the value of the weight of the distance to the left parotid
       in the search process is SMALL.
```

The fuzzy rules allow incorporation of expert reasoning and subjectivity (uncertainty) in programming. A fuzzy rule is broken down into two major parts; the antecedents (all statements on the left side of the THEN statement) and the conclusion(s) – all statements on the right side of the THEN statement. Conditions in the antecedents are separated by AND or OR. The bold parts in the above rules such as “distance to the spinal cord” are called fuzzy variables. The UPPER CASE parts of the statements are called the fuzzy values. For example “distance to the spinal cord” has fuzzy values “short”, “medium”, and “advanced”. The fuzzy variables and the fuzzy values are related by the universe of discourse. For example, the universe of discourse for “distance to the spinal cord” may be defined as in figure 4-2. The abscissa represents the distance of the tumor from the spinal cord and the ordinate represents the membership of a distance value to a class (in this case, “short”, “medium” or “long”). The membership value is between 0 and 1. Also, note that a distance value may be a member of more than one class.

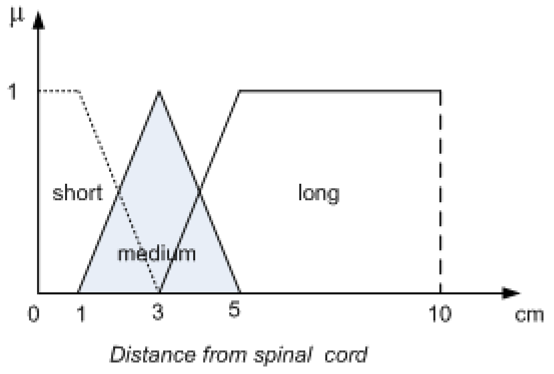


Figure 4-2: Universe of discourse of the fuzzy variable "distance from the spinal cord"

This means that if the distance from the spinal cord is less than or equal to 1cm, it is definitely short. If it is between 1cm and 3cm, it could be classified as short or medium depending on which expert's opinion is used. If the distance is 3cm, it is definitely medium. Between 3cm and 5cm, the distance could be categorized as medium or long depending on the expert. Beyond 5cm, the distance is definitely long. As one moves along the x-axis between two fuzzy values (e.g., between 1cm and 3cm), the membership of the distance in the different fuzzy sets changes. For example, at 2.8cm, the membership in fuzzy set "short" is 0.2 and it is 0.8 in fuzzy set "medium".

When fuzzy rules are executed, all rules in a given module are run. Therefore, more than one value of the solution may be obtained. If the solution is a fuzzy variable, it is *defuzzified* to obtain the final crisp solution of the problem by taking the most logical solution from the many fuzzy variables that are returned. A number of defuzzification schemes exist, the most common being *moment-deffuzification* similar to center of gravity computation. For more information on fuzzification, defuzzification and fuzzy logic, see [14] and [15].

The universe of discourse of the "weight adjustment" fuzzy variable (which is modified by the conclusion after firing each rule) is shown in figure 4-3. Where

LB = Lower Bound (-0.5 for this thesis) and

UB = Upper bound (0.5 for this thesis)

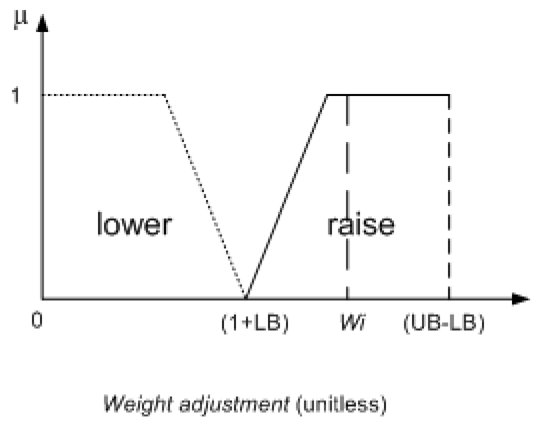


Figure 4-3: Universe of discourse of the fuzzy variable to adjust the weights of the patient attributes

You will notice that if the weight is at (1+LB), there is no adjustment and the default weight (DF) is taken as the attribute weight. In another case, given that after deffuzification, the crisp weight is at W_i, then the new attribute weight would be given by the formula 4-(i) below.

New Attribute Weight = DF + (LB + W_i) ……………………………………………..4-(i)

The maximum weight allowed for an attribute is 1 and the minimum is 0.

Figure 4-4 shows the data flow in the system from the input to the output and how modifying of the weights comes into play.

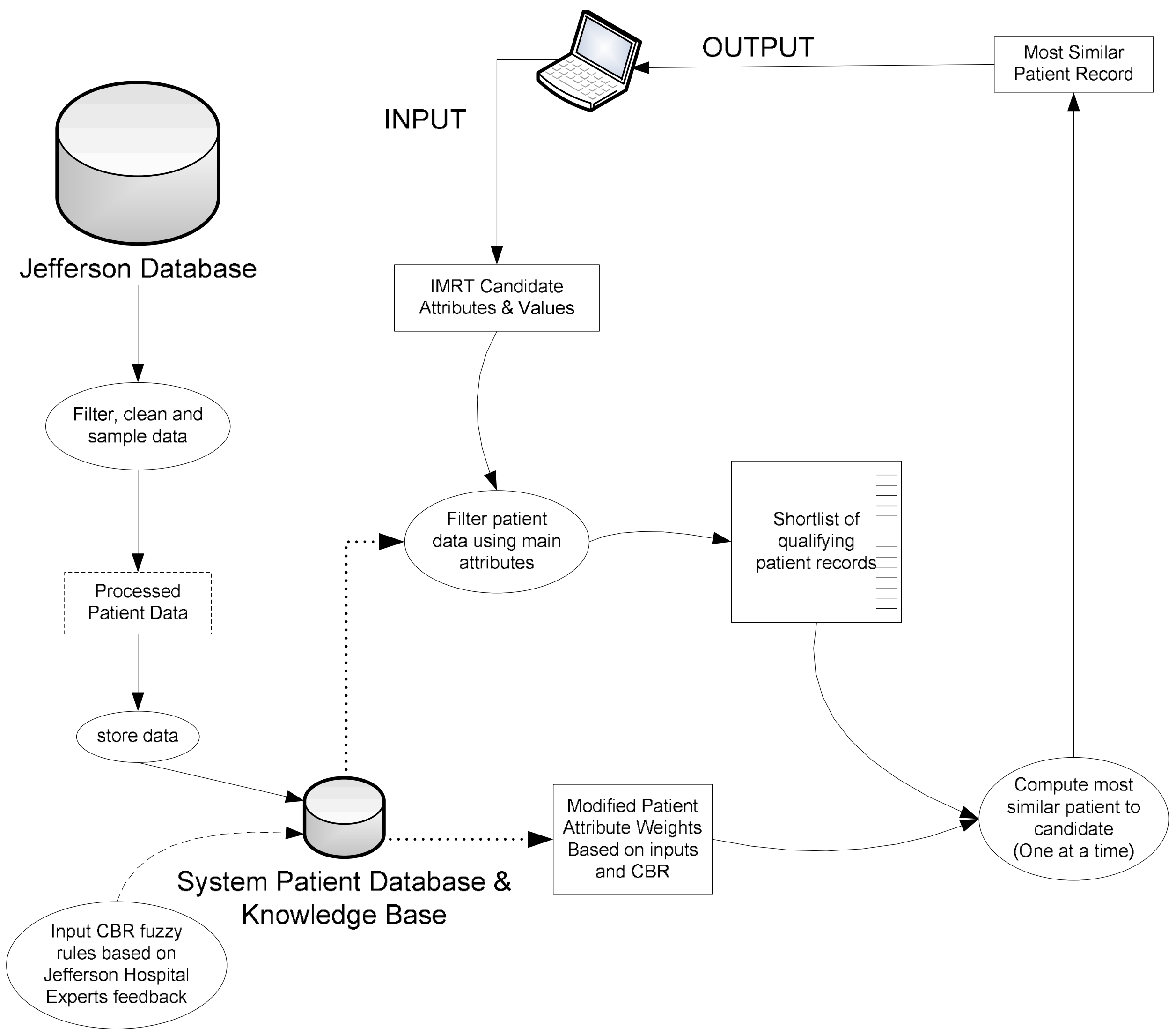

OUTPUT
Most Similar Patient Record
INPUT
Jefferson Database
IMRT Candidate Attributes & Values
Filter, clean and sample data
Filter patient data using main attributes
Shortlist of qualifying patient records
Processed Patient Data
store data
Compute most similar patient to candidate (One at a time)
Modified Patient Attribute Weights Based on inputs and CBR
System Patient Database & Knowledge Base
Input CBR fuzzy rules based on Jefferson Hospital Experts feedback

Figure 4-4: Data flow in the search system implemented for the thesis

NOTE: processing of raw data and inputting into the Knowledge Base is to be implemented in future and was not done for this thesis. It is included to better illustrate the operation of the system given the patient data. Dummy data was used in the Knowledge Base to test operation of the thesis system due to lack of ready processed data.

The search process is carried out in four major steps:

a) Input information: Values of the patient parameters of the IMRT candidate are collected from the input screen. They are converted into their computable equivalents ready to be used in the comparison process.

b) Filter patient archive: The input values of the pre-determined main attributes are used to filter the patient database and return a shortlist of records which meet all main attribute values. Each record contains the patient's diagnosis and treatment information.

c) Attribute ranking: Appropriate weights are associated with each patient attribute basing on the value of the attribute and expert reasoning (implemented by CBR). This reasoning analyses the impact of the external inputs on the attribute weights and also the impact of the attributes' values on each other's weights. For example, the importance of the distance to the left parotid (salivary gland) in a similarity search is less important if the IMRT candidate's tumor size is large and near that gland. This is because the patient can bear the discomfort of less saliva in the mouth than to have a cancerous tumor resulting in death.

d) Compute and rank shortlist by similarity: The similarity between each of the shortlist records and the IMRT candidate's record is computed according to formulae 4-(ii) and 4-(iii) below [11] and the records are then ranked according to their level of similarity to the candidate's record – most similar first. The most similar record is then displayed by the system as the output.

$$Dist = \sum_{i=1}^{8} (A_{(new)i} - A_{(old)i})W_i + \sum_{j=9}^{n+8} (A_{(new)j} - A_{(old)j})W_j \quad \text{.........................4-(ii)}$$

and $$Sim = \frac{1}{1+Dist} \quad \text{...4-(iii)}$$

Where:

- $A_{(new)}$ is the ith attribute of a new patient for whom a treatment plan is required.
- $A_{(old)}$ is the ith attribute of an existing patient in the patient database).
- W_i is the weight of the ith attribute
- *n* is the number of organs at risk and is assumed to be at least 1
- *Dist* is a measure of the distance between the new patient and the database patient.
- *Sim* is a measure of the similarity between the new patient and the database patient.

The process of implementation of the data flow was derived from [11] where the process is shown in figure 4-5 below.

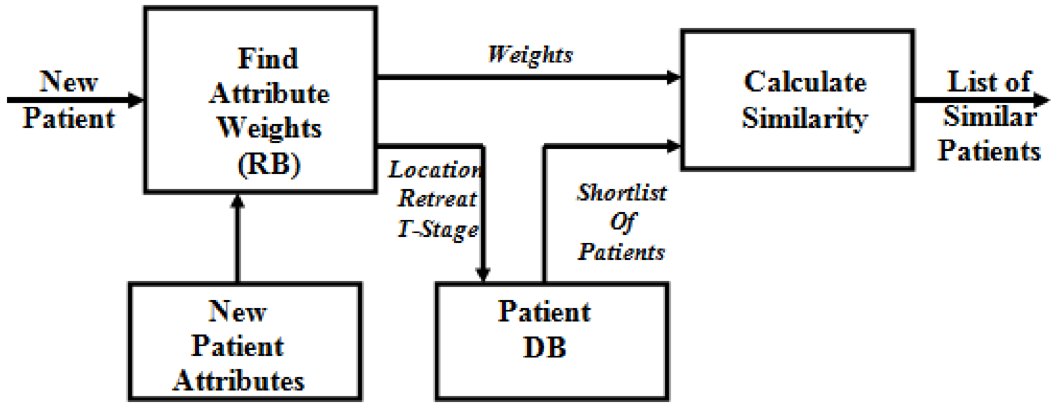


Figure 4-5: Procedure for selection of similar patients [11]

NOTE: The main attributes for this thesis were taken as; Tumor Location, Retreat (whether the candidate is being retreated or not), and the tumor T-Stage. The list of similar patients is ranked in order of similarity to get the most similar patient.

4.2 System Modules

The system for this thesis was developed mainly using the Java programming language and the C Language Integrated Production System (CLIPS) in the Java Expert System Shell (JESS). JESS was used to enable addition of Expert System capability (with fuzzy logic) to Java without requiring a separate Integrated Development Environment (IDE). JESS was installed in Eclipse IDE [12] to enable streamlined preparation of the code. The system was implemented using 8 modules each organized as a class in the Java programming language. The Java programming language (a product of Sun Micro Systems [13]) was used in the preparation of the modules because:

a) The application, once written, may be run almost anywhere. A system developed in Java can be run on almost all operating systems and browsers. It is also the most popular language for developing mobile applications and software for other electronic devices such as game consoles and other consumer devices. In most other languages, one is restricted in the operating systems or platforms where the software may be run. In addition, a different version may be required for different operating systems. To allow flexibility and portability for future applications of the system, Java was therefore a natural language of choice.

b) One can build dynamic extensible programs. This is possible because the code is organized in modular object-oriented units called “classes” which are stored in different files and loaded into the Java Interpreter only when needed. This means that a program can dynamically extend itself by loading classes it needs, when it needs them, to expand its functionality thus allowing faster operation for less intensive tasks.

c) Higher security than other languages. Java has built a reputation of being the most secure programming language because it was built from the ground-up with security in mind and is the most watched programming language by security experts. It can allow users to download un-trusted code over the network, run it in a secure environment in which it cannot do any harm (i.e., where It cannot infect the host system with a virus, read or write files to a hard drive or run other malicious code). This capability alone makes the Java platform unique and makes it the language of choice due to security requirements when handling patients’ data.

d) There have been great improvements in performance speed of Java. Currently, systems developed in Java run almost as fast as native C and C++ applications. This is because Java instructions are compiled into a portable intermediate form called byte codes which can be read by the Java Virtual Machine when the program is ran. This is much faster than purely interpreted scripted languages like PHP which require much more time to interpret compared to the byte code format of Java.

The 8 modules developed for the system are as follows:

1) Attribute Extractor: This module extracts patient variables, fuzzy variables and their definitions and fuzzy rules from the system Knowledge Base.

2) Converter: This module converts any attribute of the system passed to it into a computable format that the rest of the system can understand and use. It also is used to set the position of each field of the patient attributes in the patient records picked from the Knowledge Base.

3) Fuzzifier: Generates fuzzy variables, fuzzy values and fuzzy rules given their raw values (which are obtained from the Attribute Extractor).

4) Weight Finder: Is dedicated to determining the weight of any passed patient variable. It runs the fuzzy rules with the help of the Fuzzifier module and then returns the attribute weight. If no rules affect the passed attribute, the default weight defined in the Knowledge Base is returned.

5) UIQueryBuilder: Forms questions that the user is asked on the user interface to collect the new candidate's information. These questions are specified in the Knowledge Base. This module also stores the collected data in system memory for access by other modules when needed. In the future, this module will be replaced by a graphic user interface (GUI) that is being prepared as discussed in Chapter 6 – Future Work.

6) Search Engine: Performs any search functions of the database. It requires input data and the list of attributes to be searched and returns a list of qualifying patient records from the database.

7) Computer: Carries out any computations of similarity given the data of the records to compare and the weight of the corresponding attributes.

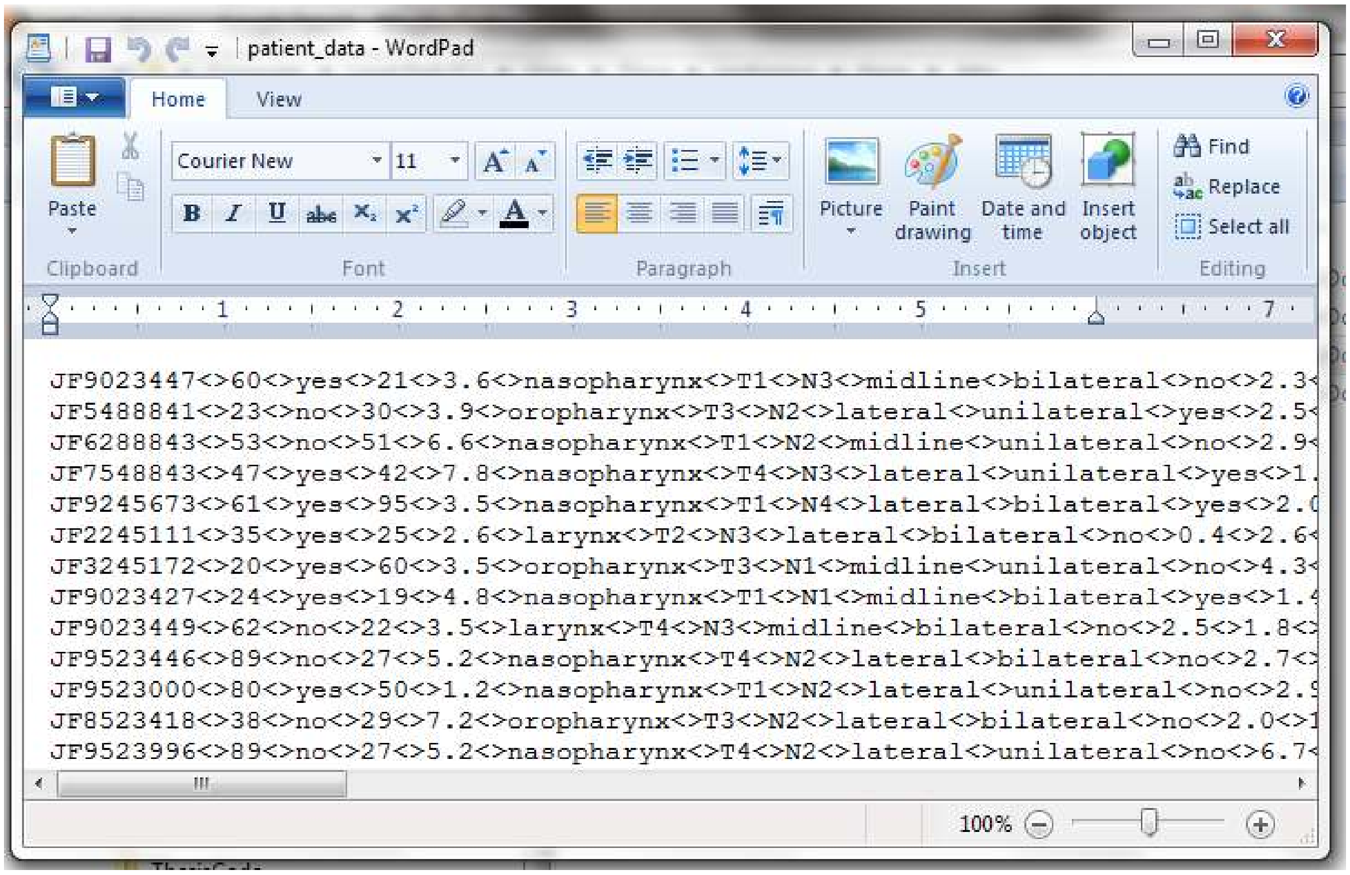
8) Project Engine: This module controls the operation of the system and calls all the other modules only when needed. However, it also allows interaction between modules to a specified extent when some functionality or data is needed between two or more modules.

The reader is referred to the appendix (System Code section) to view implementation of the above modules.

5 RESULTS

A sample of 13 dummy records was used in testing the system with data that mimics the real data processed from the patient records. The data was stored in ANSI format (text files) as the data processing system is being developed to migrate to a database system. A screen shot of the sample records is shown in figure 5-1 below for reference in the subsequent discussions. The character combination "**<>**" is used as the field delimiter (to mark separation of the different fields in the database). The fields are organized in the order:

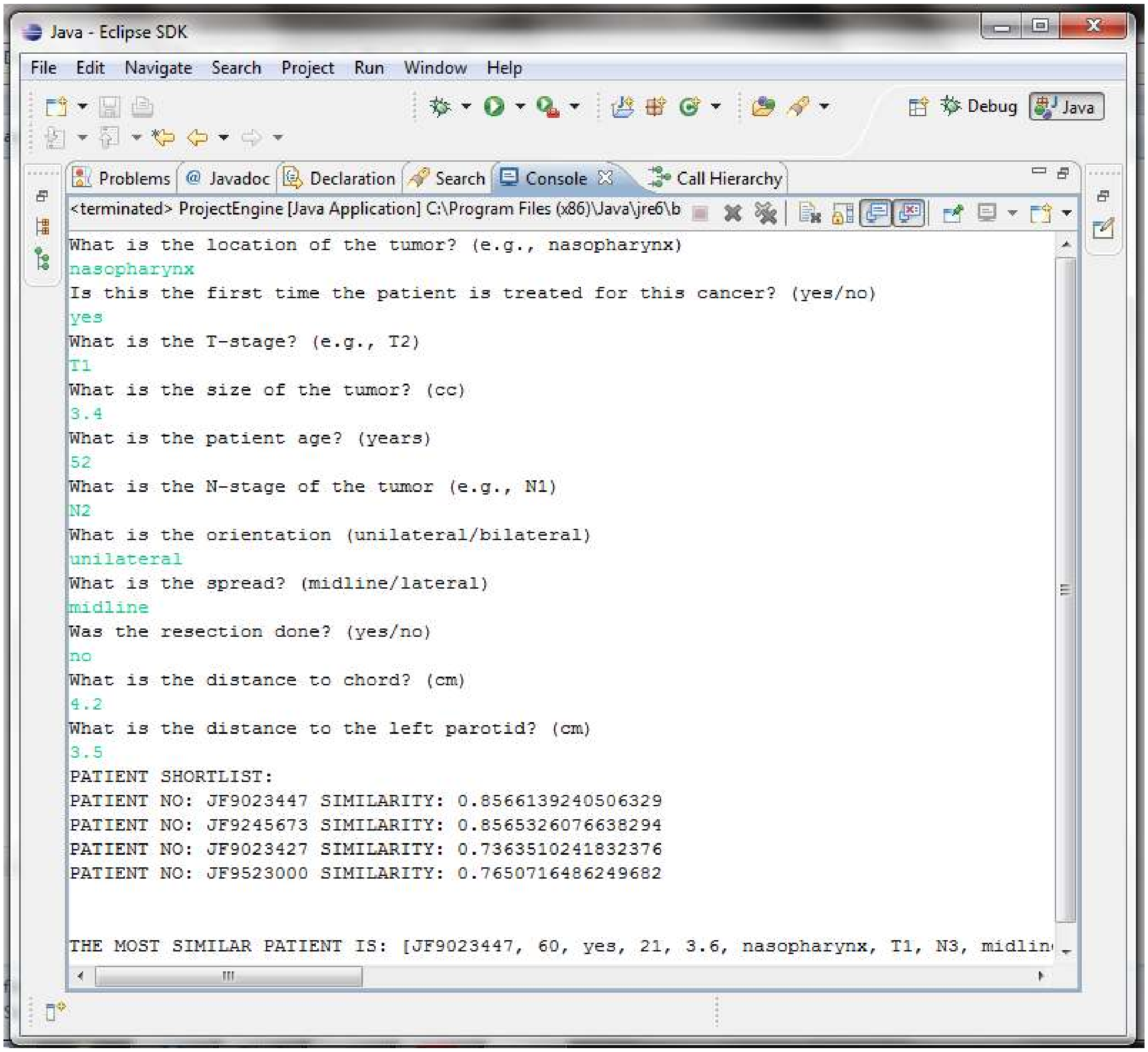
Case Number, Patient Age, Retreat, Patient BMI, Tumor Volume, Tumor Location, T Stage, N Stage, Orientation, Spread, Resection, Distance to Cord, Distance to Left Parotid, Diagnosis Comments, Minimum PTV54 Dosage, Minimum PTV60 Dosage, Minimum CTV60 Dosage, Maximum Cord Dosage, Spinal Cord DVH, Mandible DVH, Treatment Time, Number of Fractionations, Treatment Comments



```
JF9023447<>60<>yes<>21<>3.6<>nasopharynx<>T1<>N3<>midline<>bilateral<>no<>2.3<
JF5488841<>23<>no<>30<>3.9<>oropharynx<>T3<>N2<>lateral<>unilateral<>yes<>2.5<
JF6288843<>53<>no<>51<>6.6<>nasopharynx<>T1<>N2<>midline<>unilateral<>no<>2.9<
JF7548843<>47<>yes<>42<>7.8<>nasopharynx<>T4<>N3<>lateral<>unilateral<>yes<>1.
JF9245673<>61<>yes<>95<>3.5<>nasopharynx<>T1<>N4<>lateral<>bilateral<>yes<>2.
JF2245111<>35<>yes<>25<>2.6<>larynx<>T2<>N3<>lateral<>bilateral<>no<>0.4<>2.6<
JF3245172<>20<>yes<>60<>3.5<>oropharynx<>T3<>N1<>midline<>unilateral<>no<>4.3<
JF9023427<>24<>yes<>19<>4.8<>nasopharynx<>T1<>N1<>midline<>bilateral<>yes<>1.
JF9023449<>62<>no<>22<>3.5<>larynx<>T4<>N3<>midline<>bilateral<>no<>2.5<>1.8<>
JF9523446<>89<>no<>27<>5.2<>nasopharynx<>T4<>N2<>lateral<>bilateral<>no<>2.7<>
JF9523000<>80<>yes<>50<>1.2<>nasopharynx<>T1<>N2<>lateral<>unilateral<>no<>2.
JF8523418<>38<>no<>29<>7.2<>oropharynx<>T3<>N2<>lateral<>bilateral<>no<>2.0<>
JF9523996<>89<>no<>27<>5.2<>nasopharynx<>T4<>N2<>lateral<>unilateral<>no<>6.7<
```

Figure 5-1: Snapshot of the patient database

When the system is run, the shortlist of the filtered records or the most similar patient record can be returned. The screenshot in figure 5-2 below shows the results returned when a search is performed on the above data.


Java - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Debug Java
Problems Javadoc Declaration Search Console Call Hierarchy

```
<terminated> ProjectEngine [Java Application] C:\Program Files (x86)\Java\jre6\b
What is the location of the tumor? (e.g., nasopharynx)
nasopharynx
Is this the first time the patient is treated for this cancer? (yes/no)
yes
What is the T-stage? (e.g., T2)
T1
What is the size of the tumor? (cc)
3.4
What is the patient age? (years)
52
What is the N-stage of the tumor (e.g., N1)
N2
What is the orientation (unilateral/bilateral)
unilateral
What is the spread? (midline/lateral)
midline
Was the resection done? (yes/no)
no
What is the distance to chord? (cm)
4.2
What is the distance to the left parotid? (cm)
3.5
PATIENT SHORTLIST:
PATIENT NO: JF9023447 SIMILARITY: 0.8566139240506329
PATIENT NO: JF9245673 SIMILARITY: 0.8565326076638294
PATIENT NO: JF9023427 SIMILARITY: 0.7363510241832376
PATIENT NO: JF9523000 SIMILARITY: 0.7650716486249682


THE MOST SIMILAR PATIENT IS: [JF9023447, 60, yes, 21, 3.6, nasopharynx, T1, N3, midlin
```

Figure 5-2: Results of a search performed on the patient database

The results show that despite 4 records having met the requirements of the main attributes they are not all equally similar to the entered candidate information. Based on the weighting of the attributes due to

CBR, and computations discussed in section 4-1, the similarity of each patient is determined. It is found that the patient with record number “JF9023447” is the most similar patient with similarity of approximately 0.857/1.000. The record of this patient is therefore retrieved from the database and displayed on the screen.

A GUI is being developed to replace the text-based console for future iterations of this system as discussed in the next chapter. The generated results data will be passed to the GUI for a more user-friendly display in the improvements that are being carried out on the system. However, the prototype results serve to illustrate the viability of the system proposed in this thesis.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

Advancements in technology introduce new opportunities to improve human healthcare services. This has been proven by the progress made in diagnosis, patient care and quality of life of cancer patients after treatment in the United States of America.

This thesis proposes an approach to aid a head and neck cancer treatment team of a hospital by suggesting reference treatment plans when developing new plans for IMRT candidates. It makes use of the amassed hospital experience and data by capturing it in the expert system reasoning when determining the most appropriate reference patient. Therefore, this thesis capitalizes on the progress made in research for the field of Artificial Intelligence to devise ways of improving diagnosis and shortening periods taken by the cancer team to prepare treatment plans. This, it is hoped, will help cut cancer treatment costs for both the patients and healthcare institutions involved while acting as a reliable aid to the cancer experts.

6.2 Future Work

The overall system being developed is illustrated in a schematic in figure 4-2. The scope of this thesis only covered the search functionality. Work has already been done on adapting the treatment plan by another team. The following areas are still pending:

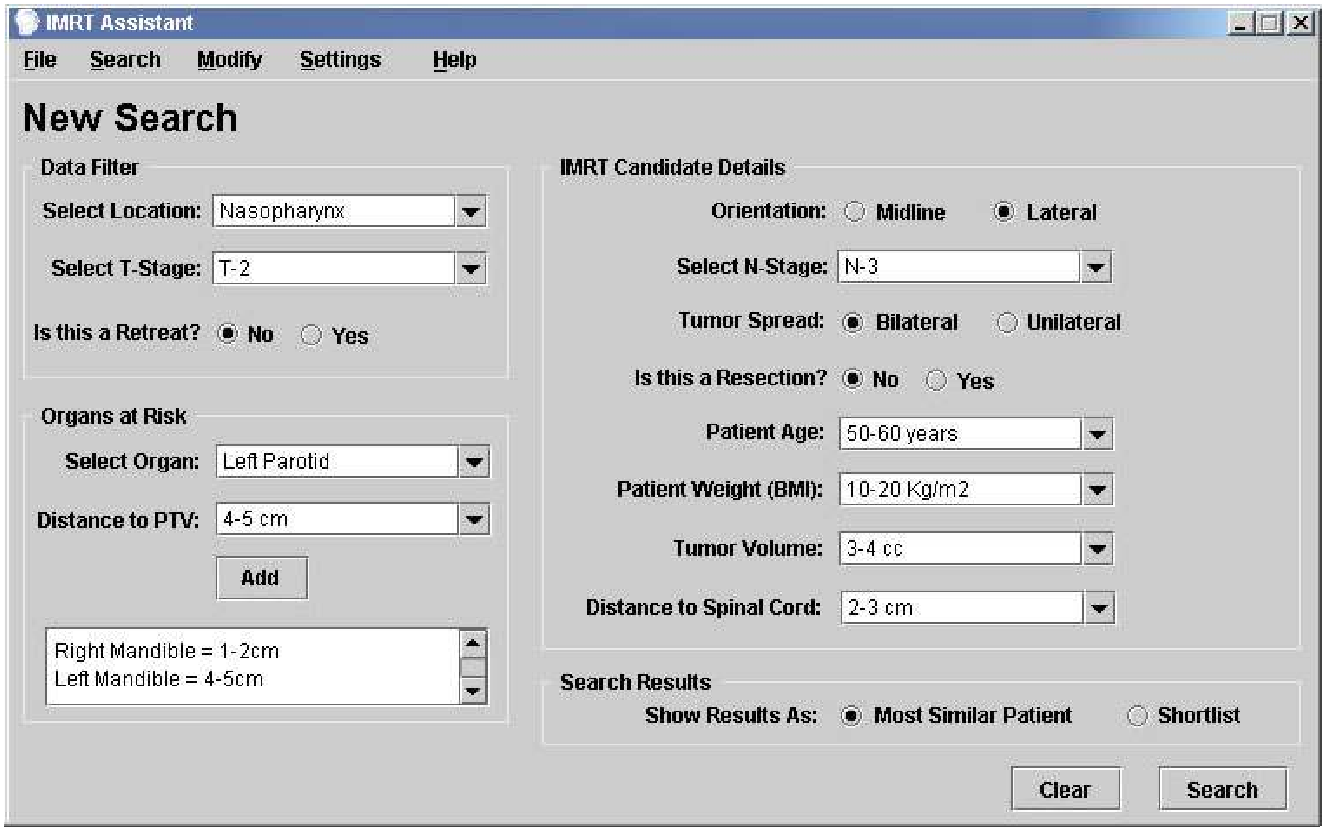
a) Processing the raw data from the hospital database to obtain data the can be efficiently searched and indexed.

b) Addition of more weighting rules to the knowledge base and connecting the search system to a more manageable database system.

c) Adapting the treatment plan of the most similar patient(s) to the candidate’s requirements.

d) Adding a user-friendly interface to allow flexible management of the knowledge base. Such functionality like importing raw data from the hospital database, exporting results for use in other systems, printing, emailing or future reference and modification of the definition and composition of the fuzzy variables, fuzzy values and patient variables should be easily changeable without directly interfacing with the database.

e) Developing a user interface that coordinates all the above systems while remaining simple, secure and user friendly. A simulation of the proposed user interface is illustrated in figures 6-1 and 6-2 to show the future of the project.



New Search

Data Filter

Select Location: Nasopharynx

Select T-Stage: T-2

Is this a Retreat? [✓ No] Yes

Organs at Risk

Select Organ: Left Parotid

Distance to PTV: 4-5 cm

Right Mandible = 1-2cm
Left Mandible = 4-5cm

IMRT Candidate Details

Orientation: Midline [✓ Lateral]

Select N-Stage: N-3

Tumor Spread: [✓ Bilateral] Unilateral

Is this a Resection? [✓ No] Yes

Patient Age: 50-60 years

Patient Weight (BMI): 10-20 Kg/m2

Tumor Volume: 3-4 cc

Distance to Spinal Cord: 2-3 cm

Search Results

Show Results As: [✓ Most Similar Patient] Shortlist

Figure 6-1: Screenshot showing a new search for an IMRT candidate

The system will allow:

a) Importation and exportation of data and outputs respectively under the File menu option,

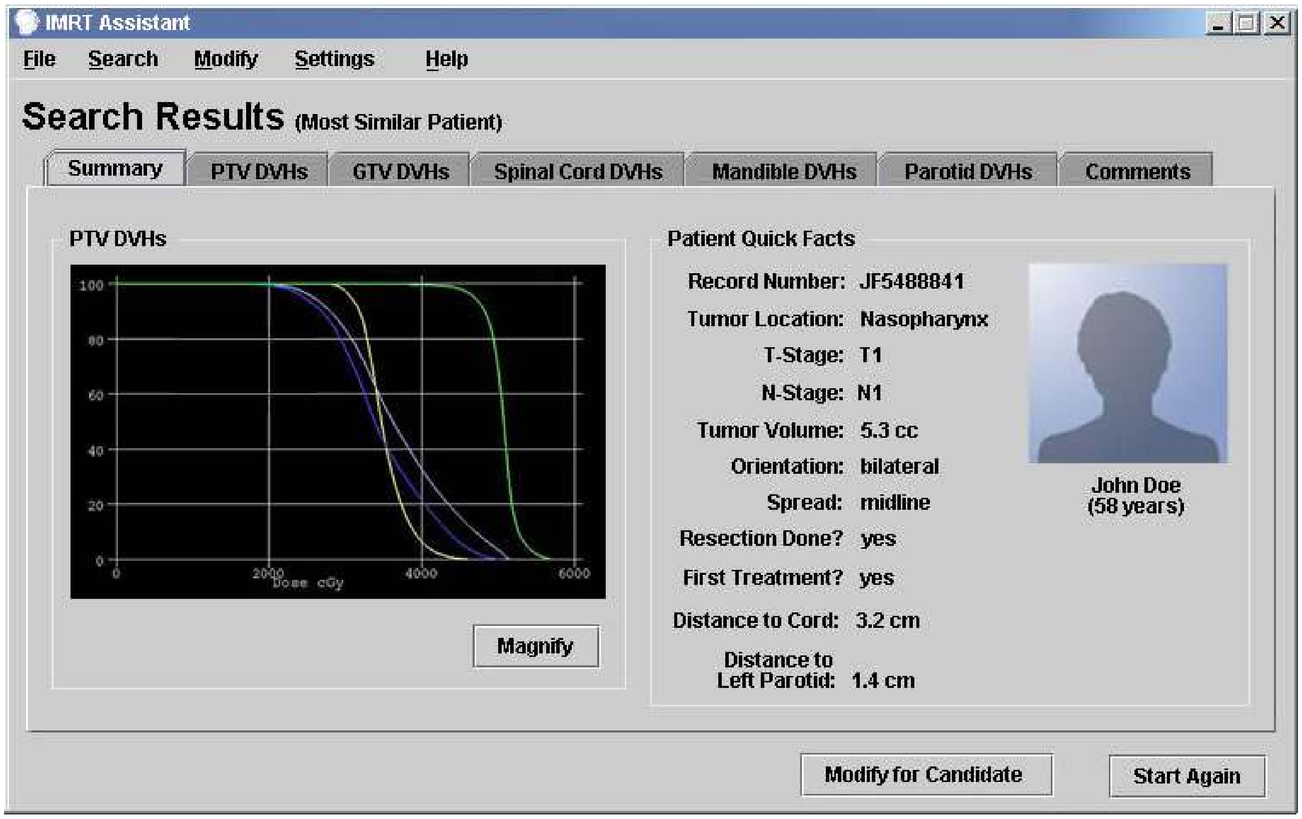
b) Under the Search menu option, the system will allow searching based on the specified candidate parameters and displaying a shortlist of the qualifying patients with their corresponding similarity rank (or only the most similar patient). It will also allow saving the search result to the results database for future reference. All back-end functionality for this part has been implemented in this thesis.

c) Under the Modify menu option, the user can make changes to the most similar patient treatment plan(s) and preview the effect on the final generated treatment plan for the IMRT candidate.

d) Under the Settings menu option, the user can make changes to the Knowledge Base settings such as fuzzy variable definitions, patient variables among others.

e) Under the Help menu option, there will be a searchable help which provides user-friendly guidance on how the system can be used.

Figure 6-2 is a simulation of search results displaying a record of the most similar patient returned from the database. It shows the various views (in form of tabs) that will be available for the user to review the full patient data before making the decision to use the patient record as a reference.



Patient Quick Facts

Record Number:	JF5488841
Tumor Location:	Nasopharynx
T-Stage:	T1
N-Stage:	N1
Tumor Volume:	5.3 cc
Orientation:	bilateral
Spread:	midline
Resection Done?	yes
First Treatment?	yes
Distance to Cord:	3.2 cm
Distance to Left Parotid:	1.4 cm

John Doe
(58 years)

Figure 6-2: Screenshot showing a record of the most similar patient returned from the search

REFERENCES

[1] Lanceford M. Chong and Margie A. Hunt, "A practical guide to intensity-modulated radiation therapy" Chapter 10, pp: 192, by Memorial Sloan-Kettering Cancer Center (2003).

[2] http://www.uihealthcare.com/topics/cancer/canc4285.html as at January 5, 2010

[3] Matthew T. Studenski, Ph. D., Medical Physics Resident, "Head and Neck Classes", Bodine Center for Cancer Treatment, Thomas Jefferson University Hospital, (2010).

[4] Xueyan Song, Sanja Petrovic, Santhanam Sundar, "A Case-Based Reasoning Approach to Dose Planning in Radiotherapy", Department of Oncology, Nottingham University Hospitals NHS Trust (UK) [Not Dated].

[5] Jeffrey Berger, "Roentgen: Radiation Therapy and Case-based Reasoning", Proceeding of the 10th Conference in Artificial Intelligence Applications (1994).

[6] Ani1 K. Jain, "Fundamentals of Digital Image Processing", Prentice Hall, Englewood Cliffs, N.J. (1989).

[7] Phyllis Koton, "Using Experience in Learning and Problem Solving", Thesis for Doctor of Philosophy, Massachusetts Institute of Technology (MIT) – Library Archive (1988).

[8] Janet L. Kolodner, "Maintaining Organization in a Dynamic Long Term Memory", Cognitive Science (1983).

[9] William J. Long, Shapur Naimi, M. G. Criscitiello, and Robert Jayes, "The development and use of a causal model for reasoning about heart failure.", IEEE Symposium on Computer Applications in Medical Care, pp 30-36, (1987).

[10] http://www.taxotere.com/consumer/headneck_cancer/about.aspx as at November 4, 2009

[11] Brian P. Butz, “Choosing Similar Patients”, Intelligent Systems Application Center (ISAC), Temple University (2010).

[12] http://www.eclipse.org/ as at April 12, 2010

[13] http://www.sun.com/ as at April 12, 2010

[14] R. A. Orchard, “FuzzyCLIPS Users Guide”, Version 6.04A, Integrated Reasoning Institute for Information Technology, National Research Council Canada (NRCC), (1998).

[15] Michael Negnevitsky, “Artificial Intelligence; A Guide to Intelligent Systems”, 2nd Edition, Pearson Education Publishers, pp: 87-126, (2005).

[16] John L. Meyer, “IMRT, IGRT, SBRT: advances in the treatment planning and delivery of radiotherapy”, Vol. 40, First Edition, S. Karger AG (Switzerland), pp: 40-57, (2007).

[17] Thomas Bortfeld, Rupert Schmidt-Ullrich, Wilfried De Neve, David E. Wazer, “Image-Guided IMRT”, First Edition, Springer Publishers, Chapter 12, (2005).

BIBLIOGRAPHY

Jeffrey Berger, “Roentgen: Radiation Therapy and Case-based Reasoning”, Proceeding of the 10th Conference in Artificial Intelligence Applications (1994).

Thomas Bortfeld, Rupert Schmidt-Ullrich, Wilfried De Neve, David E. Wazer, “Image-Guided IMRT”, First Edition, Springer Publishers, Chapter 12, (2005).

Brian P. Butz, “Choosing Similar Patients”, Intelligent Systems Application Center (ISAC), Temple University (2010).

Lanceford M. Chong and Margie A. Hunt, “A practical guide to intensity-modulated radiation therapy” Chapter 10, pp: 192, by Memorial Sloan-Kettering Cancer Center (2003).

Ani1 K. Jain, “Fundamentals of Digital Image Processing”, Prentice Hall, Englewood Cliffs, N.J. (1989).

Janet L. Kolodner, “Maintaining Organization in a Dynamic Long Term Memory”, Cognitive Science (1983).

Phyllis Koton, “Using Experience in Learning and Problem Solving”, Thesis for Doctor of Philosophy, Massachusetts Institute of Technology (MIT) – Library Archive (1988).

William J. Long, Shapur Naimi, M. G. Criscitiello, and Robert Jayes, “The development and use of a causal model for reasoning about heart failure.”, IEEE Symposium on Computer Applications in Medical Care, pp 30-36, (1987).

John L. Meyer, “IMRT, IGRT, SBRT: advances in the treatment planning and delivery of radiotherapy”, Vol. 40, First Edition, S. Karger AG (Switzerland), pp: 40-57, (2007).

Michael Negnevitsky, “Artificial Intelligence; A Guide to Intelligent Systems”, 2nd Edition, Pearson Education Publishers, pp: 87-126, (2005).

R. A. Orchard, “FuzzyCLIPS Users Guide”, Version 6.04A, Integrated Reasoning Institute for Information Technology, National Research Council Canada (NRCC), (1998).

Xueyan Song, Sanja Petrovic, Santhanam Sundar, “A Case-Based Reasoning Approach to Dose Planning in Radiotherapy”, Department of Oncology, Nottingham University Hospitals NHS Trust (UK) [Not Dated].

Matthew T. Studenski, Ph. D., Medical Physics Resident, “Head and Neck Classes”, Bodine Center for Cancer Treatment, Thomas Jefferson University Hospital, (2010).

http://www.eclipse.org/ as at April 12, 2010 – The official website of the Eclipse Foundation and source of the Eclipse IDE.

http://www.sun.com/ as at April 12, 2010 – The official website of Sun Microsystems, acquired by Oracle on January 27, 2010.

http://www.taxotere.com/consumer/headneck_cancer/about.aspx as at November 4, 2009 – A comprehensive source of information and images for head and neck cancer.

http://www.uihealthcare.com/topics/cancer/canc4285.html as at January 5, 2010 – A comprehensive source of information on the composition of a cancer treatment team.

APPENDICES

System Code

ATTRIBUTE EXTRACTOR MODULE

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class AttributeExtractor {

      public ArrayList<ArrayList<String>> getPatientAttributes() throws
IOException{
            //Variables to be used in this function
      String vLine = "";
      ArrayList<String> mainAttributes = new ArrayList<String>();
      ArrayList<String> mainAttributes_qns = new ArrayList<String>();
      ArrayList<String> otherAttributes = new ArrayList<String>();
      ArrayList<String> otherAttributes_qns = new ArrayList<String>();

        File patientVariables = new
File("data\\define_patientvariables.txt");
      // Open the file with the patient variables
        FileInputStream patientStream = new
FileInputStream(patientVariables);
        // Convert our input stream to a DataInputStream
        BufferedReader fBuffer = new BufferedReader(new
InputStreamReader(patientStream));

        int mainCounter = 0;
        int otherCounter = 0;
        while ((vLine = fBuffer.readLine()) != null){
            String[] row_question = vLine.split("<>");

            //Check whether this is one of the MAIN variables
            //and its respective question
            if(row_question[0].contains("MAIN**")){
                  mainAttributes.add(mainCounter,
row_question[0].substring(6));
                  mainAttributes_qns.add(mainCounter, row_question[1]);
                  mainCounter++;

            //Get the other variable and its
            } else {
                  otherAttributes.add(otherCounter, row_question[0]);
                  otherAttributes_qns.add(otherCounter, row_question[1]);
                  otherCounter++;
            }
        }

        ArrayList<ArrayList<String>> cleanVariables = new
ArrayList<ArrayList<String>>();

        cleanVariables.add(0,mainAttributes);
        cleanVariables.add(1,mainAttributes_qns);
        cleanVariables.add(2,otherAttributes);
        cleanVariables.add(3,otherAttributes_qns);

      return cleanVariables;
    }



      /*
       * Returns string formats of all fuzzy variables defined for the
system.
       */
      public ArrayList<ArrayList<String>> getStringFuzzyVariables() throws
IOException {
            ArrayList<String> defaultWeights = new ArrayList<String>();
            ArrayList<String> stringFuzzyVariables = new ArrayList<String>();
            ArrayList<String> stringFuzzyValues = new ArrayList<String>();
            //Saves all info that has been extracted for further use in the
            //generation of the fuzzy rules
            ArrayList<ArrayList<String>> allExtracts = new
ArrayList<ArrayList<String>>();

            File fuzzyVariables = new
File("data\\define_fuzzyvariables.txt");
      // Open the file with the fuzzy variables
        FileInputStream fuzzyStream = new FileInputStream(fuzzyVariables);
        // Convert our input stream to a DataInputStream
        BufferedReader fsBuffer = new BufferedReader(new
InputStreamReader(fuzzyStream));

        String fLine = "";
        int fuzzyVCounter = 0, rowCounter = 0;
        while ((fLine = fsBuffer.readLine()) != null){
            String[] fRow = fLine.split("=");

            //If defining a new fuzzy variable
            if(fRow[0].contains("**")){
                  stringFuzzyVariables.add(fuzzyVCounter,
fLine.substring(2));
                  String[] fuzzyVAttributes = fRow[1].split("<>");
                  defaultWeights.add(fuzzyVCounter, fuzzyVAttributes[3]);
                  fuzzyVCounter++;

            //If defining the fuzzy values
            } else {
                  stringFuzzyValues.add(rowCounter, (fuzzyVCounter-
1)+":"+fLine);
                  rowCounter++;
            }
        }

        allExtracts.add(0, stringFuzzyVariables);
        allExtracts.add(1, stringFuzzyValues);
        allExtracts.add(2, defaultWeights);

            return allExtracts;
      }


      /*
       * Returns string formats of all fuzzy rules defined for the system.
       */
      public ArrayList<ArrayList<String>> getStringFuzzyRules() throws
IOException{
            ArrayList<ArrayList<String>> allRules = new
ArrayList<ArrayList<String>>();
            //Arrays to track the antecedents and conclusions
            ArrayList<String> stringAntecedents = new ArrayList<String>();
            ArrayList<String> stringConclusions = new ArrayList<String>();

            File fuzzyRules = new File("data\\weighting_rules.txt");
      // Open the file with the fuzzy rules
        FileInputStream fuzzyRuleStream = new FileInputStream(fuzzyRules);
        // Convert our input stream to a DataInputStream
        BufferedReader frBuffer = new BufferedReader(new
InputStreamReader(fuzzyRuleStream));

        String fLine = "";
        int fuzzyRCounter = 0, rowCounter = 0;
        while ((fLine = frBuffer.readLine()) != null){
            String[] fRow = fLine.split("=");
            //Save the conclusions and antecedents separately to ease
processing
            if(fRow[0].contains("**")){
                  stringConclusions.add(fuzzyRCounter, fLine.substring(2));
                  fuzzyRCounter++;
            } else {
                  stringAntecedents.add(rowCounter, (fuzzyRCounter-
1)+":"+fLine);
                  rowCounter++;
            }
        }

        allRules.add(0, stringAntecedents);
        allRules.add(1, stringConclusions);

            return allRules;
      }
}
```

CONVERTER MODULE

```
import java.util.*;

public class Converter {

      /*
       * Get the tumor volume value given the entered value
       */
      public double getTVolume(String attributeValue){
            return Double.valueOf(attributeValue).doubleValue();
      }

      /*
       * Get the patient age value given the entered value
       */
      public double getPatientAge(String attributeValue){
            return Double.valueOf(attributeValue).doubleValue();
      }

      /*
       * Get the n-stage value given the entered value
       */
      public double getNStage(String attributeValue){
            if(attributeValue.equalsIgnoreCase("N1")){
                  return 1;
            } else if(attributeValue.equalsIgnoreCase("N2")){
                  return 2;
            } else if(attributeValue.equalsIgnoreCase("N3")){
                  return 3;
            } else if(attributeValue.equalsIgnoreCase("N4")){
                  return 4;
            } else {
                  return 0;
            }
      }

      /*
       * Get the orientation value given the entered value
       */
      public double getOrientation(String attributeValue){
            if(attributeValue.equalsIgnoreCase("midline")){
                  return 2;
            } else if(attributeValue.equalsIgnoreCase("lateral")){
                  return 1;
            } else {
                  return 0;
            }
      }

      /*
       * Get the spread value given the entered value
       */
      public double getSpread(String attributeValue){
            if(attributeValue.equalsIgnoreCase("bilateral")){
```

```
			return 2;
		} else if(attributeValue.equalsIgnoreCase("unilateral")){
			return 1;
		} else {
			return 0;
		}
	}

	/*
	 * Get the resection value given the entered value
	 */
	public double getResection(String attributeValue){
		if(attributeValue.equalsIgnoreCase("yes")){
			return 2;
		} else if(attributeValue.equalsIgnoreCase("no")){
			return 1;
		} else {
			return 0;
		}
	}

	/*
	 * Get the distance to chord value given the entered value
	 */
	public double getDistToChord(String attributeValue){
		return Double.valueOf(attributeValue).doubleValue();
	}

	/*
	 * Get the distance to left parotid value given the entered value
	 */
	public double getDistToLeftParotid(String attributeValue){
		return Double.valueOf(attributeValue).doubleValue();
	}

	/*
	 * Get the distance to right parotid value given the entered value
	 */
	public double getDistToRightParotid(String attributeValue){
		return Double.valueOf(attributeValue).doubleValue();
	}

	/*
	 * Get the default value given the entered value
	 */
	public double getDefault(String attributeValue){
		return 0;
	}

	/*
	 * Get the position of the passed attribute for a patient record
	 *
	 * NOTE: This should be updated whenever the order of the text values
	 * in the text file is changed.
	 */
	public double getAttributePosition(String attribute){
		ArrayList<String> patientAttributeArray = new 
ArrayList<String>();
		patientAttributeArray.add(0, "DIAGNOSIS_case_number");
		patientAttributeArray.add(1, "DIAGNOSIS_patient_age");
		patientAttributeArray.add(2, "DIAGNOSIS_retreat");
		patientAttributeArray.add(3, "DIAGNOSIS_patient_bmi");
		patientAttributeArray.add(4, "DIAGNOSIS_tumor_volume");
		patientAttributeArray.add(5, "DIAGNOSIS_tumor_location");
		patientAttributeArray.add(6, "DIAGNOSIS_t_stage");
		patientAttributeArray.add(7, "DIAGNOSIS_n_stage");
		patientAttributeArray.add(8, "DIAGNOSIS_orientation");
		patientAttributeArray.add(9, "DIAGNOSIS_spread");
		patientAttributeArray.add(10, "DIAGNOSIS_resection");
		patientAttributeArray.add(11, "DIAGNOSIS_dist_to_chord");
		patientAttributeArray.add(12, "DIAGNOSIS_dist_to_leftparotid");
		patientAttributeArray.add(13, "DIAGNOSIS_comments");
		
		patientAttributeArray.add(14, "TREATMENT_min_PTV54_dosage");
		patientAttributeArray.add(15, "TREATMENT_min_PTV60_dosage");
		patientAttributeArray.add(16, "TREATMENT_min_CTV60_dosage");
		patientAttributeArray.add(17, "TREATMENT_max_chord_dosage");
		patientAttributeArray.add(18, "TREATMENT_spinal_chord_DVH");
		patientAttributeArray.add(19, "TREATMENT_mandible_DVH");
		patientAttributeArray.add(20, "TREATMENT_treatment_time");
		patientAttributeArray.add(21, "TREATMENT_no_of_fractionations");
		patientAttributeArray.add(22, "TREATMENT_comments");
		
		return 
Double.valueOf(patientAttributeArray.indexOf(attribute)).doubleValue();
	}
	
	/*
	 * Returns the number equivalent of the attribute given its attribute 
name
	 * and the value
	 */
	public double getNumValue(String attributeName, String attributeValue){
		double numValue=0;
		
		if(attributeName.equalsIgnoreCase("DIAGNOSIS_tumor_volume")){
			numValue = getTVolume(attributeValue);
			
		}else 
if(attributeName.equalsIgnoreCase("DIAGNOSIS_patient_age")){
			numValue = getPatientAge(attributeValue);
			
		}else if(attributeName.equalsIgnoreCase("DIAGNOSIS_n_stage")){
			numValue = getNStage(attributeValue);
			
		}else 
if(attributeName.equalsIgnoreCase("DIAGNOSIS_orientation")){
			numValue = getOrientation(attributeValue);
			
		}else if(attributeName.equalsIgnoreCase("DIAGNOSIS_spread")){
```

```
                    numValue = getSpread(attributeValue);

              }else if(attributeName.equalsIgnoreCase("DIAGNOSIS_resection")){
                    numValue = getResection(attributeValue);

              }else
if(attributeName.equalsIgnoreCase("DIAGNOSIS_dist_to_chord")){
                    numValue = getDistToChord(attributeValue);

              }else
if(attributeName.equalsIgnoreCase("DIAGNOSIS_dist_to_leftparotid")){
                    numValue = getDistToLeftParotid(attributeValue);

              }else
if(attributeName.equalsIgnoreCase("DIAGNOSIS_dist_to_rightparotid")){
                    numValue = getDistToRightParotid(attributeValue);

              }else {
                    numValue = getDefault(attributeValue);
              }

              return numValue;
        }


        /*
         * Returns the value of an attribute given the attribute name.
         */
        public String getAttrValue(String attribute, String attributeValue){
              double valueAmount = 0;

              if(attribute.equalsIgnoreCase("DIAGNOSIS_tumor_volume")){
                    valueAmount = getTVolume(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_patient_age")){
                    valueAmount =  getPatientAge(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_n_stage")){
                    valueAmount = getNStage(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_orientation")){
                    valueAmount = getOrientation(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_spread")){
                    valueAmount = getSpread(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_resection")){
                    valueAmount = getResection(attributeValue);

              } else if(attribute.equalsIgnoreCase("DIAGNOSIS_dist_to_chord")){
                    valueAmount = getDistToChord(attributeValue);

              } else
if(attribute.equalsIgnoreCase("DIAGNOSIS_dist_to_leftparotid")){
                    valueAmount = getDistToLeftParotid(attributeValue);
```

```
            } else
if(attribute.equalsIgnoreCase("DIAGNOSIS_dist_to_rightparotid")){
                  valueAmount = getDistToRightParotid(attributeValue);

            } else{
                  valueAmount = getDefault(attributeValue);

            }
            return Double.toString(valueAmount);
      }
}
```

FUZZIFIER MODULE

```
import java.io.IOException;
import java.util.*;

import nrc.fuzzy.*;


public class Fuzzifier {
      AttributeExtractor iExtractor = new AttributeExtractor();
      Converter iConverter = new Converter();
      double tolerance = 1;


      /*
       * Takes an attribute and its value and returns its weight after
       * running it through the defined fuzzy rules
       */
      public double getAttributeWeight(String attribute, ArrayList<String>
      values) throws IOException{
            double attributeWeight = 0;
            FuzzyValue.setConfineFuzzySetsToUOD(true);

            //Get the text representation of the fuzzy variables from their
      file
            ArrayList<ArrayList<String>> allStringFVariables =
      iExtractor.getStringFuzzyVariables();
            //Generate the fuzzy variables ready for use to create the rules
            ArrayList<FuzzyVariable> allFuzzyVariables =
      generateFuzzyVars(allStringFVariables);

            //Get the text representation of the rules from their file
            ArrayList<ArrayList<String>> allStringFRules =
      iExtractor.getStringFuzzyRules();
            double defaultWeight = getDefaultWeight(attribute,
      allStringFVariables);
            //Variables for defining the modifying (conclusion) variable
            double center = 0.5; double modificationTolerance = 1-
      defaultWeight;

            try {
                  RightLinearFunction rlf = new RightLinearFunction();
                  LeftLinearFunction llf = new LeftLinearFunction();
                  //The conclusion from the fuzzy rules
                  //e.g., L<>0.3<>0.5, R<>0.5<>0.7
                  FuzzyVariable CFModifier = new FuzzyVariable("cfmodifier",
      0, 1, "");
                  CFModifier.addTerm("lower", new LFuzzySet((center-
      modificationTolerance), center, llf));
                  CFModifier.addTerm("raise", new
      RFuzzySet(center,(center+modificationTolerance), rlf));

                  //Generate the fuzzy rules given the rules string
      definitions, antecedent
                  //fuzzy variables and the conclusion fuzzy variable
                  ArrayList<FuzzyRule> allFuzzyRules =
      generateFuzzyRules(allStringFRules,
                              allFuzzyVariables, allStringFVariables.get(0),
      CFModifier);

                  //Add the attribute values provided for the new candidate
                  for(int i=0; i<allStringFRules.get(1).size(); i++){
                        allFuzzyRules.get(i).removeAllInputs();

                        //Check the variables passed for each rule
                        for(int k=0; k<allStringFRules.get(0).size(); k++){
                              String[] counter_var =
      allStringFRules.get(0).get(k).split(":");
                              if(Integer.parseInt(counter_var[0]) == i){
                                    //e.g., DIAGNOSIS_dist_to_chord=small
                                    String[] var_value =
      counter_var[1].split("=");
                                    int fPosition =
      getFuzzyVarPosition(allStringFVariables.get(0), var_value[0]);

                                    //Record input for this rule if a fuzzy
      variable exists for it
                                    if(fPosition != -1){
                                          //Get the input value in number
      format
                                          double inputValue = Double.valueOf(

            iConverter.getAttrValue(var_value[0],
      values.get(fPosition))).doubleValue();

                                          //Get the upper and lower limits of
      the fuzzy variables
                                          double [] fuzzyVLimits =
      getFuzzyVLimits(allFuzzyVariables.get(fPosition),
      Double.toString(inputValue), tolerance);

                                          //e.g.,
      coldTempRule.addAntecedent(new FuzzyValue(airTemp,[32,33,34]));
                                          allFuzzyRules.get(i).addInput(
                                                      new
      FuzzyValue(allFuzzyVariables.get(fPosition),
                                                            //Fuzzy set
      returns modified input with tolerance
```

```
                                                            //e.g.,
TriangleFuzzySet(30,32,34)
                                                            new
TriangleFuzzySet(fuzzyVLimits[0],fuzzyVLimits[1],fuzzyVLimits[2])));
                                    }
                              }
                        }
                  }


                  //Run the fuzzy rules to determine the weight of the passed
attribute
                  //if the attribute fuzzy rule exists, else return the
weight as the default weight
                  int attributePos = attributeFuzzyRExists(attribute,
allStringFRules.get(1));
                  double adjustment = 0;
                  if(attributePos != -1){
                        try{
                              //Define the handlers for the class result
                              for(int i=0; i<allFuzzyRules.size(); i++){
                                    //System.out.println("\n\nRULE
("+i+"):\nANTECENDENTS: "+allFuzzyRules.get(i).getAntecedents()
                                                //+"\nCONCLUSION:
"+allFuzzyRules.get(i).getConclusions()
                                                //+"\nINPUTS:
"+allFuzzyRules.get(i).getInputs());
                                    String[] var_adjustment =
allStringFRules.get(1).get(attributePos).split("<>");

                                    // Test rule matching and if the rule can
fire, add the resultant
                                    // adjustment amount is got by performing
a fuzzy union
                                    if
(var_adjustment[0].equalsIgnoreCase(attribute)){

      if(allFuzzyRules.get(i).testRuleMatching(0)){
                                                adjustment =
getRuleCF(allFuzzyRules.get(i));
                                          }
                                    }
                              }

                              //double adjustment =
CFResult.momentDefuzzify();
                              attributeWeight = defaultWeight + (adjustment -
center);

                        } catch (IncompatibleRuleInputsException irie){
                              System.err.println("Incompatible rule inputs.
ERROR:\n "+irie);
                        }

                  }else {
```

```
                  attributeWeight = getDefaultWeight(attribute,
allStringFVariables);
            }



      } catch(InvalidFuzzyVariableNameException fvn){
            System.err.println("Invalid fuzzy variable name. ERROR:\n
"+fvn);
      } catch(InvalidUODRangeException uodr){
            System.err.println("Invalid universe of discourse. ERROR:\n
"+uodr);
      } catch(InvalidFuzzyVariableTermNameException fve){
            System.err.println("Invalid fuzzy variable term. ERROR:\n
"+fve);
      } catch (XValueOutsideUODException xve){
            System.err.println("X Value outside Universe of Discourse.
ERROR:\n "+xve);
      } catch (XValuesOutOfOrderException xvooe){
            System.err.println("X Values out of order. ERROR:\n
"+xvooe);
      }

      return attributeWeight;
}


/*
 * Generates and returns the actual fuzzy variables as defined in the
knowledge database
 */
public ArrayList<FuzzyVariable>
generateFuzzyVars(ArrayList<ArrayList<String>> allStringFVariables){
      ArrayList<FuzzyVariable> allFuzzyVs = new
ArrayList<FuzzyVariable>();
      //Extract the text representations to get the actual variables
      ArrayList<String> varDefinitions = allStringFVariables.get(0);
      ArrayList<String> varValues = allStringFVariables.get(1);
      //Declare the variable to be used in the for loop to track the
fuzzy variable
      FuzzyVariable myVariable =  null;


      //This confines all created fuzzy sets in their universe of
discourse
      FuzzyValue.setConfineFuzzySetsToUOD(true);

      try {
            //Break down the string representations and change them to
FuzzyVariables
            for(int i=0;i<varDefinitions.size();i++){
                  String[] variable_values =
varDefinitions.get(i).split("=");
                  String[] values_array =
variable_values[1].split("<>");
```

```
                    //Remove the empty unit indicator
                    if(values_array[2].equalsIgnoreCase("_")){
                          values_array[2] = "";
                    }

                    //e.g., airTemp = new FuzzyVariable("airTemperature",
  0.0, 100.0, "Deg C");
                    myVariable = new FuzzyVariable(variable_values[0],

        Float.valueOf(values_array[0]).floatValue(),//Convert to float
                          Float.valueOf(values_array[1]).floatValue(),
                          values_array[2]);

                    //Go through all the fuzzy values and pick those
  associated with the current
                    //variable position (and therefore with the variable)
                    for(int k=0;k<varValues.size();k++){
                          //e.g., 2:young=L<>25.0<>30.0
                          String[] counter_varvalue =
  varValues.get(k).split(":");
                          if(Integer.parseInt(counter_varvalue[0]) == i){
                                //e.g., young=L<>25.0<>30.0
                                //String[] fvalue_ranges =
  counter_varvalue[1].split("=");
                                myVariable = formFuzzyValue(myVariable,
  counter_varvalue[1]);
                          }
                    }

                    allFuzzyVs.add(i, myVariable);
              }

        //Cater for possible errors
        } catch (InvalidUODRangeException uod) {
        System.err.println("Range specified is invalid. ERROR:\n "+uod);
  } catch (InvalidFuzzyVariableNameException fve) {
        System.err.println("Fuzzy variable name specified is invalid.
  ERROR:\n "+fve);
        }


        return allFuzzyVs;
  }


/*
 * Function to return the Certainty Factor of the executed rule.
 * It takes the fuzzy rule as the input.
 */
public double getRuleCF(FuzzyRule frule){
  double similarity = 1.0;
  FuzzyValueVector antecedents = frule.getAntecedents();
  FuzzyValueVector inputs = frule.getInputs();
  try {
        for (int i=0; i<antecedents.size(); i++){
              double localSimilarity =
  antecedents.fuzzyValueAt(i).similarity(inputs.fuzzyValueAt(i));
              if (localSimilarity < similarity){
                    similarity = localSimilarity;
              }
        }
  } catch (IncompatibleFuzzyValuesException ife) {
        System.err.println("Could not compare fuzzy values. ERROR: \n" +
  ife);
        System.exit(100);
  }

  return similarity;
}



  /*
   * Given a variable and a string set of the value ranges, this function
   * generates the fuzzy value and returns the new fuzzy value
   */
  public FuzzyVariable formFuzzyValue(FuzzyVariable fVariable, String
  fValueRange){
        RightLinearFunction rlf = new RightLinearFunction();
    LeftLinearFunction llf = new LeftLinearFunction();
        String[] fValueSet = fValueRange.split("=");
        String[] fRangeSet = fValueSet[1].split("<>");

        FuzzyValue.setConfineFuzzySetsToUOD(true);

        //Try assigning the fuzzy values to their respective fuzzy
  variable
        //given the type that needs to be created
        try{
              //Right fuzzy set
              if(fRangeSet[0].equalsIgnoreCase("R")){
                    fVariable.addTerm(fValueSet[0],
                          new
  RFuzzySet(Float.valueOf(fRangeSet[1]).floatValue(),

        Float.valueOf(fRangeSet[2]).floatValue(), rlf));

              //Left fuzzy set
              } else if(fRangeSet[0].equalsIgnoreCase("L")){
                    fVariable.addTerm(fValueSet[0],
                          new
  LFuzzySet(Float.valueOf(fRangeSet[1]).floatValue(),

        Float.valueOf(fRangeSet[2]).floatValue(), llf));

              //Triangle fuzzy set
              } else if(fRangeSet[0].equalsIgnoreCase("T")){
                    fVariable.addTerm(fValueSet[0],
                          new
  TriangleFuzzySet(Float.valueOf(fRangeSet[1]).floatValue(),

        Float.valueOf(fRangeSet[2]).floatValue(),

        Float.valueOf(fRangeSet[3]).floatValue()));
              }

        } catch (XValuesOutOfOrderException xvr){
              System.err.println("X-Value out of order. ERROR:\n "+xvr);
        } catch (InvalidFuzzyVariableTermNameException fvte) {
              System.err.println("Invalid fuzzy variable term name.
  ERROR:\n "+fvte);
        } catch (XValueOutsideUODException xvo){
              System.err.println("X Value Range specified is invalid.
  ERROR:\n "+xvo);
        }

        return fVariable;
  }


  /*
   * Generate the rules defined in the string passed given the fuzzy
  variables
   */
  public ArrayList<FuzzyRule>
  generateFuzzyRules(ArrayList<ArrayList<String>> allStringFRules,
              ArrayList<FuzzyVariable> allFuzzyVariables,
  ArrayList<String> stringFVariables,
              FuzzyVariable conclusionFVariable){

        //Create a rule object to track all definitions for the rule
        FuzzyRule myRule = new FuzzyRule();

        ArrayList<FuzzyRule> allFuzzyRulesIn = new
  ArrayList<FuzzyRule>();
        ArrayList<String> antecedentsStrings = allStringFRules.get(0);
        ArrayList<String> conclusionsStrings = allStringFRules.get(1);
        int fPosition = 0;

        try{
              //Check the antecedents string and record all corresponding
  antecedents
              for(int i=0;i<conclusionsStrings.size();i++){
                    myRule.removeAllInputs();
                    myRule.removeAllAntecedents();
                    myRule.removeAllConclusions();
                    allFuzzyRulesIn.add(i, myRule);

                    for(int k=0;k<antecedentsStrings.size();k++){
                          String[] counter_antecedent =
  antecedentsStrings.get(k).split(":");
```

```
                        String[] var_value =
counter_antecedent[1].split("=");
                        fPosition =
getFuzzyVarPosition(stringFVariables,var_value[0]);

                        //Record antecedent for this rule if a fuzzy
variable exists for it
                        if((Integer.parseInt(counter_antecedent[0]) ==
i) && (fPosition != -1)){
                              //e.g., coldTempRule.addAntecedent(new
FuzzyValue(airTemp,"cold"));
                              allFuzzyRulesIn.get(i).addAntecedent(new
FuzzyValue(allFuzzyVariables.get(fPosition), var_value[1]));
                        }
                  }

                  //Record the rule conclusion
                  String[] var_conclusion =
conclusionsStrings.get(i).split("<>");
                  allFuzzyRulesIn.get(i).addConclusion(new
FuzzyValue(conclusionFVariable, var_conclusion[1]));

                  //Add the complete rule to the rules array
                  allFuzzyRulesIn.add(i, myRule);
            }

      }catch(InvalidLinguisticExpressionException ile){
            System.err.println("Invalid linguistic expression. ERROR:\n
"+ile);
      }

      return allFuzzyRulesIn;
}


/*
 * Determines if the passed fuzzy variable already exists
 * by checking the fuzzy variables string array.
 *
 * It returns the first position of the attribute in the rule array.
 */
public int attributeFuzzyRExists(String attribute, ArrayList<String>
fuzzyRConclusionArray){
      int position = -1;

      for(int i=0;i<fuzzyRConclusionArray.size();i++){
            String[] var_adjustment =
fuzzyRConclusionArray.get(i).split("<>");
            if(var_adjustment[0].equalsIgnoreCase(attribute)){
                  position = i;
                  break;
            }
      }

      return position;
```

```
  }


  /*
   * Returns the default weight of the passed attribute
   */
  public double getDefaultWeight(String attribute,
  ArrayList<ArrayList<String>> allStringFVariables){
        double defaultWeight = 0;

        for(int i=0; i<allStringFVariables.get(0).size();i++){
              String[] variable_values =
  allStringFVariables.get(0).get(i).split("=");

              if(attribute.equalsIgnoreCase(variable_values[0])){
                    defaultWeight =
  Double.parseDouble(allStringFVariables.get(2).get(i));
                    break;
              }
        }

        return defaultWeight;
  }


/*
 * Function to return the limits of a fuzzy variable.
 * In case they go beyond the allowed Universe of Discourse,
 * this function keeps them within the limits.
 */
private double[] getFuzzyVLimits (FuzzyVariable fv, String value, double
  tolerance){
  double minLimit = fv.getMinUOD();
    double maxLimit = fv.getMaxUOD();
    double lowerL = 0;
    double upperL = 0;
    double nvalue = Double.valueOf(value).doubleValue();
    double ntolerance = tolerance;

    lowerL = nvalue - ntolerance;
    upperL = nvalue + ntolerance;

        if(lowerL < minLimit){
              lowerL = minLimit;
        }

        if(upperL < minLimit){
              upperL = minLimit + ntolerance;
        }

        if(upperL > maxLimit){
              upperL = maxLimit;
        }

        if(lowerL > maxLimit){
```

```
                  lowerL = maxLimit;
            }

            if(upperL < nvalue){
                  nvalue = upperL;
            }

            double[] limitArray = {lowerL, nvalue, upperL};

      return limitArray;
    }


    /*
     * Returns the position of the fuzzy variable in the variable array
     */
    public int getFuzzyVarPosition(ArrayList<String> fuzzyVarStrArray, String
      attribute){
      int fuzzyVarPos = 0;

      for(int i=0; i<fuzzyVarStrArray.size(); i++){
            String[] var_value = fuzzyVarStrArray.get(i).split("=");
            if(var_value[0].equalsIgnoreCase(attribute)){
                  fuzzyVarPos = i;
                  break;
            }
      }

      return fuzzyVarPos;
    }
}
```

WEIGHT FINDER MODULE

```
import java.io.IOException;
import java.util.*;

public class WeightFinder {
      //Instantiate objects to be used in this class
      Fuzzifier iFuzzifier = new Fuzzifier();
      /*
       * Find the attribute weights given the list of non-main
       * attributes.
       */
      public ArrayList<String> findAttributeWeights(ArrayList<String>
      attributeList,
                  ArrayList<String> attributeValues) throws IOException{
            //Declare arrays to be used in this function
```

```
        ArrayList<String> allWeights = new ArrayList<String>();
        for(int i=0; i<attributeList.size(); i++){
            allWeights.add(i,Double.toString(
        iFuzzifier.getAttributeWeight(attributeList.get(i),
    attributeValues)
                        ));
        }
        return allWeights;
    }
}
```

UI QUERY BIULDER MODULE

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.*;

public class UIQueryBuilder {

      public boolean writeRulesToScript(ArrayList<ArrayList<String>>
      allRules) throws IOException{
            boolean goDecision = false;
            String mainInputStr = "";
            String otherInputStr = "";

            //Open the input string file
      File inputst=new File("src\\inputstring.clp");
            FileOutputStream inputfop=new FileOutputStream(inputst);

            //Put the attributes in their respective arrays
            ArrayList<String> mainAttributes = allRules.get(0);
            ArrayList<String> mainAttributes_qns = allRules.get(1);
            ArrayList<String> otherAttributes = allRules.get(2);
            ArrayList<String> otherAttributes_qns = allRules.get(3);

            //String to get and save the input data from the user interface
            //This is saved into the "inputstring.clp" file and read in by
      the "systemengine.clp" file
            for(int i=0; i<mainAttributes.size();i++){
                  mainInputStr += "(printout t
      "+mainAttributes_qns.get(i).replaceAll("\"", "\\\"")+" crlf)\n"
                                          +"(bind ?"+mainAttributes.get(i)+"
      (read))\n"
                                          +"(assert (inputdata (inputfield
      "+mainAttributes.get(i)+") "
                                          +"(inputvalue
      ?"+mainAttributes.get(i)+")))";
            }

            //String to get the talent
            for(int i=0; i<otherAttributes.size();i++){
                  otherInputStr += "(printout t
      "+otherAttributes_qns.get(i).replaceAll("\"", "\\\"")+" crlf)\n"
                                          +"(bind ?"+otherAttributes.get(i)+"
      (read))\n"
                                          +"(assert (inputdata (inputfield
      "+otherAttributes.get(i)+") "
                                          +"(inputvalue
      ?"+otherAttributes.get(i)+")))";
            }

            //Save the string to the file
            if(inputst.exists()){
                  //Access the holder and save the input string
```

```
                String inputstr = mainInputStr + otherInputStr;
                inputfop.write(inputstr.getBytes());
                inputfop.flush();
                inputfop.close();
          //Confirming to the user that the rules have been generated and
      saved
          //System.out.println("Code to input data generated and saved");
          goDecision = true;

        } else {
            System.out.println("ERROR: File with code to input data does
      not exist");
        }
      return goDecision;
    }
}
```

SEARCH ENGINE MODULE

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

import jess.JessException;
import jess.QueryResult;
import jess.Rete;
import jess.ValueVector;


public class SearchEngine {

      /*
       * Search the patient DB based on the passed main and other parameters.
       */
      public ArrayList<ArrayList<String>> searchPatientsDB(ArrayList<String>
      fieldList, ArrayList<String> fieldListValues) throws IOException{
            ArrayList<ArrayList<String>> listToReturn = new
      ArrayList<ArrayList<String>>();
            Converter iConverter = new Converter();

            //Variables to be used in this function
      String sLine = "";
        File searchVariables = new File("data\\patient_data.txt");
      // Open the file with the search variables
        FileInputStream searchStream = new FileInputStream(searchVariables);
        // Convert the input stream to a DataInputStream
        BufferedReader sBuffer = new BufferedReader(new
      InputStreamReader(searchStream));

        int arrayCounter = 0;
        while ((sLine = sBuffer.readLine()) != null){
            String[] sDataLine = sLine.split("<>");

            //If searching for main attributes, make sure all values match
            int noOfYes = 0;
            for(int j=0;j<fieldList.size();j++){

            if(fieldListValues.get(j).equalsIgnoreCase(sDataLine[(int)iConver
      ter.getAttributePosition(fieldList.get(j))])){
                        noOfYes++;
                  }
            }

            //if all match, add the full patient list to the list to be
      returned
            if((noOfYes == fieldList.size()) && (noOfYes != 0)){
                  ArrayList<String> patientRow = new ArrayList<String>();
                  //Get the patient row into a string ArrayList
                  for(int i=0;i<sDataLine.length;i++){
```

```
                  patientRow.add(i, sDataLine[i]);
            }

            listToReturn.add(arrayCounter, patientRow);
            arrayCounter++;
      }
  }

      return listToReturn;
}


/*
 * Function to return the data input by the user.
 */
public ArrayList<ArrayList<String>> getInputData(ArrayList<String>
mainVariables, ArrayList<String> otherVariables){
      ArrayList<String> mainInputValues = new ArrayList<String>();
      ArrayList<String> otherInputValues = new ArrayList<String>();
      //Array to return the input data
      ArrayList<ArrayList<String>> dataInputValues = new
ArrayList<ArrayList<String>>();


//Run the CLIPS file to pick data from the DB and readin the data
try {
      //Declare all required info for running the fuzzy rules and
variables.
      Rete engine = new Rete();
      engine.reset();
      // Load the overall system engine which searches
            engine.batch("src\\systemengine.clp");

            //Pick the main variables
            for(int i=0;i<mainVariables.size();i++){
                  QueryResult inputData =
engine.runQueryStar("pick_input_data", new
ValueVector().add(mainVariables.get(i)));

                  while (inputData.next()) {
                        mainInputValues.add(i,
inputData.getString("vl"));
            }
            }

            //Pick the other variables
            for(int k=0;k<otherVariables.size();k++){
                  QueryResult inputData =
engine.runQueryStar("pick_input_data", new
ValueVector().add(otherVariables.get(k)));

                  while (inputData.next()) {
                        otherInputValues.add(k,
inputData.getString("vl"));
            }
```

```
                }

            } catch (JessException ex) {
            System.err.println("Could not extract the input values. ERROR:
        "+ex);
        }

            dataInputValues.add(0, mainInputValues);
            dataInputValues.add(1, otherInputValues);

            return dataInputValues;
        }
}
```

COMPUTER MODULE

```
import java.util.*;
import java.lang.Math;


public class Computer {
      //Used to get the values of the attributes passed to this function as
      numbers
      Converter iConverter = new Converter();


      /*
       * Computes similarity between passed attributes and a patient record
       * and returns a maximum of the passed number of records
       */
      public ArrayList<String>
      computeMostSimilarPatient(ArrayList<ArrayList<String>> patientList,
                  ArrayList<String> candidateAttributes, ArrayList<String>
      candidateValues,
                  ArrayList<String> attributeWeights){

            double attributeDist = 0, oldAttributeDist = 0;
            double similarity = 0, simTotal = 0;
            ArrayList<String> mostSimilarPatient = new ArrayList<String>();
            ArrayList<Double> simArray = new ArrayList<Double>();

            System.out.println("PATIENT SHORTLIST:");

            //Get similarity between each of the patients and the candidate
            for(int i=0; i<patientList.size();i++){
                  attributeDist = getSimilarityBtwn(patientList.get(i),
      candidateAttributes, candidateValues, attributeWeights);

                  if(i != 0 && attributeDist < oldAttributeDist){
                        oldAttributeDist = attributeDist;
                        mostSimilarPatient = patientList.get(i);
                  } else if(i == 0){
                        oldAttributeDist = attributeDist;
                        mostSimilarPatient = patientList.get(i);
                  }

                  simTotal += attributeDist;
                  simArray.add(i, attributeDist);
            }

            //Display the similarity in a user friendly way
            for(int k=0; k<simArray.size(); k++){
                  similarity = 1/(1+(simArray.get(k)/simTotal));
                  System.out.println("PATIENT NO:
      "+patientList.get(k).get(0)+" SIMILARITY: "+similarity);
            }

            return mostSimilarPatient;
      }
```

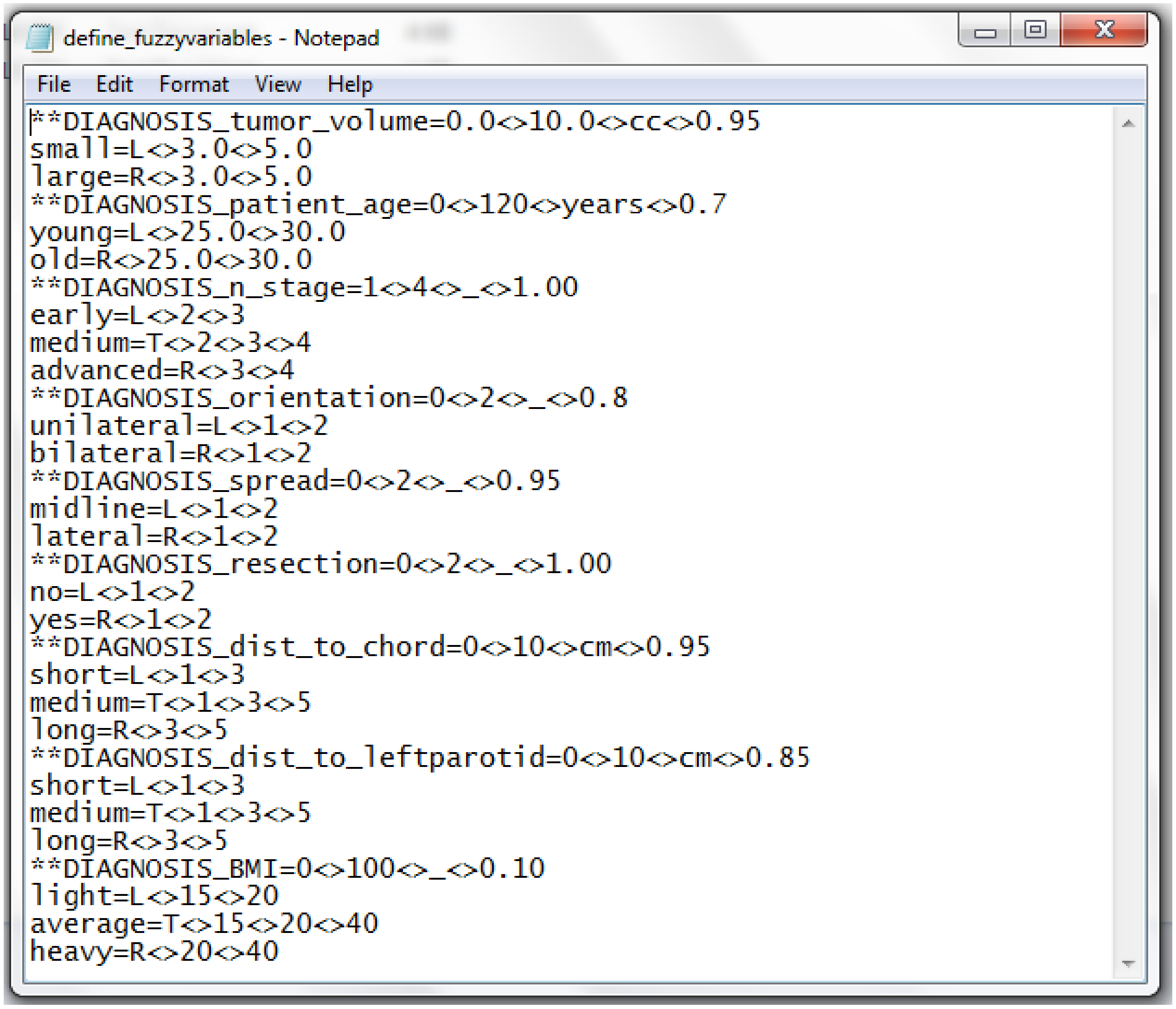
```
	/*
	 * Returns the similarity between two patient values passed to it
	 * It returns the similarity as a double value.
	 */
	public double getSimilarityBtwn(ArrayList<String> recordFromDB,
			ArrayList<String> candidateAttributes,
			ArrayList<String> candidateAttributeValues,
			ArrayList<String> weights){

		double similarity = 0, candidateValue=0, dbValue=0;
		int posToRead = 0;

		for(int i=0; i<candidateAttributes.size(); i++){
			posToRead =
	(int)iConverter.getAttributePosition(candidateAttributes.get(i));
			candidateValue =
	iConverter.getNumValue(candidateAttributes.get(i),
	candidateAttributeValues.get(i));
			dbValue =
	iConverter.getNumValue(candidateAttributes.get(i),
	recordFromDB.get(posToRead));

			similarity += Math.abs(candidateValue - dbValue) *
	Double.valueOf(weights.get(i)).doubleValue();

		}

		return similarity;
	}
}
```

PROJECT ENGINE MODULE

```
import java.io.IOException;
import java.util.*;

public class ProjectEngine {
      public static void main(String[] unused) throws IOException{

            //Gets the patient attributes from their configurations file
            AttributeExtractor iExtractor = new AttributeExtractor();
            //Writes rules to their respective script(s)
            UIQueryBuilder iUIQueryBuilder = new UIQueryBuilder();
            //Gets the weight of the passed attributes
            WeightFinder iWeightFinder = new WeightFinder();
            //Searches the database given the required list of attributes
            SearchEngine iSearchEngine = new SearchEngine();
            //Computes the most similar patients
            Computer iComputer = new Computer();

            ArrayList<ArrayList<String>> patientAttributes =
      iExtractor.getPatientAttributes();
            //Write the queries to obtain data from the input into the .CLP
      file
            //If the file is well written, then get the data entered
            if(iUIQueryBuilder.writeRulesToScript(patientAttributes)){
                  //Put the attributes in their respective arrays
                  ArrayList<String> mainAttributes =
      patientAttributes.get(0);
                  ArrayList<String> otherAttributes =
      patientAttributes.get(2);
                  //Get all data input defined by the field list (main and
      other attributes) passed
                  ArrayList<ArrayList<String>> dataFromUser =
      iSearchEngine.getInputData(mainAttributes, otherAttributes);
                  //Search the patient DB based on the passed attributes
                  ArrayList<ArrayList<String>> sub_classPatientsList =
      iSearchEngine.searchPatientsDB(mainAttributes, dataFromUser.get(0));

                  //Get the weights of the other attributes
                  ArrayList<String> attributeWeights =
      iWeightFinder.findAttributeWeights(otherAttributes,
      dataFromUser.get(1));
                  //Compute the most similar patients from the returned list
                  ArrayList<String> mostSimilar =

            iComputer.computeMostSimilarPatient(sub_classPatientsList,
      otherAttributes, dataFromUser.get(1), attributeWeights);

                  System.out.println("\n\nTHE MOST SIMILAR PATIENT IS:
      "+mostSimilar);
            } else {
                  System.err.println("THE PATIENT SEARCH COULD NOT BE
      COMPLETED.\nPLEASE TRY AGAIN OR CONTACT THE ADMINISTRATOR.");
            }
      }
}
```

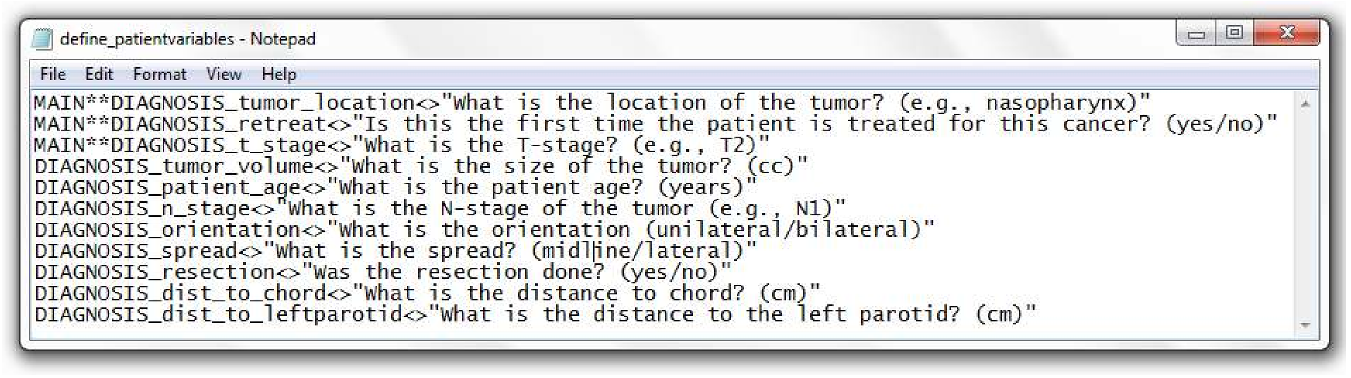
Knowledge Base Screenshots

a) The database file which defines the fuzzy variables:



```
**DIAGNOSIS_tumor_volume=0.0<>10.0<>cc<>0.95
small=L<>3.0<>5.0
large=R<>3.0<>5.0
**DIAGNOSIS_patient_age=0<>120<>years<>0.7
young=L<>25.0<>30.0
old=R<>25.0<>30.0
**DIAGNOSIS_n_stage=1<>4<>_<>1.00
early=L<>2<>3
medium=T<>2<>3<>4
advanced=R<>3<>4
**DIAGNOSIS_orientation=0<>2<>_<>0.8
unilateral=L<>1<>2
bilateral=R<>1<>2
**DIAGNOSIS_spread=0<>2<>_<>0.95
midline=L<>1<>2
lateral=R<>1<>2
**DIAGNOSIS_resection=0<>2<>_<>1.00
no=L<>1<>2
yes=R<>1<>2
**DIAGNOSIS_dist_to_chord=0<>10<>cm<>0.95
short=L<>1<>3
medium=T<>1<>3<>5
long=R<>3<>5
**DIAGNOSIS_dist_to_leftparotid=0<>10<>cm<>0.85
short=L<>1<>3
medium=T<>1<>3<>5
long=R<>3<>5
**DIAGNOSIS_BMI=0<>100<>_<>0.10
light=L<>15<>20
average=T<>15<>20<>40
heavy=R<>20<>40
```

b) The database file which defines the patient variables

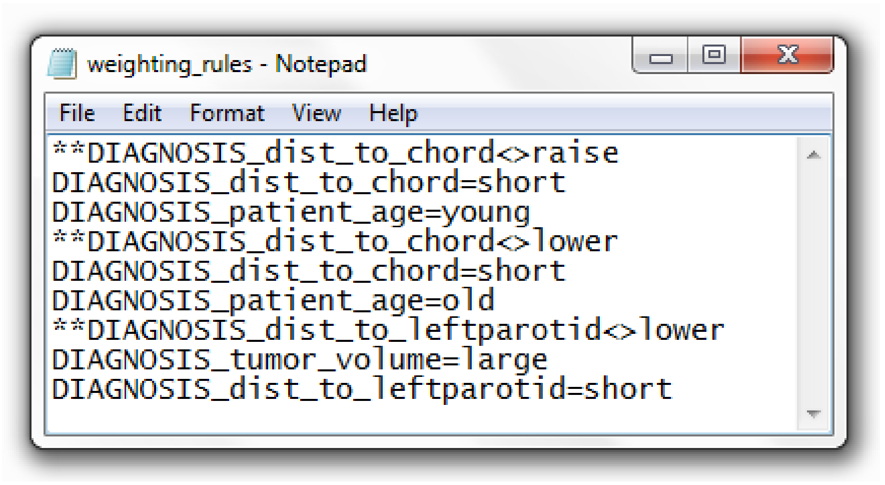


define_patientvariables - Notepad

File Edit Format View Help

```
MAIN**DIAGNOSIS_tumor_location<>"What is the location of the tumor? (e.g., nasopharynx)"
MAIN**DIAGNOSIS_retreat<>"Is this the first time the patient is treated for this cancer? (yes/no)"
MAIN**DIAGNOSIS_t_stage<>"What is the T-stage? (e.g., T2)"
DIAGNOSIS_tumor_volume<>"What is the size of the tumor? (cc)"
DIAGNOSIS_patient_age<>"What is the patient age? (years)"
DIAGNOSIS_n_stage<>"What is the N-stage of the tumor (e.g., N1)"
DIAGNOSIS_orientation<>"What is the orientation (unilateral/bilateral)"
DIAGNOSIS_spread<>"What is the spread? (midline/lateral)"
DIAGNOSIS_resection<>"Was the resection done? (yes/no)"
DIAGNOSIS_dist_to_chord<>"What is the distance to chord? (cm)"
DIAGNOSIS_dist_to_leftparotid<>"What is the distance to the left parotid? (cm)"
```

c) The database file which defines the weighting rules



weighting_rules - Notepad

File Edit Format View Help

```
**DIAGNOSIS_dist_to_chord<>raise
DIAGNOSIS_dist_to_chord=short
DIAGNOSIS_patient_age=young
**DIAGNOSIS_dist_to_chord<>lower
DIAGNOSIS_dist_to_chord=short
DIAGNOSIS_patient_age=old
**DIAGNOSIS_dist_to_leftparotid<>lower
DIAGNOSIS_tumor_volume=large
DIAGNOSIS_dist_to_leftparotid=short
```