Knowledge discovery in form of prototypical cases using advanced data mining techniques

Rathnavel Rajagopal

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2007

Program Authorized to Offer Degree:

Institute of Technology – Tacoma

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Rathnavel Rajagopal

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Isabelle Bichindaritz

_____
Ruth Rea

Date: _____

In presenting this thesis in partial fulfillment of the requirements for a master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of the thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purpose or by any means shall not be allowed without my written permission.

Signature_____

Date_____

University of Washington

Abstract

Knowledge discovery in the form of prototypical cases using advanced data mining techniques

Rathnavel Rajagopal

Chair of the Supervisory Committee:
Asst. Professor Isabelle Bichindaritz
Computing and software systems

This thesis applies advanced data mining techniques to obtain structured and abstract knowledge structures (or prototypical cases) from clinical data available in a certain medical domain. The research question addressed entails investigating if advanced knowledge structures in the form of prototypical cases can be generated from clinical data using advanced data mining techniques. Relational data mining techniques are applied on clinical data, reverse engineered from expert acquired prototypical cases. Knowledge structures in the form of clinical pathways, or prototypical cases, generated is compared to the expert acquired prototypical cases that is considered as the source for this thesis. Prototypical cases, also called clinical pathways, are the cornerstone of the knowledge acquisition process in the Memoire project, which is a framework for the development of Case-Based Reasoning (CBR) systems in biomedicine. These clinical pathways correspond to clinical diagnostic categories for the most part, some of them corresponding also to essential signs and symptoms requiring specific assessment or treatment actions for a disease. These knowledge structures are represented from ontologies such as the Unified Medical Language System (UMLS) as well as domain specific ontologies. The results obtained from this thesis will be presented and discussed. These results in the form of prototypical cases, which are generated by relational mining techniques, are validated as well.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude and appreciation to my supervisory committee members who guided me through every step of this thesis. Profound thanks to Dr.Isabelle Bichindaritz for her constant guidance and encouragement, without which this thesis would not have been possible. My gratitude to Dr.Ruth Rea for her patience and willingness to proof read the thesis and extend her valuable guidance in this domain.

1. Introduction

Artificial intelligence (AI) has been widely used in health sciences since the early 70's. In the late 80's case-based reasoning (CBR) evolved as an alternative for building AI systems in medicine. Technically most case-based reasoning methods in the medical domain follow a similar process pattern. They base their diagnostic and treatment recommendations on previous experiences represented by either real patient cases, or prototypical cases. Mémoire [2] is one such framework for the development of CBR systems in biomedicine reusing a particular kind of prototypical case called a clinical pathway. This thesis describes applying multi-relational data mining (MRDM) as a form of advanced data mining technique to obtain structured and abstract knowledge structures (prototypical cases) from clinical data available in a certain medical domain. The research question addressed entails investigating whether advanced data mining techniques such as MRDM are suitable for discovering higher level knowledge structures in the form of prototypical cases. Higher level knowledge structures discovered in the form of prototypical cases would enrich a case-based reasoning system and its reasoning ability would enhance the strength of clinical decision support systems. The algorithm described in this paper uses relational-mining techniques to evolve knowledge structures in the form of prototypical cases. An evaluation of the generated prototypical cases is done by validating them with the expert acquired prototypical cases using precision and recall factors.

2. Background information

2.1 . Case-based reasoning in bio-medicine

Case-based reasoning (CBR) is an experience-based reasoning paradigm which exploits the underlying knowledge acquired from previously experienced problem situations (referred to as cases) to solve new problems.

CBR systems are notable examples of decision support systems as they base their recommendations on the subset of the most similar experiences previously encountered. This property of CBR systems made it a choice for experimental sciences such as the natural and the life sciences [1].

An analogy of CBR in medicine is shown by a physician who examines a patient and whose reasoning is based on similarities with past examinations and diagnosis from previous patients. The physician then considers the differences between the current patient and past treated patients (or cases) and accordingly exploits their subjective experience, gained while treating similar patients, to treat the current patient. This is the property of Cognitive Reasoning [4] that is followed by CBR. CBR provides an 'analogy-based' reasoning technique that is prevalent towards addressing health-care-related decision-support problems, in particular clinical diagnostic support [6] by remembering former cases similar to the current problem and attempting to modify their solutions to fit the current case [5].

A case-based reasoning cycle developed by Aamodt [5] is shown in fig. 1 below.

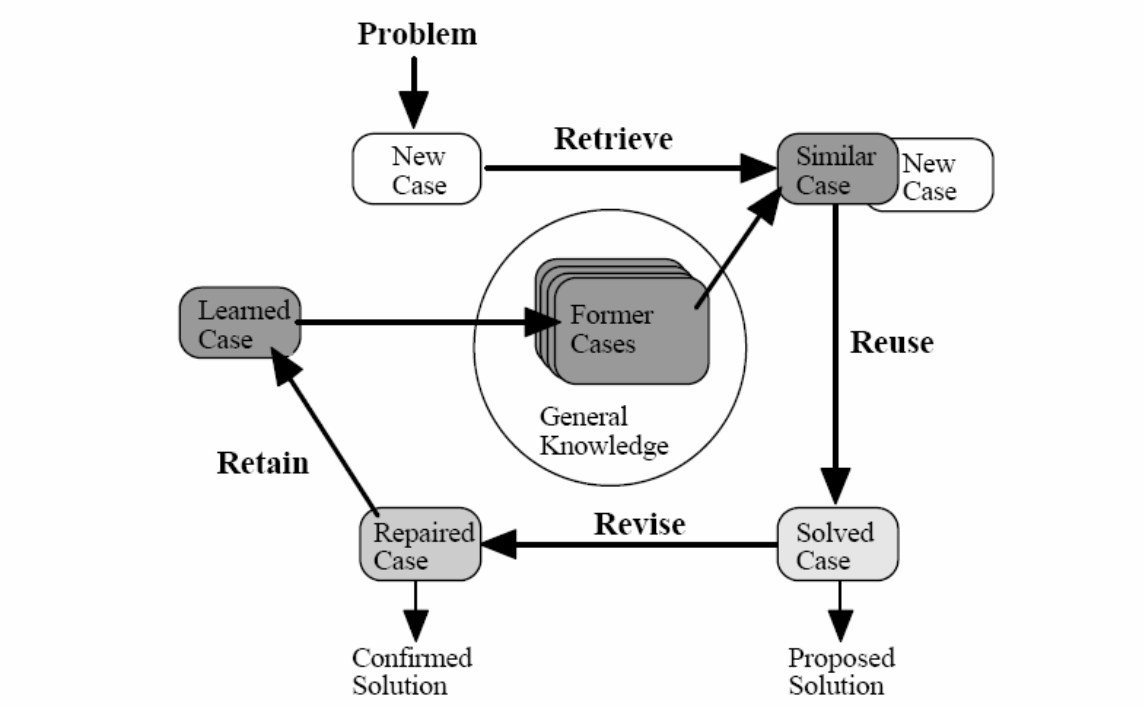

Figure 1: Case-based reasoning cycle developed by Aamodt.

Based on several readings it is being interpreted that there are early systems in bio-medicine but that these were not developed in clinical settings. The systems that are developed in clinical settings are built on disparate data models such as ALEXIA, MNAOMIA & CARE-PARTNER [1].

2.2 . Case studies in similar areas

1. CARE-PARTNER: a Computerized Knowledge-Support System for Stem-Cell Post-Transplant Long-Term Follow-Up on the World-Wide-Web [3], and
2. MNAOMIA: is a case-based reasoning system, which provides clinical assistance to clinical staff in psychiatric eating disorders for diagnosis, treatment, and research hypothesis recommendation. [8].

2.2.1. CARE-PARTNER

The CARE-PARTNER system [2] is a computerized knowledge-support system for stem-cell post-transplant long-term follow-up care on the WWW. It means that the system monitors the quality of knowledge both of its own knowledge-base and of its users.

The **aim** of the system is to function as a decision-support system (DSS) to support the evidence-based practice of the long-term follow-up clinicians and of the home-town physicians who actually extend medical care to the patients subjected to stem cell transplant.

The three fundamental characteristics of CARE-PARTNER that are accountable for its knowledge-support function are:

1. Quality of its knowledge base
2. Its availability on the WWW
3. Its capability to learn from experience.

The integration of a case-based reasoner in the reasoning framework enables the system to introspectively study its results, and to learn from its successes and failures, thus confronting the quality of the guidelines and pathways it reuses to the reality and complexity of the clinical cases.

The basic premise of the CARE-PARTNER system is evidence-based practice in medicine which emphasizes the performance of medical practice based on proven and validated practice.

Care-Partner Knowledge Base is a network of entities such as the practice guidelines, pathways and cases, where entities are nodes, connected by links, or relationships. The

links that provide ontology has been refined for this application domain. The elements of representation language comprise domain ontology, a set of individual symbols and a set of operators. The Care-Partner reasoning cycle is shown in fig.2 below [3]

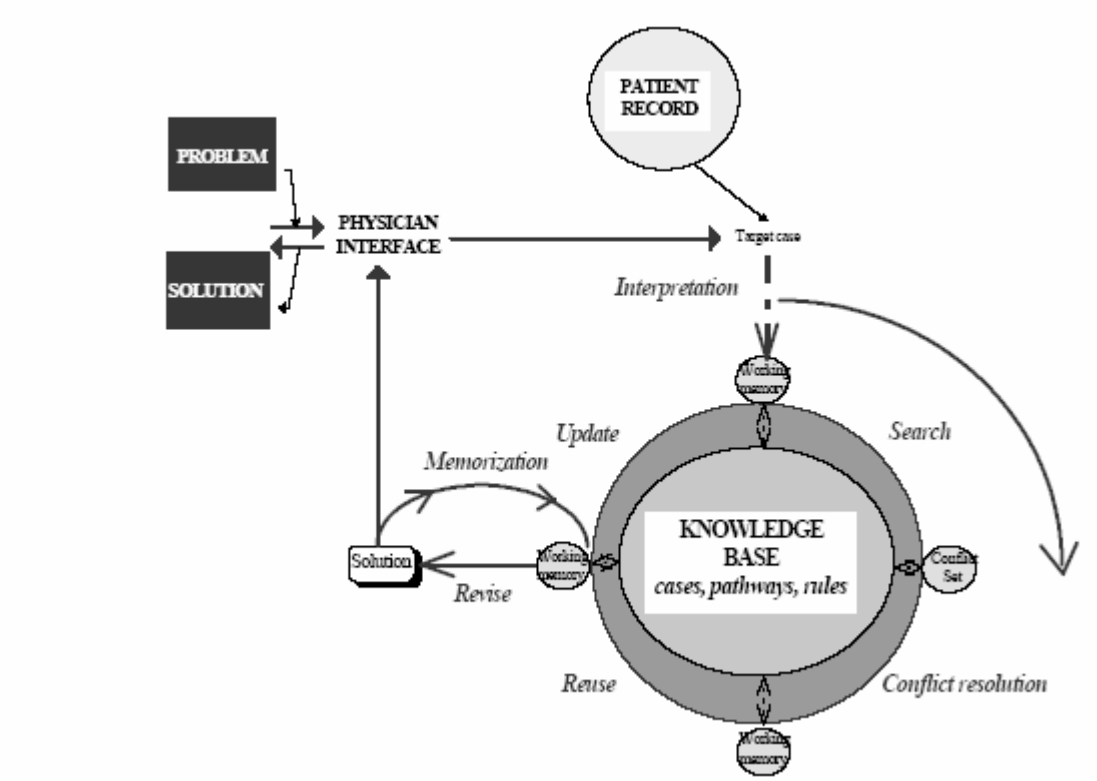

Figure 2: **Care-Partner reasoning cycle**.

Care-Partner reasoning framework can be explained based on the fig.2 above. The knowledge base is represented by the large lower circle where it contains all the previously learnt cases, pathways and rules. The Electronic Patient Record (EPR), which is represented by the smaller upper circle in fig. 2, is a notebook gathering all long-term follow-up information available about the patient. The system reasoning is represented by the series of arrows in succession starting by the problem submitted by the care provider on the top-left leading to the solution provided by the CBR system on the top-left (where it started, making it one cycle with reasoning iterations within).

The system is based on multimodal reasoning where the rule-based reasoning (RBR), case-based reasoning (CBR) and information retrieval (IR) frameworks cooperate with one another to solve the problem. The memorization step shown in fig. 2 occurs when learning from experience. Several stages of learning occur, such as case learning where a new case is created with the problem situation and its solution; positive feedback when obtained is used in validating a case; negative feedback is used to invalidate the solution and possibly add more cases to validate [8].

This system has been successful in clinical trial. The prototypical cases obtained from this system are used in this thesis as input.

2.3 . Prototypical Cases

2.3.1. Definition of a pathway

A pathway is a practice guideline expressed in a formal language that the computer system can use to reason.

2.3.2. Types of pathways

Two types of pathways are presented here, diagnosis pathways and treatment pathways.

2.3.3. Diagnosis pathways

Diagnosis pathways are diagnosis guidelines for specific diseases. There is one pathway per disease, in a list that we have determined. The language to represent a diagnosis pathway is the following:

- The list of findings that are manifestations of the disease, such as SignsAndSymptoms, Medications, Labs and/or Procedures results. Each finding, such as pimple, is described by a list of properties associated to values, such as (*color = 'red'*), an importance, ranging from *NecessaryAndSufficient* to *LowImportance*, and a level, ranging from *Absent* to *Severe.* The elements in a list are connected by connectors, such as *AND, OR, AT LEAST n, AT MOST n, EITHER, ….* The values associated to a property name are qualitative, and can take the following values: *VeryLow, Low, Normal, High, VeryHigh*. The correspondence between these qualitative values, and ranges of associated numerical values, has to be provided to the system. It will be also necessary to associate other types of qualitative values, for example for trends, for which the values will be in the range of *DecreasingVeryMuch, Decreasing, Stable, Increasing, IncreasingVeryMuch*.
- The list of the ways of assessing the diagnosis, such as Procedures or Labs. associated with an order (*1* for *first*, *2* for *second*, …) when the ways are sequential. The elements in a list are also connected by connectors.

2.3.4. Treatment pathways

Treatment pathways are treatment guidelines for specific diseases. There is one pathway per disease, and generally it is used after the diagnosis pathway has permitted to assess the disease. The language to represent a treatment pathway is the following:

- The list of PlanningActions to treat the disease, associated with an order, connected by connectors.

Treatment pathways can be associated to symptoms, and so for some of these, a list of findings will be used to describe the pathway. In the relational data mining algorithm that is discussed, the findings in the treatment pathways are also taken into consideration while generating prototypical cases.

A sample prototypical case from this document is shown in fig. 3 below.

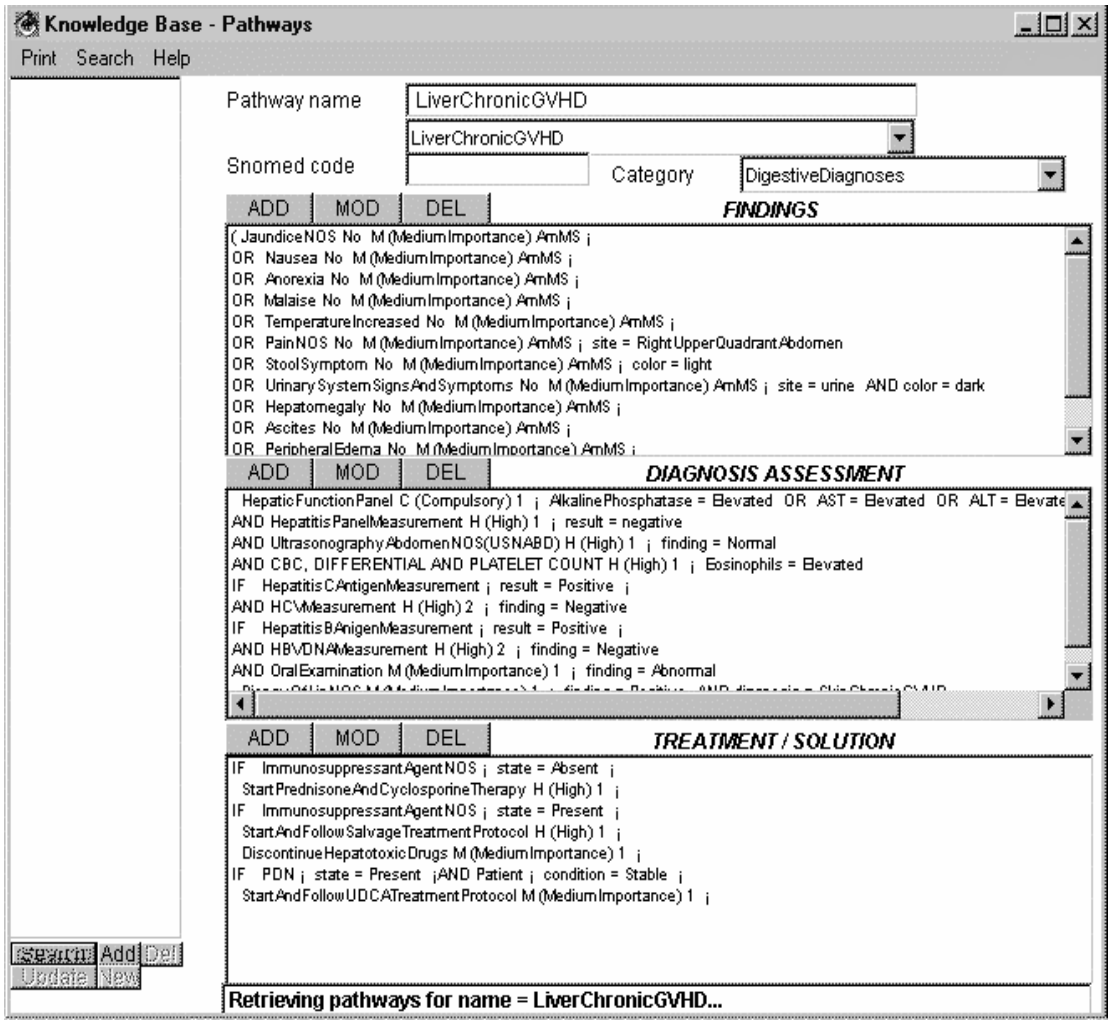

Figure 3 :A prototypical case for the disease LiverChronicGVHD

A prototypical case being studied here comprises of three parts:

1. A list of findings corresponding to signs and symptoms. They are under the heading “findings” in the document for each disease. For example, the signs and symptoms for the disease InfectiousPericarditisNOS is shown in Appendix 1.

2. A diagnosis assessment plan, which is a plan to follow for confirming (or informing) the suspected diagnosis. For example, the diagnosis assessment plan for the disease InfectiousPericarditisNOS is shown in Appendix 1.

3. A treatment / solution plan, which is a plan to follow for treating this disease or a solution when the pathway does not correspond to a disease. For example, the treatment / solution plan for the disease InfectiousPericarditisNOS is shown in Appendix 1.

2.4 . Representation paradigm of a prototypical case

2.4.1. Object representation

Each knowledge structure composing a clinical pathway is an instance of one of the classes defined in the ontology.

The diagnosis assessment part and the treatment part are seen as simplified algorithms because they use IF-then-ELSE structures, LOOP structures and SEQUENCE structures of actions in time.

To support this fact it can be seen that in the treatment plan part of the above example as

AND IF cultures.result = Negative	AskCurrentDrugHistory ConsiderDrugInducedPericarditis		3

i.e. if the pre-treatment test of cultures is negative, the treatment plan is to ask the current drug history and to consider drug induced pericarditis. This is the 3rd step in the sequence as seen in the order of treatment plans for the disease in the prototypical case in Appendix 1.

To give a diagnosis assessment or treatment plan tailored to a specific patient, instantiate the "algorithms" of diagnosis assessment and treatment plans with the patient's data.

These knowledge structures allow for sophisticated adaptation when reusing a prototypical case.

2.5 . Goal of a prototypical case

The goal of a prototypical case is to automatically learn the type of knowledge structures illustrated by the clinical pathways.

3. Problem statement

As seen from the definition of prototypical cases, they are knowledge structures which provide information about the clinical findings, diagnosis assessments and treatment plans of a disease in a clinical domain. The prototypical cases which are used as a source of input to this thesis are generated by clinical domain experts. This process could be very time consuming and not resource effective. Conventional data mining approaches, also termed propositional methods of data mining, mine patterns of data from a single relation. The data from multiple relations is preprocessed using various extraction, transformation and loading (ETL) techniques to load data into a single relation. The problem, however, is that the ETL process has a potential of causing loss of information or meaning of data [21]. Hence multi-relational data mining is considered as an approach to search for patterns from multiple relations in a relational database. In conventional data mining methods, data from various relations are processed or aggregated into a single relation. It is from this single relation that the learning algorithms learn patterns or knowledge structures. Each relation in a multi-relational database is an entity which is defined by a set of attributes [22]. There are links such as foreign keys on relations which show the relationship between the linked relations. Unlike conventional methods of data mining where data from various relations is converted into a single relation using joins and aggregations, multi-relational data mining methods use classification methods to discover patterns from the data directly from these multiple relations.

The knowledge structures or prototypical cases (as they are referred to) studied in this thesis have been derived with knowledge acquisition from experts. In this thesis

advanced data mining techniques such as multi-relational mining techniques is applied on simulated clinical patient data to investigate if advanced knowledge structures in the form of prototypical cases can be derived. The idea is to see if knowledge structures can be discovered, from the relations in a multi-relational database that have the same quality as the prototypical cases that are expert acquired.

The clinical data represented by clinical findings, diagnosis assessments and treatment plans are stored in their respective relational tables. This characteristic of the clinical data being in multiple relational tables drives the use of multi-relational mining as a relevant approach to derive prototypical cases from them. Further, the algorithm uses unsupervised learning since the clinical patients' data are "real-time" and prototypical cases are generated with no background knowledge for the algorithm to learn from.

A structured query language (SQL) based algorithm is introduced to demonstrate that knowledge structures can be discovered from multiple relations in multi-relational database. The knowledge structures obtained by applying this algorithm is evaluated for quality and compared with the existing knowledge structures obtained from conventional data mining techniques. A query based probabilistic ranking algorithm is designed to evaluate knowledge structures obtained and hence assist in evaluating their quality. The learning algorithm is based on unsupervised learning paradigm. An alternative method of multi-relational mining using supervised learning is described, but not used.

3.1 . Another multi-relational mining method to generate prototypical cases

Another method to finding prototypical cases is using Inductive Logic Programming (ILP) towards finding patterns (cases or knowledge structures) as logic programs [21].

Logic programs consist of clauses which are first-order relational rules where the conclusion is the head of the clause and the condition is the body of the clause. A clause in the form of first-order rule will look like:

patient(X, Jaundice) ^ patient(X, Nausea) ^ bilurubin(X, HighLevel) ← disease(X, ChronicLiverGVHD).

The clause reads as "If patient X has jaundice and patient X has nausea and X has high Biluribin then patient X has ChronicLiverGVHD". The atoms of the clause are patient(X,Jaundice), patient(X,Nausea), Biluribin(X, HighLevel) and disease(X, ChronicLiverGVHD). X, nausea, jaundice, high, ChronicLiverGVHD are variables. The target class or head or predicate in ILP is a relation and the arguments of a predicate are the attributes of the relation. The attributes in a relation each are associated with a domain.

Propositional patterns have been extended to their relational versions to include relational classification rules, relational regression rules, relational association rules, etc.. RDM algorithms also extend the propositional data mining algorithms such as decision-tree induction, distance-based clustering and prediction, etc. ILP is based on inductive inference which generalizes from individual instances and observations in the presence of background knowledge. The task of learning logical definitions of relations involves presenting the tuples of relations as examples. These tuples may or may not belong to the target relations. The training set comprises of training examples from which the logic program or predicate definition. The logic program corresponds to a view that defines a

target relation in terms of other relations that are given as background knowledge. This is best described by a simple ILP problem shown below in Table 1.

Table 1: Representation of an ILP training example with its binary classification and background knowledge.

Training examples	Binary Classification	Background knowledge
diagnosis(patient, jaundice)	positive	symptom(jaundice,ChronicLiverGVHD)
diagnosis(patient, anorexia)	positive	symptom(anorexia, ChronicLiverGVHD)
low(patient, bilurubin)	negative	high(biluribin, ChronicLiverGVHD)
high(patient, alka-phospate)	positive	high(alka-phosphate, ChronicLiverGVHD)

The target relation from Table 1, can be illustrated as:

diagnosis(patient, ChronicLiverGVHD) ← diagnosis(patient, jaundice), symptom(jaundice, ChronicLiverGVHD)
diagnosis(patient, ChronicLiverGVHD) ← high(patient, alkaline-phosphate), high(alka-phosphate, ChronicLiverGVHD)

The hypothesis language is a set of clauses as shown above. The illustration above depicts a data mining binary classification where one of the two classes, positive or negative, is assigned to examples.

This alternative method of generating prototypical cases is not used because this represents a supervised learning paradigm. This thesis is focused on unsupervised learning methods.

4. Generation of clinical cases

This section describes the generation of clinical cases that are used as input to the algorithm that generates the prototypical cases.

4.1 . Generation of clinical cases in multi-relational representation language

This thesis is based on discovering knowledge structures in the form of prototypical cases from relational clinical data. It is shown here how the clinical cases are generated in multi-relational representational language.

4.1.1. Input

The input data to the thesis are prototypical cases generated from the long term follow up system of the clinical pathways. These prototypical cases were generated by clinical experts. The data was obtained in a Microsoft (MS) word document. The format of the input is as shown in table 2 below. A total of 122 prototypical cases were loaded into relational tables.

Table 2: Input format of prototypical cases from which clinical patient cases are generated

Findings section of the prototypical case

Connector	Finding Name	Snomed code	(Properties, Values)	Importance	Level

Diagnosis assessment section of the prototypical case

Connector	Procedure Name	Snomed code	(Properties, Values)	Importance	Order

Treatment assessment section of the prototypical case

Condition/Connector	Planning Action Name	(Properties, Values)	Order

Each disease in the input has data in the form of the three sections shown in the table above. It is shown how the clinical patients' cases are reverse engineered from the data from the input in the format as shown in table 2.

4.1.2. Algorithm

4.1.2.1. Loading prototypical cases into relational tables

The data model to load the prototypical cases is designed to conform to a normalized relational database model. The database diagram showing the relations created to convert the prototypical cases into a relational model is as shown in fig. 4. The design of the relations was done based on the characteristics of the prototypical cases.

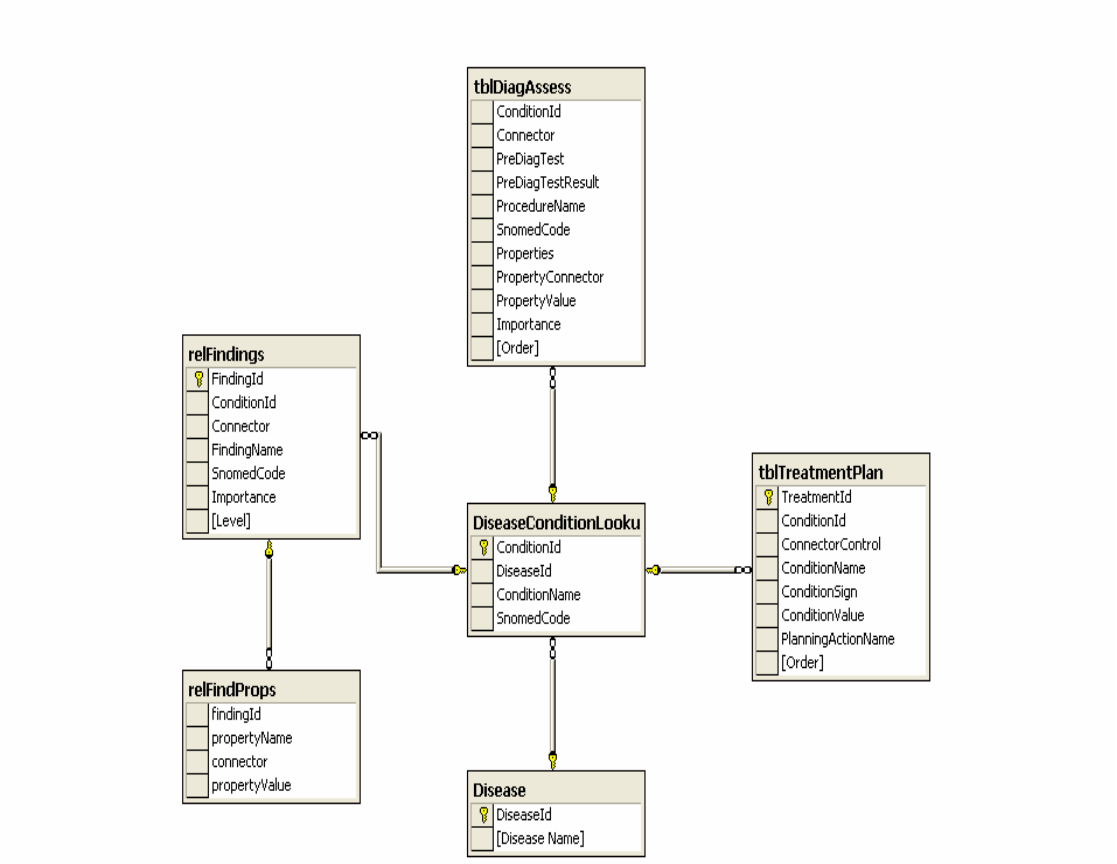

Figure 4: Database diagram of the relational data model for the prototypical cases

The details of the relations are as follows:

Disease – This relation contains the look up information for the names of the disease categories for which the prototypical cases are obtained as input. An unique identification number is assigned for each disease category and is the primary key for this relation. For example, CardiacDiagnoses is a category name indicating the root category for all diagnoses pertaining to cardiac relation conditions or diseases

Diseaseconditionlookup – This relation contains the look up information for the names of the diseases or conditions for each category. Each condition or disease is assigned a unique identification number which is its primary key and there is a foreign key relationship to the Disease relation to maintain data integrity. Each main category such as CardiacDiagnoses contains multiple conditions or diseases. For example, The diseases or conditions, InfectiousPericarditis, HypertensionNOS and NonInfectiousPericardidits belong to the category of CardiacDiagnoses.

RelFindings – This relation contains the findings or symptoms for a given disease or condition. Each finding is associated with its snomed-codes, its importance in determining the severity of the disease and its presence level. The data in this table is input from the prototypical cases obtained. Each finding is associated with a disease or condition. For example, one of the findings for InfectiousPericarditis is ChestpainNOS.

RelFindProps – This relation contains the properties and the values of those properties associated with a finding. This relation is designed as a result of normalization. Each finding could have multiple properties and as a result of this one-to-many property nature of the finding to finding-properties, this relation evolved. For example, the finding PericardiaFrictionRub for the disease InfectiousPericarditis has a property called “status” and its value is “present”.

TblDiagAssess – This relation contains the diagnosis assessments for each disease or condition. The relation is designed to conform to the atomicity property of relational data. The relation contains the pre-diagnosis test information, the pre-diagnosis test result, diagnosis procedure, snomed codes, and diagnosis properties such as the result of the diagnosis procedure or the diagnosis itself of a certain disease. The difference between this relational model and the original prototypical case is that each attribute such as the pre-diagnosis test and the properties are more atomic in the relational model, thereby making it more meaningful for data analysis.

TblTreatmentPlan – This relation contains the treatment plan information for each disease or condition. This relation is designed to satisfy the atomicity property of relational models. The structure of the data is designed to carry atomic information such as the condition that exists for a disease and a value for that condition before determining the treatment action plan. For example, if the condition pericarditis.extent has the value “small” then the treatment step to be planned is “Observe with echocardiography”.

In the process of designing the above mentioned relations, the content of data was not altered to keep its original meaning.

4.1.3. Reverse engineering of clinical patient cases

Another question that was being contemplated was, if it is possible to simulate clinical patient cases from the prototypical cases in relational form. The following sections illustrate how the clinical patient cases were generated from the input prototypical cases. To understand the structure of clinical patient cases, the differences between the prototypical cases and simulated clinical patient cases are described in more detail in the following sections.

4.1.4. Differences between the clinical pathways (prototypical cases) and simulated "real" patient clinical cases

To simulate clinical patient cases from prototypical cases, the nature of differences between the prototypical cases and simulated patient cases were studied. Prototypical cases are knowledge structures or patterns describing generically the characteristics of a disease. The findings, diagnosis assessments and treatment plans of the prototypical cases cover all aspects, optional and mandatory, for a disease. A patient case is more specific, it is an instance of a prototypical case. Some or all characteristics of a disease described in a prototypical case could be a part of the instance of a clinical patient case. The following sections describe in detail the differences between a clinical pathway or the prototypical case and the clinical patient case.

4.1.5. Findings

Table 3: Differences between the clinical pathway findings and a clinical patient case findings

Clinical Pathway (Prototypical Case)	Patient's Clinical Case
Has connectors such as AND, OR	The AND, OR connectors do not exist in a patient's clinical case. The findings are recorded as they would be seen in a real patient, hence there is no need for the AND or the OR connectors.
Has levels of severity from A (Absent) to S (Severe)	The finding levels in a patient are recorded in a real patient's case to emphasize the severity of the finding. For example: In the finding of a patient's clinical case for LiverChronicGVHD, the finding name HepatotoxicDrug is Absent. Hence, its associated with the level A.
Has Importance levels (N,C,H,M,L,S)	The importance level in a patient's clinical is assumed not relevant because the finding is there, regardless of the fact, it's important or not. For example: If the finding is Fever without any importance assigned to it, it does not mean that it is not a finding, hence the importance level is not associated with the patient's clinical finding.
Has all the findings associated with a disease	Has all the findings associated with the 'AND' connectors and a chosen few findings with 'OR' connectors. This is because a patient's clinical finding is only an instance of the prototypical case.
Properties, Values	The properties/values of the prototypical case are represented as the finding property and the finding property value respectively. For example: For the finding name PainNOS for the LiverChronicGVHD, the finding property name is "site" and the finding property value is "Right Upper Quadrant Abdomen".

4.1.6. Diagnosis assessment

Table 4: Differences between the diagnosis assessment properties of the clinical pathway and clinical patient case

Clinical Pathway (Prototypical Case)	Patient's Clinical Case
Has connectors AND	Does not have connectors AND. The diagnosis assessment steps with the AND connectors in the prototypical case is represented as the diagnosis for the patient. All the diagnosis assessments with the AND connectors in the prototypical case are considered in the patient's clinical case because they are mandatory in the diagnosis and the assessment of a disease.
Has algorithms IF <condition>.<result>	There are no IF <condition>.<result> algorithmic statements in the diagnosis assessment of a patient's clinical case. The <condition> in the IF algorithm is represented as a pre-diagnosis test in the patient's clinical diagnosis case. The <result> in the IF algorithm is represented as a pre-diagnosis test result in the patient's clinical diagnosis case. This is to represent a sequence of actions where certain tests are conducted before a certain diagnostic procedure is conducted on a patient. For example: In the diagnosis assessment of LiverChronicGVHD, before performing the HCVRNAMeasurement diagnostic procedure, the pre-diagnosis test to be conducted is to check for Hepatitis C Antigen measurement. If the pre-diagnosis test result is positive, then the diagnostic procedure is conducted for the patient.

Table 4 continued.

Have properties, values for a diagnosis assessment. Some with multiple properties.	The properties / values have been split in a patient's clinical diagnosis. Each property Value of the prototypical case is split into two distinct attributes, a) diagnostic property : which contains the property of the corresponding diagnosis assessment. For example: This could be a metric or measurement criteria. In the case of LiverChronicGVHD, the diagnostic procedure CBC has a property name "Finding". This diagnosis procedure has another property name "Result". b) diagnostic property value : which contains the property value of the diagnostic property name. Corresponding to the above example, the diagnostic property value for the name "Finding" is "Eosinophils" and for "Result" it is "elevated".
Has values of Importance N, C, H, M, L, S	The importance values are not associated with a patient's clinical diagnosis. Instead the diagnosis procedures are all considered in the formation of clinical cases.
Has order of diagnosis assessment. This is sequential information to indicate the order of diagnostic procedures to be conducted.	The order is not shown as a value in the patient's clinical case. However, the order is considered for case generation to indicate that the order of diagnosis is followed. Example: For the diagnosis associated with a patient for LiverChronicGVHD, the diagnosis step: HCVRNAMeasurement (Order # 2 in the prototypical case) is not performed before the diagnosis step : CBC (order # 1 in the prototypical case).

4.1.7. Treatment plan

Table 5: Differences between the treatment plan properties of a clinical pathway and a clinical patient's case.

Clinical Pathway (Prototypical Case)	Patient's Clinical Case
Has AND connectors	Does not have AND connectors. All the treatment plans with AND connectors in the prototypical cases exist as treatment steps administered to the patient.
Has IF<test> Then <result>=<value> algorithmic statements in the connectors	Does not have specific IF .. Then.. Result algorithmic structures. Instead the <test> of the IF condition is represented as the "pre-treatment test", which is done before administering the specific treatment to the patient. The <value> of the IF condition in the prototypical case is represented as the "pre-treatment test result" in the patient's treatment plan administered. For example: In the treatment plan for a patient with ColonChronicGVHD, the IF<test> Then <result>=<value> in the prototypical case looks as: IFColonCulture.result=PositiveForBacteria. The same is represented as the pre-treatment test done on the patient with values represented as: Pre-treatment test = ColonCulture Pre-treatment test result = PositiveForBacteria The patient's treatment step indicates that this test was done and based on the result the treatment step : AdministrationOfAntibiotics was done.

Table 5 continued

Has (properties, values) which contains properties of the treatment planning action and their corresponding anticipated values.	The (properties , values) of the prototypical case is represented as "treatment property" and "treatment property value" of the treatment step administered on the patient. The "treatment property" attribute in the patient's clinical treatment step is the metric or the property of the treatment step administered. For example: in the treatment plan of ColonChronicGVHD, the "treatment property" = "drug". The "treatment property value" attribute is the value of the metric or the property of the treatment step. For the above metric the "treatment property value" = "AntiviralAgentNOS".
Has order indicating the order of the treatment plan	Does not contain an "Order" in the treatment administered. Instead the Order is represented as the "Treatment step" of the patient. This indicates the sequence of treatment steps administered on the patient.

The pseudo code for the generation of the reverse engineered cases is in Appendix 2.

4.1.8. Output

The transact SQL (T-sql) code for the generation of the reverse engineered cases is as shown in Appendix 3.

The database diagram for the relational model of the simulated patient cases data is as shown in fig. 5 below.

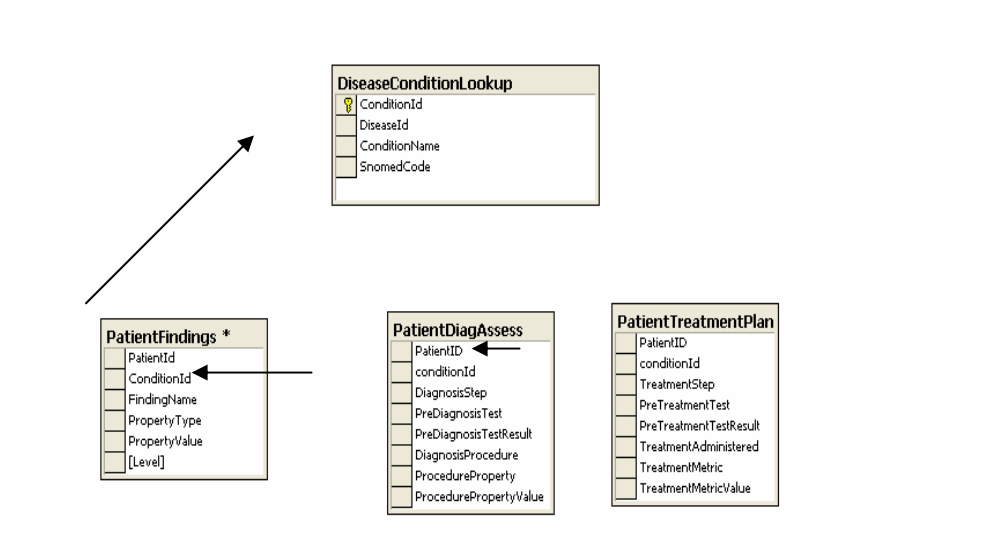

Figure 5 : Database diagram for the relational model of the simulated patient cases

The output of this algorithm to generate clinical patient cases is stored in relational tables shown in the database diagram in fig.5. An instance of a reverse engineered patient case for the disease LiverChronicGVHD is as shown in table 6 below.

Table 6: An instance of a clinical patient case for LiverChronicGVHD

Relation: Patient SignsAndSymptoms (or Findings)

PatientId	Signs	PropertyType	PropertyValue	Level
1	JaundiceNOS			
1	HepatoToxicDrug			A
1	Fever			
1	PainNOS	Site	RightUpperQuadrantAdbomen	
1	Urine	Color	Dark	

Relation: Patient Diagnosis Assessment

Patient ID	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
1			HepaticFunctionPanel	Finding	Alkaline Phosphatase Measurement
				Result	Elevated
1			HepatitisPanel	Finding	Hepatitis Panel Measurement
				Result	Negative
1			Ultrasonography AbdomenNOS	Finding	Normal
1			CBC	Finding	Eosinophils
				Result	Elevated
1	HepatitisC Antigen Measurement	Positive	HCVRNA Measurement	Finding	Negative
1	HepatitisB Antigen Measurement	Positive	HBVDNA Measurement	Finding	Negative
1			Oral Examination	Finding	Abnormal
1			BiopsyOfLipNOS	Finding	Normal
1			BiopsyOfSkin	Finding	Negative
1			SchirmerTearTest	Finding	Decreased
1			BiopsyOfLiver	Finding	Positive
				Diagnosis	LiverChronicGVHD

Table 6 continued.

Relation: Patient Treatment Plan

Patient ID	Treatment Step	Pre Treatment Test	Pre Treatment Test Result	Treatment Administered	Treatment Metric	Treatment Metric Value
1	1	Immuno Suppressant AgentNOS	Absent	Start PDNCSP Therapy		
1	2	PDN	Present	Consider UDCARx Protocol		
		Patient.condition	Stable			

4.2 . Multi-relational mining for prototypical cases

This section describes the multi-relational mining techniques applied on generated clinical patients' data to discover knowledge structures in the form of prototypical cases.

4.2.1. Input

The input to the algorithm that generates prototypical cases from clinical patients' data is the patient findings, diagnosis assessments and treatment plans in relational form. The data is obtained from the relations patientFindings, patientDiagAssess and patientTreatmentPlan. An example of the input is as shown in table 7 below.

Table 7: An instance of the input for the algorithm

Relation: Patient Signs and Symptoms (or Findings)

PatientId	Signs	PropertyType	PropertyValue	Level
2	Nausea			
2	CSP	DosageLevel	High	
2	Emesis	Synonym	Vomiting	

Relation: Patient diagnosis assessment

Patient ID	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
2			Medication Review		
2			Review of Timing of Nausea And Vomiting (N+V) In conjunction with medication		

Table 7 continued..

Relation: Patient Treatment Plan

Patient ID	**Treatment Step**	**Pre Treatment Test**	**Pre Treatment Test Result**	**Treatment Administered**	**Treatment Metric**	**Treatment Metric Value**
2	1	Ondrug	Bactrim	Hold dose	Duration	2 weeks
2	2	Symptoms Improve	Yes	Switch to Alternate Treatment (Dapsone)	Treatment	PCPProphylaxis (Prophylactic treatment) Pneumocystic Carinii
		Drug.off	Bactrim			

4.2.2. Algorithm

The algorithm for generation of the prototypical cases is called the Relational Query Language (RQL) based on simple transact SQL. The algorithm could be made more complex to simulate different propositional classification algorithms on multiple relations in the relational database. For the purpose of this thesis, it has been confined to retrieving patterns from multiple relations to generate patterns in the form of prototypical cases. The query language learns from multiple relations and builds a meta-database to learn the occurrence of each finding for each disease from the clinical patient findings. The pseudo code for the generation of prototypical cases from clinical patient data elicits a programmatic representation of how the findings, diagnosis assessment and treatment plan sections of the prototypical cases are generated.

The figure 6 below shows the pseudo code for the generation of prototypical cases from clinical patient data.

4.2.2.1. Pseudocode for generation of prototypical cases

Algorithm: Generate prototypical cases from clinical patient records

Input:
1. *RD, which is a relational database containing clinical patients' data*
2. *F, which is a relation containing clinical patients' findings/symptoms*
3. *D, which is a relation containing clinical patients' diagnosis assessments*
4. *T, which is a relation containing clinical patients' treatment assessments*

Output:
1. *The findings section of the prototypical cases*
2. *The diagnosis assessment section of the prototypical cases*
3. *The treatment plan section of the prototypical cases*

Method:

(1) Findings set Pf ← Φ;
(2) diagnosis set Pd ← Φ;
(3) treatment set Pt ← Φ;
(4) FindingsOR set Pfo ← Φ;
(5) FindingsAND set Pfa ← Φ;
(6) DiseaseCount ← 0;
(7) FindingsCount ← 0;
(8) DiagnosisANDIF set Pda ← Φ;
(9) DiagnosisProcedure set Pdp ← Φ;
(10) PreTreatmentTest set Ptt ← Φ;

(11) Start at the first patient of clinical patients' findings in F
(12) while (clinical patients' records exist in F)
(13) For each patient
(14) Set DiseaseCount ← count of the number of patients who have the same disease
(15) For each finding value of the patient's clinical record
(16) Check if finding exists in Findings set

(17) If finding does not exist in findings set Pf then

(18) Set FindingCount ← the number of occurrences of the finding in all the patients' findings who have the same disease

(19) *If FindingCount = DiseaseCount then*
(20) *FindingsAND set ← this clinical finding for the disease*
(21) *Pf ← Pfa ∪ Pfo*

(22) *else if FindingCount < DiseaseCount then*
(23) *FindingsOR set ← this clinical finding for the disease*

(24) *Pf ← Pfa ∪ Pfo*
(25) *endif*

(26) *else if finding exists in the findings set then*
(27) *go to the next finding in the patient's relation F*

(28) *Start at the first record of clinical patients' diagnosis assessments in D for this patient*
(29) *Start with diagnosis step 1 for the patient's disease*
(30) *Check the diagnosis set for existence of this diagnosis step for this disease in the diagnosis set Pd*

(31) *If diagnosis step does not exist in diagnosis set then*
(32) *If pre-diagnosis tests for the diagnosis step exists then*
(33) *Read all patients' diagnosis records for the pre-diagnosis test result for this disease and this diagnosis step*

(34) *DiagnosisANDIf set ← this diagnosis step with multiple distinct test results*

(35) *Read all patients' diagnosis records for the diagnosis procedure property and the corresponding property value for each property*
(36) *DiagnosisProcedure set ← the diagnosis procedure properties and distinct procedure property values for this diagnosis step*

(37) *Pd ← Pda ∪ Pdp for this diagnosis step for this disease*

(38) *else*

(39) *Read all patients' diagnosis records for the diagnosis procedure property and the corresponding property value for each property*

(40) *DiagnosisProcedure set ← the diagnosis procedure properties and distinct procedure property values for this diagnosis step*

(41) *Pd ← Pdp for this diagnosis step for this disease*

(42) *end if*

(43) *else*

(44) *Go to next diagnosis step of the patient's diagnosis assessment*

(45) *end if*

(46) *Start with the first treatment step based on order for this patient for this disease*

(47) *Check in the Treatment set Pt if this treatment step exists*

(48) *If the treatment step does not exist then*

(49) *If the treatment step has pre-treatment step then*

(50) *Read all patients' data for this treatment assessment step and generate distinct pre-treatment test results*

(51) *For each distinct pre-treatment test result*

(52) *PreTreatmentTest set Ptt ← this treatment step for this disease*

(53) *Treatment set Pt ← Ptt for this treatment step for this disease*

(54) *else*

(55) *Pt ← this treatment step for this disease*

```
(56)                    end if
(57)            else
(58)                    Go to next treatment step for this disease for this
                        patient
(59)            end if

(60)    endwhile
(61)    Return Pf, Pd, Pt
```

Figure 6: Pseudo code for generation of prototypical cases

4.3 . Description of the prototypical case generation method

The primary relations considered as input for the prototypical case generation from the relational database are, patientFindings, patientDiagAssess and patientTreatmentPlan

The input patientFindings data looks as shown in table 8 for the disease LiverChronicGVHD.

Table 8: Data from the clinical patients' findings relation

PatientId	Signs	PropertyType	PropertyValue	Level
1	JaundiceNOS			
1	HepatoToxicDrug			A
1	Fever			
1	Urine	Color	Dark	

PatientId	Signs	PropertyType	PropertyValue	Level
21	JaundiceNOS			
21	HepatoToxicDrug			A
21	Fever			

PatientId	Signs	PropertyType	PropertyValue	Level
15	JaundiceNOS			
15	HepatoToxicDrug			A
15	PainNOS	Site	RightUpperQuadrantAdbomen	

As seen in the table 8 above, there are 3 patient cases shown from the relation patient findings. All the three cases indicate that the patients have a common disease – LiverChronicGVHD. The algorithm to generate the findings section starts with the first patient. The disease is LiverChronicGVHD. A variable DiseaseCount is initialized with the number of clinical patients who have the disease LiverChronicGVHD. In the clinical patients' data, the number of clinical patients in the database who had LiverChronicGVHD is 150. The DiseaseCount variable will be initialized to 150. The first finding tuple is considered. In the example mentioned the first sign or finding is

JaundiceNOS. The algorithm reads all the patient finding records where the disease is LiverChronicGVHD and sets a variable findingCount to the number of occurrences of JaundiceNOS for this disease. If the findingCount is equal to the DiseaseCount then the finding is considered with the "AND" set of the findings for the disease. The representational paradigm of prototypical cases indicate that findings can be either mandatory i.e. having an "AND" connector or optional, i.e. having an "OR" connector. A factor called the findingOccurrence factor is calculated.

findingOccurrence = (findingCount / diseaseCount) * 100

If the findingOccurrence factor for the finding is 100% then the finding is added to the set of "AND" connectors P_{fa}, else, it's added to the set of "OR" connectors P_{fo}.

From the above example:

P_{fa} = LiverChronicGVHD((JaundiceNOS,M-57610, , H, ,),
(HepatotoxicDrug, , ,H,A))

The findings JaundiceNOS and HepatotoxicDrug have been added to the "AND" set of findings, P_{fa}, because these two findings occur in all the patient cases.

For the 150 clinical patients who were diagnosed with LiverChronicGVHD,

DiseaseCount for LiverChronicGVHD = 150

FindingCount for JaundiceNOS = 150

FindingOccurrence for JaundiceNOS = 150/150 * 100 = 100%

FindingOccurrence for HepatotoxicDrug = 150/150 * 100 = 100%

This is justified with the fact that if a finding for a disease occurs in all patient cases, then it is certain that the finding is prevalent in all the occurrences of the disease. That means if a patient has LiverChronicGVHD, that patient is certain to have JaundiceNOS. It is the same for HepatotoxicDrug since this occurs in all the patients who have LiverChronicGVHD too.

The finding fever occurs only in patients with patient id's 1 and 21 but not in patient with id 15.

DiseaseCount(LiverChronicGVHD) = 150

FindingCount(Fever) = 2

FindingOccurrence(Fever) = 2/150 * 100 = 1.3%

The findingOccurrence factor is an indication that fever is an optional finding in the pattern for the disease LiverChronicGVHD and hence it is classified with the "OR" findings set for this disease.

Similarly all the findings are classified in the pattern and the set of all the exclusive and mutual findings from all the patient cases for a certain disease are a part of the findings section of the prototypical case for the disease.

4.3.1. Output

The algorithm merges the two sets P_{fa} and P_{fo}, i.e. findingsAND set and findingsOR set into the findings set P_f.

The finding part of the prototypical case for the disease LiverChronicGVHD as derived by the algorithm is represented in table 9 below.

Table 9: Findings section of the prototypical case generated from relational clinical findings

Connector	findingName	SnomedCode	PropertyType	PropertyValue	Imp	Level
AND	JaundiceNOS	M-57620			H	
AND	HepatoToxicDrug				H	A
OR	Fever	F-03003			M	
OR	Urine	T-70060	Color	Dark	M	
OR	Nausea	F-52760			M	
OR	Anorexia	F-50015			M	
OR	Malaise	F-01220			M	
OR	PainNOS	F-A2600	Site	RightUpper Quadrant Abdomen	M	
OR	Stool	T-59666	Color	Light	M	
OR	Hepatomegaly	D5-81220			M	
OR	Ascites	D5-70400			M	
OR	PeripheralEdema	M-36330			M	

Another finding section of the prototypical case for LiverChronicGVHD is generated as a subset of the finding of the prototypical case shown above. However, these subsets are filtered based on their ranking and is explained in the subsequent section on ranking.

Table 10: Another prototypical case structure for the findings section

Connector	findingName	SnomedCode	PropertyType	PropertyValue	Imp	Level
AND	JaundiceNOS	M-57620			H	
AND	HepatoToxicDrug				H	A
OR	Fever	F-03003			M	
OR	Urine	T-70060	Color	Dark	M	

A subset of the clinical patient diagnosis assessment input for the generation of the diagnosis assessment section of the prototypical case is as shown in table 11 below.

Table 11: Input clinical patients' diagnosis assessment data subset

Patient ID	Diagnosis Step	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
1	1			Hepatic FunctionPanel	Finding	Alkaline Phosphasphatase Measurement (ALKP)
1	2			Hepatic FunctionPanel	Result	Elevated
1	3			Hepatic FunctionPanel	Finding	AST Measurement (AST)
1	4			Hepatic FunctionPanel	Result	Elevated
1	5			Hepatic FunctionPanel	Finding	LDH Measurement (LDH)
1	6			Hepatic FunctionPanel	Result	Elevated
1	7			HepatitisPanel	Finding	Hepatitis Panel Measurement
1	8			HepatitisPanel	Result	Negative
1	9			Ultrasonography AbdomenNOS (USNABD)	Finding	Normal
1	10			CBC	Finding	Eosinophils
1	11			CBC	Result	Elevated
1	12	HepatitisC Antigen Measurement	Positive	HCVRNA Measurement	Finding	Negative
1	13	Hepatitis B Antigen Measurement	Negative			
1	14			Oral Examination	Finding	Abnormal

Table 11 continued.

1	15			BiopsyOfLipNOS	Finding	Negative
1	16			BiopsyOfSkin	Finding	Negative
1	17			SchirmerTearTest	Finding	Decerased
1	18			BiopsyOfLiver	Finding	Positive
1	19			BiopsyOfLiver	Diagnosis	LiverChronic GCHD
1	20			RequestGI consult		

For the patient with the patient id 21, the clinical data for diagnosis assessments is as shown below in the table 12 below.

Table 12: Another subset of input clinical diagnosis data

Patient ID	**Diagnosis Step**	**Pre Diagnosis Test**	**Pre Diagnosis Test Result**	**Diagnosis Procedure**	**Procedure Property**	**Procedure Property Value**
21	1			HepaticFunction Panel	Finding	Alkaline Phosphasphatase Measurement (ALKP)
21	2			HepaticFunction Panel	Result	Elevated
21	3			HepaticFunction Panel	Finding	AST Measurement (AST)
21	4			HepaticFunction Panel	Result	Elevated
21	5			HepaticFunction Panel	Finding	LDH Measurement (LDH)
21	6			HepaticFunction Panel	Result	Elevated
21	7			HepatitisPanel	Finding	Hepatitis Panel Measurement
21	8			HepatitisPanel	Result	Negative
21	9			Ultra sonography AbdomenNOS (USNABD)	Finding	Normal
21	10			CBC	Finding	Eosinophils
21	11			CBC	Result	Elevated

Table 12 continued.

21	12	HepatitisC Antigen Measure-ment	Negative			
21	13	Hepatitis B Antigen Measure-ment	Positive	HBVDNA Measurement	Finding	Negative
21	14			Oral Examination	Finding	Abnormal
21	15			BiopsyOfLipNOS	Finding	Negative
21	16			BiopsyOfSkin	Finding	Negative
21	17			SchirmerTearTest	Finding	Decerased
21	18			BiopsyOfLiver	Finding	Positive
21	19			BiopsyOfLiver	Diagnosis	LiverChronic GVHD
21	20			RequestGIConsult		

Two instances of the diagnosis assessments for patients with LiverChronicGVHD are shown in the tables above. These instances are as they exist in the relational clinical patients' diagnosis assessments relation, i.e. patientDiagAssess. The algorithm to generate the diagnosis assessment section of the prototypical case reads the diagnosis step # 1 of patient id 1. In this case it is HepaticFunctionPanel as shown in table 13 below.

Table 13: A findings step in the clinical patients' diagnosis

Patient ID	**Diagnosis Step**	**Pre Diagnosis Test**	**Pre Diagnosis Test Result**	**Diagnosis Procedure**	**Procedure Property**	**Procedure Property Value**
1	1			HepaticFunction Panel	Finding	Alkaline Phosphasphatase Measurement (ALKP)

The algorithm checks the diagnosis set P_d for this diagnosis step for the disease LiverChronicGVHD. If this diagnosis step for this disease does not exist in the diagnosis

set, the algorithm checks to see if there are any pre diagnosis tests for this diagnosis step. There isn't any in this step. This step is added to the diagnosis procedure set P_{dp}. The algorithm then scans thru all the clinical patient diagnosis records for this diagnosis step for this disease. If there are any procedure properties and values found that are different from the one for the patient id 1, it is added to the diagnosis procedure set P_{dp}. The structure of the P_{dp} set is

P_{dp} = Disease(Connector, DiagnosisStep, PrediagnosisTest, PrediagnosisTestResult, DiagnosisProcedure, ProcedureProperty, ProcedurePropertyValue, Importance, Order)

For the diagnosis step 1 for patient id 1, the set P_{dp} will be

P_{dp} = LiverChronicGVHD(AND,1, , ,HepaticFunctionPanel, Finding, Alkaline Phosphasphatase Measurement (ALKP)

The next step in clinical diagnosis for the patient is as described below.

Table 14: Another patient's diagnosis assessment clinical data

Patient ID	Diagnosis Step	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
1	2			HepaticFunctionPanel	Result	Elevated

The same process is repeated as it did for diagnosis step 1 and the diagnosis is added to the set P_{dp}. If during the scan of other patients' clinical diagnosis the procedure property value for this step was "Not Elevated", for example, then this would be another diagnosis step recorded in the P_{dp} for the same diagnosis procedure.

The algorithm encounters a diagnosis step with a pre-diagnosis step as shown below in table 15 below.

Table 15: Clinical diagnosis data with pre-diagnosis test

Patient ID	Diagnosis Step	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
1	12	HepatitisC Antigen Measurement	Positive	HCVRNA Measurement	Finding	Negative

Since there is pre-diagnosis test for this step, this diagnosis step is added to the diagnosisAndIF set P_{da}, i.e. if this step for this disease does not already exist in the set. The reason behind adding this step to the ANDIF set is to generate a prototypical case structure that has an algorithmic structure with AND-IF conditional constructs. This adds meaning to the prototypical case in conveying a meaning that "IF" a certain test is done and a certain result is obtained, then a certain diagnostic procedure is to be performed. The structure of the set P_{da} is

P_{da} = Disease(Connector, DiagnosisStep, PrediagnosisTest, PrediagnosisTestResult, DiagnosisProcedure, ProcedureProperty, ProcedurePropertyValue, Importance, Order)

The algorithm scans thru all the patient diagnosis records for this diagnosis step for this disease and adds the variation of the diagnosis procedure to the step. For example: when the algorithm reads this diagnosis step for patient id 21, it finds the pre-diagnosis test result for this pre-diagnosis test step is negative. The diagnostic procedure HDVRNA measurement was not conducted on patient with patient id 21, but the same diagnostic procedure was conducted on patient with patient id 1 because the patient tested positive for the pre-diagnosis step.

The diagnosis steps 12 and 13 for this disease for patient id 21 are shown in the table 16 below.

Table 16: The diagnosis steps with varied pre-diagnosis test results

Patient ID	Diagnosis Step	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
21	12	HepatitisC Antigen Measurement	Negative			
21	13	Hepatitis B Antigen Measurement	Positive	HBVDNA Measurement	Finding	Negative

If there is no diagnosis procedure or procedure properties associated with a pre-diagnosis test the algorithm discards this instance of the diagnosis step. The diagnosis step where a diagnosis procedure is associated with a pre-diagnosis test is considered to be added to the P_{da} set.

The set P_d which contains the prototypical case structure of this diagnosis step is a union of the sets P_{dp} and P_{da}

i.e $P_d \leftarrow P_{dp} \cup P_{da}$

For the disease LiverChronicGVHD, the diagnosis assessment part of the prototypical case derived is as shown in table 17 below.

Table 17: Diagnosis assessment section of the prototypical case generated by the algorithm.

Connector	Diagnosis Step	Pre Diag-nosis Test	Pre Diag-nosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value	Imp
AND	1			Hepatic Function Panel	Finding	Alkaline Phosphasphatase Measurement (ALKP)	C
AND	2			Hepatic FunctionPanel	Result	Elevated	
AND	3			Hepatic FunctionPanel	Finding	AST Measurement (AST)	C
AND	4			Hepatic FunctionPanel	Result	Elevated	
AND	5			Hepatic FunctionPanel	Finding	LDH Measurement (LDH)	C
AND	6			Hepatic FunctionPanel	Result	Elevated	
AND	7			HepatitisPanel	Finding	Hepatitis Panel Measurement	C
AND	8			HepatitisPanel	Result	Negative	H
AND	9			Ultrasonography AbdomenNOS (USNABD)	Finding	Normal	H
AND	10			CBC	Finding	Eosinophils	H
AND	11			CBC	Result	Elevated	
AND IF	12	HepatitisC Antigen Measurement	Positive	HCVRNA Measurement	Finding	Negative	H
AND IF	13	Hepatitis B Antigen Measurement	Positive	HBVDNA Measurement	Finding	Negative	H

Table 17 continued.

AND IF	13	Hepatitis B Antigen Measurement	Positive	HBVDNA Measurement	Finding	Negative	H
AND	14			Oral Examination	Finding	Abnormal	H
AND	15			BiopsyOfLipNOS	Finding	Negative	H
AND	16			BiopsyOfSkin	Finding	Negative	H
AND	17			SchirmerTearTest	Finding	Decerased	H
AND	18			BiopsyOfLiver	Finding	Positive	N
AND	19			BiopsyOfLiver	Diagnosis	LiverChronic GVHD	
AND	20			RequestGIConsult			

A subset of the clinical patients' treatment plan data which is the input for the generation of the treatment plan section of the prototypical cases is as shown in the table 18 below.

Table 18: Clinical patients' treatment plan - data subset

Patient ID	**Treatment Step**	**Pre Treatment Test**	**Pre Treatment Test Result**	**Treatment Administered**	**Treatment Metric**	**Treatment Metric Value**
1	1	Immuno suppressant Agent NOS	Absent	Start PDNCSP Therapy		
1	2	PDN	Present	Consider UDCARx Protocol		

Table 18 continued.

Patient ID	Treatment Step	Pre Treatment Test	Pre Treatment Test Result	Treatment Administered	Treatment Metric	Treatment Metric Value
21	1	Immuno suppressant Agent NOS	Present	Start Salvage Rx Protocol Limit Hepatotoxic Drugs		
21	2	PDN	Present	Consider UDCARx Protocol		

The algorithm to generate the treatment section of the prototypical cases from clinical patients' treatment plan relation, i.e. patientTreatmentPlan, reads the first treatment step of the patient diagnosed with LiverChronicGVHD.

As seen in the table above, the patient with id # 1 has been administered a pre-treatment test for the presence of ImmunoSuppressantAgent. The pre-treatment test result is absent and hence the treatment administered is PDNCSP therapy. The algorithm checks in the treatment set P_t for the existence of this treatment step for this disease. If this treatment step does not exist in P_t, the algorithm adds this step to the preTreatmentTest set P_{tt}. The structure of P_{tt} is

> PreTreatmentTest set P_{tt} = Disease(Connector, TreatmentOrder, PreTreatmentTest, PreTreatmentTestResult, TreatmentAdministered, TreatmentMetric, TreatmentMetricValue)

At this stage the P_{tt} will contain

> P_{tt} = LiverChronicGVHD(IF, 1, ImmunosuppressantAgentNOS, Absent, Start PDNCS Therapy, , ,)

The algorithm scans the treatment step # 1 of all the patients with this disease and compares the pre-treatment test result with the current patient's pre-treatment test result. For patient id # 21, the pre-treatment test result shows that ImmunoSuppressantAgent is present. As a result the treatments administered are: Salvage Rx protocol and limit hepatotoxic drugs. The treatment administered for the same step #1 for patient id # 21 is different than that for patient id # 1 because of the different pre-treatment test results between them.

The algorithm adds the treatment step with the variation in pre-treatment test result and a different treatment plan to the set P_{tt}. The treatmentOrder is maintained the same, i.e. # 1, in this case. This is because the treatment step of the clinical patient data does not change since there can be only one outcome for the pre-treatment test. In this case, either ImmunosuppressantAgentNOS present or absent. The P_{tt} will now contain

P_{tt} = LiverChronicGVHD((IF, 1, ImmunosuppressantAgentNOS, Absent, Start PDNCS Therapy, , ,),
(IF, 1, ImmunosuppressantAgentNOS, Present, Salvage Rx Protocol and Limit Hepatotoxic Drugs, , ,))

The next treatment step that the algorithm comes across for patient id #1 has the pre-treatment step as well. The same process as described above is followed and if the step does not exist in P_t, it is added to P_{tt}. For the disease, LiverChronicGVHD, the set P_{tt} will now contain

P_{tt} = LiverChronicGVHD((IF, 1, ImmunosuppressantAgentNOS, Absent, Start PDNCS Therapy, , ,),
(IF, 1, ImmunosuppressantAgentNOS, Present, Salvage Rx Protocol and Limit Hepatotoxic Drugs, , ,),
(IF, 1, PDN, Present, Consider UDCARx Protocol, , ,))

When no more treatment steps for this disease is encountered the set P_{tt} is merged with the treatment set P_t.

i.e $P_t \leftarrow P_t \cup P_{tt}$

The treatment section of the prototypical case will be as shown in the table 19 below.

Table 19: Treatment plan section of the prototypical case for LiverChronicGVHD

Connector	**Treatment Order**	**Pre Treatment Test**	**Pre Treatment Test Result**	**Treatment Administered**	**Treatment Metric**	**Treatment Metric Value**
IF	1	Immuno suppressant Agent NOS	Absent	Start PDNCSP Therapy		
IF	1	Immuno suppressant Agent NOS	Present	Salvage Rx Protocol Limit Hepatotoxic Drugs		
IF	2	PDN	Present	Consider UDCARx Protocol		

Finally, the algorithm will return the three sets, P_f, P_d and P_t for the findings, diagnosis assessments and treatment plans respectively for each disease in the clinical patients' data. Each disease has the combination of all these three sets to form a prototypical case for that disease.

5. Validation of the prototypical cases generated

The prototypical cases generated are validated by using a ranking approach where each finding, diagnosis assessment and treatment step is compared with the original prototypical cases generated by clinical domain experts.

5.1 . Ranking algorithm for the prototypical cases

The ranking algorithm uses precision and recall factors to rank each of the prototypical cases generated from the clinical patients' data individually. Precision and recall is calculated on each of the prototypical case sections, i.e. findings, diagnosis assessment and treatment sections, for all the diseases.

The prototypical case generated from the relational clinical patients' data is referred to as P_{Rel}. The prototypical cases created by clinical domain experts are referred to as P_{Dom}.

The attributes of the findings section of P_{Rel} are:

P_{Rel}(disease) = findings(Connector, findingName, SnomedCode, PropertyType, PropertyValue, Importance, Level)

The attributes of the findings section of P_{Dom} are:

P_{Dom}(disease) = (Connector, findingName, SnomedCode,(Properties,Values), Importance, Level)

The prototypical cases are ranked based on their quality for each disease. The quality of the generated prototypical is measured by comparing each attribute for each finding for each disease.

For the disease LiverChronicGVHD the quality of the findings section will be ranked by starting with the first finding.

P_{Rel}(LiverChronicGVHD)= findings((AND, JaundiceNOS, M-57610, , , H, ,))

P_{Dom}(LiverChronicGVHD)=findings((, , JaundiceNOS, M-57610, , H, ,))

The attribute of the findings of P_{Rel} and P_{Dom} are compared for the finding as shown in the table 20 below.

Table 20: Attribute comparison for quality scores computation for JaundiceNOS

Attribute Name	**P_{Rel}**	**P_{Dom}**	**Quality Score**
Connector	AND	AND	100
finding Name	JaundiceNOS	JaundiceNOS	100
SnomedCode	M-57610	M-57610	100
Property Type			100
Property Value			100
Importance	H	H	100
Level			100
Overall quality score for this finding			**700/700 = 100**

The attribute connector is AND for P_{Rel} and null for P_{Dom} for the finding name Jaundice for the disease as shown in the table above. Yet, the quality score is assigned as 50 instead of 0. This is because with the presence of an AND connector, it increases the chance of a false positive rather than an false negative. If the connector was an OR where there should have been an AND, it would increase the chance of a false negative which is

more undesirable than false positives. For the findings the scores are assigned as shown in the table 21 below.

Table 21: Quality score reference table for prototypical case findings

Attribute Name	**P_{Rel}**	**P_{Dom}**	**Quality Score**
Connector	AND	AND	100
Connector	AND	OR	50
Connector	OR	AND	0
Connector	OR	OR	100
Finding Name	match	original	100
Finding Name	mismatch	original	0
Snomed Code	match	original	100
Snomed Code	mismatch	original	0
Properties,Values	Property Type match	Property type value before the comparator sign	100
Properties,Values	Property Value match	Property type value after the comparator sign	100
Properties,Values	Property type or value mismatch	Compared with Properties type value before the sign and after the comparator sign respectively	0

The quality score for another disease HypertensionNOS is shown in table 22 below.

Table 22: Attributes' comparison for HypertensionNOS

Attribute Name	PRel	PDom	Quality Score
Connector	AND	AND	100
finding Name	IncreasedBloodPressure	IncreasedBloodPressure	100
SnomedCode	F-31003	F-31003	100
Property Type	Finding	Finding	100
Property Value	ElevatedDiastolic BloodPressure	ElevatedDiastolic BloodPressure OR ElevatedSystolic BloodPressure	0
Importance	H	H	100
Level			100
Overall quality score for this finding			**600/700 = 85.71**

The quality score for the diagnosis assessment section of the prototypical case is computed similar to the findings.

The attributes of the diagnosis assessment section of P_{Rel} are:

P_{Rel}(disease) = DiagnosisAssessment(Connector, preDiagnosisTest, preDiagnosis Test Result, ProcedureName, SnomedCode, Properties, PropertyConnector, PropertyValue, Importance, DiagnosisStep)

The attributes of the diagnosis assessment section of P_{Dom} are:

P_{Dom}(disease) = DiagnosisAssessment(Connector, ProcedureName, SnomedCode, (Properties,Values), Importance, Order)

The quality score table based on which the quality of diagnosis assessments are scored is as shown in table 23 below.

Table 23: Quality score reference table for diagnosis assessment section of prototypical cases

Attribute Name	PRel	PDom	Quality Score
Connector	AND If	AND If	100
PreDiagnosisTest	Value	Original test value after the AND IF connector and before the equality sign.	100
PreDiagnosis Test Result	Value	Original test result value after the AND If connector and after the equality sign.	100
Connector	AND If	Null	0
PreDiagnosisTest	Value	Null	0
ProcedureName	Match	Original	100
ProcedureName	Value	Mismatch	0
SnomedCode	Match	Original	100
SnomedCode	Value	Mismatch	0
Properties	Match	Value prior to the operator	100
Properties	Value	Mismatch value prior to the operator	0
PropertyConnector	Match	Value of the operator	100
PropertyConnector	Value	Mismatch value of the operator	0
PropertyValue	Match	Value after the operator	100
PropertyValue	Value	Mismatch value after the operator	0
Importance	Match value	Original value	100
Order	Diagnosis Step	Order in sequence	100
Order	Diagnosis Step	Order not in sequence	0

The quality score for the treatment plan section is computed by comparing the attributes of the treatment assessment sections of the two sets P_{Rel} and P_{Dom}. The treatment plan attributes of these two sets are as:

The attributes of the treatment plan section of P_{Rel} are:

P_{Rel}(disease) = TreatmentPlan(Connector,TreatmentStep, preTreatmentTest, preTreatmentTestResult, TreatmentActionName, TreatmentProperty, TreatmentPropertyValue)

The attributes of the diagnosis assessment section of P_{Dom} are:

P_{Dom}(disease) = TreatmentPlan(Condition/Connector, PlanningActionName, (Properties,Values), Order)

The attributes are compared as described in table 24 below.

Table 24: Attribute comparison reference for treatment plan section of prototypical cases.

Treatment plan attributes of P_{Rel}	Treatment plan attributes of P_{Dom}
Connector	The AND and the AND IF part of the Condition/Connector
TreatmentStep – This is incremental in a sequence.	Order. The difference is that the Order is not incremental in a sequence as it is in the TreatmentStep of PRel. If the order value is greater than the treatmentStep value then it is a mismatch.
PreTreatmentTest	The pre-treatment test value of the Condition/Connector. Example: For CardiacDiagnosis the pre-treatment test value for a treatment step is cultures.result
PreTreatmentTestResult	The pre-treatment test result value of the Condition/Connector. Example: The test result value is Positive.
TreatmentActionName	PlanningActionName
TreatmentProperty	The value prior to the equality operator in the (Properties,Values). For example: in Site=Blood, this would be site.
TreatmentPropertyValue	The value after the equality operator in the (Properties,Values). For example: in Site=Blood, this would be Blood.

The quality score reference table for the treatment plan sections of the prototypical cases generated is as shown in table 25 below. These scores are used as a metric in the computation of precision and recall for the treatment sections of the prototypical cases generated.

Table 25: Quality score reference for treatment plan section.

Attribute Name	P_{Rel}	P_{Dom}	Quality Score
Connector	AND	AND	100
Connector	AND If	AND If	100
Connector	Value	Mismatch	0
Treatment Step	Value	Equals or greater than Order	100
Treatment Step	Value	Not equal or less than Order	0
Pre-Treatment Test	Value	Equals the pre-treatment test value in Condition/Connector	100
Pre-Treatment Test	Value	Mismatch	0
Pre-Treatment Test Result	Value	Equals the pre-treatment test result in Condition/Connector	100
Pre-Treatment Test Result	Value	Mismatch	0
Treatment Action Name	Value	Equals the Planning Action Name value	100
Treatment action name	Value	Mismatch	0
Treatment Property	Value	Equals the Properties value of the (Properties, Values)	100
Treatment Property	Value	Mismatch	0
Treatment Property Value	Value	Equals the “Values” value of the (Properties, Values)	100
Treatment Property Value	Value	Mismatch	0

5.2 . Precision and Recall

In the validation of the prototypical cases generated (P_{Rel}) the attributes of each case section, i.e. findings, diagnosis assessments and treatment plans are compared with the attributes of each respective case section of the prototypical cases generated by domain experts (P_{Dom}).

For each disease in P_{Rel}, the findings are compared with the findings in P_{Dom} and their quality scores are computed. The methodology of computing the quality scores, precision and recall is described as follows.

Let

Q_f = the score for each finding f for each disease d in P_{Rel};

F_d = the number of relevant findings, i.e. the number of findings in the prototypical case for the disease d in the set P_{Dom}, the prototypical cases generated by clinical domain experts;

F_r = the number of findings for each disease d generated by this algorithm;

F_g = the number of good quality findings f for the disease d where

$\exists f_i$, for i in 1 to n where n is the number of findings f for each disease d in P_{Dom}

and fi(PRel) = fi(PDom)

and Qf(fi(PRel)) = 100

i.e. findings with a score of 100 for each disease;

$$F_g = \sum_{i=1}^{n} fi \text{, where } Qf(fi(PRel)) = 100$$

F_b be the number of findings retrieved, but have a score of less than 100 or not a relevant finding or good finding, i.e.

$$F_b = \sum_{i=1}^{n} fi \text{, where } Qf(fi(PRel)) < 100$$

For each disease d

$$\text{Precision P} = \frac{number\ of\ \ findings\ \ for\ d\ that\ are\ good\ \ Fg}{number\ of\ \ findings\ returned\ by\ this\ a\lg orithm\ \ for\ disease\ \ d\ ,\ Fr}$$

$$\text{Recall R} = \frac{number\ of\ \ findings\ \ for\ d\ that\ are\ good\ \ Fg}{number\ of\ \ findings\ relevant\ \ for\ disease\ \ d\ \ returned\ by\ PDom\ ,\ Fd}$$

For the findings of the prototypical case of the disease LiverChronicGVHD:

d = LiverChronicGVHD

F_d = number or findings or relevant findings for d from PDom = 12

F_r = number of findings for d from PRel = 12

F_g = number of findings with a quality score of 100 for d = 12

F_b = number of findings with a quality score less than 100 for d = 0

Precision P(LiverChronicGVHD) at 12 (value of F_r) = 12/12 = 100%

Recall R(LiverChronicGVHD) at 12 = 12/12 = 100%.

Precision and recall are calculated for an instance of the prototypical case findings for disease HyperTensionNOS is shown as follows.

F_d (HypertensionNOS) = 6

F_r (HypertensionNOS) = 6

F_g (HypertensionNOS) = 5

F_b (HypertensionNOS) = 1

Precision P(HypertensionNOS) = F_g / F_r = 5/6 = 83.33%

Recall R(HypertensionNOS) = F_g / F_d = 5/6 = 83.33%

For the finding sections of the prototypical cases for all the diseases

Findings Precision(P_{Rel}) = $\sum_{k=1}^{m} Pk/k$, where

k is the number of diseases, d, for which the prototypical cases are generated from 1 to m. In this case m = 122.

Similarly, findings Recall(R_{Rel}) = $\sum_{k=1}^{m} Rk/k \sum_{k=1}$, where

k is the number of diseases, d, for which the prototypical cases are generated from 1 to m. In this case m = 122.

Findings Precision(PF_{Rel}) = 11808.18 / 122 = 96.78 %

Findings Recall(RF_{Rel}) = 10732.26 / 122 = 87.96 %

The precision and recall calculations for diagnosis assessment section of the prototypical cases are computed as:

$$\text{Precision P} = \frac{number\ of\ diagnosis\ for\ d\ that\ are\ good\ Dg}{number\ of\ diagnosis\ returned\ by\ this\ a\lg orithm\ for\ disease\ \ d\ ,\ Dr}$$

$$\text{Recall R} = \frac{number\ of\ diagnosis\ for\ d\ that\ are\ good\ Dg}{number\ of\ diagnosis\ relevant\ for\ disease\ \ d\ returned\ by\ PDom\ ,\ Dd}$$

where

Q_{da} = the score for each diagnosis assessment da for each disease d in P_{Rel};

D_d = the number of relevant diagnosis assessments i.e. the number of diagnoses in the prototypical case for the disease d in the set P_{Dom}, the prototypical cases generated by clinical domain experts;

D_r = the number of diagnoses for each disease d generated by this algorithm;

D_g = the number of good quality diagnoses assessments for the disease d where

$\exists\ da_i$, for i in 1 to n where n is the number of diagnosis assessments da for each disease d in P_{Dom}

and $da_i(PRel) = da_i\ (PDom)$

and $Qda(da_i\ (PRel)) = 100$

i.e. diagnosis assessments with a score of 100 for each disease;

$$D_g = \sum_{i=1}^{n} dai\ ,\ \text{where } Q_{da}(da_i\ (P_{Rel})) = 100$$

D_b be the number of diagnosis assessments retrieved, but have a score of less than 100 or not a relevant diagnosis step or a good diagnosis step, i.e.

$D_b = \sum_{i=1}^{n} dai$, where $Q_{da}(da_i (P_{Rel})) < 100$

For all the diagnosis assessments generated by this algorithm:

Diagnosis Assessment Precision(PD_{Rel}) = 11916.1 / 122 = 97.67 %

Diagnosis Assessment Recall(RD_{Rel}) = 11782.5 / 122 = 96.57 %

The precision and recall calculations for treatment assessment section of the prototypical cases are computed as:

$$\text{Precision P} = \frac{number\ of\ treatments\ for\ d\ that\ are\ good\ Tg}{number\ of\ treatments\ returned\ by\ this\ a\lg orithm\ for\ disease\ \ d\ , Tr}$$

$$\text{Recall R} = \frac{number\ of\ treatments\ for\ d\ that\ are\ good\ Tg}{number\ of\ treatments\ relevant\ for\ disease\ \ d\ returned\ by\ PDom\ , Td}$$

where

Q_t = the score for each treatment plans t for each disease d in P_{Rel};

T_d = the number of relevant treatment plans i.e. the number of treatments in the prototypical case for the disease d in the set P_{Dom}, the prototypical cases generated by clinical domain experts;

T_r = the number of treatment plans for each disease d generated by this algorithm;

T_g = the number of good quality treatment plans for the disease d where

$\exists\ t_i$, for i in 1 to n where n is the number of treatment plans t for each disease d in P_{Dom}

and $t_i(P_{Rel}) = t_i\ (P_{Dom})$

and $Q_t(t_i\ (P_{Rel})) = 100$

i.e. treatment plans with a score of 100 for each disease;

$$T_g = \sum_{i=1}^{n} ti \text{ , where } Q_t(t_i\ (P_{Rel})) = 100$$

T_b be the number of treatment plans retrieved, but have a score of less than 100 or not a relevant treatment step or a good treatment step, i.e.

$$T_b = \sum_{i=1}^{n} ti \text{ , where } Q_t(t_i\ (P_{Rel})) < 100$$

Treatment Assessment Precision(PT_{Rel}) = 11615.5 / 122 = 95.20 %

Treatment Assessment Recall (RD_{Rel}) = 11350.8 /122 = 93.03%

The summary of precision and recall values for all the three sections of the prototypical cases generated are as shown in table 26 below.

Table 26: Summary of the precision and recall values for the generated prototypical cases.

Prototypical case section	**Precision**	**Recall**
Findings	96.78	87.96
Diagnosis assessments	97.67	96.57
Treatment plans	95.20	93.03
Overall	**96.55**	**92.52**

6. Discussion

This thesis is based on the knowledge discovery in the form of prototypical cases. As seen from the validation results, the precision and recall values for the prototypical cases generated are fairly high. The reason behind this is also the fact that the prototypical cases were generated from the clinical patient cases that were reverse engineered from the expert-acquired prototypical cases. This is more of an idealistic situation.

The nature of the clinical patients' data is heterogeneous and it is dependent on the system vendor. The precision and recall rates could vary if the same algorithm was applied on the real-world clinical patients' data due to the diverse nature of data. The data for the thesis is stored in a relational database management system (RDBMS). The RDBMS used here is Microsoft SQL Server and the procedural language used is transact SQL (T-Sql).

The average precision and recall for all the prototypical cases generated are 96.55 and 92.52 respectively. These ratings are high because of the nature of the clinical patient cases that were generated from the expert-acquired prototypical cases. However, the question would arise as to why precision and recall are not 100 each, given the fact that the clinical patient cases are reverse engineered from the prototypical cases. The reason, however, is two-fold. The parsing algorithm that reads the expert-acquired prototypical cases in relational form and generates the patient cases does not take synonyms of disease or finding names into consideration. This leads to a few system generated prototypical case findings, diagnosis and treatments to not match with the original prototypical cases.

7. Merit

The merits of this methodology are derived from the fact that the knowledge can be discovered from applying relational mining methods on relational clinical data. The prototypical cases which were used as input to this thesis were generated by clinical domain experts. It took them nearly three years to derive prototypical cases from clinical data. Using relational mining methods on relational clinical patient data can generate knowledge structures in a fraction of the time taken by clinical domain experts, in a day based on the quantum of input data. The reverse-engineering algorithm generated 1.2 million clinical patient cases from 122 expert-acquired clinical cases in 8 hours. The prototypical case generation algorithm generated 122 prototypical cases from 1.2 million simulated clinical patient cases in 12 hours. There is a significant time saving in using relational mining methods to generate knowledge structures. Most of the current day clinical patient records (CPR) are in relational form [23], hence this thesis describes a way to discover knowledge structures in the form of prototypical cases directly from relational CPR.

8. Future work

This thesis was done as a starting step to see if advanced knowledge structures in the form of prototypical cases can be generated from advanced data mining techniques such as multi-relational data mining. As an improvement to the methods in this thesis the algorithm can be made more versatile to discover knowledge patterns based on diagnosis findings, procedures and morphologies rather than just findings. The foremost and primary step would be to start the same work on real clinical patients' data rather than simulated patient cases reverse engineered from the prototypical cases.

The diagnosis steps in the patients' data could be more meaningful if timestamps of diagnosis procedures are added to every diagnosis step. For example: if a diagnosis step A is performed and the results require patient observation for a month before moving on to the next step in the diagnosis, that could be added to help evolve better knowledge structures. There are findings in the diagnosis steps as a result of various diagnosis procedures. These findings can be taken into account for generating patterns and more detailed diagnosis and treatment procedures.

9. Educational statement

9.1 . Motivation

Data analysis has been a passion for me for quite some time. I wanted to apply what I learnt during my course of study in the CSS program to the area of knowledge discovery and pattern mining. My work experience has been on database administration all along and that helped me during the thesis methods to organize and look at data from different perspectives. The writing of the thesis itself was a new experience for me and I used the knowledge of what I learnt in the Master's seminar course (TCSS 598) to help me write this thesis. The learning from the courses, TCSS 555 – data mining, TCSS 435 – artificial intelligence contributed substantially towards the subject of this thesis and towards answering the research question.

9.2 . Knowledge gain

This thesis has helped me gain vast experience in the area of research in data mining and knowledge discovery. The thesis was focused on the medical domain which was new to me and it helped me learn new terminologies and exposed me to different ontologies in the medical domain. During the process of conducting this thesis, I was introduced to the concepts of case-based reasoning in bio-medicine and its significance in helping clinical diagnosis and treatment plans. Prototypical cases as representation of knowledge structures was a new data structure that I hadn't come across before. Now I believe that prototypical cases are essentially a robust form of knowledge structures or patterns in the mining of clinical or bio-medical data. The opportunity to use relational databases while

exploring the concept of using multi-relational data mining to discover knowledge structures was quite thrilling. The input data to the thesis, however, gave me an insight as to what needs to be improved to provide better data structures and cleaner data. I now look forward to explore deeper into the area of multi-relational data mining in the future and conduct more research in the field of knowledge discovery from clinical relational databases.

10. Conclusion

This thesis has shown that relational data mining as an advanced data mining techniques can be used to discover knowledge structures from clinical data. The prototypical cases obtained by this method have good accuracy rates with an overall precision of 96.55 and an overall recall of 92.52, when validated against the prototypical cases generated by clinical experts. The difference being that the input to this method is from simulated clinical patients' data versus the clinical domain experts who generated the knowledge structures from real clinical patients'. This thesis paved a path to generate knowledge structures in the form of prototypical cases when the same approach will be applied on real clinical patients' data. The one other factor not considered in this thesis is the performance of the multi-relational data mining algorithm. It would be a future area of research to look into this aspect of multi-relational data mining in generation of prototypical cases.

References

1. Bichindaritz, Isabelle, Editorial: Case Based Reasoning in the Health Sciences, Dept of CSS, University of Washington, Tacoma.

2. Bichindaritz, Isabelle, Memoire: A framework for semantic interoperability of case-based reasoning systems in biology and medicine, Artificial Intelligence in Medicine (2006) 36, 177-192

3. Bichindaritz, et. all., CARE-PARTNER: A Computerized Knowledge-Support System for Stem-Cell Post-Transplant Long-Term Follow-Up on the World-Wide-Web, Clinical Research Division, Fred Hutchinson Cancer Research Center, Seattle, Washington.

4. Nilsson, Markus, et. all., Advancements and Trends in Medical Case-Based Reasoning: An Overview of Systems and Systems Development, American Association of Artificial Intelligence, 2004.

5. Schmidt, Rainer, et. all., Case-based Reasoning for Medical Knowledge-based Systems, Institute of Medical Informatics and Biometry, University of Rostock.

6. Abidi, Syed, et all., Leveraging XML-based electronic medical records to extract experiential clinical knowledge: An automated approach to generate cases for medical case-based reasoning systems, International Journal of Medical Informatics 68(2002) 187-203.

7. Transational Research Data Repository report, Stanford University School of Medicine.

8. Bichindaritz, Isabelle, et all., Temporal Knowledge Representation and Organization for Case-Based Reasoning, IEEE, 1996.

9. Aha, David W, Feature Weighting for Lazy Learning Algorithms, Navy Center for Applied Research in Artificial Intelligence.

10. Paterson, Grace I, Creating a Digital Case Base for Medical and Health Informatics Education, Health Informatics Lab, Faculty of Computer Science, Dalhousie University, Canada.

11. Leroy, Gondy, et. all., Meeting Medical Terminology Needs – The Ontology-Enhanced Medical Concept Mapper, IEEE Transactions on Information Technology in Bio-medicine, Vol 5, No.4, 2001.

12. Brown, Barry, FDA XML Data Format Requirements Specification – a white paper.

13. Funk, Peter, et. all., Case-Based Reasoning and Knowledge Discovery in Medical Applications with Time Series, Dept. of Computer Science and Electronics, Malardalen University, Sweden.

14. Spyropoulos, B., A theoretical approach to artificial intelligence systems in medicine, AI in medicine, 7 (1995), 455-465.

15. Tierney, William M., Improving clinical decisions and outcomes with information: a review, International Journal of Medical Informatics 62, 2001, 1-9.

16. Hooda, Jagbir S., Health Level-7 Compliant Clinical Patient Records System, ACM Symposium on Applied Computing, 2004.

17. Ramirez, Jorge C G, et all., Medical Information Systems: Characterization and Challenges, SIGMOD RECORD, Vol. 23, No.3, Sep 1994.

18. Holbrook, Anne, et all., Applying methodology to electronic medical record selection, International Journal of Medical Informatics, 2003, 71, 43-50.

19. Aamodt, Agnar., Knowledge-Intensive Case-Based Reasoning in CREEK, Dept. of CIS, NTNU, Norway.

20. Agudo, Diaz B, et all., An Architecture for Knowledge Intensive CBR systems, Procs of the 5th European Workshop on Case-Based Reasoning, EWCBR 2000, 1898, Springer, 2000.

21. Dzeroski, Saso, et. all., Relational Data Mining, Springer, c2001.

22. Han, Jiawei, Data Mining, Concepts and Techniques, 2nd edition, Morgan Kaufmann.

23. Eichelberg, Marco, et. all., A Survey and Analysis of Electronic Healthcare Record Standards, ACM computing surveys, vol. 37, No. 4, December 2005, pp. 277-315.

Appendix 1: Expert-acquired prototypical case example

Findings section

Connector	Finding Name	Snomed code	(Properties, Values)	Importance	Level
	ChestPainNOS	F-37000		M	H
OR	DyspneaNOS	F-20040		M	H
OR	Tachypnea	F-21003		M	H
OR	PericardialFrictionRub	F-35750	Status = Present	C	
OR	Fever	F-03003			H
OR	TachycardiaNOS	F-33140			M
OR	KussmaulPulse	F-31620		C	H
OR	ParadoxicalPulse	F-31620		C	H

Diagnosis assessments section

Connector	Procedure Name	Snomed code	(Properties, Values)	Importance	Order
	EKG	P2-31022	Finding = STElevations	1	C
AND	CBC	P3-30100	Finding = WBC value = increased (> 15K)	1	C
AND	ESR		Result = increased	1	C
AND	CXR		Result = Normal OR Abnormal	1	C
AND	Echocardiography	P5-B3000	Result = Abnormal	1	N

Treatment plans section

Condition/ Connector	Planning Action Name	(Properties, Values)	Order
	PlacePatientAtBedRest		1
AND	ViralCultureNOS(P3-50260)	Site = Blood(T-C2000) AND site = Urine(T-70060) AND site = throat(T-55000)	2
AND	FungusCultureNOS(P3-50250)		2
AND	MicrobialCultureNOS(P3-50100)	Searchfor = BacteriaNOS(L-10000) OR AcidFastBacillum(AFB)(L-21800)	2
AND	MicrobialSmearExamination(P3-50300)		2
AND IF cultures.result = Negative	AskCurrentDrugHistory ConsiderDrugInducedPericarditis		3
AND IF cultures.result = Positive	PrescriptionOfTherapeuticAgentNOS	Type = AppropriateAntibiotic OR AppropriateAntiviral OR AppropriateAntifungal	3
AND	FollowEchocardiogram	Period = weekly AND until = Resolved	4

Appendix 2 : Pseudo code – patient cases' generation

Pseudo code for reverse engineering of patient cases from the relational prototypical cases.

Pseudo code: Generate patient clinical case for findings for a disease

Variable assignment:

```
SymptomName = blank
ANDCount = 0
ORCount = 0
IFCount = 0
Importance = blank
CaseGen = 0  // a factor to determine how many clinical cases can be generated
ConsideredFlag = 0 // a flag on the findings table to indicate if this case is considered

        //Start with the first finding thru all the findings

        Set ANDCount = number of AND connectors for the disease
        Set ORCount = number of OR connectors for the disease

Subroutine DisplaySymptomName( )

For each disease in the disease findings relational table

        Set SymptomName = First symptom name or the base name with no connector
        Set Importance = Value of Importance for the base symptom

        Display 'SymptomName'
End Subroutine DisplaySymptomName()

Subroutine DisplaySymptomProperties()
        Declare
        cursor = All (properties, values) from the findings properties for symptom

        Open cursor
                While there are (properties, values) in the cursor
                        Set SymptomPropertyName = PropertyName
                        Set SymptomPropertyValue = PropertyValue

                        Display 'SymptomPropertyName' 'SymptomPropertyValue'
```

```
                End While
        Close cursor
End Subroutine DisplaySymptomProperties()

If ANDCount > 0 then
        Declare
        cursor = All findings for the disease with 'AND' connector order by Importance

        While records exist in cursor
                DisplaySymptomName( )

                For this Symptom Name
                        DisplaySymptomProperties( )
                End For loop

                ANDCount = ANDCount -1

        End while loop

End If

If ORCount > 0

        CaseGen = Round( ORCount / 3)

        If CaseGen < 1
                CaseGen = 1
        End If

End If

If CaseGen > = 1

        Declare
        Cursor = All findings for the disease with OR connector

        Open Cursor

                While records exist in Cursor
                        DisplaySymptomName( )
                        DisplaySymptomProperties( )

                        CaseGen = CaseGen -1
```

Set ConsideredFlag = 1 for this finding in table

CaseGen = CaseGen - 1
End While

Close Cursor

End If
End Pseudocode for findings
Pseudocode: Generate Patient diagnostic steps for the disease

Precondition: Diagnostic steps will be generated for a patient for the findings described above

Variable Assignment:

DiagnosisCount = 0
ANDCount = 0
IFCount = 0
ConsideredFlag = 0

Declare
Cursor = All diagnostic records for the disease from the relation diagnosis ordered by Order and Importance

Set ConnectorType = blank

Set IFCount = Number of IF connectors in the diagnosis for the disease

Open Cursor

If IFCount > 0

CaseGen = IFCount / 3

If CaseGen < 1
CaseGen = 1
End If
End If

While records exist in the cursor

Set ConnectorType = Connector of the current diagnosis assessment record

```
        If ConnectorType is null then

                DisplayDiagnosisProcedure( )

                DisplayDiagnosisProcProperties( DiagnosisProcedure )
        End If

        If ConnectorType = AND then

                DisplayDiagnosisProcedure( )
                DisplayDiagnosisProcProperties( DiagnosisProcedure )
        End If

        If ConnectorType = IF and CaseGen >0  then
                CaseGen = CaseGen – 1
                Set ConsideredFlag = 1
                Update diagnosis table with ConsideredFlag for this diagnosis

                DisplayPreDiagnosisTest( )
                DisplayDiagnosisProcedure( )
                DisplayDiagnosisProcProperties( DiagnosisProcedure )

        End If

    End While loop

Close Cursor

Subroutine DisplayPreDiagnosisTest( )
Input: Diagnosis Id for the diagnosis

Variable Assignment:

PreDiagTest = blank
PreDiagTestResult = blank


Begin
    Set PreDiagTest = Pre Diagnosis Test value for the diagnosis assessment

    Set PreDiagTestResult = Pre Diagnosis Test Result value for the diagnosis

    Display 'Pre-Diagnosis Test' ='PreDiagTest
    Display 'Pre-Diagnosis Test Result' = 'PreDiagTestResult'
```

End

Return Display results

End Subroutine DisplayPreDiagnosisTest( )

Subroutine DisplayDiagnosisProcedure( )
Input: Diagnosis Id for the diagnosis

Variable Assignment:

DiagProcedure = blank

Begin

Set DiagProcedure = PlanningActionName for the diagnosis from the table

Display 'Diagnostic Procedure' = DiagProcedure

End

Return Display variables

End Subroutine DisplayDiagnosisProcedure( )

Subroutine DisplayDiagnosisProcProperties(DiagnosisProcedure)

Input: Diagnosis Id for the diagnosis

Variable Assignment:

DiagProcProp = blank
DiagProcPropValue = blank

Declare
Cursor = All propertiesValues from the DiagnosticProperties relation for the diagnosisId

// Multiple propertiesValues could be present for each Diagnosis Assessment

Open Cursor

While records exist in Cursor

Set DiagProcProp = PropertyName for this record row

Set DiagProcPropValue = PropertyValue for this record row

Display 'Diagnostic Procedure Property' = DiagProcProp

Display 'Diagnostic Procedure Property Value' = DiagProcPropValue

End while
Close cursor

Return Display variables

End Subroutine DisplayDiagnosisProcProperties(DiagnosisProcedure)

Pseudocode: Generate Patient treatment steps administered for the disease

Precondition: Treatment steps will be generated for a patient for the findings described above

Variable Assignment:

TreatmentCount = 0
ANDCount = 0
IFCount = 0
ConsideredFlag = 0

Declare
Cursor = All treatment records for the disease from the relation treatment ordered by Order

Set ConnectorType = blank

Set IFCount = Number of IF connectors in the treatment plans for the disease

Open Cursor

If IFCount > 0

CaseGen = IFCount / 3

If CaseGen < 1
CaseGen = 1
End If
End If

While records exist in the cursor

Set ConnectorType = Connector of the current treatment assessment record

If ConnectorType = blank or ConnectorType = AND then

DisplayTreatmentAction( )

DisplayTreatmentProperties( )
End If

If ConnectorType = IF then

If CaseGen > 0 then

Set CaseGen = CaseGen -1

Set ConsideredFlag = 1
Update Treatment plan relation with the ConsideredFlag

DisplayPreTreatmentTests( )

DisplayTreatmentAction( )

DisplayTreatmentProperties( )

End If
End If

End While

Close Cursor

Subroutine DisplayPreTreatmentTests( )

Input: Treatment Id

Variables Assignment:

PreTreatTest = blank
PreTreatTestValue = blank

Set PreTreatTest = Pre-treatment Test with the IF condition of the Treatment Id

Set PreTreatTestValue = Pre-treatment test value for the TreatmentId for the row

Display 'Pre-Treatment Test Name' = PreTreatTest

Display 'Pre-Treatment Test Value = PreTreatTestValue

Return Display variables

End Subroutine DisplayPreTreatmentTests( )

Subroutine DisplayTreatmentAction(TreatmentId)

Input: Treatment Id for the disease

Variables Assignment:

TreatAction = blank

Begin
Set TreatAction = Treatment PlanningActionPlan for the TreatmentId

Display 'Treatment Administered' = TreatAction
End

Return Display variables

End Subroutine DisplayTreatmentACtion( )

Subroutine DisplayTreatmentProperties( )

Input: Treatment Id

Variables Assignment:

TreatMetric = blank
TreatMetricValue = blank

Declare
Cursor = All the Treatment propertiesValues for this TreatmentId

Open Cursor
While records exist in cursor

Set TreatMetric = PropertyName
Set TreatMetricValue = PropertyValue

Display 'Treatment Metric' = TreatMetric
Display 'Treatment Metric Value' = TreatMetricValue

End While loop
Close cursor

Return Display variables

End Subroutine DisplayTreatmentProperties( )

Appendix 3: Code – clinical patient cases' generation

The code for generating the reverse engineered patient cases is written in three parts – patient clinical findings, patient diagnosis assessments and patient treatment plans.

Script to reverse engineer clinical patient findings

```
-- variables for table relfindings
update relFindings set CFlag = 0


DECLARE @condid numeric (18) -- disease condition id
DECLARE @ptid numeric (18) -- number of patient id
SET @ptid = 0

DECLARE cur1 CURSOR FORWARD_ONLY
for
SELECT conditionId from diseaseconditionlookup

OPEN cur1

FETCH next from cur1 into @condid

WHILE (@@FETCH_STATUS = 0)
BEGIN

DECLARE @var1 numeric (18) --findingId
DECLARE @var2 numeric (18) --conditionId
DECLARE @var3 varchar (10) --connector
DECLARE @var4 varchar (50) --findingName
DECLARE @var5 char (4) -- level

-- variables for table relfindProps

DECLARE @var6 varchar (100) --propertyName
DECLARE @var7 varchar (100) --propertyValue

DECLARE @gencnt int -- cases generation parameter

DECLARE @orcnt int -- or count of the findings connectors

SET @orcnt = (SELECT count(*) from relFindings where
conditionID=@condId
and connector like 'OR%' and CFLag=0)

IF @orcnt < 0 SET @orcnt = 0

IF @orcnt >= 10 SET @orcnt = @orcnt/4
```

```
execute @gencnt = apFactorial @inumber=@orcnt

PRINT +@gencnt
-- IF loop 1
if @gencnt = 0 SET @gencnt = 1

While @gencnt > 0
BEGIN

PRINT +@gencnt

SET @ptid = @ptid + 1

-- Loop for And connectors

DECLARE s cursor FORWARD_ONLY
for
Select findingId,conditionId,connector,findingName,[level] from
relfindings
where conditionId = @condid
AND (connector like 'AND%'
or connector is null)

OPEN s

FETCH next from s into @var1,@var2,@var3,@var4,@var5

      WHILE (@@FETCH_STATUS = 0)
      BEGIN -- 2 begin

            DECLARE @propcnt int

            SET @propcnt = (select count(*) from relFindProps where
findingId = @var1)

            IF @propcnt < 0 SET @propcnt = 0

            IF @propcnt > 0
            BEGIN -- 3 begin

                  DECLARE c cursor FORWARD_ONLY
                  for
                  Select propertyName,propertyValue from relFindProps
where findingId=@var1

                  OPEN c

                  FETCH next from c into @var6, @var7

                  WHILE (@@FETCH_STATUS = 0)
                        BEGIN -- 4 begin

                        INSERT INTO patientFindings
(patientId,conditionId,findingName,PropertyType,
```

```
                                                    PropertyValue,[Level])
values

(@ptid,@var2,@var4,@var6,@var7,@var5)

                        FETCH next from c into @var6, @var7
                        END -- 4

                    CLOSE c
                    DEALLOCATE c
                END -- 3

            ELSE

                    BEGIN -- 5 BEGIN

                    INSERT INTO patientFindings
(patientId,conditionId,findingName,[Level]) values
                                            (@ptid,@var2,@var4,@var5)

                    END -- 5

      FETCH next from s into @var1,@var2,@var3,@var4,@var5

      END -- 2

      CLOSE s
      DEALLOCATE s
-------------------------------------------------------------------------
---------------------------------------
-- IF loop 2

SET @orcnt = (select count(*) from relFindings where
conditionId=@condid and connector like 'OR%' and CFlag=0)

IF @orcnt = 0 --BREAK
BEGIN
UPDATE relFindings set CFlag=0 where conditionId = @condid
END

IF @orcnt > 0

BEGIN --1 begin

DECLARE s cursor FORWARD_ONLY
for
Select findingId,conditionId,connector,findingName,[level] from
relfindings
where conditionId = @condid
AND connector like 'OR%'
AND CFlag = 0

OPEN s
```

```
FETCH next from s into @var1,@var2,@var3,@var4,@var5

DECLARE @cntrec int
SET @cntrec = 1

      WHILE (@@FETCH_STATUS = 0)
      BEGIN -- 2 begin

            DECLARE @propcnt1 int

            SET @propcnt1 = (select count(*) from relFindProps where
findingId = @var1)

            IF @propcnt1 < 0 SET @propcnt1 = 0

            IF @propcnt1 > 0
            BEGIN -- 3 begin

                  DECLARE c cursor FORWARD_ONLY
                  for
                  Select propertyName,propertyValue from relFindProps
where findingId=@var1

                  OPEN c

                  FETCH next from c into @var6, @var7

                  WHILE (@@FETCH_STATUS = 0)
                        BEGIN -- 4 begin

                        INSERT INTO patientFindings
(patientId,conditionId,findingName,PropertyType,
                                                PropertyValue,[Level])
values

(@ptid,@var2,@var4,@var6,@var7,@var5)

                        IF @cntrec = 1
                        BEGIN
                        UPDATE relFindings set CFlag=1 where
findingId=@var1
                        END

                        SET @cntrec = @cntrec + 1

                        FETCH next from c into @var6, @var7
                        END -- 4

                  CLOSE c
                  DEALLOCATE c
              END -- 3

            ELSE
```

```
                    BEGIN -- 5 BEGIN

                    INSERT INTO patientFindings
(patientId,conditionId,findingName,[Level]) values
                                            (@ptid,@var2,@var4,@var5)

                    IF @cntrec = 1
                    BEGIN
                    UPDATE relFindings set CFlag = 1 where findingId =
@var1
                    END
                    SET @cntrec = @cntrec + 1

                    END -- 5

      FETCH next from s into @var1,@var2,@var3,@var4,@var5

      END -- 2

      CLOSE s
      DEALLOCATE s

END -- 1

SET @gencnt = @gencnt - 1
print +@gencnt

--SET @ptid = @ptid + 1
print +@ptid

END -- gencnt loop
------------------------------------------------------------------------
----------------------------------------
FETCH next from cur1 into @condid

END

CLOSE cur1
DEALLOCATE cur1

--END
```

Script to load patient diagnosis

```
-- LOAD table patientdiagAssess

DECLARE @condid numeric (18) -- disease condition id
DECLARE @ptid numeric (18) -- number of patient id

SET @ptid = 0

DECLARE cur1 CURSOR FORWARD_ONLY
```

```
for
SELECT distinct patientId,conditionId from patientFindings

OPEN cur1

FETCH next from cur1 into @ptid,@condid

WHILE (@@FETCH_STATUS = 0)
BEGIN -- 1

DECLARE @var1 numeric (18) --conditionId
DECLARE @var2 varchar (10) --connector
DECLARE @var3 varchar (100) -- preDiagTest
DECLARE @var4 varchar (100) -- preDiagTestResult
DECLARE @var5 varchar (50) -- procedurename
DECLARE @var6 varchar (100) -- properties
DECLARE @var7 varchar (10) -- propertyConnector
DECLARE @var8 varchar (100) -- propertyValue
DECLARE @var9 char (1) -- importance
DECLARE @var10 numeric (9) -- order

DECLARE @andcnt int -- and count of the tblDiagAssess connectors
DECLARE @orcnt int -- or count of the tblDiagAssess connectors
DECLARE @ifcnt int -- if count of the tblDiagAssess connectors


SET @andcnt = 0
SET @orcnt = 0
SET @ifcnt = 0

SET @andcnt = (select count(*) from tblDiagAssess where
conditionId=@condid and connector like 'AND%')

PRINT +@andcnt

SET @ifcnt = (select count(*) from relfindings where
conditionId=@condid and connector like 'IF%')

PRINT +@ifcnt


--IF @andcnt > 0 or @ifcnt > 0

DECLARE s cursor FORWARD_ONLY
for
Select
preDiagTest,preDiagTestResult,ProcedureName,properties,propertyConnecto
r,propertyValue,Importance,[Order]
from tblDiagAssess
where conditionId = @condid
order by [Order]

OPEN s
```

```
FETCH next from s into @var3,@var4,@var5,@var6,@var7,@var8,@var9,@var10

DECLARE @diagstep int -- diagnosis step in order
SET @diagstep = 1

      WHILE (@@FETCH_STATUS = 0)
      BEGIN -- 2 begin


      INSERT INTO patientDiagAssess values
(@ptid,@condid,@diagstep,@var3,@var4,@var5,@var6,@var7+@var8)

      SET @diagstep = @diagstep+1

      FETCH next from s into
@var3,@var4,@var5,@var6,@var7,@var8,@var9,@var10

      END -- 2

CLOSE s
DEALLOCATE s

FETCH next from cur1 into @ptid,@condid

END -- 1

CLOSE cur1
DEALLOCATE cur1
```

Script to load patient treatment plans

```
-- LOAD table patientTreatmentPlan

DECLARE @condid numeric (18) -- disease condition id
DECLARE @ptid numeric (18) -- number of patient id

SET @ptid = 0

DECLARE cur1 CURSOR FORWARD_ONLY
for
SELECT distinct patientId,conditionId from patientFindings

OPEN cur1

FETCH next from cur1 into @ptid,@condid

WHILE (@@FETCH_STATUS = 0)
BEGIN -- 1

-- variables for tblTreatmentplan
```

```
DECLARE @var1 numeric (9) -- treatmentId
DECLARE @var2 varchar (10) -- connectorControl
DECLARE @var3 varchar (50) -- conditionName
DECLARE @var4 varchar (5) -- conditionSign
DECLARE @var5 varchar (50) -- conditionValue
DECLARE @var6 varchar (150) -- planningActionName
DECLARE @var7 numeric (9) -- order

-- variables for tblTreatmentProps

DECLARE @var8 varchar(20) -- treatmentProperty
DECLARE @var9 varchar(5) -- propertySign
DECLARE @var10 varchar(80) -- propertyValue

DECLARE @andcnt int -- and count of the tbltreatmentPlan connectors

DECLARE @ifcnt int -- if count of the tbltreatmentPlan connectors


SET @andcnt = 0

SET @ifcnt = 0

SET @andcnt = (select count(*) from tblDiagAssess where
conditionId=@condid and connector like 'AND%')

PRINT +@andcnt

SET @ifcnt = (select count(*) from relfindings where
conditionId=@condid and connector like 'IF%')

PRINT +@ifcnt


DECLARE s cursor FORWARD_ONLY
for
Select
treatmentId,connectorcontrol,conditionname,conditionsign,conditionvalue
,planningactionname,[order]
from
tblTreatmentPlan where conditionId = @condid
order by [order]


OPEN s

FETCH next from s into @var1,@var2,@var3,@var4,@var5,@var6,@var7

DECLARE @treatstep int -- treatment steps in order
SET @treatstep = 1

      WHILE (@@FETCH_STATUS = 0)
      BEGIN -- 2 begin
```

```
            DECLARE @propcount int -- properties for each treatment

            SET @propcount = (Select count(*) from TreatmentProps where
treatmentId = @var1)

            PRINT  +@propcount

            IF @propcount < 0 SET @propcount = 0

            IF @propcount > 0
            BEGIN -- 3 begin

                  DECLARE c cursor FORWARD_ONLY
                  FOR
                  Select treatmentProperty,PropertySign,PropertyValue
from treatmentProps where treatmentId = @var1

                  OPEN c

                  FETCH next from c into @var8,@var9,@var10

                  WHILE (@@FETCH_STATUS = 0)
                        BEGIN -- 4 begin

                        INSERT INTO patientTreatmentPlan values
(@ptid,@condid,@treatstep,@var3,@var4+@var5,

      @var6,@var8,@var9+@var10)
                        SET @treatstep = @treatstep + 1

                        FETCH next from c into @var8,@var9,@var10

                        END -- 4
                  -- End while LOOP
                  CLOSE c
                  DEALLOCATE c
            END -- 3
            ELSE
                  BEGIN       -- 5 Begin
                        INSERT INTO patientTreatmentPlan
(patientId,conditionId,TreatmentStep,preTreatmentTest,

preTreatmentTestResult,TreatmentAdministered ) values

(@ptid,@condid,@treatstep,@var3,@var4+@var5,@var6)

                        SET @treatstep = @treatstep + 1
                  END -- 5

            -- End IF loop

      FETCH next from s into @var1,@var2,@var3,@var4,@var5,@var6,@var7
```

```
      END -- 2

CLOSE s
DEALLOCATE s

FETCH next from cur1 into @ptid,@condid

END -- 1

CLOSE cur1
DEALLOCATE cur1
```

-- Code section for generation of diagnosis assessments
-- section of prototypical cases

DECLARE @diagStep int -- variable to store the diagnosis step of the disease

DECLARE @prediagTest varchar(100) -- variable to hold the prediagnosisTest

DECLARE @prediagtestresult varchar(100) -- variable to hold the preDiagnosisTestResult

DECLARE @diagProcedure varchar(50) -- variable to hold the diagnosisProcedure

DECLARE @procProperty varchar(100) -- variable to hold the procedureProperty

DECLARE @procPropertyValue varchar(100) -- variable to hold the procedurePropertyValue

SET @counter = 0

-- For each clinical patient case consider the disease associated with the patient

DECLARE cur11 CURSOR FORWARD_ONLY
FOR
*SELECT * FROM PATIENTDIAGASSESS*
WHERE CONDITIONID = @conditionId
order by patientid

OPEN cur11

FETCH NEXT FROM cur11 INTO @patientId, @conditionId, @diagStep, @prediagTest,

```
@prediagtestResult, @diagProcedure, @procProperty, @procPropertyValue -- Fetch
the first row from the cursor


WHILE (@@FETCH_STATUS = 0) -- for each row containing a patientid and a
conditionid
                        -- process the record to find the diagnosis assessments
                        -- for the patient

BEGIN

SET @COUNTER = (Select count(*) from diagSet
where conditioniId = @conditionId and diagnosisStep=@diagStep)

IF @COUNTER < 0 SET @COUNTER = 0

IF @COUNTER = 0

BEGIN

INSERT INTO diagSetTemp values(
@conditionId, @diagStep, @prediagTest,
@prediagtestResult, @diagProcedure, @procProperty, @procPropertyValue)


        DECLARE cur12 CURSOR FORWARD_ONLY
        FOR
        SELECT DISTINCT CONDITIONID,DIAGNOSISSTEP,PREDIAGNOSISTEST,
        PREDIAGNOSISTESTRESULT, DIAGNOSISPROCEDURE,
PROCEDUREPROPERTY
        PROCEDUREPROPERTYVALUE
        FROM PATIENTDIAGASSESS WHERE
        conditionId = @conditionId
        and diagnosisStep = @diagStep


        OPEN cur12

        FETCH NEXT from cur12 into
        @conditionId, @diagStep, @prediagTest,
        @prediagtestResult, @diagProcedure, @procProperty, @procPropertyValue

        WHILE (@@FETCH_STATUS = 0) -- for each row of finding name compare to
        see how many
```

```
                                -- diseases and patients this diagnosisstep occurs in

        BEGIN

                INSERT INTO diagSetTemp values(
                @conditionId, @diagStep, @prediagTest,
                @prediagtestResult, @diagProcedure, @procProperty,
                @procPropertyValue)
                -- diagSetTemp is a temporary table that holds the sets of all the
                -- diagnosis assessments for a diagnosis step for a disease


        END
        CLOSE cur12
        DEALLOCATE cur12

        Insert into diagSet
        SELECT DISTINCT CONDITIONID,DIAGNOSISSTEP,PREDIAGNOSISTEST,
        PREDIAGNOSISTESTRESULT, DIAGNOSISPROCEDURE,
        PROCEDUREPROPERTY
        PROCEDUREPROPERTYVALUE
        FROM diagSetTemp WHERE
        conditionId = @conditionId
        and diagnosisStep = @diagStep

        -- diagSet is the relation which holds the final distinct set of all the diagnosis
        -- patterns for a certain diagnostic step for a disease

END
        FETCH next FROM cur11 INTO @patientId, @conditionId,
        @diagStep, @prediagTest,
        @prediagtestResult, @diagProcedure, @procProperty, @procPropertyValue

END

CLOSE cur11

DEALLOCATE cur11
```

```
-- Code to generate treatment plan section for
-- the prototypical cases


DECLARE @patientId int              -- variable to store the patient id for each patient

DECLARE @conditionId int  -- variable to store the condition id associated with each
patient
                                    -- to retrieve the diseasename from the disease lookup
relation

DECLARE @treatmentStep int          -- variable to store treatment step

DECLARE @preTreatmentTest varchar(50) -- variable to store preTreatmentTest

DECLARE @preTreatmentTestResult varchar(50) -- variable to store
preTreatmentTestResult

DECLARE @treatmentAdministered varchar(150) -- variable to store
treatmentAdministered

DECLARE @treatmentMetric varchar(20) -- variable to store treatmentMetric

DECLARE @treatmentMetricValue varchar(80) -- variable to store
treatmentMetricValue


DECLARE @counter int          -- a counter initialized to be used anywhere in the
algorithm

SET @counter = 0

-- For each clinical patient case consider the treatment associated with the patient

DECLARE cur21 CURSOR FORWARD_ONLY
FOR
SELECT * FROM patientTreatmentPlan
WHERE CONDITIONID = @conditionId
order by patientid


OPEN cur21
```

```
FETCH NEXT FROM cur31
INTO @patientId, @conditionId, @treatmentStep, @preTreatmentTest,
@preTreatmentTestResult, @treatmentAdministered,
@treatmentMetric, @treatmentMetricValue -- Fetch the first row from the cursor


WHILE (@@FETCH_STATUS = 0) -- for each row containing a patientid and a
conditionid
                        -- process the record to find the treatment plan
                        -- for the patient

BEGIN

SET @COUNTER = (Select count(*) from treatSet
where conditionId = @conditionId and treatmentStep = @treatmentStep)

IF @COUNTER < 0 SET @COUNTER = 0

IF @COUNTER = 0

BEGIN

INSERT INTO treatSetTemp values(
@conditionId, @treatmentStep, @preTreatmentTest,
@preTreatmentTestResult, @treatmentAdministered,
@treatmentMetric, @treatmentMetricValue)


    DECLARE cur32 CURSOR FORWARD_ONLY
    FOR
    SELECT DISTINCT conditionId, treatmentStep,
    preTreatmentTest, preTreatmentTestResult,
    treatmentAdministered, treatmentMetric
    from patientTreatmentPlan WHERE
    conditionId = @conditionId
    and treatmentStep = @treatmentStep


    OPEN cur32

    FETCH NEXT from cur32 into
    @conditionId, @treatmentStep, @preTreatmentTest,
    @preTreatmentTestResult, @treatmentAdministered,
```

```
        @treatmentMetric, @treatmentMetricValue

        WHILE (@@FETCH_STATUS = 0) -- for each row of treatment step compare to
see how many
                                   -- diseases and patients this treatment step occurs in

        BEGIN

                INSERT INTO treatSetTemp values(
                @conditionId, @treatmentStep, @preTreatmentTest,
                @preTreatmentTestResult, @treatmentAdministered,
                @treatmentMetric, @treatmentMetricValue)



        END
        CLOSE cur32
        DEALLOCATE cur32

        Insert into treatSet
        SELECT DISTINCT conditionId, treatmentStep,
        preTreatmentTest, preTreatmentTestResult,
        treatmentAdministered, treatmentMetric
        FROM treatSetTemp WHERE
        conditionId = @conditionId
        and treatmentStep = @treatmentStep

END
        FETCH next FROM cur31 INTO @patientId,
        @conditionId, @treatmentStep, @preTreatmentTest,
        @preTreatmentTestResult, @treatmentAdministered,
        @treatmentMetric, @treatmentMetricValue

END

CLOSE cur31

DEALLOCATE cur31
```

Appendix 4 : Code – prototypical case generation

The code for generating prototypical cases from clinical patient cases is as shown below. The programming language used is transact SQL to write the procedural code to generate prototypical cases from relational clinical patient cases.

```
-- Relational Query Language (RQL) Main

-- This code is built on the RQL classifier algorithm

-- The code is written in Transact SQL and the data is on MS SQL Server RDBMS


-- Part 1: Read thru the patient cases for each disease and calculate their support
ranking
-- a ranking factor designed to determine the occurance of each finding for a disease
-- and the number of their occurrences to determine their logical connector to a pattern

DECLARE @findingName varchar(50)-- variable to store the finding name from the
patient

DECLARE @diseaseName varchar(50)-- variable to store the disease name associated
with the patient

DECLARE @findingCount int          -- variable to store the finding name occurances for
each disease

DECLARE @patientId int             -- variable to store the patient id for each patient

DECLARE @conditionId int  -- variable to store the condition id associated with each
patient
                                   -- to retrieve the diseasename from the disease lookup
relation

DECLARE @counter int               -- a counter initialized to be used anywhere in the
algorithm

SET @counter = 0

-- For each clinical patient case consider the disease associated with the patient

DECLARE cur1 CURSOR FORWARD_ONLY
FOR
```

```
SELECT DISTINCT PATIENTID,CONDITIONID FROM PATIENTFINDINGS


OPEN cur1

FETCH NEXT FROM cur1 INTO @patientId, @conditionId -- Fetch the first row from
the cursor

SET @diseaseName = (select conditionName from diseaseconditionLookup
                        where conditionId = @conditionId) -- initialize the
diseaseName to the
                                                -- name of the disease from
                                                -- condition Id

WHILE (@@FETCH_STATUS = 0) -- for each row containing a patientid and a
conditionid
                        -- process the record to find the findings for the patient

BEGIN

    DECLARE cur2 CURSOR FORWARD_ONLY
    FOR
    SELECT DISTINCT FINDINGNAME from PATIENTFINDINGS
    WHERE PATIENTID = @patientId
    -- declare a cursor to obtain all the findings for the patient
    -- and rank each finding for the similar condition/disease occurring
    -- in all patients


    OPEN cur2

    FETCH NEXT from cur2 into @findingName

    WHILE (@@FETCH_STATUS = 0) -- for each row of finding name compare to
see how many
                        -- diseases and patients this finding name occurs in

    BEGIN

        SET @findingCount = 0

        SET @findingCount = (
        SELECT supportCount from diseaseRanking
        where findingName = @FINDINGNAME
```

```
                and diseaseName = @diseaseName)

                -- the diseaseRanking table ranks each finding for a disease
                -- based on its occurance in the clinical patient data set
                -- If the finding name has already been ranked then it won't
                -- be re-ranked. If it hasn't already been ranked for this condition
                -- then it will be ranked

                IF @findingCount < 0 SET @findingCount = 0

                IF @findingCount = 0
                BEGIN

                        SET @findingCount = (SELECT COUNT(patientId) from
                        patientFindings where findingName = @FINDINGNAME
                        AND conditionId = @conditionId)

                        INSERT INTO diseaseRanking
                        (findingname, diseaseName, supportCount) values
                        (@findingName, @diseaseName, @findingCount)

                -- update the diseaseRanking relation with the latest support
                -- ranking for each finding for the disease under scrutiny


                END


                FETCH next FROM cur2 into @findingName

        END
        CLOSE cur2
        DEALLOCATE cur2

        FETCH next FROM cur1 into @patientId, @conditionId

END

CLOSE cur1

DEALLOCATE cur1


--
```

```
-- Part 2

-- This part generates prototypical cases

-- Select all the findings associated with a disease


DECLARE cur3 CURSOR FORWARD_ONLY
FOR
SELECT diseaseName,findingName,supportCount
from diseaseRanking order by supportCount Desc

DECLARE @rankOrder int -- variable to store the highest ranking
                       -- or support count to determine the
                       -- AND or OR connector

DECLARE @count1 int -- variable to count the number of
                    -- findings for each disease

OPEN cur3

FETCH next from cur3 into @diseaseName, @findingName,@findingcount

SET @count1 = (SELECT count(findingName) from diseaseRanking
          WHERE diseaseName = @diseaseName)

SET @counter = @findingcount

WHILE (@@FETCH_STATUS = 0)

BEGIN

        IF @findingcount = @counter

        BEGIN

        INSERT INTO findingsPattern (diseaseName, connector, findingName)
        values
        (@diseaseName,'AND',@findingName)

        ELSE

        BEGIN
```

```
        INSERT INTO findingsPattern (diseaseName, connector, findingName)
        values
        (@diseaseName,'OR',@findingName)

        END

        FETCH next from cur3 into @diseaseName, @findingName,@findingcount


END

CLOSE cur3
DEALLOCATE cur3
```

Appendix 5 : System generated prototypical case example

An example of the generated prototypical case for the disease LiverChronicGVHD is as shown below.

Findings section

Connector	findingName	SnomedCode	PropertyType	PropertyValue	Imp	Level
AND	JaundiceNOS	M-57620			H	
AND	HepatoToxicDrug				H	A
OR	Fever	F-03003			M	
OR	Urine	T-70060	Color	Dark	M	
OR	Nausea	F-52760			M	
OR	Anorexia	F-50015			M	
OR	Malaise	F-01220			M	
OR	PainNOS	F-A2600	Site	RightUpper Quadrant Abdomen	M	
OR	Stool	T-59666	Color	Light	M	
OR	Hepatomegaly	D5-81220			M	
OR	Ascites	D5-70400			M	
OR	PeripheralEdema	M-36330			M	

Diagnosis Assessment section

Patient ID	Diagnosis Step	Pre Diagnosis Test	Pre Diagnosis Test Result	Diagnosis Procedure	Procedure Property	Procedure Property Value
1	1			HepaticFunction Panel	Finding	Alkaline Phosphasphatase Measurement (ALKP)
1	2			HepaticFunction Panel	Result	Elevated
1	3			HepaticFunction Panel	Finding	AST Measurement (AST)
1	4			HepaticFunction Panel	Result	Elevated
1	5			HepaticFunction Panel	Finding	LDH Measurement (LDH)
1	6			HepaticFunction Panel	Result	Elevated
1	7			HepatitisPanel	Finding	Hepatitis Panel Measurement
1	8			HepatitisPanel	Result	Negative

Diagnosis Assessment section continued.

1	9			Ultrasonography AbdomenNOS (USNABD)	Finding	Normal
1	10			CBC	Finding	Eosinophils
1	11			CBC	Result	Elevated
1	12	HepatitisC Antigen Measurement	Positive	HCVRNA Measurement	Finding	Negative
1	13	Hepatitis B Antigen Measurement	Negative			
1	14			Oral Examination	Finding	Abnormal
1	15			BiopsyOfLipNOS	Finding	Negative
1	16			BiopsyOfSkin	Finding	Negative
1	17			SchirmerTearTest	Finding	Decerased
1	18			BiopsyOfLiver	Finding	Positive
1	19			BiopsyOfLiver	Diagnosis	LiverChronic GVHD
1	20			RequestGIConsult		

Treatment assessments section

Connector	**Treatment Order**	**Pre Treatment Test**	**Pre Treatment Test Result**	**Treatment Administered**	**Treatment Metric**	**Treatment Metric Value**
IF	1	Immuno suppressant Agent NOS	Absent	Start PDNCSP Therapy		
IF	1	Immuno suppressant Agent NOS	Present	Salvage Rx Protocol Limit Hepatotoxic Drugs		
IF	2	PDN	Present	Consider UDCARx Protocol		