Abstract

In the field of palliative care there is a need to create adaptive questionnaires to minimize the patient's "cognitive load" when acquiring data of the patient's subjective experience of pain. A conversational case based reasoning (CCBR) system can be used as a basis for such questionnaires, and dialogue learning as a method for reducing the number of questions asked, without deterioration of the data quality. In this thesis, methods for question ranking, dialogue inferring, and dialogue learning have been reviewed. A case based reasoning framework is introduced and improved, and based on this, a CCBR system with an extension for dialogue learning has been designed and implemented. The result was tested with well known datasets, as well as new data from a survey on patients' experience of pain. Evaluation shows that dialogue learning can be used to reduce the number of questions asked, but also reveals some problems when it comes to automatically evaluation of solutions found using query biased similarity measures.

Preface

This master thesis is the completion of my master of informatics degree, and it was carried out at the Department of Computer and Information Science (IDI), at the Norwegian University of Science and Technology (NTNU).

I would like to thank my advisor, Agnar Aamodt, for guiding and motivating me, and answering all of my questions.

I would also like to thank Tor Gunnar Høst Houeland, Ida Kokkersvold, and Tor Henrik Aasness Marthinsen for interesting meetings and motivational support during the spring semester. The exchange of experiences has been very useful. Also Odd Erik Gundersen should be tanked for his work on the framework, and Frode Laugen for supplying and explaining relevant test data.

Trondheim 15. June 2007

Hans Arne Vartdal

Contents

1 Introduction **1**
- 1.1 Problem description . . . 1
- 1.2 Background and motivation . . . 2
- 1.3 Goal . . . 3
- 1.4 Research Approach . . . 3

2 Related Research **5**
- 2.1 Computer adaptive testing . . . 5
  - 2.1.1 Item response theory . . . 6
  - 2.1.2 Computer adaptive testing with IRT . . . 7
- 2.2 Case based reasoning . . . 7
- 2.3 Conversational case based reasoning . . . 8
  - 2.3.1 Similarity computation . . . 10
    - 2.3.1.1 Query-biased Similarity Calculation Methods 11
    - 2.3.1.2 Case-biased Similarity Calculation Methods . 12
    - 2.3.1.3 Equally-biased Similarity Calculation Methods 12
  - 2.3.2 Question ranking . . . 12
    - 2.3.2.1 Information gain metric . . . 12
    - 2.3.2.2 Occurrence frequency metric . . . 13
    - 2.3.2.3 Importance weight metric . . . 13
    - 2.3.2.4 Similarity variance . . . 13
  - 2.3.3 NaCoDAE . . . 14
    - 2.3.3.1 Case retrieval . . . 14
    - 2.3.3.2 Question ranking . . . 14
    - 2.3.3.3 Enhancing dialogue inferring . . . 15
    - 2.3.3.4 Model-based dialogue inferring . . . 15
- 2.4 Dialogue learning in CCBR . . . 16
  - 2.4.1 A framework to support dialogue learning in CCBR . 16
    - 2.4.1.1 Dialogue case base . . . 16
    - 2.4.1.2 Dialogue case retrieve . . . 17
    - 2.4.1.3 Dialogue case reuse . . . 18
    - 2.4.1.4 Dialogue case retain . . . 19
  - 2.4.2 Justification for the framework . . . 20

2.4.3 Trade off in dialogue learning 20

3 Framework: TCBR **21**
3.1 Introduction 21
3.2 The Representation 21
3.3 The functionality 24
3.3.1 Performance 24
3.3.2 Case retrieval 24
3.3.2.1 Object oriented case retrieval 25
3.3.3 Case weighting 26
3.3.3.1 Information gain for setting weights 26

4 Results **27**
4.1 Design 27
4.1.1 Conversational CBR system 28
4.1.1.1 Solution retrieval 29
4.1.1.2 Question ranking 30
4.1.2 Dialogue system 30
4.1.2.1 Representing a dialogue 32
4.1.2.2 Controlling a dialogue 32
4.1.2.3 Retrieving and reusing a dialogue 33
4.1.2.4 Retaining a dialogue 33
4.1.3 User simulation 34
4.1.3.1 Creating a problem description 34
4.1.3.2 Answering questions 35
4.1.3.3 Evaluating solutions 35
4.1.4 Test environment 35
4.2 Implementation 37
4.2.1 Global feature weighting 38
4.2.2 Automated agent 40
4.2.3 CCBR system 41
4.2.4 Dialogue system 45
4.2.5 Putting the system together 48
4.2.6 An example run 48

5 Evaluation **59**
5.1 Properties of the test environment 59
5.2 Datasets 60
5.2.1 Data from PAT-C 61
5.2.1.1 Pruning irrelevant data 61
5.2.1.2 Omitting incomplete or strange data 62
5.2.1.3 Resulting data 62
5.2.2 Data from UCI 64
5.2.2.1 Lung cancer 64

5.2.2.2 Soybean large 64
5.2.3 Case base generator 66
5.3 Test results 67
5.3.1 Lung cancer 68
5.3.2 Soybean large 69
5.3.2.1 Part 1 69
5.3.2.2 Part 2 69
5.3.2.3 Part 3 70
5.3.3 PAT-C 70

6 Discussion **73**
6.1 Conclusions 73
6.2 Theoretical and practical issues 74
6.2.1 Practical problem descriptions 74
6.2.2 Global and local approximation of the target function 74
6.2.3 The most general dialogue 75
6.2.4 Variance in the test results 75
6.2.5 Scalability 75
6.3 Future work 76

Bibliography **77**

A Analysis of the datasets **81**
A.1 The PAT-C dataset 81
A.2 The lung cancer dataset 98
A.3 The soybean large dataset 99

List of Figures

2.1 CBR problem solving process. 8
2.2 Conversational case based reasoning. 10
2.3 Model-based support for dialogue inferencing in CCBR. . . . 15
2.4 Framework to support dialogue learning in CCBR. 17

3.1 Simple class diagram for the TCBR framework. 22
3.2 TCBR graph representation. 23
3.3 The object oriented implementation of the case retriever. . . . 25

4.1 The main parts and concepts of the system design. 28
4.2 Design for the CCBR system. 29
4.3 Including the dialogue system. 31
4.4 Representation of a dialogue case. 32
4.5 An overview over the test environment. 37
4.6 Class diagram for the feature ranker. 38
4.7 Class diagram for the agent. 40
4.8 Class diagram for the CCBR engine. 42
4.9 Class diagram for the dialogue system. 45
4.10 Class diagram for the dialogue case retriever. 47
4.11 The internal communication sequence in the system. 50

5.1 Pat-C class distribution. 63
5.2 Lung cancer class distribution. 65
5.3 Soybean large class distribution. 65
5.4 Class diagram for the case base generator. 66

List of Tables

4.1 Pseudo code of the simplified EACH algortihm. 39
4.2 Pseudo code for the query. 43
4.3 Pseudo code of the evaluation process. 49
4.4 Start parameters for the program. 52
4.5 A data file example. 53

5.1 Case base generator example 67
5.2 Result table for lung cancer dataset. 68
5.3 Result table for part 1 of the soybean large dataset. 69
5.4 Result table for part 2 of the soybean large dataset. 70
5.5 Result table for part 3 of the soybean large dataset. 71
5.6 Result table for the PAT-C dataset. 72

Chapter 1

Introduction

This chapter includes the problem description, a description of the background, and the goals for this thesis. A description of the research methods used is also included.

1.1 Problem description

A conversational case based reasoning (CCBR) system contains a basic case based reasoning (CBR) subsystem and a dialogue management system. The dialogue learning part shall be focused in this thesis project. Learning in the dialogue part shall address the improvement of how question selection is done, with minimizing the patient's "cognitive load" as the primary test criterion. The learning that takes place may involve learning new cases, introducing new indexes to existing cases, or updating existing cases and indexes.

A part of the architecture shall be exemplified in an implemented demo. In a recent PhD research project methods for learning in CCBR were developed (Gu (2006)). They should be reviewed and analysed, and used as input to the choice of methods in this thesis.

The implemented system will be based on an existing core CBR system (throughout this thesis referred to as TCBR), implemented in C++. This thesis project is linked to the EU project European Palliative Care Research Collaborative (EPCRC) (EPCRC (2005)), in which IDI cooperates with the cancer research unit in Trondheim, and the company Trollhetta AS.

1.2 Background and motivation

The overall aim of the EPCRC is to radically improve the quality of life in a large number of patients with incurable cancer and short life expectancy by alleviating the three frequent symptoms pain, depression and fatigue. The program has four major objectives of which this thesis rests on the second:

To improve classification and assessment of pain, depression and cachexia by computer assisted approaches.

Computer-technology in assessment of pain has not fully utilized the potentials of state of the art computer science. The use of computers in general, for various types of clinical work, is spreading. The developments of electronic patient record systems are currently showing rapid progress. By applying computer adaptive testing (CAT) new measurement systems can be developed. CAT works similar to a trained clinician and the computer selects items from an item bank based upon the respondents' previous responses. By combining CAT with artificial intelligence (AI), and in particular CBR, one can develop a new direction of research through systematic capture and reuse of experience, and thereby invent "intelligent" computerised assessment tools for assessment of pain and other symptoms. Computer-technology also has potentials for development of new assessment methods that can replace complex measurement techniques which are difficult to perform in real life (i.e. in clinical practice).

This thesis will attend the challenge in how data acquisition from a patient, that is registering patient information, can be achieved with as little "cognitive load" for the patient as possible without deterioration of the quality of diagnoses found in the retrieval process. The current practise is to use paper-based questioning methods for this data acquisition. These paper-based forms may contain as many as 60 or more questions, where some always will be more relevant to one patient than another. To answer this large amount of questions is a tedious and strenuous operation for a patient. The patients have different grades of operability, and in the worst scenarios it might take hours to complete the questioning. In these cases the answers looses reliability as the patient will be more focused on finishing than considering each question. One might say that the questioning becomes a source of pain itself. A better and more effective way of describing a patient's condition will give the doctor a better decision basis for evaluating the condition as regards to diagnosis and treatment. To achieve this it is necessary to individually tailor the questioning session for each patient, asking the most relevant question at any step of the questioning, and deducting answers to other questions when possible. It is not desirable to ask further questions unless their answers discriminate between possible solutions.

CCBR is an interactive, dialogue oriented, method in CBR. The goal of systems using this method is to ask for as few attributes as possible but still be able to retrieve the best matching cases. A case is initially defined by the user by giving a few attributes to the system, which asks for more attributes in a way that discriminates between the previously stored cases in the best possible way. It seems obvious that such a CCBR system could be used to achieve a reduction in the number of questions a patient would have to answer to describe his condition in a sufficient manner.

1.3 Goal

The goal of this thesis is to extend the TCBR framework developed by Trollhetta AS for the EPCRC program by modeling and implementing a system for dialogue learning in CCBR based on TCBR.

To be more concrete this means:

- Review methods in CAT, IRT, CBR, and CCBR, especially for dialogue learning in a CCBR system.
- Specify a model for building a CCBR system with dialogue learning based on the TCBR framework.
- Implement a prototype of the system modeled.
- Test the prototype using appropriate data, e.g. from the EPCRC program.

1.4 Research Approach

This thesis is based on analytical and experimental methods. The concept of dialogue learning in CCBR systems will be analysed through a study in the literature concerning the basics of CBR and CCBR and learning in these, and more specific existing methods and implementations of dialogue learning. The alternative method IRT used in CAT is also reviewed. Based on this study, a CCBR system with dialogue learning for the TCBR framework will be designed. The most important parts of this system will then be implemented on top of the TCBR framework. Finally the system will be tested.

Chapter 2

Related Research

This chapter first gives a review of CAT and IRT and then introduces the basics of CBR, and CCBR. Some of the known methods for similarity measure and question ranking are reviewed, and a framework for dialogue learning in CCBR is introduced.

2.1 Computer adaptive testing

The EPCRC project strive for combining CAT with AI to develop a new direction of research through systematic capture and reuse of experience, and thereby invent "intelligent" computerised assessment tools. CBR is mentioned as a particular interesting AI technology, but the EPCRC project also wishes to examine current methods for applying item response theory (IRT) to dynamic questioning, and develop improvements as replacement or complement to IRT. Although the focus of this thesis will be in CBR, and more specifically CCBR, describing IRT and its use in CAT is needed to give a basis of comparison between these methods.

Researchers in the field of health outcomes and quality of life[1] have in recent years shown considerable interest in techniques for computerized adaptive assessments (Revicki and Cella (1997); McHorney and Colleen (1997)). However, few have yet established a functioning system for computerized adaptive assessment of health outcomes (one is Ware et al. (2000)), and the documentation of the theory and practice of computerized adaptive assessment in the health outcomes field is still scarce. Therefore there is very

[1] Health outcomes research is the measurement of the value of a particular course of therapy. Health outcomes research is based on the principle that every clinical intervention produces a change in the health status of a patient and that change can be measured.

good reason to learn from educational testing, where work on CAT has been carried out for many years.

Wainer (2000) lists three situations where continuous testing and CAT seem to be a good idea:

- When it is in the best interest for all to get the right answer (i.e. no incentive to cheat).
- When test results are needed all year round (i.e. no increased need for test results in connection with application deadlines).
- When the test is best administered by computer.

It is encouraging for health outcomes researchers that all three situations apply to their field. This is fairly evident for the first two types of situations. Examples of situations where the computer administration in itself is an advantage are when we need immediate feedback on assessment results or we need to monitor the consistency of responses (Ware et al. (2000); Bjorner and Jr. (1998)). Thus, the health outcomes field may be able to take advantage of the strengths of CAT without being hampered by the problems seen in educational testing, where there is both incentive to cheat for better grades, and seasonal variations.

2.1.1 Item response theory

The IRT is part of the principles underlying test development within applied psychology and psychological testing, called psychometrics. Psychometrics is concerned with the theory and technique of educational and psychological measurement. The field is primarily concerned with the study of differences between individuals and between groups of individuals, and includes the measurement of knowledge, abilities, attitudes, and personality traits. Classical test theory (CTT) has served well as the psychometric basis of test development over several decades, but IRT has rapidly become mainstream as the theoretical basis for measurement.

In IRT mathematical models are applied to analyse data from questionnaires and tests. The models are mathematical functions that specify the probability of a discrete outcome, such as a correct response to an item, in terms of person and item parameters. Person parameters may represent the ability of a student, or the strength of a person's attitude. Items may be questions that have incorrect and correct responses, or statements that allow respondents to indicate a level of agreement.

2.1.2 Computer adaptive testing with IRT

In CAT, examinees receive questions that are optimally selected to measure their potential. Different examinees do not necessarily receive common questions. IRT principles are involved in both selecting the most appropriate questions for an examinee, and equating scores across different subsets of questions. Examples of tests that apply IRT to estimate abilities are the Scholastic Aptitude Test (SAT), and the Graduate Record Examination (GRE).

2.2 Case based reasoning

Case based reasoning is a type of analogical problem solving and lazy machine learning method. A case is an instance of a previous problem that has been solved by the method, and these are stored in a case base. A new problem is represented as a new case, either complete or with missing features, and is solved by applying the solution adapted from that of the most similar case (or cases) stored in the case base. Cases stored in the case base are always "positive" examples, and applying the case solution to the case problem is assumed to be successful (Aha et al. (2001)). A case serves as a prototype for solving queries whose problem specifications are similar to the one of the case. Queries that are similar to a case's problem are expected, with high probability, to benefit from its solution.

The cases can be built in several ways but in general the content of a case should include the following three parts (Kolodner (1993)):

- Problem description: the state of the world at the time of the case, and, if appropriate, what problem needed to be solved at that time.
- Solution description: the stated or derived solution to the problem specified in the problem description.
- Outcome: the resulting state of the world after the solution was carried out.

The problem description is typically represented as a set of feature-value pairs, {f,v}, or sometimes triplets also including the importance of the feature as a weight, {f,v,w}. The solution description may be represented in many ways, and as later explained, a special dialogue case might even have a reference to another case as its solution.

As summarized by Aamodt and Plaza (1994), the problem solving process in CBR contains four steps: retrieve, reuse, revise, and retain (Figure 2.1 from Aamodt and Plaza (1994)).

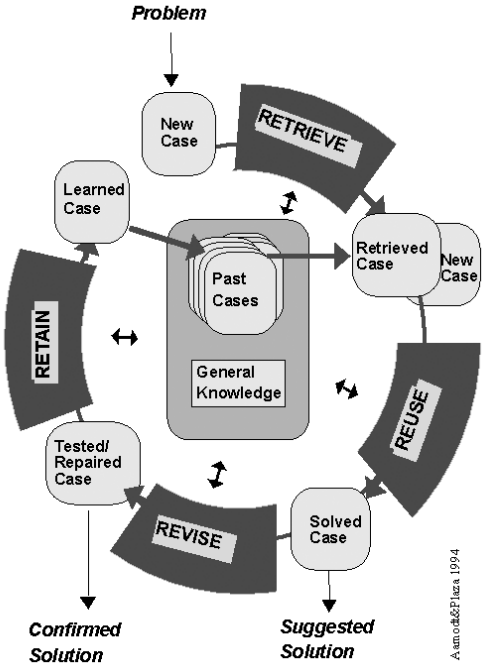


Figure 2.1: CBR problem solving process.

A CBR system transforms a given problem description into the problem description part of a new case. This new case is used to retrieve the case/cases which match this description best, based on similarity calculations between the new case and the cases in the case base. The retrieved case is then reused by applying the solution description and outcome description to the creation of a solution description for the new case, giving the most probable solution given the cases in the case base. This solution is evaluated in some environment, which may be used to revise the solution. Finally we have a new, complete case which can be retained in the case base for future use.

2.3 Conversational case based reasoning

It is not always easy for the user to define a complete problem description; in fact there often is a knowledge gap between the user of the case base and the author. The user may not have the expertise necessary for representing the problem description, such as what terms to use. As an example a patient asked to describe pain, which is very hard to describe, would need specific questions as terms of how to describe it. Another reason for incomplete problem description is that the cost of acquiring a complete description may

be too high. This is the case for the patients; a too high cost may result in the patient not being able to complete the description with reliable data, or not completing it at all.

As a solution for these problems conversational case-cased reasoning (CCBR) provides a mixed-initiative dialogue for guiding users to refine their problem descriptions incrementally through a question-answering sequence. The user may describe the problem step by step starting only with a small description including few features. The choice of the next step is guided by the CCBR system, and the cost required acquiring a sufficient description of the case is minimized. Because the user need only answer posed questions, a priori knowledge concerning their relevance is not needed. It is obvious that a CCBR system could be used to achieve a reduction in the number of questions a patient would have to answer to describe his condition in a sufficient manner. Since there is no point in answering more questions if they can not discriminate further between the diagnoses in the case base, one can hope that only relevant questions will be asked.

Aha et al. (2001) specify a generic form for a case C in a CCBR system as follows:

1. Problem Cp = Cd + Cqa: Encodes the problem solved by Cs.
   (a) Description Cd : Free text that partially describes C's problem.
   (b) Specification Cqa: A set of <question,answer> pairs.
2. Solution Cs = {Ca1, Ca2, ...}: A sequence of actions Cai for responding to Cp.

The description given as a free text requires some text recognition processor which transforms the text into a subset of the <question,answer> pairs in the specification. For CCBR systems in general the problem description might be given by other means, e.g. as <question,answer> pairs directly. These <question,answer> pairs can be treated as <feature,value> pairs, where during a dialogue process features can be transformed into user readable questions, and the values contain the answers of these questions. Aha et al. (2001) suggests that actions can be free text, hyperlinks, or other objects, but the use of a solution represented as a series of actions becomes very clear in an example from the health domain. A successful conversation between a patient and a medical doctor would often end with a diagnosis, or a treatment.

Figure 2.2 shows the CCBR reasoning process as it is described by Gu (2006), starting by transforming the user's initial problem description into an initial new case. This case may only contain a few features, and is used to retrieve a set of the most similar stored cases from the case base. The returned cases are then used as the basis for question generation and ranking, based on the

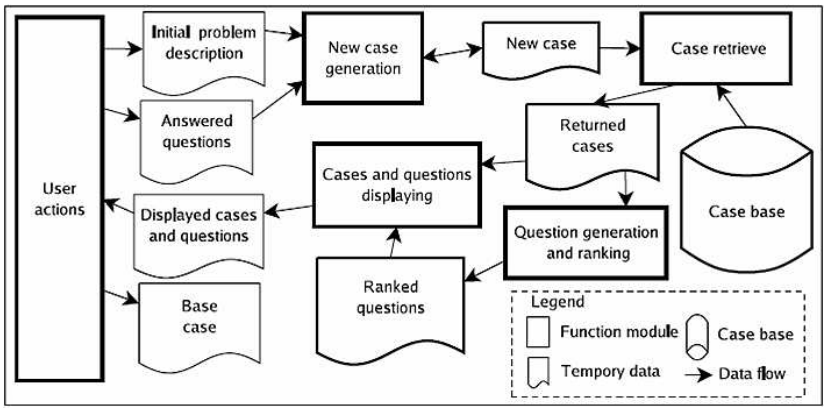


Figure 2.2: Conversational case based reasoning.

features of the cases. A group of discriminating questions are identified, and ranked according to their capability to discriminate the stored cases. The ranked questions and the solutions of the returned cases, sorted according to their similarity values, are displayed to the user. The user then either selects a satisfactory solution, or chooses a question to answer. The answered question is added to the problem description of the initial new case and the process starts again, continuing until a satisfactory solution is given or there are no more discriminating questions to left to choose.

2.3.1 Similarity computation

A key research topic in both CBR and CCBR is to calculate the similarities between a query, which is a new case, and stored cases to find which case is most similar to the new problem. The similarity between a query and a stored case is normally measured by accumulated similarities on the features involved. Gu et al. (2005) speaks of two ways to influence these similarity calculations:

1. Different methods to calculate the similarity on each feature; i.e. a nominal feature can be considered equal to another nominal feature if their values are syntactical equal (Aha (1991)), or there may be more sophisticated methods involving knowledge-intensive methods (Aamodt (1994, 2004)) where the value need only be considered equal through exploring some general domain knowledge.

2. The feature scope; what features are counted, the set of features ap-

pearing in the query, in the case, or in both of them?

This thesis will not address knowledge-intensive methods any further, but Gu et al. (2005) have shown that the choice of feature scope is important for an effective CCBR process.

The query in a CCBR process only consists of the problem description part of a case. As a consequence the similarity calculation process only takes the problem description part of the cases into consideration. Gu et al. (2005) define a framework for calculating such similarities, and then define three different methods in terms of this framework. They introduce N_q as the set of features appearing in the query, and N_c as the set of features appearing in the stored case. Further they introduce *distance* as an inverse relation to similarity, and define it as follows:

$$distance(q,c) = \sqrt{\frac{\sum_{f \in FS} w_f dif^2(q_f, c_f)}{\sum_{f \in FS} w_f}} \tag{2.1}$$

where q, c, f, FS, and w_f denote the query, the stored case, a particular feature, a selected feature set, and the weight for the feature f respectively. The function $dif(q_f, c_f)$ is used to compute the difference between the query and the stored case on a feature f, and is defined as follows:

$$dif(q_f, c_f) \quad = \quad \begin{cases} |q_f - c_f| & f\, is\, numeric\, (normalized) \\ 0 & f\, is\, nominal\, and\, q_f = c_f \\ 1 & f\, is\, nominal,\, and\, q_f \neq c_f \\ 1 & c\, or\, q\, has\, missing\, value\, on\, f \end{cases} \tag{2.2}$$

It is the content of FS which influence the similarity calculations in either a case-biased, query-biased, or equally-biased way.

2.3.1.1 Query-biased Similarity Calculation Methods

In such a method only the features appearing in the query are taken into account; $FS = N_q$. The idea behind this is that the query is assumed incomplete only representing the user's currently identified features, but the features without specific values do not necessarily have a "missing-value", the user has just not assign a value for them yet. The query-biased similarity calculation method avoids influence from these features, and ranks the case that most satisfy the currently partially specified query highest. This type of methods has been found to be more effective for CCBR systems (Gu et al. (2005)), and has been used in Gu and Aamodt (2006a).

2.3.1.2 Case-biased Similarity Calculation Methods

The features appearing in the stored case is the basis for this similarity calculation method; $FS = N_c$. The solution of the case is best satisfied by a query matching the problem description of the case, whether the solution in the stored case is suitable for the current problem is decided by the degree that the query satisfy the problem description. This type of methods has been used in Aha et al. (2001); Aha (1991).

2.3.1.3 Equally-biased Similarity Calculation Methods

Using both the features appearing in the stored case and the features appearing in the query, gives an equally-biased similarity calculation method; $FS = N_q \cup N_c$. The idea here is that similarity between the stored case and the query must take all features into consideration, to find the degree in which the solution can be reused for the current problem. This type of methods has been used in Richter and Wess (1993); Yang and Wu (2001).

2.3.2 Question ranking

Another major research topic in CCBR is how to alleviate the cognitive load demanded from users in the question-answering process. The two major ways to realize this task is dialogue inferring and question ranking. Dialogue inferring is to remove the questions whose answers can be inferred from the information the user has provided. Question ranking is to ensure that the more discriminative questions are asked at earlier stages, resulting in the shortest conversation. The main question ranking metrics proposed include information gain, occurrence frequency, importance weight, and similarity variance.

2.3.2.1 Information gain metric

Quinlan (1986) uses information gain, and it is commonly used in the fields of information theory and machine learning. For example information gain is used to choose the best feature for "splitting" decision trees. Based on the entropy, information gain measures how well the feature separates the cases according to their solutions. A very discriminative feature gives expected entropy close to 0 and an information gain close to 1 after using the feature. Questions whose corresponding feature has higher information gain are ranked higher in the question ranking. In CCBR information gains is dynamically calculated in each question ranking session at runtime. Also a group of the most informative features are selected. This distinguishes

from the decision tree learning where information gains are calculated statically at the tree construction stage forcing users to follow a static dialogue, which might cause the effect that a dialogue is terminated to early with no result.

2.3.2.2 Occurrence frequency metric

In an occurrence frequency metric the identified discriminative questions are ranked according to their frequencies of appearance in the problem description of the returned cases. The question gets a higher ranking priority if the corresponding feature is assigned a value in a larger number of the returned cases. This metric was used in the NaCoDAE system developed by Aha et al. (2001), which is reviewed in this thesis (Section 2.3.3). For this metric to perform well it is assumed that the cases in the case base are highly heterogeneous; a feature appearing in some cases may not appear often in other cases.

2.3.2.3 Importance weight metric

When calculating the similarity between cases using the Euclidean distance between them, as in the k-nearest neighbor algorithm, a problem appears if only a few of the features of the cases are relevant. Mitchell (1997) refers to this difficulty, where the distance will be dominated by the large number of irrelevant features and put the cases far from each other even though they should be close, as the curse of dimensionality. Feature weighting methods can alleviate this problem by assigning features with importance weights according to their contributions.

The importance weight based question ranking metric ranks the discriminative questions according to the weight value of the corresponding feature. The most relevant or important features provides more information than other features to discriminate cases from each other. EACH (Salzberg (1991)) and Relief (Kira and Rendell (1992)) are two such feature weighting methods.

2.3.2.4 Similarity variance

Schmitt (2002) proposed a similarity-influenced information measure (SimVar). Each case retrieval method is based on the similarity values between stored cases and a new case, and the goal state of the retrieval is to identify the correct case which has enough similarity variance from the rest of the

stored cases. A feature which can provide a higher similarity variance in the candidate cases gives a higher priority for the corresponding question.

2.3.3 NaCoDAE

Navy Conversational Decision Aids Environment (NaCoDAE) (Aha and Breslow (1997b)) was originally developed to test a strategy for simplifying case authoring (Aha and Breslow (1997a)), but its extensions for dialogue inferencing (Aha et al. (1998)) was later evaluated. NaCoDAE is described in Aha et al. (2001), and the system embodies a generic CCBR tool. The system receives a textual problem description from the user and generates an initial problem description in the system. This problem description is compared with cases in a case library resulting in a subset of ranked solution cases which serves as a basis for finding a set of ranked questions. Both the ranked solution cases and the ranked questions are displayed, and the user selects the most relevant question for answering, or a solution. Answers to questions add to the case, and the process is repeated until a satisfying solution is displayed and selected.

2.3.3.1 Case retrieval

After the textual problem description is translated into <question,answer> pairs, and any other questions answered by the user are added , the cases are ranked using a simple scoring function comparing the <question,answer> pairs Q with each case C:

$$score(Q, C) = same(Q_{qa}, C_{qa}) - diff(Q_{qa}, C_{qa})/|C_{qa}| \qquad (2.3)$$

The function $same(Q_{qa}, C_{qa})$ finds the number of shared <question,answer> pairs, and $diff(Q_{qa}, C_{qa})$ the number of conflicting. $|C_{qa}|$ being the total number of <question,answer> pairs in the case. The best k cases are displayed for the user, and used in the question ranking.

2.3.3.2 Question ranking

In NaCoDAE question ranking uses an occurrence frequency metric. Questions are ranked according to their frequency in specifications of the cases whose solutions are displayed to the user. It was also experimented with information gain metrics (in addition to frequency), but it was found that frequency was sufficient for the needs.

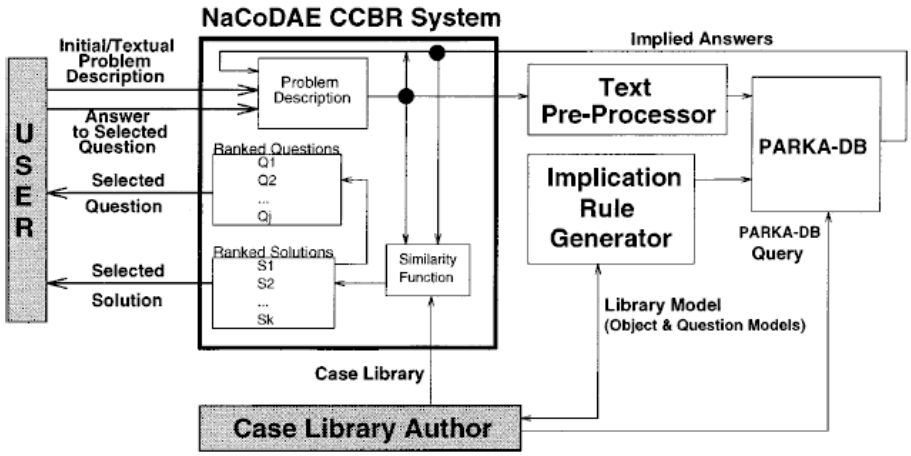


Figure 2.3: Model-based support for dialogue inferencing in CCBR.

2.3.3.3 Enhancing dialogue inferring

As a mean to avoid that users might be prompted with questions that could be automatically answered (e.g. through inferring), Aha et al. (1998) stated that CCBR tools should automatically infer answers to questions from the user's inputs whenever possible during a conversation. In NaCoDAE two types of inferences was used, text inferences and chaining rules. Text inferences relate text in the user's initial text to specific <question,answer> pairs, while chaining rules define relations among these pairs. Though some commercial CCBR tools attempt to solve this by requiring the case base designers to manually enter implication rules, this solution presents significant knowledge engineering challenges. The rule sets often get large and incomprehensive, and are difficult to maintain, resulting in CCBR case base designers avoiding the use of rules. To avoid this in NaCoDAE a model-based dialogue inferring approach was used.

2.3.3.4 Model-based dialogue inferring

Instead of constructing a rule set, the case base designer interactively enters a case base model. This model is composed of an object model which relates domain objects, and a question model which relates questions to these objects. Figure 2.3 shows how NaCoDAE was integrated with PARKA-DB, a high-performance knowledge representation system for processing such relational queries (Hendler et al. (1996)). These models are represented as semantic networks, and will be more compact than the corresponding rule set for many tasks, thus be more comprehensible and easier to maintain. The implication rule generator inputs these models and outputs a set of

rules which are input to PARKA-DB together with the problem description. PARKA-DB retrieves all answers implied by the previously answered questions, and the user's text, and adds these inferred answers to the problem description.

2.4 Dialogue learning in CCBR

The reason for learning dialogues in CCBR is the claim that successful dialogues can be captured and learned in order to improve the efficiency of CCBR from the perspective of shortening the dialogue length. Such successful dialogues that have occurred in a CCBR system can be seen as previous solutions to users' case retrieval tasks, and retained as cases themselves (Gu and Aamodt (2006a)).

2.4.1 A framework to support dialogue learning in CCBR

Gu and Aamodt (2006a) presents a framework for dialogue learning in CCBR, and how this framework can be implemented. The evaluation of the implementation gave significant evidence to support their hypotheses that the dialogue learning mechanism is effective and sustainable, and that the dialogue case base is maintainable.

To avoid confusion Gu and Aamodt (2006a) introduces application as a prefix for the concepts in the main CCBR framework, such as an application case and an application case base. As illustrated in Figure 2.4 (from Gu and Aamodt (2006a)) the CCBR framework is extended by the new concepts: dialogue cases, dialogue case base, retrieve, reuse, and retain methods for dialogue cases. The following sections will describe how Gu and Aamodt (2006a) defined and implemented these concepts.

2.4.1.1 Dialogue case base

A dialogue process is composed of the initial input from the user (a new case), the later incrementally selected questions, and their answers. This process is stored as the problem description of a *dialogue case*. The solution description of such a case refers to the application case successfully retrieved from the application case base as a result of the dialogue. A dialogue case is always terminated when the user retrieves a satisfactory application case and the dialogue process ends.

Gu and Aamodt (2006a) defines a dialogue case (dc) as

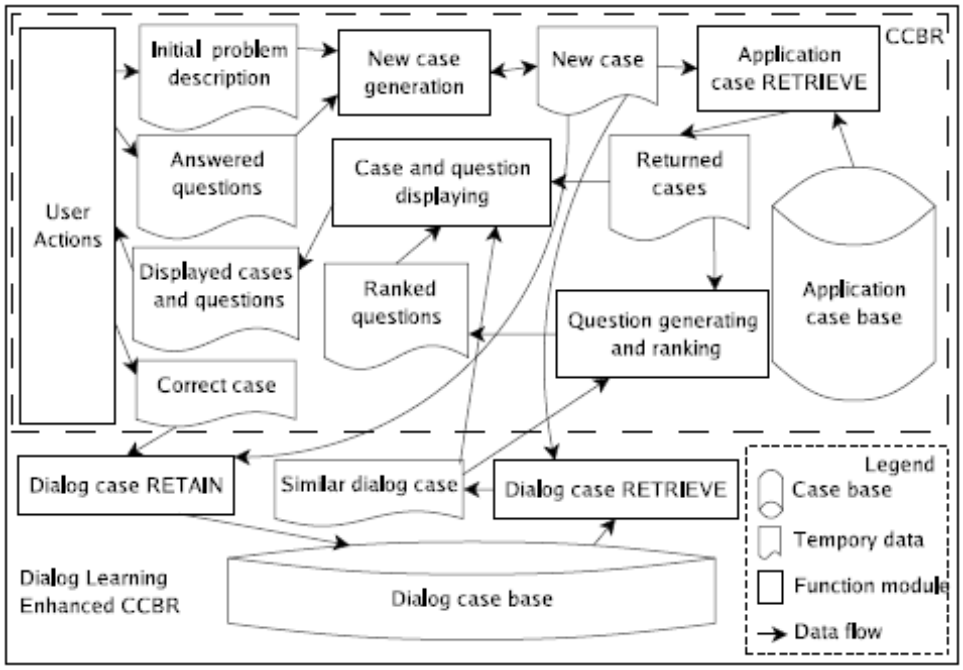


Figure 2.4: Framework to support dialogue learning in CCBR.

$$dc :< \{fvwp\}, ds > \tag{2.4}$$

The dialogue case in addition to the dialogue solution (ds), referring to the retrieved application case following the dialogue process, consist of a set of four-item vectors ($\{fvwp\}$) describing the problem description of the dialogue case dc. This vector is similar to that introduced in Section 2.2; f denoting the feature name, v the feature value, and w the importance weight for the feature f. The difference is that a new variable p, an integer value that expresses the appearance position of feature f in the dialogue process, is added to the vector.

2.4.1.2 Dialogue case retrieve

To retrieve a case from the application case base, in CCBR a new case describing an application problem is incrementally constructed. This new case is also used to retrieve a dialogue case from the dialogue case base. The similarity assessment used in dialogue case retrieval not only takes the features used in application case retrieval into account, but also the feature sequencing information. Gu and Aamodt (2006a) describes the positions of various features in a dialogue process as an expression of the change in a

users' focus of attention, which influence the similarity between the new case and a stored dialogue case.

Gu and Aamodt (2006a) used a 1-NN algorithm to retrieve the most similar dialogue case, based on this definition of the distance equation between a new dialogue case, *dn*, and a stored dialogue case, *dc*, as follows:

$$distance(dn, dc) = \sqrt{\frac{\sum_{f\in\{f\}} w_f posw(dn_f, dc_f) dif^2(dn_f, dc_f)}{\sum_{f\in\{f\}} w_f}} \tag{2.5}$$

where *{f}* denote a selected feature set, and w_f the importance weight for the feature *f*. $dif(dn_f, dc_f)$ is a function used to compute the difference between the new dialogue case, *dn*, and the stored dialogue case, *dc*, on a feature f in (2.5), and is defined as follows:

$$dif(dn_f, dc_f) = \begin{cases} |dn_f - dc_f| & f\, is\, numeric\, (normalized) \\ max\{dn_f, 1 - dn_f\} & f\, ia\, numeric\, (normalized) \\ & and\, dc_f\, is\, missing \\ 0 & f\, is\, nominal\, and\, dn_f = dc_f \\ 1 & f\, is\, nominal,\, and\, dn_f \neq dc_f \\ & or\, dc_f\, is\, missing \end{cases} \tag{2.6}$$

This follows the query-biased similarity calculation method (Gu et al. (2005)), only taking the features appearing in the query (new case) into account during similarity computation, thus *{f}* is assigned all the features appearing in the new dialogue case. The function $posw(dn_f, dc_f)$ is used to compute the weight concerning the appearance position of feature f in the new dialogue case, *dn*, and the stored dialogue case, *dc*:

$$posw(dnf, dcf) = \frac{1}{2} + \frac{1}{2} * \left(1 - \frac{|p(dn, f) - p(dc, f)|}{max(length(dn), length(dc))}\right) \tag{2.7}$$

where *p(dn,f)*, *p(dc,f)*, *length(dn)*, and *length(dc)* denote the position where feature f appear in the new dialogue case, *dn*, and in the stored dialogue case, *dc*, and the length of these cases (referring to the number of features they have).

2.4.1.3 Dialogue case reuse

Gu and Aamodt (2006a) mention two ways in which the retrieved most similar dialogue case is considered to be able to improve the efficiency of the CCBR retrieval process:

- The solution application case of the most similar dialogue case will be displayed to the user, along with the application cases found by the CCBR retrieval process.
- Features appearing in the most similar dialogue case, not appearing in the current new application case, will get improved ranking priority.

Thus both the case and question displaying module, and the question generating and ranking module in standard CCBR can be influenced (as shown in Figure 2.4). In the implementation in Gu and Aamodt (2006a) both these tasks are solved. First, if the application case acting as the solution in the most similar dialogue case is not included in the k most similar application cases retrieved by the CCBR process, it is used to replace the least similar of these k application cases. Second, an equation for adjusting the weights of the features found as question candidates that also appear in the most similar dialogue case is defined as follows:

$$wf = wf + \left(\frac{1}{2} + \frac{1}{2} * \left(1 - \frac{p(dc, f)}{length(dc)}\right)\right)\left(\frac{1}{|total\ feature\ set|}\right) \quad (2.8)$$

where */total feature set/* is the number of features appearing in the application case base. Increasing the weights of the candidate features also appearing in the retrieved most similar dialogue case, will rank the discriminative questions transferred from these features with higher priority.

2.4.1.4 Dialogue case retain

New dialogue cases are generated as the CCBR process completes more and more successful dialogues. Retaining all these dialogue cases in the dialogue case base would make the case base grow rapidly, thus a strategy for maintaining the dialogue case base during run time is needed. It should improve its capability without expanding too much. Gu and Aamodt (2006a) uses a dialogue learning strategy which only stores the most general dialogue cases in the case base. They define the relation, more general than ($\gg$), between two dialogue cases as:

$$< \{fvwp\}_1, ds_1 > \gg < \{fvwp\}_2, ds_2 >: \\ ds_1 = ds_2\, and\, \{fvwp\}_1 \subseteq \{fvwp\}_2 \quad (2.9)$$

2.4.2 Justification for the framework

To test the framework Gu and Aamodt (2006a) designed an experiment which evaluated the effectiveness of the dialogue learning mechanism from the perpective of using fewer dialogue session to find the correct stored case. They use a leave-one-out cross validation (LOOCV) method to simulate the human-computer dialogue process. Similar methods have also been successfully used by others in the CCBR community (Aha et al. (1998); Gu et al. (2005)). The LOOCV method proceeds with a series of simulated dialogues, each dialogue starting with selecting an application case from the application case base as the target case, leaving the remaining application cases to be searched. A new case is built using some of the features from the selected application case. The CCBR process then run until a correct solution is found. When the CCBR system asks for a new feature, it is found in the preselected application case.

The authors identify three hypothesis that they test (see Gu and Aamodt (2006a) for detailed descriptions):

H1: the dialogue learning mechanism is effective.

H2: the dialogue learning mechanism is sustainable.

H3: the dialogue case base is maintainable.

Using 32 datasets from the UCI repository (D.J. Newman and Merz (1998)) in the experiment, and verifing the significanse of the experiment results through hypothesis testing, they find that all three hypotesis are accepted. They point out that a long term real human-subject based experiment would give more solid evidence to the hypotheses. There is also a question about trade off using dialogue learning.

2.4.3 Trade off in dialogue learning

As Gu and Aamodt (2006a) points out, there exists a trade off between the dialogue efficiency improvement and the resource cost. Even though the experiment only ran two cycles for each dataset, the dialogue case base size compared to the application case base size, which demands a considerable memory space and CPU time to retrieve inside. This must be taken into consideration before adopting this dialogue learning mechanism in a practical CCBR application.

The trade off is closely connected to the choice of dialogue learning strategy; retaining more dialogue cases than only the most general ones would give even larger dialogue case bases than in the experiment where only the most general dilogue cases were stored.

Chapter 3

Framework: TCBR

In this chapter the framework, on which the modelling and implementations in this thesis is based on, is described. The representation and functionality of the framework, and some useful changes and extensions to the framework are presented.

3.1 Introduction

The modeling and implementations in this thesis will be based on the Trollhetta CBR (TCBR) framework developed by Trollhetta AS for the EPCRC program. This framework is a basic CBR system with some simple but useful properties. The system can be seen as two layers, where the first layer is a semantic net representing a knowledge model, containing entities and relations (see Figure 3.1). The second layer includes the case base, containing cases and features, which are all represented through the first layer. The representation includes a lot of metadata, both about the representation itself, and the concepts used in the second layer.

3.2 The Representation

Entities are represented as nodes in a graph such as in Figure 3.2. Each entity has a name, and must belong to a knowledge model. These entities can have several relations. A relation is a connection between two entities, and must have both a source entity and a target entity. The relation also has a type, and strength. All entities and relations belong to a knowledge model. This model has a name, and contains control functions for the entities and

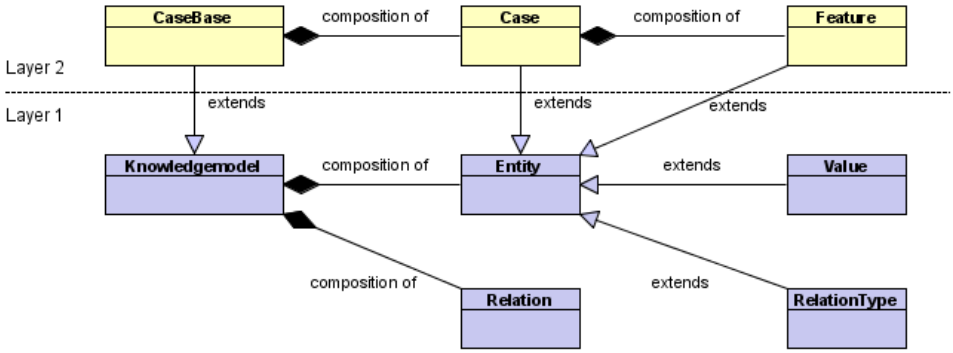


Figure 3.1: Simple class diagram for the TCBR framework.

relations. One may ask; does this entity exist? Or; what are the relations of this entity?

As the system includes metadata in the representation, the first entity is called ENTITY, and all other entities have a relation to this entity of type *instance_of*. These relations are unidirectional, but the framework also creates a corresponding *has_instance* relation which let us traverse in the graph in both directions. The concept of a case is thus represented by creating an entity called CASE with a relation to the entity called ENTITY of type *instance_of*. This is still on a metadata level, as an actual instance of a case is represented by an entity with a relation to the CASE entity of type *instance_of*. These entities can be named in any way convenient, but must be unique within the given knowledge model.

Features of a case are also represented in a similar way. All instances of features are connected to an entity called FEATURE with an instance_of relation, and this entity is connected to the ENTITY entity. Features and cases are connected with a *has_finding* relation from a case instance entity to a feature instance entity. The features have both a type and a value. The type is represented as an entity connected to the feature instance with a *has_type* relation, and to a TYPE entity with an *instance_of* relation. Several cases might have features of the same type e.g. “name”. The value of this feature extends the graph even another level through a *has_value* relation to a value instance entity, which has the relations *instance_of* to the VALUE entity, *has_type* to a entity describing the type of the value e.g. TEXT, and *has_value* to a entity holding the actual value e.g. “Roger”.

This very general way of representation has both advantages and disadvantages. The use of instance entities for the actual cases and features give us the possibility to set the weight between a case and a feature using the strength of their relation, this being set independent of all other cases having the same type of feature. This weight is important when finding the next

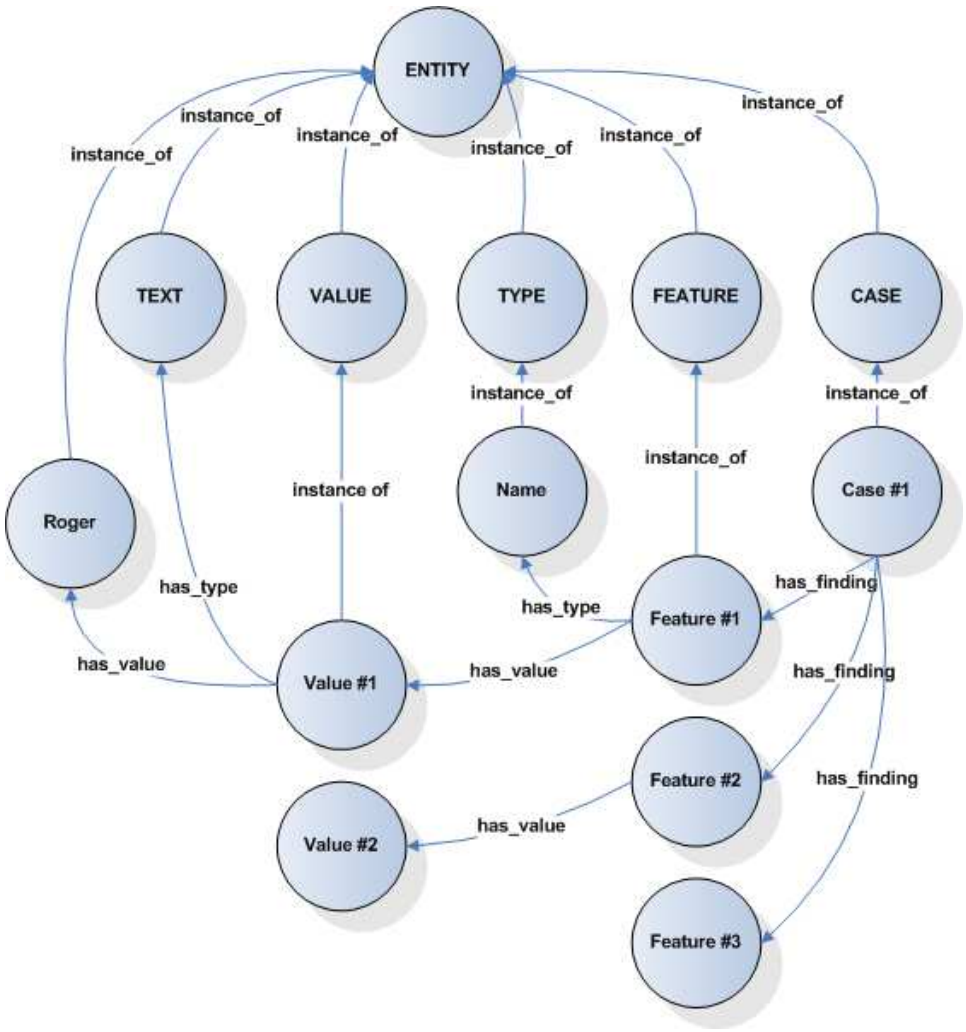


Figure 3.2: TCBR graph representation.

question to ask in a CCBR system. Even though the features are represented as different instance entities they use the same instance of the type entity, which allows us to list e.g. all "name" features in the case base by accessing the relations of the type instance and finding the target of all *type_of* relations (corresponding to the *has_type* relation). Methods like this prove useful e.g. when you need to find the number of different features in the case base, but require working with entities and relations in the first layer of TCBR.

3.3 The functionality

The TCBR framework includes only the most basic methods needed in a CBR system, and as it is the first version of the framework it has some problems. The following will include a basic description of some issues with the framework, the methods needed for implementing a CCBR system with dialogue learning as they are implemented in TCBR, and an overview of extensions to the framework that has been implemented in other thesis's using it.

3.3.1 Performance

The first version TCBR framework suffers from slow code and is extremely computationally demanding; simple case creation takes hours for small case bases. Due to memory leakage the framework also fail after some amount of time, rendering it useless in its original form. In a project thesis by Houeland (2007), also using the framework, the framework was revised and improved resulting in a stable and more computational efficient version of the framework. This version of the framework was made available for the other thesis's using the TCBR framework.

3.3.2 Case retrieval

The implementation of case retrieval in TCBR is fairly basic. The method is located in the case base, taking a new case as its input and returning a vector of the most similar cases paired with their similarity:

```
vector<pair<Case*, double> >* mostSimilarCases;
mostSimilarCases = cb.retrieveSimilarCases(newCase);
```

The way the method finds these most similar cases is by finding all cases with one or more features common with the new case, and then computing the distance between these cases and the new case in the Euclidian space, using a k-NN algorithm with k=1. The distance is increased by one for every feature in the new case which the other case does not have, and for every feature they both have but where the value is unequal. All cases with the lowest distance are included in the result set. Because the search for cases is driven by what features the new case has rather than the cases in the case base, we have a query-biased similarity calculation method, shown to be the best method for CCBR (Gu et al. (2005)).

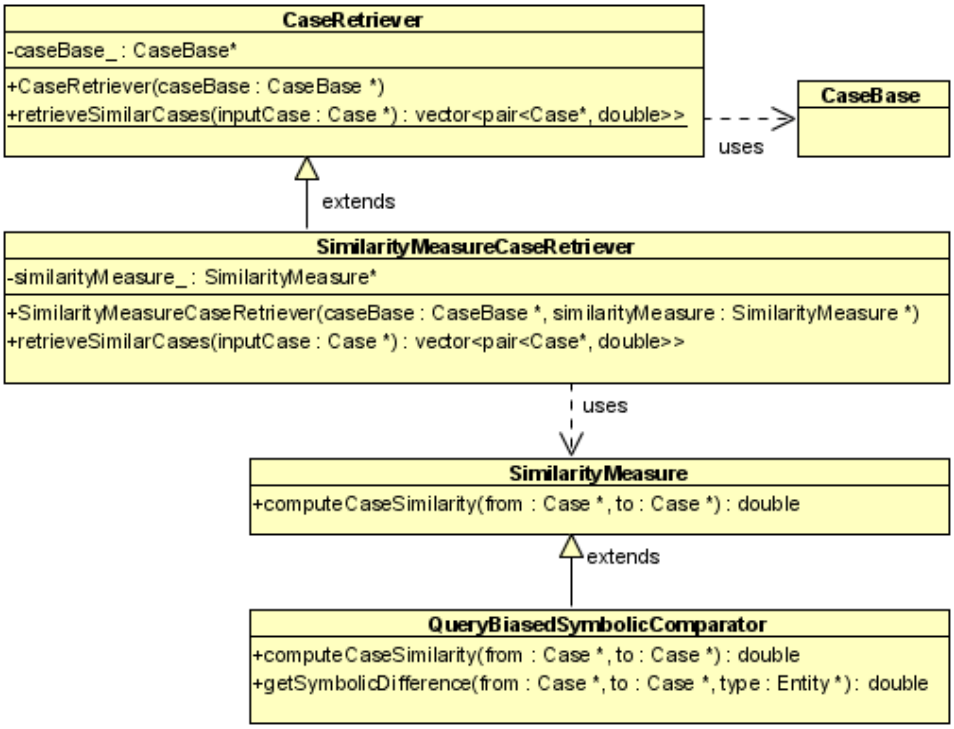


Figure 3.3: The object oriented implementation of the case retriever.

3.3.2.1 Object oriented case retrieval

In the project thesis by Houeland (2007) the basic retrieval method of the TCBR framework was rewritten to fit a more object oriented view. The concepts of a case retriever and a similarity measurer were introduced (see Figure 3.3). Rather than retrieving the most similar cases directly from a method in the case base, a similarity measurer is defined and together with a case base used to create a case retriever:

```
SimilarityMeasure similarityMeasure;
CaseRetriever caseRetriever(caseBase, similarityMeasure);
vector<pair<Case*, double> > mostSimilarCases;
mostSimilarCases = caseRetriever.retrieveSimilarCases(newCase);
```

This enables different means of retrieval to be implemented, with different similarity measures, without changing the representation of the case base; making it possible to implement different case retrievers for application cases and dialogue cases.

3.3.3 Case weighting

A case in the second layer of TCBR system does not have a built in method for setting the importance of a feature. Such weights can be important for the question selection mechanism, if one wishes to use an importance weight metric, and should be implemented. Two approaches are possible, a local approach and a global approach.

As mentioned in the representation description this functionality already exists in the first layer as the strength of a relation between a case instance entity and a feature instance entity. In the TCBR system these strengths are left out in the second layer, and defaulted to "1.0". A simple way to implement local weights is thus to extended the implementation of a case with methods for setting, and getting, this strength as its weight.

The global approach to weighting introduces the relation *has_weight* for features in the knowledge model, enabling to set and get a weight common to all instances of a feature type. This approach was implemented in the project thesis by Houeland (2007), and made available for other users of the TCBR framework.

3.3.3.1 Information gain for setting weights

In a diploma thesis by Kokkersvold (2007), also based on the TCBR framework, a version of the information gain metric for feature ranking was implemented. In this context a method for finding all values of a given feature was also added to the case base representation. The metric was made available for testing and use for the problem in this thesis.

Chapter 4

Results

This chapter consist of two parts; design, and implementation. First, a choice of design for a CCBR system with dialogue learning, and an environment for testing this system, is presented. Second, a description of the implementation of this system and its use is given.

4.1 Design

As a sub-goal for this thesis a model for building a CCBR system with dialogue learning based on the TCBR framework is to be specified. Such a system could as shown in Figure 4.1 profitably be designed as two separate parts; a dialogue system, controlling the course of the dialogues and methods for reusing and retaining these, and a basic CCBR system conducting the conversation by generating questions and retrieving possible solutions. A major advantage of designing the system as two separate parts is the ability to do ablation testing not including the dialogue system, thus be able to evaluate the impact of the dialogue system on the results. Ablation testing was for instance used in Gu and Aamodt (2006a). Both the CCBR system and the dialogue system will be using the TCBR framework as a basis for their internal methods.

A party that can be regarded as a third part of the system is the user. The tasks of the user are to initiate the conversation with a problem description, answering the questions asked, and considering the solutions presented by the CCBR system. This interaction could take place through a graphical user interface with a human user or through an interface with a software agent, the last being particular interesting during testing of the system. Clear interfaces between the two main parts, the CCBR system and the dialogue system, and between the user and the CCBR system, is therefore

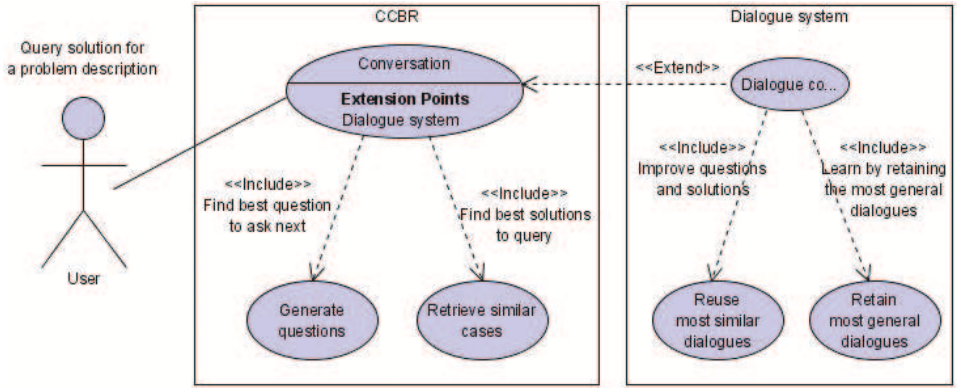


Figure 4.1: The main parts and concepts of the system design.

important.

The rest of the design section goes into detail for each of these parts of the system; first the CCBR system, second the dialogue system, then a design for a user agent is described, and lastly an environment for testing the complete system is defined.

4.1.1 Conversational CBR system

The CCBR system, as mentioned, is a basic system conducting the conversation by generating questions, retrieving possible solutions, and interacting with the user based on these. As shown in Figure 4.2 the process is similar to the one described in Figure 2.2, but with some differences:

- A new query case is generated from the problem description, but as questions are answered the query case is updated, not regenerated.
- Questions are generated on basis of features returned from the case base, not from the retrieved cases. The reason for this is explained in Section 4.1.1.2.
- Solutions are only presented for consideration if their similarity exceeds a threshold value.
- Questions are generated when either no solutions are presented, or the solutions presented are rejected

The sequential rather than parallel presentation of solutions and questions simplify the process, and has the advantage that the user is not bothered with insignificant solutions at a early stage of the conversation. The corresponding disadvantage is that if the threshold value is set too high the system might not find a solution at all, never presenting the candidate solutions for the

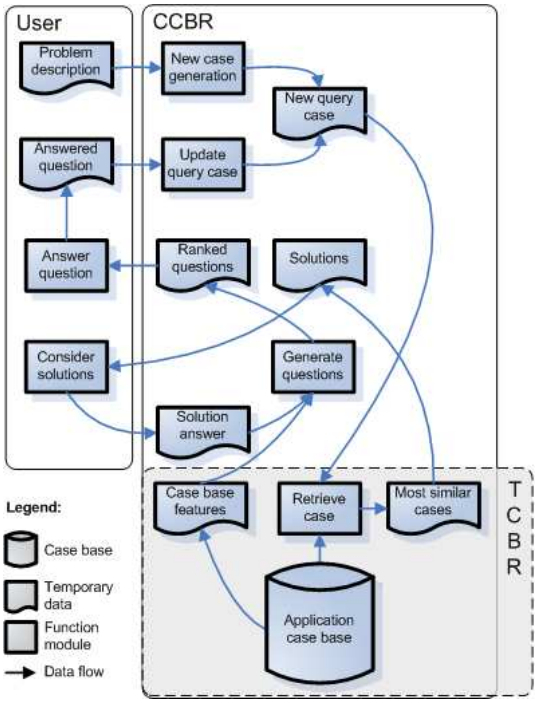


Figure 4.2: Design for the CCBR system.

user. In the setting of this thesis, where users are patients and solutions could be diagnosis, it is reasonable to believe that solutions should not be presented unless they are likely to be correct.

Both solution retrieval and question generating relay on the TCBR framework; questions are generated using the feature types stored in the case base, and solutions are found as the most similar cases returned from the case retriever.

4.1.1.1 Solution retrieval

The process for retrieving the most similar cases to the current query, solutions that should be presented to the user, relay on the modified retrieval methods implemented for the framework (see Section 3.3.2.1). These methods are query biased (only considering the features in the query case), and similar to a k-NN algorithm they use the Euclidean distance between two cases to find the similarity measure. Rather than returning the k most similar cases, a threshold value is used; only the cases with higher similarity than the threshold value are returned as possible solution. A solution is represented as a pair; the retrieved case, and the similarity value.

4.1.1.2 Question ranking

In this system questions are simply considered as features given a rank. There are no transformations from features to user readable questions, as this is not relevant to the topic of dialogue learning. Before selecting a method for ranking these questions it is important to consider which features should be included as candidates, and what scope these features should be ranked in. The set of question candidate features could contain all features in the case base, or e.g. only features included in the currently most similar cases. The selected set of features could be ranked locally for each case, or globally for the entire case base.

In a diploma master thesis (Kokkersvold (2007)) also using the TCBR framework, the information gain metric for question ranking was implemented. This method takes into consideration the most similar cases retrieved in each cycle of the CCBR process, which means that the weights are computed every time a question is to be asked. The information gain metric prove to be a very computational demanding method, and thus too time consuming to support testing of the dialogue system.

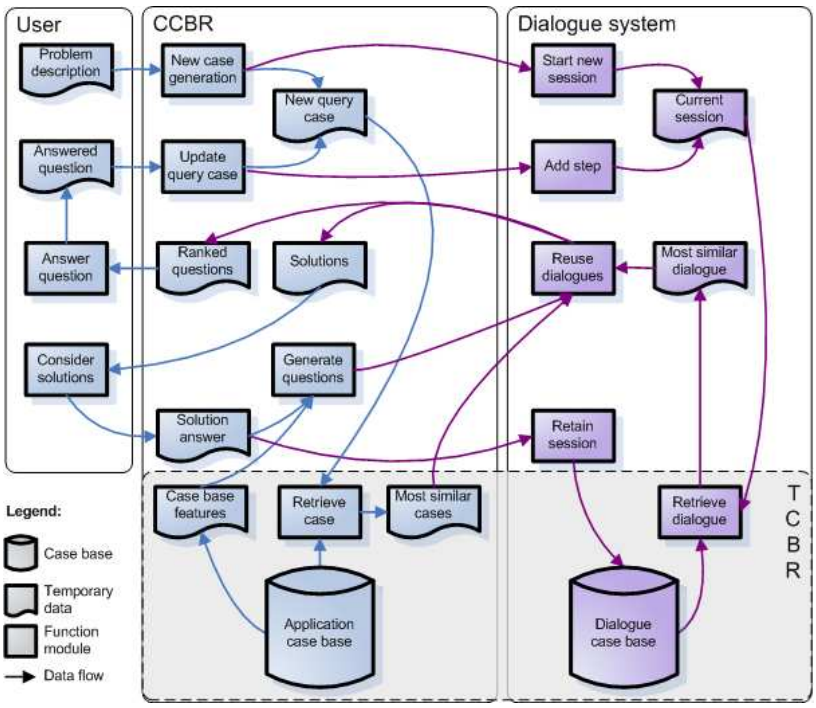
A method for question ranking used in Gu and Aamodt (2006a) is an importance weight metric similar to the EACH algorithm (Salzberg (1991)). This method is used to get a set of global weights corresponding to each of the features appearing in the case base. The EACH algorithm uses a 1-NN algorithm to for each case in the case base find the most similar case. If this most similar case suggests the same solution as the basis case, the weight of each common feature similar values is increased by a fixed positive amount, while if the solutions are dissimilar the weights for common similar features is decreased by the same amount. This method is also computational demanding due to the retrieval process done for each case, but the weights are only set once, which makes it feasible.

Although local weights could be implemented in the TCBR framework (see Section 3.3.3), for simplicity the global weighting structure implemented in the project thesis by Houeland (2007) is used.

All features that appear in the case base but have not been assigned a value in the current query are considered as possible discriminative questions, and a simplified EACH algorithm is used to set global weights for each of these.

4.1.2 Dialogue system

Figure 4.3 shows how the dialogue system extends the CCBR system. First, when generating a new query case, a new session is started in the dialogue



(a)

Figure 4.3: Including the dialogue system.

system and contained in a dialogue case as the current session. Second, when updating the query case in the CCBR system, the progress in the dialogue is added as a new step in the current session in the dialogue system. The third change is that the most similar cases and generated questions are processed by the reuse function in the dialogue system before they are presented to the user in the CCBR system. Lastly, if a solution is considered sufficient, the solution is sent to the dialogue system to be retained together with the current session. It is important to notice that the dialogue system extends the CCBR system, and thus has no purpose as a standalone module without the CCBR system.

The main concepts of the dialogue system design are equal to those described in Section 2.4.1, but differ in how a dialogue is represented due to limitations in the TCBR framework. As the dialogue cases are represented in a special way, the retrieve and retain methods must be modified correspondingly.

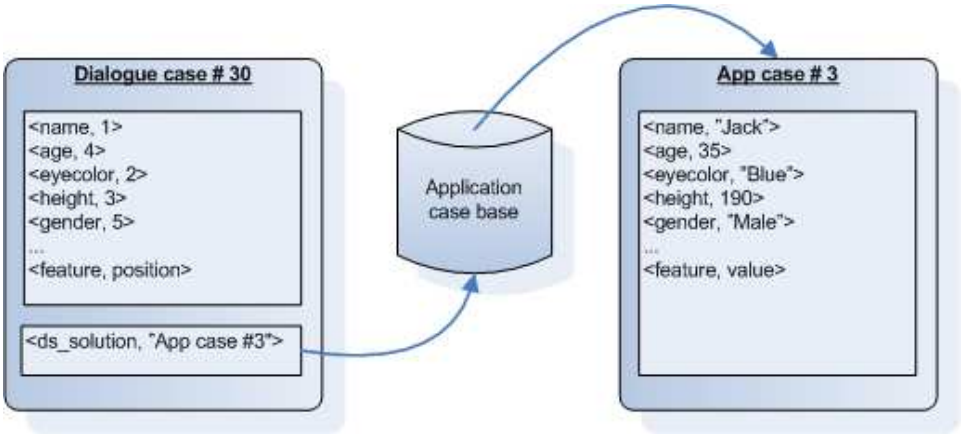


Figure 4.4: Representation of a dialogue case.

4.1.2.1 Representing a dialogue

In Section 2.4.1.1 a dialogue case representation is defined as a vector containing quadruples of feature, value, weight, and position, together with a solution ((2.4)), but case representation in the TCBR framework only allow feature-value pairs. The solution is stored as a feature of a predefined type, with the name of the corresponding application case as its value. This application case is the key to how all necessary information can be stored using only feature-value pairs. The values of the features do not need to be stored both in the application case and in the dialogue case because the cases are interconnected. In addition the weight variable can be disregarded as the weights are set globally. Figure 4.4 shows a structure where a dialogue case store the position of a feature as a feature-position pair, and a reference to the application case serving as the solution of the dialogue case. The application case stores the feature-value pairs. This design enables all features to be stored with both value and position for dialogue cases, but create a tight coupling between the application case base and the dialogue case; if the application case is removed from the case base, the dialogue case is rendered useless. A requirement for this design to work is also that the dialogue system gain access to the application case base at creation time, and the current query case when starting a new session.

4.1.2.2 Controlling a dialogue

Dialogue cases are used as an internal structure in the dialogue system. To interfere with the system and create and use new dialogue cases the methods for dialogue control must be used. To store a conversation from the CCBR system as a dialogue case, a new session must be started. That creates an

internal dialogue case containing the current session. For each question that is answered, the corresponding feature should be added as a step in the dialogue system. The position of a feature is controlled internally based on the order they are added in.

To reuse dialogues the retrieved solutions and generated questions can be sent to the dialogue system through the reuse method. The dialogue system has the currently most similar dialogue, and can update both solutions and questions.

If a conversation end in failure and no solution is found, the method for terminating the session should be used. In the case of a successful conversation, the method for committing the session should be used. The dialogue system internally decides which sessions should be retained in the dialogue case base.

4.1.2.3 Retrieving and reusing a dialogue

Based on the current session, the most similar dialogue case are retrieved using a query biased method using (2.5) defined in Section 2.4.1.2. The method is specialized for the complex representation of dialogues. The dialogue case that is retrieved is used when updating the questions and solutions. The application case serving as the solution of the dialogue case is simply added to the set of solutions, using the similarity value from the dialogue case. The weight of each question in the question set, also appearing in the dialogue case, is adjusted using (2.8) described in Section 2.4.1.3.

4.1.2.4 Retaining a dialogue

The dialogue system partly follows the retain method described by (2.9) in Section 2.4.1.4, only retaining the most general dialogue cases in the dialogue case base. When a committed session faces retention, a standard query biased retrieve is used to find all dialogue cases that have a similarity value of 1.0. Since the solution is included as a feature, this automatically ensures that the retrieved cases have the same solution as the new case. Because of the method being query biased all cases returned are either equal to the new case, or have all the features in the new case in addition to other features. In the latter, they have the same solution and the new case have a subset of the features in the retrieved case, which by the definition of the formula means that the new case is more general. Thus a simple design to trim the dialogue case base is to delete all the cases returned by the retrieve method, as the new case is more general.

4.1.3 User simulation

In order to test the total system it is necessary to simulate the actions of a user. As seen in Figure 4.2 and Figure 4.3 the user starts by giving a problem description to the CCBR system. To simulate such a problem description it is necessary for the user to have access to a set of feature-value pairs to choose from. These feature-value pairs are also needed when the user is asked to answer a question. Lastly, to evaluated solutions the user needs to know what the correct solution is. The best way to give the user realistically feature-value pairs and solutions is to use a previous stored application case as a test case.

4.1.3.1 Creating a problem description

A problem description is defined as a set of feature-value pairs supplied to the CCBR system to create the query. There are two things to consider when creating a problem description of this type; which features should be used, and how many features are needed?

First, selecting these feature-value pairs from the test case can be done in several ways; in Gu and Aamodt (2006a) they are selected randomly where the probability of a feature selected is in proportion to the weight of the feature. The conversation will thus be biased by the fact that the problem description is more likely to consist of the most relevant features, thus giving the conversation a potential unnatural good start. To simulate a user that does not know which information is most relevant when starting the conversation, a method using random selection of features with equal probability is therefore designed.

Second, the amount of features to use as a typical problem description is difficult to set. One may argue that no problem description should be needed, e.g. in a questionnaire setting, but using a query biased similarity metric this brings some problems in the aspects of testing and automatically classifying of a query. In the case where the CCBR process simply start by asking the most relevant question, let us say it asks about the feature “sex”, the user answers the question and the CCBR system starts a new retrieval process. The result of such a process based on only one feature, would be that all cases matching the supplied value for this feature are returned with a similarity value of 1.0. Consider this in our example with the feature “sex” answered to be “female”. All cases in the case base regarding females would so far match the query by 100%. This is of course the result we want from a query biased similarity metric, but if by chance one of these returned cases also have the correct solution, an automated agent would consider it as a perfectly good solution for the query; a dialogue with length 1, resulting in a classification

because the user is female. Thus in order to test the dialogue system, or other situations where automatic acceptance of a solution is necessary, a problem description of some size is preferable. Still, it is hard to set a size; we do not want to ask for more than necessary as the goal is to reduce the number of questions asked. A parameter containing the percentage of the total feature set that should be used in the problem description is therefore used, enabling testing with different sizes.

4.1.3.2 Answering questions

The CCBR system generates a list of ranked questions and asks the user for an answer to one of these questions. This enables a user to select a question that is ranked lower than others if that question is more relevant for the user. A design to simulate this is to let the automated user agent answer the highest ranked question that it knows the answer to. The agent knows the answer to a question if the corresponding feature exists in the test case used. If the agent is unable to answer any of the ranked questions, the CCBR system registers it as if there are no more questions that can be asked and the conversation ends.

4.1.3.3 Evaluating solutions

Similar to the list of questions, the CCBR system presents the user with a list of solution cases. The list is only displayed if there are any solution cases better than the predefined similarity threshold, and the similarity measure is used to rank the solution cases.

The task of an automated user agent is to either reject the solution cases, or select one solution case as the correct one, thus terminating the conversation. To evaluate the list of solution cases the agent matches the value of their solution feature against the value of the same feature in the test case. The agent can check if the desired solution is the best ranked solution, it can check if the desired solution is among the top k best ranked solutions, or it can simply be satisfied by finding the desired solution among the solutions passing the threshold check.

4.1.4 Test environment

The system designed is an interactive system, and an environment for automatic testing is needed to examine if the system improves as more dialogues are conducted. Each test should be repeated for a number of cycles to prove improvement over time due to learning. The test environment must also

conduct an ablation study for each test, performing the test both with dialogue learning, and without dialogue learning. Automatically simulating a large amount of dialogues is necessary to get a solid basis for considering the results of a test.

Based on a application case base filled with cases, a leave one out cross validation (LOOCV) method is used to simulate these dialogues; such methods have been used successfully by the CCBR community (Aha et al. (1998); Gu et al. (2005); Gu and Aamodt (2006a)). The preconditions for carrying out an experiment in the test environment are as follows:

- An application case base must be built, including only cases usable for testing.
- The cases in the application case base must be considered by a feature ranking metric to set global weights for all features.
- Threshold values for similarity measures in case retrieval, both for the CCBR system and for the dialogue system, must be set.
- The number of LOOCV cycles must be set.

Figure 4.5 shows an overview of the test environment. LOOCV starts with selecting a case from the application case base as the target case; the case supplied to the agent as a test case. This target case is left out when the system searches for the most similar case. The agent performs the dialogue returning the dialogue length required to find an adequately similar case with the same solution as the test case. The dialogue length includes the number of features in the problem description. If no similar case is found, the dialogue length is equal to the sum of features in the test case. LOOCV further selects the next case in the application case base as a new target case, returning the former target case to the case base to be included in the search. The process continues until all cases in the case base have served as a target case, and is repeated for the number of cycles wanted without dialogue learning, and with dialogue learning. Any cases with a solution not used in other cases in the case base can never serve as a target case for a successful dialogue; such cases are thus removed from the case base before LOOCV is initiated.

To ensure that the improvements in results are due to dialogue learning and not due to a growing application case base also containing the new queries, a stable application case base is used. New query cases are removed after completion of a dialogue, and, rather than the query itself, the most similar application case is used as the solution for the dialogue stored.

The post conditions of an experiment should consist of a set of data that for each cycle in the test describes the results achieved:

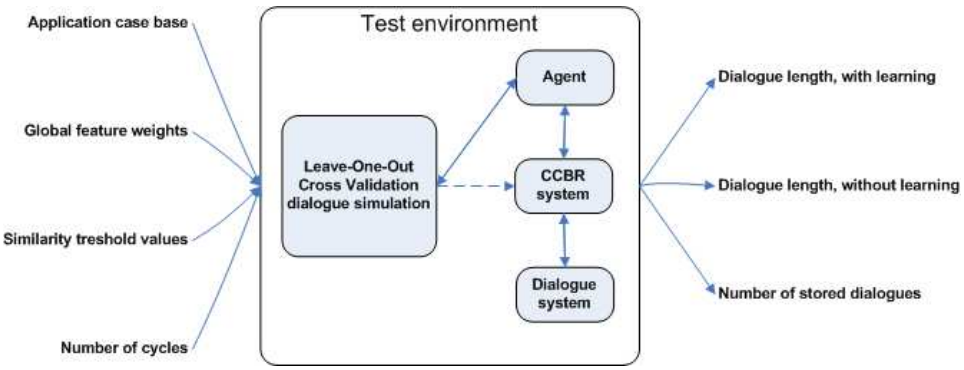


Figure 4.5: An overview over the test environment.

- The average dialogue length achieved using dialogue learning.
- The average dialogue length achieved not using dialogue learning.
- The number of dialogues currently stored in the dialogue system.

Using these output data, the improvement over time (cycles) and the difference between using dialogue learning and not using it, can be found. The number of dialogues stored over time also shows if the dialogue system is able to maintain a sustainable case base size by retaining only the most general cases.

4.2 Implementation

The third sub-goal of this thesis is to implement a prototype of the system designed. Even though the problem description only mentions implementing a part of the designed system, it is necessary to implement the complete system in order to test the methods for dialogue learning. Rather than implementing only a part of the system, the complete system is implemented but with some parts as simplified as possible. This prototype includes an agent, the CCBR system, the dialogue system, and the test environment, whereof the agent and the CCBR system are very basic simple implementations. The prototype is primarily a program for automatically testing the included parts, and can not be used for human interaction without source code changes. Still, no changes should be necessary in the CCBR system or the dialogue system.

In the rest of this section the implementation of each of the sub-parts of the prototype will be described, followed by an explanation of how they are put together, and a step by step example run of the prototype.

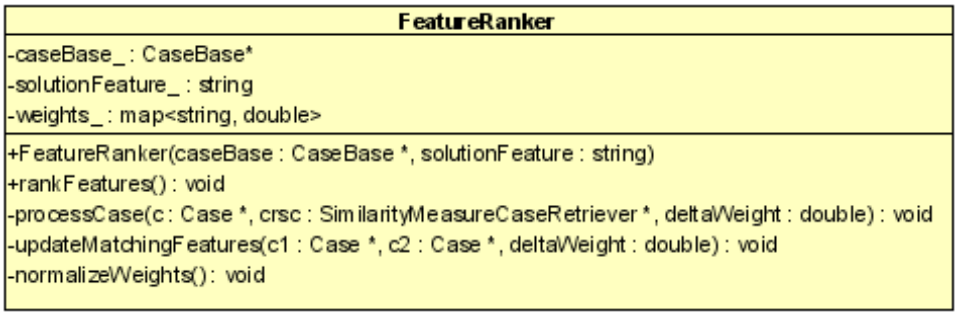


Figure 4.6: Class diagram for the feature ranker.

4.2.1 Global feature weighting

As mentioned in the design of the test environment, a method for ranking features is needed, and in the design for question ranking (Section 4.1.1.2) the EACH algorithm is chosen to set global weights for the features. The feature ranker is implemented based on this algorithm. As shown in Figure 4.6 the feature ranker has the variables *caseBase_*, *solutionFeature_*, and *weights*, denoting a reference to the case base in which the weights must be set, the feature type serving as the solution in the cases, and a map between the feature types and their corresponding weights, respectively. The feature ranker is constructed with the case base, and the solution feature type. To make sure that weights will be set for all feature types, all feature types are assigned the default weight 1.0.

After construction the ranking process is invocated through the *rankFeatures()* method, which implements a simplified version of EACH; only considering the most similar case, rather than the two most similar cases. In order to be able to retrieve the most similar case, a case retriever and a similarity measure is created. For the similarity measure the *QueryBiasedSymbolicComparator* is used, and for the case retriever the *SimilarityMeasureCaseRetriever* is used, both implemented by Houeland (2007). Following the algorithm in Table 4.1, each case in the case base is processed as a base case; finding the most similar case, checking whether the base case and the most similar case have the same solution, if the solutions are equal the weights of the matching features they share are increased by a fixed amount, if the solutions are different the weights are decreased by the same fixed amount. Finally the weights are normalized to summarize to 1.0, all weights being between 0.0 and 1.0, and the resulting feature type-weight map is used to set global weights in the case base.

Table 4.1: Pseudo code of the simplified EACH algortihm.

```
Procedure simpleEACH(CaseBase casebase)
    For Each Case c∈casebase
        processCase(c)
    End Loop
    normalizeWeights()
Return
Procedure processCase(Case case)
    Case match = findMostSimilarCase(case)
    If solution(match) = solution(case)
        For Each Feature f∈case∪match

            If value(f, match) = value(f, case)

              increaseWeight(f)
            End If
        End Loop
    Else
        For Each feature f∈case∪match

            If value(f, match) = value(f, case)

              decreaseWeight(f)
            End If
        End Loop
    End If
Return
```

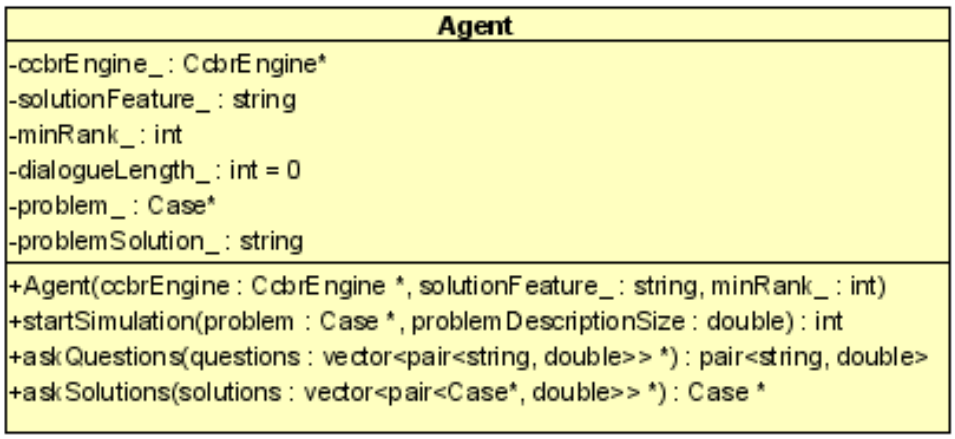


Figure 4.7: Class diagram for the agent.

4.2.2 Automated agent

The user simulation was designed as an automated agent. As shown in Figure 4.7 the agent implementation includes the variables *ccbrEngine_*, *solutionFeature_*, *minRank_*, *dialogueLength_*, *problem_*, and *problemSolution_*, denoting a reference to the CCBR system, the feature type serving as the solution in the cases, the lowest allowed ranking for the accepted case, the length of the current dialogue, the test case serving as a problem, and the solution of this case, respectively. The agent is constructed with the reference to the CCBR system, the feature type serving as the solution, and the lowest allowed ranking, as these variables are constant during a simulation.

The designed agent is used by the test environment to start the dialogue. This is implemented through the *startSimulation()* method which accepts a test case as its problem, and a double value denoting the portion of the test case that should be used as the problem description. To determine the set of questions that the agent can answer, the features of the test case is retrieved. Using the size of this set, the number of features corresponding to the portion value is found. Features are then selected randomly without replacement from the set, and added to a problem description as feature-value pairs. With the created problem description the agent starts a query to the CCBR system, which in turn interacts with the agent using the *askQuestions()* and the *askSolutions()* methods.

The *askQuestions()* method enables the CCBR system to present a vector of feature type-weight pairs (implying that the features are ranked by their estimated importance) for the agent to consider and answer. The implemented method considers the feature types in a descending order, checking if the test case has the corresponding feature; if the feature is found in the

test case its value is retrieved as the answer, if the feature is not found the method moves on to the next best feature name. The answered feature type is returned to the CCBR system as a feature type-value pair. In the special case of the agent not being able to find any of the feature names in the vector, an empty pair is returned. This, as explained in Section 4.2.3, in fact means, due to the way the vector of feature name-weight pairs is generated, that the dialogue is over.

The *askSolutions()* method, similar to *askQuestions()*, enables the CCBR system to present a vector of case-similarity pairs for the agent to consider. These cases are possible solutions ranked by their similarity value to the current query. The implemented method uses the value for lowest allowed ranking for an accepted case, and considers only the cases better than this. For example if the lowest allowed rank is 1, only the best case is considered, but if the lowest allowed rank is 3, the 3 best cases are considered. The implemented method considers a case by retrieving the solution feature of the case, and matching this with the solution stored for the problem; if the match is successful, the case is accepted as the solution to the query. If such a solution case is found, it is returned to the CCBR system. In any other case the method simply returns NULL, to imply that no solutions are acceptable.

When the query returns control to the *startSimulation()* method the accumulated dialogue length is returned to the test environment, and the simulation is completed.

4.2.3 CCBR system

In the implementation the CCBR system is called an engine, because this is the part that drives the dialogue by finding new questions to ask. As shown in Figure 4.8 the CCBR engine includes the variables *casebase_*, *dialogueSystem_*, *features_*, *queryCounter*, *solutionFeature*, *similarityThreshold_*, and *dl_*, denoting a reference to the application case base, a reference to the dialogue system, the complete set of features in the application case base, the number of queries performed, the feature type serving as the solution in the cases, the threshold deciding if a solution has high enough similarity to be used, and whether or not the dialogue system is enabled, respectively. The CCBR engine is constructed with a reference to the application case base, the feature type serving as solution, and the similarity threshold for both the application case base and for the dialogue case base as parameters. During the construction the dialogue system is also created internally in the CCBR engine, using the constructor explained later in Section 4.2.4. The number of queries performed is simply used to give the query cases created unique names.

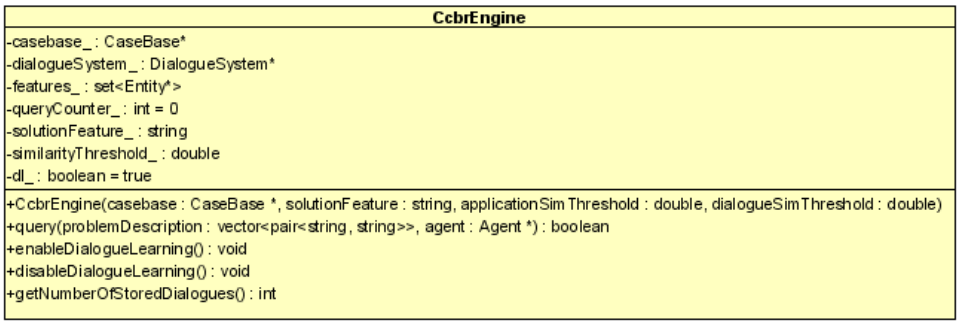


Figure 4.8: Class diagram for the CCBR engine.

When the CCBR engine is created it is ready to receive queries through the *query()* method. This method accepts a vector of feature type-value pairs as a problem description, as well as a reference to the agent invocating it. As seen in Table 4.2, the feature type serving as the solution is first removed from the set of question candidate features. Next, a new query case is created, and the feature name-value pairs from the problem description are added as features in it. These features are then also removed from the set of question candidate features. In order to be able to retrieve the most similar cases a case retriever and a similarity measure is created. As in the feature ranker (see Section 4.2.1) the similarity measure used is the *QueryBiasedSymbolicComparator*. For the case retriever a modified version of the *SimilarityMeasureCaseRetriever* is used. The modifications consist of introducing the similarity threshold, making sure that the retrieve method only returns the cases that have a higher similarity to the query case than the threshold (shown in the class diagram in Figure 4.10). The method further invocate the *newSession()* method of the dialogue system (see Figure 4.9) using the query. After this, the method enters a loop that it leaves only when a solution is found, or there are no more question candidate features.

The first step in the loop is using the case retriever to find the most similar cases to the query case, with a higher similarity than the threshold value, and storing the result as the solutions. The second step is to generate a set of ranked questions candidate features, which is done using the private method *generateQuestions()*; the method iterates the set of remaining question candidate features, looking up the global weights set (by the feature ranker described in Section 4.2.1) for their corresponding feature type in the case base, and returning a vector of feature type-weight pairs sorted descending by their weight. The third step is a check for whether any questions were generated or not. If the set is empty, the query case is deleted, and the method *discardSession()* in the dialogue system is invocated. Further the query method returns with value false, meaning that the dialogue failed to solve the query case. If the set is not empty, the process continues by

Table 4.2: Pseudo code for the query.

```
CaseBase cb
DialogueSystem ds
Agent agent
Procedure query(problemDescription)
    Feature[] candidates = cb.getFeatureTypes()
    candidates.remove(solutionFeature)
    Case query
    For Each feature f∈problemDescription
        query.addFeature(f)
        candidates.remove(f)
    End Loop
    ds.newSession(query)
    While !solution And candidates

        solutions = retrieveMostSimilarCases(query)
        questions = generateQuestions(candidates)
        If !questions Then
           Delete query
           ds.discardSession()
           Return false
        Else

           ds.reuseDialogue(questions,solutions)
           answer = agent.askQuestions()
           If !answer Then
             Delete query
             ds.discardSession()
             Return false
           Else
             query.addFeature(answer)
             ds.addFeature(answer)
           End If
        End If
        If solutions Then

           solution = agent.askSolutions(solutions)
        End If
    End Loop
    ds.commitSession(solution)
    Delete query
Return true
```

invocating the *reuseDialogue()* method in the dialogue system, possible improving the sets of question candidate features and solution cases. Further the agents *askQuestions()* is invocated and an answer is received. If the pair containing the answer is empty, it means that the agent was unable to answer any of the questions. As the nature of the question generating is to present all remaining possible question candidate features in a ranked fashion, this in turn means that none of the remaining question candidate features can be answered, thus the query case is deleted, the method *discardSession()* in the dialogue system is invocated, and the method returns with value false. In the case where the pair contains an answer, a corresponding feature is added to the query case, and the feature is removed from the set of question candidate features. The dialogue system is also informed of the new step in the dialogue using the *addFeature()* method. Finally, the last step of the loop is, if the solution set not being empty, to ask the agent for a solution using the *askSolutions()* method. If either the solution set was empty, or the agent declined the solutions in it, the loop continues. If the agent returned an accepted solution the loop ends, and the query is successful. In a normal CCBR system the query case would be considered for retaining, but as a stable application case base is wanted the query case is simply deleted. The solution found is sent to the dialogue system invocating the *commitSession()* method, and the query method returns the true value at success.

As mentioned in the explanation above; the query case is deleted both when the dialogue fails to solve the query, and when it succeeds (in order to keep the application base stable). As later described in Section 4.2.4, deleting of cases is also necessary when retaining dialogues. This ability to delete cases is not implemented in the TCBR framework, resulting in lots of faulty relations in the underlying knowledge model if any case objects are deleted. Therefore the framework had to be changed to handle this problem. As both cases and features are entities, the code for removing the corresponding relations was placed in the entity class. When deleting a case instance, all belonging feature instances is also deleted, triggering the C++ destructor code for each entity. This code is set to delete the entity from the knowledge model, and iterate trough the local relations invocating the new *removeRelationsTo(this)* method for each target entity. As given by the methods name, all relations in entities that point to the deleted entity are thus removed. The destructor for the relation class is set to remove itself from the knowledge model, making sure that the knowledge model does not contain pointers to any destroyed entities or relations. To achieve this, the code in the classes for cases, entities, relations, and knowledge models in the TCBR framework had to be extended.

Lastly, the CCBR engine has three methods for controlling the use of the dialogue system; *enableDialogueLearing()* and *disableDialogueLearning()* simply enables or disables the dialogue system for use. The method *getNumberOfS-*

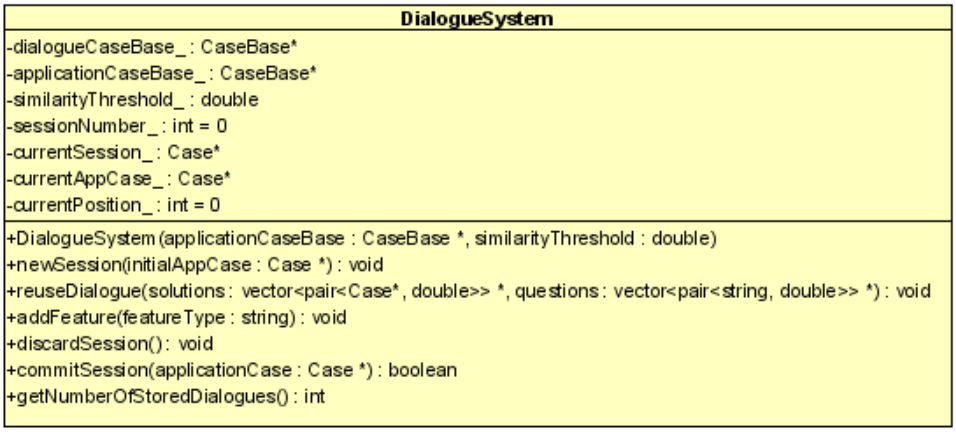


Figure 4.9: Class diagram for the dialogue system.

toredDialogues() give the test environment access to the size of the dialogue case base. This is necessary since the dialogue system is an internal part of the CCBR engine, and thus can not be reached directly from outside the CCBR engine. If the dialogue system is disabled, the query does not use any of its methods for improving the questions or solutions, or retaining dialogues.

4.2.4 Dialogue system

The implemented dialogue system, as designed, very much follows the methods described in Section 4.1.2 with some adjustment to fit the use of the TCBR framework. As shown in Figure 4.9 the dialogue system has the variables *dialogueCaseBase_*, *applicationCaseBase_*, *similarityThreshold_*, *sessionNumber_*, *currentSession_*, *currentAppCase_*, and *currentPosition_*, denoting a reference to the dialogue case base, a reference to the application case base used in the CCBR system, the threshold for how similar a dialogue case have to be to the current dialogue before reusing it, the number of dialogue sessions, the dialogue case containing the current session, the query case currently used in the CCBR system, and the current position in the dialogue (the number of steps currently conducted), respectively. The dialogue system is constructed with the reference to the application case base and the similarity threshold as parameters, as these variables are constant through the existence of the system. The session number is simply used to give the new dialogue cases unique names.

The dialogue system is a closed system; you can not manually add dialogue cases to the case base. A new dialogue case must be built using the *newSession()* method which resets the current position to zero and creates

an internal dialogue case as the current session. The features included in the initial application case is not added as steps of the dialogue, due to the totally random selection of these, and the positive computational effect of having shorter dialogue case descriptions. To extend this dialogue case when new steps in the dialogue occur, the *addFeature()* method is used. This method stores a new feature in the current dialogue case of the added feature type with the current position as its value, before incrementing the current position.

The *reuseDialogue()* method is the dialogue system's opportunity to effect the dialogue process. The method retrieves the dialogue most similar to the current session using the new *DialogueCaseRetriever* implementation of the case retriever, and the new *DialogueCaseComparator* implementation of the similarity measure (see Figure 4.10). This case retriever is created with not only a reference to the dialogue case base, but also to the application case base, thus making it possible to find both the position and the value of a feature type in a dialogue case. Both the dialogue case representing the current session, and the application case representing the current query, are necessary to retrieve the most similar dialogue. The retriever follows equation (2.5) to find the distance between the cases. The similarity measure method *getSymbolicDifference()* in *QueryBiasedSymbolicComparator* follows by equation (2.6) and is inherited and used together with the *getPositionWeight()* method in *DialogueCaseComparator* which follows by equation (2.7). The most similar dialogue case is, if better than the similarity threshold, used as the basis for updating the weights in the set of question candidate features, and potentially to add a new solution to the set of ranked solution cases. For each feature types in the set of question candidate features also included in the most similar dialogue case, the corresponding weight is updated using equation (2.8). The solution of the most similar dialogue case is simply added to the set of ranked solutions if it is not already included, using the similarity strength assigned to the dialogue case.

If the proceeding query in the CCBR system at any time fails, the *discardSession()* method should be used to make sure that the current session is not stored as a dialogue case in the case base. In the case of a successful dialogue, the *commitSession()* method should be used. This method implements the retain strategy explain in Section 4.1.2.4, using a similarity measure of the *QueryBiasedSymbolicComparator* type, and a case retriever of the modified *SimilarityMeasureCaseRetriever* (with a threshold limit, see Figure 4.10). The cases retrieved with threshold limit 1.0 are considered as equal or more specialized than the current session, and is thus deleted from the dialogue case base. Finally, the dialogue system has the method *getNumberOfStoredDialogues()* in order to let the CCBR system pass on the current size of the dialogue case base to the test environment.

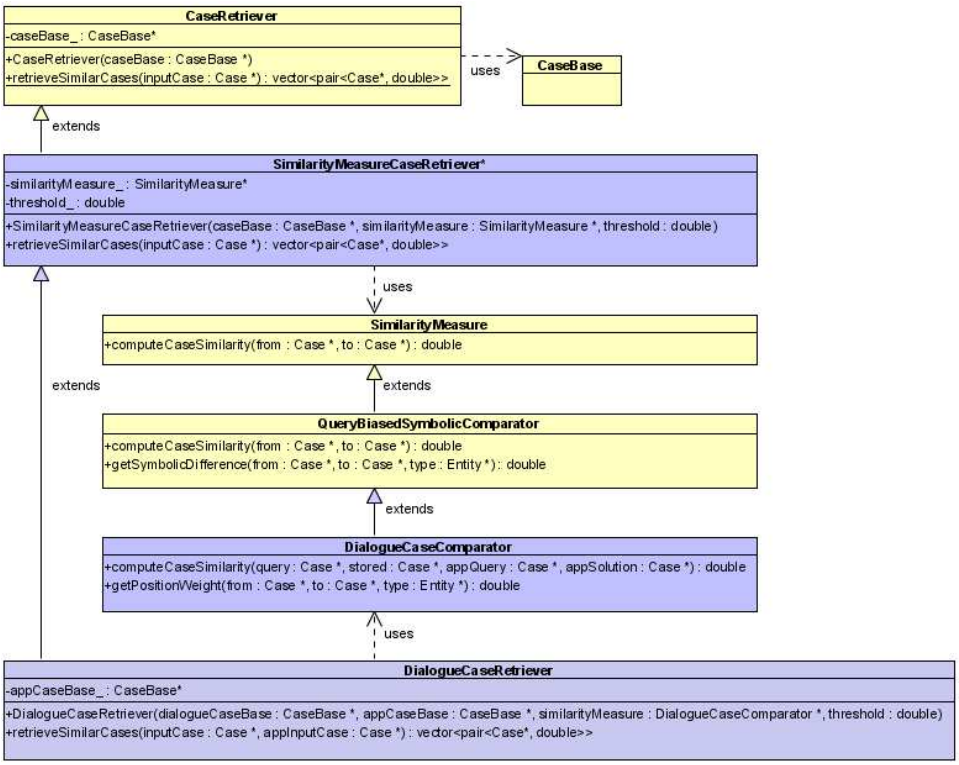


Figure 4.10: Class diagram for the dialogue case retriever.

4.2.5 Putting the system together

The task of constructing and connecting the sub parts of the system falls on the test environment, which creates a CCBR system and an agent with reference to this. The test environment needs access to the CCBR system in order to enable and disable the dialogue learning, while the simulations are started using the agent. The evaluation process follows the pseudo code in Table 4.3. A case base is created, and the features in this are ranked using the feature ranker. A LOOCV loop is started first with the dialogue learning disabled, then with the dialogue learning enabled. The results from each simulation are stored, and presented in the end. To avoid deleting the case that is left out from the case base during simulation, methods for handling deactivation of cases was developed.

The case implementation in the TCBR framework was altered to include the methods *activate()*, *deactivate()*, and *isActive()*. To store whether a case is deactivated or not, the special feature type *_deactivated* was introduced. Syntax for feature types was defined, deciding that feature types starting with the underscore symbol would regard system data rather than case data. The *activate()* method thus simply add the feature type *_deactivated* to the case, while the *deactivate()* method removes it. The *isActive()* method simply checks if the case has the feature. Knowing this, a new method was implemented in the case base representation in the TCBR framework; in addition to the old method *getCases()*, the new method *getActiveCases()* was added. This new method was then used instead of the old one in the implementations of the case retriever, ensuring that only active cases are searched. The introduction of feature types not regarding the case data demanded a change in the *getFeatureTypes()* method in the case base implementation, not returning the feature types regarding the system data.

With the evaluation process explained and the ability to deactivate cases established, simulations are performed by the LOOCV loop. The internal communication between the sub parts of the system during a simulation can best be illustrated by the sequence diagram shown in Figure 4.11. The figure shows the initial methods to activate the system, the loop which drives the dialogue, and the alternative endings of the simulation.

4.2.6 An example run

To demonstrate how the complete prototype works, the output from an example run is explained in this section. As described in Table 4.4, the program accepts a set of parameters for configuration that can be supplied at execution time, but all parameters have default values. An example start is:

Table 4.3: Pseudo code of the evaluation process.

```
Procedure evaluation()
    CaseBase casebase
    insertCases(casebase)
    rankFeatures(casebase)
    Results results
    disableDialogueLearning()
    For Each cycle
        For Each Case c∈casebase
            deactivate(c)
            startSimulation(c)
            update(results)
            activate(c)
        End Loop
    End Loop
    enableDialogueLearning()
    For Each cycle
        For Each Case c∈casebase
            deactivate(c)
            startSimulation(c)
            update(results)
            activate(c)
        End Loop
    End Loop
    print(results)
Return
```

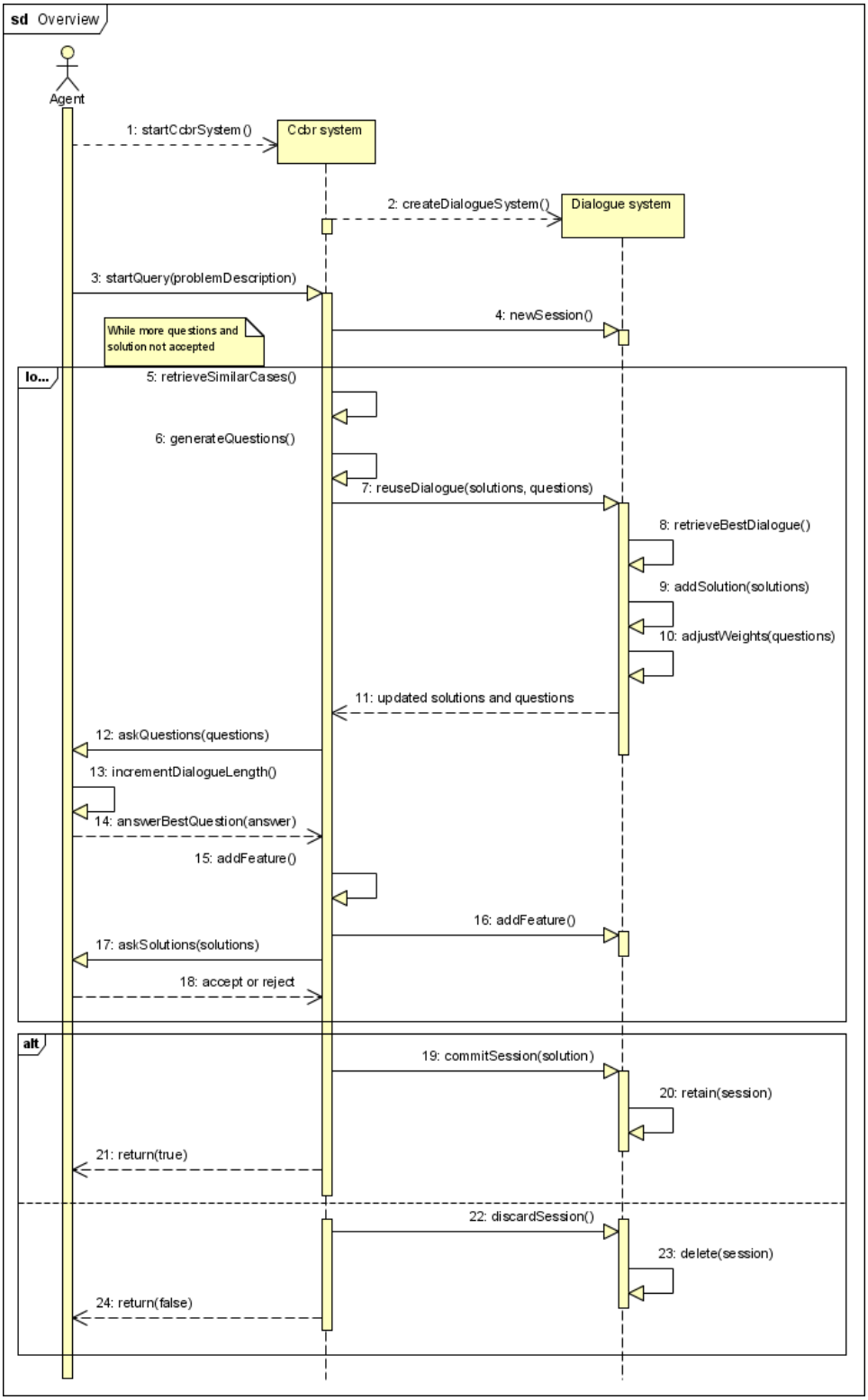


Figure 4.11: The internal communication sequence in the system.

```
dialearn.exe -sof "class" -dat "C:\\lung-cancer.dat"
-npat -pds 0.10 -ast 0.7 -dst 0.5 -noc 32 -cyc 2
```

The order of the parameters is of no consequence, as long as the correct prefix is used.

The data file must be in a tabulator separated text-format, where the first line in the file consists of the feature types, and the following lines includes the feature values, one case for each line (see Table 4.5).

The dataset used in this example run is the lung-cancer dataset from the UCI repository (D.J. Newman and Merz (1998)) (for further information on the dataset see Section 5.2.2.1).

The first output the program produces is from the case base generator (read more about the case base generator in Section 5.2.3). In the example the number of cases is set to 32, which is equal to the number of cases available, thus no random selection is necessary and the case base generator use the simple method for inserting a section of cases in the range 0 to 32.

```
CBG-> Stored 32 cases from file: C:\lung-cancer.dat
CBG-> Number of cases equal to or larger than range.
CBG-> Inserting cases 0 to 32 from a dataset with 32 cases,
into casebase ACBase
```

The second output produced is from the test environment, giving the characteristics for the case base generated.

```
MAIN-> Case base characteristics:
MAIN-> Cases: 32
MAIN-> Features: 57
MAIN-> Solutions: 3
```

The next output produced is from the test environment starting the feature ranker, which outputs a line when each case is processed. It might be a time consuming process to process all cases, and some form of progression sign is practical. When all cases is processed the feature ranker normalizes the weights.

```
MAIN-> Global feature ranking started...
FR-> Processing Case #1
FR-> Processing Case #2
FR-> Processing Case #3
...
FR-> Normalizing weights.
```

The next output produced is from the CCBR system informing that the dialogue learning was disabled in order to run the cycles with ablation of the dialogue system first.

Table 4.4: Start parameters for the program.

Prefix	Parameter	Default	Explanation
-sof	Solution feature	db_primdiagnosis	The feature type serving as the solution for the cases.
-dat	Data file		Location of the data file, e.g. "C:\\datafile.dat".
-npat	Setup for other than PAT-C	No value.	Use this if other datasets than the PAT-C dataset is used. No value should be supplied, only the prefix.
-noc	Number of cases	40	The number of cases that is randomly selected from the data file.
-pos	Selection start	-1	The start position for the case section selection, e.g. from case -pos to case -noc. Default is random selection.
-msr	Min. solution rank	1	The minimum rank for an acceptable solution case.
-msc	Min. solution count	2	The minimum number of cases in a classification.
-cyc	Number of cycles	2	The number of cycles to repeat the LOOCV process to demonstrate learning.
-pds	Problem description size	0.10	The portion of the features in a test case used as the problem description (0.10 is 10% of the features).
-ast	Application case base similarity threshold	0.6	The similarity threshold for retrieval of application cases.
-dst	Dialogue case base similarity threshold	0.5	The similarity threshold for retrieval of dialogue cases.

Table 4.5: A data file example.

Age	Sex	Height	Weight
32	M	180	78
23	F	170	56
26	F	175	69

```
CCBR-> Dialogue learning disabled.
```

Next, the test environment informs that it is starting a simulation with the case "Case #1" left out. The output also include the time currently elapsed since the program started.

```
MAIN-> Starting simulation. (Case: Case #1) 2.66667min
```

The agent has been activated and informs that it has selected 5 features for the problem description (10% of the 57 features, floored).

```
AGENT-> Selecting 5 features for problemdescription
```

The next output produced is from the CCBR system. A new query has been started, similar cases are retrieved, and questions are generated.

```
CCBR-> New query started.
CCBR-> Retrieve similar cases.
CCBR-> Generate ranked questions.
```

The agent then selects the best ranked question to answer, which has the feature type "p8" (in this data file denoting the eight parameter of the dataset). The weight of the questions is also displayed.

```
AGENT-> Answers best ranked question: p8 (0.0346667)
```

The agent further informs that it was asked to confirm solutions. The currently best case is printed with its similarity rank and solution value. The correct solution for the query is also displayed.

```
AGENT-> Solution display 6(Correct: 1 )
Case #18 (0.728503) Solution: 2
```

This process continues, building a larger query case:

```
CCBR-> Retrieve similar cases.
CCBR-> Generate ranked questions.
AGENT-> Answers best ranked question: p44 (0.0346667)
CCBR-> Retrieve similar cases.
CCBR-> Generate ranked questions.
AGENT-> Answers best ranked question: p53 (0.032)
CCBR-> Retrieve similar cases.
```

```
CCBR-> Generate ranked questions.
AGENT-> Answers best ranked question: p23 (0.0293333)
...
```

This particular query ended with failure, by the CCBR system informing that it has no more questions available, and the agent informing of a false simulation ending with maximum dialogue length:

```
CCBR-> No more questions available.
AGENT-> Simulation: 0 Length: 56
```

The test environment informs that the simulation was completed in some amount of time after the start of the program, and starts a new simulation with the next case left out.

```
MAIN-> Simulation complete. 4.65min
MAIN-> Starting simulation. (Case: Case #2) 4.65min
```

The same procedure as in the last simulation is repeated, but this simulation ends differently. The agent informs that the correct solution is found, and that the simulations thus was true, with dialogue length 11.

```
AGENT-> Solution display 11(Correct: 1 )
Case #4 (0.775079) Solution: 1
AGENT-> Solution correct. Case #4
AGENT-> Simulation: 1 Length: 11
MAIN-> Simulation complete. 4.71667min
```

After some time the CCBR system informs that the dialogue learning is enabled. This means that both cycles without learning are completed. A new simulation is started as before:

```
CCBR-> Dialogue learning enabled.
MAIN-> Starting simulation. (Case: Case #1) 29.3833min
AGENT-> Selecting 5 features for problemdescription
CCBR-> New query started.
```

When the query has started, the dialogue system presents us with new lines of output. A new session is started, and the current dialogue case base size is displayed.

```
DS-> New session started.
DS-> Current dialogue base size: 1
```

The next time the dialogue system gives any output is to inform that when trying to reuse dialogues no dialogue cases better than the similarity threshold value were found.

```
CCBR-> Retrieve similar cases.
CCBR-> Generate ranked questions.
```

```
DS-> No dialogue cases better than threshold value.
AGENT-> Answers best ranked question: p44 (0.0346667)
```

The rest of the process continues as before, until the case base has reached a size that makes it useful for reuse. In the following simulation the case base has reached a size of 18:

```
MAIN-> Starting simulation. (Case: Case #18) 33.0167min
AGENT-> Selecting 5 features for problemdescription:
CCBR-> New query started.
DS-> New session started.
DS-> Current dialogue base size: 18
```

After answering the first question the dialogue system finds a dialogue case witch has a high enough similarity measure. This dialogue case is thus used to update the weights of all question candidate features also in the dialogue case:

```
CCBR-> Retrieve similar cases.
CCBR-> Generate ranked questions.
DS-> Best: Dialogue #18 (1)
DS-> Weight for p44 changed from 0.0346667 to 0.0520136
DS-> Weight for p53 changed from 0.032 to 0.0490918
DS-> Weight for p1 changed from 0.0293333 to 0.04617
DS-> Weight for p49 changed from 0.0293333 to 0.0454047
DS-> Weight for p6 changed from 0.0293333 to 0.0459149
DS-> Weight for p23 changed from 0.0293333 to 0.0456598
DS-> Weight for p48 changed from 0.0266667 to 0.0419728
DS-> Weight for p52 changed from 0.0266667 to 0.042483
DS-> Weight for p24 changed from 0.024 to 0.0387959
DS-> Weight for p50 changed from 0.024 to 0.039051
DS-> Weight for p30 changed from 0.024 to 0.0385408
DS-> Weight for p54 changed from 0.0213333 to 0.035619
DS-> Weight for p35 changed from 0.0186667 to 0.0316769
DS-> Weight for p33 changed from 0.0186667 to 0.0314218
DS-> Weight for p19 changed from 0.0186667 to 0.0309116
DS-> Weight for p45 changed from 0.0186667 to 0.031932
DS-> Weight for p17 changed from 0.0186667 to 0.0306565
DS-> Weight for p16 changed from 0.0186667 to 0.0304014
DS-> Weight for p46 changed from 0.0186667 to 0.0321871
DS-> Weight for p21 changed from 0.0186667 to 0.0311667
DS-> Weight for p55 changed from 0.0186667 to 0.0324422
DS-> Weight for p56 changed from 0.0186667 to 0.0326973
DS-> Weight for p51 changed from 0.016 to 0.0274796
DS-> Weight for p40 changed from 0.016 to 0.0272245
```

```
DS-> Weight for p39 changed from 0.016 to 0.0267143
DS-> Weight for p2 changed from 0.016 to 0.0269694
DS-> Weight for p28 changed from 0.016 to 0.0254388
DS-> Weight for p27 changed from 0.016 to 0.0256939
DS-> Weight for p4 changed from 0.0133333 to 0.022517
AGENT-> Answers best ranked question: p44 (0.0520136)
```

The process continues, and when the correct solution is found the dialogue system informs that the session is retained.

```
AGENT-> Solution correct. Case #20
DS-> Session retained.
AGENT-> Simulation: 1 Length: 14
```

In a later simulation, the dialogue system not only informs that the session is retained, but that a formerly stored dialogue case is removed due to the new session being a more general dialogue case.

```
DS-> Removed Dialogue #34 as a result of a more general
case stored.
DS-> Session retained.
```

Finally, when all the simulations are completed, the test environment outputs the results. The total execution time is displayed, and the case base characteristics are repeated. For each cycle the average dialogue length with and without dialogue learning, and the number of stored dialogue cases in the dialogue system, is displayed.

```
MAIN-> Results for this case base:
MAIN-> Total time: 59.7333min
MAIN-> Total cases: 32
MAIN-> Features: 57
MAIN-> Solutions: 3
MAIN-> Cycle 0
MAIN-> Dialogue length, no learning: 19.1562
MAIN-> Dialogue length, learning: 12.3438
MAIN-> Stored dialogue cases: 31
MAIN-> Cycle 1
MAIN-> Dialogue length, no learning: 17.75
MAIN-> Dialogue length, learning: 16.4062
MAIN-> Stored dialogue cases: 49
Press any key to terminate experiment.
```

Using these results, the average dialogue length over the cycles can be found, and potential improvements due to dialogue learning can be shown. In the example run an average dialogue length reduction of 22.1% was achieved.

The size of the dialogue case base over several cycles also give important information about whether or not the retain strategy used is working.

Chapter 5

Evaluation

This chapter concerns the evaluation of the system designed and implemented. The properties of the test environment are explained, the choices of datasets justified and parameters for them selected, finally the results for the system on each of the datasets are presented.

5.1 Properties of the test environment

The most interesting elements included in the test environment is the use of LOOCV for simulation, the use of ablation to evaluate improvements, and the use of cycles to show learning and maintainability over time.

LOOCV is a practical method for simulating the human-computer conversation process when a sufficient number of human subjects to run the experiments are unavailable. The method excludes one case from the case base, using this case as the test case, and searching the remaining cases for similar solutions. Due to the fact that the test case is excluded there is no guarantee that a solutions will be found, in contrast to leave-one-in cross validation (LOICV) where the test case is included in the search as a possible solution. Classes consisting of only one case can never be reached because of this case being left out, and are thus not applicable for LOOCV. The test environment accepts a parameter which sets the minimum number of cases for each classification, and deletes the cases belonging to a classification that does not satisfy this number. The default minimum number is two. Methods similar to LOOCV have also earlier been successfully used in the CCBR community (Aha et al. (1998); Gu et al. (2005); Gu and Aamodt (2006a)).

Ablation evaluation is a method for analyzing the contributions of different modules of a system to the total performance improvement (Gu and Aamodt (2006b)). The tests are performed with one or more modules deactivated,

removed, or replaced, and the results are compared against the complete system. This method was used to evaluate the PROTOS system (Bareiss (1989)), the SIROCCO system (McLaren (2003)), the NaCoDAE system (Aha et al. (2001)), and was also used by Gu and Aamodt (2006a). Because of the nature of the dialogue system, being an extension to the CCBR system, ablation evaluation is a practical approach for testing. Tests are thus conducted both with the dialogue system disabled, and enabled.

To inspect the continuous learning characteristics of the dialogue system the LOOCV is repeated for more then one cycles, typically two. For the LOOCV with the dialogue system disabled there are no difference between the cycles other than the random selection of problem descriptions. On the other hand, when the dialogue system is enabled, the first cycle starts with an empty dialogue case base, delaying the effect of reusing dialogues until some are retained. The second cycle should thus improve due to the existence of retained dialogues from the first cycle. Another effect of the second cycle is the ability to see if the retain strategy is effective. Over several cycles the growth of the dialogue case base should decline. The use of random problem description have the effect that there can be created many more dialogues than there exist test cases, which in turn makes it possible to get different simulations for each run on the same dataset.

5.2 Datasets

Data is necessary in order to evaluate the system, and the choice of datasets is influenced by several criteria. First, the purpose of the system is to reduce the number of questions asked during a conversation, but as discussed in Section 4.1.3.1 there is a problem with too short problem descriptions. Datasets with a certain minimum amount of features must thus be used to get useful results. It is also more interesting to see the effect on longer dialogues, as they are in greater need of length reduction. Short dialogues do not give very heavy cognitive load for the user.

The second thing that influences the choice of datasets is the connection between the size of the dataset and the computational effort needed to use them. The number of cases in the dataset, and especially the number of features they have, thus decides the amount of time needed for a simulation. Since the system uses query biased retrieval methods the process goes slower and slower as more features are added to the query case, meaning that if an acceptable case is hard to find the amount of time used to search for it increases rapidly.

The third influencing element is the fact that the TCBR framework only supports discrete textual values for features. If the dataset includes a high

amount of continuous valued features this might be the reason for bad performance, independent of the actions of the dialogue system. It is thus preferable if the datasets have mostly discrete nominal, or Boolean, valued features, since these are handled well by the TCBR framework.

Forth, it would be interesting to use data relevant to the problem to be addressed by the EPCRC project. The datasets finally chosen were a dataset from a project related to the EPCRC project, and two datasets from the UCI repository (D.J. Newman and Merz (1998)).

5.2.1 Data from PAT-C

In relation to the EPCRC project a study have been carried out by the Department of Cancer Research and Molecular Medicine, NTNU (Pain et al. (2006)), where a series of tests were performed through computer fixed testing (CFT). Hospitals located in Trondheim, Molde, Bergen, Hamar, Oslo and Skien were involved, and data collected from approximately 1000 patients. The Palliative Assessment Tool - Computerized (PAT-C) was used to perform the testing with a computerized questionnaire where the patient answers questions in a given sequence. The reply can be added by either the patient himself, or some assisting health person, depending on the patient's abilities.

These tests resulted in a substantial amount of data on pain measurements from real patients, and give a good foundation for creating a set of test cases. Not all tests were completed, either because of the patient being too tired to continue, or of other problems, leaving some of the data incomplete and without a diagnosis. A process to convert the data to cases was necessary.

5.2.1.1 Pruning irrelevant data

The first step in creating cases was a selection of relevant variables from the dataset. The dataset included personal data such as patient id and residence, test data such as where and on what computer the test was performed, which version of the test software was used etc. These variables were left out from the case data both to make it anonymous, and to keep only the data of clinical importance. It is the answers from, and the condition of, the patient which give the foundation for a diagnosis, not the software version or computer type. The dataset also included variables containing the time used to answer each question. These variables were set automatically when the patient proceeded to the next question, and might not be a measure of how much time the patient needed, rather a measure of how long the application

remained idle. The variable for total time used was also in some occasions much higher than the sum of the time for each question because the patient, or assistant, forgot to close the test when it was done. All time variables are therefore excluded from the case data for simplicity; they may not be of clinical value, and also the TCBR framework does currently not support a time format for features. Remaining variables used for case data includes *db_bornyear*, *db_sex*, and *db_karnofsky* which all were filled before the test started, *db_primdiagnosis* holding the classification of the data, followed by a series of variables containing the answers to every question the patient got. Some of the variables are dependent on other variables; the questionnaire includes a graphical map of the human body with zones that can be marked as painful areas, this map is never displayed for the patient if he previously answered zero to two different questions about the amount of pain felt in some recent time period. Thus if the map is not displayed the variables linked to this map is not filled, but the data is still to be considered as complete. A total of 97 variables for each test are kept when the map is used, 61 variables when it is not.

5.2.1.2 Omitting incomplete or strange data

Having selected the interesting variables the second step was to extract all the complete cases from the dataset, as aborted cases not would give any useful information for the CCBR system. Another problem in the dataset was that the values supplied for patient diagnosis, which is used as classification, not always was standardized. The values mainly follow the International Statistical Classification of Diseases and Related Health Problems 10th Revision (ICD) (WHO et al. (2007)), but some values use old versions of ICD, or simply textual expressions. For some of the data several classifications are supplied. This was solved by simply excluding all data where the values for classification could not be recognized as an ICD classification, as it would require competent health personnel to reclassify this data using ICD.

5.2.1.3 Resulting data

The processing of the data results in 649 remaining cases, all qualified by the requirements decided. The total number of different features is 97, but for some cases only 61 of them is used. Most of the features in the PAT-C data have scaled values, e.g. from 0 to 10. The effect of this on the current TCBR framework is that 0 and 1 on the scale is dealt with as just as different as 0 and 10, even though the values in fact are quite similar. The feature used for classification is *db_primdiagnosis*, and based on the value restrictions for this feature the data includes 88 possible different classifications.

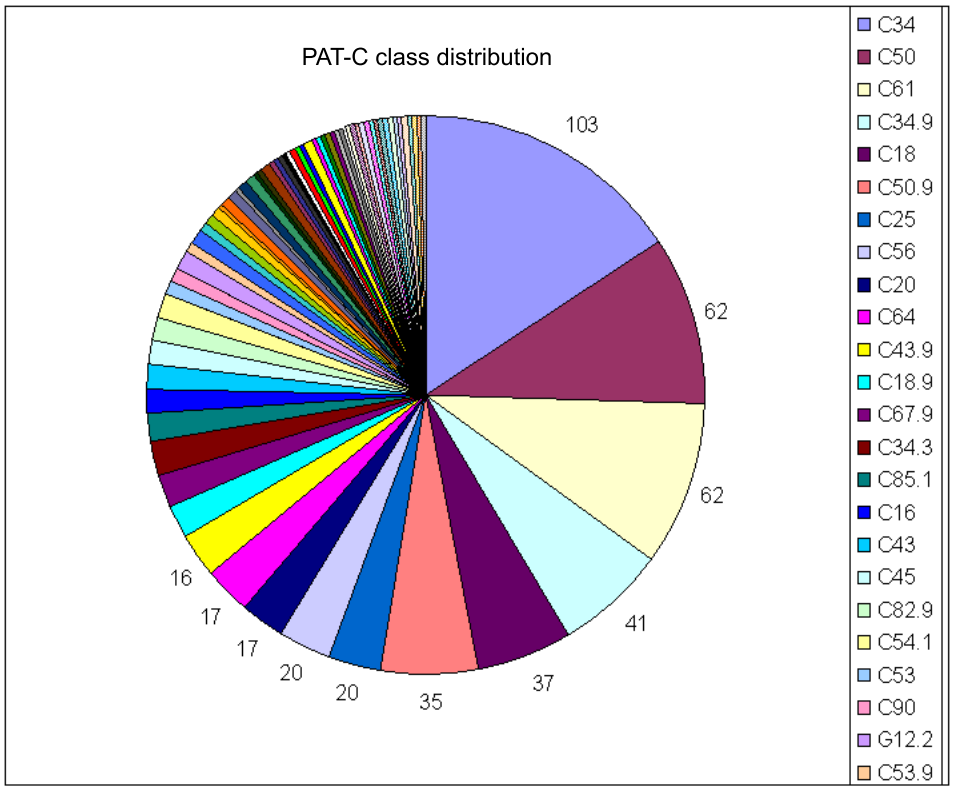


Figure 5.1: Pat-C class distribution.

An analysis of the distribution in the data reveals some interesting properties. As seen in Figure 5.1 most of the classifications only include a small number of cases. There are 40 classifications only including one case, making them unsuited for LOOCV testing, as it would be impossible to find a case with the correct classification. The largest class consist of 16% of the cases, and the six largest classes consist of as much as 53% of the cases, which shows that the cases are very unevenly distributed.

Another analysis shows that the most similar case can serve as the correct solution for only 46% of the cases. The most similar correct case exceed a threshold value of 0.5 for 94% of the cases, only leaving the 6% consisting of the 40 cases mentioned, that never can be classified correct. A threshold value of 0.6 is exceeded by 87%, 0.7 by 32%, and 0.8 by 4% of the cases. This shows that a threshold value higher than 0.6 removes the possibility of finding an acceptable solution for 68% of the cases, independent of how low the minimum rank for an acceptable solution is set. The application case retrieval threshold is thus set to 0.6 for the PAT-C data during testing.

5.2.2 Data from UCI

The UCI machine learning repository is a repository of databases, domain theories and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. As a wider basis for evaluation 2 of the datasets from this repository are selected in addition to the data from PAT-C. These datasets are well known and have been used successfully by e.g. Gu and Aamodt (2006a); Gu et al. (2005), and could compensate for risk involved in the PAT-C data being formerly unknown and potentially not well suited to show how the system performs. The selected datasets are the *lung-cancer* dataset, and the *soybean-large* dataset.

5.2.2.1 Lung cancer

The lung cancer dataset has 32 cases and 56 features, where *class* is the classification feature with the values 1, 2, and 3. All the features are symbolic, and should not be affected by the fact that the TCBR framework only supports discrete comparison of feature values. There are some missing values in the dataset, which results in some cases not having all features. The class distribution analysis for the lung cancer dataset (see Figure 5.2) shows that the 3 different possible classifications have an even distribution of the cases, and there are no problems by any cases not being suitable for LOOCV testing.

An analysis of the most similar case for each case in the dataset shows that the most similar case is an acceptable solution for 72% of the cases. The most similar correct case exceeds a threshold value of 0.6 for 100% of the cases, but the most similar correct case does never exceed a threshold value of 0.7 for any of the cases. The application case retrieval threshold value is thus set to 0.6 during testing with this dataset.

5.2.2.2 Soybean large

The soybean large dataset has 307 cases and 36 features, where *class* is the classification feature. There are 18 different classifications. Also this dataset has only symbolic features, and some missing values.

The class distribution analysis for the soybean large dataset (see Figure 5.3) shows that the four largest classes consist of more than 50% of the dataset. One class consist of only one case, and can not be used in LOOCV, the rest of the dataset should be applicable.

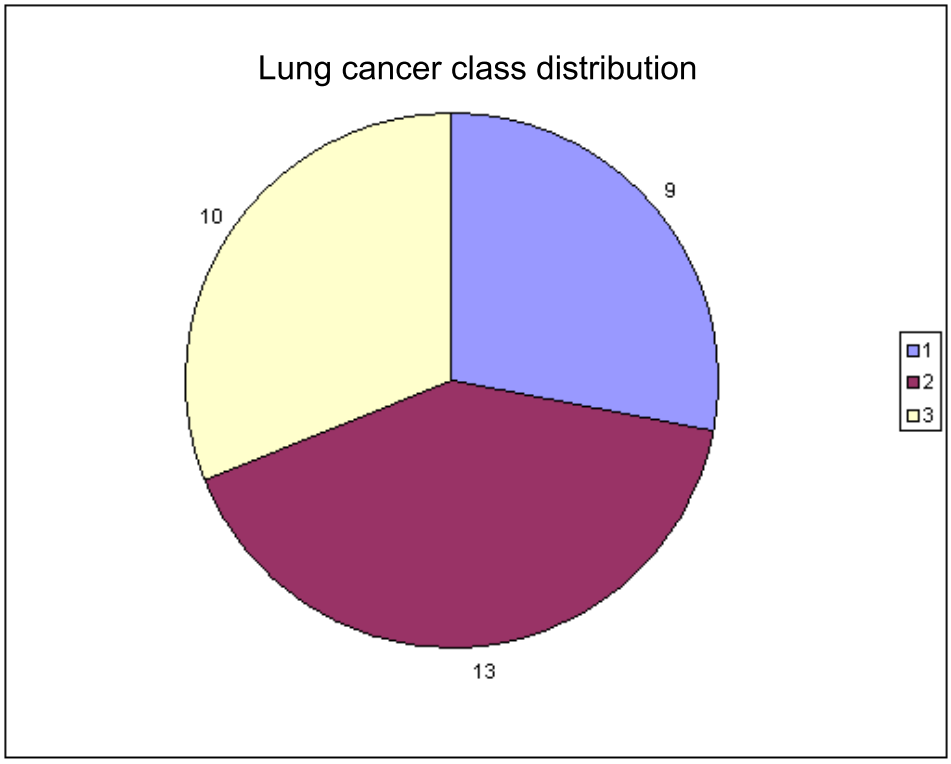


Figure 5.2: Lung cancer class distribution.

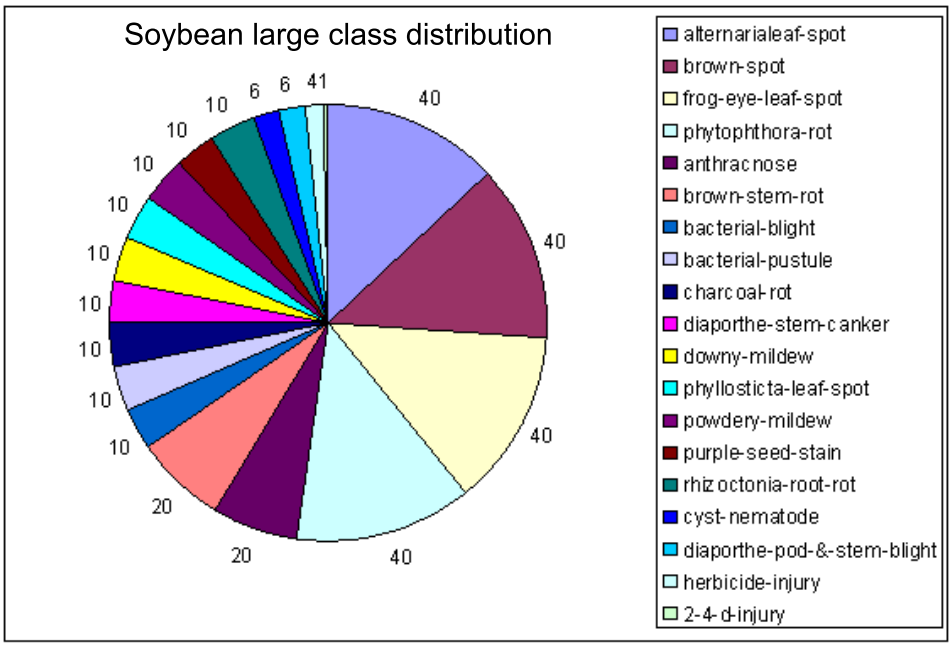


Figure 5.3: Soybean large class distribution.

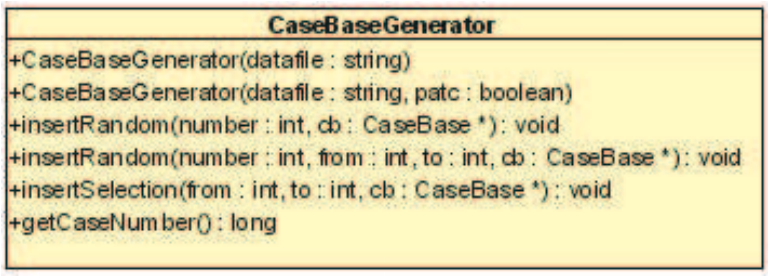


Figure 5.4: Class diagram for the case base generator.

An analysis of the most similar case for each case in the dataset shows that the most similar case is an acceptable solution for 74% of the cases. The most similar correct case exceeds a threshold value of 0.6 for 98%, 0.7 for 95%, and 0.8 for 59% of the cases. The application case retrieval threshold value is thus set to 0.7 during testing with this dataset.

5.2.3 Case base generator

The PAT-C dataset was meant to serve as test data not only for this thesis, but for both the diploma thesis by Kokkersvold (2007), and the project thesis by Houeland (2007). Both theses are connected to the EPCRC project and use the TCBR framework, so in order to share the processed dataset with these other theses easily, an interface for filling TCBR case bases with the data was implemented (see Figure 5.4).

The current version of the TCBR framework does not include any mean for serialising a case base for later use, and thus all cases must be added in the initial program load. As it is not a part of this thesis to implement such means, a simple way of solving this is to read from the dataset file in the constructor of the case base generator. The case base generator has two constructors, both accepting the path to the dataset file as a string. The simplest constructor assumes that the file contains PAT-C data, while the other one accepts a Boolean value to assign whether the data is from PAT-C or from any other source. PAT-C data is treated as described in Section 5.2.1.1 and Section 5.2.1.2. If the data file is marked to contains other data (such as from the UCI repository), it is accepted with missing values given as white spaces, without any modifications by the case base generator.

In the TCBR framework all cases must belong to a case base, but to create a more flexible interface towards the datasets a new internal representation of cases was implemented. All the data about the cases and their features were stored independent of the TCBR system in the case base generator, and three methods for inserting the data into a TCBR case base were im-

Table 5.1: Case base generator example

```
CaseBase* caseBase = new CaseBase("MyBase");
CaseBaseGenerator* cbg = new CaseBaseGenerator("C:\\pat-c.dat");
cbg.insertRandom(10, caseBase);
cbg.insertRandom(10, 100, 200, caseBase);
cbg.insertSelection(0, 10, caseBase);
```

plemented:

- There are two *insertRandom()* methods, selecting a given number of random cases from the dataset and inserting them into the case base specified. If the from and to parameters are specified, the random cases are selected within that section of the dataset. Both methods selects randomly without replacement. If the number of cases given is larger than or equal to the section given, or to the size of the dataset, the *insertSelection()* method assume the task.
- The *insertSelection()* method selects all cases in a given section of the dataset specified by the from and to parameters, and inserts them into the case base specified. This method always returns the same cases given the same parameters and dataset, making it possible to repeat tests under the same conditions.

In Table 5.1 an example is given: The case base and the case base generator is first constructed. The first *insertRandom()* method inserts 10 random cases to the case base. The second *insertRandom()* method inserts another 10 random cases, but this time all of them are found in the section from case number 100 to case number 200. The *insertSection()* method simply inserts the first 10 cases in the dataset into the case base.

5.3 Test results

The test results have been organized in tables for each of the datasets. For both cycles the dialogue length without learning (DLNL), the dialogue length with learning (DLL), and the number of stored dialogue cases (CASE) are recorded, and the average DLNL and DLL for the cycles have been computed. The last to columns of the table concerns the dialogue length (DL) and the dialogue case base (DCB). For the dialogue length the concept of length reduction (LR) is introduced; simply finding the difference between the average DLNL and DLL as a percentage of the average DLNL. A higher percentage indicates a larger reduction in dialogue length do to dialogue learning. For the dialogue case base the interesting matter to investigate is the growth reduction (GR). The growth of the dialogue case base is defined

Table 5.2: Result table for lung cancer dataset.

Cycle1			**Cycle2**			**Avg.**		**DL**	**DCB**
DLNL	**DLL**	**CASE**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**LR (%)**	**GR (%)**
19.16	12.34	31	17.75	16.41	49	18.45	14.38	22.10	40.63
20.13	17.22	24	16.47	10.91	45	18.30	14.06	23.14	9.38
19.25	14.00	27	15.81	14.41	49	17.53	14.20	18.98	15.63
20.31	15.91	22	20.69	11.53	42	20.50	13.72	33.08	6.25
19.91	14.53	28	18.16	10.63	49	19.03	12.58	33.90	21.88
15.06	12.19	28	15.28	13.09	43	15.17	12.64	16.68	40.63
19.00	14.72	25	19.94	12.19	44	19.47	13.45	30.90	18.75
18.75	14.34	28	22.53	11.16	43	20.64	12.75	38.23	40.63
16.44	12.63	24	21.13	11.53	43	18.78	12.08	35.69	15.63
18.38	11.78	28	20.97	13.09	48	19.67	12.44	36.78	25.00
15.25	12.28	22	17.06	11.91	39	16.16	12.09	25.15	15.63
17.38	16.13	21	13.50	14.19	43	15.44	15.16	1.82	-3.13
19.75	14.44	29	22.00	13.13	44	20.88	13.78	33.98	43.75
17.69	13.84	26	16.72	11.00	50	17.20	12.42	27.79	6.25
20.03	13.44	27	20.78	12.88	50	20.41	13.16	35.53	12.50
					Avg.	**18.51**	**13.26**	**27.58**	**20.63**

as the percentage of the test cases stored as new dialogue cases. The growth reduction is thus the difference between the dialogue case base growth in cycle 1 and cycle 2. If there is no reduction in growth, the dialogue retain strategy is ineffective. The growth reduction will not approach 100% by the second cycle due to the random selection of problem descriptions from the test cases, but both the growth and the growth reduction will steady over time if no new test cases is introduced.

5.3.1 Lung cancer

The lung cancer dataset is the smallest dataset with 32 cases with up to 56 features, all classified as one of 3 classes. Because of the random factors involved the test results are different for every run. An average over 15 runs show a dialogue length reduction of 27.58%, and a dialogue case base growth reduction of 20.63%. In other words; for the lung cancer dataset the dialogue learning has a significant positive effect on the dialogue length, and the retaining of dialogue cases is maintainable.

Table 5.3: Result table for part 1 of the soybean large dataset.

Cycle1			**Cycle2**			**Avg.**		**DL**	**DCB**
DLNL	**DLL**	**CASE**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**LR (%)**	**GR (%)**
6.97	7.29	80	7.23	6.52	126	7.10	6.91	2.75	34.00
6.99	7.52	77	6.83	6.68	120	6.91	7.10	-2.75	34.00
6.80	6.07	73	6.47	6.35	122	6.64	6.21	6.41	24.00
6.31	6.58	71	7.07	6.75	123	6.69	6.67	0.37	19.00
6.92	6.46	71	7.13	6.46	115	7.03	6.46	8.04	27.00
7.29	6.68	74	6.51	6.75	119	6.90	6.72	2.68	29.00
6.80	6.72	65	6.89	6.67	117	6.85	6.70	2.19	13.00
7.17	7.00	70	6.88	7.00	114	7.03	7.00	0.36	26.00
7.03	6.78	69	7.38	6.77	125	7.21	6.78	5.97	13.00
7.22	7.16	70	6.80	6.97	124	7.01	7.07	-0.78	16.00
					Avg.	**6.93**	**6.76**	**2.52**	**23.50**

5.3.2 Soybean large

The soybean large dataset is a much larger dataset than the lung cancer dataset; 307 cases with up to 36 features, distributed over 18 possible classifications. To ease the computational effort needed to run the tests the dataset is split into tree parts. The first 100 cases is used as part 1, cases 101 to 200 as part 2, and cases 201 to 300 as part 3.

5.3.2.1 Part 1

In the first part the cases are distributed over 6 solutions. Averaged over 10 test runs the dialogue case base, as for the lung cancer dataset, shows to be maintainable, but the dialogue length reduction gives moderate results. For two of the test runs the dialogue length is increased with dialogue learning, and the dialogue length is on an average only reduced by 2,52%.

5.3.2.2 Part 2

In the second part the cases are also distributed over 6 solutions. Averaged over 10 test runs the dialogue case base growth reduction shows similar results as for part 1. The dialogue length reduction gives even worse results than for part 1; the dialogue length is on an average increased by 1.23%.

Table 5.4: Result table for part 2 of the soybean large dataset.

Cycle1			**Cycle2**			**Avg.**		**DLDL**	**DCB**
DLNL	**DLL**	**CASE**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**LR (%)**	**GR (%)**
5.88	6.37	90	6.02	6.55	162	5.95	6.46	-8.57	18.00
6.07	6.14	92	6.08	6.22	168	6.08	6.18	-1.73	16.00
5.56	6.02	89	6.42	5.87	157	5.99	5.95	0.75	21.00
5.97	6.04	89	6.13	6.56	156	6.05	6.30	-4.13	22.00
6.35	5.91	91	6.12	6.04	159	6.24	5.98	4.17	23.00
6.47	6.71	96	5.93	6.11	164	6.20	6.41	-3.39	28.00
6.57	6.21	87	6.06	6.01	160	6.32	6.11	3.25	14.00
6.20	6.13	86	6.36	6.27	156	6.28	6.20	1.27	16.00
5.98	6.06	88	5.89	6.27	165	5.94	6.17	-3.88	11.00
5.97	6.12	91	6.01	5.87	159	5.99	6.00	-0.08	23.00
					Avg.	**6.10**	**6.17**	**-1.23**	**19.20**

5.3.2.3 Part 3

In the third part the cases are distributed over 5 solutions. Averaged over 10 test runs the dialogue case base growth reduction shows similar results, dialogue length reduction on the other hand, gives better results than for the former parts; the dialogue length is on an average decreased by 6.48%.

The dialogue lengths for all three parts are quite short with regard to the fact that the maximum dialogue length is 36. This implies that the classes are easy to distinguish, and that the CCBR process solves the problem in a matter of few dialogue steps, regardless of the effect of the dialogue learning. Dialogue learning gives the best results in part 3, where the dialogue length without learning on average is significant higher than for part 1 and 2, which shows that dialogue learning has better effect with longer dialogues.

5.3.3 PAT-C

The PAT-C dataset is the largest dataset with 649 cases with up to 97 features, distributed over 88 possible classifications. The computational effort needed to run tests over the whole dataset is increased tremendously compared to the soybean large dataset due to the cases having almost three times as many features. The testing was performed by randomly selecting small subsets of cases, giving a large variety of different compositions of cases. Due to the large amount of classes consisting of only a few cases, the minimum

Table 5.5: Result table for part 3 of the soybean large dataset.

Cycle1			**Cycle2**			**Avg.**		**DL**	**DCB**
DLNL	**DLL**	**CASE**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**LR (%)**	**GR (%)**
8.46	7.82	80	7.17	6.97	132	7.82	7.40	5.37	28.00
7.12	6.64	87	6.96	6.03	153	7.04	6.34	10.01	21.00
6.69	7.43	89	7.89	6.84	145	7.29	7.14	2.13	33.00
8.46	6.37	80	7.05	6.75	140	7.76	6.56	15.41	20.00
7.19	7.86	79	7.56	7.13	142	7.38	7.50	-1.63	16.00
7.39	7.01	80	8.59	7.66	135	7.99	7.34	8.20	25.00
8.79	7.57	88	7.79	7.52	146	8.29	7.55	8.99	30.00
7.44	7.52	85	7.91	7.24	147	7.68	7.38	3.84	23.00
7.85	7.14	84	8.01	7.54	154	7.93	7.34	7.44	14.00
7.69	6.81	81	7.52	7.64	155	7.61	7.23	5.00	7.00
					Avg.	**7.68**	**7.17**	**6.48**	**21.70**

number of cases for an included class was set to 5. All cases classified as a class not satisfying this limit were deleted from the case base before the simulations started. A different number of cases and classes represented in the case base for each of the tests introduced the need for two extra columns in the result table, holding these variables. The number of dialogue cases stored in each of the cycles, in proportion to the total number of cases, is very low compared to the results for the other datasets. This is because only 46% of the cases in the complete dataset are classified correct by their most similar case, which give a high probability of dialogues failing and not being retained in the dialogue system. This large amount of failed dialogues have a negative impact on the average dialogue length for each test, but most of them still result in relatively short dialogue length averages. This implies that the dialogues which succeed do so in a matter of few questions. Despite of this, the dialogue length seems to be reduced by dialogue learning, at least for the dialogues that are successful.

Table 5.6: Result table for the PAT-C dataset.

ACB		**Cycle1**			**Cycle2**			**Avg.**		**DL**	**DCB**
CASES	**SOLUTIONS**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**CASE**	**DLNL**	**DLL**	**LR (%)**	**GR (%)**
19	3	25,32	22,79	12	26,84	20,74	22	26,08	21,76	17	11
24	3	40,71	14,38	21	20,54	10,54	33	30,63	12,46	59	38
19	3	25,95	22,79	15	34,79	22,68	27	30,37	22,74	25	16
37	5	25,59	24,38	30	24,49	29,30	48	25,04	26,84	-7	32
21	3	40,95	31,05	11	40,90	18,76	19	40,93	24,90	39	14
27	4	33,33	27,56	20	33,52	22,41	32	33,43	24,98	25	30
19	3	48,37	32,89	11	44,37	38,47	18	46,37	35,68	23	21
38	5	48,34	30,26	22	41,53	30,03	35	44,93	30,14	33	24
24	4	28,83	20,08	19	34,33	22,63	27	31,58	21,35	32	46
24	3	17,25	19,83	16	29,29	17,50	29	23,27	18,67	20	13
32	4	29,91	29,47	20	43,22	25,69	40	36,56	27,58	25	0
26	3	16,85	19,00	21	20,54	18,12	28	18,69	18,56	1	54
24	2	14,25	14,08	16	19,75	17,00	21	17,00	15,54	9	46
35	4	23,03	32,83	22	26,71	27,86	35	24,87	30,34	-22	26
32	4	10,28	13,16	26	13,31	10,81	35	11,80	11,98	-2	53
32	4	10,09	13,38	23	10,47	10,53	37	10,28	11,95	-16	28
24	4	23,00	25,08	15	28,00	22,21	25	25,50	23,65	7	21
35	5	37,77	38,94	16	36,57	26,51	39	37,17	32,73	12	-20
44	5	23,59	22,18	32	23,41	22,36	60	23,50	22,27	5	9
78	4	20,33	18,83	55	16,83	17,77	96	18,58	18,30	2	18
							Avg.	**27,83**	**22,62**	**14**	**24**

Chapter 6

Discussion

This final chapter concerns the conclusions drawn based on this thesis, a number of theoretical and practical issues that have arisen during the design and development process, and some ideas for future work.

6.1 Conclusions

This thesis contains a study of methods for reducing the number of questions asked in a questionnaire. This includes CAT with IRT, but is mainly focused on CCBR with dialogue learning. Methods for similarity measure and question ranking were reviewed. The NaCoDAE system (Aha and Breslow (1997a)) and the framework for dialogue learning in CCBR developed in Gu and Aamodt (2006a) were also examined. The thesis also includes an introduction to a TCBR framework; a CBR core system used as a basis for the system designed in the thesis.

Based on this background knowledge, methods for building a CCBR system with dialogue learning were chosen, and the system was designed as presented in Section 4.1. The concrete result of this thesis is the system whose implementation is presented in Section 4.2. The system is a prototype including a test environment using user simulation, a CCBR engine, and a dialogue system.

The testing of the implemented system has been done through well-established methods for evaluation of CCBR systems, with elements such as LOOCV and ablation evaluation. Part of the testing was performed using well-known data from the UCI repository (D.J. Newman and Merz (1998)), part using new data from PAT-C (Pain et al. (2006)).

Evaluating the system has shown that dialogue learning can shorten the

number of questions asked; especially if there are a large number of questions available and the cases are hard to distinguish.

6.2 Theoretical and practical issues

In this section some issues that have arisen during the design and development processes are reviewed.

6.2.1 Practical problem descriptions

As described in Section 4.1.3.1 it can be problematic to use query biased case retrieval in systems where there is a need to classify the query automatically. It is required a certain amount of features set as a problem description before the CCBR process starts. This is in contrast to the paper based questionnaire where the patient answers questions without any information given in advance. To address this problem, a possibility would be to let the CCBR system start asking the highest ranked questions and build an internal problem description. The retrieval process should not be activated before a sufficient problem description is given, thus no solutions would be presented in the first steps of the dialogue.

6.2.2 Global and local approximation of the target function

When it comes to the goal of finding the best question to ask, the use of global feature ranking gives a global approximation of the target function. The SimpleEACH algorithm (Table 4.1) considers the case base and sets the feature weights before any simulations are started. The current dialogue is thus not taken into consideration, as it would have been when using the computational demanding information gain method (Section 3.3.3.1). The use of dialogue learning changes this and gives a combination of both global and local approximation. At every step of the dialogue the weights from the global approximation is sent to the dialogue system which takes the current dialogue into consideration and adjusts the weights according to the most similar dialogue stored, giving the local approximation typical for CBR systems.

These observations can be used to explain why some datasets get better results using dialogue learning, while other datasets hardly are improved by the method. If the dataset is easily covered by a global approximation, the CCBR system will perform well regardless of the dialogue system. As a

consequence of the fact that the local and global approximations give the same results, the dialogue system will strengthen the weights, but not in a manner that changes the ranking of the features.

6.2.3 The most general dialogue

The goal of storing dialogue cases is to reuse them in order to present smarter questions and solutions, and thus reducing the dialogue length. In order to keep the dialogue case base maintainable only the most general dialogues are stored, following the definition in equation (2.9). Despite this, it is not clear when to stop generalizing the dialogues. In theory, the effect of the dialogue learning is shorter dialogues, which in many cases typically is stored as more general dialogues that replace formerly stored dialogues. In addition, an automated agent as used in this thesis is not able to understand when a solution is struck in a matter of very few questions just by chance. This might result in storing dialogues with only one or two steps that are very general in the way they cover a large variety of possible endings for the dialogues, but they are not of any use for aiding longer dialogues. One might say that the automated agent should be able to detect such "fortunate" results, but as the goal is to reduce the number of questions asked it seems difficult to set a minimum of questions needed to say that the dialogue is interesting enough to store.

6.2.4 Variance in the test results

The use of average dialogue lengths in the test results, without performing a variance analysis on the results, may in some cases give a wrong impression of how well the system performs. In particular some of the results from the PAT-C data show this, where over half the dataset is wrongly classified by the most similar case and thus does not end in successful dialogues. A lot of the cases from the other half of the dataset are classified correctly by only the problem description and a few features, and altogether this gives some strange results that at first sight looks good. A typical sign of this situation is that very few dialogues are stored in the first cycle, due to the conversations ending in failure.

6.2.5 Scalability

A practical problem with the system is that although the representation of large case bases not is a problem, it becomes a problem when retrieving similar cases. The system performs well on the smaller datasets such as lung cancer, and the parts of soybean large, but for the complete soybean

large dataset or the PAT-C dataset the retrieval process is so computational demanding that it might not be feasible to expect a patient to wait for it to complete. This is especially a problem when a solution is hard to find and a lot of questions are asked, as this results in more features to compare during retrieval.

6.3 Future work

The implementation in this thesis regards implementing a CCBR system with dialogue learning using the CBR core of the TCBR framework. In order to achieve better results from the dialogue learning this framework must be improved with regard to the value types accepted for features. The introduction of continuous values could increase the performance on datasets such as from PAT-C dramatically.

With regards to evaluation, a variance analysis should be performed on the test results in order to expose potential problems regarding the average values.

One of the most interesting topics to investigate further is the use of query biased similarity methods in a CCBR system where classification is performed automatically. Finding criteria for deciding when a solution is adequate is problematic; the similarity measures might be high from the beginning of the conversation and a threshold value will thus not be enough, at the same time shorter conversations are preferred so it is not desirable to demand a minimum length for the conversation.

Another interesting topic, tightly connected to the former, is regarding the problems that occur when the automatic system generates very short conversations that are stored in the dialogue system as more general than the existing dialogues, even though those would be more useful for reuse.

Bibliography

Aamodt, A. (1994). Explanation-driven case-based reasoning. In *Topics in Case-based reasoning*, pp. 274–288. Springer Verlag.

Aamodt, A. (2004). Knowledge-intensive case-based reasoning in creek. In *Advances in Case-Based Reasoning*, pp. 1–15. Springer Berlin.

Aamodt, A. and E. Plaza (1994, March). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1), 39–59.

Aha, D. W. (1991). Case-based learning algorithms. In *DARPA Case-Based Reasoning Workshop*, pp. 147–158. Morgan Kaufmann.

Aha, D. W. and L. Breslow (1997a). Refining conversational case libraries. In *ICCBR*, pp. 267–278.

Aha, D. W. and L. A. Breslow (1997b). Nacodae: Navy conversational decision aids environment. Technical Report AIC-97-018.

Aha, D. W., L. A. Breslow, and H. Muñoz-Avila (2001). Conversational case-based reasoning. *Applied Intelligence* (14), 9–32.

Aha, D. W., T. Maney, and L. A. Breslow (1998). Supporting dialogue inferencing in conversational case-based reasoning. *European Workshop on Case-Based Reasoning*, 262–273.

Bareiss, R. (1989). The experimental evaluation of a case-based learning apprentice. In *Proceedings of the Case-Based Reasoning Workshop*, pp. 162–167. Morgan Kaufmann Publishers Inc.

Bjorner, J. B. and J. E. W. Jr. (1998). Using modern psychometric methods to measure health outcomes. *Med Outcomes Trust Monitor* (3), 12–16.

D.J. Newman, S. Hettich, C. B. and C. Merz (1998). Uci repository of machine learning databases.

EPCRC (2005). Fp6-2005-lifescihealth-6, combating major diseases, combating cancer, specific targeted research project. Technical report.

Gu, M. (2006). *Knowledge-Intensive Conversational Case-Based Reasoning in Software Component Retrieval*. Ph. D. thesis.

Gu, M. and A. Aamodt (2006a, May 11-13). Dialog learning in conversational cbr. In G. C. J. Sutcliffe and R. G. Goebel (Eds.), *Proceedings of the 19th International FLAIRS Conference*, pp. 358–363. AAAI Press.

Gu, M. and A. Aamodt (2006b, 4 - 7, Septemper). Evaluating cbr systems using different data sources: A case study. In *accepted by the 8th European Conference on Case-Based Reasoning*, Volume 4106 of *Lecture Notes in Artificial Intelligence*, pp. 121–135. Springer Verlag.

Gu, M., X. Tong, and A. Aamodt (2005, August). Comparing similarity calculation methods in conversational cbr. In D. Zhang, T. M. Khoshgoftaar, and M.-L. Shyu (Eds.), *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration*, pp. 427–432. IEEE Press.

Hendler, J., K. Stoffel, and M. Taylor (1996). Advances in high performance knowledge representation. Technical Report CSTR -3672, University of Maryland Institute for Advanced Computer Studies Dept. of Computer Science.

Houeland, T. G. H. (2007). Learning of feature weigths in conversational cbr. Master's thesis, Norwegian University of Science and Technology.

Kira, K. and L. A. Rendell (1992). The feature selection problem: Traditional methods and a new algorithm. In *AAAI*, pp. 129–134.

Kokkersvold, I. (2007). Conversational cbr for an adaptive qa system. Master's thesis, Norwegian University of Science and Technology.

Kolodner, J. (1993). *Case-based reasoning*. Morgan Kaufmann Publishers Inc.

McHorney and A. Colleen (1997). Generic health measurement: Past accomplishments and a measurement paradigm for the 21st century. *Ann Intern Med 127*(8), 743–750.

McLaren, B. M. (2003). Extensionally defining principles and cases in ethics: an ai model. *Artificial Intelligence Journal 150*, 145–181.

Mitchell, T. M. (1997). *Machine learning*. The McGraw-Hill Companies, Inc.

Pain, T., N. U. o. S. Palliation Research Group, Faculty of Medicine, and N. Technology (NTNU), Trondheim (2006). Protocol for the pat-c prototype: pain, physical and cognitive function.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*(1), 81–106.

Revicki, D. A. and D. F. Cella (1997). Health status assessment for the twenty-first century: item response theory, item banking and computer adaptive testing. *Quality of Life Research 6*(6), 595–600.

Richter, M. M. and S. Wess (1993). Similarity, uncertainty and case-based reasoning in patdex.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning* (6), 251–276.

Schmitt, S. (2002). simvar: A similarity-influenced question selection criterion for e-sales dialogs. *Artif. Intell. Rev. 18*(3-4), 195–221.

Wainer, H. (2000). *Computerized adaptive testing 2nd: A primer*. Lawrence Erlbaum Associates.

Ware, J. E., J. B. Bjorner, and M. Kosinski (2000). Practical implications of item response theory and computerized adaptive testing. a brief summary of ongoing studies of widely used headache impact scales. *Medical Care 38*(9), 73–82.

WHO, D. G. I. of Medical Documentation, and Information) (2007). International statistical classification of diseases and related health problems 10th revision.

Yang, Q. and J. Wu (2001). Enhancing the effectiveness of interactive case-based reasoning with clustering and decision forests. *Applied Intelligence 12*, 49–64.

Appendix A

Analysis of the datasets

To gain knowledge about the nature of the datasets a process was carried out for finding, for each case in the dataset, the solution and the similarity value of the most similar case, and the similarity value of the most similar correct case (with the same solution as the base case). This gives a basis for finding the number of cases that are misclassified by their most similar case, and also the probability of finding an acceptable solution given a specific similarity threshold.

A.1 The PAT-C dataset

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #1	C50	C50	0,679623	0,679623
Case #2	C50	C50	0,726717	0,726717
Case #3	C50	C50	0,741754	0,741754
Case #4	C50	C50	0,741754	0,741754
Case #5	C50	C50	0,719813	0,719813
Case #6	C50	C50	0,701927	0,701927
Case #7	C61	C61	0,719813	0,719813
Case #8	C61	C61	0,684666	0,684666
Case #9	C61	C61	0,684666	0,684666
Case #10	C61	C61	0,713259	0,713259
Case #11	C61	C61	0,701065	0,701065
Case #12	C61	C61	0,713259	0,713259
Case #13	C26	C26	0,796132	0,796132

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #14	C26	C34	0,796132	0,796132
Case #15	C61	C34	0,627121	0,608443
Case #16	C61	C34	0,665549	0,66117
Case #17	C64	C34	0,65693	0,648833
Case #18	C64	C61	0,670077	0,66117
Case #19	C64	C64	0,684666	0,684666
Case #20	C64	C50.9	0,641198	0,637537
Case #21	C25	C25	0,630504	0,630504
Case #22	C64	C64	0,641807	0,641807
Case #23	C64	C64	0,641807	0,641807
Case #24	C61	C61	0,66117	0,66117
Case #25	C61	C61	0,66117	0,66117
Case #26	C61	C34	0,637537	0,633975
Case #27	C61	C61	0,648833	0,648833
Case #28	C61	C61	0,670077	0,670077
Case #29	C61	C61	0,670077	0,670077
Case #30	C16	C38	0,719813	0,695369
Case #31	C16	C61	0,701065	0,695369
Case #32	C16	C16	0,670077	0,670077
Case #33	C16	C16	0,665549	0,665549
Case #34	C16	C16	0,684666	0,684666
Case #35	C16	C16	0,684666	0,684666
Case #36	C18	C92	0,65282	0,620601
Case #37	C21	C61	0,620601	0,58216
Case #38	C18	C18.9	0,679623	0,641198
Case #39	C18.9	C18	0,679623	0,665549
Case #40	C25	C34	0,695369	0,684666
Case #41	C25	C61	0,741754	0,719813
Case #42	C64	C64	0,70702	0,70702
Case #43	C64	C34	0,741754	0,70702
Case #44	C61	C61	0,648833	0,648833
Case #45	C61	C61	0,648833	0,648833
Case #46	C64	C82.9	0,66117	0,65693
Case #47	C50	C50	0,644961	0,644961
Case #48	C50	C50	0,695369	0,695369
Case #49	C08	C64	0,633975	0
Case #50	C20	C64	0,689909	0,674764
Case #51	C50	C34	0,623821	0,620601

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #52	C50	C50	0,617457	0,617457
Case #53	C61	C61	0,66117	0,66117
Case #54	C61	C61	0,66117	0,66117
Case #55	C18	C63	0,620601	0,620601
Case #56	C18	C61	0,633975	0,627121
Case #57	C61	C61	0,661311	0,661311
Case #58	C61	C61	0,661311	0,661311
Case #59	C61	C16	0,65282	0,65282
Case #60	C18	C18	0,758828	0,758828
Case #61	C18	C18	0,796132	0,796132
Case #62	C18	C18	0,777424	0,777424
Case #63	C18	C18	0,832634	0,832634
Case #64	C18	C18	0,768338	0,768338
Case #65	C18	C18	0,832634	0,832634
Case #66	C61	C61	0,648833	0,648833
Case #67	C61	C61	0,665549	0,665549
Case #68	C61	C61	0,665549	0,665549
Case #69	C61	C61	0,665549	0,665549
Case #70	C61	C70	0,637537	0,633975
Case #71	C20	C12.2	0,670077	0,627121
Case #72	C43	C18.9	0,630223	0,576198
Case #73	C50	C50	0,637537	0,637537
Case #74	C50	C50	0,633975	0,633975
Case #75	C18	C50	0,641198	0,617457
Case #76	C55	C18	0,637537	0,59206
Case #77	C77	C34	0,633975	0
Case #78	C92	C18	0,65282	0,511912
Case #79	C56	C43.9	0,65282	0,620601
Case #80	C55	C34.9	0,644961	0,59206
Case #81	C18	C25	0,701065	0,684666
Case #82	C18	C43	0,66117	0,644961
Case #83	C50	C49	0,627121	0,620601
Case #84	C43	C61	0,726717	0,65693
Case #85	C25	C20	0,637537	0,620601
Case #86	C71	C50.9	0,641198	0,528179
Case #87	C50	C50	0,623821	0,623821
Case #88	C54	C61	0,644961	0
Case #89	C43	C09	0,689909	0,614384

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #90	C48	C21	0,65282	0
Case #91	C50	C34.3	0,648833	0,630504
Case #92	C20	C20	0,65693	0,65693
Case #93	C20	C20	0,65693	0,65693
Case #94	C34	C50	0,630504	0,627121
Case #95	C34	C34	0,644961	0,644961
Case #96	C34	C20	0,66117	0,648833
Case #97	C61	C34	0,713259	0,713259
Case #98	C61	C61	0,758828	0,758828
Case #99	C25	C91	0,637537	0,630504
Case #100	C61	C38	0,713259	0,713259
Case #101	C61	C34	0,641198	0,637537
Case #102	C25	C18	0,670077	0,648833
Case #103	C90	M89	0,637537	0,627121
Case #104	C18	C18	0,674764	0,674764
Case #105	C34	C18	0,641198	0,630504
Case #106	C25	C61	0,641198	0,620601
Case #107	C90	C61	0,734014	0,633975
Case #108	C25	C61	0,734014	0,719813
Case #109	C61	C34	0,758828	0,758828
Case #110	C34	C85.1	0,641807	0,630223
Case #111	C25	C64	0,679623	0,641198
Case #112	C50	C43.9	0,684666	0,644961
Case #113	C50	C25	0,644961	0,614384
Case #114	C34	C26	0,7118	0,701927
Case #115	C61	C64	0,670077	0,641198
Case #116	C50.9	C50	0,623821	0,59206
Case #117	C50.9	C20	0,614384	0,59206
Case #118	C18	C45	0,627121	0,614384
Case #119	C71	C26	0,746963	0,5495
Case #120	C50.9	C56	0,630504	0,623821
Case #121	C64	C08	0,589515	0,579797
Case #122	C61.9	C50	0,648833	0,633975
Case #123	C18	C34	0,670077	0,66117
Case #124	C85	C38	0,7118	0,654489
Case #125	C50	C61	0,637537	0,627121
Case #126	C85.9	C53	0,65693	0
Case #127	C34.9	C34.9	0,647998	0,647998

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #128	C80	C34	0,695369	0,648833
Case #129	C24.9	C34	0,665549	0,529766
Case #130	C34	C34	0,66117	0,66117
Case #131	C34.9	C34.9	0,7118	0,7118
Case #132	C50.9	C50.9	0,641198	0,641198
Case #133	C50.9	C53.9	0,644961	0,637537
Case #134	C34.9	C34	0,647998	0,619562
Case #135	C91.1	C38	0,7118	0
Case #136	C20	C67.9	0,734014	0,674764
Case #137	C61	C82	0,648833	0,627121
Case #138	C43.9	C18	0,66117	0,623821
Case #139	C64	C43	0,66117	0,641198
Case #140	C20.9	C18	0,719813	0
Case #141	C50.9	C20	0,65693	0,633975
Case #142	C18.2	C82	0,674764	0
Case #143	C61.9	C34.9	0,637537	0,633975
Case #144	C50.9	C18	0,741754	0,679623
Case #145	C43.9	C56	0,684161	0,624787
Case #146	C71.9	C71.9	1	1
Case #147	C71.9	C71.9	1	1
Case #148	C34.9	C61	0,627121	0,602756
Case #149	C85	C34	0,726717	0,575192
Case #150	C59	C64	0,65693	0
Case #151	C34.9	C50	0,620601	0,5973
Case #152	C34	C50	0,695369	0,674764
Case #153	C34	C34	0,701065	0,701065
Case #154	C20	C20	0,726717	0,726717
Case #155	C34	C34	0,679623	0,679623
Case #156	C34	C34	0,65282	0,65282
Case #157	C34	C34	0,674764	0,674764
Case #158	C34	C34	0,674764	0,674764
Case #159	C34	C34	0,695369	0,695369
Case #160	C34	C34	0,695369	0,695369
Case #161	C43	C43.9	0,726717	0,65693
Case #162	C34	C34	0,701927	0,701927
Case #163	C34	C34	0,679623	0,679623
Case #164	C50	C50.9	0,65693	0
Case #165	C34	C34	0,846689	0,846689

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #166	C34	C61	0,758828	0,741754
Case #167	C34	C34	0,846689	0,846689
Case #168	C34	C34	0,796132	0,796132
Case #169	C34	C34	0,846689	0,846689
Case #170	C34	C34	0,846689	0,846689
Case #171	C34	C34	0,741754	0,741754
Case #172	C34	C34	0,734137	0,734137
Case #173	C34	C25.9	0,734014	0,734014
Case #174	C34	C34	0,7118	0,7118
Case #175	C34	C34	0,684161	0,684161
Case #176	C34	C25	0,719813	0,70702
Case #177	C34	C34	0,648833	0,648833
Case #178	C34	C34	0,648833	0,648833
Case #179	C34	C34.9	0,630504	0,627121
Case #180	C34	C61	0,633975	0,623821
Case #181	C34	C34	0,609688	0,609688
Case #182	C34	C25	0,623821	0,623821
Case #183	C45	C45	0,701065	0,701065
Case #184	C38	C45	0,695369	0,602756
Case #185	C45	C45	0,701065	0,701065
Case #186	C45	C45	0,695369	0,695369
Case #187	C34	C22	0,734137	0,7118
Case #188	C34	C34	0,684161	0,684161
Case #189	C34	C82	0,734014	0,726717
Case #190	C34	C34	0,695369	0,695369
Case #191	C34	C25	0,695369	0,684666
Case #192	C34	C34	0,713259	0,713259
Case #193	C34	C34	0,679623	0,679623
Case #194	C34	C34	0,741754	0,741754
Case #195	C34	C34	0,644961	0,644961
Case #196	C34	C50	0,648833	0,648833
Case #197	C34	C61	0,679623	0,665549
Case #198	C34	C56	0,661311	0,647998
Case #199	C45	C34	0,684666	0,648833
Case #200	C34	C20	0,684161	0,668501
Case #201	C34	C34	0,746963	0,746963
Case #202	C38	C39.9	0,734014	0,605569
Case #203	C34	C34	0,66117	0,66117

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #204	C34	C34	0,70702	0,70702
Case #205	C34	C34	0,637537	0,637537
Case #206	C34	C34	0,679623	0,679623
Case #207	C45	C18	0,734014	0,695369
Case #208	C34	C45	0,719813	0,70702
Case #209	C34	C61	0,713259	0,70702
Case #210	C34	C34	0,741754	0,741754
Case #211	C34	C34	0,741754	0,741754
Case #212	C34	C39.9	0,679623	0,670077
Case #213	C34	C38	0,726717	0,701065
Case #214	C34	C34	0,713259	0,713259
Case #215	C34	C34	0,741754	0,741754
Case #216	C34	C34	0,741754	0,741754
Case #217	C34	C22	0,761252	0,746963
Case #218	C34	C18	0,701065	0,665549
Case #219	C34	C61	0,734014	0,726717
Case #220	C34	C64	0,70702	0,70702
Case #221	C34	C64	0,741754	0,713259
Case #222	C34	C38	0,70702	0,70702
Case #223	C34	C34	0,674764	0,674764
Case #224	C38	C45	0,761252	0,630223
Case #225	C45	C38	0,761252	0,746963
Case #226	C45	C45	0,746963	0,746963
Case #227	C34	C34	0,734137	0,734137
Case #228	C34	C34.9	0,641198	0,627121
Case #229	C34	C82	0,719813	0,713259
Case #230	C34	C34	0,768338	0,768338
Case #231	C34	C34	0,701065	0,701065
Case #232	C50	C50.9	0,665549	0,633975
Case #233	C50	C50	0,627121	0,627121
Case #234	C50	C34	0,65282	0,644961
Case #235	C49	G12.2	0,674764	0,617457
Case #236	C64	C64	0,65693	0,65693
Case #237	C34	C45	0,684666	0,648833
Case #238	C34	C54.1	0,665549	0,644961
Case #239	C25	C34	0,63589	0,624787
Case #240	C15	C25	0,627121	0,594654
Case #241	C50	C56	0,644961	0,637537

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #242	C53	C34.9	0,627121	0,577474
Case #243	C18	C56	0,623821	0,620601
Case #244	C18	C34	0,670077	0,65693
Case #245	C56	C18	0,679623	0,633975
Case #246	C56	C25	0,641198	0,605569
Case #247	C50	C25	0,637537	0,620601
Case #248	C18	C25	0,713259	0,689909
Case #249	C56	C34	0,623821	0,611381
Case #250	C15.9	C34	0,679623	0
Case #251	C67	C82	0,65282	0,608443
Case #252	C92	C34	0,722486	0,519943
Case #253	C61	C61	0,627121	0,627121
Case #254	C61	C53	0,589515	0,584567
Case #255	C18	C34	0,695369	0,674764
Case #256	C16	C34	0,623821	0,608443
Case #257	C49	G12.2	0,644961	0,641198
Case #258	C49	G12.2	0,641198	0,641198
Case #259	C64	C50	0,644961	0,627121
Case #260	C61	C61	0,633975	0,633975
Case #261	C55	C34	0,641198	0,58216
Case #262	C61	C61	0,65693	0,65693
Case #263	C50	C50	0,633975	0,633975
Case #264	C61	C16	0,644961	0,633975
Case #265	C67	C61	0,637537	0,608443
Case #266	C64	C34.9	0,647998	0,614534
Case #267	C50	C61	0,644961	0,630504
Case #268	C15	C25	0,630504	0,589515
Case #269	C56	C25	0,648833	0,620601
Case #270	C18	C20	0,633975	0,620601
Case #271	C41	C34	0,670077	0,579797
Case #272	C50	C61	0,641198	0,630504
Case #273	C82	C34	0,674764	0,641198
Case #274	C34	C34	0,768338	0,768338
Case #275	C49.2	C18	0,674764	0,577474
Case #276	C34	C50	0,647998	0,630223
Case #277	C34	C18	0,620601	0,617457
Case #278	C97	C20	0,641198	0
Case #279	C34	C50	0,623821	0,608443

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #280	C50	C34.9	0,684666	0,684666
Case #281	C50	C83	0,608443	0,602756
Case #282	C34	C85.1	0,668501	0,668501
Case #283	C61	C34	0,644961	0,641198
Case #284	C43	C50.9	0,65282	0,614384
Case #285	C56	C34	0,734137	0,722486
Case #286	C22	C64	0,641198	0,536318
Case #287	C34	C34	0,648833	0,648833
Case #288	C90	C61	0,637537	0,614384
Case #289	C81	G12.2	0,623821	0
Case #290	C18	C61	0,641198	0,620601
Case #291	C25	C90	0,679623	0,65282
Case #292	C17	C34.9	0,684666	0
Case #293	C18	C63	0,633975	0,627121
Case #294	C56	C56	0,66117	0,66117
Case #295	C20	C20	0,726717	0,726717
Case #296	C20	C50.9	0,614384	0,605569
Case #297	C20	C16	0,65282	0,637537
Case #298	C56	C56	0,65693	0,65693
Case #299	C56	C50	0,644961	0,620601
Case #300	C56	C56	0,886496	0,886496
Case #301	C56	C56	0,846689	0,846689
Case #302	C34.9	C18.9	0,758828	0,66117
Case #303	C49.2	C49.2	0,65693	0,65693
Case #304	C34.9	C82.9	0,668501	0,641807
Case #305	C34.9	C34.9	0,633975	0,633975
Case #306	C34.9	C34.9	0,674764	0,674764
Case #307	C34.9	C34.9	0,70702	0,70702
Case #308	C34.9	C34.9	0,70702	0,70702
Case #309	C50.9	C50.9	0,70702	0,70702
Case #310	C50.9	C50.9	0,665549	0,665549
Case #311	C50.9	C50.9	0,689909	0,689909
Case #312	C50.9	C50.9	0,758828	0,758828
Case #313	C50.9	C50.9	0,758828	0,758828
Case #314	C50.9	C50.9	0,70702	0,70702
Case #315	C50.9	C50.9	0,734014	0,734014
Case #316	C50.9	C50.9	0,713259	0,713259
Case #317	C50	C49	0,637537	0,633975

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #318	C53	C53.9	0,627121	0,608443
Case #319	C53.9	C53	0,689909	0,684666
Case #320	C50.9	C50.9	0,65282	0,65282
Case #321	C50.9	C43	0,654489	0,600493
Case #322	C34.3	C34.3	0,633975	0,633975
Case #323	C34.3	C34.3	0,674764	0,674764
Case #324	C34.3	C34.3	0,70702	0,70702
Case #325	C34.3	C34.3	0,734014	0,734014
Case #326	C34.3	C34.3	0,65282	0,65282
Case #327	C34.3	C34.3	0,70702	0,70702
Case #328	C34.3	C34.3	0,778654	0,778654
Case #329	C34.3	C34.3	0,778654	0,778654
Case #330	C34.9	C54.1	0,633975	0,623821
Case #331	C34.9	C50	0,633975	0,623821
Case #332	G12.5	C12.2	0,679623	0
Case #333	C12.2	G12.2	0,679623	0,65282
Case #334	G12.2	G12.2	0,713259	0,713259
Case #335	G12.2	G12.2	0,695369	0,695369
Case #336	C43.9	C43.9	0,741754	0,741754
Case #337	C43.9	C43.9	0,768338	0,768338
Case #338	C43.9	C43.9	0,778654	0,778654
Case #339	C43.9	C43.9	0,778654	0,778654
Case #340	C43.9	C43.9	0,816504	0,816504
Case #341	C34.9	C43.9	0,741754	0,648833
Case #342	C43.9	C43.9	0,816504	0,816504
Case #343	C43.9	C43.9	0,816504	0,816504
Case #344	C20	C18	0,66117	0,65282
Case #345	C50.9	C50.9	0,719813	0,719813
Case #346	C50.9	C50.9	0,741754	0,741754
Case #347	C50.9	C50.9	0,758828	0,758828
Case #348	C50.9	C50.9	0,758828	0,758828
Case #349	C50.9	C50.9	0,734014	0,734014
Case #350	C67.9	C67.9	0,734014	0,734014
Case #351	C67.9	C67.9	0,778654	0,778654
Case #352	C67.9	C67.9	0,80245	0,80245
Case #353	C67.9	C67.9	0,832634	0,832634
Case #354	C67.9	C67.9	0,816504	0,816504
Case #355	C54.1	C54.1	0,641198	0,641198

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #356	C54.1	C50	0,644961	0,641198
Case #357	C54.1	C54.1	0,670077	0,670077
Case #358	C54.1	C54.1	0,670077	0,670077
Case #359	C54.1	C50	0,630504	0,611381
Case #360	C54.1	C54.1	0,641198	0,641198
Case #361	C82.9	C82.9	0,701065	0,701065
Case #362	C82.9	C82.9	0,719813	0,719813
Case #363	C82.9	C82.9	0,722486	0,722486
Case #364	C82.9	C82.9	0,818487	0,818487
Case #365	C82.9	C82.9	0,777424	0,777424
Case #366	C82.9	C82.9	0,796132	0,796132
Case #367	C19	C19	0,758828	0,758828
Case #368	C18.9	C19	0,734014	0,689909
Case #369	C18.9	C34.9	0,758828	0,726717
Case #370	C18.9	C18.9	0,75	0,75
Case #371	C18.9	C18.9	0,75	0,75
Case #372	C18.9	C18.9	0,741754	0,741754
Case #373	C85.1	C85.1	0,684161	0,684161
Case #374	C85.1	C85.1	0,722486	0,722486
Case #375	C85.1	C85.1	0,722486	0,722486
Case #376	C34	C34.9	0,65282	0,641198
Case #377	C34	C34	0,679623	0,679623
Case #378	C34.9	C34	0,674764	0,665549
Case #379	C34.9	C34.9	0,679623	0,679623
Case #380	C34.9	C34.9	0,679623	0,679623
Case #381	C34.9	C34.9	0,66117	0,66117
Case #382	C50	C50	0,684666	0,684666
Case #383	C50	C50	0,695369	0,695369
Case #384	C50	C50	0,701065	0,701065
Case #385	C50	C50	0,701065	0,701065
Case #386	C50	C50	0,778654	0,778654
Case #387	C50	C50	0,734014	0,734014
Case #388	C50	C50	0,75	0,75
Case #389	C50	C50	0,768338	0,768338
Case #390	C50	C50	0,741754	0,741754
Case #391	C50	C50	0,778654	0,778654
Case #392	C20	C18.9	0,623821	0,620601
Case #393	C18.9	C18.9	0,746963	0,746963

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #394	C18.9	C18.9	0,761252	0,761252
Case #395	C34.9	C34.9	0,689909	0,689909
Case #396	C34.9	C34.9	0,726717	0,726717
Case #397	C34.9	C34.9	0,713259	0,713259
Case #398	C34.4	C34.9	0,70702	0
Case #399	C56	C56	0,734137	0,734137
Case #400	C78.2	C56	0,674764	0
Case #401	G12.2	C34	0,65693	0,648833
Case #402	C50	C50	0,80245	0,80245
Case #403	C50	C50	0,80245	0,80245
Case #404	C34.9	C34.9	0,778654	0,778654
Case #405	C34.9	C34.9	0,789947	0,789947
Case #406	C34.9	C39.9	0,832634	0,80245
Case #407	C34.9	C34.9	0,789947	0,789947
Case #408	C39.9	C34.9	0,832634	0
Case #409	C34.0	C34.9	0,778654	0
Case #410	C34.9	C34.0	0,778654	0,778654
Case #411	C18	C18	0,684666	0,684666
Case #412	C18	C18	0,684666	0,684666
Case #413	C90	C18.9	0,65282	0,630504
Case #414	C25	C34.9	0,637537	0,630504
Case #415	C61	C49	0,630504	0,620601
Case #416	C43	C61	0,637537	0,617457
Case #417	C25	C43	0,66117	0,630504
Case #418	C15	C12.2	0,648833	0,594654
Case #419	C50	C34	0,630504	0,611381
Case #420	C39	C34	0,627121	0
Case #421	C45	C34	0,641198	0,614384
Case #422	C45	C45	0,695369	0,695369
Case #423	C50	C34	0,701927	0,63589
Case #424	C63	C34	0,633975	0,556259
Case #425	C56	C16	0,627121	0,605569
Case #426	C34	C81	0,623821	0,617457
Case #427	C43	C64	0,66117	0,617457
Case #428	C34	C34	0,648833	0,648833
Case #429	C61	C50	0,65282	0,648833
Case #430	C61	C18	0,641198	0,637537
Case #431	C50	C50	0,633975	0,633975

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #432	C63	C61	0,665549	0,614384
Case #433	C50	C25	0,630504	0,623821
Case #434	C34	C61	0,644961	0,641198
Case #435	C50	C25	0,648833	0,627121
Case #436	C61	C63	0,665549	0,65693
Case #437	C79	C34	0,633975	0
Case #438	C34	C79	0,633975	0,605569
Case #439	C25	C50	0,665549	0,641198
Case #440	C41	J44.9	0,623821	0,579797
Case #441	C50	C34	0,637537	0,630504
Case #442	C18	C50	0,679623	0,674764
Case #443	C34	C61.9	0,633975	0,627121
Case #444	C80	C34	0,66117	0,648833
Case #445	C61	C63	0,617457	0,611381
Case #446	C25	C50	0,644961	0,620601
Case #447	C78	C61	0,617457	0,579797
Case #448	C61	C43	0,679623	0,65282
Case #449	C62	C34	0,695369	0,679623
Case #450	C18	C18	0,623821	0,623821
Case #451	C34	C34	0,66117	0,66117
Case #452	C82	C18.2	0,674764	0,641198
Case #453	C25	C61	0,623821	0,614384
Case #454	C20	C20	0,648833	0,648833
Case #455	M54.2	C43.9	0,641198	0
Case #456	M89	C90	0,637537	0
Case #457	C38	C43	0,684666	0,605569
Case #458	C24.9	C63	0,641807	0,5495
Case #459	C34	C50	0,627121	0,623821
Case #460	C50.9	C45	0,713259	0,679623
Case #461	C50	C50	0,637537	0,637537
Case #462	C25.9	C45	0,695369	0,695369
Case #463	C25.9	C34	0,741754	0,695369
Case #464	C25.1	C50	0,637537	0
Case #465	C25	C34	0,611381	0,6
Case #466	C63	C34	0,637537	0,614384
Case #467	C34	G12.2	0,65282	0,648833
Case #468	C17.9	C50	0,65693	0
Case #469	C34	C20	0,648833	0,637537

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #470	C18	C61	0,66117	0,614384
Case #471	C21	C48	0,65282	0,58216
Case #472	C34	C50	0,670077	0,66117
Case #473	C34	C50	0,641807	0,630223
Case #474	C16	C20	0,65282	0,602756
Case #475	C25.9	C53	0,633975	0,594654
Case #476	C80	C20	0,648833	0,589515
Case #477	C61	C54.1	0,637537	0,633975
Case #478	C25.9	C49	0,637537	0,58216
Case #479	C61	C92	0,623821	0,620601
Case #480	C61	C50	0,665549	0,65693
Case #481	C67.9	C18	0,65282	0,617457
Case #482	C61	C61	0,630504	0,630504
Case #483	C56	C54.1	0,65693	0,620601
Case #484	C34.9	G12.2	0,637537	0,608443
Case #485	C61	C61	0,620601	0,620601
Case #486	C61	C50	0,620601	0,5973
Case #487	C64	C64	0,679623	0,679623
Case #488	C64	C64	0,695369	0,695369
Case #489	C61	C18	0,637537	0,633975
Case #490	C34	C50	0,623821	0,620601
Case #491	C64	C64	0,637537	0,637537
Case #492	C50	C50	0,695369	0,695369
Case #493	C09	C43	0,689909	0
Case #494	C61	C61	0,741754	0,741754
Case #495	C61	C61	0,741754	0,741754
Case #496	C20	C45	0,65282	0,627121
Case #497	C50.9	C50	0,611381	0,59206
Case #498	C61	C18	0,66117	0,65693
Case #499	C18.9	C18.9	0,665549	0,665549
Case #500	C20	C26	0,630223	0,624787
Case #501	C18	C61	0,644961	0,633975
Case #502	C61	C16	0,648833	0,623821
Case #503	C18.9	C54.1	0,641807	0,614534
Case #504	C61	C61	0,66117	0,66117
Case #505	C61	C61	0,66117	0,66117
Case #506	C20	C26	0,761252	0,661311
Case #507	C91	C25	0,637537	0

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #508	C18	C26	0,722486	0,701927
Case #509	C61	C61	0,734014	0,734014
Case #510	C19	C64	0,719813	0,641198
Case #511	C53	C67.9	0,695369	0,65693
Case #512	C50.9	C34	0,722486	0,684161
Case #513	C18	C56	0,679623	0,674764
Case #514	C18	C18	0,70702	0,70702
Case #515	C16	C34	0,661311	0,58779
Case #516	C01	C64	0,644961	0
Case #517	C53	C34.9	0,65282	0,644961
Case #518	C18.7	C18	0,689909	0
Case #519	C61	C61	0,65282	0,65282
Case #520	C62	C34	0,726717	0,679623
Case #521	C25	C25	0,719813	0,719813
Case #522	C50	C34.3	0,605569	0,6
Case #523	C50	C61	0,734014	0,684666
Case #524	C34	C82	0,674764	0,66117
Case #525	C82	C18	0,741754	0,633975
Case #526	C18	C61	0,758828	0,726717
Case #527	C61	C34	0,65693	0,65282
Case #528	C50	C61	0,65282	0,637537
Case #529	C50	C34.9	0,695369	0,684666
Case #530	J43	C81	0,605569	0
Case #531	J44.9	C49	0,630504	0
Case #532	C50.9	C50.9	0,602756	0,602756
Case #533	C25	C50	0,648833	0,6
Case #534	C88	C61	0,689909	0
Case #535	C83	C54.1	0,648833	0
Case #536	C25	C25	0,630504	0,630504
Case #537	C85	C18.9	0,668501	0,641807
Case #538	C22	C34	0,761252	0,562171
Case #539	C78	C61	0,719813	0,579797
Case #540	C34	C49	0,614384	0,608443
Case #541	C34	C61	0,66117	0,65282
Case #542	C34	C50	0,620601	0,611381
Case #543	C43	C50	0,620601	0,602756
Case #544	C79.3	C61	0,70702	0

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #545	C34.9	C82	0,719813	0,679623
Case #546	C34	C34	0,719813	0,719813
Case #547	C34.9	C45	0,611381	0,5973
Case #548	C34	C38	0,641198	0,630504
Case #549	C34.9	C34	0,630504	0,614384
Case #550	C03.1	C18	0,695369	0
Case #551	C43	C50	0,641198	0,614384
Case #552	C50	C34.9	0,608443	0,5973
Case #553	C16	C56	0,627121	0,605569
Case #554	C34.9	C18	0,637537	0,614384
Case #555	C50.9	C61	0,679623	0,65693
Case #556	C50	C25	0,665549	0,641198
Case #557	C12	C50.9	0,637537	0
Case #558	C90	C25	0,679623	0,665549
Case #559	C90	C90	0,665549	0,665549
Case #560	C70	C90	0,648833	0
Case #561	C15.5	G12.2	0,614384	0
Case #562	C53	C25.9	0,633975	0,608443
Case #563	C56	C50	0,633975	0,605569
Case #564	C83.1	C82.9	0,661311	0
Case #565	C49.2	C49.2	0,65693	0,65693
Case #566	C34.9	C34.9	0,641198	0,641198
Case #567	C34.9	C34.9	0,66117	0,66117
Case #568	C34.9	C34.9	0,674764	0,674764
Case #569	C50.9	C50.9	0,674764	0,674764
Case #570	C50.9	C50.9	0,726717	0,726717
Case #571	C50.9	C50.9	0,734014	0,734014
Case #572	C50.9	C50.9	0,614384	0,614384
Case #573	C53.9	C53.9	0,66117	0,66117
Case #574	C53.9	C53	0,778654	0,701065
Case #575	C53	C53.9	0,778654	0,65693
Case #576	C53.9	C67.9	0,719813	0,701065
Case #577	C53.9	C53.9	0,701065	0,701065
Case #578	C50.9	C50.9	0,633975	0,633975
Case #579	C50.9	C50.9	0,630504	0,630504
Case #580	C34.3	C34.3	0,70702	0,70702
Case #581	C34.3	C34.3	0,70702	0,70702
Case #582	C34.3	C34.3	0,701065	0,701065

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #583	C34.3	C34.3	0,734014	0,734014
Case #584	C34	C18	0,644961	0,641198
Case #585	G12.2	G12.2	0,701065	0,701065
Case #586	C12.2	G12.2	0,684666	0,65282
Case #587	G12.2	G12.2	0,701065	0,701065
Case #588	G12.2	G12.2	0,713259	0,713259
Case #589	C43.9	C43.9	0,768338	0,768338
Case #590	C43.9	C43.9	0,741754	0,741754
Case #591	C43.9	C43.9	0,768338	0,768338
Case #592	C43.9	C43.9	0,768338	0,768338
Case #593	C43.9	C43.9	0,778654	0,778654
Case #594	C43.9	C43.9	0,789947	0,789947
Case #595	C43.9	C43.9	0,778654	0,778654
Case #596	C20	C16	0,637537	0,633975
Case #597	C5O.9	C50.9	0,758828	0
Case #598	C50.9	C50.9	0,758828	0,758828
Case #599	C50.9	C50.9	0,734014	0,734014
Case #600	C67.9	C67.9	0,768338	0,768338
Case #601	C67.9	C67.9	0,80245	0,80245
Case #602	C67.9	C67.9	0,778654	0,778654
Case #603	C67.9	C67.9	0,832634	0,832634
Case #604	C67.9	C67.9	0,816504	0,816504
Case #605	C67.9	C67.9	0,816504	0,816504
Case #606	C67.9	C67.9	0,816504	0,816504
Case #607	C54.1	C54.1	0,623821	0,623821
Case #608	C54.1	C50	0,665549	0,65693
Case #609	C82.9	C82.9	0,734137	0,734137
Case #610	C82.9	C18	0,726717	0,719813
Case #611	C82.9	C82.9	0,818487	0,818487
Case #612	C19	C19	0,758828	0,758828
Case #613	C18	C18.9	0,726717	0,670077
Case #614	C18.9	C18.9	0,75	0,75
Case #615	C85.1	C50.9	0,701065	0,589515
Case #616	C85.1	C85.1	0,701927	0,701927
Case #617	C85.1	C85.1	0,722486	0,722486
Case #618	C85.1	C85.1	0,777424	0,777424
Case #619	C85.1	C85.1	0,777424	0,777424
Case #620	C85.1	C85.1	0,722486	0,722486

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #621	C85.1	C85.1	0,7118	0,7118
Case #622	C85.1	C85.1	0,722486	0,722486
Case #623	C34.9	C34.9	0,674764	0,674764
Case #624	C50	C50	0,701065	0,701065
Case #625	C50	C50	0,726717	0,726717
Case #626	C50	C50	0,726717	0,726717
Case #627	C50	C50	0,768338	0,768338
Case #628	C50	C50	0,734014	0,734014
Case #629	C50	C50	0,768338	0,768338
Case #630	C50	C50	0,768338	0,768338
Case #631	C20	C20	0,65282	0,65282
Case #632	C18.9	C18.9	0,746963	0,746963
Case #633	C18.9	C18.9	0,761252	0,761252
Case #634	C80	C18	0,641198	0,5973
Case #635	C34.9	C34.9	0,734014	0,734014
Case #636	C34.9	C34.9	0,734014	0,734014
Case #637	C34.9	C34.4	0,689909	0,689909
Case #638	C56	C78	0,674764	0,66117
Case #639	C56	C56	0,777424	0,777424
Case #640	C56	C56	0,886496	0,886496
Case #641	C56	C56	0,846689	0,846689
Case #642	C56	C34.9	0,684666	0,633975
Case #643	C34.9	C34.9	0,80245	0,80245
Case #644	C34.9	C34.9	0,741754	0,741754
Case #645	C34.9	C34.9	0,778654	0,778654
Case #646	C18	C50	0,633975	0,617457
Case #647	C61	C61	0,65282	0,65282
Case #648	C18	C34	0,777424	0,701927
Case #649	C18	C56	0,701927	0,684161

A.2 The lung cancer dataset

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #1	1	2	0,657568	0,644187
Case #2	1	1	0,67244	0,67244
Case #3	1	1	0,680549	0,680549
Case #4	1	1	0,65071	0,65071
Case #5	1	1	0,664797	0,664797
Case #6	1	1	0,657568	0,657568
Case #7	1	1	0,657568	0,657568
Case #8	1	1	0,657568	0,657568
Case #9	1	1	0,680549	0,680549
Case #10	2	3	0,626331	0,626331
Case #11	2	2	0,644187	0,644187
Case #12	2	2	0,65071	0,65071
Case #13	2	3	0,698428	0,65071
Case #14	2	2	0,657568	0,657568
Case #15	2	2	0,644187	0,644187
Case #16	2	3	0,644187	0,632022
Case #17	2	1	0,644187	0,644187
Case #18	2	2	0,65071	0,65071
Case #19	2	2	0,632022	0,632022
Case #20	2	1	0,664797	0,657568
Case #21	2	1	0,644187	0,637967
Case #22	2	1	0,664797	0,637967
Case #23	3	3	0,605714	0,605714
Case #24	3	3	0,644187	0,644187
Case #25	3	3	0,644187	0,644187
Case #26	3	3	0,605714	0,605714
Case #27	3	3	0,610578	0,610578
Case #28	3	3	0,615626	0,615626
Case #29	3	3	0,626331	0,626331
Case #30	3	3	0,626331	0,626331
Case #31	3	3	0,689186	0,689186
Case #32	3	2	0,698428	0,689186

A.3 The soybean large dataset

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #1	diaporthe-stem-canker	diaporthe-stem-canker	0,75	0,75
Case #2	diaporthe-stem-canker	diaporthe-stem-canker	0,809256	0,809256
Case #3	diaporthe-stem-canker	diaporthe-stem-canker	0,809256	0,809256
Case #4	diaporthe-stem-canker	diaporthe-stem-canker	0,809256	0,809256
Case #5	diaporthe-stem-canker	diaporthe-stem-canker	0,809256	0,809256
Case #6	diaporthe-stem-canker	diaporthe-stem-canker	0,775991	0,775991
Case #7	diaporthe-stem-canker	diaporthe-stem-canker	0,75	0,75
Case #8	diaporthe-stem-canker	diaporthe-stem-canker	0,775991	0,775991
Case #9	diaporthe-stem-canker	diaporthe-stem-canker	0,775991	0,775991
Case #10	diaporthe-stem-canker	diaporthe-stem-canker	0,809256	0,809256
Case #11	charcoal-rot	charcoal-rot	0,857143	0,857143
Case #12	charcoal-rot	charcoal-rot	0,75	0,75
Case #13	charcoal-rot	charcoal-rot	0,775991	0,775991
Case #14	charcoal-rot	charcoal-rot	0,775991	0,775991
Case #15	charcoal-rot	charcoal-rot	0,857143	0,857143
Case #16	charcoal-rot	charcoal-rot	0,775991	0,775991
Case #17	charcoal-rot	charcoal-rot	0,75	0,75
Case #18	charcoal-rot	charcoal-rot	0,775991	0,775991
Case #19	charcoal-rot	charcoal-rot	0,809256	0,809256
Case #20	charcoal-rot	charcoal-rot	0,809256	0,809256
Case #21	rhizoctonia-root-rot	rhizoctonia-root-rot	0,809256	0,809256
Case #22	rhizoctonia-root-rot	rhizoctonia-root-rot	0,857143	0,857143
Case #23	rhizoctonia-root-rot	rhizoctonia-root-rot	0,710102	0,710102
Case #24	rhizoctonia-root-rot	rhizoctonia-root-rot	0,809256	0,809256

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #25	rhizoctonia-root-rot	rhizoctonia-root-rot	0,809256	0,809256
Case #26	rhizoctonia-root-rot	rhizoctonia-root-rot	0,857143	0,857143
Case #27	rhizoctonia-root-rot	rhizoctonia-root-rot	0,809256	0,809256
Case #28	rhizoctonia-root-rot	rhizoctonia-root-rot	0,809256	0,809256
Case #29	rhizoctonia-root-rot	rhizoctonia-root-rot	0,857143	0,857143
Case #30	rhizoctonia-root-rot	rhizoctonia-root-rot	0,857143	0,857143
Case #31	phytophthora-rot	phytophthora-rot	0,775991	0,775991
Case #32	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #33	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #34	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #35	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #36	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #37	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #38	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #39	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #40	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #41	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #42	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #43	phytophthora-rot	phytophthora-rot	0,857143	0,857143

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #44	phytophthora-rot	phytophthora-rot	0,710102	0,710102
Case #45	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #46	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #47	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #48	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #49	phytophthora-rot	phytophthora-rot	0,775991	0,775991
Case #50	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #51	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #52	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #53	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #54	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #55	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #56	phytophthora-rot	phytophthora-rot	0,75	0,75
Case #57	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #58	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #59	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #60	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #61	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #62	phytophthora-rot	phytophthora-rot	0,857143	0,857143

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #63	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #64	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #65	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #66	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #67	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #68	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #69	phytophthora-rot	phytophthora-rot	0,809256	0,809256
Case #70	phytophthora-rot	phytophthora-rot	0,857143	0,857143
Case #71	brown-stem-rot	brown-stem-rot	1	1
Case #72	brown-stem-rot	brown-stem-rot	1	1
Case #73	brown-stem-rot	brown-stem-rot	0,710102	0,710102
Case #74	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #75	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #76	brown-stem-rot	brown-stem-rot	0,857143	0,857143
Case #77	brown-stem-rot	brown-stem-rot	0,728503	0,728503
Case #78	brown-stem-rot	brown-stem-rot	0,728503	0,728503
Case #79	brown-stem-rot	brown-stem-rot	0,710102	0,710102
Case #80	brown-stem-rot	brown-stem-rot	0,728503	0,728503
Case #81	brown-stem-rot	brown-stem-rot	0,728503	0,728503
Case #82	brown-stem-rot	brown-stem-rot	0,728503	0,728503
Case #83	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #84	brown-stem-rot	brown-stem-rot	0,710102	0,710102
Case #85	brown-stem-rot	brown-stem-rot	0,857143	0,857143
Case #86	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #87	brown-stem-rot	brown-stem-rot	0,857143	0,857143
Case #88	brown-stem-rot	brown-stem-rot	0,857143	0,857143
Case #89	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #90	brown-stem-rot	brown-stem-rot	0,809256	0,809256
Case #91	powdery-mildew	powdery-mildew	0,75	0,75

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #92	powdery-mildew	powdery-mildew	0,75	0,75
Case #93	powdery-mildew	powdery-mildew	0,809256	0,809256
Case #94	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #95	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #96	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #97	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #98	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #99	powdery-mildew	powdery-mildew	0,775991	0,775991
Case #100	powdery-mildew	powdery-mildew	0,809256	0,809256
Case #101	downy-mildew	downy-mildew	0,75	0,75
Case #102	downy-mildew	downy-mildew	0,75	0,75
Case #103	downy-mildew	downy-mildew	0,775991	0,775991
Case #104	downy-mildew	downy-mildew	0,75	0,75
Case #105	downy-mildew	downy-mildew	0,775991	0,775991
Case #106	downy-mildew	downy-mildew	0,775991	0,775991
Case #107	downy-mildew	downy-mildew	0,775991	0,775991
Case #108	downy-mildew	downy-mildew	0,728503	0,728503
Case #109	downy-mildew	downy-mildew	0,75	0,75
Case #110	downy-mildew	downy-mildew	0,775991	0,775991
Case #111	brown-spot	brown-spot	0,809256	0,809256
Case #112	brown-spot	brown-spot	0,809256	0,809256
Case #113	brown-spot	brown-spot	0,857143	0,857143
Case #114	brown-spot	brown-spot	0,775991	0,775991
Case #115	brown-spot	brown-spot	0,809256	0,809256
Case #116	brown-spot	brown-spot	1	1
Case #117	brown-spot	brown-spot	0,775991	0,775991
Case #118	brown-spot	brown-spot	0,857143	0,857143
Case #119	brown-spot	brown-spot	0,857143	0,857143
Case #120	brown-spot	brown-spot	0,809256	0,809256

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #121	brown-spot	brown-spot	0,857143	0,857143
Case #122	brown-spot	brown-spot	0,809256	0,809256
Case #123	brown-spot	brown-spot	0,857143	0,857143
Case #124	brown-spot	brown-spot	0,857143	0,857143
Case #125	brown-spot	brown-spot	0,857143	0,857143
Case #126	brown-spot	brown-spot	0,809256	0,809256
Case #127	brown-spot	brown-spot	0,857143	0,857143
Case #128	brown-spot	brown-spot	0,728503	0,728503
Case #129	brown-spot	brown-spot	0,728503	0,728503
Case #130	brown-spot	brown-spot	0,775991	0,775991
Case #131	brown-spot	brown-spot	0,809256	0,809256
Case #132	brown-spot	brown-spot	0,809256	0,809256
Case #133	brown-spot	brown-spot	0,809256	0,809256
Case #134	brown-spot	brown-spot	0,775991	0,775991
Case #135	brown-spot	brown-spot	0,857143	0,857143
Case #136	brown-spot	brown-spot	1	1
Case #137	brown-spot	brown-spot	0,857143	0,857143
Case #138	brown-spot	brown-spot	0,710102	0,710102
Case #139	brown-spot	brown-spot	0,809256	0,809256
Case #140	brown-spot	brown-spot	0,809256	0,809256
Case #141	brown-spot	brown-spot	0,809256	0,809256
Case #142	brown-spot	brown-spot	0,809256	0,809256
Case #143	brown-spot	brown-spot	0,728503	0,728503
Case #144	brown-spot	brown-spot	0,857143	0,857143
Case #145	brown-spot	brown-spot	0,809256	0,809256
Case #146	brown-spot	brown-spot	0,857143	0,857143
Case #147	brown-spot	brown-spot	0,809256	0,809256
Case #148	brown-spot	brown-spot	0,857143	0,857143
Case #149	brown-spot	brown-spot	0,809256	0,809256
Case #150	brown-spot	brown-spot	0,75	0,75
Case #151	bacterial-blight	bacterial-blight	0,775991	0,775991
Case #152	bacterial-blight	bacterial-blight	0,809256	0,809256
Case #153	bacterial-blight	bacterial-blight	0,775991	0,775991
Case #154	bacterial-blight	bacterial-blight	0,809256	0,809256
Case #155	bacterial-blight	bacterial-blight	0,809256	0,809256
Case #156	bacterial-blight	bacterial-blight	0,775991	0,775991
Case #157	bacterial-blight	bacterial-blight	0,809256	0,809256
Case #158	bacterial-blight	bacterial-blight	0,809256	0,809256

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #159	bacterial-blight	bacterial-blight	0,775991	0,775991
Case #160	bacterial-blight	bacterial-blight	0,775991	0,775991
Case #161	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #162	bacterial-pustule	bacterial-pustule	0,710102	0,710102
Case #163	bacterial-pustule	bacterial-blight	0,775991	0,710102
Case #164	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #165	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #166	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #167	bacterial-pustule	bacterial-pustule	0,775991	0,775991
Case #168	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #169	bacterial-pustule	bacterial-pustule	0,775991	0,775991
Case #170	bacterial-pustule	bacterial-pustule	0,728503	0,728503
Case #171	purple-seed-stain	purple-seed-stain	0,693982	0,693982
Case #172	purple-seed-stain	purple-seed-stain	0,809256	0,809256
Case #173	purple-seed-stain	purple-seed-stain	0,75	0,75
Case #174	purple-seed-stain	purple-seed-stain	0,75	0,75
Case #175	purple-seed-stain	purple-seed-stain	0,75	0,75
Case #176	purple-seed-stain	purple-seed-stain	0,809256	0,809256
Case #177	purple-seed-stain	purple-seed-stain	0,728503	0,728503
Case #178	purple-seed-stain	purple-seed-stain	0,75	0,75

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #179	purple-seed-stain	purple-seed-stain	0,728503	0,728503
Case #180	purple-seed-stain	purple-seed-stain	0,728503	0,728503
Case #181	anthracnose	anthracnose	0,710102	0,710102
Case #182	anthracnose	anthracnose	0,666667	0,666667
Case #183	anthracnose	anthracnose	0,775991	0,775991
Case #184	anthracnose	anthracnose	0,693982	0,693982
Case #185	anthracnose	anthracnose	0,728503	0,728503
Case #186	anthracnose	anthracnose	0,75	0,75
Case #187	anthracnose	anthracnose	0,809256	0,809256
Case #188	anthracnose	anthracnose	0,693982	0,693982
Case #189	anthracnose	anthracnose	0,809256	0,809256
Case #190	anthracnose	anthracnose	0,728503	0,728503
Case #191	anthracnose	anthracnose	0,75	0,75
Case #192	anthracnose	anthracnose	0,693982	0,693982
Case #193	anthracnose	anthracnose	0,679623	0,679623
Case #194	anthracnose	anthracnose	0,809256	0,809256
Case #195	anthracnose	anthracnose	0,710102	0,710102
Case #196	anthracnose	anthracnose	0,75	0,75
Case #197	anthracnose	anthracnose	0,775991	0,775991
Case #198	anthracnose	anthracnose	0,75	0,75
Case #199	anthracnose	anthracnose	0,809256	0,809256
Case #200	anthracnose	anthracnose	0,809256	0,809256
Case #201	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,809256	0,809256
Case #202	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,75	0,75
Case #203	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,809256	0,809256
Case #204	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,75	0,75
Case #205	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,728503	0,728503
Case #206	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,728503	0,728503
Case #207	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,728503	0,728503

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #208	phyllosticta-leaf-spot	brown-spot	0,728503	0,728503
Case #209	phyllosticta-leaf-spot	phyllosticta-leaf-spot	0,728503	0,728503
Case #210	phyllosticta-leaf-spot	brown-spot	0,75	0,728503
Case #211	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #212	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #213	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #214	alternarialeaf-spot	alternarialeaf-spot	1	1
Case #215	alternarialeaf-spot	alternarialeaf-spot	0,75	0,75
Case #216	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #217	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #218	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #219	alternarialeaf-spot	alternarialeaf-spot	1	1
Case #220	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #221	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #222	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #223	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #224	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #225	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #226	alternarialeaf-spot	frog-eye-leaf-spot	0,809256	0,775991

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #227	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #228	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #229	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #230	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #231	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #232	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #233	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #234	alternarialeaf-spot	alternarialeaf-spot	0,75	0,75
Case #235	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #236	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #237	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #238	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #239	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #240	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #241	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #242	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #243	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #244	alternarialeaf-spot	frog-eye-leaf-spot	0,809256	0,775991
Case #245	alternarialeaf-spot	alternarialeaf-spot	0,775991	0,775991

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #246	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #247	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #248	alternarialeaf-spot	alternarialeaf-spot	0,857143	0,857143
Case #249	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #250	alternarialeaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #251	frog-eye-leaf-spot	frog-eye-leaf-spot	0,728503	0,728503
Case #252	frog-eye-leaf-spot	alternarialeaf-spot	0,775991	0,75
Case #253	frog-eye-leaf-spot	frog-eye-leaf-spot	0,710102	0,710102
Case #254	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #255	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #256	frog-eye-leaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #257	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #258	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #259	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #260	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #261	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #262	frog-eye-leaf-spot	frog-eye-leaf-spot	0,775991	0,775991
Case #263	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #264	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #265	frog-eye-leaf-spot	frog-eye-leaf-spot	0,75	0,75
Case #266	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #267	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #268	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #269	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #270	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #271	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #272	frog-eye-leaf-spot	frog-eye-leaf-spot	0,728503	0,728503
Case #273	frog-eye-leaf-spot	brown-spot	0,693982	0,693982
Case #274	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #275	frog-eye-leaf-spot	alternarialeaf-spot	0,809256	0,809256
Case #276	frog-eye-leaf-spot	alternarialeaf-spot	0,775991	0,775991
Case #277	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #278	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #279	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #280	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #281	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #282	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #283	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #284	frog-eye-leaf-spot	frog-eye-leaf-spot	0,75	0,75
Case #285	frog-eye-leaf-spot	alternarialeaf-spot	0,809256	0,775991
Case #286	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #287	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #288	frog-eye-leaf-spot	frog-eye-leaf-spot	0,775991	0,775991
Case #289	frog-eye-leaf-spot	frog-eye-leaf-spot	0,857143	0,857143
Case #290	frog-eye-leaf-spot	frog-eye-leaf-spot	0,809256	0,809256
Case #291	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,855409	0
Case #292	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,807073	0
Case #293	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,855409	0
Case #294	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,725708	0
Case #295	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,807073	0
Case #296	diaporthe-pod-&-stem-blight	diaporthe-pod-&-stem-blight	0,855409	0
Case #297	cyst-nematode	cyst-nematode	0,857143	0,857143
Case #298	cyst-nematode	cyst-nematode	0,857143	0,857143
Case #299	cyst-nematode	cyst-nematode	0,857143	0,857143
Case #300	cyst-nematode	cyst-nematode	1	1
Case #301	cyst-nematode	cyst-nematode	1	1
Case #302	cyst-nematode	cyst-nematode	0,857143	0,857143
Case #303	2-4-d-injury	herbicide-injury	0,651669	0
Case #304	herbicide-injury	herbicide-injury	0,857143	0,857143
Case #305	herbicide-injury	herbicide-injury	0,809256	0,809256

Name	Solution	Solution of most similar	Similarity of most similar	Similarity of most similar correct
Case #306	herbicide-injury	herbicide-injury	0,809256	0,809256
Case #307	herbicide-injury	herbicide-injury	0,857143	0,857143