



Norwegian University of
Science and Technology

Case-Based Reasoning and Computational Creativity in a Recipe Recommender System

Kari Skjold

Marthe Sofie Øynes

Master of Science in Informatics

Submission date: June 2017

Supervisor: Agnar Aamodt, IDI

Co-supervisor: Kerstin Bach, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

A domain where humans have unfolded their creativity for thousands of years is cooking. However, can a human's creativity within cooking be transferred to a computer program? The case-based reasoning (CBR) methodology allows to incorporate creativity when using earlier experiences to solve new problems. The main objective of our research is to design and construct a CBR based recipe recommender system that enhances creativity to adapt recipes based on a given user query.

We have reviewed related work performed in the field of recipe recommendation and identified typical approaches and features within those. Further, we developed a knowledge engineering heavy system that retrieve, compare, adapt, and suggest recipes given a user query containing desired and undesired ingredients.

The system was tested with controlled observations, questionnaires, and an online quiz. Evaluation results show that the system appears user-friendly, self-explanatory, and with meaningful recipe recommendations. The resulting system is able to adapt recipes in a way that humans find it challenging to distinguish them from human created recipes.

Sammendrag

Matlagning er et domene hvor mennesker har utfoldet sin kreativitet gjennom tusener av år. Kan menneskers kreativitet innen matlagning overføres til et dataprogram? Case-basert resonnering (CBR) er en generell metode som åpner for å inkorporere kreativitet når tidligere erfaringer brukes for å løse nye problemer. Hovedmålet for vår forskning er å designe og konstruere et CBR-basert anbefalingssystem for oppskrifter som på en kreativ måte modifierer oppskrifter basert på en gitt brukerforespørsel.

Vi har vurdert relaterte anbefalingssystemer for oppskrifter, og identifisert typiske metoder og egenskaper i disse. Videre har vi utviklet et system som krever fokus på kunnskapsmodellering. Systemet henter, sammenligner, modifierer og foreslår oppskrifter for en gitt brukerforespørsel bestående av ønskede og uønskede ingredienser.

Systemet ble testet under kontrollerte observasjoner, spørreskjemaer, og en online quiz. Resultater fra evalueringen viser at systemet fremstår brukervennlig, selvforklarende og med meningsfulle anbefalinger. Systemet er i stand til å modifisere oppskrifter på en slik måte at mennesker har store vanskeligheter med å skille dem fra menneskeskapte oppskrifter.

Acknowledgements

This thesis was written during the autumn of 2016 and spring of 2017 for the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU).

The subject for the thesis was defined in cooperation with our supervisor Kerstin Bach, with background in the Computer Cooking Contest (CCC) at the International Conference on Case-Based Reasoning (ICCBR).

We would like to thank Kerstin Bach and Agnar Aamodt for enthusiastic contributions, help, and feedback throughout the project. We would also like to thank our friends and family for the invaluable support and motivation the last year.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Goals and research questions	2
1.3	Research methodology	3
1.4	Thesis structure	3
2	Background	5
2.1	The Computer Cooking Contest	5
2.1.1	Competition rules	5
2.1.2	Motivation	6
2.2	Case-based reasoning	6
2.2.1	CBR cycle	7
2.2.2	Typical CBR domains	8
2.3	CBR compared to other AI approaches	8
2.3.1	Recommender systems	9
2.3.2	Knowledge acquisition and construction	10
2.3.3	Generalization	12
2.3.4	Maintenance	12
2.4	Discussion	13
3	Related work	15
3.1	Competitors of the CCC	15
3.1.1	Taaable	15
3.1.2	CookIIS	16
3.1.3	CookingCAKE	17
3.1.4	JaDaWeb	19
3.2	Non-CBR related work	20
3.2.1	AllRecipes	21
3.2.2	Chef Watson	22
3.2.3	Yummly	22
3.3	Discussion	23

4	Method	25
4.1	Data modeling	26
4.1.1	Attributes	26
4.1.2	Case base	27
4.1.3	Similarity measures	28
4.2	IntelliMeal’s modified CBR cycle	29
4.3	Problem description	30
4.4	Retrieval	32
4.4.1	Retrieval in myCBR	32
4.4.2	New retrieval method	33
4.5	Case base retrieval	34
4.6	Reuse	34
4.6.1	Rule engine	36
4.6.2	Adaptation with undesired query	37
4.6.3	Adaptation with desired query	40
4.6.4	Suitable adaptation	43
4.7	Constructing an ephemeral case base	45
4.7.1	Filter original cases	46
4.7.2	Filter adapted instances	46
4.7.3	Creating an ephemeral case base	47
4.8	Ephemeral case base retrieval	48
4.9	Revise and retain	49
5	Implementation	51
5.1	Overview	51
5.2	Preprocessing	52
5.2.1	Generating CSV instance file	52
5.2.2	Generating JSON tree for recipe storage	53
5.2.3	Generating list of duplicate values	54
5.3	Running application	54
5.3.1	Core system	55
5.3.2	User Interface	56
6	Evaluation and results	63
6.1	Similarity score increase with adaptation	63
6.2	Ranking of recipes	65
6.3	Quality of modified recipes	67
6.4	Usability testing with questionnaire	69
7	Discussion	73
7.1	State of the art research	73
7.2	CBR and its opportunities for creativity	74
7.3	Implemented adaptation process	74
7.4	System behaviour	76
7.4.1	Drawbacks	78
7.5	Demonstrating the system	79

8 Conclusion	81
8.1 Conclusion	81
8.2 Future work	82
8.2.1 Data modeling	82
8.2.2 System improvements	82
A Evaluation and results	87
A.1 Results from evaluation of similarity scores	87
A.2 Ranking of recipes	87
A.3 Results from ranking of recipes	88
A.4 Queries used and cases added to quiz case base	90
A.5 Results from quiz	91
A.6 Usability testing	92
A.7 Results from usability testing	96
Acronyms	97
Glossary	99
Bibliography	99

List of Tables

6.1	Queries used for measuring similarity scores	64
6.2	Tasks for ranking of recipes	65
6.3	Number of recipes in common for the system and test subjects	66
A.1	The average score of the top five results	87
A.2	Ranking measure for evaluation of ranking of recipes test	87
A.3	The system's ranking of recipes per query	88
A.4	The offset of recipe ranks between the system and each test subject	89
A.5	Queries used and cases added to "Bot or Not?" case base	90
A.6	Guessing results for the "Bot or not?" quiz	91
A.7	Replies to pre usability testing form	96
A.8	Replies to SUS form	96
A.9	Replies to post usability testing form	96

List of Figures

2.1	CBR cycle	7
2.2	Knowledge-based systems overview	9
4.1	Three-layer architecture	25
4.2	myCBR attributes	26
4.3	Instance example	27
4.4	Excerpt from the Meat taxonomy defined in myCBR	28
4.5	Global similarity measures as defined in myCBR	28
4.6	Modified version of the CBR cycle	29
4.7	Splitting a user query	31
4.8	Comparing only attributes present in the query	33
4.9	Calculating the local similarity between meat ingredients	33
4.10	Case base retrieval example	34
4.11	Adaptation process	35
4.12	Rule formats	36
4.13	Undesired adaptation process overview	37
4.14	Firing the deletion rule <i>lettuce,tomato</i> \rightarrow <i>cucumber</i>	38
4.15	Firing the substitution rule <i>tuna; supplement</i> \rightarrow <i>mayonnaise</i>	39
4.16	Finding a similar ingredient in the myCBR taxonomy	40
4.17	Desired adaptation process overview	40
4.18	Excerpt from the Tomato taxonomy defined in myCBR	41
4.19	Firing the adding rule <i>tomato,lettuce*</i> \rightarrow <i>cucumber*</i>	41
4.20	Firing the substitution rule <i>pork; supplement</i> \rightarrow <i>barbeque sauce</i>	42
4.21	Finding a similar ingredient in the myCBR taxonomy	43
4.22	Suitable adaptation process overview	43
4.23	Generating title based on recipe ingredients	44
4.24	Generating title based on previous title	45
4.25	Process overview	45
4.26	Several cycles of adaptation result in equal cases	47
4.27	Creating an ephemeral case base	47
4.28	Ephemeral case base retrieval	48
4.29	Decreasing the similarity score of an adapted recipe	48
5.1	System overview	51

LIST OF FIGURES

5.2	Generating CSV file	53
5.3	UI takes in a user query and makes a request to the API	54
5.4	Input fields on front page	56
5.5	Ingredient search completion	56
5.6	Specified desired and undesired ingredients	57
5.7	Website after retrieval of similar recipes	58
5.8	Example of adapted case	59
5.9	Infotip explaining the adaptation	59
5.10	Example of a recommended recipe lacking desired ingredients	60
5.11	Adding a case to the case base	61
6.1	Average similarity score for original versus adapted case	64
6.2	Box plot of ranking of recipes result	66
6.3	Screenshot from the quiz "Bot or not?"	67
6.4	Confusion matrix for quiz responses	68
6.5	Distribution of test subjects' age	70
6.6	Issues detected during usability testing	71
8.1	Possible substitution results during adaptation	83
8.2	Example of different results with different adaptation orders	83
8.3	Filtering of alternative recipes	84
A.1	Tasks for usability testing	92
A.2	Pre user testing reply form	93
A.3	System Usability Scale Form	94
A.4	Post user testing reply form	95

Chapter 1: Introduction

1.1 Motivation

Artificial Intelligence (AI) can control self-driving cars, be a fierce poker player and beat Magnus Carlsen in chess; machines outperform humans at an impressively increasing number of tasks. The field has a long history and is still in constant growth and change. However, what exactly is intelligence exhibited by machines? In 1950, Alan Turing stated that if a machine could lead a conversation with a human and the human could not distinguish the machine from another human being, it was reasonable to say that the machine was thinking, and hence, *intelligent* (Turing 1950). The field of AI Research was founded as an academic discipline in 1956.

For the next fifty years, research within AI was mostly targeted towards solving particular sub-problems such as natural language understanding, or ways to represent knowledge (Brooks 1991). Replicating the human intelligence as a total is yet to be accomplished, and up until today, researchers do not agree on whether the so-called *Turing Test* is passed by any computing machine.

In the first decade of the twenty-first century, the access to faster computers, big amount of data and the evolution of more advanced machine learning techniques contributed to giving the AI field a boom. Today AI is infiltrating "everywhere," so to say. It is in our phones, cars, and banks. Functioning as our personal assistants, simplifying our everyday life by, amongst more, making decisions for us. However, can AI actually imitate the human decision making process?

When humans make decisions, the brain goes through three steps (Klein 2008). First, we gather information, either through sense or by remembering. Secondly, we decide on which information is relevant and important, and lastly, we act on the given information. These three steps are what the majority of AI systems are trying to imitate. Usually, the reasoning involves using generalized sets of rules to make decisions. However, there is one AI approach that takes a very different view: Case-Based Reasoning (CBR). Rather than having generalized rules as knowledge, CBR systems operate with cases recording specific previous experiences. The cases are then used to form conclusions or solutions to new problems (Leake 1996). The methodology provides a computational model that is very close to human reasoning; humans often use their previous experiences to find solutions to

new situations (Aamodt 1991; Kolodner 1992; Aamodt 1995). Still, can AI imitate human behavior completely, such as the aspect of being creative?

Well, computers have written both poems¹ and news articles (BBC 2014), and the first music composed by AI was considered good enough to be played by one of the world's leading orchestras, London Symphony Orchestra (Ball 2012). Perhaps even more impressive, a full musical was generated by AI (Brown 2015). The musical was performed with success in London, West End, at the beginning of 2016.

The potential of computer programs to be more than feature-rich tools; such as to be creative on a human level, and to act autonomous, has been studied and exploited since the very early days of computer science. The field is known as Computational Creativity (CC)², and CC is by some researchers characterized as the final frontier for AI research (Colton and Wiggins 2012).

The goal of CC is to design systems that are capable of human-level creativity, or that enhance human creativity without necessarily being creative themselves. Theoretical work on the nature of creativity is studied on in parallel with practical work on how this can be imitated by systems (Besold, Schorlemmer, and Smaill 2015).

A domain where humans have unfolded their creativity for thousands of years is cooking. For many years, the cookbook was the main source for recipes. However, in today's digital world, recipe search engines allow faster searching and scalable content. Some systems even utilize AI to find the best matching recipe given a problem. In this thesis, recipe recommender systems enhancing creativity are considered. The goal is to make the recommendations even more customized. Can a human's creativity within cooking be transferred to a computer system?

1.2 Goals and research questions

Research goal 1: Consider previous research on recipe recommender systems to discover advantages and drawbacks with their approach and to discover opportunities for improvements.

RQ1: What are features previously showcased in recipe recommender systems?

RQ2: What are approaches typically used in recipe recommender systems?

¹www.nil.fdi.ucm.es/?q=node/206

²www.prosecco-network.eu/prosecco

Research goal 2: Contribute to Case-Based Reasoning research by investigating the possibilities of enhancing creativity in a recipe recommender system using CBR.

RQ3: Is CBR a suitable approach for developing a recipe recommender system?

RQ4: How can CBR provide opportunities to being creative?

RQ5: If there exist opportunities, how can creativity be incorporated?

RQ6: How do the creative adjustments affect the system output?

RQ7: How can the system be demonstrated in a suitable manner to a broad range of people?

1.3 Research methodology

March and Smith have created a framework for IT research (March and Smith 1995). The framework consists of four steps: Build, Evaluate, Theorize and Justify. System creation and system evaluation are viewed as the basics steps of design science (Simon 1996), while theory proposal and theory justification are the basic steps of natural science. With other words, the understanding of *how* and *why* the developed system work, is crucial. This thesis starts out with a literature review, aiming for background knowledge about previous recipe recommender system. Further, the framework proposed by March and Smith is followed. This process is run by first developing a recipe recommender system, then evaluate the system and finally theorize and justify the results.

1.4 Thesis structure

The thesis is divided into eight chapters. Chapter 2 discusses the background for the project, introduces CBR and features a comparison of CBR to other AI approaches. Chapter 3 contains a detailed description of earlier work involving recipe recommendation, along with a discussion on the different systems and how they are interesting in relation to our work. In Chapter 4, the system method employed in the developed system is explained. Further, Chapter 5 describes the implementation and architecture of the implemented system. In Chapter 6, the process of evaluating the implemented system is described, and the results of the evaluation are presented. Chapter 7 follow up with a discussion on the evaluation and results. Lastly, Chapter 8 contains a summary and conclusion of the examined work, and also includes a section concerning possible future additions or adjustments.

Chapter 2: Background

The Computer Cooking Contest (CCC) is an event of the International Conference on Case Based Reasoning (ICCBR). CBR is an AI method for building learning systems. Broadly construed, the process involves solving new problems based on the solutions of similar past problems. The CCC and CBR will be explained in Section 2.1 and Section 2.2, respectively.

To reason based on concrete problem situations stands out from other AI approaches (Aamodt and Plaza 1994). In order to show that CBR is a suitable approach for developing a recipe recommender system, it is compared with other AI approaches in Section 2.3.

2.1 The Computer Cooking Contest

The ICCBR conference series was established in 1995. In 2008, the first CCC event was launched. Since then, the competition has been running almost every year, with minor adjustments. The goal is to attract people working with artificial intelligent technologies to both target problem areas in general, such as efficient retrieval and adaptation, but also to exhibit concrete solutions. A competition with common data sets allows comparison and better evaluation of developed CBR methodologies. In this section, the competition rules will be clarified, and the motivation to join the competition is pointed out.

2.1.1 Competition rules

The competition task is to develop a system that proposes recipes based on a restricted set of ingredients. Hence, a recipe recommender system. The user of the system must be able to tell the system which ingredients they desire and which ingredients it must not allow in the proposed recipe. The participants of the CCC gets access to some data sets of initial recipes. Whether the competition is limited to a specific data set varies. The participants can freely choose their focus area within CBR. A jury of experts from AI and cooking evaluates the competing systems. The jury is given access to the systems and can explore them themselves. At some of the events, the suggested recipes have been prepared and tasted by the audience.

2.1.2 Motivation

The competition will be used as a tool to showcase CBR and AI by explaining what can be done in the context of recipe recommendations. The decision-making process is forced to be creative when working with a limited data set. The idea is to compete with other AI researchers and “open the magic box a little.” Experimenting with food recipes as experiences is a great advantage in the way that there is no sensitive data, in comparison to for example experiences with patients in a hospital. In other words, the CBR system can be demonstrated in detail to a broader community.

2.2 Case-based reasoning

The idea behind CBR is to use previous problem situations to understand and solve new problems (Aamodt 1991; Kolodner 1992). The assumption that similar problems have similar solutions is the basis for the idea. Human beings reason this way all the time (Aamodt 1995). Whenever it is more convenient or simply requires less effort to reuse experience, humans reason like this rather than going back to square one.

CBR simulates this kind of human problem-solving technique. The approach has been formalized as a four-step process: Retrieve, Reuse, Revise and Retain. These steps have come to be known as the '4 REs' (Mantaras et al. 2006) and the cycle is usually referred to as the CBR cycle.

2.2.1 CBR cycle

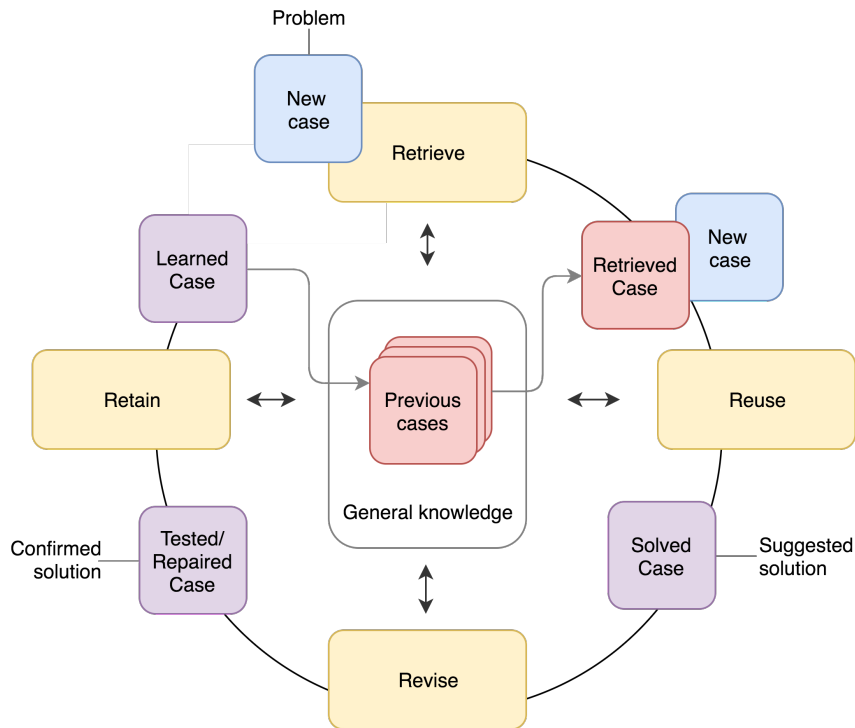


Figure 2.1: CBR cycle

An overview of the CBR cycle is illustrated in Figure 2.1. The retrieve step of solving a problem by CBR involves measuring the similarity between the new problem description to previous problem descriptions stored in a case base (Mantaras et al. 2006). After finding one or more similar matches, the solutions contained in those cases are considered as candidates for solving the current problem. Retrieval is considered the primary step in the CBR cycle and many researchers therefore focus on this step.

The reuse step involves mapping one or more of the retrieved solutions to the target problem. This process may involve adapting the best matching solution as needed to fit the new situation, or somehow combine the retrieved solutions. Similarity-based reasoning is used both when retrieving cases, and in the reuse and adaptation parts. Similarity is therefore considered a core concept in CBR. How exactly similarity between two cases is defined is domain dependent (Richter and Weber 2013).

The proposed solution for the target problem has to be tested for success, e.g. through simulation, in the real world environment or by being evaluated by a domain expert. If problems emerge, the solution must be modified accordingly. This process is called the revise step of the CBR process.

Once the new solution has been confirmed or validated, the resulting experience can be retained as a new case in the case base. In other words, the system has learned to solve a new problem. The idea is that the CBR process starts again when a new problem situation occur and this way, the system keeps on learning.

2.2.2 Typical CBR domains

CBR is usually preferred whenever it is difficult to formulate domain rules, or if the rules require more input information than is typically available at the problem-solving time (Leake 1996). Besides, exceptions from general knowledge are handled well by CBR compared to other AI approaches. CBR tends to be a good approach for complex domains where cases are available in an easy format (i.e. not cases described with natural language text) and in which there are multiple ways to generalize them. This covers a wide variety of problem-solving tasks.

CBR is not limited to the reuse of experience. For E-commerce, sales are typically recorded. If sales were recorded as experiences in the standard way humans think, old demands could be compared to new demands. However, in those scenarios, similarity is rather measured by comparing user specifications with product descriptions, where the goal is to customize an offer by finding the smallest possible gap between customer demands and product features (Richter and Weber 2013). Commercial applications have shown great success with CBR, much due to the advantage that new recommendations (i.e. solutions) can be derived from old recommendations more easily than from scratch¹.

2.3 CBR compared to other AI approaches

CBR has grown to a field of widespread interest (Montani and Jain 2014), and has become a powerful approach for computer reasoning (Brown and Gupta 1994). This section aims to show that CBR is a suitable approach for developing a recommender system by comparing it to other AI approaches. More specifically, CBR will be compared to rule-based systems and three types of recommender systems: Collaborative, content-based and knowledge-based recommender systems. The techniques employed in these types are among the most popularly used in recommender systems today (Ricci, Rokach, and Bracha 2011). The techniques have different advantages and drawbacks as will be discussed in this section. A common approach to building recommender systems is to combine techniques, using advantages of one technique to try fixing the disadvantages of others. However, combined techniques will not be considered in the comparisons.

¹www.dfki.de/web/research/km/expertise/research/case-based-reasoning

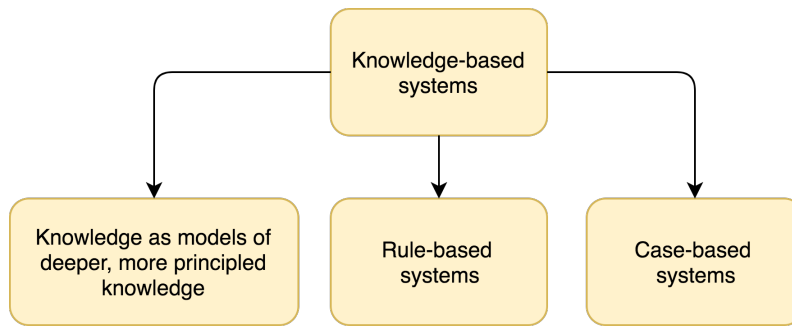


Figure 2.2: Knowledge-based systems overview

CBR is considered a knowledge-based system. Content-based and knowledge-based recommender systems also go under the term *knowledge-based systems*. In general, knowledge-based systems, also called *expert systems*, are systems using AI techniques to solve problems in specific domains. Figure 2.2 shows a common division of knowledge-based systems. How knowledge is represented in knowledge-based systems varies from simple rules to more complex models. As for these representations, this section focuses on comparing CBR to rule-based systems. Rule-based systems in the classical sense are the simplest form of AI. A set of "if-then" statements are used to make deductions or choices. Decisions are made by matching the interaction of input against the "if"-assertions. Accordingly, the "then"-part of the rules states how to act upon those assertions (Cawsey 1998; Hayes-Roth 1985). An advantage to writing everything into rules is that the human expert's knowledge within a field is captured. Even if the person retires or leaves the firm, the knowledge is not lost and can still be available for an extensive range of people².

2.3.1 Recommender systems

Recommendations are closely related to sales scenarios where customizing an offer by matching customer needs against product features is desired. However, a recommendation does not need to answer a specific demand or customer needs. The query is rather in the form of a general wish (Richter and Weber 2013). A recommendation is the result of a decision-making process, such as which items to buy, what music to listen to, or what movie to watch (Ricci, Rokach, and Bracha 2011). User preferences are taken into consideration when available.

Recommender systems are used to predict the rating that a user would give to an item. Personal tastes vary, but the predictions are based on the fact that people follow patterns. The utility function can be based on more than just item similarity, for instance, past user behavior, context, relations with other users and the fact that people tend to like the same things as similar people like.

There are several ways to develop a recommender system. A CBR approach is one of them, as will be discussed later in this chapter. However, the most well-known type of

²www.ramalila.net/adventures/ai/rule_based_systems

recommender system is the collaborative- or social-filtering type (Burke 2000). For these systems, the main idea is to exploit information about existing user’s previous behavior and use that information to predict which items the current user of the system will most likely be interested in (Jannach et al. 2010). In other words, when one user likes an item, the system can recommend that same item to similar users.

Both Kristi and Anna have bought and rated "Notting Hill," "Love Actually," and "Bridget Jones's Diary" with four or five out of five stars. After that, Anna watches "About Time." Based on the similarity in their previous purchasing patterns, the system guesses that Kristi also will like this movie and therefore recommends it to her.

With a content-based recommender system, on the other hand, the user will be recommended items similar to the ones the user preferred in the past. Items are compared based on associated features, like movie genres (Ricci, Rokach, and Bracha 2011).

Anna rates "Notting Hill" with five out of five stars. The system then guesses that she would also enjoy watching "Love Actually." The similarity between the two items is the basis for this assumption. They both have the genre romantic comedy, and they both have high ratings.

Another type of recommender system is the knowledge-based recommender system. These systems use explicit knowledge about recommendation criteria, user preferences, and the item assortment to generate a recommendation, reasoning about what products meet the target customer’s needs. Recommendation criteria are associated with the context in which items should be recommended. For instance, highlighting Halloween products on a party product site in the middle of October.

Recommender systems have become incredibly widespread in recent years. The systems are now one of the most popular mechanisms in electronic commerce and have proven to be a helpful way for online users to cope with the often overwhelming number of alternatives. Online shopping, for instance, used to be a static experience in which users searched for and potentially bought products. Due to recommender systems, it is now usually a much richer and customized experience. The popular movie streaming service, Netflix, and the popular online store, Amazon, are examples of services that employs a recommender system to personalize the experience for each customer.

2.3.2 Knowledge acquisition and construction

It has become a widely recognized problem that acquisition of high-quality knowledge is very expensive and time-consuming (Jannach et al. 2010). This problem is known as the *knowledge acquisition bottleneck* (Hayes-Roth, Waterman, and Lenat 1983; Aamodt 1991). This subsection will discuss whether or not the knowledge acquisition bottleneck is a potential problem for the mentioned systems, and if so, how crucial the problem is.

Collaborative recommender systems depend on ratings to generate recommendations (Burke 2000). Also, the number of rated items that can be associated with a given user greatly affects the accuracy of the recommendations. With only a small base of ratings, a couple of special recommendation cases can be the reason for recommendations that does not apply to the given user at all (Maes and Shardanand 1995).

Accordingly, the system will not be of service for most users without a large number of users whose habits are known. Until a sufficient number of item ratings are collected, the system cannot be useful for a particular user. Consequently, a collaborative recommender system should have a large base of ratings initially to assure accuracy.

The same type of problem applies to content-based recommender systems. To be able to provide accurate recommendations to a user, these systems need to collect enough ratings from the user. Hence, reliable recommendations can not be provided to users that have few ratings available (Ricci, Rokach, and Bracha 2011).

The problem of knowledge acquisition also applies for rule-based and knowledge-based systems (Leake 1996; Aamodt 1995). It is usually addressed by knowledge-based systems by focusing on only one type of knowledge for one application task (using a rule-based approach). However, figuring out which rules the system should depend on is challenging, and the construction of rules used in these systems requires domain-specific knowledge (Hayes-Roth 1985). Such generative systems should ideally account for all problems that are in principle possible, and there is no assurance that the rules cover the knowledge it is supposed to. The rule acquisition process is both cumbersome and unreliable.

However, a knowledge-based recommender system avoids some of the drawbacks related to collaborative recommender systems because its decisions are independent of individual user preferences. Unlike a collaborative recommender system, it does not depend on a large amount of statistical data about particular rated items or individual users. Additionally, research show that knowledge-based systems do not require as much data. These systems need only enough knowledge to decide how similar items are to each other and general knowledge concerning the context.

The problem of knowledge acquisition can be simplified with CBR. It turns out that it is easier for domain experts to recall concrete experiences which they have encountered in practice rather than define decision rules describing all possible options. In other words, their mental set evince signs of being oriented towards a CBR approach (Barletta and Hennesy 1989; Goodman 1989; Kolodner 1992).

In contrast to the construction of knowledge-based and rule-based systems, building a case base does not require expert knowledge. Also, CBR systems do not require an explicit model. Instead, the main implementation of CBR can be reduced to identify significant case features. Therefore, CBR systems work with partial knowledge and their output can be approximate answers (Richter and Weber 2013).

If cases do not come from a database or a preexisting case base, there is no fully automated technique for building a case base. However, advanced database techniques ease

the task of implementing cases. Also, as the CBR cycle in Section 2.2.1 shows, cases can be added to the case base after deployment as well as during development. Accordingly, developing a CBR system can be a lot faster and easier than constructing the same system with a rule-based or a model-based approach.

2.3.3 Generalization

Reasoning is usually described as a process of forming conclusions, judgements, or inferences from facts or premises that are chained together as generalized rules (Leake 1996). Case-based systems have a very different way of generalizing knowledge. In the retrieval part of the CBR cycle, specific previous situations in the case base are partially matched to the new problem case. This partial matching can be seen as a matching at a more general or abstract level, or, an implicit generalization process (Aamodt 1995).

2.3.4 Maintenance

Failing or detonating systems or components cannot be ignored, and the need for maintenance is evident in many CBR systems (Roth-Berghofer 2003). In rule-based systems, modification of knowledge bases can be complicated (Aamodt 1995), especially if the rules are not written clearly or if the maintenance is done by someone unqualified. This is among other factors due to possible dependencies between rules, redundant rules, and the risk of creating contradictions³ (Liao, Zhang, and Mount 2010). Difficulties in maintaining and updating a system's knowledge have resulted in research on new methods such as capturing domain knowledge as models of deeper and more principled knowledge, instead of sets of if-then rules (David, Krivine, and Reid 1993).

For knowledge-based recommender systems, new information is discovered by inference, while with CBR, new knowledge come by adaptation (Richter and Weber 2013). Maintenance by CBR is both easier and more systematic than by knowledge-based systems. A CBR system has mainly two sources of change: 1) Changes due to the use of the system such as learning a new case and 2) changes in the environment such as in legislation, in technology, or in norms. When adding a case as explained in the retain step of the CBR cycle, it does not need any further checking or debugging.

However, although one added case does not affect the functionality of the system, it may affect its outcome. Therefore, validation is necessary. Serious performance problems can arise if the case base grows large without being controlled. Retrieval efficiency weakens and redundant, incorrect or inconsistent cases can be a consequence which will be hard to detect.

The main idea behind CBR maintenance is to prevent any undesired changes in the system. The goal of maintenance is not to make improvements, but merely to be able to restore or delay a previous state of operation. Accordingly, it is challenging to improve

³www.knowledgeengineering.blogspot.no/2007/12/case-based-reasoning-cbr-vs-rule-based

system performance if the system is initially poorly designed. Consequently, maintenance has to begin early in a project.

There are alternative case base maintenance policies, regarding how maintenance operations work and when the various maintenance policies are triggered (Leake and Wilson 2001; Burke 2000). Maintenance operations may be done periodically (e.g., each time a new case is added to the case base), conditionally (e.g., when the number of times cases have been retrieved has reached a prespecified limit), or on ad hoc basis (e.g. by unpredictable intervention by a human maintainer). The periodically triggering is the most common and is explained as the revise step of the CBR cycle. It is important that the system's performance is observable and that the maintenance is designed with clear and convincing criteria and control conditions. Current CBR maintenance research primarily focus on improving case-based reasoners and, in particular, case bases.

2.4 Discussion

Based on the related work presented in this chapter, it seems that CBR is a better approach for developing a recipe recommender system than rule-based reasoning, especially when it comes to the time and effort required. Chefs over the entire world create thousands of recipes every year. It would have been extremely challenging to get a grip on the whole domain of cooking. Even if a considerably smaller design than this is considered, boiling the knowledge down to operational recipes will be challenging.

Reuse of prior solutions, both successful solutions and failed experiences, helps increase problem-solving efficiency by avoiding repetition of prior effort (Leake 1996). There are also many exceptions and specific combinations when it comes to which ingredients and flavors go well together, which accordingly will be handled better by a CBR system. As shown in this chapter, CBR seems to be a better approach than rule-based reasoning when it comes to knowledge acquisition, construction, generalization, maintenance, and revision. Since the focus is the composition of food and ingredients and not individual user preferences, the idea of using a collaborative recommender system or a content-based approach is dismissed.

By employing CBR in a recipe recommender system while targeting the CCC task description, the new problem will be the desired and undesired ingredients. The problem to be solved is to find stored experiences with food (i.e. a recipe) that will fit the query. Hence, recipes are seen as specific situation experiences on how to combine ingredients. The recipes in the case base will be the previous problem experiences that are already solved.

The idea that recipes are seen as experience by CBR seems like a great advantage that could not be handled better by any knowledge-based recommender system. The CCC provides a set of initial recipes which can easily be transferred to cases in a case base. If domain knowledge were to be captured as required by knowledge-based recommender systems, both the knowledge acquisition and construction part would have been much more challenging and time-consuming.

CHAPTER 2. BACKGROUND

Also, there are clear indications that maintenance is easier with a CBR approach since the system only needs to handle problems that occur (e.g. the dish type *salad*), while generative systems must account for all problems that are principally possible (i.e. must account for all dish types possible).

To find the best matches in the case base, the system needs a way to compare recipes. Therefore, important features of a recipe must be identified. Also, the system must measure the similarities between all features to be able to calculate how similar two recipes are. Chapter 3 looks into previous CCC participants' challenges and solutions, as well as other recipe recommender systems that does not target CBR.

Chapter 3: Related work

The goal of this chapter is to provide an overview of the state of art research within the field of computer cooking, focusing chiefly on the relevant work of earlier contributions to the CCC in Section 3.1. Further, other approaches to the computer cooking domain are presented in Section 3.2. Lastly, the chapter provides a discussion of the presented work in Section 3.3.

3.1 Competitors of the CCC

Several research groups have contributed to the CCC over the years using information retrieval, information extraction and semantic technologies along with CBR when developing recipe recommender systems. The researchers have contributed with various approaches to the task. This section introduces four of the most influential systems.

3.1.1 Taaable

The Taaable team has contributed to the CCC six times, incrementally improving the system. The system is built over a generic CBR engine called Tuurbine (Gaillard et al. 2014). The researchers aim to improve the efficiency of managing data within the system by using a homemade semantic wiki, called WikiTaaable (Blansch e et al. 2010).

WikiTaaable serves as the main knowledge base and is composed of a cooking domain ontology, a recipe base indexed by concepts, as well as else-wise knowledge which has appeared useful for efficient retrieval and adaptation (Badra et al. 2009). The domain ontology contains four concepts: ingredients, dish moments, dish types and dish origins. Further, ingredient knowledge is stored in a hierarchical taxonomy which describes the relationship of ingredients. For example, *apple juice* is a subclass of *fruit juice*.

A complex generalization-specialization method extracts necessary knowledge about which ingredients can substitute for each other in various contexts (Gaillard, Lieber, and Nauer 2015). When the system receives a query, the system first tries to find an exact match within the recipe base. If no match is found, the system aims to find the minimal generalization function in a way that there exist at least one recipe exactly matching the

generalized query. For example, the ingredient *mango* may be generalized into *tropical fruit*, which can be specialized into *fig*.

Ingredients are given a similarity score based on their position in the taxonomy. Child nodes are given a similarity of 1.0 to their parents. This means that a search including the generalized term *tropical fruit* is seen as a perfect match with *fig*, *mango*, and all other children. Mango and fig and the other hand have a similarity of 0.6 due to their sibling relationship.

The 2010 version of the system was expanded with nutritional values for all ingredients. By adding this to all ingredients, amounts of the ingredients are adjusted due to the modified ingredients nutritional values. For example, when substituting mango with fig, the amount of sugar can be decreased.

Another expansion of the 2010 version was the adaptation of *preparation steps*. The adaptation is made by adding natural language processing and Formal Concept Analysis. Preparation steps do not necessarily stay the same after ingredients are modified. The idea is that, for each ingredient, there exists a preparation prototype describing the different ways the particular ingredient can be used. Hence, *target sets* of ingredients are built. An example of such a target set is *sauce*. Also, the sequence of actions applied to any ingredient is attempted to be identified. FCA is further used to group similar measures that might not necessarily replace each other, like *peel* and *remove-pit*, or *cut* and *slice*. Hence, the method makes it possible to replace the textual preparation steps in the original recipe by retrieving the preparation steps from the recipe description in which the substitute ingredient belongs.

3.1.2 CookIIS

The developers behind the CookIIS system has contributed to the CCC three times. The system was built using the industrial strength tool Information Access Suite (e:IAS). In the second version, the researchers focused on collection adaptation knowledge from online cooking communities (Ihle, Hanft, and Althoff 2009). In the latest version of the system, the main goal was to improve the performance of the CBR '4 RE' processes (Newo et al. 2010). The focus was pre-processing to enrich the source data.

The knowledge model consist of several taxonomies. Each taxonomy represent a categorization, such as *fish*, *meat*, or *fruit*. Additionally, every ingredient is represented by at least one concept. For example, meat can be organized by both parts (e.g. fillet) and by kind (e.g. pork). All preparation steps and tools required for the preparation are also modeled in separate category taxonomies.

The system builds upon the recipes provided by the CCC, which are divided into one XML file per recipe. During project setup, the system automatically recognizes different aspects of each recipe. The ingredient type is recognized based on the categories in the knowledge model. Type of meal is recognized based on indicative keywords in the recipe title along with indicative ingredients or combinations of them. Type of cuisine is recognized

by three ruled based approaches: 1) identification of the recipes origin in the recipe title, 2) identification of characteristic strings in the recipe title and mapping them to an origin, and 3) identification of occurrences of spices, herbs or other ingredients to find characteristic elements for the given type. After this process, each recipe is defined by the taxonomy categories and the belonging ingredients.

To gather adaptation knowledge beyond the taxonomies, the researchers collected concrete pairs of ingredients given as advice in comments on online cooking communities. Also, a rule engine is used to detect similar ingredients in the recipe base.

The system performs a pipeline of occurrences when receiving a query. The pipeline aims to retrieve and adapt recipes from the case base. The user query is analyzed, and concepts are extracted using a text miner before meta information is computed based on the query input. The retrieval continues by collecting the most similar cases based on the calculated similarities.

To be able to assess dietary practices, the system uses a filter to sort out the applicable ingredients. The filter ensures that recommended recipes do not contain any undesired ingredients.

The similarity between a query and a case is calculated with local and global similarities. The system uses a weighted generalization-specialization method to compute the similarity between ingredients within a category. The weights for generalization and specialization is given so that the similarities can be automatically computed. In addition to the weights computed with the taxonomies, the system uses table based similarity measures for some ingredients, because the taxonomies do not reflect the reality in every case. These were manually added by the researchers. Each category is given a weight which reflects the overall importance of the category. The global similarity measure for a recipe is the weighted sum of all the local similarities of the categories in the case.

The adaptation process of a recipe goes sequentially through three steps. First, the system considers the community-based adaptation rules gathered from online cooking communities. Second, model-based adaptation rules are applied. These are the rules that are extracted from the taxonomy. Lastly, the system carries out the actual adaptation of the recipe in five steps: 1) searching for ingredient definitions in the text, 2) replacing any plurals with singular form, 3) considering ingredient synonyms, 4) simplifying two-word-concepts (e.g. generalizing peanut oil into oil), and 5) replacing the old ingredient with the new.

3.1.3 CookingCAKE

CookingCAKE is mainly based on Collaborative Agent-Based Knowledge Engine (CAKE) (Freßmann et al. 2005), which provides efficient ways of information retrieval and sophisticated similarity calculations (Fuchs et al. 2009).

During development of the later versions of the system, the focus was mainly targeted on the *preparation instructions*. This means adapting the cooking instructions in the form

of *cooking workflows* (Minor et al. 2010). With this approach, a recipe is represented as a workflow which describes the process of preparing a particular dish. The cooking workflows consist of a set of preparation steps (tasks), and a set of ingredients (data items) shared between its tasks. Control-flow blocks like parallel (AND), alternative (XOR), and repeated execution (loops) are also used. All tasks and control flow blocks are linked by control-flow edges, which defines an execution order.

The developers behind CookingCAKE built a hierarchical and category-based ontology in XML. The ontology consists of a taxonomy of ingredients to define the semantics of data items, and a taxonomy of preparation steps to define the semantics of tasks. The structure is based on an analysis of the recipes provided by the CCC, which gave the developers the idea of three types of nodes: category-nodes, ingredient-nodes, and synonym nodes (to represent typos, British versus American English, and so forth). Five origin types were further created: animal, herbal, baking, liquid, and spices. These origin types serve as category nodes, and all ingredient nodes are placed below them. Ingredients were extracted from the recipe case base by filtering stop words, cooking units, and state of the ingredients. For example, *1 lg Onion (sliced and quartered)* was transformed to *Onion*.

To be able to compare and adapt recipes on a higher level, the researchers decided to create a classification model on specific cuisine and meal types. To classify recipes and ingredients, the developers manually selected 70 ingredients and 130 recipes for the type of meal (e.g. breakfast, lunch), and 80 ingredients and 90 recipes for the type of cuisine (e.g. stew, salad). RapidMiner 4.3, which is an open-source Java based mining solution, along with a Naive Bayes classifier, was further used to build a classification model out of the pre-classified recipes and ingredients. Then, all recipes in the database were automatically classified using the classification model.

CookingCAKE involves a PHP-based User Interface (UI), where the user can specify any desired and undesired ingredients or preparation steps. The UI also include check boxes for dietary practices, type of cuisine, and type of meal. The system uses Process-Oriented Case-Based Reasoning Query Language (POQL) to capture the query provided by the user (Müller and Bergmann 2015). The query is further used to guide the retrieval, trying to find a workflow case that contains all desired ingredients and none of the undesired ingredients.

The taxonomies of ingredients and preparation steps are used to evaluate the similarity between ingredients and preparation steps. The similarity is calculated by mapping each query element to the case elements. The system can estimate the similarity between cases using the ingredient taxonomy; the closer the ingredients are in the taxonomy, the more similar they are. Say that a query contains the general term *meat*. Then, the immediate children, such as *beef* or *pork*, are given a similarity of 1.0 to the query.

The system uses two approaches for preparation step adaptation. The first approach involves generalization and specialization of workflows. This process is done by comparing similar workflows from the case base. For example, if three workflows contain beef, chicken, and pork, the system can generalize to *meat*. Secondly, the system uses workflow streams to adapt recipes. Workflow streams are based on the idea that each workflow can be de-

composed into sub-components. For example, in a sandwich, the sauce can be prepared separately, and hence, substituted as a whole.

To generate such adaptation knowledge, CookingCAKE uses the workflows contained in the case base. The cases are first generalized, then made into workflow streams and adaptation rules. This process is automated, and solutions are stored for future adaptation processes.

Modified cases are represented by the original workflow along with a solution part which contains the adapted workflow and with a description of the steps that are done to transform the workflow. All cases used for experimenting was created by hand by the researchers. The researchers found that the hard part in the reuse phase was finding the location of where to make the changes. Also, the generation of natural language cooking step descriptions has not yet been addressed.

When analyzing the cooking steps, the researchers found that the level of detail was not consistent, and the instructions lacked a uniform language and phrasing. Further, representation of the amount and state of ingredients was found challenging as there is no uniform representation of such information. Lastly, the researchers discovered difficulties with substituting ingredients where the old or new ingredient is a so-called aggregated ingredient (e.g. whipped cream, which consists of cream and sugar). These are obstacles that the researchers have yet to solve.

3.1.4 JaDaWeb

The JaDaWeb project is implemented with the jCOLIBRI framework, which supports textual processing and the use of ontologies (Herrera et al. 2009). In their latest contribution, a graphical, web-based, natural language UI was added (Ballesteros, Martín, and Díaz-Agudo 2010). To clarify, JaDaWeb does not adapt recipes. The system focus on optimizing the retrieval process and similarity calculations.

JaDaCook reasons using the case base of recipes provided by the CCC along with an incrementally built cooking ontology and a set of association rules. The association rules are obtained using data mining techniques to capture co-occurrences of ingredients in the recipes.

The ontology consist of a hierarchical taxonomy where ingredients are organized into categories to enable the possibility of inheritance of properties. Ingredients are classified as one of the following: 1) origin ingredients like fish, meat, milk, and eggs, 2) plant origin ingredients like cereals, nuts, fruits, and vegetables, and 3) *all other classes* like drinks, sweeteners, oils, and salt.

The taxonomy is mainly used for calculating ingredient similarity, with the assumption that two ingredients are more similar if they are closely located in the taxonomy. Association rules are used to propose suggestions for similar ingredients, along with a case base of menus.

The case base is built collaboratively using non-expert users feedback and is used to build meaningful three-course menus by combining single dishes.

JaDaWeb takes user input as a single sentence in natural English language. It then uses a MiniPar parser to understand the natural language query. The parser returns a dependency tree with all nouns from the query sentence and also differentiates between desired and undesired ingredients by detecting negation in the query. Two dependency tree lists then represent the query: 1) desired ingredients, and 2) undesired ingredients.

After parsing the query, all nouns (i.e. ingredients) in both lists are looked up in the lexical-semantic net WordNet to find their definition, synonyms, antonyms, and so forth. Further, the algorithm searches for all the ingredients, including their synonyms, in the taxonomy. If an ingredient is not found, the system uses the noun and ingredient definitions to find a similar or matching node in the taxonomy to place the new ingredient. If no suitable node is found, the system asks the user to help classify and put the ingredient where it belongs.

The Apriori algorithm was applied to find a relationship between ingredients. It is used to mine association rules using Weka (a collection of machine learning algorithms for data mining tasks) over the recipe case base. Weka sorts rules according to metrics like confidence, leverage and lift. For example, a rule can say that sugar can substitute for vanilla with a confidence of 87%.

The total similarity score of a recipe is normalized by the number of ingredients in the query. If an ingredient is not in the recipe, all siblings and parent ingredients are checked. The similarity metric is adjusted according to the hierarchical relationship, for example, siblings are consequently given a similarity of 0.8. Lastly, JaDaWeb also includes a fuzzy table of ingredients defined in a XML file. Here, for example, macaroni and rice are manually given a similarity of 0.4.

Finally, the recipe suggestions are ordered by similarity as well as season characteristics, for example, soups go well in the fall and winter, while salad is more suitable for spring and summer. Season features are subtracted from words like *cozy*, *warm*, *light*, and so forth in the recipe description.

3.2 Non-CBR related work

This section studies and discuss software approaches to the computer cooking domain where CBR has not been applied to the solutions. These are all commercial systems, specially developed to reach out to as many users as possible.

3.2.1 AllRecipes

AllRecipes¹ is one of the most popular and extensive websites for sharing of recipes, with more than forty-seven thousand recipes collected since 1997. Here, both novice and expert cooks can share and rate cooking recipes.

The website itself does not implement any AI technologies, but in 2012, a team of researchers experimented with ingredient networks on the data set of recipes downloaded from AllRecipes.com (Teng, Lin, and Adamic 2012). The idea came from the thought of how the massive database of recipes could provide significant information about which ingredients go well together based on their co-occurrence in recipes. Also, user reviews of recipes could provide clues as to the flexibility of a recipe when it comes to leaving out ingredients, modifying the quantity or substituting an ingredient with another.

The team created two networks to capture the association between ingredients; 1) a complement network to capture which ingredients tend to co-occur, and 2) a substitute network derived from user-generated suggestions for modifications. One of their overall goals was to measure whether an ingredient is essential for a recipe, along with if and how its quantity can be modified.

The team processed every recipe and its comments to create the networks. The complement network was constructed based on point-wise mutual information, defined on pairs of ingredients extracted from the recipes themselves. While setting up the substitution network, the team parsed all reviews, split them on punctuation and stop words, and used simple heuristics to detect when a review suggested modifying a recipe. Only suggestions mentioned more than five times were included. This process resulted in a weighted, directed network consisting of ingredients as nodes. The weights are based the proportion of substitutions of ingredient a that suggested ingredient b. For example, 68% of substitutions for white sugar were to Splenda. Hence the assigned weight for the sugar-Splenda edge is 0.68.

Analyses and visualization of both of the networks revealed clear patterns and strong clusters in the relationship of ingredients and also revealed regional preferences when it comes to ingredients and cooking methods. In the substitution network, it was found that additions are positively correlated with increases and deletions with decreases. Healthy ingredients such as sugars and fat are more likely to be exchanged or decrease, while flavor ingredients such as cinnamon and toppings are more apt to be increased. Recipe frequency is negatively correlated with modifications of the recipe. For example, salt appeared in over 21 000 recipes and was only modified in 18, while Worcheshire sauce appeared in 1532 recipes, and was dropped explicitly in 148 reviews.

¹www.allrecipes.com

3.2.2 Chef Watson

Watson² is a cognitive computing application developed by a research team concerning a project at IBM. Originally, Watson was created to answer questions on the quiz show *Jeopardy*. In the later years, several applications have employed the software; utilizing management decisions within lung cancer treatments and recommending new songs to users in a music application, are just two amongst more (Upbin 2013).

In 2014, IBM and Bon Appétit³ started working together, with the goal to build a creative cooking application that could help discover new recipes: Chef Watson⁴. Bon Appétit provided their database of over ten thousand recipes, in which Watson searches for patterns. Technically, the biggest challenge when creating Chef Watson was natural language processing. In other words; how to understand the recipes, and extract the relevant information into a knowledge base (Ferrucci et al. 2010). The computer attempts to combine the recipes into new, unexpected combinations by looking at pairings of ingredients that go well together. The system also uses extensive knowledge of the science behind food pairings to combine the recipes (Pinel 2015).

The outcome of the recipe generator has been widely tested, both by expert chefs and novice cooks. The results have been surprising and fun, and (mostly) well-tasting. Dishes like the *Vietnamese apple kebab* including pork, apples, mushrooms and strawberry is one of them. In the recipe, ingredients share similar flavor compounds even though it does not sound very appetizing at first.

3.2.3 Yummly

Yummly⁵ provides personalized recipe recommendations, semantic recipe search, a digital recipe box, and shopping lists. The application had fifteen million active users in the US in 2014 (Goldfisher 2010). The website and application allow users to filter their search by ingredients, diet, nutrition, price, cuisine, taste, time, allergy and much more, and aims to *learn* what the users likes and dislikes. The site also calculates nutritional values. The preferences are stored, and the information is further used to categorize food and recipes to make personalized recommendations in the future. The more it is used, the better the recommendation.

In March 2013, Yummly opened full access to its Application Programming Interface (API) as a paid service for other companies. This API allow searching for ingredients, cooking methods and nutritional data (Fitchard 2013). With more than one million recipes and one hundred thousand classifications, the API serves a great opportunity as a basis for endless of applications.

²www.ibm.com/watson

³www.bonappetit.com

⁴www.ibmchefwatson.com

⁵www.yummly.com

3.3 Discussion

The cooking domain is widely targeted within research with AI. By targeting the cooking domain, it does not necessarily exist a correct solution to any problem, which provides the contingency of being creative. Also, the cooking domain has a broad target group. After all, everyone has to eat, and many people use some cooking recipe service several times per week, may it be online or in a textual book.

In Section 3.2, three commercial products were presented. When it comes to these, entirely new solutions were implemented with the sole goal of enlisting as many users as possible. The results are astonishing, and all of the systems have reached success with a great number of sales and daily users. This factor is just another argument to the question of whether or not cooking is a convenient domain to target.

Chef Watson has proven, even to expert chefs, that a computer program can detect patterns in ingredient complementarity, and hence, suggest entirely new and creative dishes that fulfills culinary quality. The research on complementary ingredients on the AllRecipes.com database has yet again proven that there does indeed exist clear patterns within ingredients networks. Edges in the networks can indicate patterns within several different concepts: meal type, season, culture, regional preferences, and so on. Combining such extensive cooking knowledge with for example the personalized recipe recommendations that Yummly provides, could further tailor the recommendations targeted towards a specific user.

In the CCC, tasks given varies from year to year, but the competition usually includes an open challenge. For the open challenge in the CCC, the focus is adapting the cooking recipes in any way possible, as long as the recipe base provided is used as a basis. Evaluation criteria for the emitted system evolve around the scientific quality of the solution, along with culinary quality and originality.

It is mainly the solutions to the open challenges, along with the general CBR system implementations that were explained in Section 3.1. These four systems are designed and implemented for the purpose of research within CBR, and all showcase new ways of approaching CBR in a domain specific solution. They all follow the CBR cycle: retrieve relevant recipes, reuse earlier experiences, revise the given solution, and retain the new case in the case base.

When comparing the systems, it appears that the implementations are widely dissimilar in many ways, although they share common CBR principles. The Taaable researchers built their system around a collaborative, semantic wiki, which also serves as the main knowledge base. The CookIIS researchers focused on the preprocessing of data to make the substitution of ingredients fluent and more realistic. CookingCAKE targets the preparation instructions by implementing cooking workflows, and lastly, the JaDaWeb researchers focused on the implementation of natural language understanding.

All the CCC systems include some cooking ontology, which is interesting to discuss as this is the main knowledge base for ingredient similarity measurement. Taaable has a simple

hierarchical taxonomy of ingredients, not separated in any categorizations. CookIIS include one taxonomy per ingredient category, and CookingCAKE and JaDaWeb have taxonomies split into categories. Taaable, CookingCAKE, and JaDaWeb calculate the similarity between two ingredients by measuring their nearness in the taxonomy. CookIIS on the other side has specified each ingredient's similarity to its closest ingredient nodes, and the similarity measures are more specialized than with the other approaches. On the other hand, CookIIS can not measure the similarity between ingredients in different taxonomies. For example, salmon and ham is located in separate taxonomies and can hence not substitute for the other, which can sometimes be a troublesome drawback.

The results from earlier CCC systems have varied plenty, due to the complexity of the given input. Typically, challenges have arisen when working with smaller case bases. With smaller cases bases, the systems simply do not have enough freedom to find suitable substitutions or to combine recipes in a pleasing culinary fashion. For example, substituting spinach with ruculla in a salad can happen seamlessly, while completely changing the essence of the salad or swapping chicken (i.e. the main ingredient), is seen as a big challenge.

The thought of more radical adaptation reveals an opportunity for creativity. Can recipe recommender systems be implemented in such a way that the recipes are modified and combined in a more creative way, by, for example, not restricting ingredient substitutions to non-key ingredients? If a system is given the freedom to modify ingredients not specified in the user query, this opens for changing the nature of the recipe entirely.

Another key difference to note between the commercial systems and the CCC competitors, is the UI. The commercial systems have implemented well thought-through designs and easy-to-use functionality, which helps them reach out to a greater target group. The CCC competitors have not focused on usability, and it can be challenging for an inexperienced user to understand the thought process of the system. Hence, creating a user-friendly and self-explanatory UI can help a system to stand out in the CCC.

Chapter 4: Method

This chapter explains this project’s contribution to CBR research. To conduct the research, a CBR based recipe recommender system was implemented. The system is called *IntelliMeal*. The recommendations are based on a user query consisting of desired and undesired ingredients.

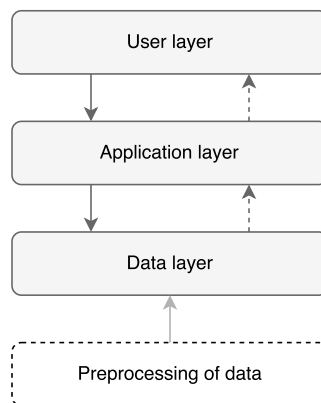


Figure 4.1: Three-layer architecture

IntelliMeal employs a three-layer architecture as illustrated in Figure 4.1. Preprocessing of data and the user layer is explained in Chapter 5, while this chapter focuses on the data layer and the application layer. Section 4.1 introduces the data used in IntelliMeal and explains how it was modeled. The following sections consider the application layer and the usage of CBR in the system. That is, what happens between the user query and the result recipes recommended to the user, as well as how the system can improve by considering feedback from the user. First, an overview of the implemented version of the CBR cycle is given in Section 4.2. After that, each process step introduced in the overview is explained in separate sections.

4.1 Data modeling

The CCC described in Section 2.1 provided access to twenty-one sandwich recipes represented by a XML file. This exact recipe data was chosen because of the narrow domain it represents. A small data set is easier to get a grip on and evaluate it. Besides, it forces the system to be more creative when customizing recipes to fit a query.

The researchers behind CookIIS, an earlier participant of the CCC, provided access to the knowledge base used in their system. The CookIIS knowledge base was modeled in myCBR. MyCBR is a similarity-based retrieval tool which provides a software development kit (SDK) and an accompanying workbench. The project contained hundreds of ingredients and similarity knowledge between these. Gathering such knowledge is time-consuming and not within the time span of this project. Therefore, the choice to take advantage of the existing myCBR knowledge base came naturally.

In this project, the workbench was used to expand and improve the already existing myCBR knowledge model. In the myCBR project, a concept is defined, and within the concept, there is a flat structure of attributes or *ingredient categories*. Making the myCBR project more suitable for IntelliMeal, some adjustments were made. This section explains the resulting data model, taxonomies, and similarity knowledge in the project.

4.1.1 Attributes

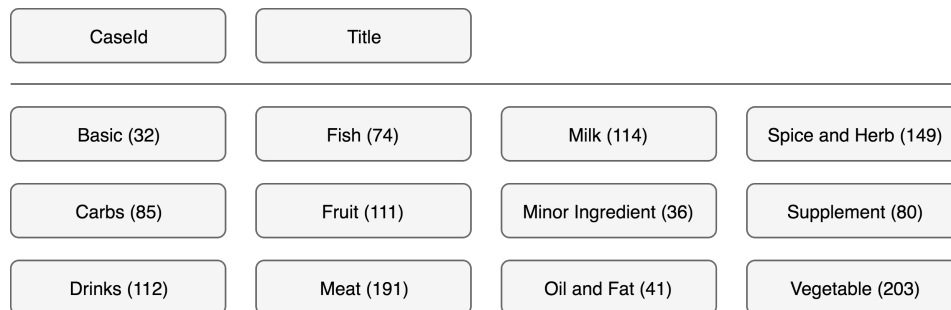


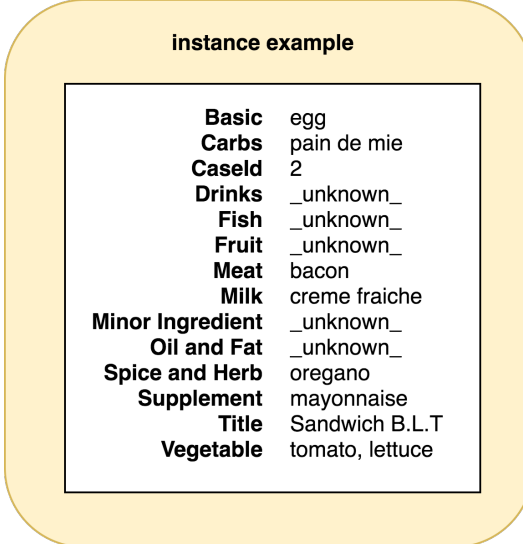
Figure 4.2: myCBR attributes

Figure 4.2 shows the attributes in the myCBR project. The top two attributes, *CaseId* and *Title*, are explaining attributes. They explain the recipes but do not say anything directly about the content. For these attributes, there is no predefined value range, and they do not take part in retrieval processes.

The remaining attributes are retrieval attributes, containing particular types of ingredients. In Figure 4.2, the number of predefined ingredients per attribute is specified. The *Vegetable* attribute, for example, contains 203 vegetable ingredients.

4.1.2 Case base

A myCBR concept can contain several case bases, and for this project different case bases was used for various purposes. For example, one case base was explicitly used for evaluation in addition to the main case base. The main case base initially contains twenty-one original sandwich recipes.



instance example	
Basic	egg
Carbs	pain de mie
Caseld	2
Drinks	_unknown_
Fish	_unknown_
Fruit	_unknown_
Meat	bacon
Milk	creme fraiche
Minor Ingredient	_unknown_
Oil and Fat	_unknown_
Spice and Herb	oregano
Supplement	mayonnaise
Title	Sandwich B.L.T
Vegetable	tomato, lettuce

Figure 4.3: Instance example

Instances can be imported to myCBR through CSV files. When added to a case base, an instance is referred to as a *case*. An instance example is illustrated in Figure 4.3. The initial sandwich recipes were only provided in a XML file which had to be parsed to CSV. This step is considered preprocessing and will be further explained in Section 5.2.1.

When importing recipe instances through a CSV file, the ingredients are matched against the myCBR model. If an ingredient in the CSV file does not exist in the model, it is simply ignored. Therefore, it is important to make sure all recipe ingredients are present in the model. All missing ingredients were added to the model as a part of the initial setup. If a recipe does not specify any ingredients for an attribute, the attribute is configured with the special value `_unknown_`, as seen in Figure 4.3. Note that all retrieval attributes can contain multiple ingredients for a case.

4.1.3 Similarity measures

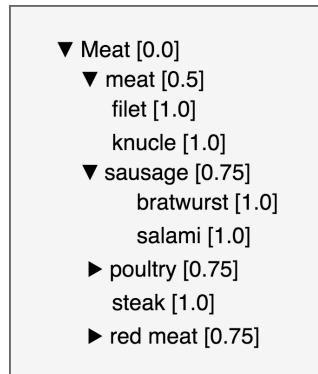


Figure 4.4: Excerpt from the Meat taxonomy defined in myCBR

Each retrieval attribute has its own taxonomy hierarchy that defines the similarities for its belonging ingredients. Figure 4.4 shows an excerpt of the active taxonomy for the meat attribute. By accessing the taxonomy, the similarity between attributes in two different instances can be fetched. If an attribute contains several ingredients, an average is calculated.

Attribute	Discriminant	Weight
Basic	true	1.0
Caseld	false	0.0
Carbs	true	1.2
Drinks	true	1.0
Fish	true	1.5
Fruit	true	1.0
Meat	true	1.5
Milk	true	1.0
Minor Ingredient	true	1.0
Oil and Fat	true	1.0
Spice and Herb	true	1.0
Supplement	true	1.0
Title	false	0.0
Vegetable	true	1.0

Figure 4.5: Global similarity measures as defined in myCBR

Global similarity refers to the similarity between a query and an adequate instance and is calculated by the weighting of local similarities. In other words, the importance of each attribute can be defined. In this project, meat and fish are weighted 1.5, while carbs (e.g. bread) is weighted 1.3, as Figure 4.5 illustrates. The *discriminant* column describes which attributes are considered in the retrieval process, and as the figure shows, *CaseId* and *Title* are set to false. When calculating the global similarity score between a query and a case the value is normalized, giving a score between zero (not similar) and one (very similar).

4.2 IntelliMeal's modified CBR cycle

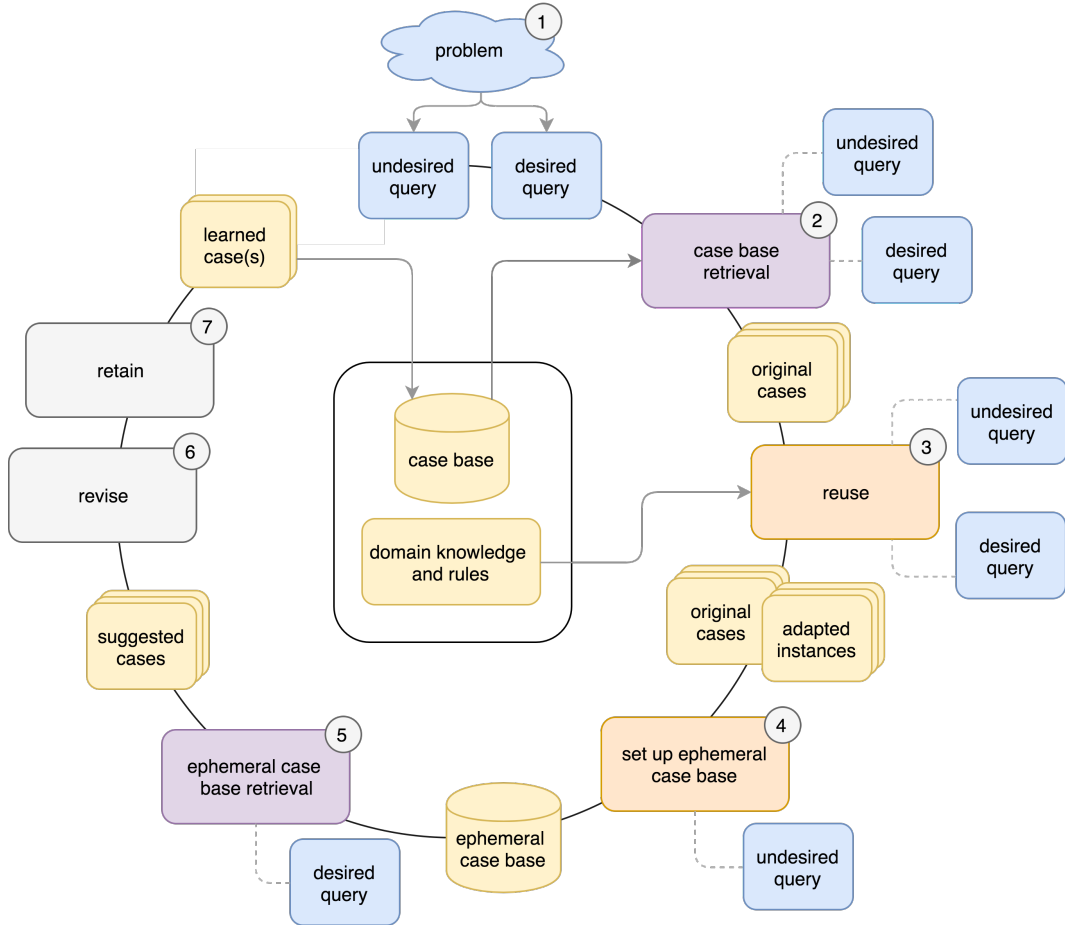


Figure 4.6: Modified version of the CBR cycle

All steps of the CBR cycle are employed in IntelliMeal. However, not in the order of the traditional '4 REs'. Figure 4.6 shows an overview of the implemented version of the CBR cycle. This section briefly explains each step in the figure and refers to following sections in this chapter for more detailed descriptions.

Step 1 The problem presented is the user query, which consists of desired and undesired ingredients. As Figure 4.6 illustrates, the query is split in two: One undesired query containing the undesired ingredients and one desired query containing the desired ingredients. Section 4.3 explains the problem in detail.

Step 2 As Figure 4.6 illustrates step two involves retrieving the cases from the case base that have the best match to the user query. Hence, cases with the best *starting point* to end

up in successful recipe recommendations. Section 4.4 explains how retrieval works in our system in general. The case base retrieval process is explained in Section 4.5. Further, the retrieved cases are duplicated. The retrieved original cases are kept for later use while the duplicated versions are considered in the reuse step.

Step 3 The reuse step is the most comprehensive step of the cycle. The goal is to customize cases (i.e. recipes) so that they better fit the user query. However, with restrictions to avoid distasteful recipe results. The customization is referred to as *adaptation*. As Figure 4.6 illustrates, domain knowledge, rules, and the queries are employed in the adaptation process. In general, the process involves three steps: 1) Adaptation with the undesired query, 2) adaptation with the desired query and 3) suitable adaptation. These processes and the reuse step are explained in Section 4.6. The result from the reuse step is adapted versions of the cases in the case base, further referred to as adapted instances.

Step 4 The strategies used to set up an ephemeral case base mainly concern using the undesired query to discard cases that are not satisfying. As Figure 4.6 illustrates, both original cases from the case base as well as the adapted instances resulting from the reuse step takes part in the setup. The result is an ephemeral case base containing a selection of both types. This step is explained in Section 4.7.

Step 5 The ephemeral case base retrieval involves comparing the cases in the ephemeral case base to the desired query. The result is a mixture of original cases and adapted instances, together with their resulting similarity score. Section 4.8 explains the ephemeral case base retrieval.

Step 6 The revise step involves getting feedback on recommendations. More specifically, the user gets the opportunity to confirm that adapted recipe recommendations are tasty.

Step 7 Only when an adapted recipe is confirmed tasty, the recipe instance will be added as a case to the case base together with the original cases. This refers to the retain step of the cycle. Both step 6 and 7 are described in Section 4.9.

4.3 Problem description

As explained, the user query consists of desired and undesired ingredients. In reference to the CBR cycle, this query is the new problem that the system has to solve. A retrieval process in CBR involves comparing a query instance to cases to find the best matching cases. However, the problem consists of two opposite types of ingredients: desired and undesired. Recipes should be a good match to the desired ingredients, but not to the undesired ingredients.

Accordingly, the desired and the undesired ingredients can not be mixed into one query instance to be used in a retrieval process.

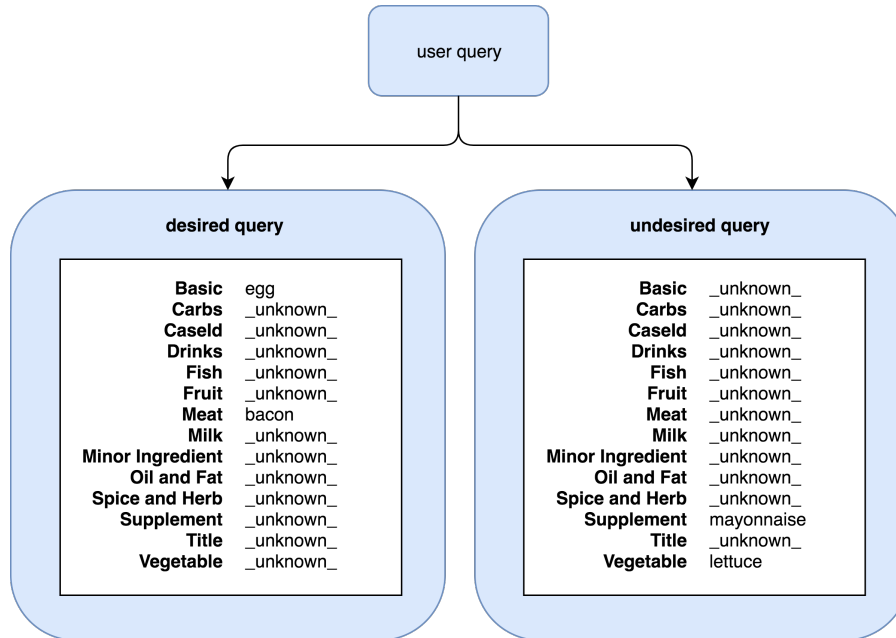


Figure 4.7: Splitting a user query

Consequently, the user query is divided in two: One desired query and one undesired query. The desired query represent the “perfect” recipe instance containing the ingredients from the desired search field. The undesired query represents the absolute worst possible recipe instance that can be presented. Figure 4.7 shows a user query example. Egg and bacon are the desired ingredients, while mushroom is an undesired ingredient.

The undesired query is considered in the case base retrieval, reuse step and when creating the ephemeral case base to strictly avoid recommending recipes that contain any undesired ingredients. However, it needs to be expanded a bit before usage. In myCBR, *mushroom* is only defined as one ingredient. However, we assume that when mushroom is defined as an undesired ingredient, this means the user does not want fossil sponge, chantarelle, or any mushroom. In general, this means that the system needs to fetch all children of all undesired ingredients and add them to the undesired query as well. Children of ingredients are found in the attribute’s taxonomy hierarchy in the myCBR project.

The desired query plays a major role in both retrieval steps and in the reuse step. It is also used when creating an ephemeral case base. The desired ingredients are however not considered as strict as the undesired ones. The user can not be ensured that all desired ingredients will be included in the presented recipes. If that was the case, queries could require the system to modify cases uncontrollably much which probably would result in

recipes that are not very tasty. Nonetheless, the system will try its best to customize recipes, but with limitations to avoid undesired results.

4.4 Retrieval

This section starts out by describing how the retrieval method incorporated in myCBR works, to give grounds for why we have implemented our own retrieval method. Further, the implemented method is explained.

4.4.1 Retrieval in myCBR

myCBR provides sequential retrieval. Sequential retrieval computes the similarity of the query and all cases in the case base to get the most similar cases. This involves first calculating the local similarities. That is the similarity between corresponding attributes. Then, they are weighted and summed to generate the global similarity. That is the similarity between the query and the case. The retrieved cases are ordered by their similarity score.

In myCBR, attributes can be disabled so that they do not partake in the retrieval process. The attribute *Title* for example, can be disabled because it is not directly relevant when comparing the ingredients in recipes. However, attributes can not be disabled dependent on a query. The fact that *all* retrieval attributes take part in *every* retrieval process appeared to be a problem for our system.

Whenever an attribute does not contain any ingredients, the attribute value is set as `_unknown_`. When comparing two and two instances in the retrieval process, `_unknown_` attributes can match against each other as any other values. Match on `_unknown_` gets a preset value as local similarity score. If this value is set to one, it means that two recipes both containing no vegetable ingredients will match as equal on this attribute.

Hence, when calculating the similarity, these `_unknown_` matches will be as important as any ingredient match if the `_unknown_` value is set as one. This means that for queries with few ingredients, recipes containing `_unknown_` values on many attributes will have an advantage. This would have been the desired behavior if the user was aware that if s/he does not specify an ingredient for a specific attribute, it means that s/he does *not* want that type of ingredients. However, the user does not necessarily know which attributes the system contain or care about what attributes various ingredients belong to. The user should be able to simply type in the desired and undesired ingredients without having to think about the model structure.

If the `_unknown_` similarity is set to zero, on the other hand, recipes containing ingredients on many attributes will have an advantage because some match is better than zero. This is also considered an undesired behavior.

4.4.2 New retrieval method

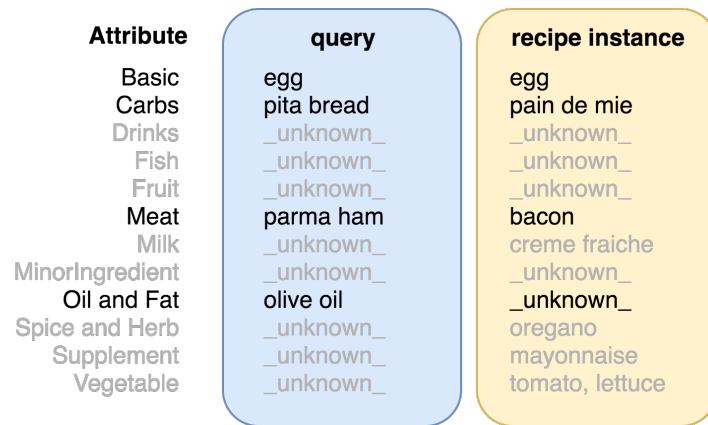


Figure 4.8: Comparing only attributes present in the query

The desired behavior for the system is to simply ignore attributes in the query instance that have the value `_unknown_`. For example, if the desired ingredients are *egg*, *pita bread*, *parma ham* and *olive oil*, the system should only consider their belonging attributes and ignore the rest. An example is illustrated in Figure 4.8.

To enable this behavior, a new retrieval method was created. As with retrieval in myCBR, one query instance is compared to all cases in a case base. The method iterates through every retrieval attribute in the query. Attributes with `_unknown_` values are ignored. The rest of the attributes are considered valid and takes part in the similarity calculation.

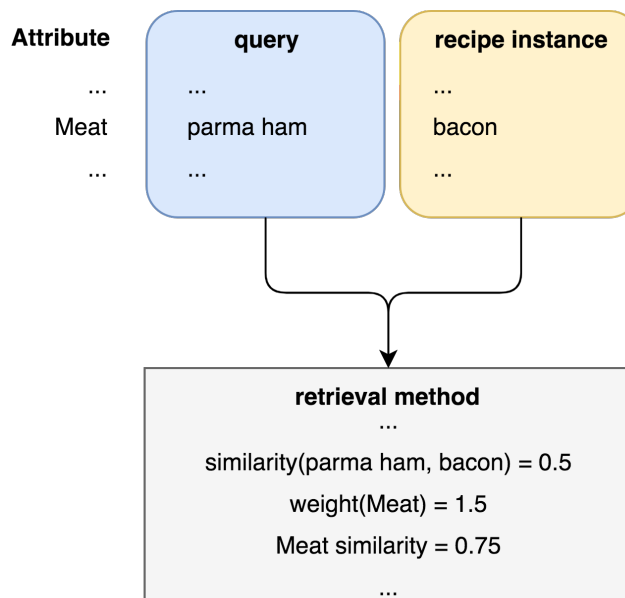


Figure 4.9: Calculating the local similarity between meat ingredients

Figure 4.9 shows how the system calculates local similarity for an attribute. The my-CBR project is used to provide similarities and attribute weights. For each valid attribute, the similarity between the query attribute and the corresponding instance attribute is fetched from myCBR. The similarity is weighted with the configured attribute weight.

When calculating the global similarity, the local similarities are summed up and divided by the number of valid attributes. To sum up, the system calculates the global similarity exactly as done in myCBR, except that the attributes that have the `_unknown_` value in the query instance are ignored.

4.5 Case base retrieval

The goal with the case base retrieval is to retrieve the cases from the case base that are to be considered in the reuse step. The system attempts retrieving the cases with the best *starting point* for ending up in successful recipe recommendations. The desired cases are cases with the best possible match on the desired query and the worst possible match on the undesired query.

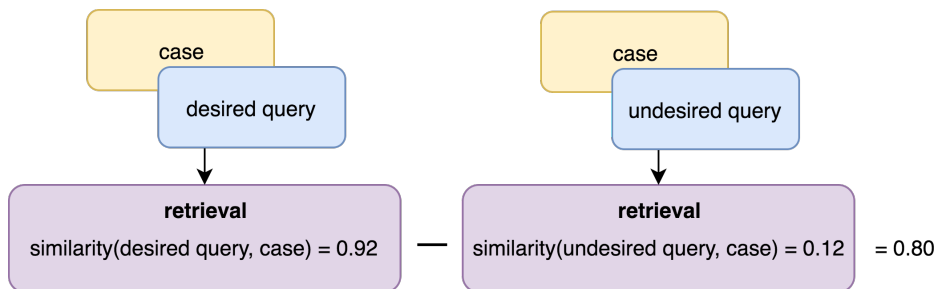


Figure 4.10: Case base retrieval example

Accordingly, the retrieval method is employed *twice*: One time comparing all cases to the desired query and one time comparing them to the undesired query. Consequently, each case receives one desired similarity score and one undesired similarity score. The total score is calculated by subtracting the undesired score from the desired score as illustrated in Figure 4.10. The result from the case base retrieval is twenty of the best matching cases, ordered by their similarity score.

4.6 Reuse

Copied versions of the retrieved original cases take part in the reuse step. The original versions are kept for later use in the cycle.

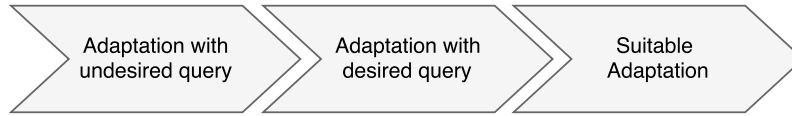


Figure 4.11: Adaptation process

As illustrated in Figure 4.11, the adaptation process involves three steps: Adaptation with undesired query, adaptation with undesired query, and suitable adaptation. By having only twenty-one recipes in the case base, chances are small that the original recipes will be good matches to an arbitrary query. Adaptation, on the other hand, enables the system to be much more flexible by customizing original recipes so that they better match the user query.

The different adaptation steps employ some shared methods to do modifications. Some of them are rule-based and some of them are based on the taxonomies provided by the myCBR project. The rule engine is explained in Section 4.6.1.

The first step in the process involves using the undesired query to modify the retrieved recipes so that they no longer contain any of the undesired ingredients. If this step is unsuccessful, the adapted recipe is rejected right away by being labeled invalid. As mentioned, undesired ingredients are very strictly considered. The undesired adaptation process is explained in Section 4.6.2.

The second step involves using the desired query to customize the retrieved recipes to increase their similarity score to the desired query. This is done by substituting desired ingredients into the recipe. The desired adaptation process is explained in Section 4.6.3.

Last, but not least, the recipe should make sense. If important ingredients are substituted in the previous steps, the system aims to further complete the change of style. For instance, if the main ingredient is substituted out of the recipe, the supplements should change to suit the new ingredients better. To set the record straight, this step is called *Suitable Adaptation*. The suitable adaptation process is explained in Section 4.6.4.

4.6.1 Rule engine

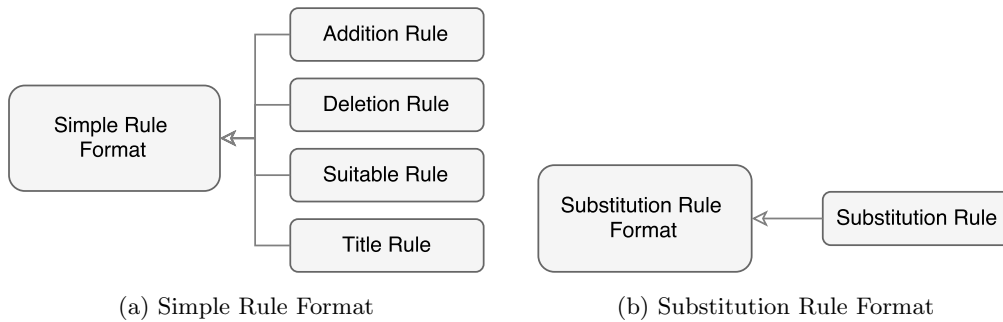


Figure 4.12: Rule formats

A rule engine was created for this specific system as a set of adaptation rules. An overview of the rules is illustrated in Figure 4.12. *Addition rules*, *deletion rules*, *suitable rules* and *title rules* are created with the *Simple Rule* format, while *substitution rules* has its own format called *Substitution Rule* format. In this section, the functionality of rules and the two formats are explained. Each specific rule is explained further in the reuse step, together with the methods where they are employed.

If one rule were to specifically target only the manually specified ingredients, numerous rules would have to be written. Therefore, the rule engine is also able to consider all children of the given ingredient. The children are fetched from the taxonomies in myCBR. This enables one single rule to apply to hundreds of ingredients. For example, consider a rule saying any type of *meat* can substitute for any type of *fish*. These ingredients have 191 and 74 ingredients, respectively. Hence, this one rule will form 14 134 various combinations. A rule containing *chicken* and *salmon* would be one of them.

The functionality to ignore children of a specific ingredient was also implemented. This is done by writing a * after the ingredient. With this functionality, a rule containing *meat** and *fish** will only form one combination.

Rule *requirements* refer to the ingredients in the recipe that have to be present for the rule to be valid. There can be zero or as many requirements as desired for a rule to fire. One requirement is satisfied if the recipe considered contains either the stated ingredient requirement or one of its children. A rule is valid when all requirements are satisfied.

Substitution rule

$$\text{requirement, requirement, requirement; ingredient1}^* \rightarrow \text{ingredient2 requirement, requirement; ingredient1} \leftrightarrow \text{ingredient2}$$

Two examples of the *Substitution Rule* format is illustrated above. A simple right arrow \rightarrow means that *ingredient1* can be substituted with *ingredient2*, and a double arrow \leftrightarrow

means that the substitution is valid in both directions. However, only when the requirements are satisfied. The ingredients may belong to different attributes.

Simple rule

requirement, requirement, requirement \rightarrow *ingredient*

The next rule format is called *Simple Rule*. The requirements work the same way. Exactly what happens with the specified ingredient depends on the situation in which the rule is used.

4.6.2 Adaptation with undesired query

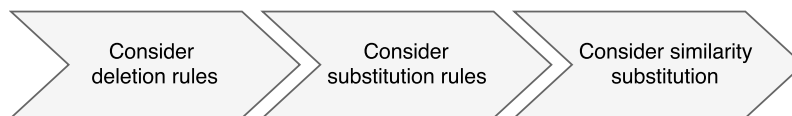


Figure 4.13: Undesired adaptation process overview

The adaptation process of a recipe starts with considering the undesired query, and the steps of the process are illustrated in Figure 4.13. The goal is to eliminate any undesired ingredient in the recipe. As explained, all children of all undesired ingredients are added to the undesired query. The system iterates all undesired ingredient and checks whether the ingredient is in the recipe and if so, action is taken. This subsection explains the various substitution methods used to get rid of undesired ingredients. If none of the substitution methods were successful for the recipe, adaptation with the undesired query is considered unsuccessful, and the adapted instance is labeled invalid.

Consider simply deletion rules

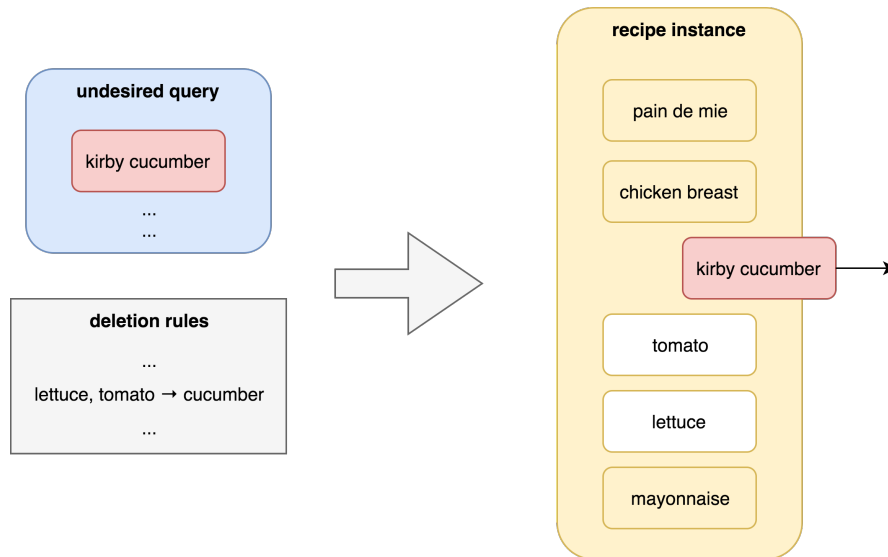


Figure 4.14: Firing the deletion rule $lettuce, tomato \rightarrow cucumber$

First, the system considers if the undesired ingredient can simply be deleted from the recipe. The system iterates a set of deletion rules to see if any of the rules apply to the currently considered undesired ingredient. The deletion rules are of the type *Simple Rule*. Hence, for the ingredient to be deleted there might be requirements that other ingredients must be present in the recipe. An example is illustrated in Figure 4.14. In the example, any type of cucumber can be deleted from the recipe if the recipe also contains lettuce (or a child of lettuce) and tomato (or a child of tomato).

Consider substitution rules

If no deletion rules are valid, the system considers if the ingredient can be substituted out in favor for another ingredient. Here, the *Substitution Rule* type is used. When looping through the rules, more than one substitute alternative may be found. All rules are considered equally good. If this is the case, a random substitute ingredient is chosen among the alternatives. Next, the undesired ingredient is deleted from the recipe, and the chosen ingredient is added.

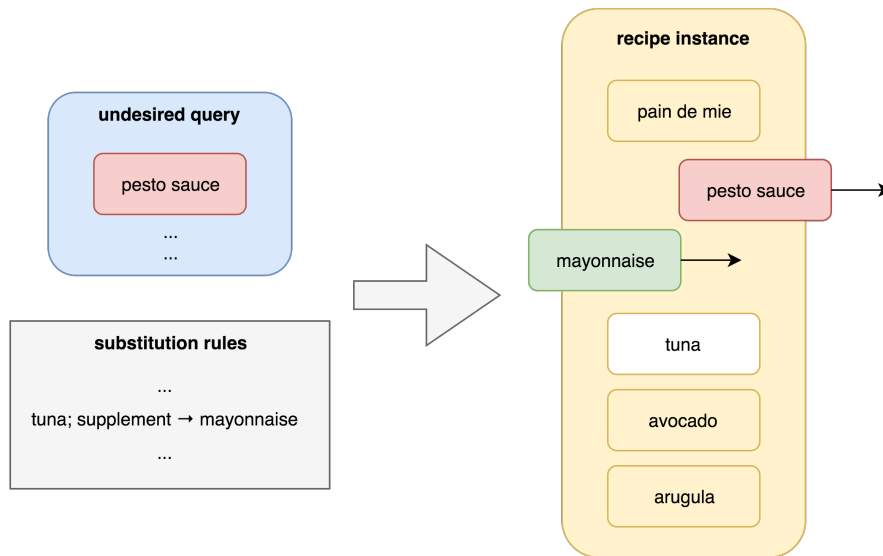


Figure 4.15: Firing the substitution rule *tuna; supplement* \rightarrow *mayonnaise*

An example is illustrated in Figure 4.15. If the recipe contains tuna, any type of undesired supplement can be substituted out in favor for mayonnaise. In this case, pesto sauce is the undesired ingredient. The system always makes sure that the alternative ingredient is not specified within the undesired query. The rule would not be applied if both pesto sauce and mayonnaise were undesired.

Consider similarity substitutions

If no substitution rules were successfully applied, similarity substitutions are considered. Similarity substitutions are based on the taxonomies provided by myCBR. With similarity substitutions, the system is only able to find a substitute ingredient within the same attribute because there are no similarities defined across taxonomies. The system aims to find the ingredient in the hierarchy that receives the highest similarity score when compared to the undesired ingredient. The system makes sure that no other undesired ingredient is found as an alternative.

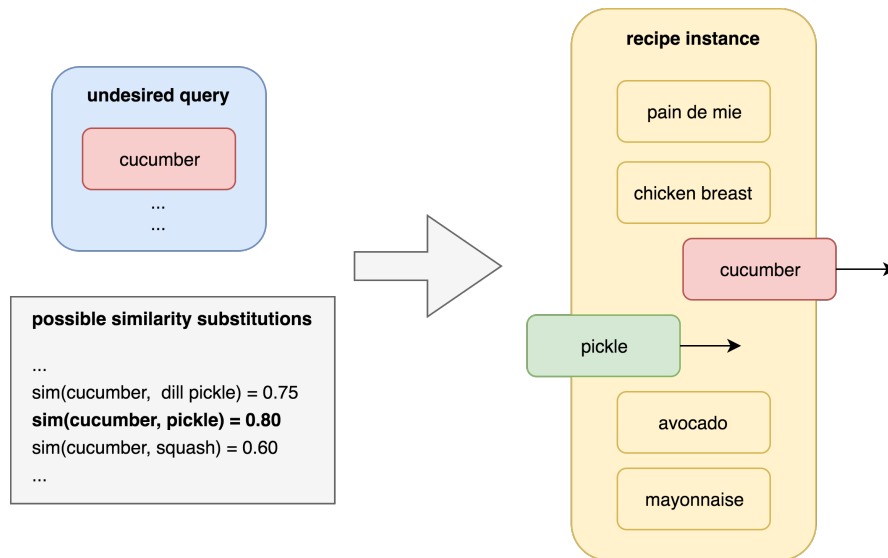


Figure 4.16: Finding a similar ingredient in the myCBR taxonomy

An example is illustrated in Figure 4.16. If several ingredients receive the best score, a random among them is chosen. If only very dissimilar ingredients are found, substitution is avoided. An adaptation threshold for each attribute is set to control this. If the best ingredient found satisfies the threshold, the substitution is carried out.

The system is configured to never substitute any ingredient with a parent. If this were not the case, the system would always end up trying to generalize. If the undesired ingredient were *cherry tomato*, the system would always suggest the parent, *tomato*, because parent ingredients always receive a score of 1.0 to their children.

4.6.3 Adaptation with desired query

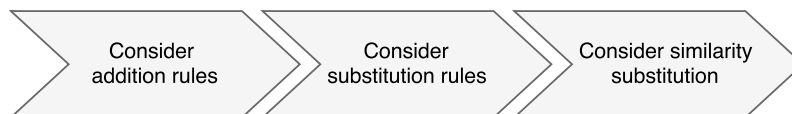


Figure 4.17: Desired adaptation process overview

The steps of adaptation with the desired query are illustrated by Figure 4.17. The process is very similar to the undesired query process. The difference is opposite goals: With the desired query, ingredients are substituted into the recipe instead of out. During the process, the system makes sure that no other desired ingredients are substituted out and that no undesired ingredients are substituted in.

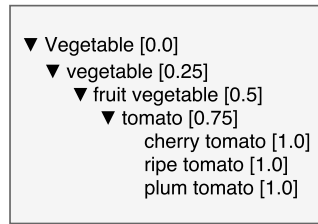


Figure 4.18: Excerpt from the Tomato taxonomy defined in myCBR

In this process, the system recognizes parents and children of the desired ingredient. Consider the tomato hierarchy in Figure 4.18. If the desired ingredient is cherry tomato and the recipe contains its parent, tomato, or its grandparent, fruit vegetables, the desired ingredient is considered satisfied. The same accounts for children. If the desired ingredient is tomato and the recipe contains cherry tomato, the desired ingredient is considered satisfied. Child-parent relationships are displayed to the user, so that s/he understands why the ingredient is satisfied. If the desired ingredient is satisfied, the system continues to the next desired ingredient. If not, it aims to find a way to add the ingredient.

Consider simply addition rules

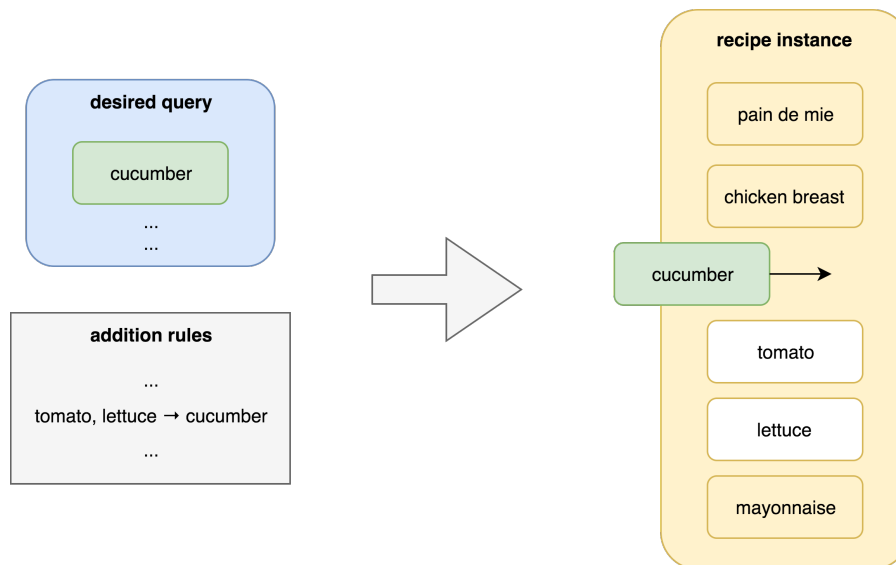
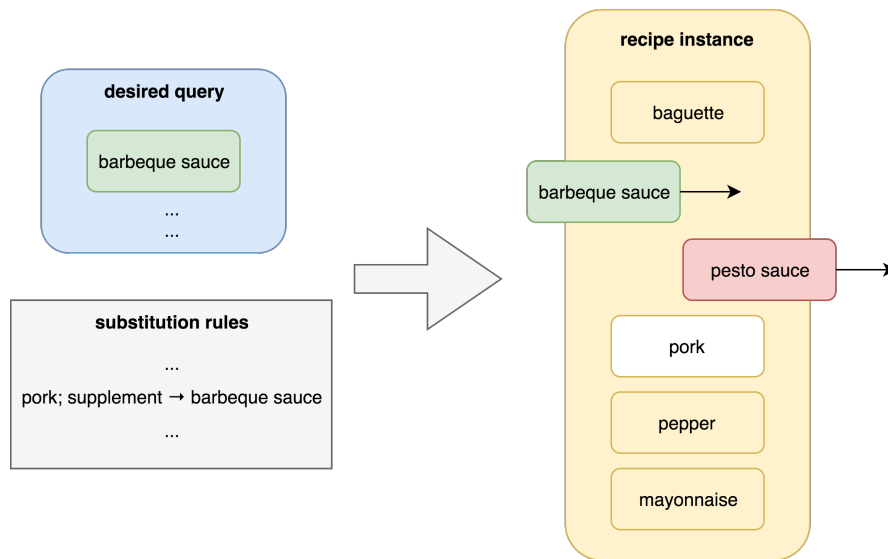


Figure 4.19: Firing the adding rule $tomato, lettuce^* \rightarrow cucumber^*$

First, the addition rules are considered. The addition rules are of the type *Simple Rule*. If the requirements are satisfied, the ingredient can be safely added to the recipe. Consider the example in Figure 4.19. Cucumber can be added if the recipe contains tomato (or a child of tomato) and lettuce.

Consider substitution rules

Figure 4.20: Firing the substitution rule $pork; supplement \rightarrow barbeque\ sauce$

If no addition rule was found, substitution rules are considered. The same substitution rules are used for the desired adaptation as for the undesired adaptation. However, this time a substitution alternative, or substitution *offer*, must be found inside the recipe. Consider the example in Figure 4.20. If the recipe contains some kind of pork and a supplement type, the supplement *barbeque sauce* is added to the recipe and the supplement *pesto sauce* is removed.

Consider similarity substitutions

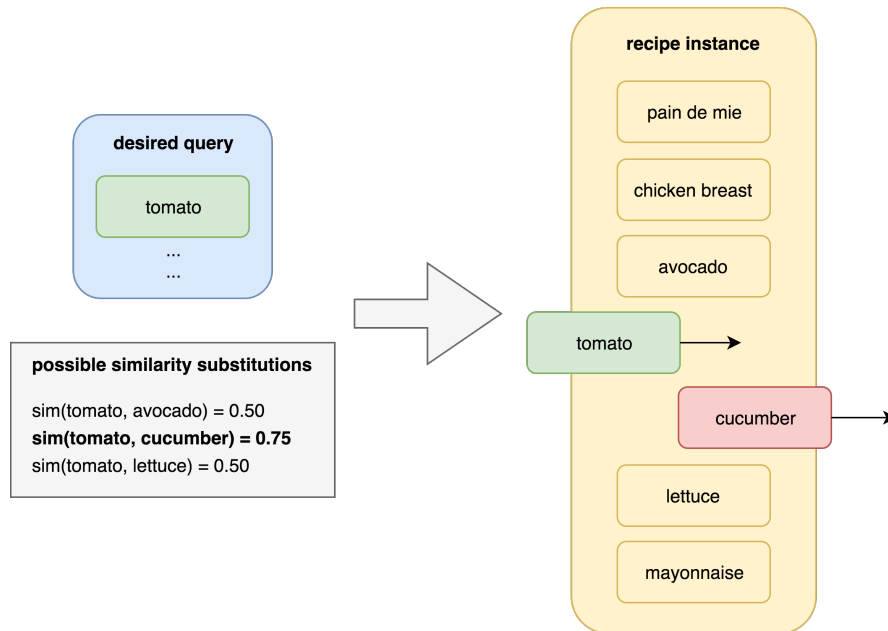


Figure 4.21: Finding a similar ingredient in the myCBR taxonomy

If no rules apply, similarity substitutions are considered. Similarity substitutions are less complex for the desired adaptation than the undesired because the substitute is found inside the recipe. Hence, the system only needs to calculate similarity between the desired ingredient and ingredients in the recipe that belong to the same attribute. Consider the example in Figure 4.21. Tomato is the desired ingredient. Therefore, cucumber, avocado, and lettuce, which also belongs to the *Vegetable* attribute are considered as substitution offers. The alternative with the highest similarity score is the chosen offer. As with the undesired adaptation, the system only goes through with the substitution if the adaptation threshold is satisfied.

4.6.4 Suitable adaptation

After the undesired and desired adaptation process, the style of the recipe may have changed. The idea with suitable adaptation is to make the new ingredients fit the recipe better.



Figure 4.22: Suitable adaptation process overview

The steps is illustrated in Figure 4.22. The process starts by iterating all modifications of the recipe instance. For every new ingredient, the system checks whether a so-called *suitable rule* applies. Suitable rules are of the type *Simple Rule*, which means that there may be requirements for the rule to be valid.

Consider the rule *pork, pita bread* \rightarrow *barbeque sauce*. If all required ingredients are in the recipe and at least one of the requirements is a new ingredient added to the recipe, the rule is valid. That means if *pork* has been added to a recipe which also contains *pita bread*, the system aims to add *barbeque sauce* to the recipe. In this case, *barbeque sauce* is the *suitable* ingredient. However, the suitable ingredients are not necessarily simply added to the recipe. The system aims to substitute *barbeque sauce* into the recipe the same way any desired ingredients are added. Hence, addition rules, substitution rules, and similarity substitutions are considered in the process. If several suitable ingredients are found, a random one among them is chosen.

Title adaptation

When ingredients are replaced by new ingredients in a recipe, the recipe title may be out of context. Two types of adaptation methods were implemented to rename the title to better fit the modified recipe.

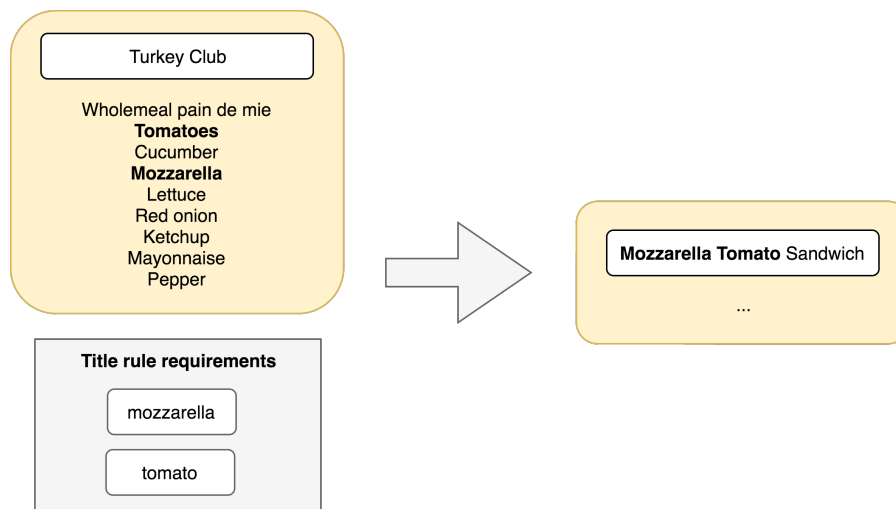


Figure 4.23: Generating title based on recipe ingredients. The system recognizes that there is both *mozzarella* and *tomato* in a modified recipe previously called *Turkey Club*. Consequently, the recipe is renamed to *Mozzarella Tomato Sandwich*.

First, the system considers pre-defined rules. For the record, the rules are called *title rules* and are of the type *Simple Rule*. An example is illustrated in Figure 4.23. In the example, an adapted recipe contains tomato and mozzarella, and the title is consequently renamed to *Mozzarella Tomato Sandwich*.

If no specific rules on ingredients apply, a different approach is considered. The method focuses on the previous title, and apply to titles containing ingredients. The system aims to match the content of the title with the substitutions that are done. If substituted ingredients are found in the title, the same substitutions are carried out in the title.

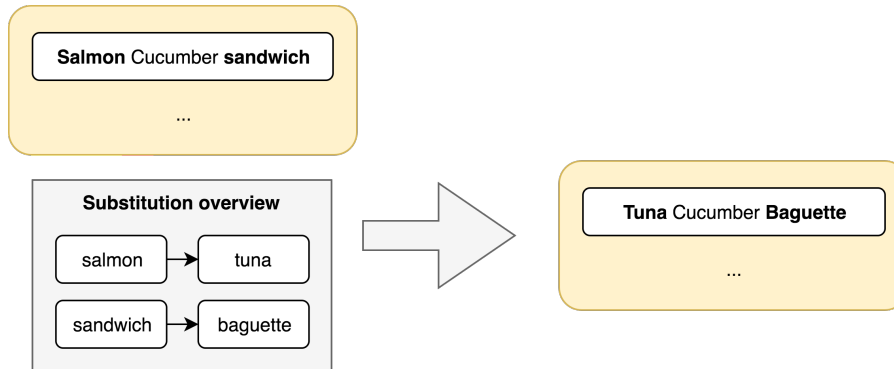


Figure 4.24: Generating title based on previous title. The system recognize recipe ingredients in the title that are part of substitutions. The same substitutions are done in the title.

An example is illustrated in Figure 4.24. The system recognizes both salmon and sandwich as ingredients that took part of substitutions. Consequently, the same substitutions are done to the title. The resulting title is *Tuna Cucumber Baguette*.

4.7 Constructing an ephemeral case base



Figure 4.25: Process overview

This section explains the strategies used to set up an ephemeral case base containing all cases that can be safely presented to the user. The steps are illustrated in Figure 4.25. The information used to build an *ephemeral* case base is gathered during the current cycle only (Mizzaro and Tasso 2002). This involves the user query, cases from the original case base and the newly adapted instances. The ephemeral case base will be further used in the retrieve step, where the best selection of the cases will be chosen to be used as recommendations. At the end of the cycle, the ephemeral case base is lost. None of the information is stored persistently for later use.

4.7.1 Filter original cases

To consider the original cases as valid recommendations, none of them should contain any undesired ingredients. By using the undesired query, the system discards all these invalid recipes. This leaves us with a selection of the original cases. Note that the cases will still be in the case base, but they will not be added to the ephemeral case base that is about to be created.

4.7.2 Filter adapted instances

The reuse step results in three categories of instances: Zero-modification instances, invalid instances, and valid instances. Zero-modification instances are instances where no modifications were made during the reuse step. These instances will still contain the same ingredients as its corresponding original case and are therefore discarded. Invalid instances are instances that still contain some undesired ingredients after the reuse step. Consequently, these are also discarded. Valid instances, however, are instances that satisfy the undesired query and are different from their corresponding original case. These are sent further to a new filtering process.

By discarding the zero-modifications instances, it is ensured that no adapted instance is equal to its corresponding original case. However, a more comprehensive check is necessary. The cases must differ from *all* existing cases. As the user can add adapted instances to the system's case base, the size of the case base will increase over time. The user query decides how cases are modified and consequently what the cases that might be added to the case base look like. The similarity between cases in the case base and the direction of their evolution will be out of control. Chances increase that adaptation will lead to a new case being equal to an already existing case in the case base. Even two modified instances that come from different roots might end up with the same ingredients.

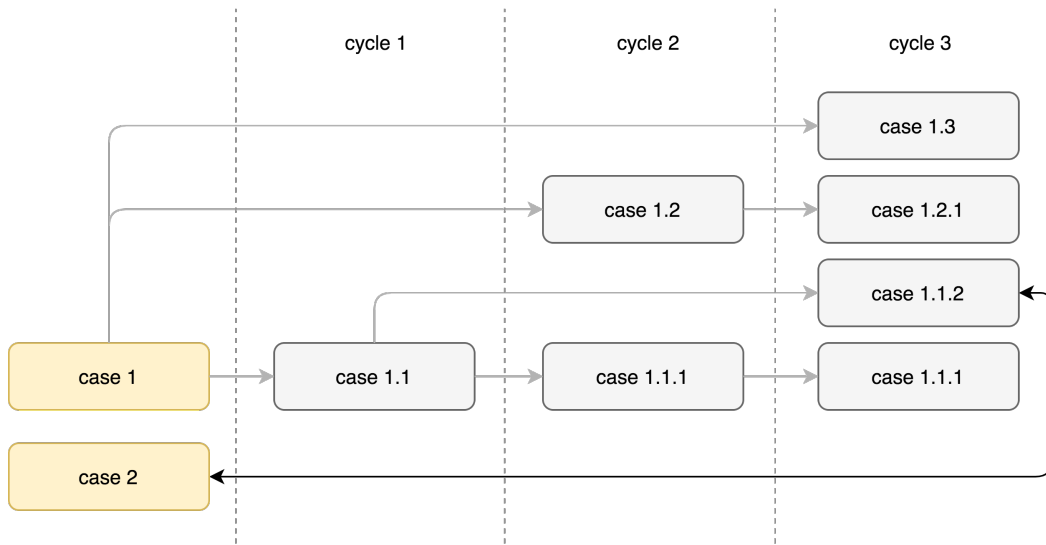


Figure 4.26: Third generation modified version of case 1 ends up having the exact same ingredients as case 2

Figure 4.26 illustrates how adaptation of cases can lead to a new case being equal to an already existing case. In the example, the adapted instance 1.1.2 is equal to the original case 2. Consequently, the system takes basis in valid instances resulting from the reuse step and discards cases that are equal to any other case in the case base. The result from this filtering process is a selection of unique and valid adapted instances.

4.7.3 Creating an ephemeral case base

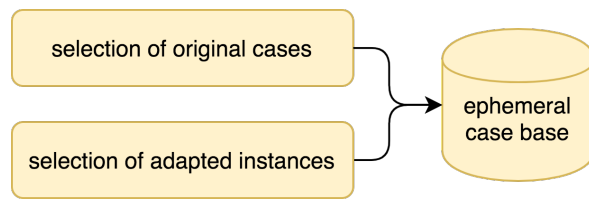


Figure 4.27: Creating an ephemeral case base

As Figure 4.27 shows, the remaining cases from the filtering of both the original cases and the adapted instances are added to an ephemeral case base.

4.8 Ephemeral case base retrieval

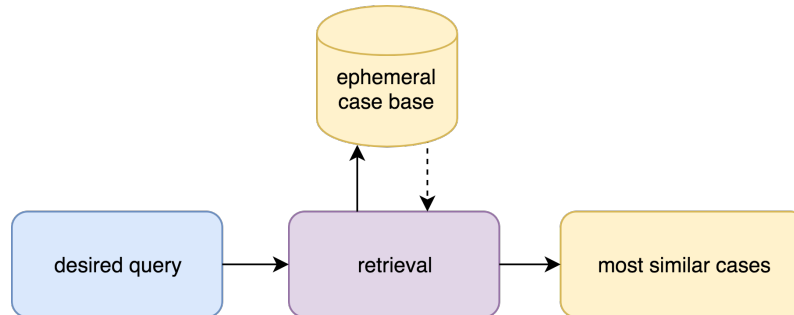


Figure 4.28: Ephemeral case base retrieval

As Figure 4.28 illustrates, the retrieval method explained in Section 4.4.2 is called with the desired query and the newly created ephemeral case base. All cases in the ephemeral case base are compared to the desired query in the retrieval process. Based on the comparison, they receive a similarity score. The undesired query is not considered in this step because all instances containing undesired ingredients have already been discarded.

Punishment of adapted instances

The ephemeral case base retrieval results in a mixture of original and adapted cases, ordered by their similarity score. Original cases refer to cases from the main case base, while adapted cases are created in the current cycle based on the user query. Accordingly, original cases are assumed to be safer suggestions. They are either one of the twenty-one initial recipes created by humans or recipes that are confirmed tasty by a user. Therefore, if an original case and an adapted instance receive the same similarity score, the original recipe should be recommended in favor for the adapted one. In general; The more a case has been modified, the more unsafe is it as a recommendation.

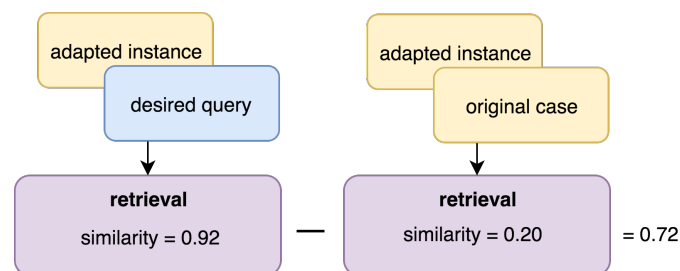


Figure 4.29: Decreasing the similarity score of an adapted recipe. The punishment score is calculated by comparing the adapted instance to its origin case (second retrieval in figure). Thereafter, the punishment score is subtracted from the similarity score that the adapted instance originally received (first retrieval in figure).

Figure 4.29 illustrates how the system punishes the similarity score of adapted instances. The punishment is based on the modifications made. That means, the more radical modifications made, the more the adapted instance is punished. The retrieval method is used to compare the adapted instance to its corresponding origin case. The resulting similarity score for the adapted instance is its similarity score minus the difference from its origin.

However, the suitable adaptations should not affect the similarity score and should not be punished. As explained in the Section 4.6, suitable adaptations are done to fix recipes where the style has changed during adaptation. Based on this, the substitutions resulting from the suitable adaptation are not considered in the ephemeral case base retrieval, and hence, not punished.

4.9 Revise and retain

To fulfill the retain step of the CBR cycle, the case base must evolve and grow over time. This calls for satisfying cases from the reuse step to be added to the case base. However, the system itself is not able to measure the level of excellence for a case. The level of similarity towards a query does not say anything about the tastefulness of a recipe. Hence, the newly adapted cases are only added to the case base if the user that did the initial query approves of it. Any cases approved by the user is considered complete and will be used for future retrieval and adaptation processes.

Chapter 5: Implementation

This chapter describes the implementation and architecture of IntelliMeal. An overview of the system is presented in Section 5.1. Further, Section 5.2 describes necessary preprocessing of the original recipe file provided by the CCC. Lastly, Section 5.3 describes the running application.

5.1 Overview

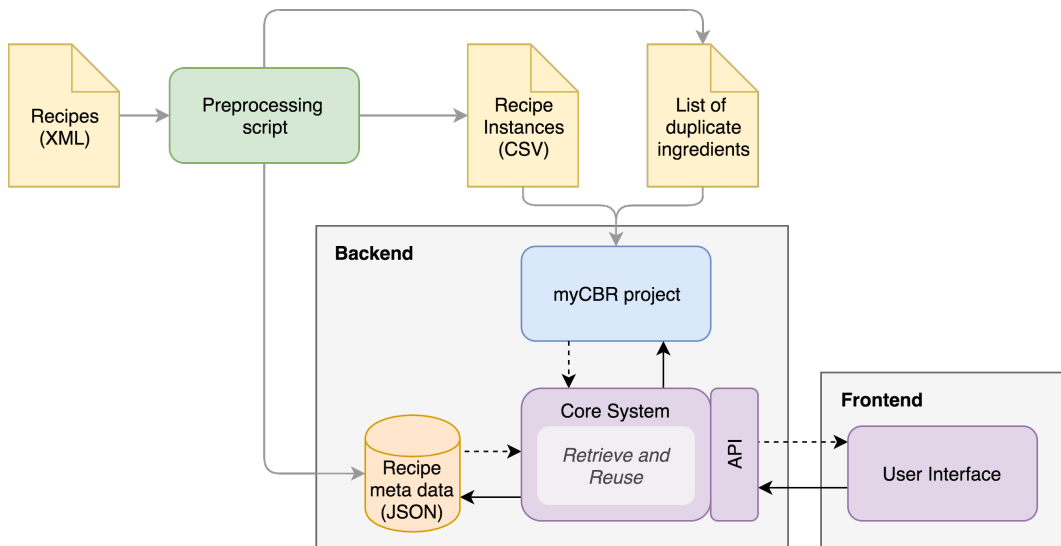


Figure 5.1: System overview

In Figure 5.1, an overview of the implemented components of the system is illustrated. All components are somehow coupled. Before setting up the system, the provided XML recipe file went through necessary preprocessing. Recipe instances were manually imported to the myCBR project, and any duplicate ingredients were removed.

The backend consists of the myCBR project file, the database of recipe metadata, and the core application itself. The core application is responsible for retrieval and adaptation

processes described in Chapter 4. The backend also provides a REST API for the frontend and UI.

5.2 Preprocessing

The preprocessing phase of this project mainly involved modeling of taxonomies and instances in myCBR. The modeling phase is explained in Section 4.1. To summarize, a myCBR project consists of a concept, and within the concept, attributes and possible values are defined. Each attribute and its belonging values are modeled as a taxonomy. In this project, the attributes are ingredient categorizations (e.g. vegetable), and the values are ingredients that belong to that classification (e.g. tomato). The attributes and values are used to define recipe instances, which in this project defines the id, title, and ingredients of a recipe.

The recipes used in this project were provided in a XML file. For the recipes to be useful, a preprocessing script was necessary. The script takes in the XML file and outputs three files: 1) a CSV file containing all instances to be imported to myCBR, 2) a JSON recipe object tree to store recipe metadata, and 3) a list of duplicate values within the taxonomies.

While the JSON tree is included in and needed for the running application, the other files were manually added to the myCBR project once, and then scrapped.

5.2.1 Generating CSV instance file

Recipe instances can be imported to myCBR from CSV files. The CSV file must specify ingredients below their corresponding attribute in the myCBR concept, which was not present in the XML file.

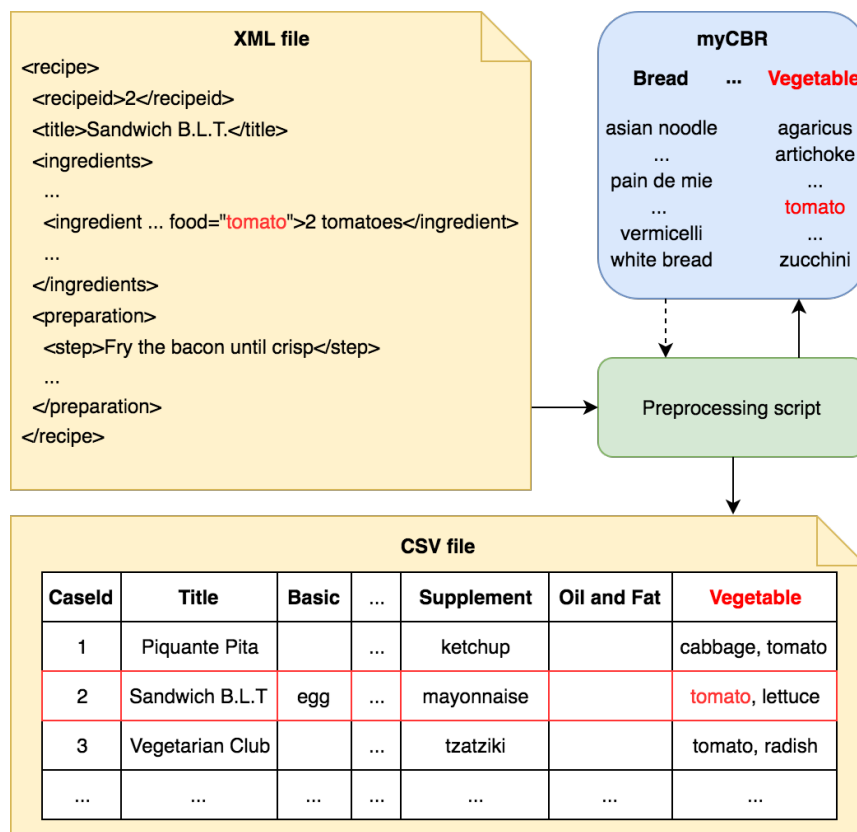


Figure 5.2: Generating CSV file

The preprocessing script aim to find the corresponding myCBR attribute for each ingredient in a recipe. An example of the process is illustrated in Figure 5.2. In the example, the script found that *tomato* is a value of the *Vegetable* attribute and therefore write this ingredient below the *Vegetable* column in the CSV file.

If any recipe ingredient did not match a myCBR value, the missing ingredients were manually added to the taxonomy, and the script was re-run before importing the instances to myCBR.

5.2.2 Generating JSON tree for recipe storage

During preprocessing a JSON tree was also generated. This tree serves as a NoSQL database for recipe representation usage. In the JSON tree each node represent a recipe, with the case instance id as the key. This database, in addition to the case instances in myCBR, was created for two reasons. Firstly, the preparation steps are needed for frontend representation of recipes, and the myCBR instances do not include these. Secondly, ingredients are written differently for the user than in the taxonomy.

<ingredient food="tomato" ..>2 tomatoes</ingredient>

In the XML file, each ingredient element contain a tag defining the ingredient name that corresponds to a value within myCBR in addition to the original ingredient text. For example, myCBR can read "tomato," while "2 tomatoes" is displayed for the user.

In addition to the information from the XML file, a boolean field was added to each JSON recipe object. The field is set to true if the recipe is adapted.

5.2.3 Generating list of duplicate values

During initial testing, duplicate values within the taxonomies were discovered. Examples are *low-fat yogurt*, *reduced fat yogurt*, and *light yogurt*. Because of this, a script detecting duplicates, plural versus singular form, and other synonyms were created. The output value list generated by the script was further used to remove duplicate values from the myCBR workbench manually. Keeping synonyms or misspellings will only make ingredient matching unnecessary complex, and removing them made the taxonomies more consistent.

5.3 Running application

As mentioned introductorily, the running application consists of two components; A backend and a frontend. The backend is the core system, and it is responsible for all CBR related functionality described in Chapter 4. The frontend implements a UI which lets users search for recipes by specifying desired and undesired ingredients.

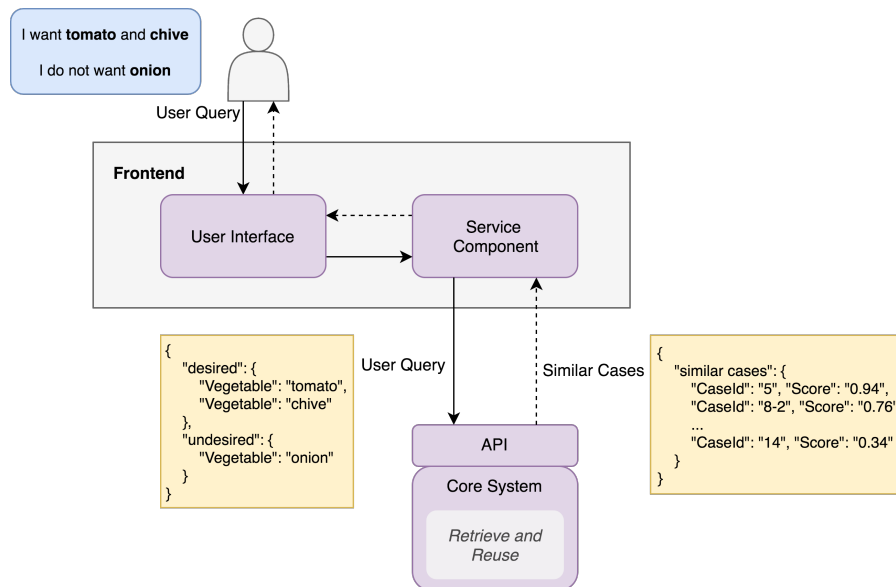


Figure 5.3: The user interface takes in a user query and makes a request to the API, which return a list with the most similar cases and their similarity score

Figure 5.3 shows how data flow from the user, via the UI and to the backend API. The core system then retrieves cases from the case base and if possible, adapt them to fit the user query. Lastly, a list of the most similar cases and their similarity score is returned. The five most similar cases are displayed to the user in the UI.

5.3.1 Core system

The backend is a Java application, which puts the system into a Spring container and provides REST services through a Swagger API. The REST API serve as the connecting link between the backend and the frontend.

The API define all necessary request methods for the frontend; Fetching all attributes and values, retrieving similar cases given a user query, adding new cases to the case base, and so forth.

The myCBR project file is included in the backend of the system. The project includes the case base of recipes and the attribute taxonomies. The project defines the weighting of each attribute, and the taxonomies specifies similarity between ingredients. The core system retrieves cases from the myCBR project, and uses the taxonomies to calculate the similarity between ingredients or cases.

As mentioned, the core system within the backend is responsible for retrieval and re-usage of cases. In this project, the CBR reuse step involves adaptation of recipes whenever this is possible. The implementation of these routines will not be further described in this chapter, as all processes are thoroughly explained in Chapter 4.

Test coverage

Functionality was tested continuously during implementation. The project includes a test environment, and JUnit tests cover all core functionality. Tests ensure that changes in prerequisites will not affect the behavior of existing methods (Do, Rothermel, and Kinneer 2006). Also, developers can be more self-confident when implementing new functionality as the tests will alert if the new feature damage performance of existing functionality. Below is an overview of the functionality that is tested by JUnit tests:

- Extension of undesired query
- Simple additions
- Substitutions for the desired query
- Similarity substitutions for the desired query
- Simple deletions
- Substitutions for undesired query
- Similarity substitutions for undesired query
- Considering suitable ingredients
- Retrieval and similarity scores

- Filtration of recipes containing undesired ingredients
- Removal of duplicate cases in the filtering process

5.3.2 User Interface

The frontend application of this project was built using Angular 2 with TypeScript. For this part of the system, the main focus was *usability*.

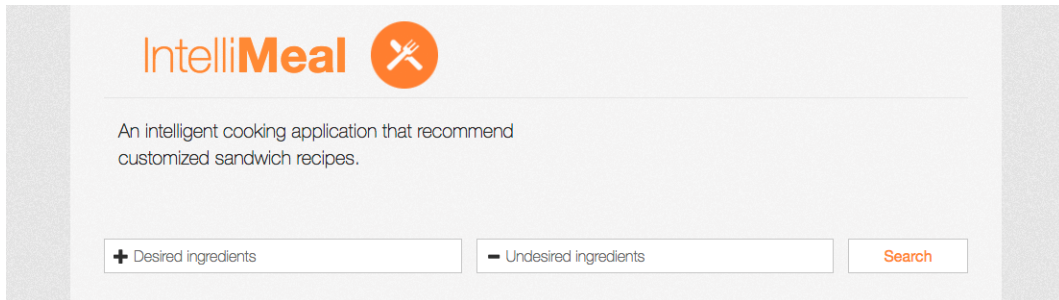


Figure 5.4: Input fields on front page

As shown in Figure 5.4, the UI prompts the user with two input fields; *desired* ingredients, and *undesired* ingredients. The user is forced to specify at least one desired ingredient before clicking the search button. This restriction is set because the system is not configured to calculate similarities to an empty query case.

+ re	
red apple	Fruit
red bell pepper	Vegetable
red cabbage	Vegetable
red currant	Fruit
red meat	Meat

Figure 5.5: Ingredient search completion

On initialization, the application retrieves all attribute values (ingredients) from the API. When the user starts typing, the system will continuously suggest matching ingredients, as shown in Figure 5.5 where the user has just typed "re". Suggesting ingredients is done to decrease the risk of the user misspelling ingredient names, which will decrease similarity. For example, a query including the word *asparages* will not match the ingredient *asparagus* from the knowledge model, and hence the similarity score will not be correctly calculated.

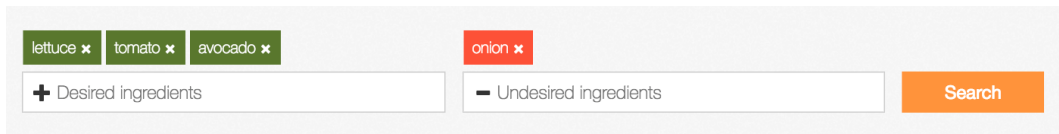


Figure 5.6: Specified desired and undesired ingredients

The user decides how many desired or undesired ingredients to specify. The specified ingredients are displayed for the user as shown in Figure 5.6. When the user has specified the ingredients and clicks the search button, the system translates this information into a JSON formatted query and further initiates a request to the backend API. The request returns a list of the most similar cases and their similarity score for the given query. The UI then displays the five cases with the highest similarity score for the user in decreasing order.

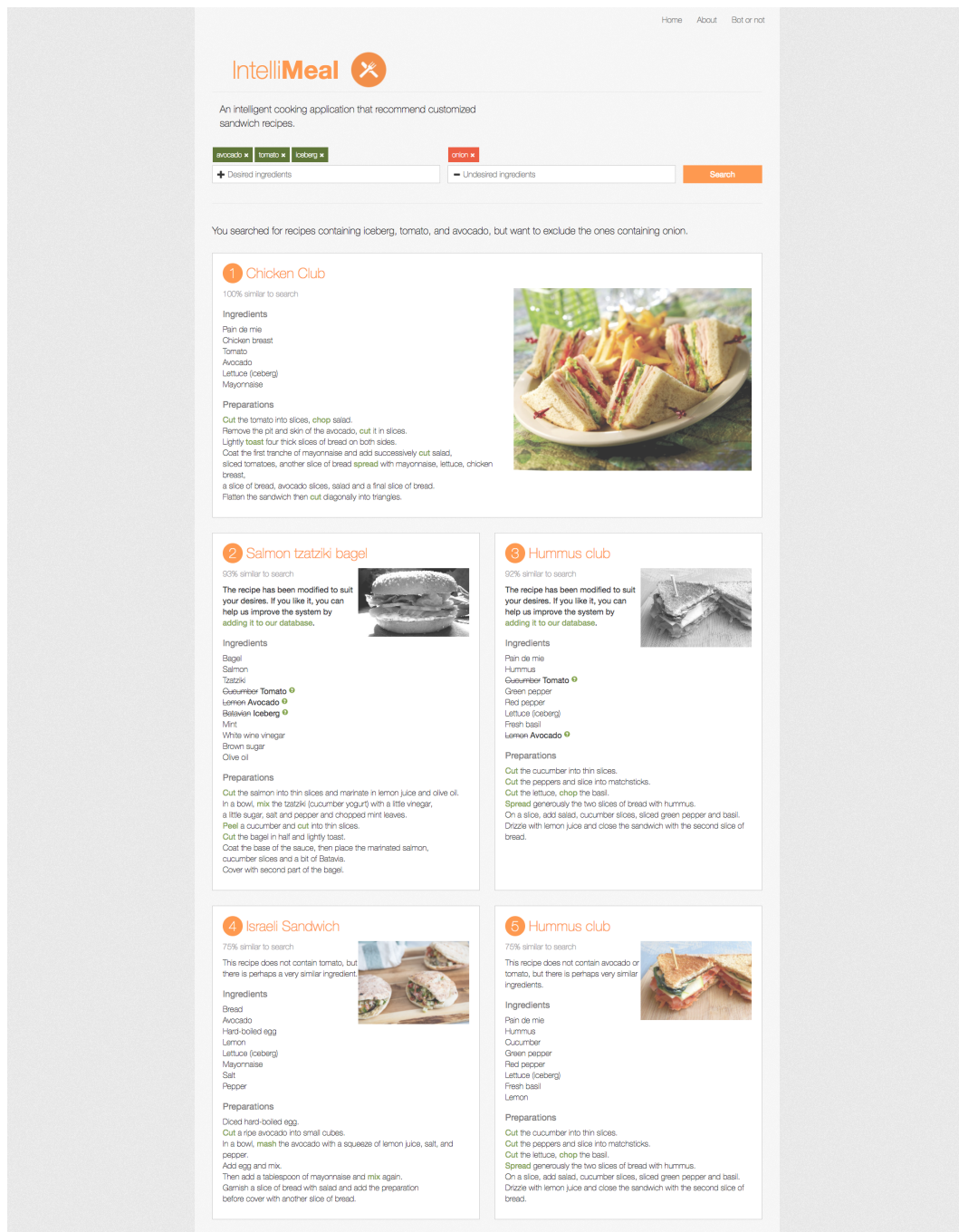


Figure 5.7: Website after retrieval of similar recipes

Figure 5.7 shows the full website after the request has returned. Any recipe with a grayed out photography represent a case that is adapted to suit the query.

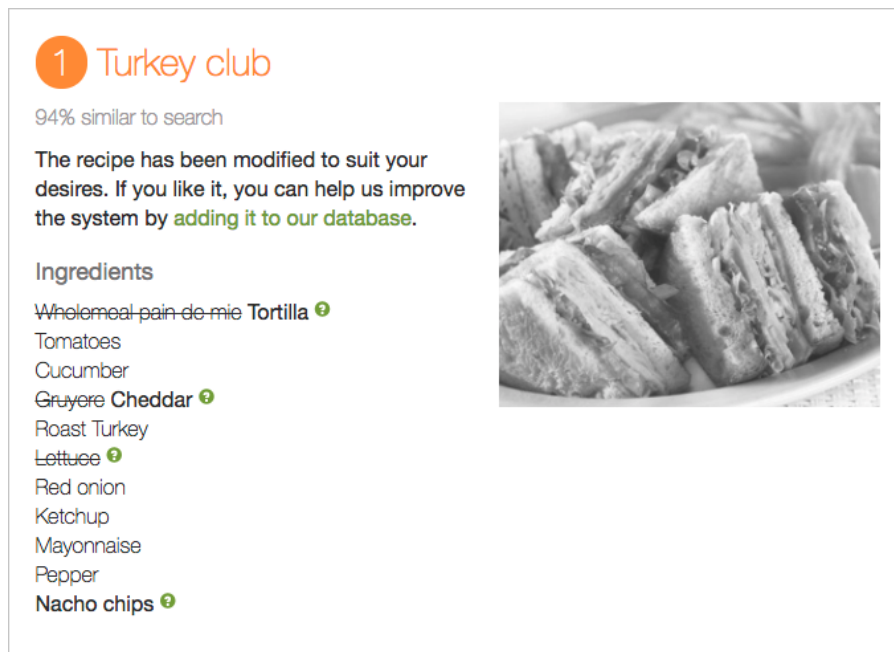


Figure 5.8: Example of adapted case

Figure 5.8 shows an example of how the UI present an adapted recipe. In this particular case, the adaptation involves both removal, addition, and substitution of ingredients. Removed ingredients (both substituted and simply removed) are styled with strike-through, while added ingredients (both substitutions and simply added) are displayed in bold. Also, a short text informing the user that the recipe is modified is displayed below the recipe title.

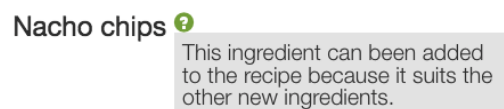


Figure 5.9: Infotip explaining the adaptation

All adapted ingredients have a question mark with an accompanying infotip next to them, as Figure 5.9 shows. Modified ingredients can look confusing for an inexperienced user; Giving an explanation of the reasoning processes employed to solve the problem is considered good practice within recommender systems (Ye and Johnson 1995; Herlocker, Konstan, and Riedl 2000). The displayed text is standardized for additions, removals, and substitutions.

1 Salmon cucumber sandwich

82% similar to search

This recipe does not contain tzatziki, but there is perhaps a very similar ingredient.

Ingredients

- Wholemeal bread
- Smoked salmon (fish)
- Cream cheese
- Chive
- Cucumber
- Onion (red onion)



Figure 5.10: Example of a recommended recipe lacking desired ingredients

If a recommended recipe is missing any of the desired ingredients, the UI will display a message below the title stating which ingredients is missing, as illustrated in Figure 5.10. This information is extracted by subtracting the intersection of desired ingredients and recipe ingredients from the desired ingredients.

Consider that a user specifies a parent or child ingredient as desired, while the recipe contains its belonging child or parent. The system will never substitute a child with its parent or a parent with its child, but their similarity is always 1. Consequently, the system aims to expose the generalization or specification for the given ingredient to the user, as displayed in Figure 5.10. Note that the recipe is *not* adapted. In the example, the desired ingredient *red onion* is generalized to the recipe ingredient *onion*, while the desired ingredient *fish* is specialized to the recipe ingredient *smoked salmon*. The necessary ingredient information to display these relationships is transmitted along with the case details from the API. As with explaining the ingredient adaptations described earlier, this is done to clarify for the user that the ingredient in question is considered equal to its child or parent in the recipe. Displaying this relationship can increase the user's trust and confidence in the system (Ye and Johnson 1995; Herlocker, Konstan, and Riedl 2000).

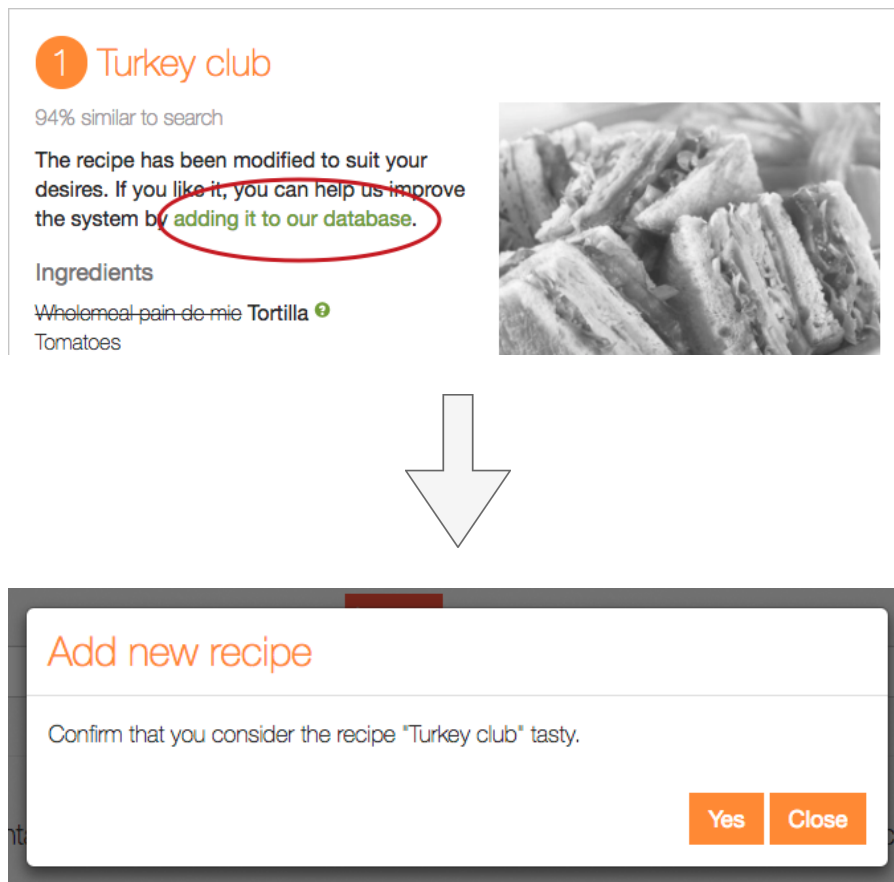


Figure 5.11: Adding a case to the case base

In the UI, the user gets the opportunity to add modified recipes to the case base. Figure 5.11 shows the link in which the user has to click to do this, as well as the modal that the user is prompted with if this takes place. If the user decides to add the case, the modal will confirm whether the process was successful, and the *yes* button will be deactivated to prevent adding duplicate cases to the case base.

Chapter 6: Evaluation and results

This chapter describes the process of evaluating the implemented system. The nature of the system demanded a varied evaluation approach. The assessment was not only concerned with improved behavior. It was also interesting to evaluate aspects such as usability and that users understand the recipe representation and layout. Among other things, it was desired to validate that the system is something that users find necessary and helpful in an everyday setting such as cooking, to further validate that cooking is indeed a proper domain to showcase CBR.

Five approaches for evaluation was used. Section 6.1 aims to show that the similarity score of a recipe increases when the recipe is adapted. Further, Section 6.2 seeks to evaluate the calculated similarity scores and hence the order of the presented recipes. Section 6.3 evaluates if the user can tell the difference between human and computer modified recipes. Lastly, Section 6.4 describes the process of usability testing.

6.1 Similarity score increase with adaptation

This section describes the process of evaluating whether the similarity score for recipe recommendations increases with the implemented adaptation process. Having a higher similarity score means that the recipe appears more useful for the given query. The end user will more often feel satisfied with the proposed recipes, as they in a higher percentage of the cases will contain more of the desired ingredients.

The process was done in three steps: 1) doing a set of queries with the adaptation process turned off and note the similarity scores achieved for the top five results, 2) doing the same set of queries with the adaptation process turned on, and 3) compare the similarity scores.

Query	Desired ingredients	Undesired ingredients
Q1	tuna	-
Q2	pain de mie, beans, avocado	hummus, mint, radish
Q3	baguette, pork, barbeque sauce	parma ham
Q4	egg, cheese, pepper, tomato, lettuce	parsley, chive
Q5	roast beef, cheese, pickled cucumber, tomato	-
Q6	avocado, bacon, mozzarella, pesto, tomato	-
Q7	tuna, onion, chick pea	-
Q8	peanut butter, banana, bacon	-
Q9	mozzarella, tomato	-
Q10	pita bread, lamb	egg

Table 6.1: Queries used for measuring similarity scores

In Table 6.1 the ten queries used to conduct this evaluation process is displayed. Note that the punishing of adapted cases described in Subsection 4.8 was turned off during the process. This choice was made to measure the exact similarity score each case receives without taking tastefulness into consideration.

Results

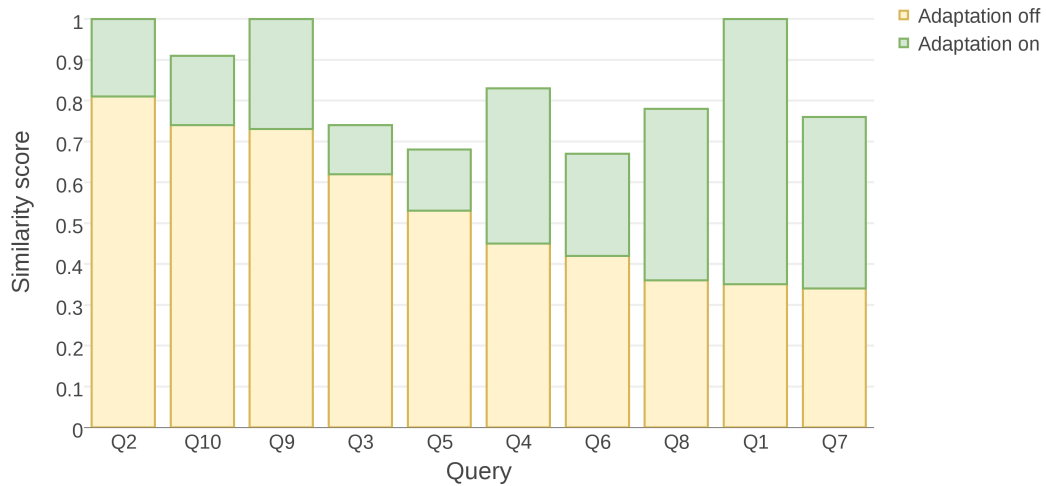


Figure 6.1: Average similarity score for the top five suggested cases with and without the adaptation process turned on

Figure 6.1 shows the results from the evaluation process. The yellow bars show the average similarity score for the top five suggested recipes with no adaptation, while the green bars show the average increase in similarity score for the top five suggested recipes when the adaptation process was turned on.

As indicated by the figure, the average similarity score for the top five recipes per query increased at all occurrences. The overall average amount of increased score for the top five recommended recipes was 0.32. Collected raw data from the process can be seen in Appendix Table A.1.

6.2 Ranking of recipes

To evaluate the system performance of ordering and calculating each recipe’s similarity score, tasks were constructed where test subjects were to order recipes based on given queries. The goal was to evaluate whether the system proposes the same recipes, in the same order, as a human would assume.

Task	Desired ingredients	Undesired ingredients
1	bread, fish, mayonnaise, avocado	egg
2	pita bread, chicken, tomato	
3	egg, oyster, dill	sausage

Table 6.2: Tasks for ranking of recipes

The three queries created can be seen in Table 6.2. For this evaluation approach, a test panel that had extended knowledge about food and ingredients was preferred. This choice was made because of the nature of the test; For a person to be able to estimate the similarity score of a recipe given a query, they would have to know a little about different foods.

The test subjects were given some information about the system. They were told that the system aims to match the user’s desires by calculating the similarity between the user’s desires and the ingredients in each recipe.

An evaluation system to measure the similarity between the test subject and system solution was created. The ranking system was created so that the score represents the *discrepancy* between the system’s and the user’s ranking for each recipe, ranging from 0 (same ranking from 1 to 5) to 25 (no recipes in common within the top 5). The evaluation system is displayed in Appendix Table A.2.

Results

The ranking of recipes process was conducted with seven randomly chosen test subjects. Giving the necessary information and conducting all three tasks took in average sixteen minutes per test subject.

Task	No of recipes in common per test subject							Average
1	4	2	4	3	3	1	4	3.00
2	4	4	4	4	4	4	4	4.00
3	4	5	3	3	3	5	3	3.71
Average	4.00	3.67	3.67	3.33	3.33	3.33	3.67	3.57

Table 6.3: Number of recipes in common for the system and test subjects for the top five query results.

Two metrics was calculated to represent the results from this process. Firstly, the number recipes in common for the top five results in the system versus the user was extracted. These metrics are represented by Table 6.3. The average number of common recipes per user can be seen in the bottom row, while the average of common recipes per query is shown in the rightmost column. As seen, the test subjects and the system had on average 3.57 recipes in common for the top 5 results.

Secondly, the offset between the system's and the test subject's recipe order was calculated using the evaluation system previously described. The system's proposal for ranking of the recipes for each query can be seen in the Appendix Table A.3. Appendix Table A.4 represent the raw data for the test subject's rankings, as well as the calculated discrepancy to the system's ranking. The calculated metrics show that the overall average offset score was 5.52 out of 25, with results ranging from 2.33 to 7.67 per user, and 2.57 to 7.57 per task.

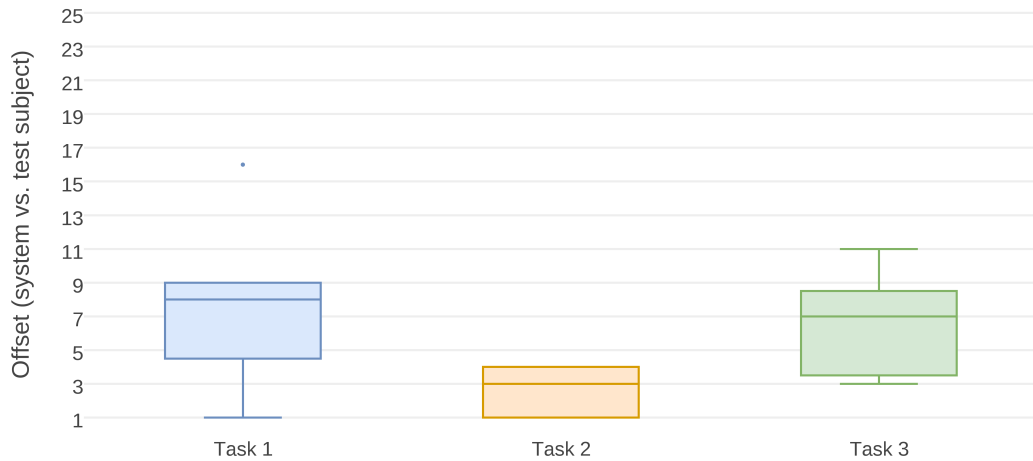


Figure 6.2: Box plot of ranking of recipes result. The boxes represent the inter-quartile range, the top line represent the upper quartile of the results, and the bottom line represent the lower quartile of the results.

Figure 6.2 illustrates the offset calculations from the three ranking tasks visually. The band inside the boxes represent the median offset per task, while the boxes themselves represent the inter-quartile range. That is, the middle 50% of the scores. Further, 75% of

the scores fall below any top line (i.e. the upper quartile), while 25% of scores fall above any bottom line (i.e. the lower quartile). The plot also reveals one outlier result for task 1, where one test subject had an offset of 16 compared to the system’s recipe ranking.

6.3 Quality of modified recipes

In principle, it is easy to modify recipes to match the user query. However, it is also of importance that the computer modified recipes are considered tasty and meaningful by humans. Taste is individual and hence hard to evaluate, but in this phase of evaluation, the goal was to assess if a human can detect whether a recipe is created/adapted by a computer. Inspired by the Turing Test introduced in Chapter 1, a quiz was set up to conduct this evaluation.

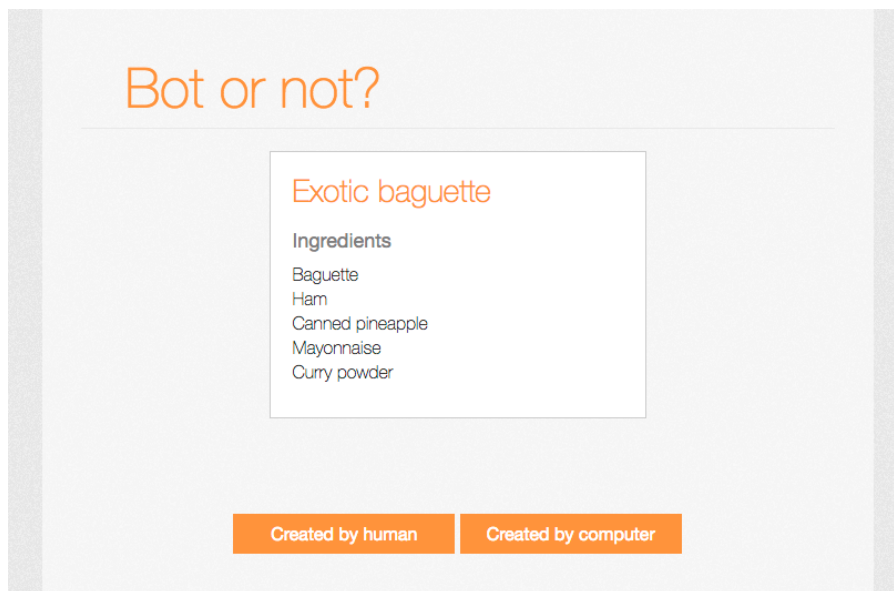


Figure 6.3: Screenshot from the quiz "Bot or not?"

A screenshot from the quiz can be seen in 6.3. As seen, the user is prompted with one recipe at a time and is to choose whether s/he think the recipe is created by a human or a computer. Using this method to gather evaluation data allowed for a wider distribution and faster response to the tasks. In general, the threshold for participating in an organized evaluation process is higher than doing an online quiz (Hohwü et al. 2013). Also, it is cost effective, scales well, and preserves user anonymity. To further lower the threshold, the quiz was created in such a way that the users could choose how many recipes they wanted to go through themselves.

To provide recipes for the quiz, a separate case base was initialized. All twenty-one original recipes were added to this case base, along with twenty-one adapted recipes. Appendix Table A.5 shows the queries made to conclude which recipes to add, along with the titles

for all added cases. The queries were made (and cases added) in the presented order, which led to adding recipes adapted for multiple generations. In the quiz, recipes are displayed to the user in random order.

The quiz itself was distributed through Facebook, Twitter, and other social media. It did not keep track on the number of users going through the quiz; it simply kept track of the number of correct and incorrect guesses per recipe.

Results

The quiz was kept running for seven days (168 hours) and gathered in total 3414 responses distributed over the 42 recipes. The raw results can be seen in Appendix Table A.6. The number of guesses per recipe varied from 64 to 93. The reason for this span is that the list of recipes to be presented was shuffled and displayed randomly to the user. The average of responses per recipe was 81.

		Predicted		
		FALSE	TRUE	
Actual	FALSE	TN 856 (50.09%)	FP 853 (49.91%)	1709
	TRUE	FN 911 (53.43%)	TP 794 (46.57%)	1705
		1767 (51.76%)	1647 (48.24%)	

Figure 6.4: Confusion matrix for quiz responses

Figure 6.4 shows the distribution of responses. To clarify, the value *true* refers to an adapted recipe. In total, the test subjects achieved an accuracy of 48.33%, which represent the number of correct guesses. In 53.43% of the cases, users guessed that a human created the computer created recipes. Besides, the original recipes were in 49.91% of the cases recognized as adapted. Together, these numbers indicate that the users struggled with separating computer and human created recipes from one another.

There is, of course, a possibility that the people that answered the quiz focused more on classifying a recipe as human created or computer created rather than food quality. However, with the assumption that people expect computer created recipes to be strangely composed the results are interpreted as valid.

6.4 Usability testing with questionnaire

To evaluate the usability of IntelliMeal as well as the general attitude towards and opinion of the system, a usability test with an accompanying questionnaire was organized.

The test was constructed with no prerequisite concerning age, cooking skills or cooking habits, as the system can be beneficial for all age groups and is targeted to neither novice chefs or amateurs. The test subjects were given no information about the system before the test, other than that it is a website for searching recipes.

A reply form concerning cooking skills and habits as well as general thoughts on already existing websites for recipes was created. This form was filled out before the usability testing and can be found in Appendix Figure A.2.

To perform the test and collect data, the participants needed tasks to solve. Three tasks were produced, and the tasks can be found in Appendix Figure A.1. All three tasks involved querying for different recipes. Task one was conducted with no adaptation. The goal was to evaluate whether users find the result more pleasing with or without the adaptation. Task three also involved adding a case to the case base.

Two people in addition to the test subject were present during the test; one to take notes and observe the subject's reactions and behavior, and one to read all tasks out loud and to take care of organizational tasks related to the process. During the task solving, test subjects were encouraged to speak their mind out loud, which makes it easier for observers to understand how they engage the task. The users were not provided any help or hints if they could not understand any functionality. However, since IntelliMeal is fully English while all test subjects were Norwegian, ingredients and metadata were translated when necessary. All environmental decisions were made based upon common usability testing guides (Dumas and Redish 1999).

After completion of the usability testing tasks, the test subjects was immediately asked to give written feedback. Two forms were set up to collect feedback. First is a System Usability Scale (SUS) form (Brooke 1996). SUS is a well known and thoroughly tested usability scale. The form used can be seen in Appendix Figure A.3. It involves ten question items, giving a global view of subjective assessments of usability. Besides, six post usability testing questions targeting online recipe recommender systems were prepared. These questions can be found in Appendix Figure A.4. The goal with these questions was mapping whether the user found such a system to be useful, whether the system could improve a person's knowledge about recipe composition, and whether such a system seems preferable over a system that does not have any knowledge about ingredient similarity (and hence do not adapt recipes).

Using questionnaires to gather information about the user's opinions allowed for easily comparing of results, because of the quantitative nature of some of the data produced. To produce measurable and comparable results, all questions in the three forms were formulated as statements with a five-stage scale with the end labels "strongly disagree" and

”strongly agree.” For simplicity and to avoid misapprehension, all tasks and questionnaires were translated into Norwegian.

Between each candidate, the case base was reset not to contain any adapted cases. The counter-argument to this is that the system under evaluation is indeed an evolving system. However, to be able to compare the user’s answers and collect quantitative data, the user’s search results had to be equal.

Test subjects

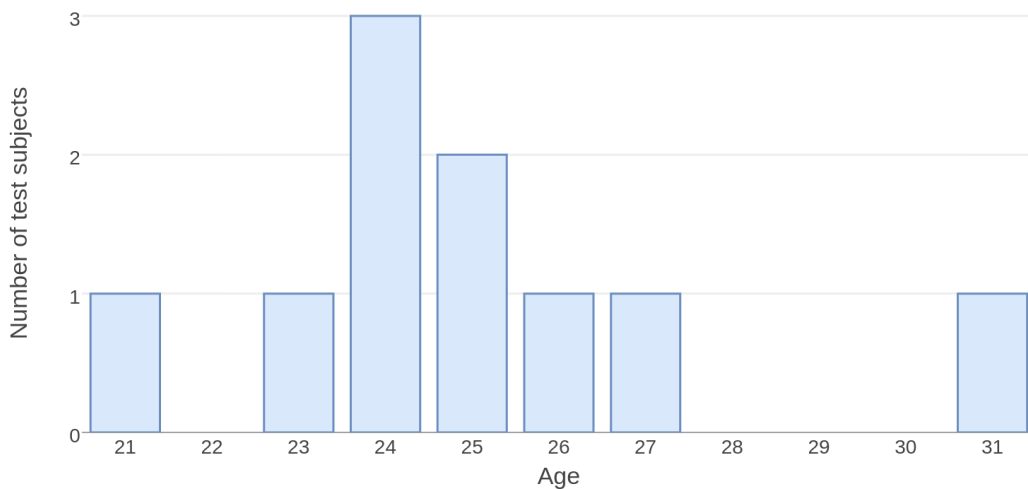


Figure 6.5: Distribution of test subjects’ age

The evaluation included ten volunteer test subjects, all non-familiar with the system. All test subjects were students. The age span of test subjects is illustrated in Figure 6.5. Ages ranged from 21 years to 31 years, with an average of 25. Four of the test subjects were women, while six were men.

In total with information, conducting tasks and filling out reply forms, the usability testing lasted for an average of twelve minutes per user.

Results

In general, the test subjects understood the tasks and had few problems understanding what to do and how to do it. All test subjects were able to complete all three tasks except one, which did not understand how to add the new recipe to the database.

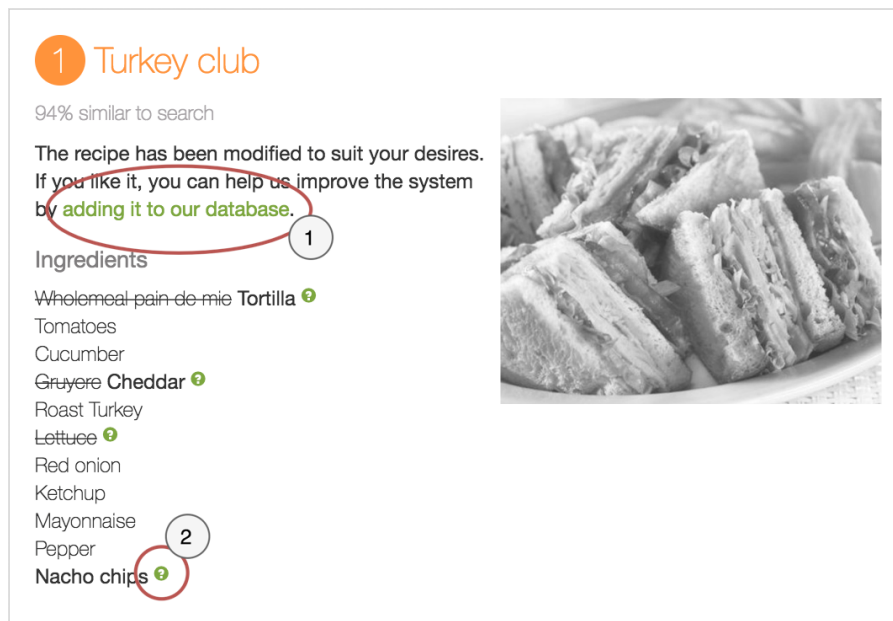


Figure 6.6: Issues detected during usability testing

One problem was repeating. Six out of ten people used a disquieting amount of time finding the *add to database*, illustrated as issue 1 in Figure 6.6. The test subjects were looking for a button, save symbol or heart next to the recipe. Out of these six, three people also commented a confusion on whether the case was added to their private page or the database of the system.

Other than this, one test subject could not understand why *Tortilla chips* was added during the execution of task 2. Five people understood by themselves that the new ingredient suits the other ingredients in the recipe, while the four remaining test subjects read the explanation by hovering the question mark next to the ingredient. One test subject suggested enlarging the question mark symbol, as they found it quite small and hard to spot. The problem is is illustrated as issue 2 in Figure 6.6.

As for positive feedback, six test subjects actively stated that the ingredient substitutions and additions made clear sense. For example, substituting *Pain de mie* with *Tortilla* seemed completely feasible. All test subjects easily identified which ingredients the adaptation embraced and whether it was added, removed, or substituted with another.

Appendix Table A.8 represent the SUS responses and results. The average SUS score from the usability testing is calculated to be 93.25. Research states that any SUS score above 68 is considered above average, while a score above 90 can be graded A (Bangor, Kortum, and Miller 2009).

Appendix Table A.7 shows the raw results from the pre usability testing reply form, while results from the reply form for post usability testing can be found in Appendix Table A.9. Finding patterns in the test subjects replies was challenging, as the usability testing only

CHAPTER 6. EVALUATION AND RESULTS

involved ten people. However, a few was uncovered. When asking the test subjects whether they feel the state-of-art recipe search engines can help improve their cooking abilities, the average response was 3.38 out of 5. When asked if they think a system such as IntelliMeal could do the same, the response averaged at 4.15 out of 5. Also, with an average of 4.40 out of 5, the test subjects stated that they would prefer using an adapting recipe search engine over a non-adapting recipe search engine.

As the primary focus of this project is CC and CBR, only one iteration of usability testing with a set of ten people was conducted. Although ten test subjects are the advised amount per iteration, the system could presumably benefit from numerous iterations of usability testing to reveal possible drawbacks of the UI.

Chapter 7: Discussion

The goal of this chapter is to target the research questions elicited in Section 1.2 by discussing our results and provide a summary of our findings. Section 7.1 discusses previous research within CBR and recipe recommendation, including their advantages and drawbacks. Section 7.2 consider CBR as an approach for recipe recommender systems, and how CBR can provide us with opportunities to being creative. Further, Section 7.3 considers the approaches that have been implemented to make the system creative, and Section 7.4 follows up with how the implemented functionality affect the system output. Section 7.5 presents the results from the usability testing and discusses the general impression of the system from a user’s perspective.

7.1 State of the art research

This section considers the following research questions

RQ1: What are features previously showcased in recipe recommender systems?

RQ2: What are approaches typically used in recipe recommender systems?

In Chapter 3, we targeted previous approaches to recipe recommender systems. The chapter discusses two types of applications: research systems which aim to adapt recipes, and four commercial systems using various approaches to recommend recipes to users.

During research on related work, we discovered that CBR is a rarely used method for commercial recipe recommender systems (as a disclaimer we want to specify that there may exist commercial applications utilizing CBR without having published a research paper or stating this on their website). The commercial systems base their recommendations on a particular user query or tailor them to a specific user given their personalized profile or previous feedback on the proposals presented to them. Further, the systems provide various features. AllRecipes is a community for recipe sharing across the world. The website provides functionality for searching both desired and undesired ingredients, in addition to regular text search. Yummly lets the user search by generalized categories like *gluten allergy*, *vegetarian dishes*, and so forth, with the goal to learn what each particular user tends to enjoy. Chef Watson is the most advanced of the commercial systems. The system does not

suggest already existing recipes, but *creates* them based on knowledge on co-occurrences of ingredients.

The CCC competitors described in Section 3.1, on the other hand, have all utilized CBR within their systems. Besides, all systems have incorporated some cooking ontology. The four systems all have a taxonomy of ingredients in hierarchical form. CookIIS is the only system that allows for specific similarity measures between ingredients. Taaable, CookingCAKE, and JaDaWeb calculate the similarity simply by measuring the nearness of ingredients in the hierarchy. Beyond this, three of the competitors have chosen different focus areas for the CCC open challenge. JaDaWeb focuses on natural language processing for the user’s query, while Taaable and CookingCAKE target preparation step adaptation.

7.2 CBR and its opportunities for creativity

This section considers the following research questions

RQ3: Is CBR a suitable approach for developing a recipe recommender system?

RQ4: How can CBR provide opportunities to being creative?

In Chapter 2, CBR was introduced and compared to other recommender system approaches. Cooking was found to be a safe domain to practice AI and CBR. Also, CBR does not require an in-depth knowledge of the field, like other recommender systems. While other recommender systems need accurate and detailed decision rules in advance, CBR can take advantage of already existing recipes and examine these when solving new problems. CBR is also time efficient as it reuses prior solutions, and hence, avoids repetition of earlier effort.

As explained in Subsection 2.2.1, CBR does not set many restrictions. The CBR cycle is a methodology on how to solve a new problem. The definition of similarities is domain-dependent, and the reuse step has no determined plan on *how* the customization should happen. Accordingly, CBR may be used to replicate old knowledge, but it can also be used to provide creative solutions by finding new ways of employing the available knowledge. Hence, CBR is an excellent basis for enhancing creativity.

7.3 Implemented adaptation process

This section considers the following research question

RQ5: If there exist opportunities, how can creativity be incorporated?

Creativity can be incorporated in a CBR system in multiple ways because CBR is such a general methodology. Typically, learning systems are either data driven or focus on preliminary knowledge engineering. A system called *Q-chef* combines CBR and deep learning to generate surprising recipe designs (Grace et al. 2016). The system acquires

knowledge by generalizing ingredient compositions, and further use these concepts to adapt recipes. Another example is CookIIS which aims for a community driven approach. The system expands its knowledge base by mining cooking communities and generalizes the acquired knowledge into rules.

IntelliMeal, on the other hand, is a knowledge engineering heavy system. Twenty-one recipes form the basis for the system, as well as knowledge about relationships within a limited set of ingredients. The latter involves taxonomies of ingredients modeled in myCBR and manually written rules.

The system learns by adding new cases to its case base. The expansion increases possibilities to match the query, but the system does not exploit the ingredient compositions within recipes further for adaptation. Taxonomies and rules are the model components used for adaptation. However, none of these are modified or expanded without manual intervention. Therefore, it is a necessity that this underlying data form a good starting point. As stated in Subsection 2.3.4, it is challenging to improve system performance if the system is initially poorly designed. The taxonomies need to be logical, clean and structured and all model components must agree on values to be able to work together. Taking basis in a clear starting point, we have further focused on finding creative ways to exploit the limited available knowledge.

The taxonomies in the myCBR model are taken advantage of by being employed in multiple parts of our system. While preparing for adaptation, they are used to generate rules and to expand the undesired query by fetching the children nodes of undesired ingredients. In the reuse step, they are used for general checks, making the UI self-explanatory, and in similarity substitutions. After that, they are used to filtrate undesired cases.

During research on related work in Section 3.1, four earlier CCC systems were presented. All four systems involve a hierarchical taxonomy. Some systems generate substitution rules from their taxonomy or cooking communities, while JaDaWeb has a table of ingredients that can substitute each other across categories. However, none explicitly define removal, addition or suitable rules like implemented in IntelliMeal. The rules and the similarity substitutions complement each other's weaknesses. Together, the components result in comprehensive adaptation of recipes.

The implemented rule engine is innovative for three reasons. First, the system employs a set of rules in its adaptation process: *Addition rules*, *deletion rules*, *substitution rules*, *suitable rules* and *title rules*. These are all created using two different formats, making the rules easy to understand. All rule types have their purpose, advancing the adaptation process.

Second, the rules employed in IntelliMeal have the ability to serve both specific and general purposes. Incorporating taxonomies into the rule engine has enabled this. The rule engine recognizes children of all ingredients involved in a rule, and if desired, considered at the same rate as the given ingredient. The opportunity to decide for how many ingredients a rule should apply makes our rules flexible. Besides, the rule engine is easier and much less time-consuming to create.

Third, rules may be specified for ingredients across attributes. The result is more radical modifications of recipes, where the general style of a recipe may change completely.

To further fulfill the change of style, an additional round of adaptation called suitable adaptation is employed. Before this adaptation, modifications are made purely based on the user query. The suitable adaptation considers these amendments to discover opportunities for improving the solution as a whole by customizing it even further towards the initiated style.

In the adaptation process, the system makes some random choices. The choices with the most considerable impact are when finding several substitution offers and when finding several suitable ingredients. In these cases, the system chooses a random among them. Consequently, equal user queries might end up with different resulting recipes.

Another creative solution in IntelliMeal is the retrieval method. The retrieval method was implemented when discovering that retrieval in myCBR resulted in undesired behavior when used for the purpose of our system. In myCBR, undefined attributes in the user query had an adverse influence on the total similarity score of a case. Consequently, the resulting order of recommendations was not as expected. The new functionality of our retrieval method is to calculate the total similarity score of a case only based on the attributes defined in the user query. Implementing this method solved all problems that we discovered regarding similarity scores. The retrieval method is utilized for multiple purposes. In addition to being employed in two different retrieval processes, it is used to punish adapted instances based on how radical modifications were done to the instance.

7.4 System behaviour

This section considers the following research question

RQ6: How do the creative adjustments affect the system output?

IntelliMeal consists of many different features. Several evaluation methods were therefore assessed to conduct the evaluation. With the evaluation as a whole, we aimed to evaluate both the outcome of the adaptation process and the calculated similarity. The following questions were used as basis when assessing these aspects

1. Can a human distinguish between an original recipe and a computer adapted recipe?
2. Does adaptation of recipes increase the average similarity score for the search results?
3. Does the ranking of recipes (as a product of similarity calculations) make sense to a human?

For the first question, results showed that people guessed that a human had created the computer adapted recipes in 53.43% of the cases. Also, people recognized the original cases as created by a human in 49.91% of the cases. This result reveals that most users are not able to distinguish the recipes from one another.

As for the second question, the evaluation results showed that the average similarity score increase for all the test queries. On average, the mean similarity score for the top five recipes suggested per query increased with 0.32.

For the third measurement, we compared the top five results from the user and the system. On average, the result of this process showed that the offset in ranking between the test subject's and the system was 5.52 out of 25. Also, the test subject's and the system shared 3.57 out of 5 recipes as their top 5.

Of course, collecting evaluation data may produce some faulty results. For example, during the ranking of recipes process described in Section 6.2, the test subjects resembled somehow uncomfortable and declared that they felt that they used much time to solve the task. Consequently, they sometimes rushed to give us their result. In retrospect, since the evaluation process still produced good results, we chose not to see the rankings as slipshod work, and hence, interpret the results as valid.

The system consists of several components working together. Hence, pointing out the specific parts that affect the system output is challenging. However, using such a small set of recipes as our starting point, the choice to implement a knowledge engineering approach is seen as vital to producing good results. The modeling of taxonomies is an essential aspect of the system. The taxonomies include a total of 1228 ingredients divided into 12 categories. For the system to be able to find a substitution for a recipe ingredient, it is crucial that the ingredient is present in the model and that ingredients are spelled equally across components. Consequently, the model and the cases were in need of thorough preliminary work. The adding of any missing ingredients and cross-reference checking of ingredient spelling have ensured that all ingredients are considered when doing retrieval and adaptation.

Further, the implemented rule engine opens for a more creative adaptation process by, among more, being able to substitute ingredients across categories. Cross-category substitutions increase the possibilities to adapt towards a user query, and hence, satisfy the user.

In the system, thresholds define how similar ingredients must be to consider them substitutes. The thresholds ensure that the system does not go crazy then adapting, which can lead to weird compositions and non-tasty recipes. This aspect is proven by users not being able to distinguish human and computer created recipes.

With the evaluation set aside, some drawbacks have been discovered by us as researchers during implementation. The following subsections will discuss these.

7.4.1 Drawbacks

Rules that result in poor substitutions

An observed effect with the system output is that some of the substitutions made based on rules are a bit too radical. In retrospective, we think there are two reasons for this. First, the currently included rules were written manually. The quality of these can not be guaranteed, as they were not revised by a domain expert. As explained in Subsection 4.6.1, the rule engine allows for creating general rules. We were probably not cautious enough when writing some of the rules, not considering possible cases where the rule could fire and result in an unfortunate substitution.

Secondly, in some cases, rules are written too generally because of limited functionality in the rule engine. The main drawback in the engine is the requirements for a rule to fire. These requirements refer to ingredients that are required to be in the recipe for the rule to fire. However, there is no functionality to set requirements on what ingredients must *not* be in the recipe for the rule to fire. This limits how specific a rule can be.

Orders in the adaptation process

IntelliMeal considers ingredients from the user query one by one and attempts adapting recipes to match these. The order of these ingredients is not known because it depends on the user query. Also, the order is not manipulated by the system in any way. Consequently, we have no control of in which order the substitution methods consider the ingredients.

The first successful substitution method considered for an ingredient is chosen. Hence, if there exist several solutions, the order of substitution methods decides which one should come into effect. In contrast to the ingredient order, the order of substitution methods is predefined, and the system is adjusted to it. Hence, the adaptation process should make sure that the optimal substitution is found for one isolated ingredient. However, the problem arises when one substitution destroys substitution opportunities for following ingredients that have not yet been considered. Accordingly, even if the system finds the optimal substitution for ingredients one by one, it does not necessarily find the optimal substitutions overall. In other words, it does not necessarily adapt an original case in the best possible way.

To recognize the importance of the ingredients order, consider an example where the user wants both *cucumber* and *avocado*. The recipe considered for the adaptation process contains *tomato* and *mayonnaise*, among other ingredients. Lets say that according to the substitution rules, *avocado* can be substituted in favor for *mayonnaise* and according to similarity substitutions, it can be substituted with *tomato*. In the meantime, *cucumber* can only be substituted with *tomato*. Further, lets say *avocado* is considered first and for substitution methods, the similarity substitutions are considered first. Then, *avocado* will be substituted with *tomato*. This will ruin the substitution possibilities for *cucumber*.

As this example shows, both the choice of substitution method and the order of the ingredients considered causing the unfortunate adaptation. If *cucumber* was considered first, the system could have safely substituted both desired ingredients into the recipe. Considering the substitution rule method before the similarity substitution would also result in this desired behavior.

Revise and retain

When conducting usability testing, one issue was re-occurring: adding new cases to the case base. In retrospect, we realize that hiding this functionality as a textual link which again is a part of a longer sentence is not very user-friendly. The test subjects repeatedly stated that they were looking for some button, star or heart symbol, or traditional save icon. The usability testing also revealed that the phrasing "add it to our database" appeared too technical and confusing for some test subjects. The discovered issue may not seem crucial, but having users not understand why or how to add new and tasty recipes can have significant consequences over time. CBR is the fundamental of IntelliMeal, and not evolving the system by adding new cases over time means having a system that does not utilize the revise and retain step of the CBR cycle.

7.5 Demonstrating the system

This section considers the following research question

RQ7: How can the system be demonstrated in a suitable manner to a broad range of people?

As mentioned in Section 2.1, we want to use the CCC as a tool to showcase our CBR system. The implemented system and a technical paper was submitted to the CCC board. The conference and competition itself is held at the end of June, 2017.

When discussing research on related work in Section 3.3, we stated that the more successful commercial recommender systems had put great effort into usability while earlier CCC participants have only used UIs for practical reasons, like not having to specify desired ingredients in a terminal. Because of this, we have focused on creating a user-friendly and self-explanatory UI with the ambition to stand out in the CCC and appeal to the biggest target group possible.

In general, results from the usability testing presented in Subsection 6.4 is satisfying, with the SUS result having an average of 93.25 out of 100. Test subjects easily understood all functionality and representation of data. As anticipated, the actions taken to provide explanations for the computer's train of thought helped with understanding and acceptance of the adaptations.

As a side comment: On March 8th, the *Telenor-NTNU AI-Lab*¹ was opened at the Norwegian University of Science and Technology (NTNU). The Norwegian Minister of Trade and Industry Monica Mæland, Norwegian Minister of Culture Linda Hofstad Helleland, CEO Sigve Brekke (Telenor), Rector Gunnar Bovim (NTNU), and CEO Alexandra Bech Gjørsv (SINTEF) were among the people attending the opening ceremony. The press covered the event, among others *Teknisk Ukeblad*, *Aftenposten* and NRK. The IntelliMeal system was selected as one of four student projects to be presented at the opening. We were interviewed about our project on national television and were mentioned in some articles on the web². Telenor also created a video, where we and a couple of other students at NTNU talked about our theses and AI in general. The video was shared on Telenor’s Facebook page and at Twitter by Sigve Brekke³. Afterward, we were asked to write a blog post about our thesis. This was displayed on the web page of *ntnu techzone*⁴.

May, 19th another event took place at the lab. This time, the main guest was Minister of Local Government and Modernisation, Jan Tore Sanner⁵. Again, we got the opportunity to present our project and have a discussion with Sanner himself.

In general, people see the value in the system. When presenting our thesis to non-technical people, the focus has been on everyday problems that our system can help solve. Mainly, this involves avoiding wasting food and making it easier to take allergies, diets and other restrictions on foods into consideration. People were also interested in the system as a tool to get rid of unnecessary food in their fridge, both for environmental and economical reasons.

¹www.ntnu.edu/ailab

²www.nrk.no/opna-lab-for-kunstig-intelligens-pa-ntnu

³www.twitter.com/sigve.telenor

⁴www.ntnutechzone.no/onsker-du-a-utnytte-maten-du-allerede-har-i-kjoleskapet

⁵www.regjeringen.no/statsrad-sanner-vitjar-trondheim

Chapter 8: Conclusion

Section 8.1 presents the conclusions drawn from the results presented in Chapter 6 and discussed in Chapter 7. Throughout the implementation and evaluation phase, ideas on possible improvements that could lead to better resolves have emerged, and these will be presented in Section 8.2.

8.1 Conclusion

As a result of our research, we found that CBR is indeed a proper approach for implementing a recipe recommender system due to the low rate of required expert domain knowledge and that the reuse part of the cycle provides adequate freedom to be inventive when adapting the recipes.

To further exam the effects Case-Based Reasoning and Computational Creativity can have in recipe recommender systems, we developed a knowledge engineering heavy system that utilizes the cooking domain by retrieving, comparing, adapting and suggesting recipes given a particular query. Modeling logical, clean and structured knowledge models was a focal point for the project. To improve the state of art recipe recommender systems, we incorporated the CBR cycle, developed a complex rule engine, and implemented a user-friendly and self-explanatory UI.

The system is creative in the way domain knowledge is exploited. The taxonomy knowledge is used to calculate similarities and to expand the rules. This contributed to overall satisfying results with the limited case base. As for system vulnerabilities, we learned that having a given order for adaptation is a significant drawback which can cause the system to miss out on potential adaptation branches. However, the measurable outcome of this project presented in Chapter 6 is exceedingly satisfying. Adaptation of cases increased similarity scores for a given user query in all test cases, and humans had difficulties distinguish computer and human created recipes from one another. Also, usability testing achieved a SUS score with an average of 93.25 out of 100.

The system was implemented as a prototype which apparently works excellent with a limited set of sandwich recipes. The incorporated adaptation and title rules were manually

written to target sandwich ingredients specifically. Achieving satisfying results with an extended case base including several types of dishes can not be guaranteed.

The system idea itself has a very clear utility value. It is a web application where users can search for recipes for given ingredients. Also, the system can assure people with allergies, intolerances and other preferences that any undesired ingredients are not present in the suggested recipes.

8.2 Future work

8.2.1 Data modeling

One possible weakness of the system is the knowledge model. Non-experts created the hierarchy, categorizations, and similarity measures within the taxonomy. Consequently, the hierarchy may have incoherently placed ingredients or ingredients are not necessarily seen as similar by the system even though they can perfectly substitute each other in reality.

Having well-organized hierarchies are crucial for several parts of the system to behave properly. For example, when excluding ingredients the children of the undesired ingredients are also considered undesired. This extension means that defining *meat* as undesired will exclude both chicken and ham. However, if for example *parma ham* is not located in its proper spot, below meat, the exclusion process will not consider the misplaced ingredient as undesired. For future work, the idea is to have a domain expert revise the hierarchy and local similarity measures.

Also, the rules included are manually written by non-experts. Future work would benefit from automating this process. For future work, the idea is to do a similar data collection as the AllRecipes research team described in Subsection 3.2.1. Crawling comments on recipe websites or forums can gather valuable data, and for this, domain expertise is not necessary.

8.2.2 System improvements

Consider several instances per original case

As explained in Section 7.4, the order of ingredients and substitution methods considered are a risk in the system. Even if the system finds the optimal substitution for ingredients one by one, it does not necessarily find the optimal substitutions overall. With other words, cases are not necessarily adapted in the best possible way.

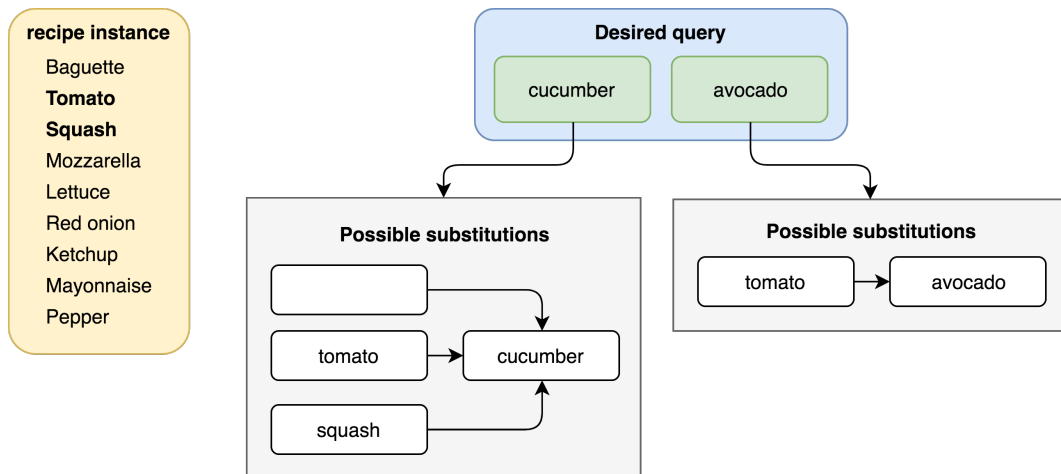


Figure 8.1: Possible substitutions for the desired ingredients *cucumber* and *avocado* considering a given recipe instance

Consider the example in Figure 8.1. In the example, the desired ingredients are *cucumber* and *avocado*. All substitution methods suggest different substitution offers for the desired ingredient *cucumber*. For the desired ingredient *avocado*, there is only one substitution alternative. Whether both substitutions can be carried out, depends on the order in which the desired ingredients are considered.

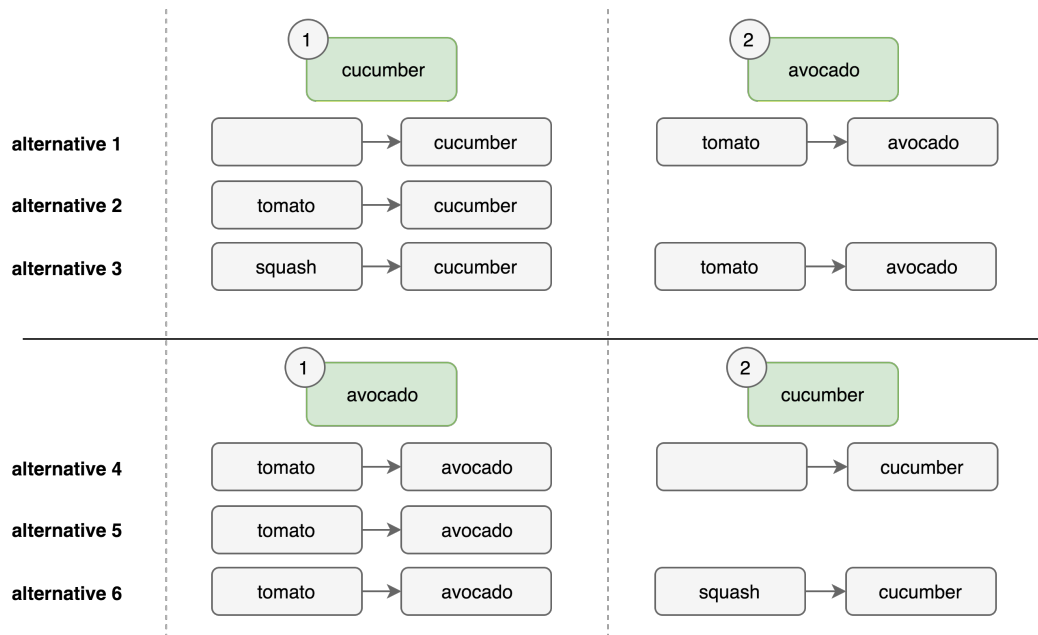


Figure 8.2: The example in Figure 8.1 results in 6 recipe alternatives when considering both possible orders and their possible substitution choices given the order

Figure 8.2 illustrates alternative recipes resulting from considering various orders. The order *cucumber, avocado* is considered at the top. Having three possible substitutions for *cucumber* and one possible substitution for *avocado*, results in $3 * 1 = 3$ alternative recipes for this order. The same applies to the second order, *avocado, cucumber*, illustrated at the bottom of the figure. Considering every order possibility *and* the order of substitution methods results in six recipe alternatives. As the system was implemented, only one alternative is created. It can not be guaranteed that the proposed alternative is always the best choice. Considering all orders open for the opportunity to find the most satisfying recipe result.

Creating several instances per original case is an idea worth exploring for the random choices made in the system as well. Instead of choosing randomly among several valid alternatives, an idea is to create one new instance per alternative. This would be an interesting approach, especially for when several suitable ingredients are found. That means the style of a recipe evolves in several directions. Implementing this can result in increased variation among the recipe recommendations.

However, this brute force method is expensive. Considering n ingredients in the user query results in $n!$ different orders to consider. In worst case scenario, all n ingredients have three substitution alternatives. This means there are $3 * 3 = 9$ different substitution method orders to consider. Consequently, there are $9 * n!$ recipes to consider per original case. When retrieving 20 of the best matching cases for adaptation, a user query containing five ingredients would result in $20 * 9 * 5! = 21600$ recipes to consider. With such a heavy adaptation process, an idea is to decrease the number of retrieved cases. Then again, less retrieved cases increase the chances of leaving good cases behind.

If the ephemeral case base retrieval includes every alternative, a significant risk is that some of the recommended recipes look extremely similar. This would probably be very confusing to the user. One idea is to advance the filtering process when creating the ephemeral case base. The system already discards duplicates. Hence, the system discards alternative four and six in this example. For further work with the idea presented, the easiest and safest filtration to do would be to discard alternatives that contain fewer desired ingredients.

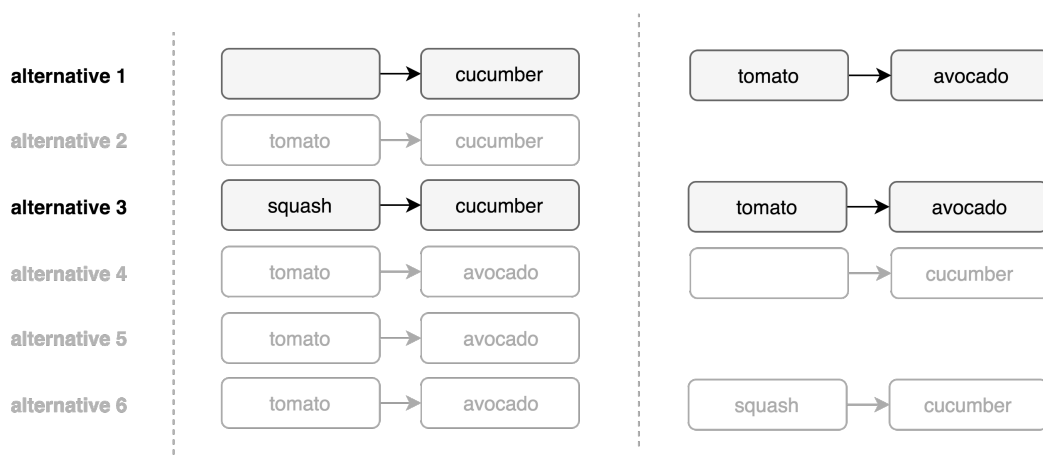


Figure 8.3: Filtering of alternative recipes

Figure 8.3 shows an example of the different resulting alternative substitutions. In the example, option two and option four are not as good because they only include one of the desired ingredients. Further, alternative five and six produced the same result as alternative one and three. For final filtration, one of the two remaining alternatives can be chosen. If the idea presented in this section is implemented in the future, searching for an optimal filtration method will be a crucial part of the work.

Advance the rule engine

The implemented rule engine is flexible in the way that one single rule can apply to one, ten or even hundreds of ingredients. The rule engine employs two specific rule types which are used in various contexts. For both types, requirements for the rule to fire can be specified. However, the requirements in rules only refer to ingredients that are *in* the recipe. It is not possible to require that an ingredient is *not* in the recipe for a rule to fire. The idea is to differ between these two types of requirements. To set the record, this functionality is called + - functionality. Further, two main advantages are presented.

-meat; fish

First, this functionality prevents the system from adding ingredients to a recipe where it will be a mismatch with other ingredients. Above is an example of an addition rule, saying "if meat is not in the recipe, fish may be added."

-meat, -fish, +mozzarella, +tomato; mozzarella tomato sandwich

Second, the functionality enables the creation of much more accurate rules, which Section 7.4 discussed as a drawback. Consider a rule saying "if mozzarella and tomato are on the recipe, the title tomato mozzarella sandwich can be generated." Whenever a recipe contains both mozzarella and tomato, this title will be generated. The problem is that other possible important ingredients in the recipe are ignored, for example, *fish* and *meat*, which are typically considered main ingredients. The rule illustrated above is an example of a rule which could prevent this behavior. In the example, the rule says "if meat or fish is not in the recipe, but mozzarella and tomato are, the title mozzarella tomato sandwich can be generated."

Scoring system for revision of cases

As mentioned in Section 7.5, the test subjects in the usability testing described in Section 6.4 had troubles finding the "add to database" link when asked to add one of the suggested cases to the database. Besides, some test subjects expressed confusion on who's database the case was being added. This section proposes a solution to the presented problem.

When revisiting recipe websites and recommender systems, a clear pattern reveals itself when it comes to the revision of suggestions. For application with personalized user profiles, recipes typically have a heart or traditional save symbol beside them which lets the user

save the recipe to a personal favorites list. Another commonly used approach is a *star rating system*. During general feedback on IntelliMeal, some test subjects mentioned that they were looking for something like this. Across the web, the approach is familiar and widely used.

The idea to improve the revise and retain part of the CBR cycle incorporated in IntelliMeal is to implement a star rating system. With this, the end user would be given the opportunity to give all suggested recipes (both adapted and not adapted) a score from one to five. Then, if an adapted recipe receives a rating higher than for example two stars, it is automatically added to the case base.

In addition to being more user-friendly, this ensures a safer revise and retain approach in the long term. With the currently implemented revise and retain approach, one single user can corrupt the quality of the case base by adding poorly adapted recipes. When doing further retrievals, these recipes are treated as complete and will be suggested, with or without adaptations, as normal.

With a rating system, however, several users influence the score of all cases. If poorly adapted recipes are added to the case base and at a later point suggested for a different user, this user gets the chance to lower the rating of this recipe by giving it zero or few stars. If a recipe over time achieves a really bad score, say less than two stars, it can either be automatically removed from the case base or manually eliminated in a regularly conducted maintenance routine.

Besides, by rating both adapted and original recipes, the system can point out which recipes are usually well received by the users. If two recipes receive the same score, it is probably preferable to display the one with the higher rating first.

Functionality expansion

In addition to the core functionality improvement suggestions, some commercially related ideas have evolved. The more obvious idea is expanding the query functionality to exclude groups of food for a set of diets. For example, by having checkboxes for diets like *vegetarism*, *dairy-free*, *low-carb*, and so forth. Another idea is letting users have private profiles. In the profile, the user could explicitly define ingredients or groups of them that they do not want recipes to contain.

Having user profiles would also give the opportunity for more personalized suggestions; if a user tends always to search for healthy ingredients and upvote healthy dishes, it is probably inadequate to suggest a dish in the complete opposite direction.

The more extreme idea is to extend the system with hardware. If a user could check in and out groceries in the fridge and kitchen cabinets, the system could automatically utilize these ingredients when suggesting a dish. If the system has knowledge of all available ingredients, it could even propose a complete meal plan for users.

Appendix A: Evaluation and results

This appendix contains documents and papers used during the evaluation of IntelliMeal, as well as the raw results from the evaluation.

A.1 Results from evaluation of similarity scores

Adaptation	Query										Avg.
	1	2	3	4	5	6	7	8	9	10	
FALSE	0.35	0.81	0.62	0.45	0.53	0.42	0.34	0.36	0.73	0.74	0.56
TRUE	1	1	0.74	0.83	0.68	0.67	0.76	0.78	1	0.91	0.84
Improvement	0.65	0.19	0.12	0.38	0.15	0.25	0.42	0.42	0.27	0.17	0.30

Table A.1: The average score for the top five results with and without adaptation

A.2 Ranking of recipes

System ranking	Equal ranking	Offset	Opposite ranking	Offset
1	1	$1 - 1 = 0$	10	$10 - 1 = 9$
2	2	$2 - 2 = 0$	9	$9 - 2 = 7$
3	3	$3 - 3 = 0$	8	$8 - 3 = 5$
4	4	$4 - 4 = 0$	7	$7 - 4 = 3$
5	5	$5 - 5 = 0$	6	$6 - 5 = 1$
...				
10				
Sum		0		25

Table A.2: Ranking measure for evaluation of ranking of recipes test. This gives 0 being the best possible score (no offset between ranked recipes from system to test subject) and 25 the worst possible score.

A.3 Results from ranking of recipes

Task	CaseName	System ranking
1	Cookery16	1
	Cookery11-1	2, 3, 4
	Cookery11	2, 3, 4
	Cookery11-2	2, 3, 4
	Cookery19	5
	Cookery14-1	6, 7, 8
	Cookery13	6, 7, 8
	Cookery14	6, 7, 8
	Cookery4	9, 10
	Cookery17	9, 10
2	Cookery2	1
	Cookery2-1	2
	Cookery10	3, 4
	Cookery4	3, 4
	Cookery12	5
	Cookery8-1	6, 7
	Cookery8	6, 7
	Cookery1	8, 9
	Cookery15-1	8, 9
	Cookery3	10
3	Cookery 11-1	1
	Cookery15	2
	Cookery13	3
	Cookery6	4, 5
	Cookery20	4, 5
	Cookery10	6, 7
	Cookery19	6, 7
	Cookery16	8
	Cookery5	9
	Cookery13-1	10

Table A.3: The system’s ranking of recipes per query. Some recipes have several rankings because they share the exact same similarity score with other recipes.

Rank	Task 1		Task 2		Task 3		Avg.
	Case	Offset	Case	Offset	Case	Offset	
1	Cookery16	0	Cookery2	0	Cookery 11-1	0	
2	Cookery11	0	Cookery2-1	0	Cookery13	1	
3	Cookery11-1	0	Cookery4	0	Cookery6	1	
4	Cookery11-2	0	Cookery10	0	Cookery15	2	
5	Cookery13	1	Cookery8-1	1	Cookery19	1	
Sum		1		1		5	2.33
1	Cookery16	0	Cookery2	0	Cookery 11-1	0	
2	Cookery11	0	Cookery2-1	0	Cookery15	0	
3	Cookery13	3	Cookery10	0	Cookery20	1	
4	Cookery14	2	Cookery4	0	Cookery6	0	
5	Cookery14-1	1	Cookery8	1	Cookery13	2	
Sum		6		1		3	3.33
1	Cookery19	4	Cookery2	0	Cookery 11-1	0	
2	Cookery16	1	Cookery10	1	Cookery13	1	
3	Cookery11	0	Cookery2-1	1	Cookery13-1	7	
4	Cookery11-1	0	Cookery4	0	Cookery15	2	
5	Cookery17	4	Cookery8	1	Cookery19	1	
Sum		9		3		11	7.67
1	Cookery16	0	Cookery2	0	Cookery 11-1	0	
2	Cookery19	3	Cookery10	1	Cookery15	0	
3	Cookery14	3	Cookery4	0	Cookery13	0	
4	Cookery13	2	Cookery2-1	2	Cookery13-1	6	
5	Cookery11-2	1	Cookery8	1	Cookery19	1	
Sum		9		4		7	6.67
1	Cookery16	0	Cookery2	0	Cookery 11-1	0	
2	Cookery11	0	Cookery10	1	Cookery15	0	
3	Cookery13	3	Cookery4	0	Cookery13	0	
4	Cookery11-1	0	Cookery2-1	2	Cookery13-1	6	
5	Cookery14-1	1	Cookery8	1	Cookery16	3	
Sum		4		4		9	5.67
1	Cookery16	0	Cookery2	0	Cookery 11-1	0	
2	Cookery17	7	Cookery10	1	Cookery15	0	
3	Cookery4	6	Cookery4	0	Cookery20	1	
4	Cookery13	2	Cookery2-1	2	Cookery6	0	
5	Cookery14-1	1	Cookery8	1	Cookery13	2	
Sum		16		4		3	7.67
1	Cookery14	5	Cookery2	0	Cookery 11-1	0	
2	Cookery16	1	Cookery2-1	0	Cookery15	0	
3	Cookery11-1	0	Cookery10	0	Cookery13	0	
4	Cookery19	1	Cookery4	0	Cookery13-1	6	
5	Cookery11	1	Cookery8	1	Cookery19	1	
Sum		8		1		7	5.33
Avg.		7.57		2.57		6.43	5.52

Table A.4: The offset of recipe ranks between the system and each test subject. The average number of correct recipes per user can be seen at the rightmost column, while the average of correct recipes per query is shown in the bottom row.

A.4 Queries used and cases added to quiz case base

	Desired ingredients	Undesired ingredients	Added recipe
1	tuna	-	Tuna Cucumber Sandwich Tuna and Avocado Sandwich
2	pain de mie, beans, avocado	hummus, mint, radish	Baked Bean
3	baguette, pork, barbeque sauce	parma ham	BBQ Sandwich
4	egg, cheese, pepper, tomato, lettuce	parsley, chive	Breakfast Pita
5	roast beef, cheese, pickled cucumber, tomato	-	Bauru Sandwich
6	avocado, bacon, mozzarella, pesto, tomato	-	California Club
7	tuna, onion, chick pea	-	Tuna Chick Pea Sandwich
8	peanut butter, banana, bacon	-	Elvis sandwich
9	mozzarella, tomato	-	Mozzarella Tomato Sandwich Tuna Tomato Sandwich
10	pita bread, lamb	-	Lamb Sandwich
11	baguette, minced meat, cheese, cumin, garlic powder, salt, chili powder, ketchup, oregano	-	Taco Baguette
12	salmon, tomato	-	Salmon Tzatziki Bagel Salmon Tomato Sandwich
13	tortilla, cheddar	-	Taco Tortilla Nacho Cheddar Sandwich
14	tortilla, cream cheese, salmon	-	Salmon Roll
15	roast beef	-	Roast Beef Sandwich
16	parma ham, cheese	-	Creamy Parma Sandwich
17	parma ham, avocado	curry powder	Exotic Baguette

Table A.5: Queries used to create the cases added to the "Bot or Not?" case base. The right column shows the name of the added recipes.

A.5 Results from quiz

CaseName	Adapted	Responses				
		Correct	Incorrect	Total	Correct in %	Incorrect in %
Cookery12	FALSE	47	25	72	65,28%	34,72%
Cookery4	FALSE	43	27	70	61,43%	38,57%
Cookery10	FALSE	44	28	72	61,11%	38,89%
Cookery11	FALSE	40	29	69	57,97%	42,03%
Cookery13	FALSE	40	32	72	55,56%	44,44%
Cookery6	FALSE	32	26	58	55,17%	44,83%
Cookery19	FALSE	38	31	69	55,07%	44,93%
Cookery1	FALSE	39	33	72	54,17%	45,83%
Cookery8	FALSE	32	30	62	51,61%	48,39%
Cookery14	FALSE	39	37	76	51,32%	48,68%
Cookery7	FALSE	34	35	69	49,28%	50,72%
Cookery5	FALSE	34	35	69	49,28%	50,72%
Cookery20	FALSE	36	40	76	47,37%	52,63%
Cookery3	FALSE	28	32	60	46,67%	53,33%
Cookery16	FALSE	34	40	74	45,95%	54,05%
Cookery17	FALSE	30	37	67	44,78%	55,22%
Cookery15	FALSE	28	37	65	43,08%	56,92%
Cookery2	FALSE	28	38	66	42,42%	57,58%
Cookery0	FALSE	29	41	70	41,43%	58,57%
Cookery9	FALSE	28	44	72	38,89%	61,11%
Cookery18	FALSE	20	34	54	37,04%	62,96%
Cookery3-1	TRUE	40	21	61	65,57%	34,43%
Cookery14-3	TRUE	41	26	67	61,19%	38,81%
Cookery16-1	TRUE	39	31	70	55,71%	44,29%
Cookery4-1-2	TRUE	36	29	65	55,38%	44,62%
Cookery2-3	TRUE	38	31	69	55,07%	44,93%
Cookery8-2	TRUE	30	30	60	50,00%	50,00%
Cookery6-1	TRUE	30	31	61	49,18%	50,82%
Cookery5-2	TRUE	28	30	58	48,28%	51,72%
Cookery5-6-1-1	TRUE	31	34	65	47,69%	52,31%
Cookery5-5	TRUE	31	37	68	45,59%	54,41%
Cookery0-3	TRUE	35	42	77	45,45%	54,55%
Cookery4-3	TRUE	29	35	64	45,31%	54,69%
Cookery4-1-1	TRUE	31	38	69	44,93%	55,07%
Cookery2-3-1	TRUE	31	43	74	41,89%	58,11%
Cookery9-1	TRUE	27	40	67	40,30%	59,70%
Cookery12-2-1	TRUE	28	42	70	40,00%	60,00%
Cookery12-2	TRUE	27	42	69	39,13%	60,87%
Cookery5-6-1	TRUE	27	43	70	38,57%	61,43%
Cookery5-5-1	TRUE	27	44	71	38,03%	61,97%
Cookery5-6	TRUE	26	44	70	37,14%	62,86%
Cookery4-1	TRUE	20	48	68	29,41%	70,59%
		1375	1472	2847	48,30%	51,70%

Table A.6: Guessing results per recipe for the "Bot or not?" quiz

A.6 Usability testing

Tasks

1.
 - a. Search for a sandwich recipe containing “egg”, “avocado”, “cherry tomato” and “tuna”.
 - i. Does any of the recipes proposed fulfill all your wishes?
 - b. You realize you do not want the recipe to contain “tabasco sauce”. Add this to your search.
 - i. Does any of the recipes proposed fulfill all your wishes?

2.
 - a. Search for a sandwich recipe containing “tortilla” “chicken” and “avocado”.
 - b. Look at the top 1 result. Identify whether the recipe is adapted, and if so, identify which ingredients this may concern.

3.
 - a. Search for a sandwich recipe containing “tortilla” and “cheddar”, but no “lettuce”.
 - b. Look at the top 1 result. Identify whether the recipe is adapted, and if so, identify which ingredients this may concern.
 - c. Add the top 1 recipe to the database and confirm the operation was successful.

Figure A.1: Tasks for usability testing

Prior to user testing

A.1 I enjoy making food

	Strongly disagree		Strongly agree	
1	2	3	4	5

A.2 I see myself as a clever cook

	Strongly disagree		Strongly agree	
1	2	3	4	5

A.3 I use online recipe search engines regularly ("matprat.no" etc.)

	Strongly disagree		Strongly agree	
1	2	3	4	5

The following three questions is to be answered only if you indeed use recipe search engines regularly:

A.4 I am usually satisfied with the recipes search engines propose to me

	Strongly disagree		Strongly agree	
1	2	3	4	5

A.5 I feel that using recipe search engines help me learn about cooking and recipe composition

	Strongly disagree		Strongly agree	
1	2	3	4	5

A.6 I often feel that online recipe search engines do not take all my desires into consideration

	Strongly disagree		Strongly agree	
1	2	3	4	5

A.7 If you do **not** search for recipes on online recipe pages - where do you find recipes?

--	--

A.8 If you find recipes other places than online recipe pages - why do you prefer this?

--	--

Figure A.2: Reply form to be filled out by the test subject prior to the usability test

System Usability Scale

B.1	I think that I would like to use this system frequently	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.2	I found the system unnecessarily complex	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.3	I thought the system was easy to use	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.4	I think that I would need the support of a technical person to be able to use this system	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.5	I found the various functions in this system were well integrated	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.6	I thought there was too much inconsistency in this system	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.7	I would imagine that most people would learn to use this system very quickly	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.8	I found the system very cumbersome to use	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.9	I felt very confident using the system	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					
B.10	I needed to learn a lot of things before I could get going with this system	<div style="display: flex; justify-content: space-between; font-size: small;"> Strongly disagree Strongly agree </div> <table border="1" style="width: 100%; height: 20px; border-collapse: collapse;"> <tr> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> <td style="width: 20%;"></td> </tr> </table> <div style="display: flex; justify-content: space-around; font-size: small;"> 1 2 3 4 5 </div>					

Figure A.3: System Usability Scale Form to be filled out by the test subject after the usability test

Subsequent to user testing

- C.1 I believe that such a system could improve my knowledge about ingredient similarity/recipe composition
- Strongly disagree Strongly agree
- | | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
- C.2 I believe that such a system could, over time, enhance my cooking abilities
- Strongly disagree Strongly agree
- | | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
- C.3 I believe that such a system could help me throw away less food and ingredient remains
- Strongly disagree Strongly agree
- | | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
- C.4 I would recommend such a system to my friends or family
- Strongly disagree Strongly agree
- | | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
- C.5 I believe I would prefer using an *adapting* recipe search engine over a non-adapting recipe search engine
- Strongly disagree Strongly agree
- | | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

Age:

Any additional comments:

Figure A.4: Reply form to be filled out by the test subject after the usability test

A.7 Results from usability testing

Question	Answers per test subject										Average
A.1	5	5	5	5	3	4	3	5	4	3	4.20
A.2	4	4	5	5	2	4	2	3	4	3	3.60
A.3	1	5	5	2	3	3	1	4	2	4	3.00
A.4		4	4	3	4	4		4	4	5	4.00
A.5		2	5	4	3	3		4	4	2	3.38
A.6		4	3	4	3	3		2	3	3	3.13

Table A.7: Replies to pre usability testing form. The rightmost column represent the average score per question.

Question	Answers per test subject										Score
B.1	5	5	5	5	3	5	5	4	3	5	87.50
B.2	1	1	1	1	2	1	1	1	1	1	97.50
B.3	5	5	5	5	4	4	4	5	5	4	90.00
B.4	1	1	1	1	1	2	1	1	1	1	97.50
B.5	5	5	5	5	4	4	5	4	5	5	92.50
B.6	1	2	1	1	2	1	1	1	2	1	92.50
B.7	5	5	5	5	3	4	4	5	5	5	90.00
B.8	1	1	1	1	1	2	1	1	1	1	97.50
B.9	5	4	5	5	4	4	5	4	5	5	90.00
B.10	1	1	1	1	2	1	1	1	1	1	97.50
Score	100	95	100	100	75	85	95	92.5	92.5	97.5	93.25

Table A.8: Replies to SUS form. The score per questions can be seen in the rightmost column, while the score per user can be seen at the bottom row.

Question	Answers per test subject										Average
C.1	5	2	5	5	4	5	4	4	4	3	4.10
C.2	5	3	5	4	4	4	5	4	4	4	4.20
C.3	3	4	5	5	3	5	5	4	5	5	4.40
C.4	4	5	5	5	3	4	5	5	5	5	4.60
C.5	3	4	5	5	4	4	4	5	5	5	4.40
Average	4	3.6	5	4.8	3.6	4.4	4.6	4.4	4.6	4.4	4.34

Table A.9: Replies to post usability testing form. The average score per question can be seen in the rightmost column, while the average score per test subject is represented by the bottom row.

Acronyms

AI Artificial Intelligence. 1–3, 5, 6, 8, 9, 21, 23, 74, 80

API Application Programming Interface. 22, 52, 55–57, 60

CBR Case-Based Reasoning. 1, 3, 5–9, 11–16, 20, 23, 25, 29, 30, 49, 54, 55, 63, 72–74, 79, 81, 86

CC Computational Creativity. 2, 72, 81

CCC Computer Cooking Contest. 5, 13–16, 18, 19, 23, 24, 26, 51, 74, 75, 79

ICCBR International Conference on Case Based Reasoning. 5

SUS System Usability Scale. 69, 71, 79, 81

UI User Interface. 18, 19, 24, 52, 54–57, 59–61, 72, 75, 79, 81

Glossary

CSV In computing, a comma-separated values (CSV) file stores tabular data, both numbers and text, in plain text. 27, 52, 53

framework A reusable set of libraries or classes for a software system. 19

JSON A simple, text based standard for exchange of data on the web. 52–54, 57

myCBR An open-source similarity-based retrieval tool and software development kit (SDK). 26, 27, 31–36, 39, 51–55, 75, 76

ontology The formal naming and definition of the types, properties, and interrelationships of the entities that fundamentally exist for a particular domain. 15, 18, 19, 23

Spring An application framework used for building web applications on top of the Java EE platform. 55

Swagger An open source framework that helps design, build, document, and consume RESTful APIs. 55

taxonomy The classification and naming of concepts in an ordered system that is intended to indicate natural relationships. 15–20, 24, 26, 28, 31, 35, 36, 39, 52–55, 74, 75, 77, 81, 82

TypeScript An open-source programming language - a superset of JavaScript which adds optional static typing and class-based object-oriented programming to JavaScript. 56

wiki A website that provides collaborative modification of content directly in the browser. 15, 23

XML In computing, Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. 16, 18, 20, 26, 27, 51, 52, 54

Bibliography

- Aamodt, Agnar (1991). “A Knowledge-Intensive, Integrated Approach to problem Solving and Sustained Learning”. PhD thesis. University of Trondheim.
- (1995). “Knowledge Acquisition and Learning by Experience - The Role of Case-Specific Knowledge”. In: *Machine Learning and Knowledge Acquisition; Integrated Approaches*.
- Aamodt, Agnar and Enric Plaza (1994). “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”. In: IOS Press.
- Badra, Fadi et al. (2009). “Knowledge Acquisition and Discovery for the Textual Case-Based Cooking system WIKITAAABLE”. In: *Computer Cooking Contest Workshop Proceedings*.
- Ball, Philip (2012). “Iamus, Classical Music’s Computer Composer, Live from Malaga”. In: *The Guardian*.
- Ballesteros, Miguel, Raúl Martín, and Belén Díaz-Agudo (2010). “JADAWeb: A CBR System for Cooking Recipes”. In: *Computer Cooking Contest Workshop Proceedings*.
- Bangor, Aaron, Philip Kortum, and James Miller (2009). “Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale”. In: *Journal of Usability Studies*.
- Barletta, Ralph and Dan Hennesy (1989). “Case Adaptation in Autoclave Layout Design”. In: *Proceedings in the 2nd Workshop on Case-Based Reasoning*.
- BBC (2014). “Robot Writes LA Times Earthquake Breaking News Article”. In: *BBC News*.
- Besold, Tarek R., Marco Schorlemmer, and Alan Smaill (2015). *Computational Creativity Research: Towards Creative Machines*. Atlantis Press.
- Blansch e, Alexandre et al. (2010). “TAAABLE 3: Adaptation of ingredient quantities and of textual preparations”. In: *Computer Cooking Contest Workshop Proceedings*.
- Brooke, John (1996). “SUS - A quick and dirty usability scale”. In: *Usability Evaluation in Industry*.
- Brooks, Rodney (1991). “Intelligence Without Representation”. In: *Artificial Intelligence*. Elsevier.
- Brown, Carol and Uma Gupta (1994). “Applying Case-Based Reasoning to the Accounting Domain”. In: *Intelligent Systems in Accounting, Finance and Management*.
- Brown, Mark (2015). “World’s First Computer-Generated Musical to Debut in London”. In: *The Guardian*.
- Burke, Robin (2000). “Knowledge-Based Recommender Systems”. In: *Enclopydia of Library and Information Systems*.
- Cawsey, Alison (1998). *Rule-Based Systems*. Tech. rep. Heriot-Watt University.
- Colton, Simon and Geraint Wiggins (2012). “Computational Creativity: The final Frontier?”. In: *Proceedings of the 20th European Conference on Artificial Intelligence*.

- David, Jean-Marc, Jean-Paul Krivine, and Simmons Reid (1993). *Second generation expert systems*. Springer Verlag.
- Do, Hyunsook, Gregg Rothmel, and Alex Kinneer (2006). “Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis”. In: *Empirical Software Engineering*.
- Dumas, Joseph and Janice Redish (1999). *A Practical Guide to Usability Testing*. Intellect.
- Ferrucci, David et al. (2010). “Building Watson: An Overview of the DeepQA Project”. In: *AI Magazine*.
- Fitchard, Kevin (2013). “Yummly opens up its recipe API to food app developers”. In: *Gigaom*.
- Freßmann, Andrea et al. (2005). “Collaborative Agent-Based Knowledge Support for Empirical and Knowledge-Intense Processes”. In: *Multi-Agent System Technologies*. Springer-Verlag, 235–236.
- Fuchs, Christian et al. (2009). “Cooking Cake”. In: *Computer Cooking Contest Workshop Proceedings*.
- Gaillard, Emmanuelle, Jean Lieber, and Emmanuel Nauer (2015). “Improving Ingredient Substitution using Formal Concept Analysis and Adaptation of Ingredient Quantities with Mixed Linear Optimization”. In: *Computer Cooking Contest Workshop Proceedings*.
- Gaillard, Emmanuelle et al. (2014). “Tuurbine: A Generic CBR Engine over RDFS”. In: *Case-Based Reasoning Research and Development*. Ed. by Luc Lamontagne and Enric Plaza. Springer, pp. 140–154.
- Goldfisher, Alastair (2010). “Startup Yummly like ”Google for food””. In: *Reuters*.
- Goodman, Marc (1989). “CBR in Battle Planning”. In: *Proceedings in the 2nd Workshop on Case-Based Reasoning*.
- Grace, Kazjon et al. (2016). “Combining CBR and Deep Learning to Generate Surprising Recipe Designs”. In: *Lecture Notes in Computer Science*.
- Hayes-Roth, Frederick (1985). “Rule-based Systems”. In: *Communications of the ACM*.
- Hayes-Roth, Frederick, Donald Waterman, and Douglas Lenat (1983). *Building Expert Systems*. Longman Publishing.
- Herlocker, Jonathan L., Joseph A. Konstan, and John Riedl (2000). “Explaining collaborative filtering recommendations”. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*.
- Herrera, Javier et al. (2009). “JaDaCook 2: Cooking Over Ontological Knowledge”. In: *Computer Cooking Contest Workshop Proceedings*.
- Hohwü, Lena et al. (2013). “Web-Based Versus Traditional Paper Questionnaires: A Mixed-Mode Survey With a Nordic Perspective”. In: *Journal of Internet Research*.
- Ihle, Norman, Alexandre Hanft, and Klaus-Dieter Althoff (2009). “Extraction of Adaptation Knowledge from Internet Communities”. In: *Computer Cooking Contest Workshop Proceedings*.
- Jannach, Dietmar et al. (2010). *Recommender Systems: An Introduction*. Cambridge University Press.
- Klein, Gary (2008). “Naturalistic Decision Making”. In: *Golden Anniversary*.
- Kolodner, Janet (1992). “An Introduction to Case-Based Reasoning”. In: *Artificial Intelligence Review*.
- Leake, David (1996). “CBR in Context: The present and Future”. In: *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press.

- Leake, David and David Wilson (2001). “Maintaining Case-Based Reasoners: Dimensions and Directions”. In: *Computational Intelligence*.
- Liao, Warren, Zhiming Zhang, and Claude Mount (2010). “Similarity Measures for Retrieval in Case-Based Reasoning Systems”. In: *Applied Artificial Intelligence*.
- Maes, Pattie and Upendra Shardanand (1995). “Social Information Filtering: Algorithms for Automating “Word of Mouth””. In: *CHI '95 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.
- Mantaras, Ramon et al. (2006). “Retrieval, Reuse, Revision and Retention in Case-Based Reasoning”. In: *The Knowledge Engineering Review*. Cambridge University Press.
- March, Salvatore T. and Gerald F. Smith (1995). “Design and natural science research on information technology”. In: *Decision Support Systems*.
- Minor, Mirjam et al. (2010). “Adaptation of Cooking Instructions Following the Workflow Paradigm”. In: *Computer Cooking Contest Workshop Proceedings*.
- Mizzaro, Stefano and Carlo Tasso (2002). “Ephemeral and Persistent Personalization in Adaptive Information Access to Scholarly Publications on the Web”. In: *Proceeding AH '02 Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*.
- Montani, Stefania and Lakhmi C. Jain (2014). “Case-Based Reasoning Systems”. In: *Successful Case-based Reasoning Applications 2*. Springer Berlin Heidelberg.
- Müller, Gilbert and Ralph Bergmann (2015). “CookingCAKE: A Framework for the adaptation of cooking recipes represented as workflows”. In: *Computer Cooking Contest Workshop Proceedings*.
- Newo, Régis et al. (2010). “On-Demand Recipe Processing based on CBR”. In: *Computer Cooking Contest Workshop Proceedings*.
- Pinel, Florian (2015). “What’s Cooking with Chef Watson?” In: *IEEE Pervasive Computing*.
- Ricci, Francesco, Lior Rokach, and Shapira Bracha (2011). *Recommender Systems Handbook*. Springer.
- Richter, Michael and Rosina Weber, eds. (2013). *Case-Based Reasoning*. Springer.
- Roth-Berghofer, Thomas R. (2003). *Knowledge Maintenance of Case-Based Reasoning Systems*. Akademische Verlagsgesellschaft Aka GmbH, Berlin.
- Simon, Herbert A. (1996). *The Sciences of the Artificial*. MIT Press.
- Teng, Chun-Yuen, Yu-Ru Lin, and Lada A. Adamic (2012). “Recipe recommendation using ingredient networks”. In: *WebSci*.
- Turing, Alan (1950). “Computing Machinery and Intelligence”. In: *Mind*.
- Upbin, Bruce (2013). “IBM’s Watson Gets Its First Piece Of Business In Healthcare”. In: *Forbes Tech*.
- Ye, L. Richard and Paul E. Johnson (1995). “The Impact of Explanation Facilities on User Acceptance of Expert Systems Advice”. In: *MIS Quarterly*.