Doctoral theses at NTNU, 2020:193

Tor Gunnar Høst Houeland

Automated lazy metalearning in introspective reasoning systems

Doctoral thesis

NTNU
Norwegian University of Science and Technology
Thesis for the Degree of
Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Tor Gunnar Høst Houeland

Automated lazy metalearning in introspective reasoning systems

Thesis for the Degree of Philosophiae Doctor

Trondheim, June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Computer Science

ISBN 978-82-326-4734-7 (printed ver.)
ISBN 978-82-326-4735-4 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2020:193

Printed by NTNU Grafisk senter

Abstract

Machine learning systems are becoming increasingly widespread and important, and these days machine learning is used in some form in most industries. However, the application of machine learning technology still has a relatively high barrier to entry, requiring both machine learning expertise and domain knowledge.

In this thesis, we present a metareasoning approach to multi-method machine learning that allows the system to adapt and optimize learning for a given domain automatically, without requiring human expert judgment. In contrast to popular deep learning methods for similar situations, the approach presented here does not require specialized hardware nor large data sets.

Multiple machine learning components are continuously evaluated at run-time while solving problems, using a framework to analyze overall system performance based on observed prediction performance and time spent. The system automatically learns to prioritize the methods with the best empirical performance for a given domain.

In experiments using data sets from the UCI machine learning repository and machine learning methods from the Weka suite, an example implementation outperformed individual methods and other metareasoning approaches.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of philosophiae doctor. The research work reported herein was conducted at the Department of Computer Science (IDI), NTNU.

This doctoral work was financed by and conducted at the Department of Computer Science, NTNU, Trondheim, with Agnar Aamodt as the main supervisor and Helge Langseth as co-supervisor.

Additionally, Google supported me by allowing me to spend part of my time at work to complete writing the last paper "A Learning System based on Lazy Metareasoning" and this thesis itself.

Acknowledgments

Firstly, I would like to thank my primary supervisor Agnar Aamodt for his help throughout my research. Our discussions and his feedback were foundational for the research presented in this thesis, and I could not have completed this work without his continued support and encouragement.

Secondly, I would also like to thank my co-supervisors, my research group and other researchers at IDI, and my fellow PhD students for their collaboration, helpful feedback, and many interesting discussions.

I would also like to thank Google and my managers there for letting me spend "20% time" and take time off when needed to complete the work on my thesis.

Finally, I would like to thank my wife Weilin and my parents for their love and support, especially during the personally challenging third phase of my thesis work. I would not have been able to complete this work without them.

Contents

Part I

Research overview

Chapter 1
Introduction

1.1 Motivation

Machine learning systems have achieved considerable success in recent years, and are being used in more and more applications. However, new applications commonly rely on human experts to perform crucial tasks: modeling the domain, processing the data to be suitable for machine learning, selecting an appropriate machine learning method, optimizing parameters, etc. These decisions are domain-dependent and often rely on domain experts working together with knowledge engineers to extract and structure the required domain knowledge, which is costly and time-consuming. This is known as the knowledge acquisition bottleneck, which is limiting the usefulness of machine learning for practical applications.

Automatic machine learning (AutoML) is an alternative to this expertise-driven process, where machine learning systems instead learn entirely autonomously from data without human intervention. To achieve this goal, a computer system needs to simultaneously reason about different knowledge types and about how to combine them to meet higher-level goals.

This thesis explores how a practical system can effectively combine several types of reasoning based on explicit representations of reasoning methods and meta-reasoning processes, without requiring human experts or specialized hardware. In particular, the ALMA system presented in this thesis is a metareasoning system [15] addressing three fundamental issues identified by Leake and Wilson [33]: having a flexible learning focus, enabling multistrategy introspective learning, and monitoring processing characteristics in addition to outcomes.

1.2 Research objectives

The overall goal of the research presented in this thesis is to examine the role of learning in introspective reasoning systems, and how this kind of learning can be achieved in fully automated systems that do not require manual expert configuration and tuning.

This overall goal is divided into four specific research questions. The first two research questions form the background for developing such systems: what advantages do the combination of learning and introspection provide for such systems, and how does it set them apart from other paradigms? The latter two research questions concern the development of a new system demonstrating these advantages, which includes a working metalearning design that can be applied to practical problem-solving.

Introspective learning research questions

RQ1 How can learners perform better using introspective capabilities?

This research question is concerned with examining what the capabilities of introspective reasoners are, and specifically in which ways they can improve learning abilities. The focus is on fully automated reasoning systems, which means that any such introspective capabilities need to be clear and explicit enough to be programmable and able to run autonomously without human assistance.

RQ2 How can the performance of reasoning systems with different capabilities be evaluated?

Evaluating a system's performance is essential to know whether it is actually improving or not. A high-level goal might be to increase the intelligence of a system, but the exact meaning of intelligence is notoriously vague and difficult to define precisely. Instead, performance can be based on the results achieved, which are objective and unambiguous but do not necessarily generalize beyond exactly what was measured. The challenge here is to determine meaningful performance measurements that are sufficient to demonstrate useful and effective overall system improvements, particularly with limited computing resources available.

Architecture development research questions

RQ3 How can a lazy metareasoning architecture take advantage of introspective capabilities?

Metareasoning enables a system to reason at a higher level about its own reasoning capabilities, not just to reason at the object-level about the specific problems to be solved. Lazy learning allows a system to learn from new knowledge that was not known ahead of time. The goal of this research question is to demonstrate a system architecture based on lazy metareasoning that can learn about its own reasoning capabilities and is able to improve them with experience.

RQ4 How can a practical metareasoning system automatically adapt to application-specific characteristics?

Inductive reasoning, predicting the future, or generalizing beyond training examples is not justifiable in the general case without making any assumptions. To be effective, machine learning always depends on an application

domain having sufficient structure and redundancy to make meaningful inferences. The effectiveness of a machine learning method can be viewed as how well the learning method assumptions match the domain characteristic. This research question is concerned with developing methods to allow a lazy metareasoning system to reason efficiently and effectively in diverse and practical application domains without specific manual tuning.

1.3 Research methodology

The work presented in this thesis is primarily pure research to discover new knowledge, with an objective of exploring new forms of learning. Initially, a literature study was performed to describe how learning was performed in existing systems and machine learning paradigms, and then the novel work was to examine several different ways to extend and generalize these approaches to gain new capabilities. The focus was on new ways of combining existing methods from different sub-fields, rather than creating new methods from scratch.

An important part of this research was to consider empirical performance - how well methods actually make predictions in concrete reasoning scenarios. This is in contrast to a popular direction in statistical learning theory, which assume a given form of underlying process is generating the data, and then analyze models and derive statistical guarantees that apply more generally to all methods when the assumptions hold. The problem with that approach for this research is two-fold: first, the assumptions typically don't hold perfectly, for example samples are often not drawn completely independently from the exact same distribution but many analyses assume so. Second, there are large amounts of unknown internal structure in the problem domains that make the problem much easier than solving "the general case". To perform well without requiring human tuning, learning methods have to autonomously discover and take advantage of this structure in some way.

Empirical performance for real scenarios was especially important in the last part of the work, which was a constructive proof that the proposed ALMA architecture can achieve the desired improvements, by actually implementing a sample system and running it with a varied set of reasoning methods and data sets.

To evaluate the research results, the performance of the implemented ALMA system was compared with other approaches in an online learning benchmark with 21 data sets. This also includes a statistical analysis of the per-data set results, showing that most of the results are highly statistically significant.

1.4 Thesis structure

This thesis is a compilation thesis, with the main research contributions contained in already-published papers which are reproduced in the second part of the thesis. The overall structure of the thesis is as follows:

1.4.1 Part I

Chapter 1 introduced the underlying motivation and main objectives for the thesis research.

Chapter 2 presents background and a sample of related research work from the literature, together with basic design principles for this research work.

Chapter 3 presents an overview of the research process followed for this thesis, including an overview of the papers and how their topics correspond to the stated research objectives.

Chapter 4 evaluates and discusses our results.

Chapter 5 ends the thesis with concluding remarks and possible directions for future research.

1.4.2 Part II

The papers build on top of each other, matching the way the research questions build on top of each other. Each paper has been classified as primarily addressing one research question, based on the most important contribution:

[RQ1] How can learners perform better using introspective capabilities?

Paper A Houeland, T.G., Aamodt, A.: An Introspective Component-Based Approach for Meta-Level Reasoning in Clinical Decision Support Systems. In: Kofod-Petersen, A., Langseth, H., Gundersen, O. E. (eds.) Proceedings of the First Norwegian Artificial Intelligence Symposium (NAIS'09). pp. 121–132. Tapir Forlag (2009), ISBN: 978-82-519-2519-8

[RQ2] How can the performance of reasoning systems with different capabilities be evaluated?

Paper B Houeland, T.G., Aamodt, A.: The Utility Problem for Lazy Learners - Towards a Non-eager Approach. In: Bichindaritz, I., Montani, S. (eds.) Case-Based Reasoning. Research and Development, Lecture Notes in Computer Science, vol. 6176, pp. 141–155. Springer (2010), `https://doi.org/10.1007/978-3-642-14274-1_12`

[RQ3] How can a lazy metareasoning architecture take advantage of introspective capabilities?

Paper C Houeland, T.G.: An Efficient Random Decision Tree Algorithm for Case-Based Reasoning Systems. In: Murray, R.C., McCarthy, P.M. (eds.) Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, pp 401-406. AAAI Press (2011), `https://aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/view/2639`

Paper D Houeland, T.G., Aamodt, A.: An Efficient Hybrid Classification Algorithm – An Example from Palliative Care. In: Corchado E., Kurzyński M., Woźniak M. (eds) Hybrid Artificial Intelligent Systems. HAIS 2011. Lecture Notes in Computer Science, vol 6679, pp. 197–204.. Springer, Berlin, Heidelberg (2011),
https://doi.org/10.1007/978-3-642-21222-2_24

[RQ4] How can a practical metareasoning system automatically adapt to application-specific characteristics?

Paper E Houeland, T.G, Bruland, T., Aamodt, A., Langseth, H.: Extended abstract: Combining CBR and BN using metareasoning. In: A. Kofod-Petersen et al. (Eds.) Eleventh Scandinavian Conference on Artificial Intelligence, pp. 189–190. IOS Press (2011),
https://doi.org/10.3233/978-1-60750-754-3-189
Full paper, with title "A hybrid metareasoning architecture combining case-based reasoning and Bayesian networks" (unpublished)

Paper F Houeland, T.G., Aamodt, A.: A Learning System based on Lazy Metareasoning. Progress in Artificial Intelligence, vol 7 issue 2, pp. 129–146. Springer Berlin Heidelberg (2018),
https://doi.org/10.1007/s13748-017-0138-0

Chapter 2
Background and design principles

This chapter begins with a brief overview of machine learning and introduces an approach for categorizing and evaluating learning methods, which underpins the automatic learning systems described later in this thesis. Section 2.3 introduces lazy and eager learning, two approaches for how to think about learning systems. Section 2.4 discusses learning architectures that incorporate metareasoning to increase their effectiveness, and section 2.5 discusses methods for combining multiple algorithms in reasoning systems. Finally section 2.6 gives an overview of bandit methods, which are used at the metareasoning level throughout this thesis.

2.1 Predicting from examples

There is no assumption-free basis for making predictions - at a minimum one would for example have to assume that the future has some sort of dependency on what happened in the past. This has been known since ancient history, and expressed by for example Pyrrhonian skeptics as the impossibility of establishing a universal rule based on particulars, Hume as the problem of induction for reasoning from the observed to the unobserved, or Wolpert as that estimating generalization accuracy cannot be formally justified without assumptions [51].

On the other hand, there is ample evidence that predictions often work well in practice, and the future tends to depend on the past. For example, the sun keeps rising every day, and the position of objects is related to where they were a second ago. In particular, it has been observed that many interesting problems have a similar tendency relating complexity and generalization error: simpler models that match observed data tend to generalize to unseen data better than more complex models.

For example, Ptolemy preferred simpler hypotheses, Occam's razor is a famous principle for preferring hypotheses with fewer assumptions, and Solomonoff induction assigns a higher a priori weight to computable theories generated from shorter

programs [42].

There are some notable exceptions though - e.g. predicting the next output of a cryptographically secure function. Although not completely random, this type of problem is close to an uninformative scheme where the future doesn't depend on the past, and is generally unsuitable for learning from examples.

Learnability also depends on the *distribution* of examples. The theoretical hypothesis boosting problem fundamentally asks whether the existence of a learning algorithm that is only slightly better than random guessing implies the existence of a learning algorithm that is arbitrarily accurate, and surprisingly this has been shown to be true [29]. What this also means is that unless there exists an arbitrarily strong learner for a given domain, there are sets of adversarial inputs (e.g. problem distributions only containing corner cases) where it is not possible to do better than chance. For example, recognizing handwritten digits can be learned with high precision from uniformly randomly chosen samples of handwriting, but not if the distribution is skewed such that all the test examples are ambiguous edge-cases indistinguishable by human experts (such as only images half-way between a 1 and a 7).

2.1.1 Practical machine learning

Assuming the domain is learnable, a general way to think about the usefulness of a prediction system is to imagine that every action performed by the system can be measured and result in some form of numerical score indicating how useful it was. Reinforcement learning is a very general framework for this scenario, where a learning system is given feedback in the form of reward signals indicating desirable behavior, and is supposed to learn how to behave in order to optimize this reward (including how rewards work - the reward doesn't necessarily correspond to the latest behavior).

If the reward provided corresponds to the usefulness of predictions, this would guide a reinforcement learning agent towards making useful predictions. However, there is often more information available, which can make the reasoning task simpler than in a pure reinforcement learning system. In this thesis we will consider structured problem-solving tasks, where the system provides a solution for each problem, and is scored according to the quality of the solution. In this case there is a clear correspondence between problems, solutions, and scores. We will also provide additional information regarding the desired solutions, typically in the form of providing answers to some of the problems.

There is nothing preventing a pure reinforcement learning agent from eventually learning equivalent information through interaction with the environment, but it can be expected that in practice these types of additional information are highly beneficial and allow the system to learn more easily, faster, and produce more useful results.

2.2 Effective machine learning

This thesis is about automatically learning to predict the unknown. Based on this perspective, we categorize reasoning methods into three categories based on how they learn:

- L_0: Static predictors, which don't change and always behave in the same manner, without any learning. For example traditional expert systems with hand-crafted rules would fall into this category.
- L_1: Methods that build models from domain training data. For example ID3 and many other hand-crafted machine learning algorithms belong to this category. L_1 methods can be viewed as creating and combining L_0 static predictors.
- L_2: Methods that optimize learning based on observed system behavior. For example neural networks trained through weight back-propagation and most boosting algorithms belong to this category. Methods in this category are typically based on using a significant amount of processing power to repeatedly apply iterative optimization steps a large number of times, guided by checking the performance of the system at each step. L_2 methods can be viewed as creating and combining L_1 model-learning methods.

The focus for this thesis is on systems that use L_2 methods to combine problem data, training data, and performance data when making predictions. The learning scenario we use is as follows, from the perspective of the machine learning system:

- Receive a problem instance to predict a label for.
- Respond with a prediction within a given time limit.
- Receive a score based on how good the prediction was and the time taken to produce it.
- Optionally, also receive a solution label for the problem instance.
- Repeat from the beginning for the next problem instance.

Solution labels are not necessarily immediately available before the next problem instance is received, as the labels may arrive later or not be available at all.

The goal here is to have a machine learning system learn during this process, by repeatedly processing the problem instances, scores, and solution labels in a cycle while the system is running. As long as this protocol is followed, a machine learning system is free to decide how and when to train or update models based on data seen so far, what methods to use, to synthesize new features or select feature subsets to use, etc.

We use this scenario for evaluating such learning systems according to their observed performance, based on two high-level dimensions which roughly correspond to the benefit and cost of the system's predictions:

- Prediction accuracy. The machine learning system should make as accurate predictions as possible. We measure this based on accumulated score received over time.
- Computational speed. A machine learning system needs some resources in order to run - for the experiments in this thesis we use a fixed hardware and software platform and measure the time elapsed to make predictions.

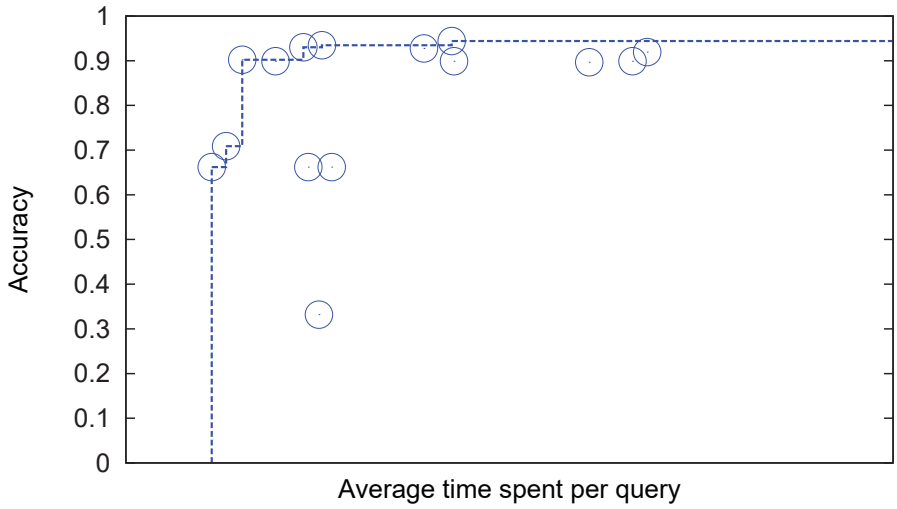

Figure 2.1: Accumulated score vs. computational requirements, with different methods shown as circles. The dotted line in the graph shows the Pareto frontier for these methods, i.e. the highest score that can be achieved for a given cost.

An example of this is shown in figure 2.1, for an example scenario with 15 systems. Each of the points in the graph shows the results for a particular system, based on accumulated score (average accuracy) and computational cost (average milliseconds spent per problem). It is desirable to have as high accuracy as possible, and as short running time as possible, which presents a trade-off. This forms a Pareto frontier, consisting of the most efficient methods that have the highest possible accuracy for any given running time (or equivalently lowest running time for a given accuracy). The other methods are inefficient - either slower or producing less accurate results than some method on the Pareto frontier.

If we had accurate numbers for each method, it would be relatively simple to choose which method to employ based on the options on the Pareto frontier. In practice we will only have noisy estimates of the accuracy and runtime, and the real underlying values will change while the system is running (that is the point of learning, after all).

2.3 Lazy and eager learning

Lazy and eager learning are two different machine learning paradigms, closely related to the trade-off between prediction accuracy and computational speed. Eagerness and laziness in this machine learning sense refers to *when* generalization takes place.

In lazy learning, generalizations are formed in the problem-solving phase, when the specific problem instance to solve is already known. This is the most general paradigm, and allows the most possible information to be used for generalization. The advantage is that the solution can be focused around the specific problem at hand, for example only generalizing based on the local situation and the most similar previous experience. This can be simpler and produce better results than generalizing across the entire domain at once.

However, laziness has a cost. As discussed, the utility of a reasoning system can be expressed as the benefit it brings, minus the costs associated with the system. The "utility problem for lazy learners" [20] occurs when the addition of more information to the system increases the costs more than the benefit it brings, thus reducing utility. In theory this will *always occur* in a fully lazy system that gains more experience and generalizes from all of it, as the additional experience will take more time to process, and this increase in time will eventually outweigh any utility the extra information can bring. In practice, the incremental cost from adding more information is usually not important [48], but learning is typically relatively costly so re-learning from scratch for every problem instance means that the set of viable methods is limited to just very fast ones.

The alternative, eager learning, is the most popular form of machine learning. This paradigm uses an alternative mode of operation by generalizing to train a model of the domain before the problem instance is known. While this means less information is available during generalization, the important advantage is that because it doesn't depend on the input problem, the training phase can be performed just once ahead of time, and reused for all future problem instances. This approach is less sensitive to the training time and allows slower generalization methods to be used, as the cost of training can be amortized over a large number of problem instances. To solve a problem only requires applying the already-trained model, which can be highly optimized and significantly faster than learning from the training data.

2.4 Metareasoning architectures

The existence of different learning paradigms and the abundance of machine learning methods available suggests that there is no single particular learning method that is universally applicable and useful in all cases, but instead a variety of methods with varying performance across different domains. As is often the case in machine learning, "it depends" is usually an appropriate answer when abstractly asking whether a method or approach is good or bad.

However, limiting the problem domains to ones that can be addressed in a practical and efficient manner actually narrows the number of viable methods significantly. When given a concrete task, all methods produce actual measurable and comparable results. And if there is no known way to achieve good results for a given domain with a given approach, then it is currently not practically useful.

Even with this restriction, there is still a multitude of machine learning methods, and no approach that is clearly the best in every situation. Metareasoning provides a way to address this: by reasoning about the learning methods in the context of a specific domain, we can pick the methods that perform well in the specific domain the system is addressing. Metareasoning architectures do this *automatically* - they allow the learning process to be dynamically adapted to the domain.

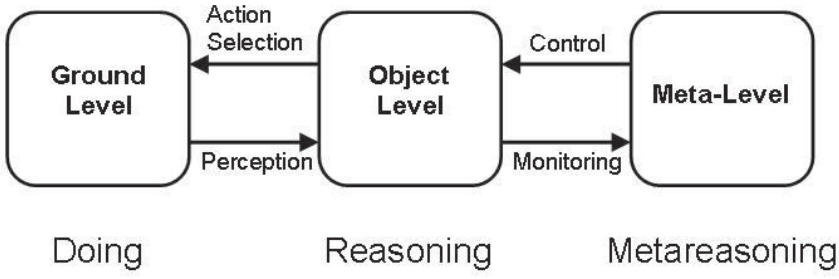

Figure 2.2: Duality in reasoning and acting [15], a high-level framework for describing metareasoning processes.

There is a rich variety of results related to metareasoning from different research fields, with sometimes overlapping and contradictory definitions and approaches. Cox and Raja [15] presented a general high-level framework for analyzing metareasoning systems, which is illustrated in figure 2.2. This framework describes such systems on three levels and the relations between them: the ground level (physical perception and action), the object level (reasoning about action), and the meta-level (reasoning about reasoning).

Another perspective is that metalearning is about choosing the correct learning bias [46] - *some* inductive bias is needed to generalize beyond training examples, and in this view the role of meta-learning is to dynamically shift this bias to achieve better results.

We use both these perspectives to examine three main groups of metareasoning approaches based on how they operate on the meta-level:

- Section 2.4.1: Using meta-knowledge to select an appropriate learning method. A typical example would be to identify a set of important data set characteristics, e.g. training set size and whether features are categorical or continuous, and using rules based on these values to select an algorithm. This form of meta-reasoning determines a bias per dataset based on assumptions about

what is likely or unlikely to work. The bias is set before training begins, and does not change at run-time.

- Section 2.4.2: Identifying and correcting problems [16, 36]. In this case we do not try to determine which methods will perform well ahead of time, but instead notice and react when a method runs into problems: e.g. a generated solution doesn't actually fit within the width and height specified in the problem, or uses too many steps to reach a goal. This form of metareasoning can be seen as changing the bias of the underlying base-reasoner at run-time when a failure pattern is triggered.
- Section 2.4.3: Iteratively optimizing generalization performance [11, 34, 50, 45]. Here we might not know anything about the data set or what process should be followed, but use a meta-level algorithm to learn what methods empirically increase generalization accuracy. This form of meta-reasoning can be seen as continually changing the bias of the underlying base-reasoner at run-time, based on empirical performance. Our thesis work uses this approach, which supports continuous learning and improvement without relying on human experts.

2.4.1 Data set meta-knowledge

This is perhaps the simplest metareasoning approach, based on classifying data sets. The data set metadata (e.g. number of training instances, number of features, categorical vs. numerical features, etc.) is used as input, and the output is a class corresponding to the prediction algorithm and hyperparameters to use for this data set. The metareasoning function from metadata to algorithms can e.g. be learned using a standard object-level multiclass classifier, trained on a collection of previous data sets and the performance the algorithms had on those data sets.

This type of metareasoning is analogous to how a human might analyze a problem domain, e.g. deciding whether a linear learner is applicable, or that a simple model with few parameters should be used to avoid overfitting because very little training data is available. This metareasoning approach can improve the results by initially choosing to use a better method, but would not autonomously continue to increase accuracy at run-time.

2.4.2 Failure-driven learning

A popular direction for using metareasoning in lazy reasoning systems is to use traces of object-level reasoner behavior as the meta-data [16]. These traces list the steps of the process the object-level reasoner performs, and then some form of meta-level control is used to detect and correct problems.

The most common form of such meta-level control is driven by *reasoning failures*, where an unexpected result is encountered while solving a problem. This approach is based on the intuition that there is no reason to change the system if it is performing as expected without any problems. A reasoning failure can e.g. be arriving at an incorrect answer, producing an answer without evaluating the

previous case that is most similar to that answer, or even lacking any method to solve a certain kind of problem.

Adding this type of introspective error detection and triggering of repair actions can improve the performance of an underlying reasoning system automatically, even enabling the system to correctly solve problem instances that the underlying reasoner could not solve on its own [19].

However, detecting failure, determining the reasons for it, and then correcting them is a difficult process and a research problem in its own right. For a complicated system, what constitutes a reasoning failure is not a simple well-defined yes/no problem, and in practical systems will often be pre-determined by human experts. For example, the implementation of Gila used failure patterns provided by the author of the system, and investigating whether the patterns could be learnt from experience was left as a possible line of future research [36].

2.4.3 Optimizing generalization performance

The core idea behind this metareasoning approach is to evaluate multiple different potential methods, and choose or prioritize the one(s) that perform the best.

The processes used by humans when experimenting with different machine learning methods can often be seen as a variation of this approach, e.g. selecting the best-forming approach based on performance on a test set, or using cross-validation to select the best parameters.

In the next sections we expand on this metareasoning approach, and specifically on meta-level methods that explicitly reason over multiple specific object-level learning methods.

2.5 Multiple classifier systems

The idea of systematically combining multiple predictors to produce more accurate composite predictions can be traced back at least to Laplace in the early 1800s, and has been highly successful in practice in various fields [47, 13, 27].

Our focus is on methods for automatically learning higher-level combiners, which we will refer to as *multiple classifier systems*, a generic term not connected to any particular method. However there is significant overlap in terminology for these types of systems from related fields:

- Cross-validation [31]: A procedure for training multiple instances of the same method on different training and evaluation sets. This can be used to estimate the accuracy for unknown data, and is often used to choose which learning method to use or which parameter settings to use.

- Hybrid reasoning [6]: A generic term referring to combining learning methods from different subfields. This includes the idea of combining heterogeneous methods, which is often observed to perform better than homogeneous combinations.

- Hyperparameter optimization [18, 25, 5, 2, 44, 22]: Choosing the best parameters for a learning algorithm, with a focus on convergence speed and not getting stuck in local optimums.
- Bayesian optimization [28, 40]: A meta-level method for hyperparameter optimization, based on Bayesian probability and sequential iterative improvements. This approach is focused on minimizing the number of evaluations of the underlying function.
- Stacking [50]: Wolpert argued for learning multiple levels of predictors, called "stacking", which in its most general form covers any form of combining multiple levels of generalizers. However, as most commonly used the term stacking refers to training a meta-level model, usually a regression model [7], based on the outputs of object-level predictors.
- Ensemble learning [41]: Generating and combining multiple hypotheses using the same base learner. These types of methods typically have a strong connection to the statistical machine learning community, and a similarity to statistical ensembles.
- Portfolio selection [34]: Optimizing wealth across multiple assets, with a focus on the trade-off between expected return and risk.
- Prediction with expert advice [11]: Minimizing regret compared to multiple reference forecasters, with a focus on worst-case performance (adversarial environments).
- Deep neural networks and neural architecture search [23, 14, 49, 17, 52, 35]: Deep neural networks can be viewed as layers of simple generalizers on top of each other, that are all tuned to work well together. This approach has been highly successful on visual and image-recognition tasks, based on adding repeated structure to the generalization process. In addition to automatic feature engineering as part of training these networks, in recent years the network structure has also started to be learned automatically, enabling state-of-the-art performance without task-specific human expert tuning.
- Bayes optimal classifier: This classifier is a theoretical construct for obtaining the best possible prediction which requires additional statistical knowledge that is not available for most real-world data sets. AIXI [26] is a related architecture that assigns priors to each hypothesis based on their complexity, inspired by Occam's razor. AIXI is still not computable in practice, but approximations using restricted hypothesis classes can be implemented [45].

The purpose of metareasoning in this thesis is to achieve automatic machine learning, often referred to as "AutoML" (which is similarly also not clearly defined, but generally refers to automatic machine learning without task-specific tuning or design by human experts). As a topic and research interest, AutoML is most closely associated with deep neural networks and neural architecture search, which typically involves specialized hardware and large amounts of eager pre-computation.

In contrast, the approach in this thesis is focused on incremental lazy learning and highly-efficient methods that can run quickly on consumer hardware, and methodologically is most closely related to minimizing regret for prediction with expert advice (though without human experts).

2.6 Bandit algorithms

As stated earlier, in reinforcement learning an agent attempts to maximize its rewards from an environment that is not fully known. This creates two opposing and contradictory goals: the agent should spend time performing the actions that it expects will provide the greatest reward, but at the same time it should try to learn more about the environment to make sure that the agent's expectations are correct and it is performing the correct actions. If new information from the environment shows that another action would be more beneficial, the agent should adapt its behavior. This is called the exploration-exploitation trade-off, as an agent is typically not able to do both at the same time, and has to make a choice between them. (E.g. to learn the effect of choosing option X over option Y, you actually have to choose option X. If the agent is expecting option Y to perform better, this exploration of option X incurs an expected loss, or regret.)

A famous problem exemplifying the exploration-exploitation trade-off is the multi-armed bandit problem. In this problem a gambler can choose between many different slot machines in a casino, where each machine has its own probability distribution for what rewards it will produce. The problem is for a gambler to choose which machines to play. In the traditional setting the gambler has no initial knowledge of the machines, and can only learn about their reward distribution by attempting to play them, which means not playing one of the other - possibly better - machines. Solutions to this problem can be used to address many real-world situations, from clinical trials [43] to internet advertising [12], where choosing between multiple options with uncertain results can be modeled as choosing which slot machine to play.

We will model the problem of selecting a learner in a multiple classifier system as a bandit problem, with the different learning methods corresponding to different slot machines.

2.6.1 Multi-armed bandit policies

A policy or strategy for the multi-armed bandit problem determines which slot machine to play next, based on the current knowledge about the machines. These choices are made incrementally for each new play in a lazy manner, and the strength of different strategies can be compared based on the results they produce. In particular we can measure the regret r that a policy produces after choosing a machine to play T times. The regret is defined as the difference between the rewards collected by a policy and the maximum possible rewards that could have been achieved by knowing which machine is the best and always playing that one.

A policy where the fraction r/T tends to 0 when the number of times T tends to infinity is said to be *zero-regret*, and is asymptotically optimal. This is a theoretically desirable property, but can be difficult to prove, as *zero-regret* only describes the theoretical behavior at the limit. There are simpler strategies that are not known to be asymptotically optimal (or even known not to be optimal) but still often produce better results for finite T when tuned appropriately [32].

2.6.2 Epsilon-based policies

A simple policy called *epsilon-greedy* plays the estimated best machine for exploitation most of the time, but with a low probability ϵ (epsilon) will instead play a random machine for exploration. This is one of the simplest exploration-exploitation trade-off policies, with decent performance in practice if the ϵ value is tuned well for the specific task to be solved.

When the total number of trials T_{max} available is known ahead of time, a simple modification of this strategy can be made, called *epsilon-first*: We know (on average) how many times the strategy will perform exploration: $T_{max} \times \epsilon$. The strategy can be improved by performing all this exploration up-front, i.e. starting with $T_{max} \times \epsilon$ random plays, and then continually playing the estimated-best machine afterwards. By reorganizing the steps to place all the exploration steps at the beginning, all the information learned from exploration will be available to every exploitation step.

Another approach using ϵ random machine plays is *epsilon-decreasing*, where ϵ is not treated as a constant but instead varies over time. In the beginning, ϵ is high to perform a large degree of exploration, similar to *epsilon-first*. Then over time ϵ is lowered to focus comparatively more on exploitation when more information about the alternatives has already been obtained.

2.6.3 UCB and UCT policies

The Upper Confidence Bound (UCB) policy [3] achieves the same gradual shift from exploration to exploitation as *epsilon-decreasing*, but using a different underlying approach. Instead of trying to tune the right amount of random exploration to perform, UCB uses statistical confidence bounds to determine which machine to play each time, which incorporates exploitation and exploration in the same formula. The formula used is always the same at every step and evaluated for every machine, and the machine with the highest computed value is played next.

The formula used for UCB evaluation is split into two parts that are summed together: the first part is the expected exploitation value, and the second part is an optimistic estimate of the exploration value. The exploitation value is estimated as the average value so far, while the exploration value is modified each time a machine is played: the exploration value increases for a machine that does not get played, and decreases for a machine that gets played. This happens quickly in the beginning, and then slows down when the machine has been played more and tighter confidence bounds can be determined. The exploration value is always increasing without bound for a machine that does not get played, which means

that every machine will eventually get explored again, no matter how poorly it has performed so far. However this will take longer and longer each time if the machine continues to produce poor rewards, and a higher and higher fraction of the plays will be spent exploiting the better-rewarding machines.

In practical experiments, an optimally tuned ϵ can often perform better than UCB, especially if there are only a small number of machines and they all have approximately similar expected rewards [32]. However, the optimal value for ϵ varies significantly depending on the exact machine setup and the total number of trials, and badly-tuned ϵ values perform poorly. In comparison UCB typically always performs reasonably well.

The Upper Confidence Bounds applied to Trees (UCT) algorithm [30] is a generalization of UCB, which can be applied to general tree structures. The regular flat row of slot machines can be considered as a simple tree with all the machines as direct leaf nodes under the root, and in this scenario the UCT algorithm behaves the same as the UCB algorithm. However, UCT also supports multi-level tree structures with internal nodes - the UCB formula is used when deciding between child nodes, an internal non-leaf node is chosen, and then the UCT algorithm is applied again recursively to choose between that node's children. To make this work, the total number of plays as used in the UCB formula is adapted to instead be the total number of plays for the parent node (giving the same result as UCB for a tree with only one internal root node and all bandit machines as direct child leaf nodes).

The UCT algorithm is particularly well-known for computer Go, where the use of Monte Carlo tree-based exploration techniques ushered in a stronger generation of Go-playing programs than the previous state-of-the-art expert systems [21, 10], and was later used by the first computer player to beat professional human Go players [39].

Chapter 3
Research summary

3.1 Research process overview

The primary topic of my thesis research was on systems that reason about themselves and learn, within the context of case-based and model-based reasoning in our research group. Based on initially examining relevant literature in the CBR and machine learning fields, my focus narrowed towards meta-level learning in introspective reasoning systems, and a specific approach to how these metareasoning tasks should be performed.

3.1.1 Main research phases

Phase I of this research was to study relevant literature and determine which elements were most important for learning. This started with the CBR cycle, and initially looking at the Retain step as the focus of learning since that is the part that changes the long-term state of the system. However, based on existing literature and experience gained while investigating and experimenting, it became clear that the goals of the Retain step should not be optimized in isolation, but instead be optimized based on how retained cases will be used by the Retrieve step in the following cycle. The Retrieve step should in turn be optimized based on how retrieved cases will be used by the Reuse step, and so on. The overall conclusion of this first phase was that independently optimizing just one part of the CBR cycle is insufficient and fundamentally unable to take full advantage of the possible opportunities for learning, as what matters is the combined overall system performance, i.e. how a set of methods work together.

Phase II of this research was to investigate how to improve the overall performance of a system based on combining lazy and eager learning, in an attempt to achieve the combined benefits of both. This was informed by the background from phase one, with the goal being to improve the empirical results from an overall system instead of individual components. The main takeaway from this work was that the empirical results and evaluations should be for each specific reasoning scenario, with a specific domain and set of requirements, and in particular including resource

limitations which are the main reason to pursue eager learning methods. This research phase was conducted together with the TLCPC (Transactional Research in Lung Cancer and Palliative Care, Norwegian Research Foundation, contract no. NFR-183362) project in my research group. My role in this work consisted of investigating learning approaches that could be used in computer-assisted decision systems. The concrete outcome of this collaboration was that as I developed new methods they could be applied to and evaluated on a data set of cancer patient treatment, and resulted in co-authoring a paper with the other TLCPC participants about high-level approaches for combining different learning approaches.

Phase III of this research was to propose a new approach to metareasoning based on lazily optimizing and evaluating overall system performance: ALMA. Implementing and testing ALMA had originally been intended as an item for future work and not as a part of the thesis. To additionally include an empirical evaluation of ALMA as part of the thesis required significant implementation work, and at this point I was no longer working full-time on my research. I eventually completed this phase after implementing 3 different generations of ALMA, with the final version able to integrate with the WEKA reasoning system and work as a separate metareasoning layer on top of it, which could then be used to evaluate many different combinations of algorithms and data sets in a uniform manner.

3.2 Phase I: Learning in case-based reasoning

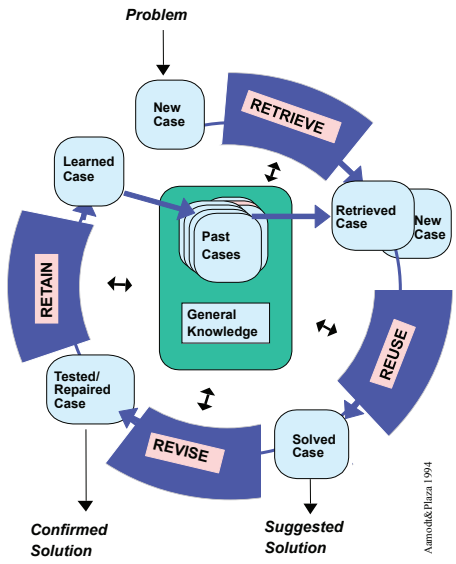

Figure 3.1: The case-based reasoning (CBR) cycle [1], illustrating the 4 major "Re-" steps and how they interact with the case base.

The first phase of research was focused on learning within case-based reasoning (CBR) systems, a type of lazy learning system that is a major focus in our research group. This started by examining how the case base is updated, which is how learning traditionally occurs in CBR systems. As shown in figure 3.1, problem-solving in CBR occurs in a cycle, which proceeds through a number of steps and then repeats from the beginning for the next problem query, after updating the case base based on the first problem-solving experience. The exact steps in the cycle can vary between implementations as there have been a number of additional steps and alternatives proposed in the literature, but the basic and most common cycle consists of four steps:

- Retrieve. One or more cases similar to the problem query are retrieved from the case base.
- Reuse. The retrieved case(s) are processed and used to determine an answer for the problem query.
- Revise. The solution is evaluated or tested in some form, and repaired if needed.
- Retain. The case base is updated by adding the experience from this problem-solving attempt to the case base.

A basic CBR system thus consists of a set of methods for these 4 "Re-" steps, including a representation format for storing past cases and general knowledge. Learning occurs in the system at the end of each problem-solving cycle, based on the method used in the Retain step to update the case base. Updating the case base in this way after every problem-solving attempt is what allows a CBR system to support sustained learning, which is an important feature of CBR systems. Maintaining a growing case base by e.g. generalizing or deleting cases is in itself an interesting research topic [37], but has so far not received as much attention as the Retrieve step.

However, CBR systems also contain additional knowledge beyond the case base, such as the case vocabulary, similarity knowledge, and adaptation knowledge [38]. This is often implicitly encoded in the methods used for the different "Re" steps, and not changed after the system has been developed. For example, similarity knowledge is used during case retrieval, even if the similarity knowledge is never represented explicitly and only encoded as part of an algorithm implementation.

Learning these other knowledge types is an important opportunity to improve the system, but it is complicated by the fact that how good a particular method is depends on what other methods are used in the other steps of the CBR cycle. For example, to be most useful the Retrieve step should not retrieve the cases from the case base that are most "similar" in some purely abstract sense, but more specifically cases that will be useful for the method used in the Reuse step to determine an answer to the problem query. Similarly, the Retain step should not be optimized in isolation based on e.g. measures of case base size and coverage, but instead based on how it affects the rest of the system, starting with how it

affects the method used in the Retrieve step that accesses the case base. In fact "optimizing" e.g. the Retain step to score higher on a local Retain-specific metric can hurt the overall performance of the system. This means the best method to use depends on both the problem domain and also all the other parts of the system. Which methods perform best can also potentially change after a CBR system has been deployed, due to e.g. changing characteristics from a much larger case base accumulated over time, or due to unforeseen shifts in the kind of problems encountered.

Based on these results from examining learning in CBR, the research focus changed from learning new domain knowledge in the Retain step to instead learn multiple forms of system-level knowledge, and using this system-level knowledge to determine full sets of methods that work well together to improve the overall performance of the system as a whole. This also led to examining other forms of learning, as the CBR cycle is not as well suited as a framework for examining whole-system optimization.

3.2.1 Paper A: An Introspective Component-Based Approach for Meta-Level Reasoning in Clinical Decision Support Systems

This paper introduced an initial framework for creating reasoning systems with a meta-level control component. All the different methods in the system are represented as components, which can be used directly or combined to form larger and more powerful components. The role of the meta-level control component is to decide which components should be used to create the system, by assigning individual or combined components to each of the reasoning tasks required.

Component combinations in this case were based around task templates created by a human expert. The primary example investigated was a CBR template (described in the paper), where the tasks were exactly the 4 "Re" steps and sub-tasks associated with the 4 steps. The meta-level control component's job was to select the set of methods that would be used for these sub-tasks. The components for this system were self-describing using a vocabulary introduced in the paper, with specified inputs, outputs, pre-conditions and post-conditions that described when each component could be used and what the result would be.

The meta-level control component used this metadata to reason about which components could be concatenated together, and which components could be used to perform tasks. The constraints on task assignments were specified as conditions using the same vocabulary as the components, and based on problem domain characteristics such as the size of the data set and the type of input data.

An initial version of this paper was discussed in the health workshop at ICCBR 2009 [24].

RQ1: This research provided a partial answer to research question 1: *"How can learners perform better using introspective capabilities?"*:
Introspective capabilities can be used to choose between alternatives at the system level, i.e. not just learning domain-specific knowledge but additional system-level knowledge types such as similarity knowledge and adaptation knowledge.

The paper presented this framework in the context of clinical decision support systems, with a goal of using these ideas as part of the TLCPC project in Phase II. However the framework itself was more general, and could also be used for other machine learning tasks.

An example implementation of this system architecture was implemented that verified the constraints at compile-time when creating the system, and automatically determined the right component that fits the requirements. However, human expert input was required to choose between multiple valid component choices, which in particular meant that the system's expressiveness was limited regarding what components could be combined automatically. For example, if the Retain sub-tasks were not sufficiently constrained and there were 2 or more valid component assignments, a human expert would have to manually provide additional constraints to make the selection unambiguous. For small systems it was also possible for a meta-level component to explore all combinations of components, although in general that would lead to an exponential number of attempts required.

This implementation worked overall for a simple set of example components, but was not fully satisfactory as it required significant human expert annotations both for the methods used in the system and also for each new problem domain to be addressed. The same high-level framework was reused later in Phase III, but with a new metareasoning component that addressed this shortcoming by fully automatically exploring the space of valid component assignments.

3.2.2 Paper B: The Utility Problem for Lazy Learners - Towards a Non-eager Approach

This paper examined how to evaluate the usefulness of reasoning systems, based on a new General System Utility measure that combined the accuracy of generated solutions, the time it takes the system to solve a problem, the usability of the system for human operators, and the costs associated with developing, running and maintaining the system.

The focus of the paper is on lazy learners, particularly advocating for retaining more laziness when creating new systems instead of eagerly optimizing parts of the system to achieve lower resource usage. This was based on two main reasons:

- From a perspective of leaving more possibilities open for a metareasoner to take advantage of, it is generally undesirable to eagerly commit to decisions that are not certain to be correct and remain correct forever (i.e., most decisions).

- Based on evaluating the full overall usefulness of the system, the positive impact of these kinds of eager optimizations are usually small. There is a significant chance of a more significant negative impact, the simplest example being that implementing them increases development costs which are typically much larger than hardware costs for CBR systems.

RQ2: This research provided a partial answer to research question 2: *"How can the performance of reasoning systems with different capabilities be evaluated?"*: Reasoning systems can be compared based on the overall empirical usefulness they provide when used in specific problem-solving scenarios. This can vary greatly based on both the problem domain and what requirements the system has to run under. Generally, it can be expressed as the benefit the system brings when run, minus the costs associated with the system.

In the paper, several different forms of systems are used to illustrate different trade-offs, with the main point being that the correct decision can vary greatly based on different algorithm choices. This was in contrast to various pieces of advice and rules-of-thumb presented in other papers, which did not consistently replicate when attempting to implement them for the metareasoning system. In particular the paper illustrated realistic scenarios where eagerly limiting the case base size, eagerly creating case index structures, and eagerly applying complex case base maintenance methods were all harmful to overall system performance, even without considering development costs.

The paper also highlighted several methods from the literature that combine lazy and eager learning, as examples of better ways to speed up lazy learning. This approach is revisited in Phase II, where new methods were developed that combined lazy and eager learning to produce results very quickly.

3.3 Phase II: Performance/cost trade-off in specific scenarios

The next phase of research examined and experimented with various methods and approaches to improve results on a data set related to TLCPC (Translational Research in Lung Cancer and Palliative Care), a project in our research group aimed at improving palliative care for lung cancer patients. TLCPC's goals included examining the role of computer-assisted decision systems by developing a new prototype decision support system. The TLCPC project was closely connected to EPCRC, a larger EU project with a broader scope covering all types of cancer and many other research goals, which involved medical research teams from multiple countries.

Part of this work included examining a palliative care dataset consisting of 55 features including self-reported pain, medical doses administered, and other measurements. The data was collected for 1486 patients, with feature values reported for the first 3 weeks of treatment for each patient. To use this data set for investigating learning algorithms, a concrete reasoning task was devised which consisted

of predicting the self-reported pain in the 3rd week based on the observed data in the first 2 weeks. The idea was that if there were a model that could predict pain in this manner, a decision support system could use the observed values from the first 2 weeks for a patient and display the predicted 3rd week pain for different possible dosages, helping the clinician decide how much medicine should be administered. The intention here was to identify a potentially valuable example reasoning task that could then be used to guide methodological machine learning research, not to actually develop a system that would be used by clinicians.

This reasoning task was used to investigate a new approach for using eager learning to speed up lazy learning CBR systems. This approach was inspired by the results from Phase I, specifically to avoid destructive eager case base maintenance. As the beneficial properties of case base maintenance can always be achieved in a more lazy manner by performing them in the next cycle's Retrieve step instead (when the next query is already known), the main theoretical benefit of performing maintenance ahead of time is to reduce the computational cost and time of the Retrieve step. To address this, a new and highly efficient RDT (Random Decision Tree) similarity measure was developed, allowing larger case bases to be used while still performing a full match against all previously seen cases. The basic idea was to eagerly prepopulate a caching data structure that allowed smaller problem characterizations to be compared instead of the full cases. The problem characterization was automatically created to approximate a normal similarity computation, compared to traditional systems where this is usually a manual process.

In experiments, the RDT method was significantly faster and achieved better results than a full similarity comparison using a manually created problem characterization based on expert domain knowledge about the most relevant features specifically for pain classification. The best (but slowest) results were achieved with a hybrid approach combining both the RDT method and the expert knowledge, by using RDT to determine "overall similarity" between cases, but only considering those cases with the most similar expert problem characterizations. These better results from the hybrid method were at a cost of ten times slower execution, unfortunately losing one of the main benefits of the RDT method in the first place and highlighting the importance of considering computational costs.

These experimental results reinforced the previous results from Phase I, that the usefulness of different methods is highly dependent on the specific problem-solving scenario they're used in, and therefore providing good results in all scenarios is likely to require a diverse set of methods with different advantages and drawbacks. Based on this the research focus returned to the metareasoner idea, with a goal to make it fully automatic without the need for human expert analysis of methods and domain for each new situation.

3.3.1 Paper C: An Efficient Random Decision Tree Algorithm for Case-Based Reasoning Systems

This paper describes a new highly efficient random decision tree (RDT) similarity algorithm for CBR. The method is based on initially creating a set of random

decision trees, and computing similarity between two cases based on how many of the trees classify them in the same way. This is implemented with an eagerly computed data structure that stores a compact signature for each case, which can be used to very efficiently compute the similarity between two cases from just their signatures. New cases can be incrementally added to the data structure by computing and storing their signature.

This similarity measure was inspired by *kd*-trees [4] used in case-based reasoning systems, and the Random Forest classifier which creates a similar set of trees (but constructed differently) and can also compute similarity based on the tree classifications. The main advantage of the RDT method is that it can be implemented very efficiently, as the trees can be constructed before observing the full data set and new cases can be incrementally added one by one.

The RDT algorithm is compared to and combined with an expert-defined similarity measure on the palliative care dataset (computed lazily by comparing the most relevant case attributes), as well as a baseline uninformative random similarity measure. Experimentally, the RDT algorithm resulted in better predictions than the expert-defined similarity measure, and was significantly faster. The best (but slowest) predictions came from a hybrid combination using both the RDT algorithm and expert-defined similarity.

In this paper, the usefulness of different algorithms was compared not just based on their internal parameters, but also based on the externally observable prediction accuracy (as the benefit) vs. the computational run-time required (cost). This is a refinement and specialization of the usefulness evaluation described in Paper B. Together these two dimensions provide a solid basis for estimating usefulness in a concrete problem-solving situation, for a specific domain and running on specific available software and hardware.

RQ3: This research provided a partial answer to research question 3: *"How can a lazy metareasoning architecture take advantage of introspective capabilities?"*:
Reasoning methods have different performance curves, which can be empirically measured in an algorithm-agnostic way based on their predictive accuracy and run-time cost in a specific reasoning scenario. Through introspection, a metareasoning system can access and react to this performance meta-data, and use it to optimize the choice of method based on how well it is performing.

3.3.2 Paper D: An Efficient Hybrid Classification Algorithm - an Example from Palliative Care

This paper describes a procedure to directly build a classification method based on RDT instead of a similarity measure as in Paper C. A benefit of addressing classification directly is that predicting patient pain is closer to what would be useful in a palliative care application, instead of only detecting similar patients.

In this paper, RDT was used to compute a weighted average of pain values instead of as a similarity measure. This weighted average performed better than a

simple unweighted average, but not as well as the expert-defined similarity measure (with an appropriately tuned k value). A combination of RDT and the expert-defined similarity performed best, as in paper C. However the results from the two papers are not directly comparable, as the RDT algorithm was used in a different way and the generalization error was computed differently.

Only the most representative results were presented in paper C and D, but at this point there were a number of possible options for how to use RDT and how to combine it with other algorithms. Particularly the results in paper D were achieved through manual trial and error to find what worked well specifically for the palliative care domain. The selected combination even for this relatively small task relied on human expert knowledge, and would not necessarily be appropriate for tasks in other domains.

RQ3: This research provided a partial answer to research question 3: *"How can a lazy metareasoning architecture take advantage of introspective capabilities?"*:
There are a large number of possible reasoning methods, meta-algorithms to combine them, and possible hyperparameter settings that affect the performance. Which methods perform best depends on the domain and the task to be solved.
A metareasoning system can use performance meta-data to determine which methods should be used, allowing optimization for specific domains without requiring a human machine learning expert to be involved.

3.3.3 Paper E: A hybrid metareasoning architecture combining case-based reasoning and Bayesian networks

This paper was a collaboration in the TLCPC research group to summarize the current state of our research. The focus was on two different types of uncertainty that are present in machine learning tasks, and how methods with different strengths and weaknesses can potentially be combined in a hybrid system to address both types of uncertainty.

The first type of uncertainty is aleatoric uncertainty, which refers to events having a certain probability of happening given the right conditions. I.e. this refers to the stochastic nature within the domain itself, and the relationships between domain variables not being deterministic predictors in the actual observed data. As a simple example, the height of humans in the world varies, and this height probability distribution also varies based on other characteristics, e.g. age and sex. The proposal in the paper was that Bayesian networks provide a solid foundation for representing and reasoning about aleatoric uncertainty, using a directed acyclic graph to express conditional probabilities and a sound inference engine to update beliefs when presented with new evidence.

The second type of uncertainty is epistemic uncertainty, which refers to an incomplete understanding of the domain itself, e.g. incorrect beliefs about a causal relationship. This uncertainty is not an intrinsic part of the nature of the domain,

but instead reflects incomplete human knowledge. An example of this is leaving out important characteristics or misunderstanding their effect, e.g. modeling the human height distribution without taking age into account. A Bayesian network would still be able to "correctly" model the relationship, in that it could reflect the statistical relationships between variables that are included in the model. But it would not be a satisfactory explanation of human height variation. The proposal in the paper was that CBR using local models is well-suited to handle situations with high epistemic uncertainty. In contrast to building out a full detailed model of the domain, CBR methods can be more easily used to capture individual pieces of expert knowledge that can be reused if similar problems re-occur in the future.

The paper outlined 4 sequential combinations for integrating BN and CBR, based on an earlier paper [9]. 3 of these combinations involve using a first method to process the original input and generate a new modified input set that is used as input for the second method. The 4th combination involved using an initial CBR method to select between multiple different BN models, and applying the BN model that was picked to the problem.

Finally, the paper outlined a metareasoning approach to combining BN and CBR, inspired by ideas from Phase I of this thesis research work. The idea was that for a new domain, there would initially be a high degree of epistemic uncertainty. At the beginning, a selection of hand-crafted cases could be created by domain experts, that represent a set of prototypical problems and solutions. Using these cases with a CBR method, the system could provide acceptable answers for new problems that were close enough to these existing prototypical cases. Then over time, as more and more problem cases were collected and answered, the epistemic uncertainty about the domain would decrease, as the new cases provide more information about the domain that can be learned from and used to construct better models. A separate BN model would be trained based on these cases, providing a more accurate model of the domain and better solutions over time as the training set becomes larger.

The expected behavior was for the CBR method to provide a constant base level performance, while the BN model would perform significantly worse in the beginning, but eventually improve and continue to get better and better. By empirically evaluating the answers provided by the two approaches, a metareasoning control agent could start by using the prediction from the CBR method initially, detect when the answers from the BN model surpassed those produced by the CBR method, and from then on use the predictions from the BN model.

RQ4: This research provided a partial answer to research question 4: *"How can a practical metareasoning system automatically adapt to application-specific characteristics?"*:
The metareasoning system can evaluate the performance of different methods for the specific application, and prefer the method with the best empirical performance.

3.4 Phase III: Autonomous self-optimizing learning systems

The last phase of this research included creating ALMA, a lazy metalearning architecture based on the research experience from the previous phases. The overall structure is based on the component-based meta-level reasoning concept from Phase I, but with a new fully automatic metareasoner that doesn't require human expert annotations for the different methods or the domain. This metareasoner is based on the results from Phase II, and chooses which components to use entirely based on empirical results that were previously observed for the same specific reasoning scenario where the system is used.

Creating this metareasoner component started by first investigating how metareasoning was being used in CBR systems. The most popular approach was identified as adding a new meta-level control agent that monitors what is happening in the system, detects when something is not working as desired, and then initiates corrective actions to rectify the situation. In practice there's usually a small set of detection and correction methods that have been explicitly implemented by the system developers, and each of the methods only applies to one targeted part of each individual problem-solving experience. This general approach is implemented in different ways, e.g. based on examining collected traces that record the reasoning steps taken when solving a problem, or by detecting that there's no viable method available that can make further progress for a given problem-solving attempt.

What these popular CBR metareasoning approaches share is that they're focused on using metareasoning to fix problems that occur while solving problems, i.e. they could also be described as meta problem-solving. While a meta problem-solving component would be valuable in ALMA, at this stage of the research the goal was instead to apply metareasoning to the learning methods, i.e. metalearning in the sense of *learning to learn*.

For a reasoning system to learn from a problem-solving experience necessarily means that the system has to change in some way, to be different than how the system was before the experience. To more clearly define the metalearning problem and separate it from metareasoning that doesn't involve learning, a new framework for describing the changes that occur in a reasoning system was created. The framework identifies 3 different levels of changes that are important for a metalearning system (as described in section 2.2):

L_0: Static non-learning reasoning methods (no changes).

L_1: Learning algorithms (change based on domain data).

L_2: Metalearning algorithms (change based on system performance data).

Using this classification, even some advanced meta problem-solving would only be L_0 if it doesn't result in learning anything that persists after the specific problem-solving experience, while very simple and basic tuning based on validation set performance would be L_2. In this way L_2 is decidedly different from other terms

such as "metareasoning" or "intelligence", but it expresses exactly the type of learning methods needed for the desired metalearning component: introspective methods that learn and change based on how the system is performing, not just from domain training data.

L_1 methods can also be described as methods that have a static bias, i.e. they will learn the same model for a given data set. L_2 methods on the other hand have a dynamic bias, and can adapt how a model is formed from training data based on how well it performs. These terms do have substantial overlap, but traditionally the exact meaning of "static bias" and "dynamic bias" and the difference between them has not been as clearly defined, while reasoning methods can be precisely classified as L_0, L_1, or L_2.

These concepts do not just apply to case-based reasoning, and in fact such L_2 learning should ideally be able to choose between CBR methods or other machine learning methods based on how suitable they are for any given situation. How suitable a method is can here be seen as how well its inductive bias matches the unknown underlying structure in the domain data. All machine learning methods need *some* bias to generalize from observed to unobserved data, and the difference in predictive accuracy between different methods when trained on the same data comes from differences in inductive bias.

To try to gain new knowledge of learning algorithms and understand their inductive bias, researchers often investigate and make broad claims, for example that a certain algorithm is well-suited or poorly-suited to deal with sparse data. One approach to finding a suitable machine learning method is to characterize a large number of methods in this way, e.g. how they work for sparse data, real-valued data, a large number of attributes, a large data set, etc. Then to select a method for a new domain, these characteristics are determined for the available domain data, and a learning method is chosen that matches these characteristics and therefore is expected to work well for the domain (as described in section 2.4.1). This was the approach planned for the metareasoning component in Phase I, using a vocabulary of such characteristics to describe when each component could be used.

However, for a fully automatic reasoning system there are two large drawbacks to using these types of characterizations to make decisions: First, generating these characterizations requires a large manual effort by human experts, which limits the usefulness in new scenarios. Second, these characterizations are typically very general, outlining what to expect in broad strokes. But the actual generalizations and predictions made by a method depend on minute details of its inductive bias, e.g. even just different random initializations or training for a longer time will result in differences in exactly what the resulting inductive bias is. While general characterizations can aid understanding, to get the best performance for a specific domain human experts usually have to try several different variations and iteratively tune parameters based on empirical results.

In ALMA, metareasoning is based on this latter approach: iteratively trying and tuning different variations to optimize observed empirical results (as described in section 2.4.3). The overall component-based architecture from Phase I is used,

together with a new metareasoning component that optimizes performance according to the predictive accuracy and run-time cost from Phase II. This combination can perform learning at the metalevel without any special human annotations for the methods nor the domain, and achieves a lazier form of metareasoning where the system continuously adapts at runtime.

Reasoning in ALMA is not limited to CBR methods, and the overall goal for ALMA is to only choose to use CBR or any other methods when they demonstrably work well for a given scenario. Using more diverse types of reasoning methods and data sets was also desirable to better evaluate ALMA's performance, and necessary in practice to have enough methods and dataset combinations available for a meaningful empirical evaluation of the research.

The focus in ALMA is on the L_2 metareasoning components, and in particular the method used to select which components to use. To avoid having to examine a combinatorial explosion of possibilities, the selection method is based on the UCT algorithm for balancing exploration and exploitation. Trying a new component to gain more knowledge about it is modeled as "exploration", and using the presumed best component combination found so far as "exploitation". The algorithms used in ALMA's L_2 components are more closely related to ensemble learning systems and hyperparameter optimization than case-based reasoning, while the overall architecture and learning paradigm is based on lazy learning.

3.4.1 Paper F: A Learning System based on Lazy Metareasoning

This paper presents ALMA, the autonomous metalearning architecture developed in Phase III. The goal of ALMA is to be able to automatically perform domain-specific optimizations with no additional pre-specified task metadata or prior knowledge of the domain. ALMA is focused on cost-effective prioritization and can run in resource-constrained scenarios. This means that ALMA can be used to solve a novel data set while running on a regular personal computer, without any special hardware, manual tuning, or expert knowledge.

The paper introduces the lazy online learning paradigm used in ALMA, the L_0, L_1, and L_2 layers, and how components are chosen based on a node hierarchy. For each new problem query, ALMA creates a learning system to address it, and uses this learning system and the previously seen data to predict an answer for the problem query. This approach is compared and contrasted with other metareasoning systems and related methods and algorithms from other artificial intelligence sub-fields. Three novel L_2 components are introduced in the paper:

- The UCT-based child node selection method used to choose between components based on empirical results.
- A caching component for reusing previously trained models to efficiently approximate the output of a newly-trained model.
- An example parameter-tuning component that is suitable for selecting a small integer parameter, e.g. the k parameter for k-NN.

The node selection method is the core metareasoning component in ALMA, which is used to create the per-problem learning systems by choosing components based on their previous performance. The node selection method learns at the meta-level based on the observed empirical system performance when using each component, while individual learning methods learn from domain data as more problem queries and solutions become available. The caching component is required to achieve acceptable performance when using eager learning methods that are slow to train, as otherwise they would need to retrain the model for every single problem query which can be very slow. The parameter-tuning component is not used in the comparison experiments, to ensure the results are directly comparable by always using the default method parameters as specified in WEKA.

In experiments using ALMA with the reasoning methods available in WEKA as components, ALMA performed better overall than any individual reasoning method and better than the existing meta-level multi-reasoner methods in WEKA.

RQ4: This research provided a partial answer to research question 4: *"How can a practical metareasoning system automatically adapt to application-specific characteristics?"*:
ALMA is a full working example of a lazy metareasoning architecture that is able to perform automatic domain-specific optimizations, by using introspection to monitor and adapt to the performance of different reasoning components for the given domain and reasoning constraints.

Chapter 4
Evaluation and discussion

Breiman [8] described two different cultures for approaching the problem of reaching conclusions from data. The first is the data modeling culture, which assumes a given form of underlying process is generating the data. Given this assumption, practitioners choose a model that is suitable for this type of process, use the observed data to fit the parameters of this model, verify that the model appears to work as intended, and then are able to make theoretically justified predictions and explain how the domain works based on this model.

The other culture is the algorithmic modeling culture, which focuses on achieving the best accuracy. No particular form of underlying process is assumed, and in particular a high predictive accuracy simply means that the model is a useful tool for making predictions, not that the model explains the domain.

We believe this second approach is better suited for evaluating the overall effectiveness of fully automatic reasoning systems, and consider the ALMA system successful primarily because it empirically improved performance while controlling for CPU time spent.

This chapter evaluates the characteristics of the methods developed during this research, but the evidence that the methods also really work in practice is in the empirical results that were achieved.

4.1 Research questions and contributions

[RQ1] How can learners perform better using introspective capabilities?

> From paper A summary: *"Introspective capabilities can be used to choose between alternatives at the system level, i.e. not just learning domain-specific knowledge but additional system-level knowledge types such as similarity knowledge and adaptation knowledge."*

Machine learning fundamentally relies on data and introspection provides additional data, allowing a system to learn more. This additional data is of a different form, not providing new insights about the domain but instead about the reasoning

system itself. This insight allows automatically learning entirely different types of knowledge, e.g. similarity or adaptation knowledge for a CBR system.

[RQ2] How can the performance of reasoning systems with different capabilities be evaluated?

From paper B summary: *"Reasoning systems can be compared based on the overall empirical usefulness they provide when used in specific problem-solving scenarios. This can vary greatly based on both the problem domain and what requirements the system has to run under. Generally, it can be expressed as the benefit the system brings when run, minus the costs associated with the system."*

Machine learning methods have different strengths and weaknesses, and there is no universally best method that is always preferable. Even theoretical results about generalization performance typically rely on assumptions that do not hold perfectly in practice, e.g. that data samples behave as independent and identically distributed random variables.

However, comparing empirical performance is relatively straight-forward, even when the empirical results are produced through wildly different processes. Of course this does not in any way solve the general problem, as these results may not be reproducible, or may not generalize beyond the specific observations. But it does provide a means to make informed decisions in a given specific situation.

This kind of performance could be based on how well the system satisfies business needs in a given environment. In our research work this is modeled as the number of correct predictions, evaluated across a range of different scenarios with different resource constraints.

[RQ3] How can a lazy metareasoning architecture take advantage of introspective capabilities?

From paper C summary: *"Reasoning methods have different empirical performance curves, which can be empirically measured in an algorithm-agnostic way based on their predictive accuracy and run-time cost in a specific reasoning scenario. Through introspection, a metareasoning system can access and react to this performance meta-data, and use it to optimize the choice of method based on how well it is performing."*

From paper D summary: *"There are a large number of possible reasoning methods, meta-algorithms to combine them, and possible hyperparameter settings that affect the performance. Which methods perform best depends on the domain and the task to be solved. A metareasoning system can use performance meta-data to determine which methods should be used, allowing optimization for specific domains without requiring a human machine learning expert to be involved."*

A metareasoning system can use introspective capabilities to extract performance meta-data, which in turn can be used to make provisional decisions about what methods and parameters to choose. By experimentally modifying and then evaluating the system based on performance meta-data, a meta-level learner can determine which changes are effective and should be kept, and which changes are unhelpful and should be reverted, thereby optimizing system performance over time.

In a lazy metareasoning system this performance meta-data can be incrementally collected during execution and the decisions made just-in-time when needed, which enables sustained meta-level learning at run-time after a system has been deployed.

[RQ4] How can a practical metareasoning system automatically adapt to application-specific characteristics?

From paper E summary: *"The metareasoning system can evaluate the performance of different methods for the specific application, and prefer the method with the best empirical performance."*

From paper F summary: *"ALMA is a full working example of a lazy metareasoning architecture that's able to perform automatic domain-specific optimizations, by using introspection to monitor and adapt to the performance of different reasoning components for the given domain and reasoning constraints."*

Generalizing beyond training examples is not theoretically justified in the "general case" without making any assumptions, but it demonstrably works very well for a wide range of practically important real-world applications. There are theoretical reasons for this, as in the mathematical "general case" an arbitrarily chosen situation will be extremely complicated and chaotic, while in situations of real-world importance this is rarely the case as there will typically be a large degree of some form of inherent structure (even if nothing about this structure is known a priori, and it must all be learned from data - the important part is that some structure *exists*).

For a practical reasoning system dealing with a specific task in a given domain this is mostly a non-issue, as learning from training data precisely means to learn new knowledge that is inherently connected to the specific domain. To perform well, it is important that such a system is able to adapt as needed, meaning that general knowledge and rules-of-thumb can provide an initial bias for the reasoning process, but should not present artificial restrictions that preclude certain knowledge from being learned. This means that the parameters of a machine learning model should not be fixed as static domain-independent default values ahead of time, but should be learned. Preferably the machine learning model structure itself should also be adaptable, adjusting to the needs of the specific reasoning task.

A lazy metareasoning system can take advantage of introspective capabilities to automatically adjust parameters and method combinations based on performance

meta-data. This allows a system to incrementally learn and iteratively explore the decision space based on the most promising results seen so far. To work well in practice, such a metareasoner should be able to integrate with other systems to take advantage of the vast variety of existing machine learning methods and technology that already exist, and automatically extend the full optimization and exploration functionality of the system.

This can be achieved through a uniform representation of the various parts of the system that can be understood and modified by generic meta-level operations, as demonstrated by the component-based approach and meta-level components used in ALMA that successfully integrates with reasoning method implementations provided by WEKA.

4.2 Random decision tree algorithm

The new RDT algorithm from paper C and D was effective in producing similarity estimates very efficiently, and research showed benefits from hybrid learning systems that use multiple methods. However the trees produced by the RDT algorithm were not sufficient to create a strong classification algorithm directly from data.

In additional exploratory research, the trees were assigned randomized importance weights, which were sequentially iterated upon whenever a new set of weights improved the observed empirical results. This approach showed an initial improvement over the basic unweighted trees, but ended up quickly getting stuck with locally optimal weights that were harder and harder to improve further. Overall this random approach to setting weights did not manage to fit the training data as well as other learning methods.

This result suggests that using random weights is insufficient to learn how to generalize from raw data, and instead approaches that set and adjust weights more directly based on the available data are preferable, for example the methods used in random decision forests or gradient boosted decision trees.

4.3 Comparing ALMA to Leake and Wilson's position paper

As mentioned in 1.1, part of the inspiration for this thesis work was a list of issues and areas to explore that were identified by Leake and Wilson [33] to guide introspective learning research. ALMA addresses 7 out of 9 of these fundamental issues. The approaches taken to address three of the issues together form the foundation of metareasoning in ALMA: how to achieve a flexible learning focus, enable multistrategy introspective learning, and monitor processing characteristics in addition to outcomes.

✓ Flexible learning focus
Introspective learning research often has a narrow focus, only monitoring and re-

pairing one specific part of the system. In a complete practical system with many parts, it is not clear what process or knowledge should be assigned blame for any specific reasoning failure and needs to be repaired. Attempting all possible repair actions is not a good solution, as it has a high cost and may actually degrade the overall performance of the system as a whole.

ALMA does not have a narrow focus, and is always prioritizing holistic whole-system performance. The system changes corresponding to a "repair action" in ALMA is a new node, which can be evaluated based on how well the system actually performs with and without including it.

✓ Enabling multistrategy introspective learning
Most learning systems rely on a single learning method. ALMA on the other hand is continuously exploring and reprioritizing a wide spectrum of learning methods.

In ALMA, introspective learning is evaluated in the same manner as object-level learning, and e.g. the caching and parameter-tuning components are applied only when including them improves the measured performance. Additional forms of meta-level learning can similarly be included in ALMA by adding a new meta-level component, e.g. to change how a case base is indexed.

✓ Monitoring processing characteristics in addition to outcomes
Despite research in both areas, system repair and metamanagement of processing resources have traditionally been examined separately, with little overlap.

ALMA learns both the predictive accuracy of its learning components, and their resource cost. These performance measures are tightly integrated in ALMA, focusing on cost-effective combinations and achieving the best result given available resources.

✓ Reasoning about failure detection and response
Unlike a model-based introspective reasoner, ALMA does not have a "gold standard" to compare individual problem-solving attempts against. For example the ROBBIE system has built-in domain-specific strategies for how to respond to failures, while ALMA is focused on continually adjusting learning goals. The challenge identified by Leake and Wilson was to determine what to do in response to a failure - e.g. learning from it immediately, waiting to gather more information, or to just ignore it entirely.

The meta-level node selection algorithm used in ALMA addresses this by continuously reprioritizing where to focus additional learning resources, effectively waiting until the right time to learn from each problem-solving experience.

✓ Learning for self-understanding in addition to self-repair
This challenge relates to the ability of a system to predict its own performance, e.g. to understand its limitations and choose which methods to apply based on its own characteristics and anticipated failure modes.

ALMA does not directly anticipate failure, although strategies to do so could be included as additional meta-level components. However ALMA is continuously learning about its performance, and will also explore nodes that are considered

suboptimal, in order to gain additional self-understanding. This allows ALMA to understand when these nodes do and do not work.

✓ **Adjustable modeling levels**
Self-models are often high-level domain-independent descriptions of idealized processing. This makes it difficult to connect the model to domain- and implementation-specific details.

In ALMA, the self-model is always based on the specific implementation's performance in a specific domain. The meta-level components in ALMA are not domain-specific, but their performance and the decision about whether to use them at all is always determined empirically.

✓ **Extending models and handling imperfect self-models**
Approaches based on self-models often assume the self-model is perfect, and detrimental repair actions can be applied when an imperfect self-model incorrectly believes the repair will be helpful. One suggested approach to get away from this is to monitor the introspective learning process itself, to determine whether changes should be retained or discarded.

This is similar to ALMA's approach, where many alternative predictors are explored and trained, and the empirically-best predictor is used. ALMA's approach is effectively a set of provisional self-models whose performance is continuously evaluated to determine which one is most appropriate.

× **Supporting self-explanation**
Self-explanation is valuable as a part of human cognition and learning. Human experts often have a higher awareness of their own problem-solving process than non-experts, and are better able to explain their reasoning.

ALMA does not address these issues, neither advancing the understanding of what makes a good self-explanation, nor having the ability to explain its decisions.

× **Exploiting interaction at the meta level**
If introspective systems can explain themselves, they may also be able to receive guidance from humans to assist their learning process and correct self-models, and start to build a shared model and increased understanding together with their users.

ALMA is intended to be able to operate fully autonomously and does not address these issues.

4.4 The effect of adding a metareasoning layer

The most common effect of adding metareasoning is to increase accuracy while also increasing the computational cost (often significantly).

In our research we measured and compared the computing resources required, and showed that ALMA can improve overall classification efficiency with the same level of computation, i.e. that the metareasoning performed is more valuable than

simply spending more resources to train the underlying base reasoner. This is in contrast to e.g. common versions of Stacking [50], which impose a large overhead before any results can be produced and in our classification tests actually performed worse than most individual reasoning algorithms when performing similar amounts of computation.

The metareasoning added by ALMA still contributes to an increase in system complexity, but having some form of meta-level reasoning is necessary to achieve the benefits of introspection, including dynamic bias-shift at run-time. The way ALMA is decomposed into separate components helps to alleviate the associated complexity. This component decomposition allows single methods to remain relatively simple, while still being evaluated based on how they empirically contribute to improving the overall reasoning system, which was our goal.

Chapter 5
Concluding remarks

In this thesis we have investigated learning aspects of machine learning systems, beginning with learning in case-based reasoning (CBR) systems. This included an initial framework for CBR systems, where individual components could be replaced to change the learning behavior of the overall system to be suitable for different problem domains. This approach enables creating systems that can easily be modified and adapted to new circumstances.

By specifying the characteristics of a new problem domain, the system could automatically select which components to use. The major limitation of this system was that it relied on human domain experts to characterize the problem domain and human machine learning experts to characterize the learning components, and there was no guarantee that a matching component would really work well in practice.

This component-based approach was then extended with a framework for analyzing overall system performance in a generic manner, which supported automatically evaluating different learning approaches. This framework enables the best set of components for a particular problem domain to be chosen based on how well they actually perform, rather than being selected based on the experience and skill of human experts. The framework is generic enough to evaluate both eager and lazy learning methods in a uniform manner, and is no longer limited to CBR systems.

Finally a fully automatic metareasoning architecture was developed, which uses the previous framework to continuously estimate the performance of individual machine learning methods at run-time. This approach allows a learning system to adapt and optimize itself while running, and in experiments can do so efficiently enough to outperform individual machine learning methods and other metareasoning approaches that were evaluated.

5.1 Future work

Transfer learning to bootstrap new tasks

To achieve even better results than in our tests, it is possible to include extra prior information about the individual learning components instead of starting every experiment with a blank slate and viewing them as indistinguishable black boxes. For example, based on our results the overall system would perform better by simply omitting half the algorithms. This selection should be based on how they contribute to the overall performance across datasets, which is *not* the same as their average individual performance - in our experiments the MultiLayerPerceptron is one of the most important components with the best performance on several data sets, despite its average performance being quite poor (because it is too slow for many of the other data sets).

Instead of selecting a subset of components, this component-biasing could also be implemented as a prior weight for components where an additional term is added to the exploration part of the UCT score. This is similar to how the RAVE algorithm and AlphaGo's neural networks influence the tree-search when playing Go. For Alma, this approach would bias the search to explore certain preferred components first when there is very high uncertainty in early phases of learning, while still retaining the UCT behavior of eventually exploring all the components and making the final decision based on empirical results.

Applications for lifelong learning

The Alma architecture was designed to support lazy incremental learning, which means systems can continue to learn and adapt autonomously after being deployed. These ideas have recently received more attention within the AutoML community, with a competition and workshop at NeurIPS 2018 specifically dedicated to lifelong learning without any human intervention.

The ideas from this thesis research can potentially improve the capabilities and performance of new lifelong learning systems, by explicitly learning meta-level performance data, considering exploration/exploitation trade-offs to optimize performance over time, and by adopting a modular architecture that can efficiently reuse and tune already established components for new situations where they empirically work well.

Bibliography

[1] Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications 7(1), 39–59 (1994), https://doi.org/10.3233/AIC-1994-7104

[2] Andrychowicz, M., Denil, M., Colmenarejo, S.G., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. pp. 3988–3996. NIPS'16, Curran Associates Inc., USA (2016), https://dl.acm.org/citation.cfm?id=3157382.3157543

[3] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. 47(2-3), 235–256 (2002), https://doi.org/10.1023/A:1013689704352

[4] Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (Sep 1975), https://doi.org/10.1145/361002.361007

[5] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems. pp. 2546–2554. NIPS'11, Curran Associates Inc., USA (2011), https://dl.acm.org/citation.cfm?id=2986459.2986743

[6] Brachman, R.J., Gilbert, V.P., Levesque, H.J.: An essential hybrid reasoning system: Knowledge and symbol level accounts of krypton. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1. pp. 532–539. IJCAI'85, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1985), https://dl.acm.org/citation.cfm?id=1625135.1625237

[7] Breiman, L.: Stacked regressions. Machine Learning 24(1), 49–64 (1996), https://doi.org/10.1007/BF00117832

[8] Breiman, L.: Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statist. Sci. 16(3), 199–231 (2001), https://doi.org/10.1214/ss/1009213726

[9] Bruland, T., Aamodt, A., Langseth, H.: Architectures integrating case-based reasoning and bayesian networks for clinical decision support. In: Shi, Z., Vadera, S., Aamodt, A., Leake, D. (eds.) Intelligent Information Processing V. pp. 82–91. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), https://doi.org/10.1007/978-3-642-16327-2_13

[10] Brügmann, B.: Monte Carlo Go. In: AAAI Fall symposium on Games: Playing, Planning, and Learning (1993), http://www.ideanest.com/vegos/MonteCarloGo.pdf

[11] Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge University Press (2006), https://doi.org/10.1017/CBO9780511546921

[12] Chakrabarti, D., Kumar, R., Radlinski, F., Upfal, E.: Mortal multi-armed bandits. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in Neural Information Processing Systems 21, pp. 273–280. Curran Associates, Inc. (2009), http://papers.nips.cc/paper/3580-mortal-multi-armed-bandits.pdf

[13] Clemen, R.T.: Combining forecasts: A review and annotated bibliography. International Journal of Forecasting 5(4), 559–583 (1989), https://ideas.repec.org/a/eee/intfor/v5y1989i4p559-583.html

[14] Cortes, C., Gonzalvo, X., Kuznetsov, V., Mohri, M., Yang, S.: AdaNet: Adaptive structural learning of artificial neural networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 874–883. PMLR, International Convention Centre, Sydney, Australia (06–11 Aug 2017), http://proceedings.mlr.press/v70/cortes17a.html

[15] Cox, M.T., Raja, A.: Metareasoning: A manifesto. Tech. rep., BBN TM-2028, BBN Technologies (2007)

[16] Cox, M.T., Ram, A.: Introspective multistrategy learning: On the construction of learning strategies. Artificial Intelligence 112(1), 1 – 55 (1999), https://doi.org/10.1016/S0004-3702(99)00047-8

[17] Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research 20(55), 1–21 (2019), http://jmlr.org/papers/v20/18-598.html

[18] Feurer, M., Springenberg, J., Hutter, F.: Initializing bayesian hyperparameter optimization via meta-learning (2015), https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029

[19] Fox, S., Leake, D.B.: Introspective reasoning for index refinement in case-based reasoning. J. Exp. Theor. Artif. Intell. 13(1), 63–88 (2001), https://doi.org/10.1080/09528130010029794

[20] Francis, A.G., Ram, A.: The utility problem in case-based reasoning. In: AAAICBR-93, the Proceedings of the 1993 Case-Based Reasoning Workshop (1993)

[21] Gelly, S., Wang, Y.: Exploration exploitation in Go: UCT for Monte-Carlo Go. Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006) (2006), https://hal.archives-ouvertes.fr/hal-00115330

[22] Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier: A service for black-box optimization. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1487–1495. KDD '17, ACM, New York, NY, USA (2017), https://doi.org/10.1145/3097983.3098043

[23] Hinton, G.E.: Learning multiple layers of representation. Trends in Cognitive Sciences 11(10), 428–434 (2007), https://doi.org/10.1016/j.tics.2007.09.004

[24] Houeland, T.G., Aamodt, A.: Towards an introspective architecture for meta-level reasoning in clinical decision support systems. In: Proceedings of the ICCBR 2009 Workshops. pp. 235–244. Springer Verlag (2009), https://doi.org/10.21427/D71G70

[25] Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proceedings of the 5th International Conference on Learning and Intelligent Optimization. pp. 507–523. LION'05, Springer-Verlag, Berlin, Heidelberg (2011), https://doi.org/10.1007/978-3-642-25566-3_40

[26] Hutter, M.: The Universal Algorithmic Agent AIXI, pp. 141–183. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), https://doi.org/10.1007/3-540-26877-4_5

[27] J. M. Bates, C.W.J.G.: The combination of forecasts. OR 20(4), 451–468 (1969), https://doi.org/10.2307/3008764

[28] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization 13(4), 455–492 (Dec 1998), https://doi.org/10.1023/A:1008306431147

[29] Kearns, M.: Thoughts on hypothesis boosting. Unpublished manuscript (Machine Learning class project) (1988)

[30] Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: ECML-06. Number 4212 in LNCS. pp. 282–293. Springer (2006), https://doi.org/10.1007/11871842_29

[31] Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI. vol. 14, pp. 1137–1145. Montreal, Canada (1995), https://dl.acm.org/doi/10.5555/1643031.1643047

[32] Kuleshov, V., Precup, D.: Algorithms for multi-armed bandit problems. CoRR (2014), https://arxiv.org/abs/1402.6028

[33] Leake, D., Wilson, M.: Extending introspective learning from self-models. Proceedings of the AAAI 2008 Workshop on Metareasoning: Thinking About Thinking (2008)

[34] Li, B., Hoi, S.C.H.: Online portfolio selection: A survey. ACM Comput. Surv. 46(3), 35:1–35:36 (Jan 2014), https://doi.org/10.1145/2512962

[35] Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 4095–4104. PMLR, Stockholmsmässan, Stockholm Sweden (10–15 Jul 2018), http://proceedings.mlr.press/v80/pham18a

[36] Radhakrishnan, J., Ontañón, S., Ram, A.: Goal-driven learning in the GILA integrated intelligence architecture. In: Boutilier, C. (ed.) IJCAI. pp. 1205–1210 (2009), https://dl.acm.org/doi/abs/10.5555/1661445.1661638

[37] Reinartz, T., Iglezakis, I., Roth–Berghofer, T.: Review and restore for case-base maintenance. Computational Intelligence 17(2), 214–234 (2001), https://doi.org/10.1111/0824-7935.00141

[38] Richter, M.M., Aamodt, A.: Case-based reasoning foundations. The Knowledge Engineering Review 20(3), 203–207 (2005), https://doi.org/10.1017/S0269888906000695

[39] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature 529(7587), 484–489 (01 2016), https://doi.org/10.1038/nature16961

[40] Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2. pp. 2951–2959. NIPS'12, Curran Associates Inc., USA (2012), https://dl.acm.org/citation.cfm?id=2999325.2999464

[41] Sollich, P., Krogh, A.: Learning with ensembles: How over-fitting can be useful. In: Proceedings of the 8th International Conference on Neural Information Processing Systems. p. 190–196. NIPS'95, MIT Press, Cambridge, MA, USA (1995), https://dl.acm.org/doi/10.5555/2998828.2998855

[42] Solomonoff, R.: A formal theory of inductive inference. part i. Information and Control 7(1), 1 – 22 (1964), https://doi.org/10.1016/S0019-9958(64)90223-2

[43] Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika 25(3-4), 285–294 (12 1933), https://doi.org/10.1093/biomet/25.3-4.285

[44] Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 847–855. KDD '13, Association for Computing Machinery, New York, NY, USA (2013), https://doi.org/10.1145/2487575.2487629

[45] Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A Monte-Carlo AIXI approximation. J. Artif. Int. Res. 40(1), 95–142 (Jan 2011), https://dl.acm.org/citation.cfm?id=2016945.2016949

[46] Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. 18(2), 77–95 (2002), https://dx.doi.org/10.1023/A:1019956318069

[47] Wallis, K.F.: Combining forecasts – forty years later. Applied Financial Economics 21(1-2), 33–41 (2011), https://doi.org/10.1080/09603107.2011.523179

[48] Watson, I.: A case study of maintenance of a commercially fielded case-based reasoning system. Computational Intelligence 17(2), 387–398 (2001), https://dx.doi.org/10.1111/0824-7935.00151

[49] Weill, C., Gonzalvo, J., Kuznetsov, V., Yang, S., Yak, S., Mazzawi, H., Hotaj, E., Jerfel, G., Macko, V., Adlam, B., Mohri, M., Cortes, C.: AdaNet: A scalable and flexible framework for automatically learning ensembles (2019), https://arxiv.org/abs/1905.00080

[50] Wolpert, D.H.: Stacked generalization. Neural Networks 5(2), 241 – 259 (1992), https://doi.org/10.1016/S0893-6080(05)80023-1

[51] Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Computation 8(7), 1341–1390 (2013/11/19 1996), https://doi.org/10.1162/neco.1996.8.7.1341

[52] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning (2017), https://arxiv.org/abs/1611.01578

Part II

Selected papers

Paper A
An Introspective Component-Based Approach for Meta-Level Reasoning in Clinical Decision Support Systems

Authors: Tor Gunnar Høst Houeland and Agnar Aamodt

Published in: Proceedings of the First Norwegian Artificial Intelligence Symposium (NAIS'09). pp. 121–132. Tapir Forlag (2009), ISBN: 978-82-519-2519-8

My contributions: I developed the meta-level architecture presented in the paper, wrote the sections on metareasoning, vocabulary, and architecture, and incorporated comments and suggested changes into the text.

An Introspective Component-Based Approach for Meta-Level Reasoning in Clinical Decision Support Systems*

Tor Gunnar Houeland, Agnar Aamodt
Department of Computer and Information Science,
Norwegian University of Science and Technology,
NO-7491 Trondheim, Norway
{houeland, agnar}@idi.ntnu.no

Abstract

The paper presents core elements of a meta-level architecture for clinical decision support, in the domain of palliative care. The goal of the reported research is to develop an architecture and an integrated set of methods for an introspective meta-level reasoner. Within the architecture a system is under development that addresses the identification and utilization of clinical guidelines for the assessment and treatment of cancer pain. Case-based reasoning is a core component of the architecture, which also incorporates rule-based and probabilistic model-based methods. The paper presents the overall architectural constraints and exemplifies parts of it through structured component descriptions.

1 Introduction

A clinical decision support system that covers several clinical tasks, such as patient examination, disease hypotetization, diagnosis determination, treatment planning, and drug administration, would typically need to combine several types of knowledge and several reasoning methods to provide good advisory support. There has recently been a renewed interest in meta-level reasoning in which a computer reasons about its own reasoning processes as well as the problem at hand [1].

The idea is that this allows the system to improve its own reasoning processes, for example by determining why a problem was solved incorrectly. By identifying where the error occurred in the reasoning process, the faulty part can be amended which would let the system solve such problems correctly in the future. In order to enable the meta-level reasoner to improve its own performance, learning within the meta-level reasoner itself, i.e. introspective learning, is also called for.

The focus of the research presented here is on meta-level reasoning for clinical decision support. A component-based architecture and an integrated set of methods for a meta-level reasoner to improve its reasoning and learning abilities are currently being

*An earlier version of this paper was presented at the Seventh Workshop on Case-Based Reasoning in the Health Sciences, July 21, 2009. We'd like to thank the other participants for their helpful and encouraging feedback.

This paper was presented at NAIS-2009; see http://events.idi.ntnu.no/nais2009/.

developed. The architecture allows for the integration of all three reasoning paradigms in symbolic AI, i.e. rule-based, case-based, and (deeper) model-based reasoning.

We are addressing this problem in the domain of clinical decision support for palliative care. In a cooperation with the Palliative Medicine Unit, Cancer Department, St. Olavs University Hospital in Trondheim, we are studying the potential for proactive, advice-giving systems for the improved treatment of pain in cancer patients. In our current project, short-named TLCPC, which has national funding, the focus is on lung cancer. However, this research is tightly linked to a larger EU project called EPCRC [2], which covers all forms of cancer pain as well as other problems related to palliative care for long-term cancer patients.

A motivation for that project is also to standardize procedures and to unify clinical practice in palliative care [3], a goal for which computerized decision support systems have a strong potential. The system under development will in particular address the identification and utilization of clinical guidelines [4].

In the next section we review some of the related research in metareasoning with case-based components, and CBR used for guideline supported clinical decision-making. In section 3 our metareasoning approach and introspective architecture are introduced. Section 4 illustrates important semantic types used to describe components in the architecture, which is discussed in section 5. The status of the clinical guideline application is presented in section 6. Concluding remarks end the paper.

2 Related Research

There is a significant amount of research that has addressed metareasoning in relation to case-based reasoning systems. In Meta-AQUA [5, 6] introspection is used in the retain phase to learn from mistakes. What is referred to as *introspective meta-XPs* are used to represent failures encountered while the system operates. The system constructs learning plans that consist of calling various learning algorithms.

Early accounts of meta-level architectures involving CBR also include BOLERO [7] and ANALOG [8]. In BOLERO, a case-based meta-level planner controls the execution of a rule-based reasoner designed to accomplish different medical diagnosis tasks. New plans are constructed during the problem solving process when needed, and captured as new cases by the meta-level learner. ANALOG provides a knowledge modeling framework and a system architecture that links various types of tasks, methods, and domain knowledge types. A selected method runs until an impasse situation is encountered, at which time a new method selection process is run. The meta-level learner remembers failed and successful instantiations of methods.

Christodoulou and Keravnou describe a meta-level architecture [9] in the domain of breast cancer histopathology. Each problem solver is associated with a task, an inference mechanism, and domain knowledge constraints. A set of meta-parameters is used by the metareasoner to characterize knowledge types (e.g. experience-based, causal), and desired solution properties (e.g. level of detail, accuracy, efficiency). The metareasoner is a case-based reasoner that captures and stores problem-solving paths as strategy cases incrementally.

In the ADAPtER system [10], an architecture that combines model-based and case-based reasoning for medical diagnosis, the CBR component is the primary object level reasoner. The model-based method is triggered either when the retrieval method fails to find a matching case, or the case adaptation module fails. Cases are captured as a compilation of the model-based process.

ROBBIE [11] is a case-based planning system in which an introspective model-based reasoner provides learning goals for the system when it fails to meet reasoning performance expectations. This way of generating learning objectives based on reasoning failures is similar to Meta-AQUA, although it is performed in different ways.

jCOLIBRI [12] is a Java framework for building CBR systems with metareasoning capabilities. It uses a two-layer architecture that separates the user interface and core classes, and uses advanced feature of the Java language and libraries to simplify development. jCOLIBRI uses a task structure with semantics defined in CBROnto, an ontology created for representing the terms and concepts that are important for CBR development.

The principles of evidence-based medicine, in which systematic research evaluation regimes are used to assess and justify results from medical research, form a well-established basis for medical practice [13]. One manifestation of this is the specification of clinical guidelines, i.e. operational procedures for how to conduct a particular type of patient examination, make a diagnosis, administer a type of drug, or perform other types of treatment.

Another source of information used by clinicians in their daily practice is, of course, the set of experiences a clinician has from earlier patients. While guidelines are important in unifying high-quality practice at a general level, past patient cases provide another level of specificity, closer to concrete actions. Hence, a combination of general guidelines with past experience cases is potentially a strong combination. Identifying the strengths and weaknesses of a case-based method vs. a generalization-based one can be done analytically or experimentally.

Marling et. al. [14] reports on an experiment in the domain of nutritional menu planning, in which a hybrid system was developed by combing the strengths of separate rule-based and case-based reasoners. The hybrid system outperformed both separate systems.

In the CARE-PARTNER system [15] medical guidelines are realized in the form of problem solving pathways, implemented as a rule-based system combined with information retrieval, and with CBR as the main problem solving method. GLARE [16] is a general decision-support system for managing and utilizing clinical guidelines, in which guidelines are represented as a hierarchical structure of decision paths. Based on that system a CBR system was incorporated as additional knowledge to handle situations that are not covered by the guidelines, so-called non-compliances [17].

Even if several medical guideline systems combine different reasoning paradigms, there has been very little work on moving the combined reasoning up to the metareasoning level.

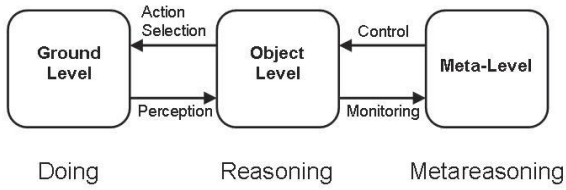

Figure 1: Duality in reasoning and acting (from Cox and Raja 2008 [1])

3 Metareasoning Approach

Recently, Cox and Raja [1] presented a general high-level framework of metareasoning, shown in figure 1. It relates three levels, i.e the ground level (physical perception and action), the object level (reasoning about action), and the meta-level (reasoning about reasoning).

Within this high-level perspective, Leake and Wilson [18] address the learning part of metareasoning, and present a set of challenging issues, such as learning for self-understanding and self-explanation. Many meta-level reasoning systems start by building a partial solution and then invoking meta-level reasoning functionality to determine whether the object-level reasoning processes are working satisfactorily or not. Such systems then either proceed as normal, or classify the reasoning process as a failure and create a new metareasoning goal to learn from this failure.

This is in contrast to the meta-level control agent in our approach, which operates on the object-level reasoning components directly without a specific reasoning failure to address. This allows the system to have a clearer broad focus on performing changes that affect the entire reasoning system instead of correcting single failures, and this broadening is also identified by Leake and Wilson [18] as an important opportunity for a more flexible learning focus.

Future planned meta-level components in our approach such as a competence-evaluator for problem-solving methods can also assist in providing the system with a level of self-understanding, which is another identified opportunity for introspective learning systems.

A system using our introspective architecture with CBR reasoning methods which follows this general framework is shown in figure 2. The ground level is represented as I/O for the system, which is not covered in our architecture but must necessarily exist in some form for a complete CBR system. In the shown system reasoning tasks are explicitly represented, with a “CBR-based problem solving task” at the object level and several metareasoning tasks.

In our architecture there is no fundamental difference in how reasoning tasks for the two levels must be represented, but they are shown as seperate boxes in the figure to ease understanding and fit the metareasoning framework of Cox and Raja. The importance of our introspective architecture lies in the components, each of which implements a specific functionality, and is modified and assigned to a particular reasoning task by meta-level reasoning components.

Richter [19] introduced the knowledge container model, where pieces of knowledge can be categorized into four categories: case vocabulary, cases, similarity assessment and adaptation knowledge. Given that a set of precise and consistent terms is important for reasoning at the meta-level, we explicitly include the CBR system vocabulary as part of

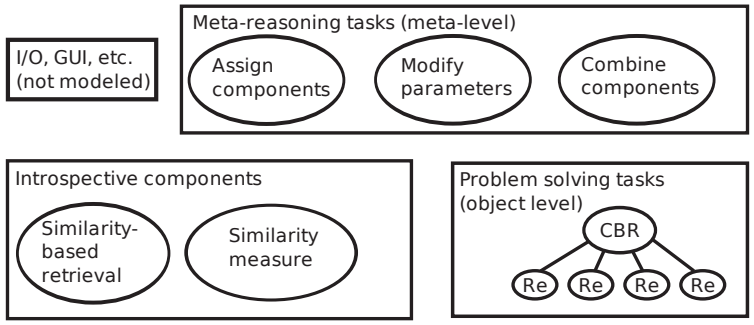

Figure 2: A CBR system based on the introspective architecture

the case vocabulary category.

To make sure that the vocabulary contains semantics that are useful for human designers, we suggest that the vocabulary should be created and updated manually, unlike the other knowledge sources which will be changed and influenced by the meta-level reasoner.

In our architecture the case knowledge is stored in cases, similarity assessment as part of the components providing methods for object-level retrieval and the adaptation knowledge is shared between object level reuse components and a more profound system-level adaptation in meta-level components.

For the meta-level control agent to be able to examine and calibrate the system's components, it's important that the settings available can be interpreted by the control program, and that the components expose interfaces at an appropriate level of abstraction. To facilitate this, we are developing a vocabulary of CBR-related terms and the intended semantics as part of our approach.

An important aspect of the vocabulary is that it describes what the terms mean at a semantic level without relying on the specific realizations in any particular CBR system implementation. An example of this is a case base, which is simply defined as a set of cases. While in practice many CBR systems store the cases as a form of ordered lists, their reasoning processes typically do not rely on the particular order the cases are listed in within computer memory or on disk.

The purpose of the vocabulary is to abstract away the particular specifics in implementations and generalize the terms to a semantic level where it describes the actual requirements for a term without imposing a needlessly specific design.

Our vocabulary is being developed to be conceptually compatible with CBROnto [20], a modeling framework that has already examined the meanings and relations between terms from an ontological perspective. As an added benefit the CBROnto ontology is compatible with the "4 REs" CBR process cycle [21] which is widely quoted and referred to in the CBR literature.

4 Vocabulary Examples

Of particular importance for our semantic task description is the type system for inputs and outputs. These types are meant to represent at the knowledge level the essential meaningful content that is to be processed. For system implementations the exact structures used to realize these knowledge types may differ, and often the same programming language structures may be used for several different knowledge types. E.g.

for a case-based reasoning system based on feature vectors, the stored cases, problem specifications, retrieval queries and produced solutions will typically all be implemented using the same type of programming language data structure, but they are still different on a semantic level.

Important semantic input and output base types.

Case A record of a problem solving experience, consisting of the encountered problem and the solution details

AttributeCase A subtype of Case where the problem descriptions are represented as a set of features

Feature A particular aspect of an AttributeCase, where each separate case can either lack the feature or have a corresponding value selected from the feature's set of possible values

ProblemInput A problem specification for a problem-solving method

ProblemSolution A solution to a problem

SolutionMethod A method used to create a problem solution

SolutionEvaluation An evaluation of a solution applied to a problem

Similarity A floating point number used specifically to represent similarity

Derived semantic types.

Set$< T >$ A collection of objects of type T

List$< T >$ An ordered collection of objects of type T

(*a***,** *b***,** *c***, ...)** An n-tuple such as **(Case, Similarity)** consisting of n values where the first is of types a, the second of type b, etc.

$A \rightarrow B$ A function such as **Case** $\rightarrow$ **Similarity** mapping inputs of type A to output of type B

Other terms used in the example figures.

SizeOf The number of objects in a *Set* or *List* collection

PC An identifier used in the examples to specify an unrestricted type for problem characterizations

CB An identifier used in the examples to specify a type adhering to the **Case** semantics for cases in a case base

Task	Input	Output
Case-based reasoning	**ProblemInput**	**ProblemSolution**
- Retrieve	**ProblemInput**	**Set**<*CB*>
– Problem characterization	**ProblemInput**	*PC*
– Case retrieval	*PC*	**Set**<*CB*>
– Focus	**Set**<*CB*>	**Set**<*CB*>
- Reuse	(*PC*, **Set**<*CB*>)	**ProblemSolution**
– Adapt solution method	(*PC*, **Set**<*CB*>)	**SolutionMethod**
– Adapt solution	(*PC*, **SolutionMethod, Set**<*CB*>)	**ProblemSolution**
- Revise	**ProblemSolution**	**(PS, SE)**
– Evaluate solution	**ProblemSolution**	**SolutionEvaluation**
– Repair solution	**(PS, SE)**	**ProblemSolution**
- Retain	**(PS, SE)**	
– Update general knowledge	**(PS, SE)**	
– Add to case base	**(PS, SE)**	

(PS, SE) is an abbreviation for **(ProblemSolution, SolutionEvaluation)** due to space concerns.

Figure 3: Inputs and output for the CBR tasks

5 Introspective Architecture

In our conceptual framework we consider the traditional core case-based reasoning process as one possible problem solving method at the object level. We consider this to be one limited part of the combined reasoning system, with specific responsibilities, and there can be potentially many other problem solving methods implemented in the same architecture.

A high-level task decomposition of the CBR process for our architecture is shown in figure 3, where the task "Case-based reasoning" is split into the 4 RE subtasks, and each of these are further split into smaller subtasks. Any components that are assigned to parts of this process must match the semantic input and output types for the corresponding tasks, which are also included in the figure.

To elaborate on the case-based retrieval task, it starts from a **ProblemInput** and retrieves a set of cases **Set**<*CB*> where *CB* is a type adhering to the **Case** semantics, and is further subdivided into subtasks that further specify how this is performed. It starts with a subtask identifying the important aspects of the problem and characterizing it as a query, transforming the **ProblemInput** into an intermediate form *PC*.

This intermediate form is not restricted by the architecture, and the only requirement is that the methods performing the subsequent CBR subtasks can accept the characterization form *PC* as input. This characterization is used to retrieve a number of previous cases from the case base, and then this is further narrowed to just focus on the most relevant information by e.g. filtering out a subset of cases or generalizing cases. The other subtasks are formalized in a similar way and correspond to the well-known steps that are often used to describe CBR systems.

In our approach, the metareasoning components exist separately from the object-level CBR reasoning components, and in fact influence and controls how the CBR problem solving method is performed, which corresponds to the aforementioned metareasoning cycle [1]. By evaluating how the CBR method performs while solving actual new problem

instances, the meta-level control agent can identify the strengths and weaknesses of the current system and attempt to use this to improve the system's competence or use alternative reasoning methods. For our meta-level component-combiner this is achieved by attempting to re-solve problems using different assignments of methods for each of the tasks and subtasks in the architecture. Whether the the newly combined reasoning process is an improvement is then evaluated based on whether the solutions produced for individual problem queries are correct for more problem instances than before.

Architectural components

One of the most important features of this architecture is that each component contains extra structures that semantically describe the component semantically using our CBR vocabulary. It is on the basis of this added information that the meta-level control component can automatically assign components to perform the system's reasoning tasks. Figure 4 shows such a self-describing semantic structure for an example object-level retrieval component.

Each component contains a list of types that apply for the component, which is listed first in the figures. This can either just be a single *name* to indicate that any type is supported, or a statement of the form "*name* isa *supertype*", which indicates that the *name* type must support the same operations as *supertype*. This is useful when a component performs a generic operation that can apply to many different types of input, and e.g. only requires that the input and output types are the same or that two inputs are comparable. This section is empty for simpler components that only refer directly to specific types in the vocabulary, such as the illustrated example retrieval component.

After that the input and output variables are listed. Each variable consists of a line of the format "*name*: *type*", where *name* is an identifier that is used to refer to the input or output throughout the component specification and *type* is the semantic type, which can either be a specific from the vocabulary or one of the types specified in the component's Types section. An example of this is the line "*query*: **AttributeCase**" from the example component, which means that the component receives an input of type **AttributeCase** which is referred to as *query* in the component description.

The following section lists Conditions that have to be fulfilled for the component to produce the expected results. As long as the input variables conform to the specified conditions, the output variables are guaranteed to follow the specifications in the Guarantees section. While the type specifications are semantic restrictions on what kind of operation the component can perform, the conditions specify for which values of inputs the component will actually behave as intended, and these conditions are not necessarily checked by the component.

Because of this the component's operation can usually be performed for non-conforming inputs, but this can produce undefined results and should be avoided. By listing the conditions in the self-describing structure, the metareasoning component can make sure that only compatible components are combined, or that the conditions are checked on-demand before the operation is performed where this cannot be guaranteed.

The final section is a short semantic description of the core functionality performed by the component. This has the format "*x* based on *y*", where *x* and *y* are statements composed using the input and output variables as well as a set of pre-specified terms representing important concepts related to the reasoning system and the application domain.

Based on these two examples, the meta-level control agent can create a new

Types:	
Input:	*query*: **AttributeCase** *casebase*: **Set**<**AttributeCase**> *similarityfunction*: (**AttributeCase** *source*, **AttributeCase** *target*) → **Similarity**
Output:	*ordered*: **List**<**AttributeCase**>
Conditions:	$5 \leq$ **SizeOf**(*casebase*) < 100000 $0 \leq similarityfunction(x, y) \leq 1$
Guarantees:	**SizeOf**(*ordered*) = 5
Approach:	order(*casebase*) based on *query*

A component that assigns an order to the *casebase* based on similarity to the *query* using the specified *similarityfunction*.

Types:	*cCase* isa **AttributeCase**
Input:	*source*: *cCase* *target*: *cCase*
Output:	*similarity*: **Similarity**
Conditions:	
Guarantee:	$0 < similarity \leq 1$
Approach:	compute similarity *similarity* based on *source* features, *target* features

A component for computing similarity based on local feature similarities between *source* and *target*.

Types:	
Input:	*query*: **AttributeCase** *casebase*: **Set**< **AttributeCase** >
Output:	*ordered*: **List**< **AttributeCase** >
Conditions:	$5 \leq$ **SizeOf**(*casebase*) < 100000
Guarantee:	**SizeOf**(*ordered*) = 5
Approach:	order *casebase* based on *query* similarity

A combined component that assigns an order to the *casebase* based on feature weighted similarity to the *query*.

Figure 4: Example components for object-level reasoning tasks.

component that accepts a query and case casebase of type **AttributeCases** and a feature weight map and returns a ranked list of matching cases. This is done by substituting **AttributeCase** for *cbType* and *cCase* (which fulfills the listed type requirements) and using the feature similarity measure as the $similarityfunction$ (by identifying that the output guarantee $0 < similarity \leq 1$ fulfills the $0 \leq similarityfunction(x, y) \leq 1$ condition).

The meta-level learning component can also further refine this into a component that learns the feature weight map automatically while it is being used. This can be done by matching it with an appropriate learning method that e.g. takes a casebase (**Set**< **AttributeCase** >) as input and produces a similarity importance value for each feature among cases in the casebase.

6 Palliative Care Application

Although there has been a lot of focus on developing clinical guidelines within the medical community, and several guideline systems have been developed by medical professional organizations, there is not a consensus as to what is a good guideline system. Further, the active use of guidelines in a clinical setting is far from the level desired, both from the perspective of quality of treatment and the perspective of unified treatment across hospitals and countries.

In our partner project EPCRC, being a collaboration involving many highly influential medical groups across Europe, the aim is to reach a consensus on a set of high-quality and operational guidelines that will be used in practice. While awaiting the results from EPCRC, we currently work with an existing set of guidelines defined by NCCN (National Comprehensive Cancer Network) in the US. An ontology is currently being built based on a combination of generic UMLS terms combined with terms from the SNOMED and NCI (National Cancer Institute in the US) ontologies.

The top-down design process is combined with bottom-up experimental system building, starting from simple system components that will be combined. Currently a simple a rule-based reasoner is being implemented at the object level. Example guidelines link patient data related to pain level, pain history, and history of treatment, to the next treatment. Type of treatment considered is the administration of different types of analgesics, with opioids as the largest subclass. Based on the results from the initial model, the system will be extended with case-based and model-based components. The model-based component will a be Bayesian network for reasoning about causality under uncertainty.

Acquiring the necessary medical knowledge needed for our experiments is a continuing process. To advance the method development we are developing a toy example system in the domain of advice-giving for film selection, in parallel to the more complex clinical guidelines system. In that system CBR methods are currently focused on both the meta and object levels. Cases, problems and solutions are currently represented as feature vectors, and there are no explicitly represented general knowledge structures outside of the CBR reasoning components. To predict a movie rating for a user, a retrieval component retrieves a number of similar other users that have seen the movie, where similarity is determined as the average difference in ratings for movies both users have rated. A simple reuse component then copies the majority rating among the retrieved cases. The system has a meta-level control agent that adheres to the principles of the metareasoning approach described earlier. Component combinations are tried out based on matching the input and output type descriptions. Although simple and different

from the clinical application, the system assists in the bottom-up specification of the introspective architecture by providing a test-bed for experiments.

In a clinical guideline support system past cases may be utilized in several ways. The role we have intended for the cases in our guideline system is two-fold. Having arrived at a leaf node in the guideline structure, CBR will be used to continue from there by providing a more specific and detailed advice, based on adapting a past result. On the other hand, if the guideline system cannot provide reasonable advice, CBR is triggered as a complementary method. The latter approach is similar to the non-compliance method [17] referred to earlier.

7 Conclusions

In this paper we have presented an introspective approach to meta-level learning and outlined a component-based architecture for designing reasoning systems which supports our introspective methods. The main contribution of our approach is the way our architecture allows for gradual additions of metareasoning methods that focus on broad, system-wide improvements.

We are working on further developing this architecture, and adding new components directed towards both object-level and meta-level reasoning methods for a clinical decision support system based on medical treatment guidelines.

Acknowledgements

This research is supported by Norwegian Research Council (NFR) grant, Contract no 183362, Translational Research in Lung Cancer and Palliative Care (TLCPC), together with funds from the Norwegian University of Science and Technology (NTNU). Thanks to Tore Bruland and Helge Langseth who contributed on the AI method side through a series of discussions, Sunil Raja who is our main contact on the medical side and has provided insight into clinical pain assessment and treatment, and Stein Kaasa who is the initiator and project leader of both the TLCPC and EPCRC projects.

References

[1] Cox, M.T., Raja, A.: Metareasoning: A manifesto. Technical report, BBN TM-2028, BBN Technologies (2007)

[2] Haugen, D.F., Kaasa, S.: The EPCRC. Eur J Palliat Care (2007; 14(3):130)

[3] Kaasa, S.: Palliative care research – time to intensify international collaboration. Palliat Med (2008; 22:301-2.)

[4] Quaseem, A.e.: Evidence-Based Interventions to Improve the Palliative Care of Pain, Dyspnea, and Depression at the End of Life: A Clinical Practice Guideline from the American College of Physicians. Annals of. Internal Medicine, vol. 148, no. 2 (2008:141-146)

[5] Cox, M.T., Eiselt, K., Kolodner, J., Nersessian, N., Recker, M., Simon, T.: Introspective multistrategy learning: On the construction of learning strategies. Artificial Intelligence **112** (1999) 1–55

[6] Cox, M.T.: Multistrategy learning with introspective meta-explanations. In: Machine Learning: Proceedings of the Ninth International Conference, Morgan Kaufmann (1992) 123–128

[7] Lopez, B., Plaza, E.: Case-based planning for medical diagnosis. In Ras., Z. (ed.), Proceedings of ISMIS-93. LNAI 837, Springer, 1993: 96-105

[8] Arcos, J., Plaza, E.: A reflective architecture for integrated memory-based learning and reasoning. In Weiss, S. et. al. (eds.): LNAI 873 , Springer, 1994: 289-300

[9] Christodoulou, E., Keravnou, E.: Metareasoning and meta-level learning in a hybrid knowledge-based architecture. Artificial Intelligence in Medicine 14 (1998): 53-81

[10] Torasso, P.: Multiple representations and multi-modal reasoning in medical diagnostic systems. Artificial Intelligence in Medicine, 23 (2001): 49-69.

[11] Fox, S., Leake, D.B.: Combining case-based planning and introspective reasoning. In: Proc. of the 6th Midwest Artificial Intelligence and Cognitive Science Society Conference. (1995) 95–03

[12] Recio-García, J.A.e.a.: Ontology based CBR with jCOLIBRI. In Ellis, R., Allen, T., Tuson, A., eds.: Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, Springer (December 2006) 149–162

[13] Friedland, D.: Evidence-based medicine: A framework for clinical practice. (1998)

[14] Marling, C., Petot, G., Sterling, L.: Integrating case-based and rule-based reasoning to meet multiple design constraints. Comp. Intell., 15 (3), 1999, 308-332

[15] Bichindaritz, I., Kansu, E., Sullivan, K.M.: Case-Based Reasoning in CARE-PARTNER: Gathering Evidence for Evidence-Based Medical Practice. EWCBR'98, LNAI 1488, pp. 334-345 (1998)

[16] Terenziani, P., Molino, G., Torchio, M.: A modular approach for representing and executing clinical guidelines. Artificial Intelligence in Medicine, 23(3):249–276 (2001)

[17] S. Montani, S.: Case-based reasoning for managing non-compliance with clinical guidelines. ICCBR 2007, 5th Workshop on CBR in the Health Sciences (2007)

[18] Leake, D., Wilson, M.: Extending introspective learning from self-models. Proceedings of the AAAI 2008 Workshop on Metareasoning: Thinking About Thinking (2008)

[19] Richter, M.M.: The knowledge contained in similarity measures. Invited Talk at ICCBR-95 (1995)

[20] Díaz-Agudo, B., González-Calero, P.A.: CBROnto: A Task/Method Ontology for CBR. In: Procs. of the 15th International FLAIRS'02 Conference, AAAI Press (2002) 101–105

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

Paper B
The Utility Problem for Lazy Learners - Towards a Non-eager Approach

Authors: Tor Gunnar Høst Houeland and Agnar Aamodt

Published in: Bichindaritz, I., Montani, S. (eds.) Case-Based Reasoning. Research and Development, Lecture Notes in Computer Science, vol. 6176, pp. 141–155. Springer (2010)

https://doi.org/10.1007/978-3-642-14274-1_12

My contributions: I developed the ideas, research, and experiments presented in the paper, wrote the initial draft, and incorporated comments and suggested changes into the text.

The Utility Problem for Lazy Learners - Towards a Non-eager Approach

Tor Gunnar Houeland, Agnar Aamodt

Department of Computer and Information Science,
Norwegian University of Science and Technology,
NO-7491 Trondheim, Norway
{houeland,agnar}@idi.ntnu.no

Abstract. The utility problem occurs when the performance of learning systems degrade instead of improve when additional knowledge is added. In lazy learners this degradation is seen as the increasing time it takes to search through this additional knowledge, which for a sufficiently large case base will eventually outweigh any gains from having added the knowledge. The two primary approaches to handling the utility problem are through efficient indexing and by reducing the number of cases during case base maintenance. We show that for many types of practical case based reasoning systems, the encountered case base sizes do not cause retrieval efficiency to degrade to the extent that it becomes a problem. We also show how complicated case base maintenance solutions intended to address the utility problem can actually decrease the combined system efficiency.

1 Introduction

A concern for case-based reasoning (CBR) systems that are deployed and will keep running for many years is how the system will change over time. The capability for learning is an important aspect of many such systems, but by its very nature the act of learning will change the system from its current state to something that is partially unknown. There will normally be a desirable improvement from learning, but its effects may also include unwanted changes. One of these changes is that as the system's knowledge increases, the space needed to store the knowledge and the time it takes to process it also increases. The storage space and time taken to process the knowledge will increase without bounds, and eventually go far beyond the space and time taken by the original system. Because of this behavior, there will always be some theoretical point where the total performance of the system is degraded by adding additional knowledge. Many different methods have been suggested to address this problem, which is often included under the wider umbrella of case base maintenance.

The maintenance methods used to address this problem can be split in two: maintaining the case base indexes, and maintaining the case base contents. Indexing methods work by quickly identifying the relevant parts of the knowledge base, which can allow a system to examine only a small fraction of its available

knowledge. Methods for maintaining case base contents aim to reduce the size of the system's case base, making it both faster to examine since there is less knowledge, and reducing the storage space needed to store the data.

The *utility problem* in learning systems occurs when knowledge learned in an attempt to improve a system's performance degrades it instead [17, 3]. This is most common in speed-up learning systems, where the system's knowledge is used to reduce the amount of reasoning required to solve a problem. For pure speed-up learners it is assumed that there is already a slower method available for finding an acceptable solution to the problem. From a simplified perspective, cases in a CBR system may be viewed as a form of speed-up knowledge, where storing, retrieving, and adapting cases provides for more efficient problem solving than first-principles or model-based methods [15, 9]. The goal is to produce acceptable results more quickly, and hence the time taken to perform the system's reasoning is of primary concern.

Case-based reasoning is also known as a *lazy* approach to learning and problem solving. The very essence of lazy learning is that choices regarding the solution of a problem will be postponed as long as possible, until the problem query is posed and as much information as possible is available. Building index structures and deleting cases from the case base are both eager methods, and hence somewhat counter-intuitive to the CBR idea. Indexing and deletion reduce the amount of knowledge available, without knowing whether that information could have been useful for solving a future problem. Hence, indexing and deletion methods should only be used when they are really needed. In the work reported here we explore the hypothesis that for a wide range of CBR application systems, addressing real world problems, they may not be needed.

This work is situated within our research on a new architecture for meta-level reasoning and introspective learning [10]. A less eager approach to indexing and case deletion, if feasible, will allow more freedom to the meta-level reasoner.

This paper examines the utility problem in CBR as it applies to most of the CBR systems we have built and are building today, with case bases of reasonable sizes. Based on existing literature and several example utility models for case-based reasoners, we show that neither indexing nor case deletion policies are necessary for a wide range of CBR systems, and in fact can be detrimental to a CBR system's overall goal.

In section 2 the background of the utility problem is summarized, and some earlier research results relevant to our work is discussed. This is followed in section 3 by an analysis of the utility concept and a comparison of three different speed-up learner scenarios. Section 4 discusses the use of indexing strategies for speeding up retrieval. Section 5 discusses the benefit of limiting case base size and shows how the cost of advanced maintenance methods may negate the benefits of a reduced case base through an illustrative experiment. Concluding remarks end the paper.

2 Background and related research

A substantial amount of research has addressed the utility problem in CBR [19, 13] Over the past few years there has been a broad range of research addressing specific issues of case deletion, addition and efficient indexing [20, 25, 2]. Wilson and Leake [14] present a thorough examination of the dimensions of maintenance strategies and a survey of maintenance research in terms of those dimensions.

The utility problem is also referred to as the "swamping problem". In their seminal paper on the utility problem in CBR, however, Francis and Ram [9] refer to swamping as one of three types of utility problems in CBR. Swamping is the phenomenon that a single unit of knowledge added to the knowledge base - i.e. a case added to the case base - improves problem solving speed at the individual level, because a more similar case to a query may be found, while the performance over the knowledge base as a whole is degraded due to increased retrieval and matching overhead. Swamping is also referred to as the "core" utility problem, which is probably why the two have become synonyms. The other types of utility problems listed are the "expensive chunks problem" and the "search-space utility problem". The first refers to the problem of performance degradation at the level of individual knowledge units, because of the matching cost of a single unit (a macro operator, for example). The second refers to degradation due to increased complexity of search control knowledge, of particular relevance to the learning of meta-level knowledge. Although all three have relevance for CBR systems, the focus in this paper is on the swamping problem.

The processing power available for modern CPUs continues to increase, continually reducing the problems associated with large amounts of data, and allowing large case bases to be handled which would have been considered impossible for a reasonable budget 10 years ago. However, this is typically only true for polynomial-time algorithms, and especially for algorithms that run in (sub-)linear time. Other algorithms that have an exponential running time are unlikely to ever be practical for large inputs, such as many graph-matching algorithms. This means that the "expensive chunks problem" is unlikely to be greatly affected simply by advances in computer hardware, since they are NP-hard in the worst case [23]. On the other hand these advances affect the degradations experienced due to the swamping problem, since algorithms for searching the knowledge base are typically either O(N) or O(log N).

The main cause of the swamping problem is that retrieval time will increase with a growing case base while adaptation time will decrease. The latter is due to a smaller distance between a query case and the best matching case on average. As the case base grows retrieval time is likely to dominate, however, which leads to a logarithmic or higher increase in processing time. For a speed-up learner this increase may negate the efficiency gains during adaptation and eventually even cause the system to be slower than the underlying "slow" solver. The speed increase has been reported as being substantial in some systems where this logarithmic increase has no significance, for example a thousand-fold increase in CASEY [12]. Other machine learning systems have reported more modest figures, such as a factor up to six in SOAR and ten in PRODIGY/EBL [9]. Systems

with these speed-up figures that are left running and collecting experience for a long time will gradually slow down, and eventually be slower than the unassisted "slow" solver [9].

The trade-off is perhaps most clearly illustrated and explored for some types of control rule learning (CRL) systems, where every individual control rule is guaranteed to have a positive utility (improve performance) but, in concert, also have a negative utility (degrade performance) [9, 17]. Francis and Ram [8] describe a framework for modeling CRL and CBR systems, and for analyzing the utility problem for these reasoning models. The authors identified that the retrieval costs for CBR increase without bound, and that in the limit, CBR systems will get swamped and the cost of retrieval will outweigh the benefits of case adaptation. The authors conclude that CBR is nevertheless relatively resistant to the utility problem, compared to CRL systems, because the cases have the potential to greatly reduce the amount of problem solving needed, and that the cost of retrieval is amortized across many adaptation steps.

Smyth and Cunningham [19] examine the utility problem in CBR through experimenting with a path finding system that combines Dijkstra's algorithm with CBR for speed-up. They show how case-base size, coverage and solution quality affect the utility. The authors find that varying these characteristics significantly alters how the system responds to different case base sizes. This indicates that the need for case deletion and indexing is strongly related to requirements for solution quality and retrieval time. We will discuss this issue later in the paper.

A proposed policy for case deletion with minimal effects on case base competence was presented by Smyth and Keane [20]. A footprint-driven method that also accounted for utility gave the best test results. An alternative method was proposed by Zhu and Yang [25] with the emphasis on careful addition of new cases rather than on deleting old ones. Their method performed better than the footprint deletion method, under some conditions.

A deliberate case addition process may be viewed as a form of case deletion as well (i.e. by not adding cases that might otherwise have ended up in the case base). From the perspective of more or less eager case base maintenance methods, assuming an existing case base and only incremental updates, a considerate case addition policy will generally be a more lazy approach than a deletion approach. An even lazier approach is of course to keep all the cases, and rely on the retrieval and adaptation methods to do the "deletion" on the fly.

One solution to the indexing problem is to apply suitable methods for refining indexing features and matching weights. Jarmulak et al. [11] use genetic algorithms to refine indexing features and matching weights. Another approach is to view this problem from a meta level perspective, and use introspective learning techniques to handle the refinement of indexes, triggered by retrieval failures [7, 4].

We are developing an introspective architecture for lazy meta-level reasoning characterized by creating and combining multiple components to perform the system's reasoning processes [10]. Each component represents part of a reasoning

method and its parameters using a uniform interface, which can be used and modified by the meta-level reasoner. This will enable selecting which reasoning methods to use after a description of the problem to be solved is known.

Although it is known that laziness and indexing strategies impact each other [1], we have not come across work which expressly views indexing also from a maximally lazy perspective. This means to avoid building such indexes at all for the purposes of improving search efficiency, as a way to avoid committing to eager indexing decisions. Indexes may still be built, but in order to improve matching quality only. A typical example is a structure of abstract indexes, which interpret lower-level data in the input cases in order to achieve an improved understanding of the case information and hence more accurate solutions.

Watson [24] discusses case base maintenance for *Cool Air*, a commercially fielded CBR system that provides support for HVAC engineers. The system retrieves cases for similar previous installations, and Watson explains the case-base maintenance required for the system, most notably the removal of redundant cases from a rapidly growing case base. By the nature of the application domain, installing the system in different locations means that the same product will operate under several similar conditions, and result in very similar cases being created. In their client-server design the server selects a small set of cases to be sent to the client, and the client then uses the engineers' custom-tailored similarity measure to rank them. The problem with this design is that many redundant cases are sent to the client, and it was decided to remove the redundant cases from the case base. Although primarily motivated by other purposes than the utility problem, this type of case redundancy avoidance in client-server architectures seems to be generally useful.

3 Describing and analyzing utility

In general, the utility of a reasoning system can be expressed as the benefit it brings, minus the costs associated with the system. The benefits are typically achieved over time while the system is in operation, while a large part of the costs of the system are up-front, such as gathering expert knowledge, developing the system and integrating it with the organization that will be using it. Another large source of costs is the continued maintenance of the system, which is often overlooked but should be included when the system is initially planned [24].

The benefit of a generated solution can be measured as its usefulness for addressing the task at hand, which is primarily characterized by the solution accuracy and solution time. When considered in this manner, the time taken to solve a problem can be naturally expressed as a lessened benefit. Then the direct costs associated with the problem solving are related to the resources spent computing them, which are very small for typical systems.

For clarity, we define our use of the terms as follows:

General system utility: GSU The combined benefit of the reasoning system, minus the associated costs. This consists of the solution usefulness for the solutions generated by the system, the usability of the system for human

operators, and the costs associated with developing, running and maintaining the system.

Solution usefulness: SU The benefit of a generated solution, estimated as a function of solution accuracy, solution time and resource costs, not including human factors.

Solution accuracy: SA The accuracy of a generated solution, which depends on both the case base and the methods used by the system.

Solution time: ST The time it takes the system to solve a problem.

Resource cost: RC The cost of solving a problem. This is primarily the time spent operating the systems, but also includes hardware costs that can potentially be significant for long computations in large systems.

Using the same approach as Smyth and Cunningham [19] for analyzing these concerns, we assume that the solution accuracy increases with a larger case base, and that the solution time is divided into two parts: retrieval time, which increases with a larger case base, and adaptation time, which decreases with a larger case base (or stays the same). By noting that the retrieval time increases with the size of the case base, which is unbounded [17], that the retrieval time is similarly unbounded for any retrieval approach that can potentially reach all the cases, and that the reduction in adaptation time is bounded (since it can never be faster than 0 time units), we see that there will be some point where adding further cases to the case base will slow down the total solution time.

The utility problem is most easily analyzed for speed-up learners, where the solution time is of primary importance. We explore three different scenarios for speed-up learners with different time complexities, and examine the different amounts of utility degradation experienced by modeling the solution time **ST** as a function of case base size N. We use a simplified model of a speed-up learner, where the system will always produce the same correct answer, and the only criterion for the solution's utility will be its solution time. We model the solution usefulness $\mathbf{SU} = 1/\mathbf{ST}$ for such systems, ignoring the solution accuracy.

As reported by Smyth and Cunningham [19], when increasing the size of the case base, the solution time for a case-based speed-up learner will typically consist of an initial rapid improvement, followed by a continuing degradation as the retrieval part begins to dominate the total time spent to solve the problem. For a typical speedup-learner, the total solution time will initially be approximately monotonically decreasing, followed by an approximate monotonic increase, and the optimal solution time and preferred case base size will be where the rate of increase in retrieval time matches the rate of decrease in adaptation time. The exact behavior of the total solution time and the order of magnitude of this preferred case base size depends greatly on the algorithmic complexity classes of the algorithms used, and in general there is no guarantee that the solution time will follow this pattern at all. The exact characteristics also depend on how the case base content is created and maintained, since a maintained case base can have different case distributions and behave quite differently than an approach retaining all cases.

Fig. 1 shows the solution time for an idealized speed-up learner, where retrieval is a linear search through the case base, adaptation time is proportional to the distance to the retrieved case, and there is no overhead: $\mathbf{ST} = N/5 + 100/N$. In this situation the efficiency of the system initially improves quickly, and then starts degrading slowly as the increased time to perform retrieval eventually becomes greater than the time saved during adaptation. In systems with retrieval and adaptation algorithms displaying this kind of behavior, the solutions will be generated most quickly when retrieval and adaptation times are approximately equal, since that coincides with their derivatives having the same magnitude and opposite signs. Smyth and Cunningham [19] report very similar results to this speed-up learner scenario from experimenting with the PathFinder system.

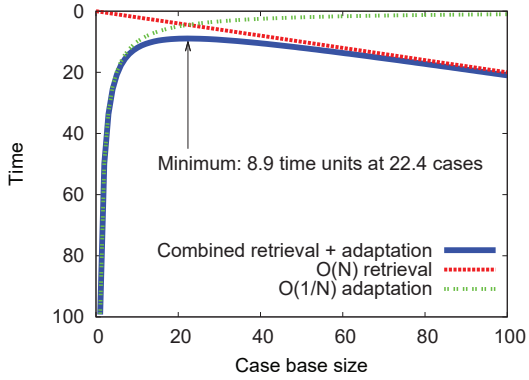

Fig. 1. Retrieval, adaptation and combined solution time compared to case base size for a speed-up learner using an O(N) retrieval algorithm and a O(1/N) adaptation algorithm.

Fig. 2 shows the solution time for similar speed-up learning systems, but using different algorithms. The graph on the left uses an indexing scheme for retrieval that causes the retrieval step to run in O(log N) time, and with a comparatively larger constant factor (5 vs 0.2) than the previous example: $\mathbf{ST} = 5 * ln(N) + 100/N$. With this change to the retrieval function, the slowdown associated with adding more cases to the case base happens very slowly, and the total solution time for the full case base is just 20% slower than the minimum solution time, even though the full case base is 5 times larger. The graph on the right shows a much more drastic increase in solution time. A more complicated O(N log N) case retrieval algorithm is shown that compares the retrieved cases against each other, and case adaptation has a significant overhead of 20 time units: $\mathbf{ST} = N * ln(N) + 100/N + 20$. In this situation the combined solution time increases quickly as more cases are added beyond the optimal amount. For

domains where this type of algorithm is desirable, a similar increase in solution quality would be expected, otherwise the case base should be kept relatively small through aggressive case base content maintenance. Due to the very limited number of cases the system can handle before slowing down, these latter types of algorithms appear to be a poor choice for pure speed-up learners, although the constant factors could potentially be of very different magnitudes for some domains. These alternative combinations scale very differently, and illustrate the importance of examining the algorithms used when analyzing the effect of larger case bases.

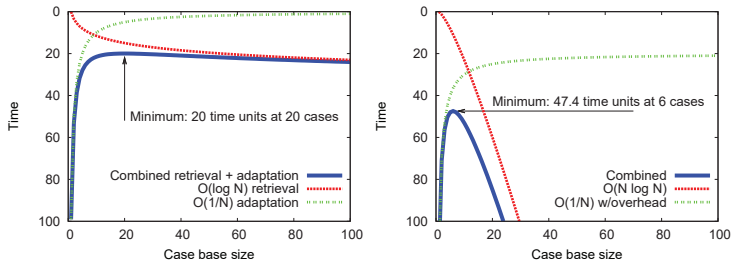

Fig. 2. Retrieval, adaptation and combined solution time compared to case base size for two other speed-up learner scenarios: On the left, using an O(log N) retrieval algorithm and a O(1/N) adaptation algorithm. On the right, using an O(N log N) retrieval algorithm and a O(1/N) adaptation algorithm with overhead.

For general case-based reasoners, the utility function becomes much more complicated. When still examining only the usefulness of solving one problem while the system is running, and just moving away from the simplified speed-up learner model, we need to also include the accuracy of the solution in our evaluations. The impact the accuracy has on the usefulness of the system will vary greatly based on the domain and the specifics of the application.

Smyth and Cunningham [19] report empirical results from the PathFinder system, where at one point the quality of solutions increased from 94% to 96%, while the solution time increased by 50%. Whether such a trade-off is considered beneficial or not depends on the application and the initial starting values for the retrieval accuracy and solution time. For a speed-up learner this might be unacceptable, while for many other applications a 33% reduction in errors (from 6% to 4%) at the expense of waiting longer or using two computers instead of one would be a great improvement. As in the fielded *Cool Air* system, the solution time might simply be considered acceptable and not be a problem that has to be addressed at all, and then a larger case-base might be purely beneficial without encountering this trade-off.

4 Indexing vs. no indexing

There are many possible indexing strategies for speeding up retrieval, and their effects are highly dependent on the domain and the similarity measures used. Selecting a good indexing strategy can often require significant expert knowledge, although various automated methods exist [11, 7, 4] for refining the indexing strategy. This can be particularly helpful for index maintenance, since an appropriately chosen method can potentially handle most indexing maintenance operations without manual intervention.

However, from a lazy learning perspective, an indexing structure that allows the retrieval method to only consider a subset of the case base is an eager optimization, which is made before all the potentially useful information about the target problem is known, and is therefore not always appropriate.

In systems that handle thousands of cases or less, the processing time is not necessarily a critical factor, and might very well increase slower than the increase in processing power available over time, i.e. the solution usefulness **SU** is primarily a function of the solution accuracy **SA**. This is particularly true for knowledge-intensive CBR, where less than a hundred cases is common (but the time required to perform individual similarity measurements is often extensive).

By matching with every case in the case base, the retrieval method is guaranteed not to miss any cases in the case base, and without needing to regularly maintain the indexing structures.

Unlike pure speed-up learners, producing results as quickly as possible is rarely the main concern for fielded CBR systems. The cost of developing and maintaining a system is usually much larger than the cost of using and maintaining the hardware it runs on, and the direct resource cost **RC** can sometimes even be considered negligible. CBR systems with extensive reasoning also do not usually act as speed-up learners, since they can actually produce better solutions with a larger case base. For such systems the utility problem is a trade-off between solution quality and the efficiency degradation associated with a large case base [19].

As an alternative to purely eager indexing, footprint-based retrieval [21] allows for a kind of mix of indexing and similarity measurements. The indexing is eagerly pre-generated based on competence evaluations performed before the input problem is known. During retrieval, the index is used to quickly identify what is believed to be the most relevant knowledge items, which are then evaluated lazily with full similarity-based knowledge. Although the retrieval is less efficient than purely eager methods, this partially lazy approach can produce good results for some domains.

For the commercially fielded *Cool Air* [24] CBR system, processing time was much cheaper for the company than consultancy time for developing the system. The efficiency slow-down associated with an increasing case base did not become a problem, even though the case base doubled over two years.

In another commercially fielded system, the DrillEdge system for fault prediction in oil well drilling [22], case retrieval is an automatic process triggered by a pattern in the continuous stream of drilling data. The cases are indexed by

abstract features derived from the numerical drilling data stream. This is done in order to improve the matching process for retrieval of the most relevant cases. The indexes are not used to improve retrieval efficiency - the case base is always searched sequentially. As long as the number of cases is in the range of hundreds, this is not regarded as a performance problem.

For some applications, like the one just mentioned, solution quality is of utmost importance. In oil well drilling, the costs of time lost due to unwanted events can be huge, since drilling operations typically cost around 200 000 USD per day [18]. In this case the value of the positive utility associated with higher quality solutions is of a different order of magnitude than typical costs of negative utility caused by decreased efficiency. For the knowledge-intensive oil well drilling system, the main cost of a large case base is the amount of expert knowledge required, not the computer systems it runs on.

As an alternative to performing eager indexing at all, a two-step approach [6] to case retrieval has often been employed for systems with expensive retrieval operations, e.g. for knowledge-intensive CBR systems. This consists of first using a fast and resource-efficient scan through the case base to identify relevant knowledge, and then performing more advanced (and comparatively slow) reasoning for this restricted set of cases. This is conceptually very similar to indexing, but is done using a lazy approach, entirely after the input problem query is known.

In this way, there is no need to update indexing structures, and more powerful methods can be performed for identifying relevant knowledge when you already know the problem to be solved. Similarity assessment is usually very important for CBR systems, because there is often no easy way to model the structure of the entire problem space, and there may even be no expert knowledge that directly applies to all problem instances in general. Using similarity measurements to locally identify relevant knowledge for a specific problem is thus likely to produce better results than pre-generated structures.

To perform large numbers of similarity assessments quickly, it might be necessary to increase the amount of computational resources available by examining the cases in the case base in parallel. Many modern distributed computing frameworks available for processing very large data sets in parallel are based around ideas similar to the MapReduce [5] algorithm. MapReduce works by first chopping up a problem into many parts, then distributes each of these parts across a cluster of computers and each node processes only a subset of the problems. The answers are then returned to a master node, which combines them to create a final answer for the entire problem. This is very similar to parallel case retrieval, where each case is assigned a similarity score and then ranked at the end.

While this form of case evaluation producing independent results for every query-to-case comparison does not let us express the most general forms of case retrieval, they are sufficient for most systems that are used in practice. The kind of similarity assessment methods supported by this approach are also typically more flexible than those supported by common indexing schemes. Avoiding the need for additional expert knowledge that is often required to create a good indexing solution is another potential benefit of this approach.

For commercial applications, these kinds of large parallel processing frameworks are typically used to process terabytes or petabytes of data, and provide a possible means to perform full sequential evaluations for the complete case base during retrieval even for very large case bases.

5 Case base maintenance

Case-based reasoning system maintenance is important and can involve processes that modify all parts of the system, including updates to each of the knowledge containers. Most often this includes reducing the number of cases in the case base, which is primarily useful for two purposes:

- Reducing the size of the case base used by retrieval methods, which can make retrieval faster.
- Reducing the space required for storing the case base.

Various case base content maintenance algorithms exist for reducing the size of the case base, while optimizing the remaining cases according to some criteria. A fast and simple content maintenance strategy is to delete cases at random, which has been reported to produce good results [16]. Since the case base essentially becomes a random subset of all encountered cases, or effectively just a smaller case base, this strategy also has the added benefit of maintaining the same case distribution as the encountered cases, on average. Other approaches for content maintenance usually examine the relations between cases in the case base, and e.g. attempt to maximize the coverage of the remaining cases in the reduced case base through adding, deleting or combining cases [20].

We conducted a set of experiments to compare these two approaches, using a random set of cases versus the coverage-based case addition algorithm proposed by Zhu and Yang [25] as the content maintenance strategy. The results shown in figs. 3-5 are the average from running each test 10 times. Cases were described by 5 features, each with values ranging from 0 to 1, and new cases were picked uniformly from this 5-dimensional space. Euclidean distance was used as the basis for the similarity measure, and a case was considered to be solvable by another case for the purpose of competence evaluation if the distance between the two cases was less than 0.25. We used the same similarity measure to estimate the solution accuracy **SA** on the basis of the distance between the retrieved case and the query, which is optimistic and more advantageous for the maintained case base strategy than a real world scenario, since the competence evaluations will be flawless. Thus using larger case bases can be expected to usually be at least as good compared to this kind of computationally expensive content maintenance strategy for real-world systems as in the experiments.

Fig. 3 shows the estimated coverage and error for an optimized case base of size N compared to a case base consisting of N random cases. The case base generated by the case addition algorithm has higher resulting coverage (measured as the covered proportion of new queries randomly generated from the underlying domain, which competence-driven maintenance strategies seek

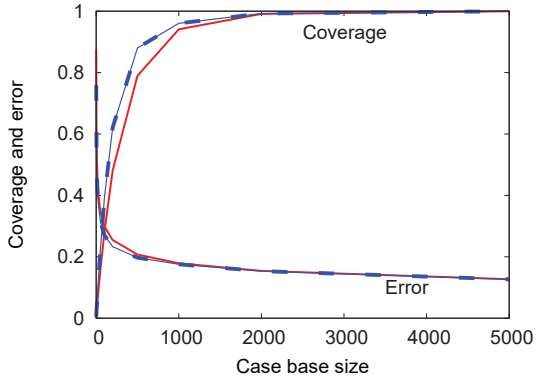

Fig. 3. Coverage and error comparing retaining the full case base of size N (*straight lines*) and using a content maintenance strategy (*dotted lines*) to create a case base of size N based on a larger initial set of 5000 cases. Higher coverage and lower error is better. The time required to perform the maintenance is not considered. In this setting the maintenance strategy outperforms retaining all cases, with higher coverage and slightly lower error.

to optimize), and lower error (measured as the average distance from the best retrieved case to randomly generated new queries). Only the sizes of the case bases are considered, and the computations required to perform the maintenance operations are ignored. Approximately this situation can occur when there are established procedures to run case base maintenance while the system is not being used, e.g. at night, during weekends or during vacations.

However, the computational costs of running case base content reduction algorithms can be extensive. Figs. 4 and 5 show the coverage and error rates for the same two case base content maintenance strategies, but compared according to the time required to perform both maintenance and retrieval. This was examined by running experiments for many different combinations of initial and reduced case base sizes, and choosing the Pareto efficient combinations that gave better results than any faster combinations. The size of the resulting reduced case base size used for retrievals is included in the figures. For each data point the case base maintenance was run only once, and its potentially costly computation was amortized over a large number of retrievals. However, this maintenance cost can still be very high, depending on the number of retrievals performed compared to maintenance operations. The examples shown in the figures consists of an up-front case maintenance step followed by 1000 and 10000 retrievals respectively (chosen as examples of large numbers of retrievals, since more retrievals favors the maintenance strategy), and shows the combined time for these operations. Even with this relatively large number of retrievals, the simpler strategy of retaining all cases generally performs as well or better

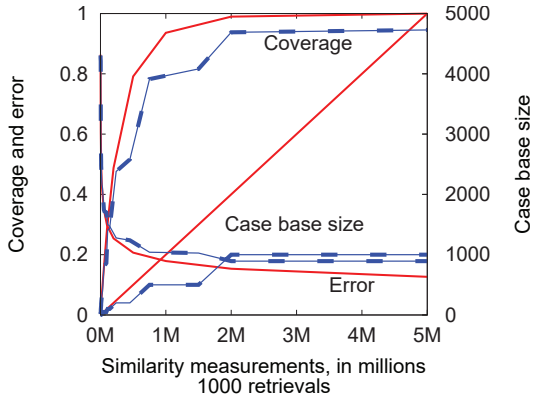

Fig. 4. Coverage and error shown according to the amount of computation required (measured as number of similarity measurements), when retaining the full case base (*straight lines*) and using a content maintenance strategy (*dotted lines*). For 1000 retrievals, the larger case bases supported by not having a maintenance cost means that the strategy of retaining all cases performs better, with higher coverage and lower error.

than the content maintenance strategy, due to supporting larger case bases in the same time frame. This means that using a maintenance strategy to reduce the case base size for efficiency reasons may sometimes be counter-productive, in addition to size reduction being an eager strategy that limits the potential options available for further problem solving.

The other aspect of reducing the number of cases in the case base is the reduced storage capacity required to hold the case base. Current computer systems intended for personal use can store hundreds of gigabytes of data, which is much much larger than many typical CBR application case bases. Maintaining the set of cases exposed to the retrieval method can be a very useful approach for some applications, but the case base used for retrieval at any given moment does not have to be the full set of cases archived by the system.

Based on this observation, we conclude that many practical CBR system can instead flag the cases as no longer being active and store them in another location that is not searched by the retrieval methods, since conserving disk space is not required for systems that do not generate vast amounts of data. In these situations the archival storage can be done at negligible cost, and provide the advantage that deletions are no longer completely irreversible. During later system maintenance some time in the future, the reason for the original deletion may no longer be relevant or the algorithms used by the system may have changed, and in such cases it would be beneficial to be able to undo such eager deletion optimizations, in the spirit of lazy learning.

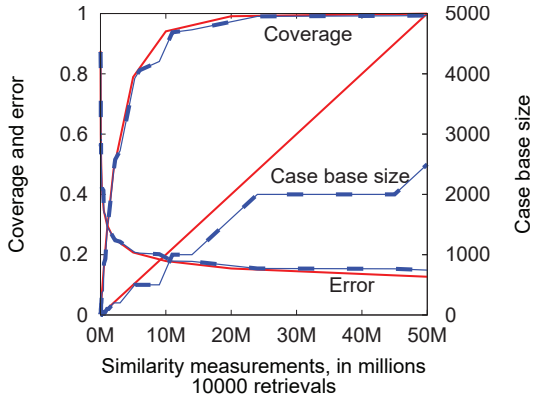

Fig. 5. Even when performing 10000 retrievals, the strategy of retaining all cases generally performs slightly better, with higher coverage and lower error.

6 Conclusions

In this paper we have examined the utility problem from a lazy learning perspective, as it applies to speed-up learners and general case-based reasoners. The two primary approaches to addressing the utility problem are through indexing and by reducing the size of the case base itself during case base maintenance.

These approaches are eager compared to the lazy core CBR process, and we have shown how many practical CBR systems do not require the use of these eager optimizations and can be limited by committing to decisions prematurely.

References

1. Aha, D.W.: The omnipresence of case-based reasoning in science and application. Knowledge-Based Systems 11, 261–273 (1998)
2. Bergmann, R., Richter, M., Schmitt, S., Stahl, A., Vollrath, I.: Utility-oriented matching: A new research direction for case-based reasoning (2001)
3. Chaudhry, A., Holder, L.B.: An empirical approach to solving the general utility problem in speedup learning. In: IEA/AIE '94: Proceedings of the 7th international conference on Industrial and engineering applications of artificial intelligence and expert systems. pp. 149–158. Gordon and Breach Science Publishers, Inc., Newark, NJ, USA (1994)
4. Cox, M.T.: Multistrategy learning with introspective meta-explanations. In: Machine Learning: Proceedings of the Ninth International Conference. pp. 123–128. Morgan Kaufmann (1992)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation. pp. 137–150. USENIX Association, Berkeley, CA, USA (2004)

6. Forbus, K.D., Gentner, D., Law, K.: Mac/fac: A model of similarity-based retrieval. Cognitive Science 19(2), 141–205 (1995)
7. Fox, S., Leake, D.B.: Combining case-based planning and introspective reasoning. In: Proc. of the 6th Midwest Artificial Intelligence and Cognitive Science Society Conference. pp. 95–03 (1995)
8. Francis, A., Ram, A.: A comparative utility analysis of case-based reasoning and control-rule learning systems. In: In Proceedings of the Eighth European Conference on Machine Learning. pp. 138–150. Springer (1995)
9. Francis, A.G., Ram, A.: The utility problem in case based reasoning (1993)
10. Houeland, T.G., Aamodt, A.: Towards an introspective architecture for meta-level reasoning in clinical decision support systems. ICCBR 2009, 7th Workshop on CBR in the Health Sciences (2009)
11. Jarmulak, J., Craw, S., Rowe, R.: Genetic algorithms to optimise cbr retrieval. In: EWCBR. pp. 136–147 (2000)
12. Koton, P.A.: A method for improving the efficiency of model-based reasoning systems. Hemisphere Publiching Corporation (1989)
13. Leake, D.B., Smyth, B., Wilson, D.C., Yang, Q.: Introduction to the special issue on maintaining case-based reasoning systems. Computational Intelligence 17(2), 193–195 (2001)
14. Leake, D.B., Wilson, D.C.: Categorizing case-base maintenance: Dimensions and directions. In: Advances in Case-Based Reasoning:Proceedings of the Fourth European Workshop on Case-Based Reasoning. pp. 196–207. Springer-Verlag (1998)
15. de Mántaras, R.L., McSherry, D., Bridge, D.G., Leake, D.B., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K.D., Keane, M.T., Aamodt, A., Watson, I.D.: Retrieval, reuse, revision and retention in case-based reasoning. Knowledge Eng. Review 20(3), 215–240 (2005)
16. Markovitch, S., Scott, P.D.: The role of forgetting in learning. In: Proceedings of The Fifth International Conference on Machine Learning. pp. 459–465. Morgan Kaufmann, Ann Arbor, MI (1988)
17. Minton, S.: Quantitative results concerning the utility of explanation-based learning. Artif. Intell. 42(2-3), 363–391 (1990)
18. Shokouhi, S.V., Aamodt, A., Skalle, P., Sørmo, F.: Determining root causes of drilling problems by combining cases and general knowledge. In: McGinty, L., Wilson, D.C. (eds.) ICCBR. LNCS, vol. 5650, pp. 509–523. Springer (2009)
19. Smyth, B., Cunningham, P., Cunningham, P.: The utility problem analysed - a case-based reasoning perspective. In: Proceedings of the Third European Workshop on Case-Based Reasoning. pp. 392–399. Springer Verlag (1996)
20. Smyth, B., Keane, M.T.: Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence. pp. 377–382. Morgan Kaufmann (1995)
21. Smyth, B., McKenna, E.: Footprint-based retrieval. In: In Proceedings of the Third International Conference on Case-Based Reasoning. pp. 343–357. Springer Verlag (1999)
22. Sørmo, F.: Real-time drilling performance improvement. Scandinavian Oil & Gas Magazine, No. 7/8 2009 (2009)
23. Tambe, M., Newell, A., Rosenbloom, P.S.: The problem of expensive chunks and its solution by restricting expressiveness. Machine Learning 5, 299–348 (1990)
24. Watson, I.: A case study of maintenance of a commercially fielded case-based reasoning system. Computational Intelligence 17, 387–398 (2001)

25. Zhu, J., Yang, Q.: Remembering to add: Competence-preserving case-addition policies for case-base maintenance. In: IJCAI'99: Proceedings of the 16th international joint conference on Artifical intelligence. pp. 234–239. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)

Paper C
An Efficient Random Decision Tree Algorithm for Case-Based Reasoning Systems

Author: Tor Gunnar Høst Houeland

Published in: Murray, R.C., McCarthy, P.M. (eds.) Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, pp 401-406. AAAI Press (2011)

https://aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/view/2639

My contributions: I developed the RDT algorithm, research, and experiments presented in the paper, and wrote the text (sole author).

An Efficient Random Decision Tree Algorithm for Case-Based Reasoning Systems

Tor Gunnar Houeland
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
houeland@idi.ntnu.no

Abstract

We present an efficient random decision tree algorithm for case-based reasoning systems. We combine this algorithm with a simple similarity measure based on domain knowledge to create a stronger hybrid algorithm. This combination is based on our general approach for combining lazy and eager learning methods. We evaluate the resulting algorithms on a case base of patient records in a palliative care domain. Our hybrid algorithm consistently produces a lower average error than the base algorithms.

Introduction

Lazy learning approaches do not draw conclusions until it is necessary, allowing them to collect all available information before doing any generalization. This has the potential advantage of including highly relevant information that an eager approach would not have access to, and adapting the reasoning to the particular characteristics of the problem query to solve. The drawback is that the system's reasoning (and computations) will only be performed all at the very end when an answer is required. Eager methods have the advantage that parts of their reasoning can be precomputed during training, and they only need to store abstract generalizations which can typically take only a fraction of the storage space.

In this paper we examine a hybrid approach that uses a modified version of an eager method (random decision trees) that can be partially precomputed and partially adapted to the particular problem query. We examine the results for determining similarity in a data set describing the results of palliative care for cancer patients, which is an ongoing topic of investigation in our research group.

For our random decision tree (RDT) algorithm the forest of random trees can be grown once before the system begins reasoning about cases, and we use internal data structures that can be incrementally updated in an efficient manner when new cases are added to the case base. The data structures additionally support efficiently considering only selected subsets of the case base as training data at runtime. This capability is combined with a simple similarity measure based on domain knowledge to simulate having trained the algorithm on only the most relevant cases in a computationally inexpensive manner. This high computational efficiency and ability to be integrated with other uses of cases are the primary strengths of our RDT algorithm.

In the next section we summarize some earlier research results and relate it to our work. This is followed by a description of our RDT-based experiment in the domain of palliative care for cancer patients. We describe and compare 4 different algorithms and discuss empirical results for running the algorithms on a case base of palliative care patients. Concluding remarks end the paper.

Related Research

The topic of indexing cases beforehand to support efficient retrieval during problem solving has been extensively studied in the literature. A popular form of indexing structure is a tree with cases at the leaf nodes. One early example of such a tree-based indexing structure used in case-base reasoning is a kd-tree (Wess, Althoff, and Derwand 1994), which partitions the indexing space into disjoint areas. Each leaf node is called a $bucket$, and contains the cases within a particular area of the indexing space.

An ensemble method combines multiple models to produce a better result than any individual model would. This approach has also been used for indexing trees, where multiple trees are created and combined to address a single problem. Perhaps the most well-known example of such a tree ensemble is the Random Forest (RF) classifier (Breiman 2001). RF grows a number of decision trees based on bootstrap samples of the training data. For each node of a tree, m variables are randomly chosen and the best split based on these m variables is calculated based on the bootstrap data. Each decision tree results in a classification and is said to cast a vote for that classification, and the ensemble classifier returns the class that received the most votes. RF can also compute $proximities$ between pairs of cases that can be used for clustering and data visualization, and as a similarity measure for case-based reasoning. We use a similar concept of $proximity$ to measure similarity in our random decision tree algorithm.

Diversity is an important aspect of an ensemble classifier that affects its accuracy. Bian and Wang (2007) found that the performance of an ensemble learning approach varied considerably for different applications. They studied homo-

geneous and heterogeneous ensembles and found connections between diversity and performance, and an increased diversity for heterogeneous ensembles.

Gashler et. al. (2008) examined ways to increase the diversity for ensembles of decision trees. They compared an approach that split on randomly selected attributes as a means to increase diversity, with an approach that combined multiple tree algorithms to increase diversity. For our random decision tree algorithm we similarly use entirely random attributes, and also perform the splits at random.

Ferguson and Bridge (2000) describe a generalization of similarity measures they call similarity metrics, and build upon a way of combining similarity metrics called prioritization. This approach uses an indifference relation to prioritize a secondary metric when no significant difference is found based on the primary similarity metric. In our approach we also use two similarity measurements, but they are integrated in a hybrid combination.

Richter (1995) introduced the knowledge container model for CBR systems. In this model the system's knowledge is categorized into four categories: case vocabulary, cases, similarity assessment and adaptation knowledge. A distinction was also made between compiled knowledge, which is "compiled" in a very general sense before actual problem solving begins, and interpreted knowledge that is assessed at run time, during the process of problem solving. In our research we integrate the similarity assessment with the internal storage of cases, and develop a method for efficiently "compiling" the similarity knowledge for an ensemble of random decision trees.

The utility problem occurs when additional knowledge learned decreases a reasoning system's performance instead of increasing it (Minton 1990; Smyth and Cunningham 1996). Theoretically this will always occur for a CBR system when the system's case base increases without bound, and has therefore been the subject of considerable research, since the problem remains relevant even as computers become faster and cheaper.

Patterson et. al. (2003) examine efficient retrieval as a means to reduce the effects of the utility problem. They present two indexing approaches that give efficiency gains of up to 20 times faster retrieval, with a small reduction in case base competency. As a similar trade-off, our research focuses on a highly efficient method for lazy reasoning, with a limitation on the type of decisions that can be performed to determine similarity.

In an earlier study (Houeland and Aamodt 2010), we suggest that the usefulness of an optimization should be measured by the effect it has on the reasoning system's overall utility. We continue this line of reasoning in the research presented here, by examining the trade-off between accuracy and speed that occurs in the developed algorithms.

Random Decision Tree Experiment

Our random decision tree (RDT) algorithm is an example of a general approach to combining machine learning methods with case-based reasoning. Like a traditional CBR approach we retrieve the local cases nearest to our input query (shown in figure 1), but we retrieve an unconventionally large number of cases (in the following experiment we retrieve half the case base). We run a machine learning algorithm on this subset of cases as training data, partially combining the lazy and local attributes of a CBR retrieval with the eager and global methods often used for more traditional machine learning algorithms.

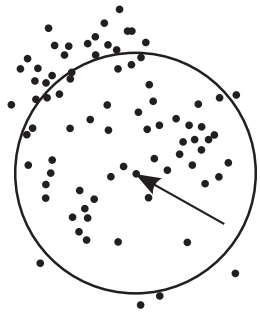

Figure 1: Selecting a similarity-based local case subset for use as training data. The cases within the circle are retrieved as the closest neighbors of the marked case, and are used to train an independent learning algorithm.

This combination is based on the intuition that CBR similarity measures can often easily express rough case relevance estimates based on domain knowledge, while eager machine learning algorithms are typically very effective at selecting precisely the best option among possible alternatives, but without the beneficial features of laziness.

A drawback of this approach is that a straightforward application of an eager method would have to be trained from scratch on the case subset during problem solving. This can be prohibitively expensive because eager methods typically perform a lot of computations that are normally amortized over many subsequent problem solving sessions.

We develop an RDT variant where the storage of the tree knowledge is inverted, by associating each case with its result for every tree and storing the knowledge with the cases. This is in contrast to the more traditional approach of creating the tree structure based on the training data and implicitly storing the knowledge in the trees. Our variant allows very efficient incremental updates for additional training data, i.e. learning new cases one by one.

In this experiment we use a forest of randomly grown trees to determine the similarity between two cases. Each tree is a fully grown binary tree of height 5, where one particular measurement is compared to a threshold value at each node.

An example tree generated by our implementation is shown in figure 2. The root node "Addiction $<$ 0.5739" in the example compares the patient's *addiction* value to the specified threshold 0.5739, and continues down the left branch marked with a dotted line in the figure if it's below the threshold value and otherwise down the right branch marked with a solid line.

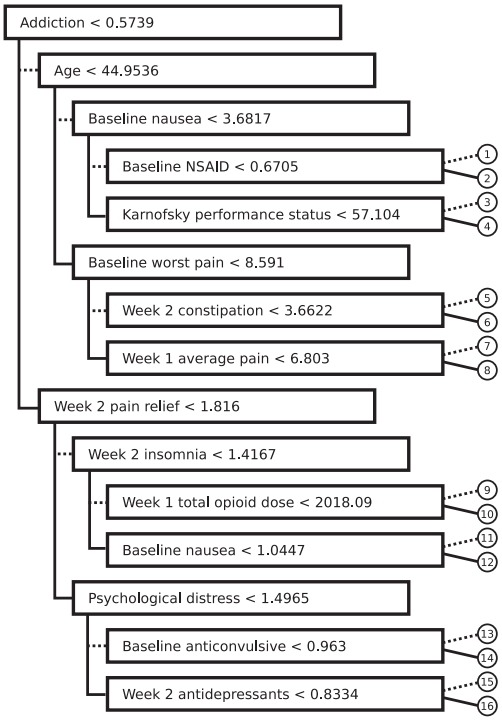

Figure 2: A randomly generated decision tree for the palliative care domain.

Each tree sorts a case into one of 16 leaf node $buckets$, or no bucket if a node would compare a measurement the case does not include (because it is a partial patient record). The forest of trees acts as a measure of similarity between cases, where two cases are said to be more similar if they're in the same bucket for a higher number of trees.

This is the same as the $proximity$ value uses in the Random Forest (RF) classifier, but with a different algorithm for growing trees. While the RF classifier generates strong trees based on the training data, we generate completely random trees.

This aspect is more similar to Random Decision Trees (Fan et al. 2003) and the Max-diverse Ensemble (Liu, Ting, and Fan 2005) algorithm, which was shown to have nearly comparable accuracy to RF but without using bootstrap samples based on the training set.

The important advantage of completely random trees for our experiment is that growing the trees does not depend on the training data, which means that a single forest can be generated once and used for case bases with different sets of case instances. This allows us to keep the same precomputed trees when adding a case to the case base, and only incrementally update the data structures that are used to represent the cases.

Algorithms

For each new case we go through all the trees and compute which $bucket$ the case belongs to, and for each case we store these computed bucket values for each tree. The advantage of this representation is that the proximity of two cases can be computed by only iterating through the stored bucket values which can be implemented efficiently.

For our experiments we are interested in combining this knowledge-lean random decision forest method with a simple CBR-like similarity measurement based on variables medical doctors consider relevant for pain classification. This similarity measurement is the sum of the normalized differences of patients' *pain intensity*, *breakthrough pain*, *pain mechanism*, *psychological distress*, *cognitive functioning*, *addiction* and *pain localization*.

We compare 4 different similarity methods: the random decision forest, the least difference in relevant variables, a completely random approach, and a hybrid approach based on the random decision forest plus the differences in relevant variables.

Unfortunately the European Palliative Care Research Collaborative (EPCRC) has not been able to reach a consensus on how to classify pain, so there is no clear "solution" or "answer" for our cases. We have decided to estimate the similarity by comparing the difference in *worst pain* and *average pain* experienced after three weeks and basing our comparisons only on the data acquired before three weeks had passed.

Our data set consists of 1486 cases with 55 numerical features from the first two weeks as the problem description, and these two pain classifications as the solution. This is relevant because the main palliative treatment is started during week 2 and the pain experienced afterwards depends on the treatment and is of utmost importance for a patient receiving palliative care. Our presented research uses the difference in pain levels as a means to indirectly estimate the correctness of the computed similarity.

We evaluate a similarity by measuring this combined difference in *worst pain* and *average pain* between the input query and the case in the case base that was determined to be the most similar, in effect measuring the performance of using the similarity measure for a 1-NN classifier. We simulated problem solving by going through each of the patient cases in order, attempting to solve them using the cases in the case base so far and then adding the case to the case base.

The observed results are sensitive to both random chance and the order of the cases in the case base. To achieve a fair comparison we generated 100 different versions of input where the order of the patient data cases had been randomly shuffled, and used the same set of 100 input orders for each similarity method. We compared the average measured difference for all problem solving attempts for the 100 different orders between the different similarity methods (excluding the first case in each order, when the case base is empty). This approach was used to empirically evaluate 4 different similarity methods for our domain, which are presented in increasing order of complexity.

Algorithm 1 RANDOM-SIMILARITY

```
1.  Cases ← EMPTY
2.  for each patient p ∈ PATIENTS
3.    do q ← CREATE-INPUT-QUERY(p)
4.       x ← RANDOM-INTEGER(1, length[Cases])
5.       best-case ← cases[x]
6.       ▷ Use best-case as the solution for query q
7.       APPEND(Cases, p)
```

RANDOM-SIMILARITY simply chooses a random case from the case base and naturally gives the worst results and an average error of about 4.98, but is a baseline to work against that signifies no correlation to real similarity. Like the other algorithms it is used to select a case *best-case* as the solution (line 6) in a case-based reasoning system, which is afterwards learned and added to the case base before the next query is received. Line 3 extracts the problem description for a case, which in our domain is the patient data from the first two weeks.

Algorithm 2 LEAST-DIFFERENCE

```
1.  Cases ← EMPTY
2.  for each patient p ∈ PATIENTS
3.    do q ← CREATE-INPUT-QUERY(p)
4.       closest ← ∞
5.       best-case ← NIL
6.       for each case c ∈ Cases
7.         do diff ← CBR-DIFFERENCE-MEASURE(q, c)
8.            if diff < closest
9.              then closest ← diff
10.                  best-case ← c
11.      ▷ Use best-case as the solution for query q
12.      APPEND(Cases, p)
```

LEAST-DIFFERENCE is a straight-forward CBR system that uses the simple CBR-DIFFERENCE-MEASURE based on the normalized differences in the 7 variables a medical doctor expected to be relevant. Using this method as essentially a 1-NN classifier gives an average error of about 4.55.

Algorithm 3 N-RANDOM-TREES

```
1.  Cases ← EMPTY
2.  Buckets ← EMPTY
3.  Trees ← GENERATE-TREES(N)
4.  for each patient p ∈ PATIENTS
5.    do q ← CREATE-INPUT-QUERY(p)
6.       Buckets[q] ← COMPUTE-TREE-BUCKETS(q, Trees)
7.       most-similar ← (−∞)
8.       best-case ← NIL
9.       for each case c ∈ Cases
10.        do sim ← COMPUTE-PROXIMITY(Buckets[q],
           Buckets[c])
11.           if sim > most-similar
12.             then most-similar ← sim
13.                  best-case ← c
14.      ▷ Use best-case as the solution for query q
15.      APPEND(Cases, p)
```

N-RANDOM-TREES is based on our approach for efficiently using random decision trees in a CBR system. The GENERATE-TREES procedure builds N fully grown binary trees of height 5, which is stored as an array of 15 *attribute-threshold* pairs that represent the nodes in the tree. When generating a tree, the attribute to compare at each node is selected randomly, and the threshold is set to a random value chosen from that attribute's range of possible values. For our experiment we do not have the domain knowledge to determine the true distribution of possible attribute values. For simplicity we choose to select uniformly random values between the highest and lowest values that are contained in the data set.

The COMPUTE-TREE-BUCKETS procedure computes the resulting $bucket$ for each of the N trees and returns the previously described N-value representation that is used for efficient comparisons. The COMPUTE-PROXIMITY procedure iterates through the bucket values for two cases and returns the number of matching values, i.e. the number of trees where the two cases end up in the same leaf node.

Algorithm 4 N-HYBRID

```
1.  Cases ← EMPTY
2.  Buckets ← EMPTY
3.  Trees ← GENERATE-TREES(N)
4.  for each patient p ∈ PATIENTS
5.    do q ← CREATE-INPUT-QUERY(p)
6.       Buckets[q] ← COMPUTE-TREE-BUCKETS(q, Trees)
7.       Distance ← EMPTY
8.       for each case c ∈ Cases
9.            do Distance[c] ← CBR-DIFFERENCE-
           MEASURE(q, c)
10.      Closest-Cases ← The closest half of Cases sorted
         according to Distance
11.      most-similar ← (−∞)
12.      best-case ← NIL
13.      for each case c ∈ Closest-Cases
14.         do sim ← COMPUTE-PROXIMITY(Buckets[q],
         Buckets[c])
15.            if sim > most-similar
16.              then most-similar ← sim
17.                   best-case ← c
18.      ▷ Use best-case as the solution for query q
19.      APPEND(Cases, p)
```

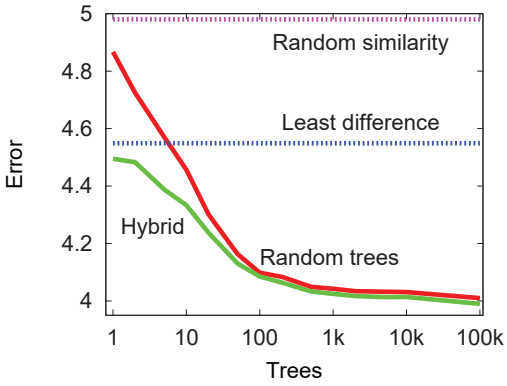

Figure 3: Measured error in the palliative care domain compared to the number of trees for our 4 algorithms. (Lower error is better.)

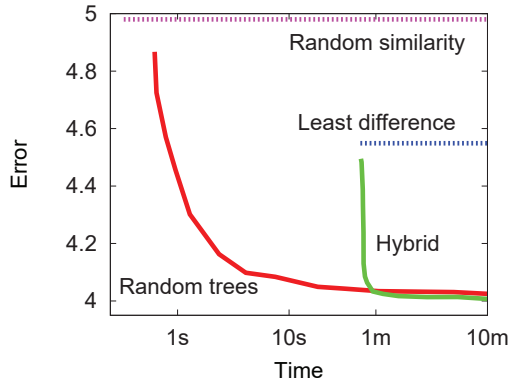

Figure 4: Measured error in the palliative care domain compared to computational time required for our 4 algorithms. (Lower error is better.)

N-HYBRID is a hybrid combination of the LEAST-DIFFERENCE approach and our efficient random decision tree implementation. For a given problem query it considers only the closest half of the case base, as measured by the CBR-DIFFERENCE-MEASURE function. This hybrid approach is proposed as a general way to combine similarity measures based on domain knowledge with knowledge-lean methods, and can be straight-forwardly adapted to ensembles of classification trees.

In our domain, the CBR-DIFFERENCE-MEASURE acts as a domain knowledge-based guard against spurious similarities detected by the random trees. Our random forest implementation performs an unguided similarity comparison, without considering whether that similarity applies for classifying pain or not. The pain treatment domain knowledge present in the CBR-DIFFERENCE-MEASURE function complements the ”raw” similarity computed by the trees, combining the two diverse knowledge sources in a CBR system in a way that has some similarities to the approaches used for very heterogeneous combinations in ensemble classifiers. Just as for ensemble classifiers these combinations do not necessarily improve performance for CBR systems if the knowledge sources do not complement each other.

In our experimental setup where the first similarity measure selects a local case subset that contains a great number of cases (half the original cases), the effect of the second similarity measure that reduces this down to a single case will be larger than the first. Because of this the combination is most successful when the stronger tree-based approach is used as the second similarity measure. It is possible to use the tree-based approach to reduce the case base and then the CBR-DIFFERENCE-MEASURE function to select the single nearest case, but this does not produce as good results for our domain, with an error of around 4.3 (depending on the number of trees). This is as expected, because our hybrid approach improves the results compared to only using the second similarity measure. For a large number of trees the error from the N-RANDOM-TREES algorithm is significantly lower than for LEAST-DIFFERENCE, and thus the starting point for the hybrid combination is better.

Results

Figure 3 shows the measured error compared to the number of trees in the forest for the 4 different algorithms. RANDOM-SIMILARITY and LEAST-DIFFERENCE do not depend on the number of trees and are shown as horizontal lines for their error level.

The error for N-RANDOM-TREES and N-HYBRID rapidly decreases for N values up to around 100 trees, and then starts flattening out, and more than 1000 trees only gives a slight decrease in error. This is expected as additional trees provide less new information when a larger proportion of possible attribute combinations have already been examined, and in general it becomes increasingly difficult to improve a result the lower the remaining error is.

The N-HYBRID algorithm consistently provides lower error than the base algorithms N-RANDOM-TREES and LEAST-DIFFERENCE, which suggests that our combined hybrid approach works successfully in our domain. For a large number of trees the errors for the N-HYBRID and N-RANDOM-TREES algorithms are relatively close in absolute value, but achieving the same incremental improvement using only a larger N value would be much more computationally expensive than using the hybrid approach, and might possibly be unachievable at the highest end as the error continues to flatten out around 4.0.

Figure 4 also shows the measured error for the 4 different algorithms, but compared according to the time (computational resources) required. (For legibility the RANDOM-SIMILARITY and LEAST-DIFFERENCE algorithms are shown as extended horizontal lines, while in reality they consist of only the leftmost point since their execution time for a given set of inputs remains constant apart from small random fluctuations in the computing environ-

ment.)

While the exact values are highly dependent on the type of computing machine it's measured on, we are interested in the relative differences between algorithms which are mostly determined by their computational complexity.

We see that the LEAST-DIFFERENCE algorithm is considerably more computationally expensive than the efficient N-RANDOM-TREES implementation for low values of N. Running the LEAST-DIFFERENCE algorithm is roughly comparable to 1000 random trees, and this overhead is reflected for the N-HYBRID algorithm as well.

The increase in computational cost to generate hundreds of trees for N-HYBRID is relatively modest compared to the cost of including the LEAST-DIFFERENCE computations at all, which means that the N-HYBRID algorithm rapidly becomes the best-performing of the 4 algorithms once enough computational resources are allotted to run it at all.

Algorithm	Time	Error
RANDOM-SIMILARITY	0.3 seconds	4.98
LEAST-DIFFERENCE	45 seconds	4.55
1-RANDOM-TREE	0.5 seconds	4.87
10-RANDOM-TREES	1 second	4.46
100-RANDOM-TREES	4 seconds	4.10
1000-RANDOM-TREES	30 seconds	4.04
10000-RANDOM-TREES	300 seconds	4.03
100000-RANDOM-TREES	3030 seconds	4.01
1-HYBRID	45 seconds	4.50
10-HYBRID	45 seconds	4.33
100-HYBRID	50 seconds	4.08
1000-HYBRID	70 seconds	4.02
10000-HYBRID	320 seconds	4.01
100000-HYBRID	2730 seconds	3.99

Table 1: Numerical results for our 4 algorithms in the palliative care domain.

Table 1 shows a more detailed numerical display of an illustrative subset of the results for power-of-10 values of N. 100-RANDOM-TREES is a very quick algorithm that produces a relatively low error compared to the other algorithms, while the best results are achieved by the N-HYBRID algorithm for high values of N. It is also interesting to note that 100000-HYBRID runs faster than 100000-RANDOM-TREES, which is because in the HYBRID version the somewhat expensive operation of comparing 100k trees is only performed for half as many cases.

Conclusions

In this paper we have developed a new random decision tree (RDT) algorithm that can be very efficiently implemented in a CBR setting. We have combined this RDT algorithm with a simple traditional similarity measure based on domain knowledge to create a hybrid similarity assessment algorithm. The hybrid combination outperformed the base algorithms it was based on by returning predictions with consistently lower error on average for cases from a palliative care domain.

Acknowledgments

We wish to thank Cinzia Brunelli for providing the data set, and Anne Kari Knudsen for interpreting the data and analysing the relevance of the features from a clinical perspective.

References

Bian, S., and Wang, W. 2007. On diversity and accuracy of homogeneous and heterogeneous ensembles. *Int. J. Hybrid Intell. Syst.* 4:103–128.

Breiman, L. 2001. Random forests. *Machine Learning* 45:5–32. 10.1023/A:1010933404324.

Fan, W.; Wang, H.; Yu, P. S.; and Ma, S. 2003. Is random model better? on its accuracy and efficiency. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, 51–. Washington, DC, USA: IEEE Computer Society.

Ferguson, A., and Bridge, D. 2000. Generalised weighting: A generic combining form for similarity metrics. In *Procs. of Eleventh Irish Conference on Artificial Intelligence & Cognitive Science, J.Griffith and C.O'Riordan*, 169–179.

Gashler, M.; Giraud-Carrier, C.; and Martinez, T. 2008. Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous. In *2008 Seventh International Conference on Machine Learning and Applications*, 900–905. IEEE.

Houeland, T., and Aamodt, A. 2010. The utility problem for lazy learners - towards a non-eager approach. In Bichindaritz, I., and Montani, S., eds., *Case-Based Reasoning. Research and Development*, volume 6176 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 141–155. 10.1007/978-3-642-14274-1.

Liu, F. T.; Ting, K. M.; and Fan, W. 2005. Maximizing tree diversity by building complete-random decision trees. In Ho, T. B.; Cheung, D.; and Liu, H., eds., *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 605–610. 10.1007/11430919.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artif. Intell.* 42(2-3):363–391.

Patterson, D. W.; Rooney, N.; and Galushka, M. 2003. Efficient retrieval for case-based reasoning. In Russell, I., and Haller, S. M., eds., *FLAIRS Conference*, 144–149. AAAI Press.

Richter, M. M. 1995. The knowledge contained in similarity measures. Invited Talk at ICCBR-95.

Smyth, B., and Cunningham, P. 1996. The utility problem analysed - a case-based reasoning perspective. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, 392–399. Springer Verlag.

Wess, S.; Althoff, K.-D.; and Derwand, G. 1994. Using k-d trees to improve the retrieval step in case-based reasoning. In *Selected papers from the First European Workshop on Topics in Case-Based Reasoning*, EWCBR '93, 167–181. London, UK: Springer-Verlag.

Paper D
An Efficient Hybrid Classification Algorithm - an Example from Palliative Care

Authors: Tor Gunnar Høst Houeland and Agnar Aamodt

Published in: Corchado E., Kurzyński M., Woźniak M. (eds) Hybrid Artificial Intelligent Systems. HAIS 2011. Lecture Notes in Computer Science, vol 6679, pp. 197–204.. Springer, Berlin, Heidelberg (2011)

https://doi.org/10.1007/978-3-642-21222-2_24

My contributions: I developed the RDT algorithm, research, and experiments presented in the paper, wrote the initial draft, and incorporated comments and suggested changes into the text.

An Efficient Hybrid Classification Algorithm - an Example from Palliative Care

Tor Gunnar Houeland, Agnar Aamodt

Department of Computer and Information Science,
Norwegian University of Science and Technology,
NO-7491 Trondheim, Norway
{houeland,agnar}@idi.ntnu.no

Abstract. In this paper we present an efficient hybrid classification algorithm based on combining case-based reasoning and random decision trees, which is based on a general approach for combining lazy and eager learning methods. We use this hybrid classification algorithm to predict the pain classification for palliative care patients, and compare the resulting classification accuracy to other similar algorithms. The hybrid algorithm consistently produces a lower average error than the base algorithms it combines, but at a higher computational cost.

Keywords: hybrid reasoning systems, classifier combination, case-based reasoning, random decision trees

1 Introduction

Case-based reasoning (CBR), including instance-based methods, represents a unique approach to learning and problem solving compared to generalization-based methods. It is therefore often a choice of one method in a hybrid system, complementary to generalization-based and inductive methods. Examples include using an ensemble of different inductive methods to perform adaptation in CBR [12], and a neural network approach for adaptation, revision, and retainment of cases [5]. As a lazy learning method that postpones the generalization step until problem solving time [1], CBR has the advantage of including contextual information that an eager approach would not have access to, thereby adapting the reasoning to the particular characteristics of the problem to solve. Eager methods, on the other hand, have the advantage that parts of the problem solving behaviour can be precomputed during training, which enables reduced storage space and faster query processing.

A path of our research is to explore the combination of model-based reasoning, starting from a predefined model that make some top-down commitments, with case-based reasoning that make very few high level commitments but rather grows its knowledge base (case base) in a bottom-up fashion. An example is the combination of Bayesian Networks with case-based reasoning [4]. In this paper we examine a hybrid approach that uses a modified version of an eager method,

Random Decision Trees (RDT), that can be partially precomputed and partially adapted to the particular problem query.

As of today, there is no consensus about the set of classes that should be used for pain classification in palliative care [9]. Our domain is open and changing, which is why we study methods of machine learning and decision support that are able to produce useful results without making very strong commitments about the domain.

In an earlier study we examined the problem of determining case similarity in our palliative care domain, and created a hybrid RDT approach to locate the most similar case in the case base [7]. In the work presented here we extend our approach by developing algorithms for classifying cases. We do this by predicting the *average pain* and *worst pain* values for the third week after the first consultation, based on the information collected for the first two weeks. These values are important because the objective is to minimize the patient's pain, and the doctor's approach for relieving the patient's pain is applied in full during the second week. In the third and following weeks, the patient is mainly observed and pain medication is modified according to needs.

In the next section we review some earlier relevant research, which is followed in section 3 by a description of our RDT-based experiment and the algorithms used in the experiment. In section 4 we compare the algorithms and their parameters and discuss empirical results from running the algorithms on a case base of palliative care patients. Concluding remarks end the paper.

2 Related Research

Studies of ensembles of random decision trees have been extensive. Among the most well-known is the Random Forest (RF) classifier [3] which grows a number of trees based on bootstrap samples of the training data. For each node of a tree, m variables are randomly chosen and the best split based on these m variables is calculated based on the bootstrap sample. Each decision tree results in a classification and is said to cast a vote for that classification. The ensemble classifier returns the class that receives the most votes. RF can also compute *proximities* between pairs of cases that can be used for clustering and data visualization, and hence as similarity measures for case-based reasoning.

In a thorough study of ensemble method types it was found that the performance of an ensemble learning approach varies substantially across applications. Bian et. al. [2] studied homogeneous and heterogeneous ensembles and found connections between diversity and performance, and an increased diversity for heterogeneous ensembles.

A contribution to the analysis of the laziness vs. eagerness distinction, which corresponds with the distinction between global and local approximations to the target function, was made by Hendrickx and den Bosch [6]. They studied several hybrid methods as well as their single components. The analysis showed that the k-NN method outperformed the eager methods, while the best hybrid methods outperformed any single methods on combined generalization performance and

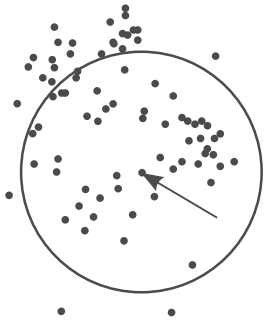

Fig. 1. Using a similarity-based local case subset as training data. A set of neighbors of the marked problem query to solve are shown as the cases that lie within the circle. That set is used to train an independent learning algorithm.

statistical error bias. A combined approach for optimizing the combined learning and classification time of lazy and eager learners was developed by Mohebpour et.al. [11], a problem also addressed by Veloso and Meira [14].

A particular problem relates to the utility of learned knowledge. The "utility problem" occurs when additional knowledge learned decreases a reasoning system's performance instead of increasing it [10, 13]. Theoretically this will always occur in a CBR system when the system's case base increases without bound. The utility problem is not necessarily observed in practice for real-world CBR systems with moderately-sized case bases, however.

Based on one of our own studies [8], we suggest that the usefulness of an optimization should be measured by the effect it has on the reasoning system's overall utility. We measured an example system's total solution time to show that case base size reduction methods can be counterproductive because the methods were more computationally demanding than simply reasoning using the larger unreduced case base.

3 Random Decision Tree Classification Experiment

The hybrid random decision tree (RDT) algorithm presented here is an approach to combining machine learning methods with case-based reasoning. We retrieve the most similar half of the available cases using a domain-specific relevance measure (a general illustration of this approach is shown in figure 1). We then run our RDT algorithm as a computationally efficient machine learner using this subset of cases as training data. This approach combines the lazy and locally specific characteristics of a CBR retrieval with the more eager and global characteristics often seen in traditional machine learning algorithms.

In the presented research we expand the use of our RDT algorithm from being a pure similarity measure to also predicting the classification of unseen

cases. As part of its internal computations, each decision tree in our algorithm is partitioning the cases in the case base between its leaf nodes. This is conceptually similar to how indexing trees used for efficient retrieval in CBR are constructed. We exploit this insight to create a classification algorithm where each tree classifies a new problem query based on the previous cases that lead to the same leaf node as the new problem query.

If each tree classifies cases as the arithmetic mean of the classification of previous cases in the same leaf node, and the average of each tree is used as the combined classification, then it is not necessary to enumerate the specific subsets of cases present in each leaf node. It is sufficient to know *how many times* each case shares a leaf node with the problem query, and then the combined classification can be determined by taking the weighted average, where each case is assigned a weight equal to the number of times it shares a leaf node with the problem query.

The number of times a case shares a leaf node with the problem query is precisely the *proximity* of the case, for which we have previously developed an efficient computational method while exploring the use of RDTs to determine similarity [7].

Using this proximity-weighted averaging approach, we have implemented a purely RDT-based classifier and a hybrid RDT+CBR classifier. We explore their characteristics related to the palliative pain classification domain. For comparison purposes we also test a k-NN classifier corresponding to the CBR part of the hybrid, and a simple and very fast algorithm based only on averaging. We compare the results obtained from these algorithms according to their computational complexity.

Our data set consists of 1486 cases with numerical features based on patients in the palliative care domain. The problem description we use for input queries to be solved consists of 55 numerical features based on measurements and classifications obtained during the first two weeks after the first consultation. Examples of these features include the patient's age, the reported average pain for week 1 on a scale from 0-10, the total opioid dose given as pain relief for week 2 as a floating point number, and similar features for other aspects such as insomnia, cognitive functioning and use of antidepressants. As the solution to predict we use 2 classifications related to the patient's pain for the third week: the reported *average pain* and *worst pain* on scales from 0-10.

3.1 Algorithms

Computed-Average computes the mean *average pain* and *worst pain* values based on the cases encountered so far, and uses these computed means as the predicted classifications for the new problem query. It is a simple and fast algorithm which only learns from the problem solutions. It performs this limited task well, and is used as a baseline comparison for the other algorithms which attempt to also learn domain knowledge from the more complicated problem descriptions.

CBR-k-NN selects the average of the k most similar previously encountered cases. Similarity is measured using a simple CBR-Difference-Measure function that was provided as a rough relevance estimate. This estimate is based on differences in 8 values in the data set that correspond to the variables a domain expert considers most important. For $k = 1$ this is the same as retrieving and copying the solution from the most similar case, while $k \geq 2$ performs averaging as a simple and knowledge-lean multi-case adaptation step during reuse.

N-RandomTrees-Classifier is based on our presented approach for classification by efficiently evaluating random decision trees on case subsets. N trees are grown and the *average pain* and *worst pain* values are predicted as the average of all cases in the case base weighted by their computed proximity to the problem query.

N-Hybrid-Classifier is our hybrid combination of the CBR relevance measure and using our RDT algorithm for classification. For every input problem query, the CBR-Difference-Measure function is used to narrow the case base down to the most similar half. Then N trees are used to compute the average and worst pain as in the N-RandomTrees-Classifier algorithm, but based only on the cases from this most relevant half of the case base.

4 Results and Discussion

To achieve a fair comparison we generate 10 versions of the input where the same patient cases are used, but in 10 different randomly shuffled orders. We evaluate the algorithms by their average result from each of these modified case bases. We use this approach because the results of a single run-through of the case base can vary, due to intrinsic randomness in the RDT-based algorithms and differences caused by the order in which the cases are presented. For each algorithm we measure the root mean square error (RMSE) for solving each of the 10 permuted case bases, and report the average RMSE value.

Figure 2 shows the measured average root mean square error for the different algorithms, compared according to the time (computational resources) required. The result for the Computed-Average algorithm is shown as a single point, as there is no varying parameter and the execution time for a given set of inputs remains constant apart from small random fluctuations in the computing environment. The exact time required depends on the type of computing device that is used to run the algorithms, but we focus on the relative differences between these algorithms which is primarily determined by their computational complexity.

The results for CBR-k-NN are not as sensitive to the exact value of k as an initial reading of the graph might suggest, because the value of k has a relatively small effect on the time required to run the algorithm. In fact the visible line for CBR-k-NN in the graph spans from around $k = 5$ to $k = 1000$.

Additional details are shown in table 1, with numerical values for a subset of the results. The results shown in the table are marked as points in figure 2.

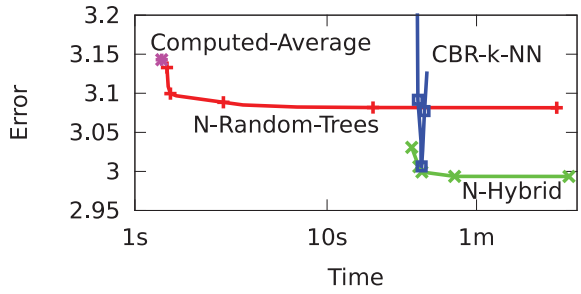

Fig. 2. Experimental results for the different algorithms and parameters, compared to the computational resources required. (Lower error and faster time is better.)

The underlying CBR-Difference-Measure function is not in itself particularly potent as a direct similarity measure. CBR-1-NN produces an error of 4.14. This is worse than a trivial classifier that always predicts 5 as the solution, which produces an error of 3.62 using the same experimental setup. However, the variables identified by the domain expert are indeed relevant, as a completely random similarity measure that retrieves a case at random produces an error of 4.46.

This indicates that the similarity measure is helpful for locating the most relevant cases, but that predicting the pain values based on only a single similar patient is unlikely to work well in this domain. A relatively large k value of around 75 produces the best result for the CBR-k-NN algorithm in this experiment.

For our RDT approaches a higher number of trees N produces better results. Unlike how k affects CBR-k-NN, there is no particular sweet spot for N in either the RDT trees or the hybrid approach above which the results start deteriorating. However, the improvements flatten out to become negligible compared to the increase in computational resources required when using more than around 1000 trees.

N-Hybrid-Classifier has the lowest overall error but a comparatively high computation costal, while Computed-Average and N-Random-Trees-Classifier are good choices to produce results very quickly.

This illustrates an important trade-off between speed and accuracy when choosing a classifier. In this experiment, our approach to combining lazy and eager classifiers to make a hybrid classifier produced better predictions, but at an increased computational cost. Whether the increased accuracy is worth the additional complexity and increased resource cost depends on the exact application and usage of the reasoning system. Given a time limit for a particular application, the algorithm that produces the best results can e.g. be determined as the lowest line at that point in a graph such as the one shown in figure 2.

Table 1. Numerical results for our algorithms in the palliative care domain, showing the computation time required and the average root mean square error.

Algorithm	**Time**	**Error**
Computed-Average	1.4 seconds	3.14
CBR-1-NN	26 seconds	4.14
CBR-10-NN	30 seconds	3.09
CBR-75-NN	31 seconds	3.01
CBR-500-NN	32 seconds	3.08
1-Random-Trees-Classifier	1.5 seconds	3.13
10-Random-Trees-Classifier	1.6 seconds	3.10
100-Random-Trees-Classifier	2.9 seconds	3.09
1000-Random-Trees-Classifier	17 seconds	3.08
10000-Random-Trees-Classifier	150 seconds	3.08
1-Hybrid-Classifier	28 seconds	3.03
10-Hybrid-Classifier	30 seconds	3.01
100-Hybrid-Classifier	31 seconds	3.00
1000-Hybrid-Classifier	46 seconds	2.99
10000-Hybrid-Classifier	180 seconds	2.99

5 Conclusions and further research

In this paper we have presented an approach for classifying unseen cases in the palliative care domain by extending our efficiently computable random decision tree (RDT) algorithm. We have developed methods for predicting the *average pain* and *worst pain* values for palliative care patients. We used a case-based k-NN method using a domain-specific relevance measure, a knowledge-lean implementation of our RDT method and a hybrid combination of the relevance measure and the RDT approach. The base RDT approach produced results very quickly, while the hybrid approach produced better results than either of the base algorithms at a comparable computational cost to running the k-NN method.

In the palliative care domain, where patients receive treatment over several months and a better result can potentially result in reduced suffering, using the best possible algorithm is usually worthwhile. However, in this domain, increasing the parameter for the number of trees in the hybrid algorithm above around 1000 increases the computational cost with negligible improvements in accuracy.

In our ongoing and future work, we are experimenting with using meta-level reasoning as part of the problem solving process. Our goal is to automatically determine which algorithm produces the best results for a given data set, and to use that algorithm for solving future problem queries.

Acknowledgments This research is partly conducted within the project TL-CPC (Transactional Research in Lung Cancer and Palliative Care), a nationally funded project in cooperation with the Medical Faculty of our university and the St. Olav Hospital in Trondheim.

We wish to thank Cinzia Brunelli for providing the data set, Anne Kari Knudsen for interpreting and analysing the data from a clinical perspective, and Tore Bruland for his analysis of the data from a data structure and machine learning perspective.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications 7(1), 39–59 (March 1994), http://portal.acm.org/citation.cfm?id=196108.196115
2. Bian, S., Wang, W.: On diversity and accuracy of homogeneous and heterogeneous ensembles. Int. J. Hybrid Intell. Syst. 4, 103–128 (April 2007), http://portal.acm.org/citation.cfm?id=1367006.1367010
3. Breiman, L.: Random forests. Machine Learning 45, 5–32 (2001), http://dx.doi.org/10.1023/A:1010933404324
4. Bruland, T., Aamodt, A., Langseth, H.: Architectures integrating case-based reasoning and bayesian networks for clinical decision support. In: Shi, Z., Vadera, S., Aamodt, A., Leake, D.B. (eds.) Intelligent Information Processing. IFIP, vol. 340, pp. 82–91. Springer (2010)
5. Corchado, J.M., Lees, B., Aiken, J.: Hybrid instance-based system for predicting ocean temperatures. International Journal of Computational Intelligence and Applications pp. 35–52 (2001)
6. Hendrickx, I., van den Bosch, A.: Hybrid algorithms with instance-based classification. In: Gama, J., Camacho, R., Brazdil, P., Jorge, A., Torgo, L. (eds.) ECML. LNCS, vol. 3720, pp. 158–169. Springer (2005)
7. Houeland, T.G.: An efficient random decision tree algorithm for case-based reasoning systems. In: FLAIRS Conference. AAAI Press (2011), to appear
8. Houeland, T.G., Aamodt, A.: The utility problem for lazy learners - towards a non-eager approach. In: Bichindaritz, I., Montani, S. (eds.) Case-Based Reasoning. Research and Development, LNCS, vol. 6176, pp. 141–155. Springer (2010), http://dx.doi.org/10.1007/978-3-642-14274-1, 10.1007/978-3-642-14274-1
9. Knudsen, A., Aass, N., Fainsinger, R., Caraceni, A., Klepstad, P., Jordhy, M., Hjermstad, M., Kaasa, S.: Classification of pain in cancer patients a systematic literature review. Palliative Medicine 23(4), 295–308 (2009), http://pmj.sagepub.com/content/23/4/295.abstract
10. Minton, S.: Quantitative results concerning the utility of explanation-based learning. Artif. Intell. 42(2-3), 363–391 (1990)
11. Mohebpour, M.R., Adznan B. J., Saripan, M.I.: Grid Base Classifier in Comparison to Nonparametric Methods in Multiclass Classification. Pertanika J. Sci. & Technol. 18(1), 139–154 (2010)
12. Policastro, C., Delbem, A., Mattoso, L., Minatti, E., Ferreira, E., Borato, C., Zanus, M.: A Hybrid Case Based Reasoning Approach for Wine Classification. ISDA pp. 395–400 (2007)
13. Smyth, B., Cunningham, P.: The utility problem analysed - a case-based reasoning perspective. In: Proceedings of the Third European Workshop on Case-Based Reasoning. pp. 392–399. Springer Verlag (1996)
14. Veloso, A., Meira, Jr., W.: Eager, lazy and hybrid algorithms for multi-criteria associative classification. In: Proceedings of the Data Mining Algorithms Workshop. Uberlandia, MG. (2005)

Paper E
Extended abstract: Combining CBR and BN using metareasoning

Authors: Tor Gunnar Høst Houeland, Tore Bruland, Agnar Aamodt, and Helge Langseth

Published in: A. Kofod-Petersen et al. (Eds.) Eleventh Scandinavian Conference on Artificial Intelligence, pp. 189–190. IOS Press (2011)

https://doi.org/10.3233/978-1-60750-754-3-189

Full paper, with title "A hybrid metareasoning architecture combining case-based reasoning and Bayesian networks" (unpublished)

My contributions: I proposed the overall paper with one section contributed from each co-author, wrote the abstract, introduction, metareasoning, and discussion sections, and edited the paper down to an extended abstract for publication.

Extended abstract: Combining CBR and BN using metareasoning

Tor Gunnar HOUELAND, Tore BRULAND, Agnar AAMODT and Helge LANGSETH
Norwegian University of Science and Technology, Norway

Abstract. In complex domains, it is often necessary to determine which reasoning method would be the most appropriate for each task, and a combination of different methods has often shown the best results. We examine how two complementary reasoning methods, case-based reasoning and Bayesian networks, can be combined using metareasoning to form a more robust and better-performing system.

Keywords. metareasoning, case-based reasoning, bayesian networks

1. Introduction

For real life applications, there are two different categories of uncertainty that are usually both present to some degree: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty refers to the general stochastic nature of the domain, where events have a certain probability of happening given the right conditions. On the other hand, epistemic uncertainty is a general lack of knowledge. This refers to our incomplete understanding of the domain, e.g. inaccurate beliefs about the causal relationships and models that do not include all significant effects. To address these different types of uncertainty, we study the combination of case-based reasoning (CBR) for epistemic uncertainty and Bayesian networks (BN) for aleatoric uncertainty, based on their respective strengths in this area. We present an automatic reasoning architecture that uses metareasoning to determine whether CBR or BN should be used to solve a new problem query, based on their past performance.

In CBR [1], a computer model is built up of a set of concrete past situations, called cases. Reasoning methods of similarity assessment, pattern recognition, and analogical mapping, rather than theory-driven methods, operate over these cases. An initial set of cases is often easier for a human domain expert to provide than a generalized reasoning model, and can often be transferred from written documentation. This makes CBR suitable as a complementary method to generalization-based models: CBR uses localized models to model a domain, and in that sense is able to fill in holes in a knowledge base in a straightforward manner, which can address problems of epistemic uncertainty.

A BN [2] models a domain using probability theory. One of the main arguments for using BNs over other model-based reasoners is the framework's sound inference engine. The belief in any query can be accurately calculated for the model given any combination of available evidence, such as the probability of an object being from a specific class given the attributes describing that object. BNs encode assumptions regarding con-

ditional independencies in the domain, and use probability distributions as its representation. BNs are well-suited to model aleatoric uncertainty, while epistemic uncertainty is not as easily captured.

2. Hybrid CBR-BN combination

In previous research we have identified four basic sequential combinations for integrating BN and CBR reasoning processes [3]. These combinations have different strengths and it is not clear in advance which combination is preferable for a given problem.

To address this issue we have created a new adaptive architecture which combines CBR and BN automatically. This is achieved by performing metareasoning about the methods at runtime based on collected performance data. For a new problem query, our metareasoner chooses either to use CBR or BN based on which algorithm produced the best answers on the most recent problems the system has solved. The underlying assumption in this approach is that a reasoning method, which performs well on a certain task, will continue to do so in the future, and that changes in classification accuracy occur gradually while learning.

We focus on domains with stochastic elements where human experts rely on their experience to solve new problems. For the CBR method a human expert provides a set of illustrative cases, which attempts to cover the range of problem categories the expert usually considers in their work.

Using CBR with the expert knowledge can immediately produce fairly good results, as a simplified model of the expert's methodology and understanding of the domain. However, future learning for the CBR system based on expert cases is limited by the availability of the expert, and consists of only adding new exceptional cases to address identified shortcomings in the model. On the other hand, BNs' statistical learning are founded on learning the probabilistic relations in the domain from training data. Because of the way our combination is structured, BNs are able to take full advantage of the new problems the system encounters as training data, and improve from an initially poor performance to bypass the performance achieved using CBR.

A benefit of using an automatic metareasoning approach to combine reasoning methods is that the computations can be dynamically adapted. For our scenario, this means that we empirically detect when there is enough representative training data for the BN to generally produce better results than the expert-modeled CBR cases, and can choose the currently best-perfoming method to solve the next problem query.

Acknowledgements The reported research is partially funded by the TLCPC project, Norwegian Research Foundation under contract no. NFR-183362.

References

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

[2] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA.: Morgan Kaufmann Publishers, 1988.

[3] T. Bruland, A. Aamodt, and H. Langseth, "Architectures Integrating Case-Based Reasoning and Bayesian Networks for Clinical Decision Support," in *Intelligent Information Processing V* (Z. Shi, S. Vadera, A. Aamodt, and D. Leake, eds.), pp. 82–91, Springer, 2010.

A hybrid metareasoning architecture combining case-based reasoning and Bayesian networks

Tor Gunnar HOUELAND [a], Tore BRULAND [b], Agnar AAMODT [a] and Helge LANGSETH [a]

[a] *Department of Computer and Information Science*
[b] *Department of Cancer Research and Molecular Medicine*
Norwegian University of Science and Technology, Norway

Abstract. In complex domains, a single type of knowledge and reasoning method is often not sufficient for a decision support system to address the variety of tasks a user performs. It is often necessary to determine which reasoning method would be the most appropriate for each task, and a combination of different methods has often shown the best results. We examine the strengths and weaknesses of two complementary reasoning methods, case-based reasoning and Bayesian networks, and discuss how they can be combined to form a more robust and better-performing hybrid. We present a metareasoning system for automatically selecting the most viable reasoning method for a particular input query at runtime, for a sustained learning scenario where an expert provides initial domain knowledge through modeling illustrative cases.

Keywords. metareasoning, hybrid reasoning systems, case-based reasoning, bayesian networks

1. Introduction

Automated computer reasoning is a challenging field. There are a multitude of different reasoning methods available, with different strengths, assumptions and learning biases. The accuracy of a method depends on the specific domain and can be difficult to predict in general, while a human expert can often determine whether a reasoning method will be either particularly useful of inappropriate for a specific problem. Recently, there has been a renewed interest in reasoning about reasoning, i.e. *metareasoning*, within automated computer reasoning systems [1]. In metareasoning there are generally two important aspects: introspection and meta-level control. Introspection refers to a system's ability to gather information about its own (object-level) reasoning processes, e.g. computational performance data, and records of the steps taken during reasoning. Meta-level control aims to increase the quality of the combined reasoning system by deciding what type of object-level reasoning to perform, based on the information collected from introspection. Meta-level control can for example be used to balance a system's limited resources between computations and external (ground-level) actions that affect the world, and dynamically adapting the specific computations and actions that are performed.

When using reasoning methods for real life applications, there are two different categories of uncertainty that are usually both present to some degree: aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty refers to the general stochastic nature of the domain, and refers to events having a certain probability of happening given the right conditions. This is in contrast to e.g. axiomatic logic where an valid implication means that a conclusion *always* occurs given the premise. On the other hand, epistemic uncertainty is a general lack of knowledge. This refers to our incomplete understanding of the domain, e.g. inaccurate beliefs about the causal relationships and models that do not include all significant effects. To address these different types of uncertainty, we are examining hybrid reasoning systems that combine reasoning methods that handle each of the uncertainty categories well. In the research presented in this paper the two methods we study are Bayesian networks (BN) for aleatoric uncertainty and case-based reasoning (CBR) for epistemic uncertainty.

We present a reasoning architecture that uses metareasoning to automatically detect at runtime whether CBR or BN should be used to solve a new problem query, given the system's current state of uncertainty. The approach is based on the identified characteristics of CBR and BN regarding uncertainty, and the decision for each problem query is based on the empirical results of using CBR and BN to solve other recently seen input queries. This approach applies to domains with both aleatoric and epistemic uncertainty, where there is a human expert who can provide a set of illustrative cases to initially model the domain, and the system receives feedback about the actual outcomes of using its predicted solutions to the problems it solves.

In the next two sections we summarize the essential properties of CBR and BN, with an emphasis on their complementary strengths and weaknesses. Section 4 discusses how hybrid combinations can be made from CBR and BN methods, and section 5 examines how a meta-reasoner can be used to create such hybrid reasoning systems automatically. A discussion and concluding remarks end the paper.

2. Case-based reasoning characteristics

In case-based reasoning (CBR) [2,3] a computer model is built up of a set of concrete past situations, called cases, stored in a knowledge base referred to as a case base. CBR is a method for problem solving that also incorporates learning from problems just solved. The core knowledge component is the case, which in its basic form has two parts, a problem description part and a problem solution part. The notion of a "problem" should here be viewed in a general sense, i.e. as a state for which some inference or action is called for that leads to a "solution", i.e. some result. Hence, a problem may be interpreting a situation, posing a question, assessing a feature value, etc., as well as solving a larger diagnostic, planning, or scheduling problem. The problem description part constitutes the set of input features to the reasoning process, while the problem solution part is the system's suggested solution to the problem. A third part is often added: outcome, i.e. the result after having applied the solution to the problem. In its simplest form a single case is represented as a list of attribute-value pairs, but more complex representations are also frequently adopted, such as hierarchical description structures.

Reasoning methods of similarity assessment, pattern recognition, and analogical mapping, rather than theory-driven methods, operate over this case base. A four-step

cycle describes the CBR problem solving and learning process (Figure 1): In the first step, Retrieve, an input case is matched against the cases in the case base, and the best matching case is retrieved. In the next step, Reuse, the solution part of the best matching case is used to construct a solution to the input case. The two final steps enable the system to learn from this problem solving session and update its case base accordingly. The proposed solution is first, in the Revise step, evaluated in order to assess its quality. This may be done by a human expert, by applying the solution to the problem, or by some additional computer model outside the CBR system - such as a simulation system. After evaluation the solution may be adjusted, and the case then enters the Retain step, in which a new case may be constructed, the new case may be merged with an existing case, or only the index structure may be updated.

The above description has focused solely on the case knowledge, i.e. the cases in the case base. As Figure 1 shows, a CBR system's knowledge base may also contain general domain knowledge, for example a set of rules, a taxonomy, an ontology, or another multi-relational domain model. Although all subareas of CBR have been subjects of active research, up to now the Retrieve step, and in particular similarity assessment, has been the most focused [4]. It is important to note that case retrieval is a partial matching process, in which the best partial match is targeted. The definition of a best match is contained in the similarity assessment procedure. In the simplest CBR method, also referred to as an instance-based method, a case is represented by a flat attribute-value list, and no general domain knowledge is used. The degree of similarity between two cases is calculated by summing up the number of identical attribute-value pairs, if we are dealing with symbolic feature values, and by computing the numerical differences if the attributes have numerical values. When general domain knowledge is used in the similarity assessment method, a similarity metric can take different forms depending on how the knowledge is used. An example expression is given in Equation 1:

$$sim(C_{IN}, C_{RE}) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} sim(f_i, f_j) \times w(f_j)}{\sum_{j=1}^{m} w(f_j)} \quad (1)$$

Here each feature in a stored case is given a weight (w) that reflects its relevance for that particular case. General domain knowledge is used to determine the degree of similarity between any two features, independent of their position in the case or whether they are superficially similar or not. In the above expression, C_{IN} and C_{RE} are the input and retrieved cases, and n and m are the number of findings in each of them, respectively. f_i is the ith finding in C_{IN}, f_j the jth finding in C_{RE}, and $w(f_j)$ the weight of that finding.

It should be noted that case-based reasoning has been strongly motivated by cognitive science research [3] and the similar abilities of the human mind [5]. For example, in their daily practice clinicians make use of personal specific experiences gained through daily work [6]. Past patient cases provide a level of specificity that focuses on single patients rather than generalized principles, and are easier to describe than generalized principles and dependencies. When modeling decision knowledge in a computer system, an initial case base is therefore often easier to come up with than a generalized model, and can often be transferred from written documentation such as patient journals.

An important property of the CBR method is its "lazy" approach to modeling and learning. Lazy in this sense means that CBR does not eagerly build generalizations of its experiences in the learning phase, but store them as specific instances, and wait un-

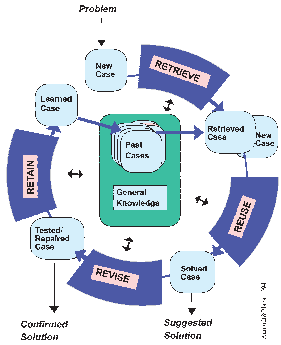

Figure 1. The CBR cycle.

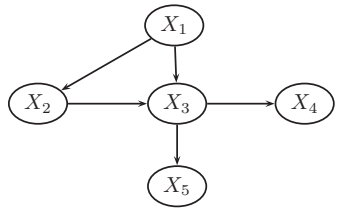

Figure 2. An example BN over the nodes $\{X_1, \ldots, X_5\}$. Only the qualitative part of the BN is shown.

til the problem solving phase to determine in what way they should be used. This also makes CBR suitable as a complementary method to generalization-based models: CBR uses localized models to model a domain, and in that sense is able to fill in holes in a knowledge base in a straightforward manner, i.e. just by adding another case. Hence, epistemic uncertainty can easily be captured by a CBR model, whereas aleatoric uncertainty, for which a known probability distribution over a set of states is assumed, favors a generalized knowledge model, such as a BN model.

3. Bayesian networks characteristics

A Bayesian Network (BN), [7,8], is a compact representation of a multivariate statistical distribution function. A BN encodes the probability density function governing a set of random variables $\{X_1, \ldots, X_n\}$ by specifying a set of conditional independence statements together with a set of conditional probability functions. More specifically, a BN consists of a qualitative part, a *directed acyclic graph* where the nodes mirror the random variables X_i, and a quantitative part, the set of conditional probability functions. An example of a BN over the variables $\{X_1, \ldots, X_5\}$ is shown in Figure 2, where only the qualitative part is given. We call the nodes with outgoing edges pointing into a specific node the *parents* of that node, and say that X_j is a *descendant* of X_i if and only if there exists a directed path from X_i to X_j in the graph. In Figure 2 X_1 and X_2 are the parents of X_3, written $\mathrm{pa}\,(X_3) = \{X_1, X_2\}$ for short. Furthermore, $\mathrm{pa}\,(X_4) = \{X_3\}$ and since there are no directed path from X_4 to any of the other nodes, the descendants of X_4 are given by the empty set and, accordingly, its non-descendants are $\{X_1, X_2, X_3, X_5\}$.

The edges of the graph represents the assertion that a variable is conditionally independent of its non-descendants in the graph given its parents in the same graph. Other conditional independence statements can be read off the graph by using the rules of *d-separation* [7]. The graph in Figure 2 does for instance assert that for all distributions compatible with it, we have that X_4 is conditionally independent of $\{X_1, X_2, X_5\}$ when conditioned on $\{X_3\}$.

When it comes to the quantitative part, each variable is described by the conditional probability function of that variable *given the parents* in the graph, i.e., the collection of conditional probability functions $\{f(x_i|\mathrm{pa}\,(x_i))\}_{i=1}^{n}$ is required. The underlying assumptions of conditional independence encoded in the graph allow us to calculate the joint probability function as

$$f(x_1,\ldots,x_n)=\prod_{i=1}^{n} f(x_i|\mathrm{pa}\,(x_i)). \tag{2}$$

A BN is a model of general domain knowledge. One of the main arguments for using BNs over other model-based reasoners is the framework's sound inference engine. One can calculate the belief in any query given any evidence (e.g. the probability of an object being from a specific class given the attributes describing that object). Mathematically, we say that we can calculate arbitrary marginal distributions, $f(x_i,x_j,x_k)$ as well as arbitrary conditional distributions, $f(x_i,x_j|x_k,x_\ell)$. The inference engine utilises the conditional independence statements asserted by the model rather efficiently, which makes BNs well suited for modelling complex systems. Models over thousands of variables are not uncommon.

The qualitative part of the BN (the graph) has an intuitive interpretation as a model of causal influence. Although this interpretation is not necessarily entirely correct, it is helpful when the BN structure is to be elicited from experts. Furthermore, it can also be defended if some additional assumptions are made [9]. These assumptions are often acceptable in real-life situations, in particular if cause-effect relations are the most important pieces of knowledge to encode. In such cases, the building of a BN structure is a viable (although sometimes rather time-consuming) task to perform by domain experts.

To elicit the quantitative part from experts, one must acquire all conditional distributions ($\{f(x_i|\mathrm{pa}\,(x_i))\}_{i=1}^{n}$ in Equation 2). Once again the causal interpretation can come in as a handy tool, but the quantification of the probability distributions is commonly regarded as the most challenging phase of the BN modelling process. Empiric studies (e.g., [10]) have shown that the results of a query are rather insensitive to small variations in the definition of the conditional distributions, but rather sensitive to variations in the model structure.

As a modelling framework thoroughly grounded in probability theory, there are efficient methods for learning BNs from data. Methods for batch learning of the quantitative part of the BN from data date back to the work of [11], see also [12]. Methods for batch learning the qualitative part from data was pioneered by [13], see also [14,15] . Incremental techniques have also been explored, see e.g. [16].

The BN theory mentioned so far assumes that all the variables in the domain are *categorical*. Defining extensions of the BN framework to also support models that contain both discrete and continuous variables is currently a lively research area, but not yet fully matured. The common approach is to translate continuous variables to discrete interval-variables, then consider all discrete variables (including categorical, ordinal, and interval variables) as if they were categorical. This process inevitably leads to a loss of precision.

Another property of the BN modelling framework worth mentioning, is that as it uses probability distributions to model a domain, it is viable to model aleatoric uncertainty. Epistemic uncertainty cannot easily be captured naturally by a BN model, and if a decision maker is faced with this type of uncertainty, one may consider alternative modelling frameworks, such as CBR.

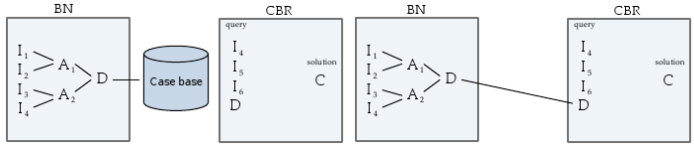

Figure 3. BN-CBR-1 Architecture

Figure 4. BN-CBR-2 Architecture

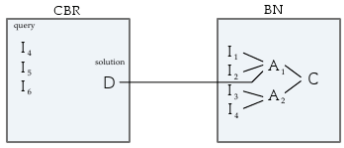

Figure 5. CBR-BN-1 Architecture

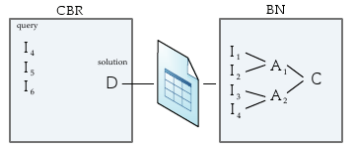

Figure 6. CBR-BN-2 Architecture

4. Hybrid CBR-BN combinations

Before we build a hybrid with BN and CBR, we must determine how the systems should be combined. We split the reasoning about the problem data according to their characteristics. For example, BN is suitable for global models with uncertainty. CBR on the other hand is suitable for local models with a lot of details and a good similarity function, and for easily representing situations to avoid as exceptions.

In a recent study we identified four basic sequential combinations for integrating BN and CBR reasoning processes [17]. These combinations are depicted in Figures 3 to 6, and use the following variable types: input variables I_i that are taken from the problem description, mediating variables A_j that represent nodes in the BN model, an intermediate variable D that is calculated from the first reasoner and transferred to the second, and the final classification variable C that is computed by the second reasoner. There are two fundamental BN-CBR sequences and two CBR-BN sequences. Figure 3 shows the BN-CBR-1 sequence, where the BN component is used to identify and activates the most relevant cases in the case base, and then the CBR component solves the problem by using only the activated cases as its case base. This architecture was used by Gomes [18] and early work in our research group [19].

Figure 4 shows the BN-CBR-2 sequence. The BN component calculates an intermediate variable D from the input variables, which contains additional useful information for retrieving cases and is transferred to the CBR system. The CBR component uses the problem's input variables together with this additional variable D to find a similar case to solve the problem. A similar architecture was used by early work in our research group [19].

The CBR-BN-1 sequence is shown in Figure 5, which follows the same principle as BN-CBR-2 but adapted to transfer from BN to CBR. The CBR component finds a similar case from the input variables and transfers an intermediate variable D to the BN system. The BN system calculates the class C from the given input variables and variable D.

CBR-BN-2 is the last sequence and it is shown in Figure 6. In this combination the case base contains different BN models, and CBR is first used to find a similar case which has a specific BN model as its solution. This BN model is then loaded and used to

calculate the class C based on the input variables. This architecture was used by Pavón et al. [20] and Louvieris et al. [21].

One or more of these basic sequences can also be put together into a larger hybrid system. As an illustrative example, let us assume that the task is to recommend a suitable place of study for a person, using CBR-BN-1 and BN-CBR-2 to create a CBR-BN-CBR hybrid. CBR is first used to detect exceptions, such as the student already being enrolled elsewhere, or that the student has cheated and is not qualified. If one of these exceptions is found the reasoning stops and no new place of study is found. When there are no exceptions, the next step is using a BN to calculate a general place of study. If the student is good with numbers then a school of management is suggested, or if the student has good grades in natural science then an engineering program from a college is suggested. CBR is then used again to take the suggestion and find a concrete program at a particular higher education facility, perhaps with further constraints based on the student's grades. In this way the first CBR system filters out persons, the BN system calculates a type of school, and the second CBR system finds the most suitable school or university.

5. Automatic combination using metareasoning

A practical problem for creating hybrid systems is that there are many different architectures to choose from, and it is time-consuming to build a reasoning system to examine and evaluate even just one of the possible combinations. This is partially alleviated by following sound software engineering concepts, and providing modular implementations with abstractions that encapsulate the executable code in components that can be reused. The same BN implementation can be used both for a system that only uses a BN to solve problems directly, and for a combination that first uses CBR to retrieve a problem-specific BN to be used. In this way a new system can be specified easily, by just changing which component sends its output to the other, or by changing a parameter such as which similarity measure to use.

In our metareasoning architecture algorithms that are implemented as reusable components in this way are used as building blocks for reasoning during run-time. They can be viewed as a form of language for specifying reasoning systems at the meta-level. This connection to (formal) languages is not accidental [22]. A *metalanguage* is a language used to discuss a language, and such metalanguages have been formally explored in logic and linguistics. The act of introspection or reflection generally corresponds to an *upward shift* from one level to a meta-level, and for languages this means *referring* to a word instead of *using* a word in written text. For a reasoning system it means referring to a reasoning process through a form of possibly implicit naming or labeling, instead of directly referring to the problems that are solved. In an earlier paper we introduced a vocabulary to describe problems and reasoning methods for such a meta-level reasoning system [23].

A popular direction for using metareasoning in CBR systems is to use traces of object-level reasoner behavior as the meta-data [24]. These traces list the steps of the process the object-level reasoner performs, and then a form of meta-level control is used to detect and correct problems. In contrast to this error-correcting approach, we focus on using computational performance data as the meta-data. Based on their complemnatry strength related to the two categories of uncertainty, our metareasoning architecture

combines CBR and BN, by detecting at runtime which algorithm is producing the best solutions. We focus on a class of domains containing both aleatoric and epistemic uncertainty. The domain has stochastic elements but human experts use experience and remember previous incidents and consider them when solving new problems, and do not think about the domain as conditional probability distributions. For such domains, a human expert provides an initial set of illustrative cases for the CBR component, which attempts to cover the range of problem categories the expert usually considers in their work.

Additonally, our architecture is designed to support sustained learning, where the system continues to learn from new problem solving experiences while it is running, by evaluating the proposed solution of a problem based on its outcome. In the CBR component this is the purpose of the retain step, which adds new cases to the case base and generalizes and improves existing cases. On the other hand, the underlying aleatoric uncertainty in the domain means that a BN system can perform well if it is trained on statistically representative training data.

For a new problem query, our system chooses either the CBR or BN component based on their empirical performance on the most recent problems the system has solved. The underlying assumption in this approach to hybridization is that a reasoning method which performs well on a certain task will continue to do so in the future, and that changes in classification accuracy when learning occur gradually.

6. Discussion

When our assumptions are correct, the characteristics of such a domain means that a CBR system created using expert knowledge will quickly be able to produce fairly good results, by using a simplified model that's a direct expression of the expert's methodology and understanding of the domain. However, for the way the CBR system is used in this scenario, the sustained learning aspect will be limited. This is because the cases learned from the domain will not have the additional richness of an expert trying to communicate the significant effects in the domain, they will just be the problems the system happens to encounter. The sustained learning aspect will therefore mostly be accomplished by filling in data for which the expert didn't already provide guidance, while the performance on already covered cases will stay but not increase significantly.

On the other hand, the BN reasoner will start out from a very disadvantaged position. BN's statistical learning is founded on learning the probabilistic relations in the domain from training data, and using these learned relations to solve problems. A set of illustrative cases provided by an expert will typically not have the statistical properties as the real problems to be solved. However, the BN will be able to take full advantage of the new problems the system encounters. These problems will naturally be drawn from the same distribution the system has to solve, because they *are* the problems the system encounters. Because of how the BN does not receive useful information from the expert in this scenario, the BN component will initially perform poorly, but learn well from the problem solving experiences that contain the real outcome from using the solutions to address the problems in the real world. In our scenario the BN component uses a fundamentally different approach to solving problems in the domain, which will improve as new problems are solved and can go beyond the human expert's and the CBR component's approach.

A benefit of using an automatic metareasoning approach to create these combinations is that the computations can be dynamically adapted. For our scenario, this means that we empirically detect when there is enough representative training data for the BN to generally produce better results than the modeled CBR knowledge base. Even when our qualitative assumptions hold, we cannot reliably predict in general how much training data would be needed for the BN to be preferable, which depends on the exact characteristics of the specific task being performed.

The metareasoning layer also adds additional learning challenges to the combined reasoning system. If only the outcome of the final chosen solution can be retrieved, then there is an additional exploration/exploitation aspect connected to choosing whether to update the performance data for the assumed "worst" method, or simply using the "best" method to get the best results. The meta-level learning also adds an additional learning bias about the reasoning methods. For an especially peculiar domain where the relative performance of CBR and BN methods fluctuated wildly as they learned, the metareasoning layer would not be able to learn from its performance measurement meta-data, and this approach would not work.

Our metareasoning architecture can be extended in numerous ways, using both CBR and BN methods and possibly including other reasoning methods as well. The presented architecture matches the identified strengths and weaknesses of CBR and BN regarding types of domain uncertainty, but it is possible to add additional improvements on top. One approach is to also include hybrid sequential combinations as base reasoning components, and choosing the hybrid combination sequence that produces the best empirical results. Another approach is to use a form of CBR at the meta-level: instead of evaluating the most recent performance, we can evaluate the most recent performance *on similar problems*, using an additional meta-level similarity measure that is suitable for this purpose.

The CBR and BN methods used in the metareasoning hybrid approach can also provide extended functionality in addition to solving problems. As two notable examples, the CBR approach is particularly good at adding exceptions that can be used to retrieve short-cut solutions instead of performing the normal reasoning, and BN can answer many other questions about the statistical properties of the domain, such as the probability for every possible classification as a solution, or the chance of a problem having particular attributes given the observed outcome.

7. Acknowledgements

The reported research is partially funded by the TLCPC project, Norwegian Research Foundation under contract no. NFR-183362.

References

[1] M. T. Cox and A. Raja, “Metareasoning: A manifesto,” tech. rep., BBN TM-2028, BBN Technologies, 2007.

[2] A. Aamodt and E. Plaza, “Case-based reasoning: Foundational issues, methodological variations, and system approaches,” *AI Communications*, vol. 7, pp. 39–59, March 1994.

[3] J. Kolodner, *Case-based reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[4] R. L. de Mántaras, D. McSherry, D. G. Bridge, D. B. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. D. Forbus, M. T. Keane, A. Aamodt, and I. D. Watson, "Retrieval, reuse, revision and retention in case-based reasoning," *Knowledge Eng. Review*, vol. 20, no. 3, pp. 215–240, 2005.

[5] R. C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. New York, NY, USA: Cambridge University Press, 1983.

[6] V. L. Patel, D. R. Kaufman, and J. F. Arocha, "Emerging paradigms of cognition in medical decision-making," *J. of Biomedical Informatics*, vol. 35, pp. 52–75, February 2002.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA.: Morgan Kaufmann Publishers, 1988.

[8] F. V. Jensen and T. D. Nielsen, *Bayesian Networks and Decision Graphs*. Berlin, Germany: Springer-Verlag, 2007.

[9] J. Pearl, *Causality – Models, Reasoning, and Inference*. Cambridge, UK: Cambridge University Press, 2000.

[10] M. Henrion, M. Pradhan, B. Del Favero, K. Huang, G. Provan, and P. ORorke, "Why is diagnosis using belief networks insensitive to imprecision in probabilities?," in *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, (San Mateo, CA.), pp. 307–314, Morgan Kaufmann Publishers, 1996.

[11] D. J. Spiegelhalter and S. L. Lauritzen, "Sequential updating of conditional probabilities on directed graphical structures," *Networks*, vol. 20, pp. 579–605, 1990.

[12] S. L. Lauritzen, "The EM-algorithm for graphical association models with missing data," *Computational Statistics and Data Analysis*, vol. 19, pp. 191–201, 1995.

[13] G. F. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[14] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, pp. 197–243, 1995.

[15] N. Friedman, "The Bayesian structural EM algorithm," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, (San Francisco, CA.), pp. 129–138, Morgan Kaufmann Publishers, 1998.

[16] J. R. Alcobé, *Incremental Methods for Bayesian Network Structure Learning*. PhD thesis, Universitat Politécnica de Catalunya, 2004.

[17] T. Bruland, A. Aamodt, and H. Langseth, "Architectures Integrating Case-Based Reasoning and Bayesian Networks for Clinical Decision Support," in *Intelligent Information Processing V* (Z. Shi, S. Vadera, A. Aamodt, and D. Leake, eds.), pp. 82–91, Springer, 2010.

[18] P. Gomes, "Software design retrieval using Bayesian Networks and WordNet," *Lecture Notes in Computer Science*, pp. 184–197, 2004.

[19] A. Aamodt and H. Langseth, "Integrating Bayesian Networks into Knowledge-Intensive CBR," in *AAAI Workshop on Case-Based Reasoning Integrations*, 1998.

[20] R. Pavón, F. Díaz, R. Laza, and V. Luzón, "Automatic parameter tuning with a Bayesian case-based reasoning system. A case of study," *Expert Systems With Applications*, vol. 36, no. 2P2, pp. 3407–3420, 2009.

[21] P. Louvieris, A. Gregoriades, and W. Garn, "Assessing critical success factors for military decision support," *Expert Systems with Applications*, 2010.

[22] S. Costantini, "Meta-reasoning: A survey," in *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, (London, UK), pp. 253–288, Springer-Verlag, 2002.

[23] T. G. Houeland and A. Aamodt, "An introspective component-based approach for meta-level reasoning in clinical decision-support systems," in *Proceedings of the First Norwegian Artificial Intelligence Symposium (NAIS'09)*, pp. 121–132, Tapir Forlag, 2009.

[24] M. T. Cox, K. Eiselt, J. Kolodner, N. Nersessian, M. Recker, and T. Simon, "Introspective multistrategy learning: On the construction of learning strategies," *Artificial Intelligence*, vol. 112, pp. 1–55, 1999.

Paper F
A Learning System based on Lazy Metareasoning

Authors: Tor Gunnar Høst Houeland and Agnar Aamodt

Published in: Progress in Artificial Intelligence, vol 7 issue 2, pp. 129–146. Springer Berlin Heidelberg (2018)

https://doi.org/10.1007/s13748-017-0138-0

My contributions: I developed the ideas, research, and experiments presented in the paper, wrote the initial draft, and incorporated comments and suggested changes into the text.

Progress in Artificial Intelligence manuscript No.
(will be inserted by the editor)

A Learning System based on Lazy Metareasoning

Tor Gunnar Houeland · **Agnar Aamodt**

Received: date / Accepted: date

Abstract Metareasoning has been widely studied in the literature, with a wide variety of algorithms and partially overlapping methodological approaches. However, these methods are typically either not targeted towards practical machine learning systems, or alternatively are focused on achieving the best possible performance for a particular domain, with extensive human tuning and research, and vast computing resources.

In this paper our goal is to create systems that perform sustained autonomous learning, with automatically determined domain-specific optimizations for any given domain, and without requiring human assistance. We present ALMA, a metareasoning architecture that creates and selects reasoning methods based on empirically observed performance. This is achieved by using lazy learning at the meta-level, and automatically training and combining reasoning methods at run-time.

In experiments across diverse data sets, we demonstrate the ability of ALMA to successfully reason about learner performance in different domains and achieve a better overall result than any of the individual reasoning methods, even with limited computing time available.

Keywords Metalearning · Automated reasoning · Integrated learning architectures · Lazy learning · Machine learning

1 Introduction

There are a wide variety of different computer reasoning methods available, which all have their own strengths and weaknesses. Determining the most appropriate method to use in a system depends on the particular domain it will be used for, and can be both challenging and time-consuming. This general topic has seen increased interest in the last years, as noted by the e.g. the AutoML challenge [18] and similar efforts within deep learning where the structure of the network is learned automatically [53]. In this paper we present ALMA, a metareasoning architecture that addresses this problem by creating systems that adapt to specific domains automatically, within an online learning paradigm.

In a previous paper [23] we examined how existing expert knowledge and new knowledge learned from training data can be combined in hybrid systems. This included a metareasoning scenario with three pieces: a static prespecified expert model, a classification model trained from data as it becomes available, and a meta-level control agent. The meta-level agent worked by detecting at run-time when the classification learning algorithm had been sufficiently trained to outperform the expert model, and switched to using the trained model from that point onwards.

In the current work we examine how to create this type of hybrid reasoning systems *automatically*, without requiring new system- and domain-specific expert knowledge for each situation. The goal of our research is to develop a framework for metareasoning based on lazy learning, i.e. to learn which reasoning methods to apply, and when to apply them, at run-time. The core of the architecture is to be implemented in a learning system and tested, as a proof-of-concept, in comparison with other systems.

To achieve this goal, ALMA is designed to support efficient and sustained meta-level learning over time, where the system continues to learn from new problem solving experiences while it is running, without human intervention. The basis for this learning in ALMA is that we assume that while reasoning method performance will vary across different tasks and domain characteristics, a method which has

T. Houeland · A. Aamodt
Department of Computer and Information Science
Norwegian University of Science and Technology,
Trondheim, Sør-Trøndelag, Norway
E-mail: houeland@gmail.com

performed well so far for a given task will typically continue to work well for that same task in the future.

This separates ALMA from many other forms of metareasoning, notably forms that focus only on optimizing a single model, and metareasoning that focuses on addressing specific identifiable failures in the system's behavior.

We explore the differences between these metareasoning approaches and their motivation and related literature in section 2. We then present an overview of the ALMA architecture in section 3, and the specific metareasoning components we use in section 4. Section 5 describes experimental results from using ALMA, and section 6 compares ALMA's behavior with related metareasoning systems. Concluding remarks in section 7 end the paper.

2 Metareasoning systems

ALMA is an architecture for incrementally predicting solutions and learning from feedback, with a focus on learning at the meta-level to continually improve prediction performance over time. In the following sections we compare and contrast this behavior with other forms of learning systems.

2.1 Lazy learning

Laziness in the machine learning sense is used in contrast to eager learning, and means to *avoid* eagerly building generalizations from experiences in a preliminary learning phase [1]. Instead, the source data is retained, and generalizations are built in the problem solving phase when the specific problem query is already known. This allows more information to be used during the generalization, for example by building simpler models for only the part of the problem space near the particular query to solve instead of global models covering the entire space.

Certain aspects of lazy learning are essential for ALMA, as the underlying goal is to autonomously learn new information at run-time, without human assistance. To achieve this goal, generalizations *must* be created by the system while it is running and solving problems.

However, laziness comes at a cost. The "utility problem for lazy learners" [16] occurs when the addition of more information to the system increases the costs more than the benefit it brings, thus reducing utility. However, Watson [48] reported that in a commercially fielded system, the utility problem did not become a problem in practice. In an earlier paper [22] we showed that whether techniques to address the utility problem are beneficial or not always depends on the specific reasoning scenario, and that premature attempts to address it eagerly can do more harm than good.

Our goal in ALMA is to optimize whole-system performance - to produce the best possible *overall* results - and not just perform isolated optimizations to parts of the system that may or may not end up being beneficial. Because of this, we focus on how the different parts of the system combine and work together for a specific situation, and on optimizing the combined functionality of the entire system problem-solving cycle. One of the major parts of this is increasing the capabilities for being lazy at the metareasoning level compared to other architectures and typical systems in use.

In ALMA, choosing the reasoning method is postponed until the problem solving phase and performed automatically at run-time. This means that more information is available, such as performance data on how reasoning in the domain has behaved so far (e.g. whether techniques to address the utility problem are helpful or not), possible shifts in the domain over time, and even being able to adapt new reasoning methods that were not available during the initial system design but were plugged in later.

Using a lazy approach at the meta-level, we can selectively include the optimizations that are actually improving the observed results for each specific domain, while avoiding optimizations that might have only looked good on paper or only perform well on other data sets. The benefit of our approach is that the system *automatically* learns which optimizations work for a given domain.

2.2 Meta-level learning

To learn and generalize beyond the training examples *some* inductive bias is always needed [34], and whether learning is successful or not depends on how the bias interacts with the unknown domain characteristics. The study of meta-learning is to examine how to increase learning efficiency through experience, which Vilalta and Drissi [46] defines as choosing the correct inductive bias. In this view the role of meta-learning is precisely to dynamically shift the inductive bias in a way that achieves better results, which in ALMA corresponds to changing which reasoning components are selected.

There is a rich variety of research related to meta-learning from different specialized fields [28]. To illustrate where ALMA fits in this research landscape, we compare and contrast important related metareasoning approaches by roughly dividing them into three groups:

- Using dataset metadata characteristics to select an appropriate learning method [32]. A typical example would be to identify e.g. training set size and whether features are categorical or continuous as being important, and using rules based on these values to select an algorithm. This form of metareasoning is often performed manually, in which case a common approach is to select

methods based on how well their biases are perceived to match domain characteristics.
- Identifying and correcting problems [36, 13]. In this case we do not know what methods will perform well, but we can identify when something has gone awry: for example when the generated solution is too costly, has conflicting steps that need to be resolved, or an internal part of the system did not perform as expected.
- Repeatedly generating and empirically testing potential solution methods [50, 35, 40]. Here we might not know about important data set characteristics or what form the solution process should take, but a meta-algorithm can attempt to solve the problem using many different methods, and evaluate how well these methods actually work in practice for the given data set. New approaches and modifications are attempted until an acceptable solution is found. This is the approach used in ALMA. The main feature that distinguishes ALMA from other methods is its focus on lazy learning to achieve this.

2.3 Reasoning from dataset characteristics

This is the most straight-forward form of metareasoning, where certain metadata describes the dataset, and is used to select a particular well-suited method. This can e.g. be a set of hand-crafted rules, a learned classifier based on examining a large number of datasets and automatically determining relationships between characteristics and methods, or an expert's intuition.

This form of metareasoning is very common when performed by humans, and is well established in the literature with a large number of approaches and methods proposed. However, it is less relevant for this paper as the types of rules humans apply are often too vague to apply algorithmically, and the general approach of pre-determining a single method based on static dataset metadata is not compatible with our goal of sustained autonomous learning.

An alternative is to use dynamic metadata that is collected at runtime instead of a static set of dataset characteristics. In the case of collecting metadata about empirically observed performance, we consider this a form of repeatedly generating and testing potential solution methods as will be described in 2.5.

2.4 Failure-driven learning

Failure-driven learning systems are not as extensively researched as the other meta-level learning groups, but we highlight them as they are well-established within *lazy* reasoning systems. This type of system is most commonly based on learning from traces of object-level reasoner behavior [13] (which can be viewed as using dynamic metadata). These traces list the steps of the process that the object-level reasoner performs, and then a form of meta-level control is used to examine the list of steps, detect problems, and correct them.

The usual form of such meta-level control is focused on 'reasoning failures', where the system runs as normal without metareasoning until an unexpected result, or an 'impasse' [27], is encountered while solving a problem. Such results can e.g. be arriving at an incorrect answer, determining that a more similar case to the answer exists but was not retrieved, or even lacking any method to solve a certain kind of problem. This approach is based on the intuition that metareasoning is particularly valuable when something unexpected happens, and that there is no reason to change the system if it is performing as expected without any problems.

Adding this type of introspective error detection and triggering of repair actions can successfully improve the performance of an underlying reasoning system automatically, even enabling the system to correctly solve problem instances that the underlying reasoner could not solve on its own [15].

However, detecting failure, determining the reasons for it, and then correcting them is a difficult process and a research problem in its own right. As an example, starting with an example from the literature of a lawn has not been cut properly [12], you would first have to figure out that the lawnmower is malfunctioning, that this is caused by a loose wheel, and then figure out how to repair it. This is a cognitively harder problem than simply cutting the lawn.

In particular, in a complex system it is typically not straightforward to determine what constitutes a reasoning failure, nor the way to fix it. In practice these systems will often consist of rules that have been pre-determined by human experts. For example, the implementation of GILA used failure patterns provided by the authors of the system, and investigating whether the patterns could be learnt from experience was left as a possible line of future research [36].

In contrast to this failure-driven approach to metareasoning, our work focuses on learning in a more general and flexible sense, without necessarily being tied to a problem solving failure. Learning opportunities are assumed to always be present, even when nothing is explicitly identifiable as being 'wrong'.

2.5 Method combination and evaluation

The essence of multi-method learning is to use and combine multiple learning algorithms in some way to improve the overall performance. Even within this sub-field, there is a vast and varied number of approaches in the literature, and it is growing rapidly [50]. For ALMA we are particularly interested in the *meta-algorithms* that are used to combine machine learning methods and evaluated based on empirically estimated accuracy.

One well-known form of this multi-method learning has been found in the boosting literature [41], where meta-algorithms train many different versions of basic learners and combine them in an ensemble that is better than any of the single individual learners, based on the observed performance of the learners. Research into this topic has provided useful algorithms for boosting the performance of practical machine learning approaches, by treating the machine learning methods as black box abstractions and improving them at the meta-level. In this regard it has the same objective as our research, where we use a different approach to follow some of these same ideas.

This form of meta-level combination and evaluation has also been studied with different names and approaches in several other related fields, for example:

- Hyperparameter optimization [43], choosing the best parameters for a learning algorithm, with a focus on minimizing the number of evaluations.
- Prediction with expert advice [8], minimizing regret compared to reference forecasters, with a focus on worst-case performance (adversarial experts).
- Ensemble learning [26,33,7,14], based on combining multiple different versions of a classifier, reminiscent of statistical ensembles that consider all possible world states at once.
- Multi-level learners [49,10] (also called stacking or blending), where multiple base-level learners produce output, and higher levels of learners are trained with this output from the lower levels as input.
- Portfolio selection [30], optimizing wealth across assets, with a focus on the trade-off between expected return and risk.
- Combining forecasts [47,11,3], the general notion of combining multiple different predictions of the future with independent errors and variance to produce an overall more accurate outcome.

These approaches represent different focus areas, but have significant overlaps and are all based on using meta-level methods to evaluate and use different versions of object-level methods. Our focus will be on describing the process used in ALMA, as iteratively making decisions in an unknown decision space. This process could be studied in any of the fields listed above, but is most closely associated with hyperparameter optimization and prediction with expert advice.

We later compare ALMA against a selection of related metareasoning systems in section 6, after describing how ALMA works in theory and practice.

2.6 Reinforcement learning and multi-armed bandits

In reinforcement learning an agent attempts to maximize its rewards from an environment that is not fully known, and has to learn how to behave and what actions to take based on receiving a reward signal. This creates two opposing and contradictory goals: the agent should spend time performing the actions that it expects will provide the greatest reward, but at the same time it should try to learn more about the environment to make sure that the agent's expectations are correct and it is performing the correct actions. If new information from the environment shows that another action would be more beneficial, the agent should adapt its behavior accordingly. This is called the exploration-exploitation trade-off, as an agent is typically not able to do both at the same time, and has to choose between them. (E.g. to learn the effect of choosing option X over option Y, you actually have to choose option X. If the agent is expecting option Y to perform better, this exploration of choosing option X naturally incurs an expected loss.)

A famous problem exemplifying the exploration-exploitation trade-off is the multi-armed bandit problem. In this problem a gambler can choose between many different slot machines in a casino, where each machine has its own probability distribution for what rewards it will produce. The problem for the gambler is to choose which machines to play. In the traditional setting the gambler has no initial knowledge of the machines, and can only learn about their reward distribution by attempting to play them, which means not playing one of the other - possibly better - machines. Solutions to this problem can be used to address many real-world situations, from clinical trials [44] to internet advertising [9], where choosing between multiple options with uncertain results can be modeled as choosing which slot machine to play. The multi-armed bandit problem can be seen as a more restricted and structured instance of reinforcement learning, as e.g. the rewards can be directly linked to the corresponding actions with certainty.

In ALMA we model decisions at the metareasoning level as essentially such a multi-armed bandit problem, and adapt existing multi-armed bandit strategies to work in the metareasoning setting.

3 The ALMA architecture

The ALMA architecture is based on a functional decomposition of system behavior into three reasoning layers, which correspond to basic problem-solving (L_0), domain learning (L_1), and introspective metareasoning (L_2). In this section we describe the ALMA reasoning process and how these reasoning layers are used.

3.1 Problem solver selection

Metareasoning in ALMA consists of choosing what reasoning methods to use for each input query, which can be seen as repeatedly solving the algorithm selection problem [37]: select a good algorithm, for a large and diverse problem space, from a large and diverse algorithm space, using complex performance criteria.

The ALMA metareasoning process consists of evaluating a sequence of reasoning components, starting with a fixed metareasoning root component (L_2). Iteratively, the current active component is evaluated and passes control to the next component, which is either at the same metareasoning level or a lower one. Often one or more of these components will be new, never-before-used components that are internally constructed at run-time during the reasoning process. In this way the system is dynamically exploring the space of possible components, not just evaluating a fixed set of options. Once an L_0 component is reached, it can simply be evaluated with the current problem query as the input, and will output a proposed solution to the input problem query. The meta-level learning in the system comes from remembering which components were used for each problem-solving attempt, and assigning them credit based on the quality of the solution.

As an example, the process of solving a single input problem query might be: start with the root L_2 component and use performance data as input $\rightarrow$ create a new L_2 component and use performance data as input $\rightarrow$ choose an existing L_1 component and use training data as input $\rightarrow$ create a new L_0 component and use the problem query as input $\rightarrow$ return the output as the solution to the problem query.

As all these problem-solving attempts start at the root component, the set of all reasoning paths encountered across multiple problem solving attempts can be viewed as a tree. Each node in this tree is assigned a score based on the aggregate credit from all solutions that were generated based on using that component, i.e. based on observed performance of the subtree formed under the node. Figure 1 shows a simple example of such a component tree.

For smaller systems it is possible to simply try every reasoning method on every input, and thus gather information about all of them at the same time when the correct solution is supplied later. This allows full exploration gains while the best-performing subtree can be exploited every time without losing out on exploration opportunities. However, there can be a combinatorial explosion of possible methods and the ability to try different methods (particularly training new models) is limited by runtime CPU resources, which is an important issue often not sufficiently addressed in the literature. In our work we also use unbounded trees to explore new combinations and new parameter settings - subtrees that lazily expand themselves forever to grow larger when they are explored. Trying every possibility is of course not possible in this case.

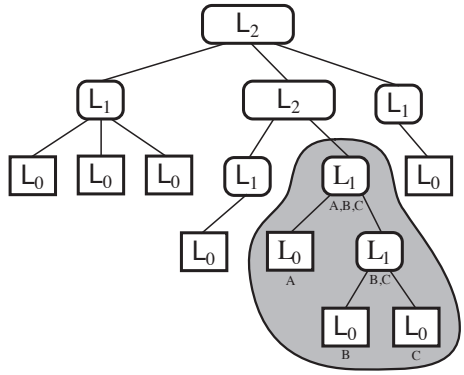

Fig. 1 Snapshot of a (tiny) ALMA component hierarchy. The L_0 boxes represent non-learning instances of fully trained classifier functions, e.g. decision trees and neural networks. A single ALMA system typically contains a heterogenous variety of such classifier instances. L_1 boxes represent methods that learn from training data, and L_2 boxes represent the metareasoning components that ALMA is focused on. The shaded subtree illustrates score aggregation, where the L_1 component marked "A,B,C" is scored based on the combined scores of the L_0 components marked "A", "B", and "C"

3.2 Metareasoner properties

Our metareasoner's task is to achieve the best overall result by incrementally selecting which reasoning method to use for each problem. This directly maps to choosing the best child node, which we view as conceptually playing an arm in the multi-armed bandit setting. Instead of having a single multi-armed bandit, we have a horizontally and vertically unbounded hierarchical structure of bandits and sub-bandits, representing the possibility of creating new components instead of reusing existing ones.

To evaluate the available options for a single component at a given step in the process, we can still model this as a regular multi-armed bandit problem. Each of the previously visited child node components represents an arm on the bandit, with the observed solution quality from previous visits forming the basis for estimating the value of the component, just like previous rewards from playing a bandit arm is used to estimate the value of playing the arm. The option to create a new child component is another arm on the bandit, with an unknown value.

There is a certain degree of uncertainty about how well the reasoning methods will perform, because of uncertainty in the domain itself, stochastic behavior for some of the reasoning methods, and fundamentally because the reasoning performance changes over time as more training data becomes available. This is precisely captured by the evalua-

tion of the reasoning methods as an exploration/exploitation problem and using established strategies for the multi-armed bandit problem to explore the decision space. Our metareasoner therefore also inherits this ability to optimize its own performance by choosing between focusing on the assumed best method to get good results, or to further examine other methods with more uncertain characteristics to find a potentially better reasoning method for the future.

3.3 Reasoning meta-layers

ALMA describes reasoning systems using functional layers, in the sense that all components on the same layer implement functions that operate on the same kinds of inputs and outputs, and these outputs are deterministic functions of the input[1]. We define these functions as belonging to L_0, L_1, or L_2 (and refer to components as belonging to the same level as their corresponding functions):

Definition 1 Let L_0 be the set of functions mapping input problem queries to solutions: $Query \rightarrow Solution$.

Let L_1 be the set of functions mapping training data consisting of problem queries and their solution to L_0 functions: $TrainingData \rightarrow L_0$.

Let L_2 be the set of functions mapping performance data to L_1 functions: $PerformanceData \rightarrow L_1$.

L_0 corresponds to fixed problem-solving procedures that can output a *Solution* for any given *Query*. This could e.g. be a set of if-then rules, a decision tree, or a feed-forward neural network. This could for example be "Always predict *rain*", or "If $x_1 \geqslant 3$, predict 0, else predict $\sqrt{x_2}$". These functions do not change and therefore necessarily cannot learn.

L_1 corresponds to algorithms that learn L_0 functions from domain training data. In ALMA the *TrainingData* is a list of $(Query, Solution)$ pairs corresponding to previously issued queries. An example L_1 function is ID3, a specific algorithm for learning decision trees. These functions can learn by selecting or creating new L_0 functions that match the provided training data.

L_2 corresponds to systems that learn from how well they are doing, i.e. not just from the object-level domain data, but based on how well the system's reasoning processes are performing. These functions can learn by selecting or creating new L_1 functions that generalize well when predicting solutions for unseen queries. This is the focus for ALMA, which is directly based on learning based on previously observed performance. The *PerformanceData* in ALMA thus consists of scores for these functions, represented explicitly in the form of *(Query, components used, Solution, Result)* performance records, which are used to assign accuracy scores for the components in the system. E.g. for a classification component the overall score is the number of correct predictions made for newly received queries when using a component, divided by the number of predictions made.

In other systems the form of *Performance* data used will be different. For example, using back-propagation to iteratively refine weights in a neural network will implicitly encode parts of the performance data into these learned weights. Another example is using a pruning algorithm to modify and improve the performance of an already-built decision tree based on a hold-out data set.

[1] Algorithms with stochastic behavior are modeled as different random number generator states being different functions.

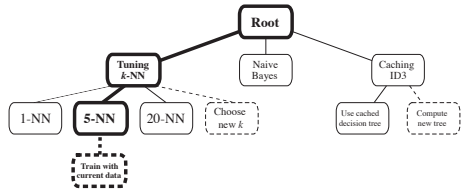

Fig. 2 Example problerm-solving flow. For each new query, components are recursively evaluated from the root node until arriving at a leaf node. The leaf node component is used to predict the output solution for the query. In this example the Tuning *k*-NN sub-tree is first selected, then 5-NN, a new L_0 component is trained using 5-NN, and this new component is used to answer the problem query. (Previously trained L_0 components are not shown in this figure)

3.4 Node hierarchy

When choosing how to solve a new problem query, the system consults the *component node hierarchy*, which contains the components in the system and additional information about them. A core element of ALMA is to decompose the reasoning process into these components that can be modified independently by the system.

The hierarchy is represented as a tree (as seen in figure 1), where each node contains a component and branches represent alternative specialized choices that can be made. The basic operation of the system is to evaluate each of the possible branches and choose which child node to explore further. Then the same process is repeated, until a final leaf node is reached, which determines the L_0 function to apply to solve the problem instance.

Figure 2 shows an example of this node-selection behavior. At the top layer, the system is choosing between three components:

Tuning *k*-NN L_2 parameter tuning component on top of *k*-NN algorithm.

Naive Bayes L_1 component using Naive Bayes algorithm.

Caching ID3 L_2 caching component on top of ID3 algorithm.

Then, if k-NN is chosen, at the next layer it would choose the value of k among the three components 1-NN, 5-NN, and 20-NN that have already been created (or create a new component with a new value of k). These are L_1 components, which apply the k-NN algorithm with k = 1, 5, or 20. Choosing the 5-NN component, it is then applied to all the training data seen so far as input, to produce a new L_0 component as output. This L_0 component is then applied to the problem query as input, and the output is used as the final prediction of the system.

When a node is evaluated, it can also change the node subtree below it, in particular creating new components. Using meta-components in this way, we represent an unbounded node hierarchy as sub-trees that are not expanded until the node is evaluated.

The node selection algorithm is applied recursively at each layer of the tree, and the process of choosing between alternative nodes is modeled as a multi-armed bandit problem as described earlier. To "play" a node, we apply it to the current problem instance, and the "reward" is the observed accuracy of the overall problem solving experience the node was a part of.

3.5 Problem solver evaluation

We assume that a problem can be answered in many different ways, and that each answer can be scored and compared to the scores for other answers. In this way different components can also be indirectly compared based on scoring the answers they produce - if problems are being correctly solved, that indicates the component is performing well.

A common way to compare different general algorithms in this way is to manually evaluate them on data sets with known answers [20] and measure the accuracy as how often they predict the correct answer. The empirically measured accuracy is then used as a measure of quality, to say that an algorithm that scores higher *is a better algorithm*.

In our architecture the system itself is automatically and continuously comparing the reasoning methods based on their results in this same way. When the real solution to a problem query is provided, the scores of components are updated based on scoring the predictions they made. This is an integral part of the reasoning process, and empirically measured accuracy forms the basis for how the overall reasoning system decides which methods to use.

Many of the reasoning methods also have tuning parameters. These parameters affect both the results and the resource requirements for performing the computations. This has practical consequences, because the final result is not the only factor that matters for a reasoning system application - the answer also has to arrive quickly enough to be useful. There is usually a trade-off between solution quality and computational complexity.

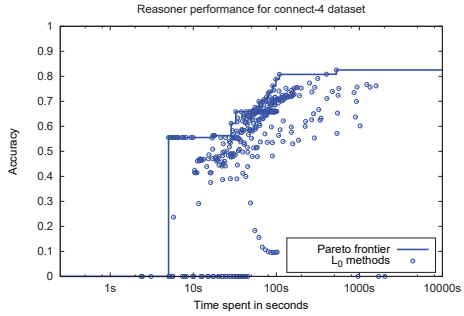

Fig. 3 Example reasoner performance differences (using the methods from 4.4). The points in the centers of the circles represent different L_0 reasoning methods and show their accuracy vs. time performance records. The Pareto frontier of possible optimal choices representing different accuracy/time trade-offs is shown as a solid line.

We formally analyze these differences in method performance as a mapping from specific fully parameterized L_0 components to the results they achieve. The results include both the performance of the component's function in accurately solving problems, and the component's computational performance as measured by the resources consumed to execute the specific implementation of the algorithm the component represents.

The best possibilities from these results form a Pareto frontier - the set of optimal dominating choices. Choices on the Pareto frontier involve a trade-off between accuracy and resource cost, while the other dominated options are simply inferior. Figure 3 shows an example of such a Pareto frontier, with a two-dimensional plot of the benefits of different reasoning methods measured as average accuracy vs. time taken to compute the answers. The results on the frontier dominate the rest of the results by either producing a higher score with the same computational requirements, the same score with less computation, or even a higher score with less computation required. *Some* option on the frontier will always be preferable to any dominated option not on the frontier.

In ALMA we assume that reasoner performance will not drastically change based on small changes in the training data. Specifically we assume that past performance is indicative of future performance, and that the discrepancy between previously observed and future performance is higher for larger increases in available training data. This is the motivation for treating scoring as an exploration-exploitation tradeoff - the potential increase in performance and therefore the value in obtaining new information for a component is higher for a larger gap in training data between what has previously been observed and what is currently available.

In figure 3, the L_1 components that produced the L_0 components with points closer to the Pareto frontier would be evaluated more frequently than the ones far below the

frontier, as they're more likely to increase to a level above the previously best choices. But even the poorly performing components are retried eventually, since there's a possibility for large increases in performance once enough additional training data is available.

4 ALMA components

ALMA uses a uniform component representation for all reasoning layers in the system. Problem solving is performed by recursively evaluating components in the component node hierarchy to determine which L_0 function to use. Each component is self-describing, using a component description language to specify which inputs it requires and what outputs are produced (for details see [21]).

In this section we first describe 3 ALMA components that illustrate the capabilities of the system related to our research goals:

1. Child node selection component, which performs metareasoning by choosing which components to use.
2. L_0 function caching component, which trades off laziness and higher accuracy in exchange for lower computational requirements.
3. Parameter tuning component, which gradually optimizes parameters based on observed empirical performance.

Finally we describe the learning methods available to the system for our experiments, which are implemented as L_1 components.

4.1 Child node selection component

The bandit strategy we use in our experiments to choose which child node to evaluate next in the component node hierarchy is to evaluate nodes according to the Upper Confidence Bounds applied to Trees (UCT) algorithm [24]. We chose this bandit strategy as it is particularly well-suited for tree structures by applying it recursively at each level as described earlier. This UCT algorithm is previously particularly well-known from computer Go, where the use of tree-based exploration techniques ushered in a stronger generation of Go-playing programs than the previous state-of-the-art expert systems [17,6], and was later used by the first computer player to beat professional human Go players [42].

Applying the UCT algorithm consists of applying a uniform scoring function that automatically balances exploitation and exploration (based on UCB1 [2]) to each of the possible choices: $score_i = \bar{x}_i + \sqrt{C_i \times \frac{\ln N_i}{n_i}}$. Here $\bar{x}_i$ is the measured average accuracy so far from choosing node i, n_i is the number of times node i has been explored, N_i is the number of times node i's parent node has been explored, and C_i is a parameter that controls the weighting between the exploitation and exploration term for node i.

In standard UCB1, the same constant $C_i = 2$ is used for all nodes. However, some variants of UCB1 vary the C_i parameter to achieve better practical performance on finite problem sets. In our experiments we use UCB1TUNED (also from [2], and recommended as performing better in practice), which has $C_i = \min(\frac{1}{4}, \text{estimated variance}_i)$ instead of a fixed constant value[2].

4.2 Caching component

Instead of using an L_1 function to relearn the problem from scratch for every problem query, this metareasoning component wraps the learning component with a cache. The L_1 function will only be used to compute a new L_0 function when the number of node visits is a power of 2, i.e. after 1, 2, 4, 8, 16, 32, ... explorations. For other node visits, a trivial L_1 function is used instead, which simply ignores the training data and reuses the latest previously computed L_0 function. This approach is illustrated in figure 4.

The marginal accuracy for the caching component displays a step-wise behavior where the accuracy remains flat when the cached classifier function is reused, and then increases up to the underlying Naive Bayes classifier's accuracy when it is recomputed at 2^N queries. In the example, the cumulative accuracy of the caching version is 97% of the underlying classifier, while requiring 4% of the computational resources. Whether this trade-off is beneficial or not is *not* known a priori but depends on the reasoning scenario.

Performing this caching is an eager optimization, and undesireable from a lazy learning perspective. But it is one way to enable using slower non-incremental methods that would otherwise be too computationally expensive to include in a purely lazy manner.

Whether to apply the caching to a component or not is automatically optimized in our system based on observed behavior, in the same manner as choosing between any other components, and it is only included when it is demonstrably producing better outcomes.

4.3 Sequential parameter-tuning component

This component parameterizes an underlying component to create many different versions, e.g. k-NN with different k values. Initially, the component's subtree consists of a single tuning node with $k = 1$. When it is explored, the subtree is

[2] Standard UCB1 can provably achieve asymptotic zero-regret for the basic multi-armed bandit problem, while this has not been shown for UCB1TUNED. However, the zero-regret proof doesn't apply to the harder metareasoning problem ALMA is addressing, and for our components we are primarily interested in practical performance.

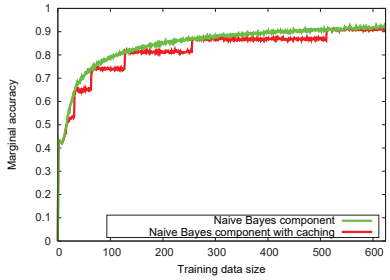

Fig. 4 The performance of an underlying learning component and the effect of adding caching. The graph shows marginal accuracy, which clearly displays the step-wise behavior of the caching component.

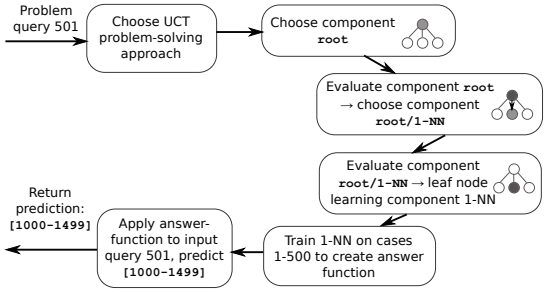

Fig. 5 Example system flow. This shows the process the system goes through to solve a single problem query. The nodes in the tree correspond to components that are recursively evaluated, until a leaf node is reached which in turn produces a prediction for the given problem query

replaced with two child nodes: a regular node applying the underlying component with $k = 1$, and a new tuning node where k has been incremented by 1, i.e. a tuning node with $k = 2$.

These nodes are explored using the normal UCT behavior, and will recursively build a subtree for $k = 1$, $k = 2$, $k = 3$, $k = 4$, etc. Using the UCT exploration/exploitation algorithm, the choice about how high k should be increased is automatically learned in the same way as other component choices.

This is a very simple tuning component suitable for choosing between relatively small positive integer values. Whether it is beneficial to include or not is automatically optimized at run-time in the same way as other components.

4.4 System reasoning components

An L_1 function is implemented in our system as a procedure taking a data set as input and returning as output a procedure representing an L_0 function. The k-NN, ID3, and Naive-Bayes reasoning components use standard machine learning methods from the literature, implemented directly as components in our system with minimal overhead. These implementations are primarily used to verify and demonstrate the core behavior of ALMA.

Our implementation also has a component for using classifier implementations from WEKA [19]. This component has significant overhead for learning an L_0 function based on the WEKA implementations, but gives access to plugging into the system a wide variety of classifiers. This is useful for evaluating practical multi-method metareasoning performance.

In our experiments we used the following list of classifier implementations from WEKA: AdaBoostM1, AODE, BayesNet, DecisionStump, HyperPipes, IBk, Id3, J48, JRip, LBR, LogitBoost, MultiLayerPerceptron, NaiveBayes, KStar, LMT, OneR, Prism, RandomForest, RandomTree, SMO, VotedPerceptron, Winnow, ZeroR.

5 Experimental results

In this section we first illustrate a real problem solving attempt to show how a small ALMA system works. Afterwards we present experiments that illustrate ALMA's behavior and performance when learning which reasoning methods to use at run-time (our research goal) on a variety of data sets.

5.1 System processing flow

Figure 5 shows the process our system goes through when solving a problem query using the UCT metareasoning component, in this case showing the next query for the Travel domain after having completed 500 previous queries (an approximately 'average' query scenario for that domain). The objective in this classification task is to predict the price, which has been discretised to ranges from `[0-499]` up to `[7000-7499]`.

The UCT algorithm scores each node according to the UCB1TUNED formula to determine which child branch to select. The score consists of two parts: the 'exploit' part which here estimates the accuracy of a learning algoritm (Correct / Attempted), and the 'explore' part which is an optimistic estimate of the uncertainty surrounding this accuracy estimate. The 'explore' part decreases every time a component is selected, and increases every time a component is *not* selected, ensuring that even an underperforming component will eventually be revisited again (these increases are logarithmic, meaning it takes longer and longer between revisits if the component keeps performing poorly).

The system state before answering problem query 501 is shown below:

Component	Result	Exploit + explore = value
root	152 / 500	
root/1-NN	107 / 332	0.322 + 0.068 = **0.390**
root/ID3	44 / 153	0.288 + 0.101 = 0.389
root/NB	1 / 15	0.067 + 0.322 = 0.389

When answering problem query 501:

- The UCT algorithm starts at the `root` node and evaluates each of the children: `root/1-NN`, `root/ID3`, and `root/NB`.
- Child node `root/1-NN` is chosen because it has the highest combined exploitation + exploration value computed for the UCT algorithm: `0.390`.
- `root/1-NN` is an L_1 node for applying the 1-NN learning algorithm.
- 1-NN is trained on problem queries 1 to 500 to create a new L_0 component.
- The new L_0 component predicts an answer for query 501: `[1000-1499]`.
- The case 501 solution is `[1000-1499]`, so the prediction was correct.
- Performance records are added for the components involved in the problem query: `root` and `root/1-NN`.
- Case 501 is added to the training data.
- When evaluating case 502, the "Result" column for `root/1-NN` will now be 108 / 333 with an "exploit" value of 0.324. The "explore" value will decrease for `root/1-NN` and increase for `root/ID3` and `root/NB`.

5.2 Example implementation results with limited feedback

Scenario and restrictions:

- *Problem queries arrive as described in 5.1.*
- *Exactly 1 method is used per problem query.*
- *Feedback is only provided for the method used.*

The system's task is to choose one of the available problem-solving methods to solve each problem, and the metareasoning problem is to sequentially decide *which* method to assign to each problem instance.

For this scenario the system will exclusively perform method-selection and only run a single prediction algorithm for each problem query, without knowing the predictions from the other unselected algorithms. This is the most restrictive scenario, where no additional time for exploration is available, and is primarily suitable for testing and understanding how the UCT-based prioritization in ALMA works in isolation.

The focus here is on the core objective, namely to test ALMA's ability to automatically learn which method best suits each particular domain, at runtime, with no prior information (but will not produce the best possible result, as these restrictions are partially artificial). The system cannot perform any other metareasoning such as averaging or voting, and will not learn from any other sources than these core problem-solving efforts.

The problem-solving task we use in this first example is the Balance Scale data set [31], which consists of 625 cases each having 4 nominal attributes and a solution classification. The 4 attributes interact in that they actually represent the weight and distance on each side of a scale, and the class represents whether the scale will tip to the left, tip to the right, or that it is balanced.

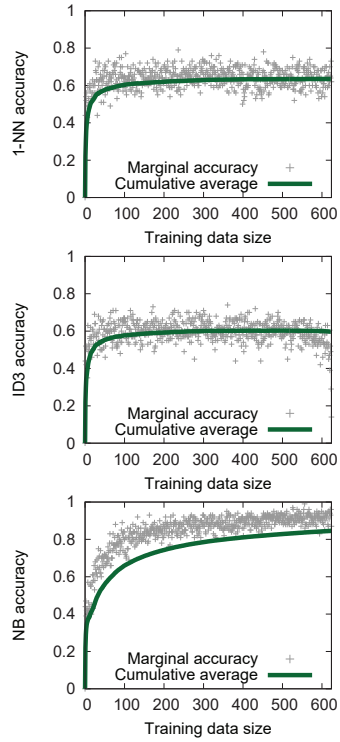

Fig. 6 Average marginal and cumulative accuracy for the Balance Scale data set. The data points show the average marginal accuracy for each incremental problem solving attempt, while the solid curves shows the cumulative average accuracy over the entire data set up to that point

For these experiments the reasoners learn from the set of previous cases with attributes and solutions, and attempt to predict the classification for a new case based on its attributes. The goal is to maximize the cumulative accuracy (number of correct predictions made for the entire data set).

5.2.1 Individual learning algorithm behavior

The system behavior is illustrated in figure 6, which shows the individual performance of the reasoning methods. The data points at the right-hand side of the graphs are mostly above the curve for Naive Bayes, which means the average would continue to increase if there were more samples. This observed behavior is in line with our expectation that the methods will generally converge to different accuracy values. The marginal accuracy in the figure roughly corre-

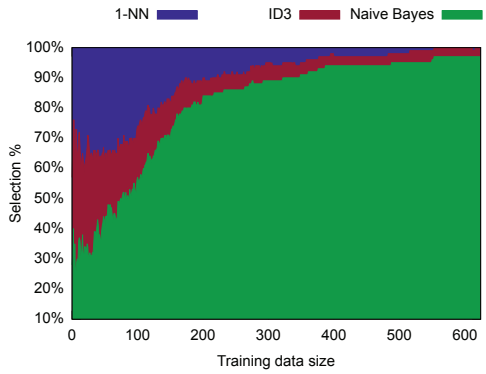

Fig. 7 UCT-select behavior for the Balance Scale data set. The graph shows the average fraction of queries where each of 1-NN, ID3, and Naive Bayes are chosen for the Nth problem query. Naive Bayes performs the best on this data set, and the system correctly learns to prioritize using Naive Bayes higher than the other methods

sponds to a best estimate of method performance (the figure shows the average over hundreds of trials, which is not available to the system as it only has a single history of hits and misses available). The cumulative accuracy represents the overall score achieved by each method (and still roughly identifies which methods are better or worse, while being significantly less noisy).

The method's observed performance varies from data set to data set, and forms the input for selection at the metareasoning layer. This reasoner data is not directly available to our metareasoner, but must be estimated by sampling at run-time, which is conceptually sampling points from these graphs to determine which method performs best.

5.2.2 UCT metareasoning component

Figure 7 shows the average selection behavior of the UCT metareasoning component for the Balance Scale dataset using these 3 methods.

Initially, with no data collected about their performance, the three methods are chosen approximately 1/3rd of the time each, as expected. As more and more data is collected, the system is increasingly focused on Naive Bayes, having identified it as the best method (highest 'exploit' part in the UCB1TUNED scoring formula). The confidence in the accuracy estimates also increase over time, and the system spends less and less time on exploring the other two methods (slower increases for the 'explore' part).

These results illustrate how our child node selection component can address the metareasoning problem in practice and enable the system to continuously learn and improve at the meta-level.

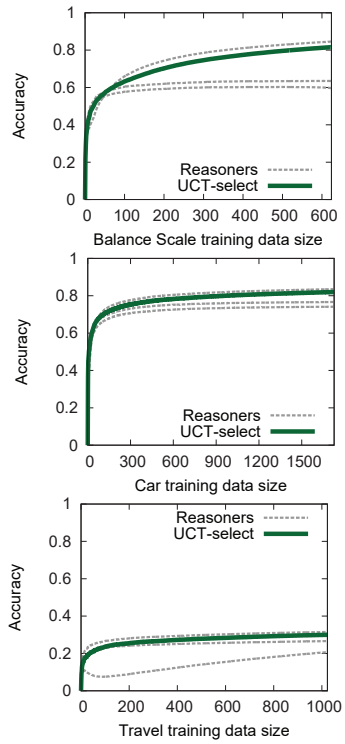

Fig. 8 Aggregate system behavior for 3 different data sets. The thin dashed lines in the graphs represent the performance of individual reasoning methods (1-NN, ID3, Naive Bayes) while the thick line is UCT-select's performance using those three methods

5.2.3 Aggregate system behavior for multiple data sets

Table 1 and figure 8 show the performance of the reasoning methods across three different data sets: Balance Scale [31], Car [31], and Travel [29].

The optimal behavior for the reasoning metalayer given complete and perfect information would be to always pick the highest-accuracy method for each data point. This ideal corresponds to always picking the top line in the graph for each training data size, and represents the set of Pareto efficient options (those not dominated by any other option).

The behavior of our system using empirical observations of reasoning method behavior and optimistic UCT estimates to approach this ideal is shown as a solid line in the graph. Each method is expected to roughly converge towards an average accuracy number, and the system attempts to find and choose the method with the highest cumulative accuracy so far.

Table 1 Cumulative accuracy for 3 small data sets. UCT-select does not reach the highest accuracy on any individual data set, but has the highest average overall accuracy. The "Percentage of best" columns show the cumulative accuracy as a fraction of the best method for each data set

Data set	Cumulative accuracy				Percentage of best	
	1-NN	ID3	NaiveBayes	UCT-select	UCT-select	Random selection
Balance scale	63.8%	59.4%	**84.6%**	81.8%	96.7%	81.9%
Car	76.7%	73.9%	**83.5%**	82.3%	98.5%	93.5%
Travel	**31.5%**	26.5%	20.6%	30.1%	95.6%	83.1%
Average	57.3%	53.3%	62.9%	**64.7%**	96.9%	86.2%

5.3 Reasoning performance with limited resources

Scenario and restrictions:

- *Problem queries arrive as described in 5.1.*
- *Problem queries must be answered within a certain time limit, which is explicitly specified to the system.*
- *Feedback is in the form of the correct classification, and can be used to train and evaluate multiple methods.*
- *Default parameters are used for all learning algorithms, and the parameter-tuning component is not included.*

As shown in figure 8, the accuracy and computational cost of reasoning methods vary from data set to data set. The optimal behavior for the reasoning metalayer would be to always pick the highest-accuracy method that completes the task within the allotted time. In this scenario ALMA attempts to do so, but the time limit means that the ability to train is limited and has to be prioritized between components.

In these experiments we use a variety of data sets and the WEKA reasoner implementations combined with the caching component as the underlying reasoning components. The metareasoner's job is to find reasoners that perform well for each given data set within the time limit. We do this by prioritizing which components should receive more time to train new L_0 functions, based on the score and time taken by the components so far. At the time a problem query deadline is reached, we use the presumed best L_0 function to produce an answer, and when a new solution is provided as feedback we update all component scores. (E.g. after a RandomForest model has been successfully trained for 256 cases, that model will be used until 512 cases have been observed and a new model based on those 512 cases has been trained, however long that takes. For ALMA similar model training takes place, but using the UCT metareasoning component to decide which algorithms to use and when to retrain their models.)

Some form of normalization is required to compare accuracy across data sets, as they vary wildly in difficulty and data set size. We consider each of the data sets as equally important, and normalize performance per dataset against a target performance level. We set this target for each data set as the highest accuracy achieved by any of the individual reasoning methods with default settings, and scored each reasoning method according to their relative accuracy compared to this benchmark. I.e. if the best of these methods achieved 80% accuracy on a data set, scores would be scaled by 1/80% = 1.25x. In this case achieving an accuracy of 80% would get a score of 100%, 0% accuracy would score 0%, and a hypothetical new component that reached 100% accuracy would score 125%. The overall score for a reasoning method is then the mean score across all data sets. (The important effect of this normalization is to make sure the largest data sets don't entirely dominate the results. Beyond that the exact procedure used does not significantly alter the results.) Additionally, each method/dataset pair was tested with 4 different random seeds to determine the order of training examples, and the results show the average performance across dataset orders.

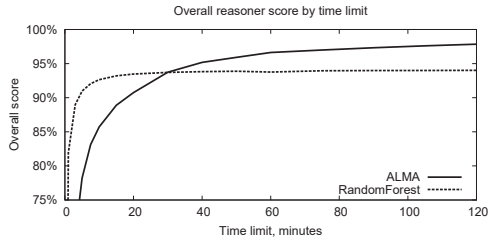

Fig. 9 Overall reasoning performance across all data sets, comparing ALMA and the best individual reasoning method RandomForest. Given sufficient time, ALMA considerably outperforms RandomForest and the other individual reasoning methods

Figure 9 and table 2 show an overview of the performance in this scenario, comparing the overall scores of ALMA and RandomForest (the best individual method). Figure 9 shows the overall performance at different time limits, while table 2 shows more details broken down per dataset. Table 2 also includes the results from the previous limited feedback scenario for comparison, showing (as expected) that ALMA performs better when the restrictions from 5.2 are removed and more methods are available.

The p-values listed in the table are the probabilities of observing the differences between the classifiers by chance, computed by an exact binomial test (as recommended in [39]). For each dataset, where X is the number of times ALMA predicted correctly and RandomForest predicted incorrectly, and Y is the number of times ALMA predicted incorrectly and RandomForest predicted correctly, the listed p-value is $Pr(k \geq max(X,Y); n = X+Y; p = 0.5)$. I.e. the

Table 2 Data sets used and experimental results with a 1 hour time limit. Score differences larger than 5 percentage points are highlighted. The UCT-select results from table 1 (converted to scores) are also included for comparison with the previous limited feedback scenario

Data set	Size	Target	ALMA	RandomForest	Difference	p-value	UCT-select
agaricus-lepiota.data	8.1k	0.9978	99.19%	99.90%	-0.71%	10^{-52}	
balance-scale.data	0.6k	0.8968	**98.93%**	81.18%	+17.75%	10^{-74}	91.21%
car.data	1.7k	0.9401	**99.42%**	94.00%	+5.42%	10^{-40}	87.54%
connect-4.data	68k	0.7913	90.57%	**99.45%**	-8.88%	10^{-2552}	
covtype-subset.data	50k	0.6370	**91.20%**	78.60%	+12.59%	10^{-679}	
house-votes-84.data	0.4k	0.9420	99.08%	99.33%	-0.24%	10^{-1}	
kddcup-subset.data	50k	0.9976	95.04%	99.14%	-4.10%	10^{-2303}	
kr-vs-kp.data	3.2k	0.9750	99.07%	99.21%	-0.14%	10^{-0}	
monks-1.test	0.4k	0.8941	**98.64%**	90.68%	+7.96%	10^{-14}	
monks-2.test	0.4k	0.7743	**96.34%**	84.30%	+12.03%	10^{-24}	
monks-3.test	0.4k	0.9653	99.28%	98.32%	+0.96%	10^{-02}	
nursery-casebase.json	13k	0.9700	98.74%	98.40%	+0.34%	10^{-02}	
phishing-websites.data	11k	0.9565	98.87%	99.87%	-1.00%	10^{-52}	
poker-hand-subset.data	50k	0.7975	83.99%	88.03%	-4.04%	10^{-603}	
secom_processed.data	1.6k	0.9330	96.94%	99.76%	-2.82%	10^{-54}	
Skin_NonSkin.txt	245k	0.9720	91.55%	**98.84%**	-7.29%	10^{-14535}	
soybean-large.data	0.3k	0.7728	**98.74%**	93.47%	+5.27%	10^{-04}	
SPECTF.test	0.2k	0.9144	98.83%	99.27%	-0.44%	10^{-02}	
ThoracicSurgery.data	0.5k	0.8489	98.87%	99.25%	-0.38%	10^{-01}	
tic-tac-toe.data	1.0k	0.9444	**99.25%**	89.14%	+10.11%	10^{-72}	
travel-casebase.json	1.0k	0.3662	**96.00%**	78.73%	+17.27%	10^{-17}	82.20%
Average (mean score)			96.60%	93.76%	+2.84%	N/A	

probability of observing an event at least k times out of n trials (results at least as extreme as those we observed), given that the chance is 50% each time . We know the actual probability is highly unlikely to be 0.5 since the methods work in very different ways, nor will it be independent between samples since the methods are continually learning, but this provides some idea of how well the differences can be detected empirically from the data alone. To illustrate, for soybean-large.data, over the 4 random dataset orders, ALMA and RandomForest were both correct 840 times, only ALMA was correct 96 times, only RandomForest was correct 50 times, and both were wrong 242 times. The p-value, or the chance of observing this by chance if they were actually equally good, is $Pr(k \geq 96; n = 146; p = 0.5) \approx 0.0000875$. Since there are 21 datasets, the chance is approximately 21 times larger (using a Bonferroni correction) that some dataset will produce spurious results, though with many of the p-values below 10^{-10} it is clear that ALMA and RandomForest do not have the same behavior. The robustness of the results for these particular datasets is also supported by the smooth shape of the performance curve in figure 9, as there are no wild fluctuations in the overall performance.

Table 3 shows two reasoning scenarios with different time limits: our focus is on the "1 hour" scenario wherein each problem instance is presented in sequence and must be answered within (1 hour / number of instances in the data set), leading to a varied range from 14ms to 19250ms per query. In this scenario our metareasoning system performs well on every data set, and achieves an average score of 96.6% of the best reasoning method for each data set, outperforming any single reasoning method overall.

The only way ALMA can gain the information required to choose the right algorithm is by trying to apply the algorithms to the data set in question. As can be seen by the increasingly higher performance in figure 9 when more time is available, ALMA is able to do this and autonomously learn how to increase performance by making better choices. For comparison, a similar but severely constrained scenario with only 1 minute to classify the dataset is also included in the table, showing a situation where ALMA does not perform well. In this scenario there are purposefully not enough resources available to spend time exploring multiple methods, and it is significantly better to just pick one single method and train it as much as possible.

6 Comparison with other systems

Bonissone's approach to lazy meta-learning [5] is based on the same type of meta-level model selection as ALMA, but in a scenario where a large number of models have already been built and are available in the cloud. Therefore the focus is on model selection and fusion, and it relies on model meta-information being available. This differs significantly from ALMA in that domain learning is not addressed (and the details of the two approaches are quite different), but we share a similar high-level research direction towards dynamic method combination, and in particular have the same view on the ideal Pareto optimal set of methods for multi-objective optimization.

Table 3 Overall scores for ALMA and individual reasoning methods. Our focus is on the 1 hour scenario, with the 1 minute scenario shown for comparison

Method	1 hour	1 minute	Description
ALMA	**96.6%**	46.5%	**Our UCT-based metareasoning system**
RandomForest	93.8%	81.9%	Randomized ensemble learner creating decision trees
SMO	93.3%	76.2%	Support vector machine algorithm using iterative optimization
LMT	93.0%	74.1%	Classification trees with logistic regression at the leaf nodes
J48	91.7%	82.5%	C4.5: Decision trees based on minimizing entropy, with pruning
LogitBoost	91.2%	**82.7%**	Logistic regression boosting algorithm
IBk	90.7%	80.9%	1-NN: Uses classification from most similar case in training data
JRip	90.1%	80.2%	RIPPER: Repeatedly grows, prunes, and optimizes rule set
KStar	90.0%	78.0%	Instance-based learner using an entropy-based distance function
BayesNet	87.9%	77.5%	Bayesian network learner with simple estimator
NaiveBayes	87.4%	76.2%	Bayesian classifier with naive independence assumption
RandomTree	85.1%	77.2%	Decision tree considering randomly chosen attributes
AdaBoostM1	83.7%	77.8%	Boosting algorithm using one-level decision trees
LBR	82.9%	70.3%	Learns local Bayesian rules at classification time
DecisionStump	79.4%	73.9%	One-level decision tree with two branches
AODE	78.2%	71.4%	Average of Bayesian classifiers with one dependence each
MultiLayerPerceptron	77.3%	58.3%	Artificial neural network using backpropagation
OneR	76.9%	67.0%	Based on only the most informative attribute's value
HyperPipes	70.8%	67.0%	Uses attribute bounds to handle large attribute sets
Prism	70.0%	60.8%	Creates modular rule sets, aiming to be understandable
ZeroR	66.2%	61.9%	Average classification, ignores the problem query
Id3	61.1%	54.7%	Decision tree, greedily picks nodes to minimize entropy
VotedPerceptron	51.6%	45.8%	Linear classifier, works best with large margins
Winnow	46.0%	40.1%	Linear classifier, handles many irrelevant dimensions

Auto-WEKA [45,25] is a tool for automatically selecting and parameterizing reasoning algorithms for a given data set using Bayesian optimization techniques. In their experiments, Auto-WEKA often performed better than standard algorithm selection/hyperparameter optimization methods, especially on large datasets. They conclude that "some form of algorithm selection is essential for achieving good performance". Their system serves the same purpose as the metareasoning component in ALMA, but uses offline analysis on a large training set to determine one algorithm that will then be used for a separate test data set. In contrast ALMA performs incremental learning and performs both classifications and optimizations at run-time.

Stacking [49] can refer to a very general scheme of using multi-layer generalizers, or the more specific approach of training a meta-level learner using the output of base-level learners as inputs. This has been further developed as blending, which proved to be highly successful in the Netflix Prize challenge [4] when tuned appropriately. The basic form of stacking is less successful, and in particular does not perform well for our experiment in section 5.3 with an overall score of 67%, worse than most individual reasoning methods. Even achieving this score required some manual tuning in selecting which algorithms would be enabled per dataset, since by default the Stacking implementation would fail if any of the individual methods were unable to process the dataset, resulting in overall score of only 18%. The reason for the poor behavior is two-fold: first, in this case the meta-level method is trained based on already-made classifications, and in effect would have to reverse-engineer how the classifications were made just to have the same information available as the base-level learners. Second, the stacking meta-algorithm relies on having classifications available for all the base learners and has no means of prioritization. This means that all the base learners have to be trained before any classifications can be made, which is highly inefficient and uses up most of the available time on training the slowest methods.

The MultiScheme classifier combiner available in WEKA can be used to decide which algorithm to use based on the observed training error, or by the estimated error based on cross-validation. The default mode is to use the training error, which achieves a score of 73% - better than Stacking, but still worse than most of the individual reasoning methods. Using MultiScheme with 5-fold cross-validation performs worse, with a score of 58%. As expected, when directly comparing models based on the same training data, the accuracy achieved by using cross-validation is actually higher than when just using the training error. However, in our experiments where computing resources are limited, it is more valuable to spend time on building new stronger models that include additional training data than to spend extra time on selecting the very best among a set of poorer models. This result supports the general approach behind ALMA - the key to good overall performance is to prioritize effectively based on whole-system performance, not using the most powerful local optimizations in isolation.

Sartre [38] is a computer poker player that works by imitating other previous computer programs by using a large case base of their behaviors in previous games. When Sartre decides which action to take, the case base is examined to find similar previous situations, and the past behavior is reused and applied to the current situation to decide the action. Sartre uses one case base for each of three different computer programs to imitate, and has to incrementally choose which case base to use at each step to determine what action to take. Sartre uses the results from playing using each case base so far against a given opponent, and applies the UCB algorithm to determine which case base should be used next. In experiments with limit Texas Hold'em with two very different opponents, using UCB to choose the case base produced the best results, performing better than imitating any of the individual case bases and better than a plurality-vote ensemble version using the same case bases. Sartre builds on an earlier incarnation of ALMA's general approach to multi-method learning, but selecting between a small fixed number of case bases instead of a wide variety of parameterized reasoning methods.

Soar [27] is a cognitive theory, aimed to explain human cognitive behavior as well as to serve as a platform for developing artificial intelligence systems. Symbolic Concept Acquisition (SCA) is a concept learning algorithm that has been built on top of the Soar architecture using production rules. The rules employed in SCA are very simple, but through observing which rules are most useful in practice, the system can determine which of these learned rules are the most important. This approach is similar to the motivation behind ALMA, but with a basic implementation that is focused on illustrating cognitive concepts rather than predictive accuracy.

GILA (Generalized Integrated Learning Architecture) [51] is based on the combined use of several heterogeneous and independent problem solving and learning methods, using an ensemble architecture in which a central metareasoning executive (MRE) controls the processing. It is aimed at addressing problems in difficult real-world domains, where tasks are complex with multiple interacting subproblems, and where near-optimal solutions are called for. The architecture is developed as a joint effort of 11 universities, institutes, and companies, under a DARPA contract to enable better adaptive control of the airspace under traffic-intense airplane operations [52]. In GILA, metareasoning is used to combine and coordinate methods during a single problem-solving attempt, while in ALMA it is used to learn about the system at a meta-level based on previous problem-solving attempts.

7 Conclusions

We have presented a system for automatically combining reasoning methods in a hybrid system. Our metareasoning architecture and system design emphasizes laziness, and evaluates the various reasoning methods at run-time, based on their actual empirical performance on the specific domain problems to be solved.

We treat the task of discovering which reasoning methods perform well as an exploration/exploitation trade-off problem, where the system has to decide whether to improve the accuracy of lower-performing methods or concentrate on the highest-performing ones. Using this approach, our system will only learn to approach the performance of the best individual reasoning method, and does not perform well initially (when the system starts as a blank slate and nothing is known about their performance). Over time our system's performance increases, and in experiments is able to increasingly identify the best-performing reasoning method for each data set, outperforming any individual reasoning method.

Using the caching component presented in this paper, our lazy metareasoning system can also employ significantly more computationally expensive methods to make predictions, because their runtime requirements to build a model are amortized over a large number of problem query predictions. While this approach does not include the latest available information for every problem solving attempt (i.e. is not fully lazy), the predictive power is often significantly better and provides better overall performance. Using our metareasoning architecture this trade-off between the advantages of eager and lazy reasoning methods can be performed lazily at the meta-level at runtime.

While the experiments in this paper only explore a limited number of data sets, the results validate our assumptions that different methods perform well on different data sets, and that accuracy for any given method/dataset pair tends to increase relatively smoothly. The results also show that using a metareasoning system such as ALMA can take advantage of this fact, and achieve higher performance than any of the base algorithms even using the same time constraints. This is a noteworthy difference compared to other metareasoning approaches such as blending that typically require vastly more computational resources than a base algorithm.

The results from our experiments also confirm the general usefulness of method combination and evaluation based on empirically measured performance, as the best overall base-level methods also use various form of metareasoning, multiple methods, cross-validation, ensembles, iterative optimization, pruning, etc.

Using the component framework in ALMA, the system's performance can be further improved in the future by adding

new components with additional capabilities. For example, in our resource-limited experiment it was a net loss to always perform cross-validation because that consumed too much time, but it is not necessary to choose between either always or never performing cross-validation. By creating a new cross-validation component, it would be possible to adaptively employ cross-validation during node evaluations at run-time, based on whether or not it's producing better empirical results for a given reasoning scenario. Similarly, a more powerful tuning component could be created based on hyperparameter optimization methods, which would presumably increase accuracy at the cost of increased computational requirements. Then it could be automatically included only when it was actually beneficial, based on observed performance.

Including additional components has a cost though, since additional exploration time is required to determine whether they should be included or not. For example, in our experiment with WEKA reasoning methods, the system would have performed slightly better (+0.16% score) if the Winnow, VotedPerceptron, and Id3 components were not included. The effort spent on evaluating these components was a waste, since the result was always that they shouldn't be used. Given a sufficiently large and representative benchmark collection of datasets, it might be feasible to determine which components are worthwhile or not based directly on how they affect overall performance across the benchmark. However, as can be seen from the relatively modest difference in performance, ALMA is still able to achieve good results even when poorly-performing components are included.

References

1. Aha, D.W. (ed.): Lazy Learning. Kluwer Academic Publishers, Norwell, MA, USA (1997)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2-3), 235–256 (2002). URL https://doi.org/10.1023/A:1013689704352
3. Bates, J.M., Granger, C.W.: The combination of forecasts. Journal of the Operational Research Society **20**(4), 451–468 (1969)
4. Bennett, J., Lanning, S., et al.: The Netflix prize. In: Proceedings of KDD cup and workshop, vol. 2007, p. 35. New York, NY, USA (2007)
5. Bonissone, P.P.: Lazy meta-learning: creating customized model ensembles on demand. In: IEEE World Congress on Computational Intelligence, pp. 1–23. Springer (2012)
6. Brügmann, B.: Monte Carlo Go. In: AAAI Fall symposium on Games: Playing, Planning, and Learning (1993)
7. Caruana, R., Niculescu-Mizil, A., Crew, G., Ksikes, A.: Ensemble selection from libraries of models. In: Proceedings of the twenty-first international conference on Machine learning, p. 18. ACM (2004)
8. Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge University Press (2006)
9. Chakrabarti, D., Kumar, R., Radlinski, F., Upfal, E.: Mortal multi-armed bandits. In: D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (eds.) Advances in Neural Information Processing Systems 21, pp. 273–280. Curran Associates, Inc. (2009)
10. Chan, P.K., Stolfo, S.J.: Experiments on multistrategy learning by meta-learning. In: Proceedings of the second international conference on information and knowledge management, pp. 314–323. ACM (1993)
11. Clemen, R.T.: Combining forecasts: A review and annotated bibliography. International journal of forecasting **5**(4), 559–583 (1989)
12. Cox, M.: Introspective multistrategy learning: Constructing a learning strategy under reasoning failure. Ph.D. thesis, College of Computing, Georgia Institute of Technology (1996)
13. Cox, M.T., Eiselt, K., Kolodner, J., Nersessian, N., Recker, M., Simon, T.: Introspective multistrategy learning: On the construction of learning strategies. Artificial Intelligence **112**, 1–55 (1999)
14. Dietterich, T.G.: Ensemble methods in machine learning. In: Proceedings of the First International Workshop on Multiple Classifier Systems, MCS '00, pp. 1–15. Springer-Verlag, London, UK, UK (2000). URL http://dl.acm.org/citation.cfm?id=648054.743935
15. Fox, S., Leake, D.B.: Introspective reasoning for index refinement in case-based reasoning. J. Exp. Theor. Artif. Intell. **13**(1), 63–88 (2001). URL https://doi.org/10.1080/09528130010029794
16. Francis, A.G., Ram, A.: The utility problem in case-based reasoning. In: AAAICBR-93, the Proceedings of the 1993 Case-Based Reasoning Workshop (1993)
17. Gelly, S., Wang, Y.: Exploration exploitation in Go: UCT for Monte-Carlo Go. Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006) (2006)
18. Guyon, I., Chaabane, I., Escalante, H.J., Escalera, S., Jajetic, D., Lloyd, J.R., Macià, N., Ray, B., Romaszko, L., Sebag, M., Statnikov, A., Treguer, S., Viegas, E.: A brief review of the ChaLearn AutoML challenge: Any-time any-dataset learning without human intervention. In: F. Hutter, L. Kotthoff, J. Vanschoren (eds.) Proceedings of the Workshop on Automatic Machine Learning, *Proceedings of Machine Learning Research*, vol. 64, pp. 21–30. PMLR, New York, New York, USA (2016). URL http://proceedings.mlr.press/v64/guyon_review_2016.html
19. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
20. Holte, R.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11**, 63–91 (1993)
21. Houeland, T.G., Aamodt, A.: Towards an introspective architecture for meta-level reasoning in clinical decision support systems. ICCBR 2009, 7th Workshop on CBR in the Health Sciences (2009)
22. Houeland, T.G., Aamodt, A.: The Utility Problem for Lazy Learners - Towards a Non-eager Approach, pp. 141–155. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). URL https://doi.org/10.1007/978-3-642-14274-1_12
23. Houeland, T.G., Bruland, T., Aamodt, A., Langseth, H.: Extended abstract: Combining CBR and BN using metareasoning. In: A. Kofod-Petersen, F. Heintz, H. Langseth (eds.) SCAI, *Frontiers in Artificial Intelligence and Applications*, vol. 227, pp. 189–190. IOS Press (2011). URL https://doi.org/10.3233/978-1-60750-754-3-189
24. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: ECML-06. Number 4212 in LNCS, pp. 282–293. Springer (2006)
25. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. Journal of Machine Learning Research **17**, 1–5 (2016)
26. Krawczyk, B., Minku, L.L., Gama, J., Stefanowski, J., Woźniak, M.: Ensemble learning for data stream analysis: a survey. Information Fusion **37**, 132–156 (2017)
27. Laird, J.: The Soar Cognitive Architecture. MIT Press (2012)

28. Lemke, C., Budka, M., Gabrys, B.: Metalearning: a survey of trends and technologies. Artificial intelligence review **44**(1), 117–130 (2015)
29. Lenz, M.: CABATA: A hybrid CBR system. In: M.M. Richter, S. Wess, K.D. Althoff, F. Maurer (eds.) First European Workshop on Case-Based Reasoning (EWCBR-93): Posters and Presentations (Volume I), pp. 204–209 (1993)
30. Li, B., Hoi, S.C.H.: Online portfolio selection: A survey. ACM Comput. Surv. **46**(3), 35:1–35:36 (2014). URL https://doi.org/10.1145/2512962
31. Lichman, M.: UCI machine learning repository (2013). URL http://archive.ics.uci.edu/ml. Accessed: 2017-05-26
32. Masoudnia, S., Ebrahimpour, R.: Mixture of experts: a literature survey. Artificial Intelligence Review pp. 1–19 (2014)
33. Mendes-Moreira, J., Soares, C., Jorge, A.M., Sousa, J.F.D.: Ensemble approaches for regression: A survey. ACM Computing Surveys (CSUR) **45**(1), 10 (2012)
34. Mitchell, T.M.: The need for biases in learning generalizations. Tech. rep. (1980)
35. Nascimento, D.S., Canuto, A.M., Coelho, A.L.: An empirical analysis of meta-learning for the automatic choice of architecture and components in ensemble systems. In: Intelligent Systems (BRACIS), 2014 Brazilian Conference on, pp. 1–6. IEEE (2014)
36. Radhakrishnan, J., Ontañón, S., Ram, A.: Goal-driven learning in the gila integrated intelligence architecture. In: C. Boutilier (ed.) IJCAI, pp. 1205–1210 (2009). URL http://dblp.uni-trier.de/db/conf/ijcai/ijcai2009.html
37. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976). URL https://doi.org/10.1016/S0065-2458(08)60520-3
38. Rubin, J., Watson, I.: On combining decisions from multiple expert imitators for performance. In: T. Walsh (ed.) IJCAI, pp. 344–349. IJCAI/AAAI (2011)
39. Salzberg, S.L.: On comparing classifiers: Pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery **1**(3), 317–328 (1997). URL https://doi.org/10.1023/A:1009752403260
40. Schaffer, C.: Selecting a classification method by cross-validation. Machine Learning **13**(1), 135–143 (1993). URL https://doi.org/10.1007/BF00993106
41. Schapire, R.E.: The Boosting Approach to Machine Learning: An Overview, pp. 149–171. Springer New York, New York, NY (2003). URL https://doi.org/10.1007/978-0-387-21579-2_9
42. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). URL https://doi.org/10.1038/nature16961
43. Simm, J.: Survey of hyperparameter optimization in NIPS2014. https://github.com/jaak-s/nips2014-survey (2015). Accessed: 2017-05-26
44. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**(3-4), 285–294 (1933)
45. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proc. of KDD-2013, pp. 847–855 (2013)
46. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artif. Intell. Rev. **18**(2), 77–95 (2002). URL https://doi.org/10.1023/A:1019956318069
47. Wallis, K.F.: Combining forecasts – forty years later. Applied Financial Economics **21**(1-2), 33–41 (2011). URL https://doi.org/10.1080/09603107.2011.523179
48. Watson, I.: A case study of maintenance of a commercially fielded case-based reasoning system. Computational Intelligence **17**, 387–398 (2001)
49. Wolpert, D.H.: Stacked generalization. Neural Networks **5**(2), 241 – 259 (1992). URL https://doi.org/10.1016/S0893-6080(05)80023-1
50. Woźniak, M., Graña, M., Corchado, E.: A survey of multiple classifier systems as hybrid systems. Information Fusion **16**, 3–17 (2014)
51. Zhang, X.S., Shrestha, B., Yoon, S., Kambhampati, S., et al.: An ensemble architecture for learning complex problem-solving techniques from demonstration. ACM Trans. Intell. Syst. Technol. **3**(4), 75:1–75:38 (2012)
52. Zhang, X.S., Yoon, S., DiBona, P., Appling, D.S., Ding, L., et al.: An ensemble learning and problem solving architecture for airspace management. In: K.Z. Haigh, N. Rychtyckyj (eds.) Proceedings of Twenty-First Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-09). AAAI (2009)
53. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning (2017). URL https://arxiv.org/abs/1611.01578

ISBN 978-82-326-4734-7 (printed ver.)
ISBN 978-82-326-4735-4 (electronic ver.)
1503-8181

NTNU

Norwegian University of
Science and Technology