LEVERAGING STRUCTURED CASES: REASONING FROM PROVENANCE CASES TO SUPPORT AUTHORSHIP OF WORKFLOWS

Joseph Kendall-Morwick

Submitted to the faculty of the University Graduate School

in partial fulfilment of the requirements

for the degree

Doctor of Philosophy

in the School of Informatics and Computing

Indiana University

December 2012

Acknowledgements

I thank my advisor, David Leake, for the countless hours he has spent speaking with me in his office, advising me in my work. I'm also deeply thankful for the effort he has spent working with me on the papers we've co-authored. I additionally thank the rest of my committee for their help over the years, and specifically Beth Plale, for her tireless work on the NSF grant that funded much of my work. I also thank other members of the Data to Insight group and other graduate students in the Indiana University School of Informatics for their support, especially Yogesh Simmhan, Satoshi Shirasuna, and You-Wei Cheah. Additionally, I thank Geoffrey Brown, Raquel Hill, Suzanne Menzel, Amr Sabry, and Donald Byrd for their research support and guidance early in my graduate career.

I also thank my family (in particular my mother, Tina Morwick), friends, and all the other graduate students I've worked with at Indiana University for their support and camaraderie.

Finally, I thank my wife, Karalyn Kendall-Morwick, who has stood with me throughout this process. She has made the completion of my dissertation only the second most significant development during my time in Bloomington, after meeting her.

Joseph Kendall-Morwick

Leveraging Structured Cases: Reasoning From Provenance Cases to Support Authorship of Workflows

In many software domains, a user's task is to edit and develop complex structured data, such as workflows. Because complexity can make authorship difficult and error prone, effort has been made towards developing artificial intelligence systems to assist authoring. However, such systems often rely on hand-coded domain knowledge requiring significant time and resources to develop, and generally either attempt to independently solve the problem faced by the user, or retrieve similar structures for re-purposing by the user. These approaches require very little or a great deal of initiative, respectively, on the part of the user.

Case-Based Reasoning (CBR) offers an alternative between these extremes. This dissertation argues that CBR can provide a natural, knowledge-light approach to user assistance by leveraging past experience from prior works, eliminating the need for comprehensive domain theories. When cases are mined rather than created by hand, case-based reasoners rely on very little hand-coded domain knowledge to perform their tasks. Additionally, a case-based reasoner can be combined with other methods within a hybrid framework to provide superior assistance to the user by offering only the most accurate, relevant, and helpful recommendations. CBR also affords a truly mixed-initiative approach to providing assistance through recommendation.

Several key concerns exist in providing this case-based support, including addressing the complexity of performing similarity-based retrieval of structured cases, finding adequate data sources for case mining, extracting recommendations from relevant cases, combining recommendations from multiple sources, and evaluating the effectiveness of such recommendations. These concerns are addressed through the implementation of Phala: a system which mines provenance records to provide case-based assistance through recommendations of edits issued to scientific workflow authors. Phala is shown to offer valuable assistance to workflow authors and exemplifies the utility of the novel application of recommendation-based, knowledge-light, mixed-initiative assistance for such synthesis tasks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In many software domains, a user's task is to edit and develop complex structured data. Such domains include workflows [125], plans [54], and concept-maps [19]. This data can be complex, making authorship difficult and error prone. Some techniques exist for assisting authors, but these are often brittle, relying on hand-coded domain knowledge, the development of which requires significant time and resources. Some tools use generative planning to write the work entirely [36], while others retrieve similar works and present them to the user [56], leaving the user to extract the useful portions. These methods allow users to specify high-level constraints on the desired product but eliminate user discretion in the design *process*, or they may instead have the completely opposite problem – burdening the user with de-coding relevant, retrieved knowledge in order to apply it.

Case-based reasoning offers a natural, *knowledge-light* approach to user assistance by leveraging past experience from prior works, eliminating the need for comprehensive domain theories [57]. Just as a struggling author may look to prior works for inspiration, a case-based reasoning tool can provide recommendations towards completing a partially authored work by extracting recommendations from similar prior works. When cases are mined rather than created by hand, case-based reasoners rely on very little pre-coded domain knowledge to perform their tasks. Additionally, a case-based reasoner can be combined with other methods (with potentially varying degrees of domain independence) within a hybrid framework to provide superior assistance to the user by offering only the most accurate, relevant, and helpful recommendations. This dissertation will explore this process through a domain in which significant demand exists for author assistance: development of scientific workflows.

Development of scientific workflows is an ideal domain for a case-based reasoning application since there are currently many efforts in e-Science towards recording past experiences in the form of provenance [100]. Provenance can be a useful knowledge source for mining cases to be used for the purpose of assisting workflow authors. Phala is a system utilizing such a technique, along with several others, to provide assistance through a hybrid framework for selective combination of individual recommendations for presentation to the user [60, 61]. Phala was developed to exemplify the techniques presented in this dissertation as a solution for supporting authorship of structured data and to provide an opportunity for analysis of these techniques.

Research performed through the development of the Phala recommender system touches on a variety of topics, each of which is relevant to the proposed thesis. This chapter will introduce each topic and illuminate its relationship with the proposed dissertation work, closing with a presentation of the thesis to be explored.

1.1.1 Case-Based Reasoning

Case-based reasoning is a technique involving recall of specific cases (experiences, problem descriptions) in order to produce solutions to novel problems [58]. CBR is an alternative to rule-based systems, which are burdened by the requirement of a comprehensive domain knowledge. By relying on this pre-coded knowledge, rule-based systems often fall prey to the gaps which invariably arise through the scope of a designed domain theory intended to be comprehensive or fail to adapt to unique problems, which could not have been predicted at the time the knowledge base was created. Furthermore, domain knowledge is often difficult to capture, especially considering that experts possessing such knowledge may not may not be sufficiently adept at relating their knowledge in a useful encoding.

Case-based reasoners combat this problem by relying principally on past experiences, called cases, rather than solely on a domain theory. When a problem is presented to a case-based reasoner, similar cases are retrieved and solutions from these cases are adapted and applied to the current problem. After analyzing the result from applying the adapted solution, the current problem and solution are stored in the case-base for re-use with future problems. This four-step process is illustrated in Figure 1.1, which is derived from work developed by Aamodt and Plaza [1].

- **Retrieve** - Given a target problem, cases with similar problem descriptions or solutions that may be useful are retrieved.
- **Reuse** - The solution from the retrieved case is adapted to the target problem.
- **Revise** - The adapted solution is applied to the target problem, and revisions

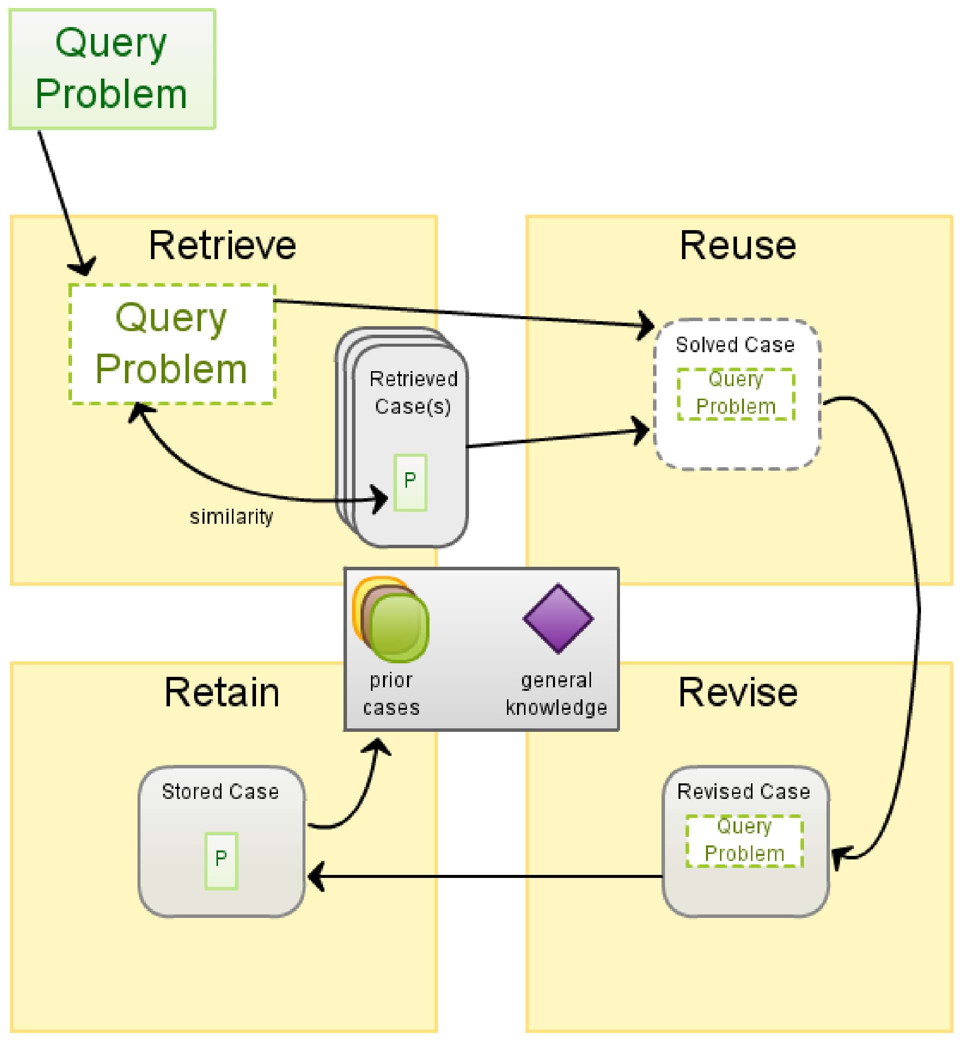


Figure 1.1: The CBR cycle

are made if unforeseen problems arise.

- **Retain** - The new problem-solving episode is stored in the case-base for future re-use with new problems.

Many case-based reasoners are considered *knowledge-light* because they do not need to rely heavily on a domain theory and because cases are usually easier to obtain and encode than are the rules of a rule-based system. Cases may be easier to create because, as opposed to the rules of a rule-based system, cases need not generalize. Furthermore, one may obtain cases *for free* by mining them from available data sources, potentially eliminating the labor of case encoding entirely, thereby unburdening a knowledge engineer and increasing the flexibility of a system to quickly adapt to new problems.

CBR was chosen for the current dissertation work because of this particular quality. Editor software often utilizes a knowledge base in order to check for errors, but rarely can it make recommendations for additional content without a much more substantial knowledge base. Phala utilizes its case-base, which contains records from previous works, to provide such assistance.

Phala implements a novel extension of the CBR process by extracting solutions from cases rather than retrieving stored solutions. Partially authored works are mapped directly to cases and the differences between the two are analyzed to predict both where extensions to the partially authored work should be applied and where errors may exist. Revision is a cooperative effort between Phala and the user as recommendations can be incorporated, altered, or ignored at the user's discretion. When users complete their task, their work can become part of a new case for future re-use. This process is explained in more detail in later sections.

1.1.2 Recommender Systems

Recommender systems have gained popularity mostly through commercial product and media recommendation [87] (for example, Amazon's product recommendation [64]). The majority of recommendations made by such systems are flat, atomic values, and the problems for which they are recommended are static and lack complexity, such as “what camera should I buy” or “what song should I listen to next”?

Both the questions and their solutions become more complicated when assisting authors of structured data. The most general question, ”What changes should be made to this structure?”, can be very complicated in the realms of plan and workflow authoring, where many types of alterations can be made involving various existing portions of a partially completed structure. In fact, in many situations, no changes should be made at all.

Researchers in e-Science have determined that support in the form of brief recommendations is desirable amongst many workflow authors [98]. To provide such support to workflow authors, Phala must compensate for the complexity of the recommendation process in how recommendations are represented internally, presented to the user, and evaluated in testing. To support this functionality, Phala relies on the Hybrid Recommendation Framework (HRF) developed as part of this dissertation work.

1.1.3 Ensemble Techniques for Machine Learning

Ensemble methods have proven useful to *adapt* recommendations produced through a variety of machine learning methods with a redundant approach. They improve performance of a machine learning technique by exploring, to some degree, the non-deterministic aspects of that technique, avoiding some degree of random bias in the technique’s output. If several instances of a machine learning technique differ because of random choices made during training, then the hope is that, in a single episode of usage for the ensemble, any error from a single instance of the technique, resulting from its particular choices made during training, will not be repeated throughout the ensemble. Such ensembles use voting techniques to arbitrate between decisions from multiple instances of a machine learning technique in order to provide a single answer. These voting methods can be much more complicated than a simple plurality, often utilizing continuously-valued *confidence estimates*, indicating the degree to which the voter believes the solution is correct, and have received a great deal of attention over the past decade [97].

Phala uses an ensemble framework for the purpose of creating a *hybrid*, a topic that has received considerably less attention in the scope of ensemble techniques. The extraction method developed for creating recommendations from prior works (cases) is made part of an ensemble with what are called *Highly Procedural Adaptation Rules* (HPAR's): light-weight standalone recommenders which specialize in how they generate recommendations, correcting some of the error in the bias of the extraction method and also providing recommendations in situations where the extraction method may not. The HPAR's for Phala have varying degrees of domain independence and rely mostly on knowledge mined from the case-base – maintaining Phala's knowledge-light goal. These HPAR's are grouped with the case-based extraction recommender through HRF, which provides a means of combining, selecting, and presenting recommendations to the user.

1.1.4 Scientific Workflows

Scientists engaging in experimentation involving massive datasets and complex computations have sought the assistance of a new cyberinfastructure for performing their experiments. Those responsible for such assisting technologies have dubbed it *e-Science* [45]. Within e-Science, scientists perform virtual experiments on a highly distributed grid network, processing immense amounts of information and sharing results electronically with colleagues [7, 38]. Workflow technology has emerged as a means of managing processes and dataflow within these experiments as well as a formal means of representing experiments [24, 125].

Scientists responsible for creating these workflows are often faced with a unique problem. The services, scripts, data warehouses, and other options from which authors of scientific workflows must choose while authoring such workflows is growing quickly, and scientists writing workflows may not be as familiar with the emerging technology as those who are providing it.

To combat this problem, many have developed systems and products designed to ease the task of authoring and enactment of scientific workflows including workflow editors [99, 8, 5] and other tools [38, 26]. Phala joins this collection with a unique contribution of both providing directed, small-step recommendations and also avoiding the need for complex domain-specific knowledge bases. Phala is also flexible enough to be included in a variety of workflow editors. A plugin for one such editor, Taverna [83], has been developed and is pictured in Figure 1.2. Additionally, by avoiding premptive planning for an entire experiment, Phala can assist real-time experimentation, adapting to any contrary choice a scientist may make throughout the episode

[20].

1.1.5 Provenanace

Provenance traditionally refers to the history of ownership of an antique or work of art. In the information and computing world, however, the term refers to information that describes the history and progression of a particular piece of data. If a data product was produced through a complicated procedure, each step of that procedure and the relevant details may be stored in a provenance record for that piece of data.

The concept of electronic provenance has sparked interest in many areas throughout computer science. In collaborative web communities such as Wikipedia, provenance is important for judging the potential quality of a document when many individuals are involved in the creation of a document or the completion of a task [65]. In Artificial Intelligence, particularly case-based reasoning, provenance can be of value when judging the usefulness of stored knowledge [62]. The study of provenance has even drawn interest from security researchers interested in the verifiability of such records [42].

In e-Science, the collection of provenance has been particularly compelling for supporting the veracity of experimental results when subject to peer review. As previously mentioned, these experiments may be complicated and involve massive amounts of data. Therefore, in order to preserve trust in their results for years to come, records of the process must be preserved so that, if necessary, the creation of the data may be retraced or even reproduced.

Provenance does not contain the same information that workflows contain. There may be gaps in a provenance trace due to dropped messages or other errors in data collection, making the trace incomplete. Also, provenance only records actual actions taken during enactment of the workflow, excluding the control structure and other potential actions which were not taken, both of which reside in the orginal workflow. In this same sense, however, provenance also provides information which workflows do not: actual run-time results from enacting the workflow. The information provenance does provide can be useful in assisting human authors of workflows.

Phala mines provenance records for a trace of the dataflow between processors for experiments. This is compared to the flow of data through a partially completed experimental workflow in order to produce recommendations for further completion. When a scientific workflow is later executed, the resulting provenance collected and recorded can be later mined by Phala to capture the revised actions chosen by the user after receiving Phala's initial recommendations, completing the retention phase of the CBR cycle.

1.1.6 Thesis Statement

The main thesis to be defended is as follows:

- Recommendation-based assistance for authors of structured data can leverage case-based reasoning to greatly reduce knowledge acquisition, computational resource, and maintenance requirements while maintaining effectiveness.

However, through implementation of such a case-based reasoning system, two sub-theses have been developed, which will also be defended:

- Ensemble frameworks can be extended to support knowledge-light adaptation within a case-based reasoner.

- Provenance can be mined, without significant domain knowledge, to build a case-base that can support authorship of scientific workflows.

Neither are intended to be recognized as requirements for a knowledge-light approach for assisting authors of structured data. However, these sub-theses support the main thesis in providing that building a case-base and adapting solutions can also be performed in a knowledge-light manner for this domain.

1.1.7 Research Questions

Given the positions taken, it will be important to answer the following questions to provide a sufficient defence of these theses:

- Can case-based reasoning enable value-added assistance for authors of scientific workflows with minimal reliance on hand-coded domain knowledge?

- Can provenance provide a substantial datasource for facilitating this process?

- Can retrieval and comparison of these cases be performed in a computationally feasible manner when dealing with structured cases?

- How can edit recommendations for authors of structured data and, in particular, workflow authors, be encoded and expressed in an abstract manner?

- When producing such recommendations through a case-based process, how can adaptation be performed while continuing to minimize the knowledge overhead?

- What is a meaningful way of evaluating a recommender system assisting with a synthesis task?
- How effective is Phala in providing assistance to authors of scientific workflows?

1.2 Chapter Summary

The first chapter is this introduction, which provides background and identifies goals for the dissertation.

The second chapter, “Intelligent Assistance for Synthesis Tasks”, conveys a broad perspective on the challenges presented by synthesis tasks, focusing on development of scientific workflows to support the perspective, which is presented with context from an important emerging area of application. This chapter begins by introducing recommendation and synthesis tasks and, in particular, their confluence. The chapter next explores the application of these concepts to assisting authors of scientific workflows and contextualizes recommendation within existing methods for assisting in this task. Following this, performance metrics for recommendations in synthesis tasks are discussed. The chapter closes with an exploration of provenance as a datasource for generating these recommendations.

The third chapter, “Case-Based Generation of Edit Recommendations”, addresses CBR’s specific role within the recommendation task. The chapter begins by describing CBR and how it can be applied to recommendation in synthesis tasks. Next, the gestalt case concept is introduced, including information on how such cases are mined from provenance, which is followed by a description of the recommendation task for assisting authors of scientific workflows, including use cases and recommendation types. Next, the process of extracting these recommendations from gestalt cases built from provenance traces is discussed as well as how new cases are recaptured. The final three sections discuss adaptation in CBR and how to employ hybrid techniques to adapt recommendations extracted from gestalt cases, including an introduction to the Hybrid Recommendation Framework (HRF), a software framework developed through this dissertation to address needs for such a hybrid adaptation approach.

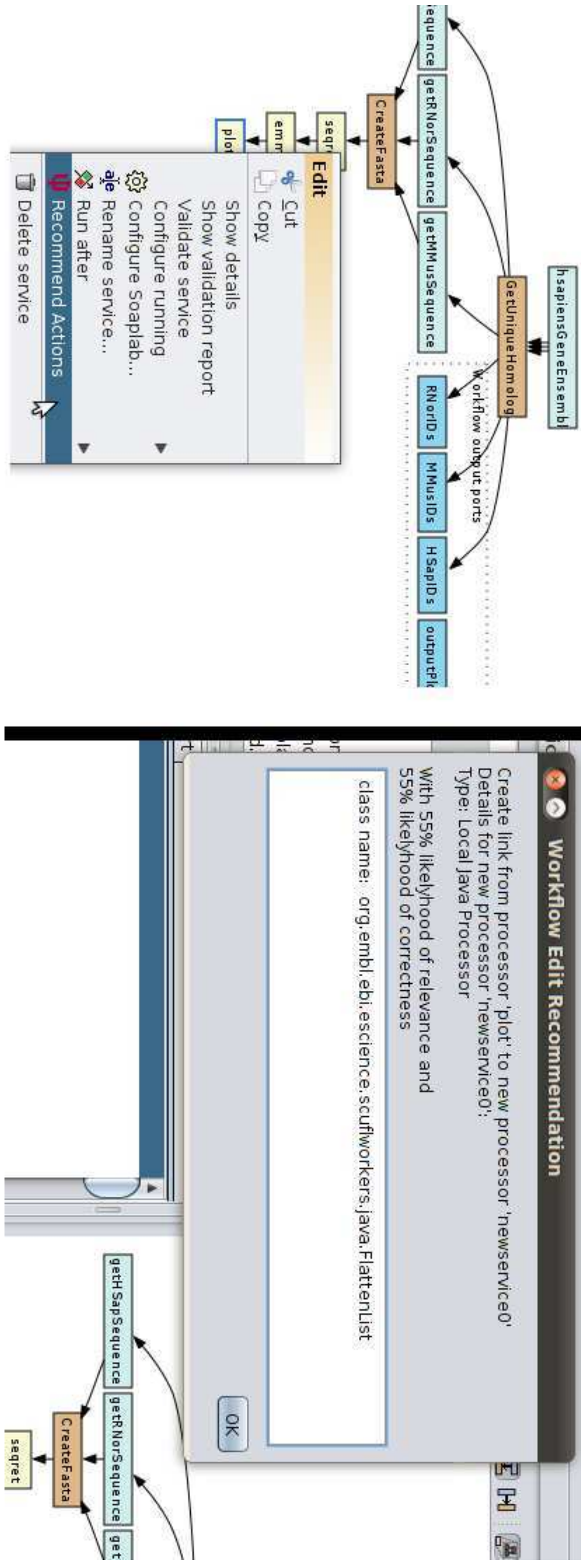
The fourth chapter, “Comparison and Indexing of Structured Cases”, looks into the knowledge representation, storage, and retrieval issues important to case-based reasoners using structured cases. It begins with a discussion of desiderata of a system addressing these issues. Following that is an overview of existing work with structured data in reasoning systems and how these systems compare and/or map structures to one another. The next two sections introduce the Structure Access Interface (SAI), an abstract structure storage and retrieval library developed for use with Phala and other CBR systems requiring structured cases. Particular attention is paid to the sub-tasks identified through SAI in the retrieval process. SAI’s retrieval performance is evaluated in the final section.

The fifth chapter, “Evaluation of the Phala Workflow Edit Recommender System”, begins by discussing several metrics for evaluating the performance of Phala or a similar system recommending in a synthesis task. Next, a series of tests designed to measure Phala’s performance on each metric are explained. The next section explains some of the implementation choices made within Phala, addressing the more abstract issues presented in Chapter 3. Following this, results from performing these tests are analyzed for recommendations generated through Phala’s extraction method as well as those which additionally are processed by Phala’s hybrid adaptation phase.

The final chapter, “Conclusions”, determines the findings of this dissertation work as well as the many avenues this work opens for future related work, including work in ensembles and hybrid systems, workflow authorship assistance, reasoning with structured data, and comparison and indexing of structures.

1.3 Contributions from Published Papers

Portions of this thesis appear in works published by the author and other collaborators: [60, 61, 50, 51, 53, 52]. Other published works contain material directly related to material within this thesis: [21, 21, 62].



Workflow Edit Recommendation

Create link from processor 'plot' to new processor 'newservice0'
Details for new processor 'newservice0':
Type: Local Java Processor

With 55% likelyhood of relevance and
55% likelyhood of correctness

class name: org.embl.ebi.escience.scuflworkers.java.FlattenList

OK

Figure 1.2: The Phala Recommender Plugin for Taverna

Chapter 2

Intelligent Assistance for Synthesis Tasks

2.1 Introduction

Intelligent assistance for authors of scientific workflows is a relatively new area of research developed in association with the recent excitement surrounding e-Science. Workflows have had a longer tenure, mostly through use in business processes for which a wider body of work in intelligent authorship assistance has been developed in case-based reasoning and other sub-disciplines [94]. For instance, case-based reasoning has only been applied to e-Science workflows since 2007 [121], whereas case-based reasoning was applied to the business process domain several years earlier [118].

Much of this work accompanies older work on intelligent assistance of authorship of other types of structures such as plans. Like workflows, plans can be represented with a directed graph, where each node in the graph represents a sub-goal that may

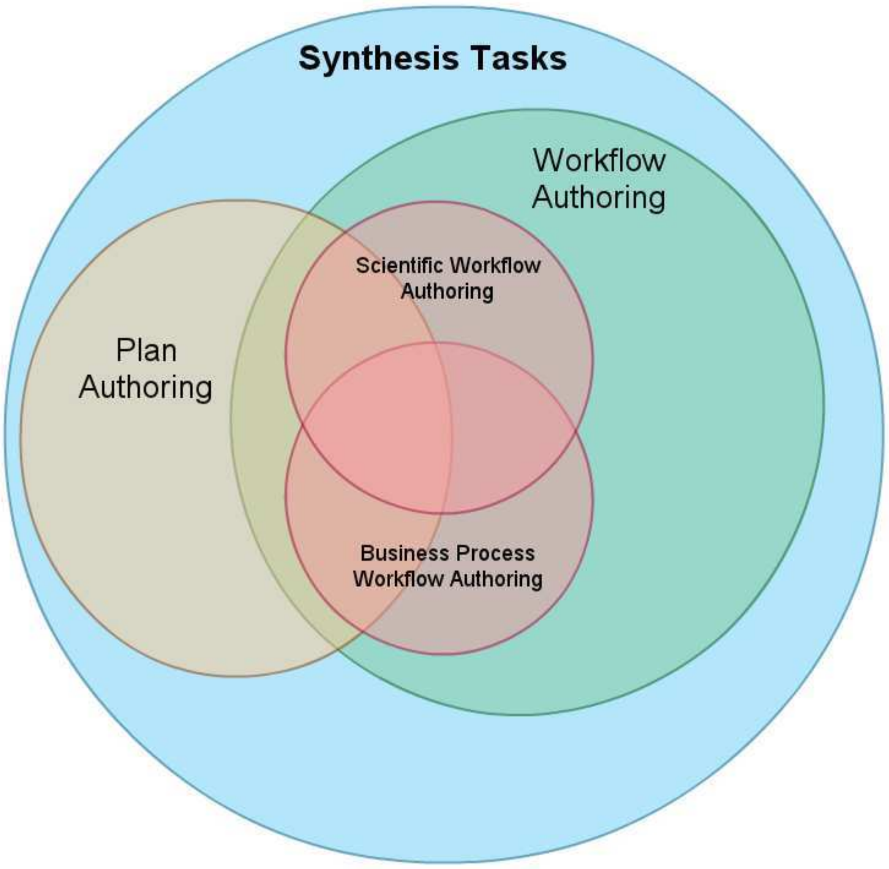

Synthesis Tasks
Workflow Authoring
Plan Authoring
Scientific Workflow Authoring
Business Process Workflow Authoring

Figure 2.1: A Venn diagram displaying the overlap between relevant synthesis tasks.

be a pre-condition required by other sub-goals. Such relationships are represented by edges in the graph. Because there is a correspondence in the formal methods used in both domains, there is also overlap in the benefits of studying any one domain. It is the goal of the research performed for this dissertation to discover techniques that may be applicable beyond the domain of e-Science and, in particular, may be applicable to other synthesis tasks, such as plan authorship. Other domains that are more related to the e-Science domains, such as business processes, which also use workflows, will benefit even more from the research presented through this dissertation. See Figure 2.1 for an illustration of the relationship between the domains discussed in this introduction.

Synthesis tasks comprise the unifying category for all domains that are relevant to this research. This is in contrast to assisting in analytical tasks, for which the sought solutions are much less complex; they are frequently as simple as the title of a movie or the name of a restaurant. The simplicity of these solutions allows the problems themselves to be modeled as a classification or regression task. Such tasks are particularly amenable to machine learning techniques.

Such machine-learning techniques are often employed by recommender systems – systems that interact with the user through recommendations of actions for the user to take. Assistants that adhere to this established mode of user interaction and are applied to analytical tasks have been well received by users and developers alike. Such a consensus has not been reached, however, on how to interact with users seeking assistance in a synthesis task.

This chapter will explore the application of the recommendation-based mode of intelligent assistance to synthesis tasks through the lens of assisting authors of scientific workflows. Different perspectives on how to interact with users in scientific workflow authorship, as well as related tasks, will be examined, including their implications for reasoning techniques employed in the task. Additionally, the gap between existing analytical recommender systems and the application of the recommender concept to synthesis tasks will be bridged, including identification of new concerns in this expanded task and important criteria for evaluating such systems. The work in this chapter serves as the basis of the Phala system for assistance of authors of scientific workflows, which is built upon two frameworks discussed in the next two chapters.

2.2 Recommenders as Assistants

Recommender systems are systems that provide *recommendations* to a user for a particular choice in the face of a myriad number of alternatives. The topic of recommender systems has recently gained much popularity. Workshops [104, 29, 9], many special journal issues, and now an ACM conference series[1] have all sprung up through the last decade on the heels of financially successful applications of the concept by companies such as Amazon [64]. These companies profile users of their software and recommend *items* to the user, such as, in Amazon's case, products a user may be interested in purchasing. In other cases the items may be media, such as movie recommendations made by Netflix [11]. Other times, the item is not intended for purchase at all but instead improves a user's experience with the software, such as in the case of Last.fm in which songs recommended for the user are automatically streamed through a personalized radio station.

[1]ACM Conference on Recommender Systems, 07 - current

Collaborative Filtering is the main technique applied to recommendation and most e-commerce recommenders employ a technique that was derived from traditional collaborative filtering, such as Amazon's "item-to-item collaborative filtering" [64]. Users are represented as a vector of preferences towards each of these items. A similarity metric is used to determine similarity between users, and items are recommended based on the frequency with which similar users have shown preference for the items.

Each of the previously mentioned systems assists the user with a task, whether it is shopping or selecting entertainment. The recommender systems provide assistance in the form of a single step in a user's task, acting as an *assistant* that provides advice in a somewhat constrained manner. However, recommender systems can assist in more complex tasks than those mentioned. One familiar scenario is word processing, where recommendations can be made for spelling, word choice, and grammar. In another scenario, the complexity of the task may warrant the use of a recommender, such as in design with CAD software. For example, Autodesk Research has developed a collaborative filtering technique to recommend commands a user may want to use, easing the user's burden of familiarizing themselves with the immense and consistently growing command-set of AutoCAD [71].

2.3 Assisting In Synthesis Tasks

This dissertation focuses on what are called **synthesis tasks**. Most of what was mentioned in the previous section is known as **analytical tasks**. In these cases, recommendations serve as the solution to a known problem:

- "*What product should I buy?*"
- "*What command should I use?*"
- "*What song should I listen to?*"

Many existing machine learning techniques can be easily applied to these problems, such as those used in classification. These types of recommenders commonly use a variant of collaborative filtering [39].

Synthesis tasks involve the design of complex structures, such as plans [54], state machines [46], concept maps [19], process models [54], or workflows [61]. When providing assistance for such tasks, there isn't a similarly well-defined problem to provide simple answers to. In fact, determining what the problem is embodies much of the task for assistants within a synthesis task. For instance, each word in a text document represents an opportunity to recommend a word-choice edit. It is up to the assistant to determine which words represent a problem in the larger context of the document. Since these tasks cannot simply be approached by seeking a classification or some other single value for an input, it is not straightforward to apply techniques aimed towards analytical tasks [3].

Throughout the rest of this section, reference will be made to two examples of assisting in an analytical and a synthesis task. The analytical task will be shopping. In this task, the user is seeking to purchase a product, and the recommender system may be aware of the user's likes, dislikes, shopping habits, social network, etc. The goal of the recommender is to produce a product or list of products that the user is most likely interested in purchasing. The synthesis task will be a sub-problem of a word processing assistant's task. In this task, the user will be writing an article and may wish for recommendations pertaining to word choice within the article. These recommendations may include a suggestion for a more appropriate term or one that avoids repetitive verbs. The recommender system will again be provided with context and must supply the user with words in the user's document that the user would want to replace along with the words the user would want to replace them with. Recommendations of this nature will further be referred to as *synthesis recommendations*.

2.3.1 Iterative Recommending of Problems and Solutions

A synthesis recommendation has three components: a problem, a solution, and a confidence value or multiple confidence values. The problem denotes the context in a structure where a change is to be made, and the solution denotes the actual change. Confidence values indicate the degree to which the system is confident that the recommendation represents a positive step in editing the structure. This confidence may be split among the problem and solution components. The distinction between these concepts is important not only for the purposes of user interaction but also in evaluating the performance of the recommender system.

In the product recommendation task, the problem to be solved is known: "*What product would this customer most want to buy?*" and the solution is simply a product or product category. In contrast, word choice revisions would not take the form of a simple classification, since such a solution lacks the context required to understand the specifics of the problem to be solved. In the latter case, recommendations of revisions must also provide some further specification of the problem itself, identifying the relevant context. That is, they must provide the location in the document of the word to be replaced (the problem) along with the word to replace it with (the solution).

For example, a recommendation may be to replace the word “dog” with “mutt”. Such a recommendation is more complex than a product recommendation, as it clearly has two components. The component identifying the problem is that the word “dog” must be replaced, and the proposed solution is to replace it with “mutt”. This can be further complicated when considering that “dog” itself may not uniquely identify the problem. It may be that each instance of “dog” should be replaced with “mutt”, but depending on the context, a sentence such as “She picked up the dog from the pound” may be preceded by a sentence such as “Men are such dogs!” In addition to specifying the word to be replaced, the recommender must also identify whether it is intended to be a global or local replacement, and if it is local, precisely which specific word instance it applies to.

2.3.2 Unique Concerns

Beyond the issue of synthesis recommendations being composite, there are other concerns that are unique to recommending in synthesis tasks, which are described in the following sections.

2.3.2.1 Overlapping Problems

Consider a generalization of the word replacement task in which any arbitrary edit to the text may be recommended. A **problem** can be as general as “*revise the introductory paragraph*” or as specific as “*replace the n-th word in the m-th paragraph in this text*”; in every case the problem consists of a task specification and some subsequence of words in the text, and some problems generalize over other *sub-problems*. Recommended problems must be *answerable* in that they identify all of the context necessary and only the context necessary for providing a *solution*, which, in the latter problem example, would be the replacement word.

2.3.2.2 Withholding Recommendations

An additional consideration is whether any recommendation should be made or not. In work on LookOut, a calendar assistant, Horvitz discusses the importance of judging when and how an intelligent assistant should intervene and offer assistance to the user since frequently offering assistance when none is needed may make the assistant less valuable than not having an assistant at all [47]! This differs slightly from the shopping scenario (product recommendation) in which it is known that the customer is looking to buy *something*. Otherwise, browsing an e-commerce site would not be a very productive use of his or her time. Therefore, a product recommender will make a recommendation if it can, only withholding recommendations in the case that it has insufficient confidence in any recommendation, not because it believes the customer is not interested in making a purchase. However, in the word replacement task, every word represents a potential problem, but it's also possible that none of these are actual, existing problems! Additionally one may seek assistance only to confirm a belief that the text is in no need of further revision. Such is also the case in any other incremental design task.

2.4 e-Science Workflow Authorship

2.4.0.3 Workflow Uses in e-Science

Information technology is playing an increasingly critical role in scientific experimentation, particularly in *e-Science*. E-Science consists of *in silico* experimentation (experiments that are run completely through execution of computer programs) as well as computational processes intended for data analysis and knowledge discovery [84].

Workflow technology is often employed in e-Science to manage these processes. Workflows developed for these purposes are called *Scientific Workflows*. Scientific workflows consist of references to services (remote web-services, scripts, and local programs) and instructions for controlling the order of their execution and the flow of data between them. These services enact simulations, analysis, or transformations of data.

Figure 2.2 depicts a scientific workflow in the workflow editing environment XBaya [99]. This example is a weather simulation experiment using the WRF forecasting model performed in the LEAD project [28], a project for mesoscale meteorology research and education. Input data is represented by the left-most node labeled "Assimilated ADAS Data/Config." Intermediate nodes represent web services that process data from the preceding nodes, and the edges between them represent data flowing from one service to another. Finally, the output of the workflow is represented by the right-most node labeled "WRF Output Data/Config."

The depicted workflow is small, but scientific workflows may contain one hundred steps or more, making them challenging for humans to compose. Also, the number

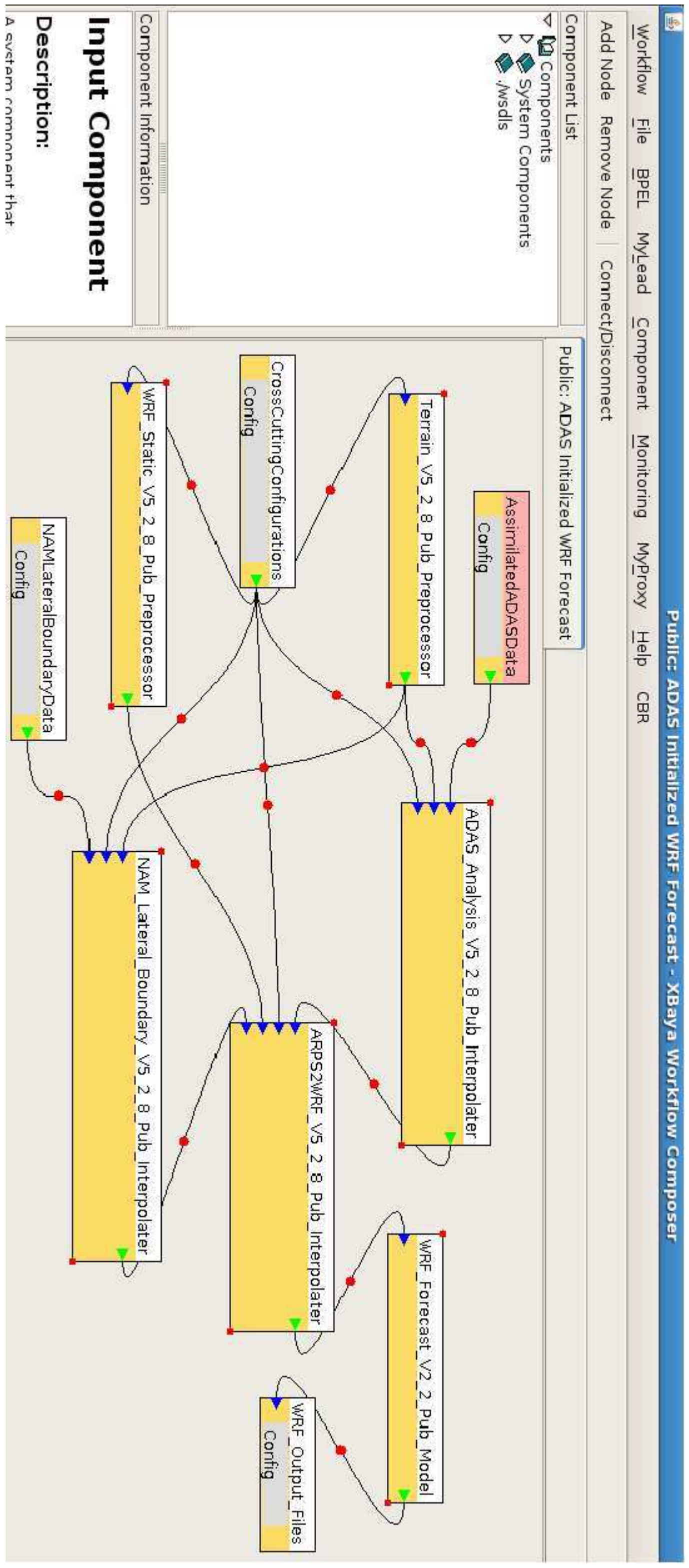

Public: ADAS Initialized WRF Forecast - XBaya Workflow Composer
Workflow
File
BPEL
MyLead
Component
Monitoring
MyProxy
Help
CBR
Add Node
Remove Node
Connect/Disconnect
Component List
Components
System Components
./wsdls
Component Information
Input Component
Description:
A system component that
Public: ADAS Initialized WRF Forecast
AssimilatedADASData
Config
Terrain_V5_2_8_Pub_Preprocessor
CrossCuttingConfigurations
Config
WRF_Static_V5_2_8_Pub_Preprocessor
NAMLateralBoundaryData
Config
ADAS_Analysis_V5_2_8_Pub_Interpolater
NAM_Lateral_Boundary_V5_2_8_Pub_Interpolater
ARPS2WRF_V5_2_8_Pub_Interpolater
WRF_Forecast_V2_2_Pub_Model
WRF_Output_Files
Config

Figure 2.2: A Workflow in XBaya

of workflows that a scientist runs may be large. For example, ensemble simulation workflows run hundreds of highly similar workflows, differing slightly in structure or in parameters, to perform a parameter sweep study. In addition, the amount of data produced and consumed by services on the grid can be extremely large, and processing times long, making it important to generate the right workflow on the first attempt to avoid wasting computational resources.

In general, workflows can be represented by labeled directed multi-graphs in which nodes represent services and edges can represent either control or dataflow between these services as well as input and output from the workflow itself. Details of these services are captured in labels affixed to nodes. Datatypes and input or output ports (names associated with parameters or data products from a service) may be labels of the edges between services. These labels may be associated with a domain ontology to enable type-checking and composition assistance based on generative planning.

2.4.0.4 Differentiating e-Science Workflows from Business Processes

Workflows are not a new technology developed for e-Science. They have existed for quite some time for use in business processes [94]. Existing workflow technologies such as the BPEL language for modeling workflows have been used in scientific contexts, but more and more, e-Science is moving toward development of new tools that more closely fit the unique challenges presented to its researchers [129].

One important aspect to understanding how scientific workflows may differ from others is their intended role. Scientific workflows are intended for application to novel problems and to discover new knowledge [35]. This requires flexibility on the part of the intelligent assistant. For example, developing workflows for business processes involve discovery of rote procedure within a business, for which workflow mining techniques are often used [96]. Authors of scientific workflows are seeking to develop novel processes, so the entire process cannot solely be mined from execution logs. However, such records may serve to assist in the process when the workflow being developed involves use of common sub-processes, as described in later sections.

Another important distinction is that, in e-Science, workflows are *computational* [25]. In this sense, services represent defined and accessible computational processes, and workflows manage control and data flow between these processes. Other workflows represent processes in a more abstract sense, including those performed by human agents such as employees. An additional difference, and challenge, is that scientific workflows deal with very large datasets and extremely expensive computations. Scientific workflows can run on supercomputing grid middleware on the order of weeks before completion of execution. Scientific workflows are also often deterministic and fully automated. Non-determinism typically only comes into play for the purpose of fault-tolerance, but it is much more common-place in business processes where the goal is to react to a dynamic environment rather than test a hypothesis.

2.4.0.5 Foundational Assistance for Scientific Workflow Authors

e-Science researchers are answering the call for tools to meet the specific needs of scientists using workflows and have produced a variety of tools for developing and enacting workflows, sometimes called *Workflow Management Systems*. These tools provide a foundational means of modeling experiments and executing workflows on grid middleware. These tools are not intelligent assistants in the sense that they do not play a part in the problem-solving aspect of developing a scientific workflow for an experiment. However, they do provide a rich graphical interface for manipulating workflows and thus an ideal environment for incorporating an intelligent workflow composition assistant.

Each of the following workflow systems provide a graphical user-interface for workflow authorship and execution. XBaya is a graphical workflow composer that works with the BPEL language for modeling workflows [99]. Kepler, Taverna, and Trident are open-source projects which, in addition to supporting workflow modeling, offer services for enactment and management of scientific workflows [83, 5, 8].

2.5 Modes of User Interaction for Intelligent Workflow Authorship Assistants

So far, systems that assist users with the mechanical construction of workflows have been discussed, but these tools do not assist with the creative process. They do not have a working knowledge base of the problem being solved by the workflow and cannot assist with the user's decisions in how to solve the problem for which the workflow is being developed. This section will examine systems that extend foundational toolkits for developing workflows by offering direct, intelligent assistance in the authorship process. In particular, motivation will be provided for using recommendation-based assistance in the task of assisting authors of scientific workflows.

2.5.1 Mixed-Initiative Interaction

In Artificial Intelligence literature, an important distinction is drawn between agents that perform tasks autonomously and agents that interact with a human user to collaboratively achieve a goal. The latter type of intelligent assistant is said to take a *mixed-initiative* approach by sharing the initiative in the problem-solving process, though the exact definition of what *initiative* means in this context is disputed [82]. To shed more light on the important aspects of mixed-initiative computing, Horvitz mentions several principles for designing mixed-initiative intelligent assistants that stress the importance of the following: [47]

- shared problem-solving effort between the user and the agent with deference to the user
- dialog between the user and agent to resolve conflicts
- reasoning on the part of the agent over when and how to intervene, including weighing competing costs and benefits when deciding when and how to interact with the user to maintain a "value-added" presence

Myers et al. prescribe models of user-interaction based on the needs of the user. In particular, they recommend that for tasks in which a user's cognitive skills are needed, it is best for an intelligent assistant to avoid "strategic decision making" and focus on "routine tasks" that can be more easily automated, thus allowing users to focus their energy on the contributions for which they are needed [80]. They also recommend that the assistant should additionally share problem solving responsibilities with the user in cases where both have significant ability to solve problems related to the task.

Machine-Directed	User-Directed	Mixed-Initiative
Pegasus/Wings	**myExperiment**	**Phala**
GTrans	DWMSS	**CAT**
		KANAL

Table 2.1: Modes of user interaction for various authorship assistants.

2.5.2 Further Delineating Modes of Interaction for Assistance of Scientific Workflow Authors

Developing scientific workflows falls somewhere between these two modes of operation recommended by Myers et al., given that the users are uniquely qualified in understanding the subject matter and the experiment goals, whereas the intelligent assistant can uniquely contribute by searching large collections of services and workflows for important ingredients to aid in the workflow design. Through this sub-section, three distinct categories of user assistance, which follow from Myers et al's model of user-interaction, will be identified from related works, which are summarized in Table 2.1. Assistants for the e-Science domain are identified in boldface.

2.5.2.1 Machine-Directed Assistance

In this mode of user interaction, the user will specify the goals of the workflow to be developed, and the intelligent assistant will take full initiative to develop the workflow. The specification of the goals of the process is not a trivial task, however, and may be an extensive process involving a conversation between the user and the intelligent assistant to develop a goal specification, in which an automated planner may be invoked to test the soundness of the goal specification.

One example of a system utilizing this form of interaction lies outside the domain of scientific workflows: GTrans [23]. This system works alongside the Prodigy automated planner, which develops plans for the user's specified goals, allowing GTrans to assist the user in modifying the goals until a satisfactory plan is developed by Prodigy. A good example of a system within the workflow authorship domain is the work by Ambite et al. [6], which uses the PowerLoom planning system to generate scientific workflows.

Wings/Pegasus is a set of tools that also take the machine-directed assistance approach to assisting scientific workflow authors. Among others, Pegasus/Wings is used by scientists at the Southern California Earthquake Center (SCEC) for experiments such as simulating seismic activity. The workflows used in these simulations can be very large (thousands of components) [36]. Such large workflows are particularly difficult to develop, making the automatic generation feature of Pegasus/Wings very desirable for these users.

2.5.2.2 User-Directed Assistance

In a paper discussing lessons learned from developing e-Science collaboration tools [98], Goble et al. note the disconnect between some knowledge-intensive generative-planning tools and the users they are intended for:

> ...in Taverna we worked on fancy knowledge-based descriptive techniques for services so that workflows would be composed automatically. It turned out that this wasnt what the users wanted at all. They wanted quick ways

of finding a relevant service and then they wanted help for them to build workflows themselves.

It is important to note that many of Goble et al's users are developing significantly smaller workflows than those developed at SCEC, often on the order of tens of nodes with a ceiling in the low hundreds. These workflows mostly consist of bioinformatics experiments but also include topics as disparate as socio-informatics.

The principle that can be drawn from this perspective is that expert users often wish to be primarily in control of the authorship process. Perini et al. echo a similar perspective in their work developing a system that aids expert users in planning for forest fire intervention [89]. This principle should be seen as a guiding factor rather than a rule. It is tempered by the complexity of the design process, which in the case of SCEC, was high enough to make a generative planning approach attractive to expert users.

One means of keeping the user at the helm is to create an assistant that only helps the user in the sub-task of searching for relevant past works, which can serve as examples and sources of inspiration in their task. This type of assistance will be referred to as *User-Directed Assistance*.

Assistance from case-based reasoning often falls into this category when the system in question retrieves cases for the user to analyze and re-purpose. Kim et al. developed a case-based tool for modeling non-e-Science workflows, DWMSS, which retrieves workflows based on similarity to a negotiated input-model and allows the user to either adopt the retrieved workflow or integrate it with the new input model and store it for use in future assistance episodes [56].

In the domain of scientific workflows, Goble et al. developed myExperiment as a consequence of their observations noted above. Myexperiment is a website that allows scientists to share information and provide feedback for each others' experiments. This is mainly intended to provide a means for discovering, re-purposing, and re-using workflows from prior experiments to reduce the amount of time a scientist spends composing workflows and to provide creative insight into which services and techniques may be most useful for a particular task. One method used to organize these workflows is to allow semantic tagging. These tags identify what tasks a workflow is relevant for and which significant techniques are used within the workflows. Workflows can also be discovered through browsing by author. MyExperiment can also be classified as user-directed assistance, as its service for browsing scientific workflows fits this mold.

2.5.2.3 Mixed-Initiative Assistance Through Recommendation

In user-directed assistance models, the user is given full authority over what knowledge from prior cases is incorporated into their work, which was noted as an advantage of the approach. The associated trade-off is that the user is responsible for extracting relevant knowledge from the prior works and also for incorporating it into their work, which machine-directed interaction did not require of the user. Neither the user-directed or machine-directed approaches provide the ideal benefits of a mixed-initiative system in that initiative *during* the authorship process rests solely on the user in user-directed assistance and solely on the assistant in machine-directed assistance. A more balanced approach would better bestow the benefits of mixed-initiative interaction.

An approach to user assistance that balances between user-directed and machine-directed assistance is a recommender system, as discussed earlier in this chapter. A recommender system carries benefits of both approaches: users are not burdened with developing a formal specification of their problem, nor are they burdened with extracting relevant knowledge from the prior works that are retrieved. The task of authoring the workflow is still directed by the user, but the assistant can now play a part in the creative process as well by making targeted recommendations for edits to partially authored workflows.

A project that offers mixed-initiative assistance in the e-Science domain is the Composition Analysis Tool (CAT) project, developed by Kim et al. [55]. This tool validates workflows and provides an interface that recommends edits to correct errors. A planning framework is used to generate recommendations for node additions and replacements that move the workflow toward the goal of producing the desired outputs, at times allowing for abstract nodes to act as place holders for concrete, executable nodes.

Xiang et al. mention a recommendation-based usage scenario of their tool, which informs users of services that can be added to a partially completed workflow based on a service linkage graph developed by examining prior workflows [121]. KANAL is another intelligent assistant that falls into this category [54]. KANAL offers "error-correcting" recommendations to authors of process models, which are similar in concept to workflows.

In a recommendation-based model of user interaction, the user remains in principal control of the authorship process. The recommender does not commandeer the interface, yet it is able to take initiative in performing some of the more mechanical reasoning and decision-making tasks the user would otherwise be saddled with and it leaves the remaining reasoning to the user by making recommendations. Recommendations are a low-impact means of computer aided assistance that can be accepted or rejected at the user's discretion, and at a pace that is most comfortable to the user. These aspects fall in line with most of the principles of mixed-initiative assistants laid forth by Horvitz [47]. Recommenders capture the important wants and needs addressed by Goble et al., while taking assistance a step further by cutting through irrelevant details of prior works.

Recommendation in a synthesis task such as workflow authorship may take one of two forms. In the first, the recommender system will recommend both potential problems and their solutions. In the latter, the system will only recommend solutions, and the user will identify a partial or full description of the problem they are encountering. In the latter case, this could be as simple as selecting a node in the workflow representing the source of any recommendations they wish to see. Both modes of operation can be supported by a single application, which endows the application with more flexibility and the ability to adapt to a particular user's preferred method of interaction.

The Phala project was developed as a part of this dissertation work to assist authors of scientific workflows. Phala recommends "next steps," or extensions, from a partially authored workflow to an incrementally more developed workflow. The goal of providing these recommendations is twofold: first, to inform users about data and services that they might not have considered when authoring a workflow (e.g., by making new recommendations based on the workflows of others), and second, to make familiar components conveniently accessible for re-use (e.g., by making recommendations based on similarities to the user's own prior workflows).

2.6 Performance Metrics for Individual Recommendations

An important consideration for any user assistant is how to evaluate the quality of the assistance offered. Evaluation of a recommender system endeavors to determine what many call the "goodness" of a recommendation [73]. This term is intentionally vague, as the quality of a recommendation contains many dimensions, several of which will be explored in this section. These dimensions are often collapsed into a single rating for the recommendation.

Goodness may also be referred to as *confidence* in the case where the system is self-evaluating its own recommendations. In this case, the system is estimating the goodness of its recommendations, whereas goodness is measured through real-world application of the recommendation. Additionally, users can specify how they perceive goodness by indicating which of the performance metrics they prefer, allowing the system to specify confidence through a weighted average of several performance measures and tailor it to a specific user's definition of goodness.

Evaluation of the performance of a recommender, either externally through assessment of goodness or internally through assessment of confidence, ultimately serves three purposes:

- The first is the most obvious: to determine the efficacy of the system and diagnose its faults. Such evaluations are necessary to scientific endeavors.

- The second is less obvious and will be an important topic in this chapter: to offer a degree of explanation for the recommendations made to the users of the system. This is particularly important in a synthesis task in which the user may be unsure of the wisdom of integrating a recommendation. Riely et al. note that a modern recommender system must have a component that can "explain or justify" its recommendations to its users [95].

- The third is to allow introspection within the system. Horvitz notes how confidence estimates may result in different levels of intrusiveness of communication by the intelligent assistant [47]. Riely et al. also note that a minimum confidence threshold should determine whether a recommender should make any recommendation at all, or simply report that it doesn't know what to recommend [95]. Chapter 5 will discuss in detail how confidence estimates can play an important role in developing hybrid recommenders.

Recommendations in synthesis tasks bear another similarity to explanation, apart from the role of confidence estimates. A recommendation, such as those generated by Phala, attempts to point out to the user a gap in a design, a source of incompleteness or incorrectness, and a repair. This is similar to the process of explaining an anomaly to repair a gap in understanding, examined by Leake, who encouraged three criteria for evaluating such an explanation: relevance, plausibility, and usefulness [59]. These three have been altered and incorporated with two additional metrics, which are all described in the context of recommendation through the remainder of this subsection. Of note, Tintarev also identifies a set of criteria for evaluating explanations of recommendations that partially aligns with the criteria identified below

[112]. However, Tintarev's criteria focus more on a user's perception and use of an explanation, whereas the criteria below are more of an independent judgement of the recommendation itself.

2.6.0.4 Relevance

Relevance addresses whether or not the recommender is generating a response that has bearing on the problem of interest. It can be determined solely from the problem component of the recommendation. For instance, if a recommendation were made to add a new processor whose output was tied to an existing processor, but the existing processor already had sufficient input to execute, the recommendation would be deemed *irrelevant* since it attempted to correct a problem that didn't exist.

2.6.0.5 Correctness

Correctness pertains to whether the solution component of a recommendation indeed solves the problem the recommendation identifies, and it is most similar to plausibility within Leake's framework for evaluating explanations. The recommendation need not be relevant in order to be correct. For instance, in the prior example, if the recommended service was indeed of importance to the execution of the existing processor, regardless of whether that processor had sufficient input, it would be correct.

2.6.0.6 Usefulness

So far, we have mentioned criteria that could be judged independently of the user. The user could deem a recommendation as irrelevant, and it could be possible that the user was wrong in doing so. *Usefulness* is a quality that, on the other hand, can be solely and subjectively judged by the users themselves. The usefulness of a recommendation is the degree to which it helps the user solve the problem. For instance, recommending a service the user frequently involves in their workflows may not be particularly useful, since it only saves the user a minimal amount of work. However, recommending a service the user may be unaware of could save them a great deal of research, and would be more useful. This quality has also been described as novelty or serendipity in recommender system literature [44]. However, recommending in synthesis tasks broadens the meaning of usefulness beyond serendipity.

Usefulness could also be viewed in a less subjective way by examining the extent of a recommendation. Perhaps a problem within a workflow is that there are two processors that should be, in some way, linked. Processor A produces data that processor B requires, but not in a format that processor A understands. Assume also that there is no single shim service [6] that can translate the data directly, but a sequence of two or three shim services can. A recommendation could be to add all of these shim services in a sequence of processors between processor A and processor B. This recommendation would be more helpful than simply recommending one of the services, since that only partially solves the problem.

It's important to note, in the latter (syntactic) sense of helpfulness, that there is a conflict between helpfulness and correctness. As Leake points out in reference to explanations, plausibility is tied to Occam's razor. That is, the more complex an explanation is, or in Phala's case, a recommendation, the less likely it is to be plausible, or in Phala's case, correct. A user seeking recommendations that are more helpful may need to sacrifice a degree of correctness in the recommendations they receive.

2.6.0.7 Diversity

If presenting multiple, mutually exclusive recommendations to a user, the collection of recommendations should have some degree of diversity. For instance, if a system recommended “Superman 1”, “Superman 2”, and “Superman 3” to a movie-goer interested in a super hero movie, there isn’t much choice left for the user, who may actually be more interested in a “Batman” movie, thus defeating the purpose of providing multiple recommendations.

Diversity is a unique performance metric in that it is not actually a judgement of a single recommendation, as each of the others were, rather it’s a judgement of a collection of recommendations. The topic of judging lists of recommendations, particularly in terms of diversity, is explored in detail by Ziegler et al. [130] and in the context of case-based recommendation, by Smyth et al. [103]

2.6.0.8 Speed

A very natural performance metric for any computational task is speed. Recommender systems are by nature interactive, so speed is a particularly important performance metric. Speed differs from each of the other metrics in that it can be viewed in a completely objective manner. It can be directly measured in units of time. However, it is perceived subjectively by the user. For instance, the user doesn’t need to know how long it took to form a recommendation to know that it was *too long.* There are very clear diminishing returns in regard to this metric, and the subjective perception of time by the user must be taken in to account. For instance, a recommender that takes only 0.1 seconds to produce its recommendations may be twice as fast as a recommender that takes 0.2 seconds, but in most cases the user will not notice a difference. It may be more advisable for the system to use this time to generate better recommendations, depending on the user's preference for speed. This reflects the decisions made for SAI that allow for a degree of anytime processing for mapping structures.

2.7 Knowledge Sources for Supporting the Assistance Task

Supporting the assistance task, particularly in the case of recommendation, is often made possible through mining knowledge used in generating recommendations. For example, the collaborative filtering approach mentioned earlier in this chapter relies on records of preferences of a large number of other uses to generate its recommendations. This section will explore the use of an abundant source of useful data for recommendation in the scientific workflow domain: provenance records.

2.7.1 Provenance from Scientific Workflow Execution

Provenance is meta-data recorded during execution of a collection of integrated processes, which can be later used to re-trace the steps taken to derive a data product [100]. In the e-Science context, provenance traces the derivation of experimental data produced by workflows linking independent web services and other types of *processors* together [101]. Provenance is collected by provenance management systems, such as Karma [102]. Karma collects provenance through software sensors embedded in the services being executed. These services are represented as processors in workflows, operating independently of the workflow management system used. The information collected can be mined to produce workflow execution traces that identify producer/consumer relationships for data exchange by these services.

When speaking of provenance in a case-based reasoning context, which most of the next chapter will entail, it is important to distinguish data provenance (also called external provenance) from case provenance (also called internal provenance) [62]. The latter refers to the derivation history of a case within a case base, tracking its development through each iteration of the CBR cycle. A case may be derived in this way through retrieval of a parent case, adaptation to solve a query problem, and retention of the result as a new case into the case-base. This contrasts with data provenance, which refers to any data artifact "in the wild" and is not constrained to a case-base within a case-based reasoning system. From this point forward, the term *provenance* should be taken to mean only data provenance and not case provenance. Specifically, we will be speaking only of provenance in the e-Science context: that which is collected from the execution of scientific workflows.

2.7.2 Differentiating Provenance from Workflows

Both provenance and workflows can be a source of knowledge about scientific processes, but should not be confused with one another. A workflow is a plan for carrying out a scientific experiment and developing data artifacts which comprise the results of the experiment. Workflows can adapt to changing operation states by conditional invocation of processors. Control structure within workflows can also include conditional early termination of the workflow and loop constructs. Additionally, failures and other such phenomena during the execution of a workflow may alter the expected sequence of events or the expected final product. In short, the execution of a workflow is not static, nor even deterministic, and many parts of a workflow may never be executed.

In contrast, provenance is a record of what occured during the execution of a workflow [32]. A provenance record does not include any reference to the portions of workflows which were not executed, and in fact doesn't contain explicit records of the control structure of the original workflow (though in many cases such control structure can be inferred from examination of multiple executions of the same workflow [126]). Provenance records also contain information which workflows do not. As mentioned, the exact sequence of actions taken by a workflow may not always be predictable, but a provenance record indicates exactly what did happen. The information in a workflow is more of a hypothetical specification of what could happen. Additionally, provenance records reference data artifacts produced throughout the execution of the workflow, and additionally at its conclusion. Workflows may only reference data existing before the execution of the workflow.

2.7.3 Representing Provenance with SAI

Phala consumes provenance in the Open Provenance Model (OPM) format. OPM was developed to enable inter-operability between many systems which either record or consume provenance. OPM was developed as a joint venture between many researchers in the e-Science field through a series of "provenance challenge" workshops.

The website for OPM, *http://openprovenance.org/*, links to the most recent specification (1.1 as of this writing) as well as a number of applications that work with the OPM standard.

OPM graphs store information about artifacts (data) and processors, among other entities (such as users). Edges between these entities define relationships, such as a processor producing an artifact. When mining OPM, Phala transforms these provenance graphs into traces of data flow between services, replacing artifacts with edges that directly connect processors.

2.7.4 Mining Provenance

Data flow between services within an instance of workflow execution can be derived from provenance traces. At times, problems may arise in the recording of provenance data, particularly in cases in which semi-structured workflows are employed, causing gaps in the provenance record which reduce the amount of useful information that can be extracted from the data flow record by a tool seeking to assist authors of scientific workflows. Though incomplete records are still useful to a tool like Phala, research is currently underway to develop automated systems capable of correcting incomplete provenance traces [21].

Given this limitation, it may seem reasonable to instead mine workflows, as they also model data flow between services and would be similarly useful as a source of data for mining (in fact, a collection of workflows was mined to generate data for the evaluation in Chapter 5); however, this dissertation focuses on the importance of mining provenance because cyberinfrastructures using provenance management systems will reliably have this data readily available, without direct user intervention through a workflow publishing process. Workflows can vary greatly in specification language used and support for various control and data flow patterns [113], but execution traces mined from provenance afford a simplified view of the execution of a workflow, while providing sufficient detail to support the task of assisting workflow authors. Provenance can be communicated in a standard format for inter-operability through OPM, a goal that workflow technologies are significantly less likely to reach, given the many varied and incompatible formats available.

Chapter 3

Case-Based Generation of Edit Recommendations

3.1 Introduction

Chapter 2 explored the problem of recommending in a synthesis task by examining the problem of assisting authors of scientific workflows. This chapter details a specific solution to this problem, which meets the goal of providing knowledge-light assistance to users through application of case-based reasoning.

Traditionally, the case-based reasoning cycle is broken into four steps: retrieval, reuse, revision, and retention; however, this framework does more to identify problems to be solved by the CBR application than it does to prescribe a general solution [1]. Each step in this cycle presents unique challenges to CBR researchers aiming to apply CBR to the constraints of a specific problem domain, and user assistance in synthesis tasks is no different. This chapter provides an interpretation of each of these subtasks (apart from retrieval, which is discussed in detail in chapter 4) that meets the needs of assistance in synthesis tasks and is exemplified through the implementation of Phala and its use in the scientific workflow domain.

Just as chapter 2 identified how the structured nature of problems in synthesis tasks complicate the structure of solutions, this chapter will explore how representation of complete cases is additionally complicated in synthesis tasks. This chapter will introduce a perspective on case representations, dubbed *gestalt cases*, in which there is no clear delineation between typical case components, such as problems and solutions. Rather, gestalt cases will represent not only a complete reasoning episode but also the completion of the overall design process, embodying the many sub-problems an intelligent assistant would be faced with throughout assisting with the design of a structure. The appeal of such a case representation lies in the reduced size of the case-base (an important consideration for retrieval of structured cases, as studied in chapter 4) and also the ease in mining cases from existing databases of completed designs. Mining gestalt cases comprises the implementation of the retention phase of the CBR process for Phala, in that Phala's cases are derived from records of the execution of completed workflows, called provenance.

This chapter additionally delves into how a CBR system should approach reuse for case-based recommendation in a synthesis task. In order to do so, a new concept will be introduced: extraction. Since gestalt cases contain a multitude of potential problems and solutions, the CBR application must extract these features, select among them, and adapt them for optimal effectiveness before presenting them to the user.

Phala's mixed-initiative approach taken to assistance also indicates the user's role in the revision process. Phala offers assistance to its users (domain experts), allowing them to have the final say in how their workflows are developed. Thus, the users themselves revise the recommendations presented by Phala by choosing whether to incorporate them. Provenance recorded from the execution of these new workflows then completes the cycle by allowing Phala to recapture and retain new cases containing these revisions through the mining of provenance.

In total, these new concepts present a novel perspective of the CBR cycle, which aligns with the requirements of recommending in a synthesis task explored in Chapter 2. This chapter begins by introducing the implications for knowledge-light and knowledge-heavy approaches to support. Next, gestalt cases are discussed in the context of provenance from scientific workflow execution. The next section further explores the scientific workflow domain by defining use-cases and recommendation types for Phala, which is followed up by the next section that discusses how to extract such recommendations from gestalt cases. The remaining sections explore the adaptation problem in CBR and how hybrid approaches can be effective in solving this problem for the recommendation task. Finally, the Hybrid Recommendation Framework (HRF) designed through this dissertation to address the needs of adaptation of recommendations for assisting in synthesis tasks is discussed through its integration with Phala.

3.2 Case-Based Reasoning and the Knowledge Acquisition Bottleneck

This section will explore the role of general knowledge (including ontologies and other manifestations of rules) for many of the reasoning systems discussed in section 2.5 and the relative advantages and constraints brought forth by varying degrees of reliance on such knowledge. The motivation for pursuing a *knowledge-light* (also called *knowledge-poor*) model, one in which relatively little general knowledge is leveraged, will be discussed as well as the relevance of this approach to the task of assisting scientists in constructing workflows. In particular, motivation will be provided for using a case-based approach, which replaces much of the requirements for general knowledge with episodic knowledge, which can be developed more easily by mining knowledge sources such as provenance records.

3.2.0.1 Knowledge-Intensive Intelligent Assistance

Ontologies and rule-based reasoning are leveraged by many intelligent assistants as a means of injecting expertise on the part of the assistant. These rules and definitions can be used by the assistant to validate choices made by the user and also to perform the duties of the user (in terms of generating content).

In the e-Science domain, the CAT tool, mentioned in section 2.5.2.3, relies on a sophisticated domain theory to perform its validation. Domain-specific ontologies are leveraged by its planning framework to form recommendations to correct errors, mistakes, and omissions. The Pegasus/Wings tools, mentioned in section 2.5.2.1, leverage knowledge-intensive service annotations to support their planning approach to workflow generation. Each of these tools rely on domain ontologies being built before the tools are applied to a new domain.

3.2.0.2 Knowledge-Light Assistance

In section 2.5.2.2, a perspective on the desires of scientists authoring workflows, which identified a desire for expert users to retain initiative in the authorship process, was shared by Goble et al. However, part of the problem identified in that quote was in reference to "fancy knowledge-based descriptive techniques". Frustration from this frequently-used technique in intelligent assistance is often due to what is called the *knowledge acquisition bottleneck*. Developing a sophisticated ontology of rules and domain-specific knowledge is a task that is much more difficult than one may realize, since problems often arise as the knowledge base is *scaled-up* to larger and more substantial domains [63]. Xiang et al. note that small domains used for testing may not illuminate the true difficulty in developing consistent ontologies on a larger scale [121]. As the success of wings for SCEC scientists has demonstrated [36], knowledge-rich service annotations can be very effective [72]. However, experts are required to develop ontologies and annotate services for this process to work. Additionally, agreement on an ontology is crucial for interoperability of a system. Special cases become a scourge for ontology developers, and maintaining a knowledge base as new services are developed, old services are changed or updated, or new domains are sought may require more work than what is gained by using the developed ontology. If assistance can be provided while side-stepping the knowledge acquisition process, a great burden will be lifted from the experts seeking to use such a system.

Case-based reasoning can provide a means for significantly mitigating the knowledge acquisition bottleneck. Reasoning from cases is a natural approach to addressing exceptional circumstances, unlike the brittle, inter-dependent constraints of a rule-based system. As seen with the myExperiment usage scenario, workflow authors often seek similar past works to re-purpose when designing a new workflow. From a case-based perspective, prior designs can be seen as cases in a CBR cycle. Similar cases are retrieved and adapted to solve problems with a partially-designed structure. When the structure is complete, it can be added to the case-base for future use in problem solving episodes.

While reliance on cases rather than general knowledge eases the burden of knowledge acquisition, it does not entirely eliminate it. Developing cases themselves can be difficult, though this process holds an advantage over development of general knowledge in that cases can be developed independently, since the definition of one case does not directly depend on that of any others. Additionally, case-mining can take the place of case authorship, in which cases are automatically created from available data sources. Cases are easier to automatically mine from data sources, since data being mined are often already segmented along the lines of cases, whereas rules must be inferred from generalizations of the data, requiring a more thorough analysis. Techniques for case mining will be explored for scientific workflow authorship assistance in Chapter 4.

Another constraint is that the adaptation phase in case-based reasoning often relies on a significant general knowledge base, or a case-based reasoner may be a component within a hybrid system that relies on a sophisticated ontology (such as CBRFlow, described at the end of this section). However, efforts have been made to mitigate the knowledge engineering burden in these stages as well. Adaptation

Knowledge-light	Knowledge-Heavy
myExperiment	GTrans
Phala	Wings
DWMSS	CAT
CODAW	**KANAL**
	CBRFlow

Table 3.1: Knowledge-light and knowledge-heavy workflow assistants

knowledge can also be mined, in particular from the case-base, though other sources may be used. This process will be discussed in section 3.6.

The Phala system was developed specifically to explore the extent of contributions a knowledge-light system can bring to authorship assistance, but is also capable of being augmented to incorporate knowledge-heavy techniques. Phala realizes the knowledge-light goal through its application of case-based reasoning. It does not rely on domain theories, which must be constructed on a domain by domain basis. Instead, Phala relies on mining its knowledge from sources that are aggregated by users without any need for direct intervention. This also furthers Goble et al's ideals on assisting scientific workflow authors.

Phala's knowledge requirements are noted in Table 3.1 alongside those of other systems discussed in this chapter. **Bold-face** projects rely moderately on both categories of knowledge, but fall mostly within their specified category. Phala is not the only such system that is knowledge-light, nor is it the only e-Science system in Figure 2.5 to use a recommendation-based approach, but to the best of the author's knowledge it is currently the only system of authorship assistance of any kind to have both of these qualities.

3.2.1 Other Relevant Work in Case-Based Workflow Assistance

The goal of this dissertation is not only to explore a solution to the problem of assisting authors of scientific workflows, but also to use this example to provide an existential proof that case-based knowledge-light recommendation is a tenable solution for assistance for some synthesis tasks. Thus, related work approaching other synthesis tasks will be evaluated alongside the previously mentioned e-Science work. This sub-section will examine such projects.

The first is a mixed-initiative case-based planner called MI-CBP developed by Veloso et al. This project was developed in response to perceived short-comings of a user-driven planner for deployment of military troops called ForMAT. Veloso et al. identified that novice users of ForMAT were not able to produce expert-quality plans with the support of ForMAT alone. MI-CBP combines ForMAT with the Prodigy planner in a mixed-initiative environment that keeps the user in the loop and in control of adaptations made by Prodigy to plans retrieved by ForMAT.

Madhusudan and Zhao have investigated case-based support for workflow modeling in the business process domain [66]. Their system, CODAW, utilizes workflow templates of varying degrees of generality as well as concrete cases of previously defined workflows. Their work includes various indexing techniques and graph-based case retrieval. Their system also supports workflow composition through generative planning.

Beyond the domain of both e-Science and authorship assistance, Minor et al. have developed a system for case-based agile workflow support in the business process domain [77, 76]. Rather than supporting authorship of new workflows, their system supports updating workflows to work within anomalous or changing execution requirements and circumstances. In their system, cases represent a workflow revision as a pair of two workflows: one representing the workflow before the revision and the other representing the revised workflow. Along with contextual information and domain knowledge, these cases are retrieved in order to determine how a similar workflow can be altered.

Predating Minor et al's work is CBRFlow: a project developed by Weber et al. that also supports modification of workflows (rather than authorship from scratch) within the business process domain [118]. CBRFlow is a hybrid CBR system that relies on rule-based modification of workflows to meet execution-time conditions and exceptions and incorporates a case-based component used only when the rule-based component fails. Cases are developed by users through annotation of modifications made to workflows to adapt to anomalous and exceptional events during execution.

Another project, Ghostwriter, seeks to assist authors of product reviews by making recommendations for their content [116]. Ghostwriter works with Gestalt cases, in that the cases used are complete reviews. Features are manually extracted from cases for Ghostwriter to use at the time they are incorporated with the case-base. These features are global features representing content within the review and are used to determine what additional content a partially completed feature may require. The main differences between this process and Phala's is that Ghostwriter's recommendations lack context and are only applicable for altering global features of the case, rather than the local features that Phala focuses on.

3.3 Gestalt Cases

Case-based reasoning systems are queried with a problem description in order to generate a solution. Early CBR researchers identified three major components of a case: a problem description, a solution, and an outcome [57]. A case's problem description is matched with that of the query in order to determine the relevance of the case for problem solving. The outcome of the case can then be analyzed to determine how to adapt the case's solution to the query problem.

When seeking to assist a synthesis task, the lines delineating a problem description, a solution, and an outcome are not as well defined. For example, in the domain of workflow development, the problem can be seen as a specification of the workflow to be developed, and the solution would be the workflow itself. However, this representation does not fit the requirements of the mixed-initiative model set forth for author assistance in Chapter 2. Authors seeking support throughout the authorship process are faced with many smaller-scale sub-problems. It is these smaller problems that tools like Phala seek to provide assistance for, and thus the representation of a case must shift to accommodate CBR processes that address these tasks.

Another relevant approach to representing cases for this task comes from Minor et al's work on agile workflows [77]. In this work, the problem description for a case consisted of a complete workflow, and the solution was a modification to be made to that workflow. These cases are retrieved for queries consisting of a similar workflow and used to adapt the workflow to changing runtime circumstances. This approach can fit the composite case representation model of problem descriptions and solutions by having partially completed workflows as problem descriptions and edit recommendations as solutions. Such cases would represent the author's perspective of one of the many sub-problems s/he is faced with throughout the authorship process. However, when the case modeling task is broadened beyond Minor et al.'s domain of completed workflows to the domain of editing partially completed workflows, these cases may become too specific to be useful. There are an exponential number of intermediate states of an incomplete structure, relative to its size, and a case-base would have to be substantial to provide cases that are similar enough to any such intermediate structure to provide an adaptable solution useful for problem solving. Alternatively, if a smaller case-base is used with such a representation, a heavier burden will be placed on adaptation, inhibiting the system's effort to achieve knowledge-light assistance.

Authors of scientific workflows naturally seek prior works from which to gain inspiration or to re-purpose, emulating a case-based process. The myExperiment website was produced to enable such behavior [38]. In this sense, the prior works are the cases. In fact, several case-based systems built for this and other domains employ this strategy. One example is the ForMAT system in the military deployment planning domain [79]. Additionally the growing research area of trace-based reasoning involves the use of traces of user activity as a datasource for CBR or related processes. Traces often form the basis for case-mining, just as provenance is used by Phala.

An important distinction of this perspective on case representation is that there is no longer a clear separation of case components, nor do cases reflect a specific problem description. Rather, as artifacts of a complete authorship episode, they represent the problem descriptions, solutions, and outcomes of many of the smaller problems faced during the authorship process. Such cases will be refereed to as *gestalt cases*.

Since gestalt cases do not contain neatly delineated problem descriptions and solutions, these must be *extracted* from the cases at the time the case is used. This is similar to the process a workflow author goes through when scanning workflows on myExperiment for portions to re-purpose. A case-based reasoner, such as Phala, can complete this task automatically for the user.

A similar approach is that taken by the CELIA framework developed by Redmond [92]. This framework assists in learning problem solving processes by predicting single steps in the process. These steps are extracted from "case snippets", which embody a more precise problem-solving context than the overall case does, in effect making these gestalt cases.

Assessing problems for recommendation within a query in a synthesis task mirrors the task of locating solutions within a gestalt case. These subtasks can be performed simultaneously once the query is mapped on to the case during recommendation extraction. This relationship is illustrated in Figure 3.3. The extraction process aligns complex queries with gestalt cases to identify potential problems and solutions. This process is described in detail later in this chapter in section 3.5.

3.3.1 Mining Gestalt Cases from Provenance

Using gestalt cases carries another benefit: the relative ease in which they can be mined. Gestalt cases are raw, unprocessed accounts of a cases that postpone solution extraction to the reuse phase. In the scientific workflow domain, gestalt cases occur naturally as provenance records, which record the derivation of data products gener-

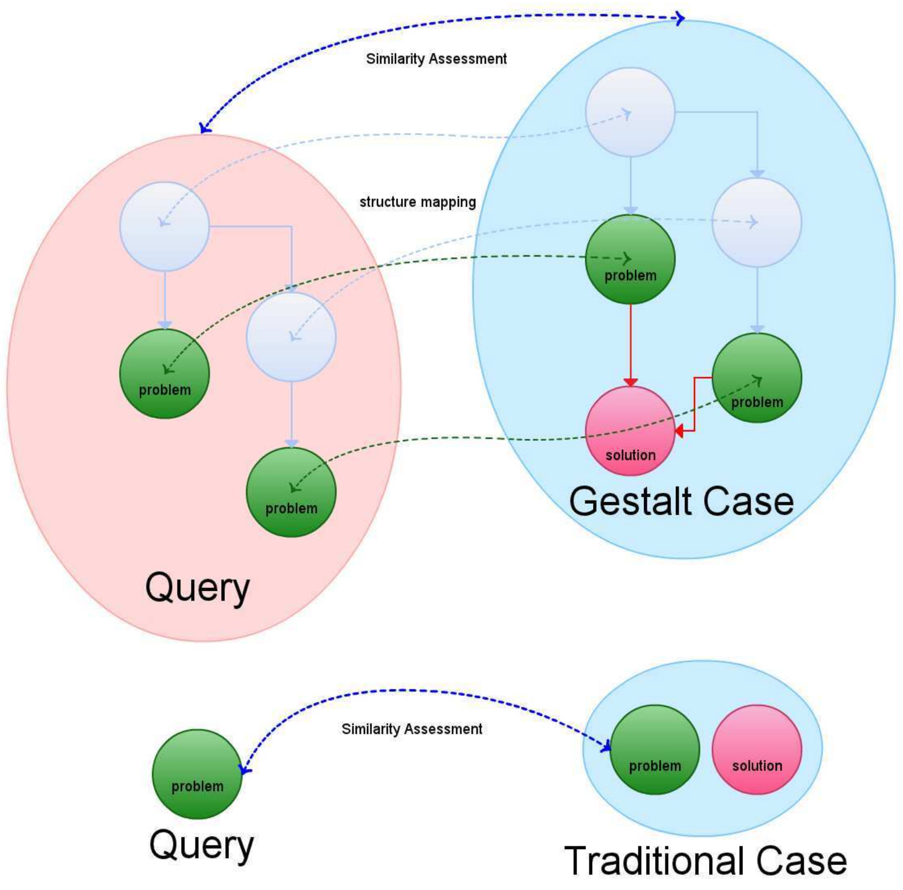


Figure 3.1: A comparison of similarity assessment with traditional and gestalt cases

ated from the execution of workflows and can be easily mined in a knowledge-poor environment to form a versatile case-base for user assistance.

3.4 Recommendations for Workflow Edits

Recommendation in synthesis tasks has been discussed in an abstract sense in Chapter 2. In this section, practical aspects of recommendation will be thoroughly examined in the context of assisting authors of scientific workflows. Since Phala only makes recommendations that affect the production, consumption, and flow of data within a workflow, recommendations for adding control structures will not be discussed.

3.4.1 Use Cases

There may be many different reasons why an author may seek assistance when developing a structure. In this subsection, two main categories will be examined for workflow authors using Phala.

3.4.1.1 Extension

Workflow authors seek assistance for quickly finding services for inclusion in partially authored workflows. The author may have a service in mind and may simply be looking for the name or a means of connecting the service, or the author may know generally what type of service s/he is looking for but may wish to see options or versions that have been used in similar situations. In these cases, Phala can offer service addition recommendations that allow the user to confirm the proper selection of the service they are looking for.

A similar scenario in which a user may seek a service addition recommendation is when it is known that some conversion must take place between data output by one service and consumed by another. In such situations, *shim services* are sought, which convert between the expected data formats [6]. These services are frequently needed by scientific workflow authors who may not know which service is necessary for the process. Phala's assistance can be particularly effective in serving these needs.

3.4.1.2 Refinement

Many workflow assistants mentioned in Chapter 2 assist users by offering recommendations for repairing what they see as errors within the workflow, particularly incorrectly typed data consumed by a service. Phala can also intervene in a similar manner by issuing any of the recommendation types other than service addition recommendations. Phala can also recommend service replacements in cases where another service may be a better fit for the workflow than the service chosen by the author.

3.4.2 Types of Recommendations

Phala offers several types of recommendations, each of which address unique categories of problems that arise throughout the processes of authoring a workflow. This sub-section provides a precise definition of each of these recommendation categories.

3.4.2.1 Workflow Preliminaries

A workflow specifies the coordination of several sub-processes (called processors) that work in concert to achieve an overall goal. In the e-Science context, workflows represent experiments, the processors of which consist of web services, local scripts, or other computational resources that consume and produce data products. These data products are traced through edges in the workflow, which denote the transfer of a data-product from one processor to another. Workflows also denote constraints on the order involved in invoking processors and additionally provide a means for higher-level control, including conditional execution and looping.

Figure 4.1 depicts a SAI representation of a workflow from the myExperiment website [38]. This workflow is used to report gene alignment information, retrieved from the Ensembl gene database, for three different species.[1] Each of the pink rectangles in the workflow represent processors, and the yellow and black edges between them represent the flow of data between the processors.

3.4.2.2 Service Addition Recommendations

This is a recommendation to add an additional processor to the workflow with a specified service. The problem description portion of these recommendations consists of a single existing processor within the query workflow with an optional output port also specified. The solution component consists of the service for the new processor and optionally an input port. The converse of such recommendations will also be possible, where the problem description may include an input port and the solution may include an output port.

3.4.2.3 Edge Addition Recommendations

These recommendations connect two existing processors within the query workflow. Delineating the problem description within these recommendations is not straightforward. It could encompass both of the processors to be joined and potentially also the associated input and output ports; however, this would leave nothing left for the solution portion of the recommendation. It should not be unnatural or awkward to consider a recommendation without a solution; indeed, some problems have only one

[1]This workflow can be viewed from the myExperiment website at the following URL: http://www.myexperiment.org/workflows/158.html

solution which can be directly inferred from the problem description. For example, indicating to someone that their shoes are untied carries the direct implication that they should tie them. However, it is important that the problem description for these recommendations matches that of other recommendations that may be alternative solutions to the same problem in the workflow. For instance, some workflows contain multiple instances of a particular service. If a problem with the workflow is defined such that there is an output of a processor (processor A) that should be consumed by some other processor, and another processor (processor B), which consumes the type of data processor A produces, already exists in the query workflow, it may not be clear whether to recommend adding a new processor duplicating the service in processor B or to simply recommend adding an edge between the two existing services. For this reason, the problem description will consist of only one of the two processors and an input or output port, while the solution will contain the other processor and the opposite type of port to that of the problem description.

3.4.2.4 Service Deletion Recommendations

If the presence of a particular service represents an error, it may not be clear whether it should be replaced or deleted. For reasons similar to those for edge addition recommendations, the problem description will be slightly more abstract. Problem descriptions will consist of the processor within the query workflow to be deleted. A solution, in this case, is simply the directive to delete the processor from the workflow; that is, service deletion recommendations are a more concrete example of a complete recommendation without a solution description.

3.4.2.5 Edge Deletion Recommendations

Rather than deleting an entire processor, it may be that only a single edge represents an error or another sort of problem within the workflow. In this scenario, an edge deletion recommendation may be issued. Such recommendations will have similar problem descriptions to the edge addition recommendation, but solutions may further indicate which edge is to be deleted.

3.4.2.6 Service Replacement Recommendations

If a newer or better version of a service is available for a processor it may be prudent to recommend replacing the service. However, the user may be confused when presented with a service deletion recommendation, not realizing that the recommender will eventually also recommend adding a new processor, along with several edges. In order to better communicate the goals of the system, it would be better to issue a single recommendation identifying that the only change to be made is switching services within a processor. However, it is worth noting that some edges may become invalidated after integration of such a recommendation if their ports no longer exist with the new service.

3.5 Extracting Recommendations from Provenance

As mentioned earlier in the chapter, extraction is the process through which recommendations are drawn out of a gestalt case. Given that gestalt cases have no discretely segmented problem description, such a process is necessary for applying the case to an author assistance scenario. This section explores the extraction process in detail.

3.5.1 Mapping to Discover Differences

The process of extraction relies on mapping as many components of a query to a retrieved case as possible in order to discover differences between the case and the query. Once a mapping is developed, unmapped elements of the retrieved case that are associated with mapped elements of the retrieved case are interpreted as differences and can be presented as recommendations. For instance, if node B1 is linked to node A1 in the case, and node A1 is mapped to node A2 in the query, a recommendation to create a new node similar to B1 and link it to A2 may be made.

3.5.2 Query Formation

At any point during the construction of a workflow, the current state of the workflow can be considered a query to the Phala plug-in. Phala will regularly search prior cases for relevant information and, if successful, return a collection of recommendations for changes. These queries are performed automatically every time the user makes a change to the workflow. If results are found, they are displayed as soon as they are received. No results are displayed if Phala determines that the quality of the recommendations is too low (based on a confidence threshold). The SAI system discussed in Chapter 3 assures that recommendations will be generated by Phala in a reasonable time frame, while maximizing the quality of the recommendations within the time available.

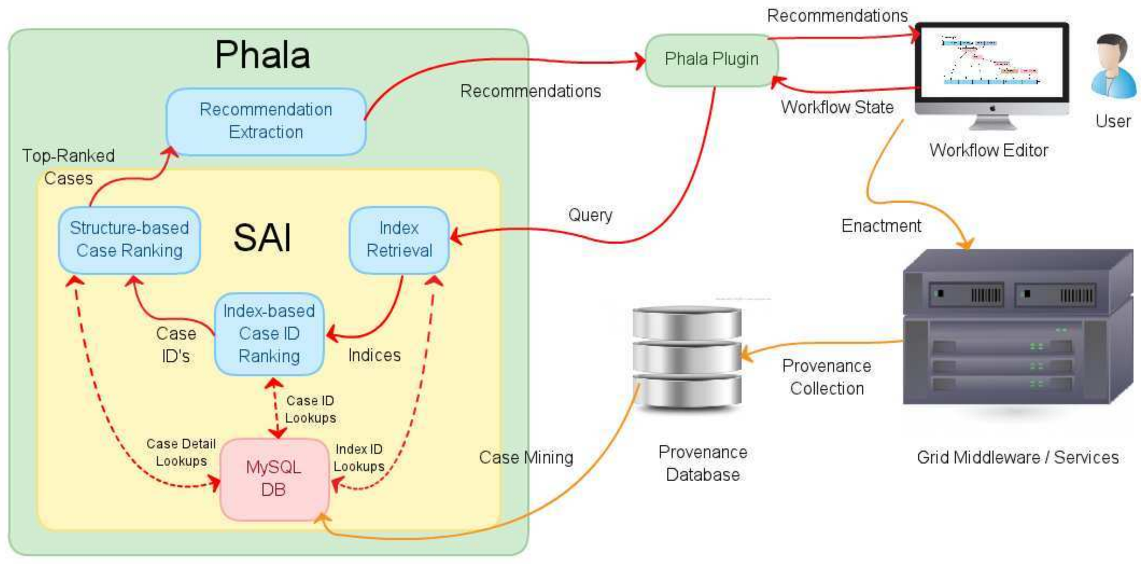


Figure 3.2: Phala's re-use cycle

3.5.3 Recapture

The final phase of a traditional case-based reasoning cycle is retention. The query to the reasoner is committed to memory, as is the solution provided by the reasoner and the outcome, which together form a complete case. After a reasoning episode using extraction from a gestalt case, the current state of the structure developed by the user may not yet be complete and thus not represent a complete case. Therefore, rather than having retention be part of the reasoning cycle for a single reasoning episode, retention must be deferred until the structure being developed by the user is complete. Retention is approached less frequently under these circumstances than other steps.

In the e-Science domain, Phala's CBR process supports the development of a workflow by a human. However, the final result of executing the workflow is a provenance trace, which will be collected from the execution of the workflow. This provenance presents another mining opportunity for developing new cases. Mining cases from workflows authored with assistance by the case-based reasoner is called *recapture*, and it replaces the retention phase in Phala's reasoning cycle. This cycle is illustrated as follows:

- The user creates a new workflow, incorporating recommendations made by Phala.
- The user executes the workflow, capturing provenance information within a provenance database.
- Once the execution of the workflow is complete, the control decisions made during the execution are reflected in the recorded provenance, resulting in a more specific case on the execution path taken than that of the original workflow.
- A new case is then captured through provenance mining for future use in creating new workflows.

This process is further illustrated in Figure 3.2. Note that the orange lines trace the steps in the recapture cycle, while the red lines outline a single reasoning episode.

3.6 The Role of Adaptation

Determining how to automate case adaptation is a classic problem for case-based reasoning. Numerous adaptation methods have been developed, requiring varying amounts of knowledge (see [67] for an overview). The use of *knowledge light* approaches (e.g., [120]) facilitates the development of adaptation systems but may not capture important aspects of the domain; knowledge-rich approaches are powerful, but the needed knowledge may be expensive to encode and may have gaps if the knowledge engineer fails to correctly anticipate future adaptation problems.

Hanney et al. classified many CBR systems by the type of adaptations performed, including whether a system performed any adaptation at all [41]. They discovered that adaptation was particularly important in CBR systems involved in a design task. Indeed, adaptation is important in Phala's case-based assistance framework, since otherwise, Phala will slowly transform the query into the most similar case. If this were desired, then Phala's interactive recommendation-based assistance would be unnecessary, and a simple similarity-based case retrieval would be sufficient to solve the user's problem. In fact, authors would be using Phala to create novel workflows, which may incorporate some elements of Phala's stored cases but would be better served by a concerted effort to adapt recommendations extracted from similar cases to the workflow being developed.

3.6.1 Adaptation in Inexact CBR

Recommendation systems issue recommendations instead of commands to the user because the system is deferring to the user's better judgement. Implicitly, the system concedes the possibility that its recommendations may either be incorrect or not what the user optimally preferred. Because of this, it is impossible for a case-based recommender to guarantee that its adaptation procedure generates perfect recommendations. In this case, adaptation takes on a slightly different role: improving recommendations rather than perfecting them. However, adaptation is still very important in inexact CBR for the same reasons it is important in exact CBR. In both exact and inexact CBR, adaptation is required to modify semi-related solutions to meet the specific requirements of the query problem, given that the query is not an exact match with the problem description of the retrieved case.

3.6.2 Designing a Framework for Developing Hybrid Recommenders for Synthesis Tasks

When trying to solve a problem, it is a common practice to seek the advice of others as to what decision to make next. One may seek advice from a single trusted colleague, but commonly one seeks advice from multiple sources, weighing the advice from each before making a decision [91]. This behavior is mimicked by ensemble and hybrid methods for machine learning which combine outputs from multiple independent components to arrive at conclusions that can be more accurate and reliable than those attained from any one member of the ensemble. This section briefly introduces the concepts behind hybrid and ensemble systems and explains how hybrids can be used for adaptation purposes in synthesis tasks.

3.7 Ensemble Techniques for Recommender Systems

Ensemble methods work well with analytical recommendation, which can be viewed as a classification task. Recommendations in an analytical task are flat values that are easily compared and combined. Ensemble classifiers commonly employ voting methods to negotiate a final classification from among those generated by each constituent recommender [70, 27]. For instance, if two constituent recommenders recommend the user purchase "Moby Dick", and a third recommends "Gone with the Wind", the ensemble will recommend "Moby Dick". Depending on the voting methodology applied, constituent recommenders may weight their votes using a confidence estimate.

Classification systems have been studied extensively through ensemble methods, and though efforts have also been made to study clustering and regression ensembles, some have suggested wider application of ensemble methods [97]. Combining recommendations for synthesis tasks complicates the task of combining recommendations, since doing so involves negotiating the two common recommendation components simultaneously among constituent recommenders. Methods for performing such combinations weighted by confidence estimates are discussed later in this chapter.

3.7.1 Hybrids vs Ensembles

Most studies of ensemble methods have focused on varying the training set, feature set, or random values, initializing a single inducer to produce a diverse set of classifiers. An example would be to create a neural network ensemble by varying the initial random weights before training. Even though there is no change in the inductive bias for each component of the ensemble, ensemble methods can increase the accuracy or diversify the possible decision boundaries learnable by a particular technique.

This section elaborates on a different sort of ensemble technique, what is sometimes called hybridization, in order to support synthesis tasks. The principle difference between a hybrid approach and an ensemble approach is that an ensemble is composed of several instances of the same technique, each of which vary only in settings, training set used, or other parameters [97]. In contrast, hybrid systems may employ vastly different reasoning techniques, such as case-based reasoning and neural networks. Techniques used for ensembling are relevant for hybrid frameworks since the task remains mostly the same: combine results from several constituents and present them to the user.

Synthesis tasks are complex, often involving a disparate collection of goals and potential problems. Such tasks may also involve several sources of knowledge, such as user-generated content (tags, ratings, etc.), prior completed structures, prior interaction episodes, expert knowledge, etc. One could produce a complex system aimed at encompassing strategies addressing every conceivable aspect of the task, but there is value in involving multiple techniques, which may either specialize in incorporating knowledge, such as a case-based reasoner whose cases record episodic knowledge, or specialize in addressing a particular domain-specific issue, such as a rule-based reasoner. An ensemble framework simplifies the design of such a hybrid system and allows for easy ablation studies and analyses of refinements to components. Additionally, ensemble systems can learn weights for individual components, normalizing the support offered from potentially unrelated rules or techniques.

3.7.2 CBR Hybridization and Adaptation

Any CBR system using adaptation can be thought of as a hybrid system. Adaptation is performed as an important component of the reuse process by adapting the solution retrieved from the case-base to fit the specific problem constraints of the query. This process is carried out by an independent component of the system through various means potentially involving application of engineered domain knowledge or mining the case-base.

Given the potential trade-offs of different adaptation methods, an interesting alternative to using a single adaptation method for a CBR system is to instead endow the system with a range of methods—potentially including a standard mix of knowledge-light methods with any available knowledge-rich methods—for the system to automatically assess the set of recommendations and exploit whichever it expects to be most useful, given the task context and the relationships between the current recommendations. Not only is the CBR system hybridized by the inclusion of an adaptation procedure, but through this perspective the adaptation procedure itself is also a hybrid.

Combining multiple adaptation methods relates to previous work on learning multiple local adaptation methods [88], but the approach presented here contrasts by considering multiple adaptation methods for use in each problem. The approach is in the spirit of the ensemble techniques that have been applied to CBR (e.g., [90]), but it emphasizes combining different types of methods that may not individually be capable of generating solutions for the entire space. This enables the system's starting set of adaptation methods to be chosen not for their global characteristics, but for the degree to which the methods complement each other and fill specific knowledge gaps in the system.

3.7.3 Highly Procedural Adaptation Rules

Adaptation often involves the application of rules that address procedural knowledge about the task being performed, which is not captured in the cases themselves. Procedural knowledge complements factual knowledge [14], the latter of which is provided in Phala by gestalt cases. Procedural knowledge is an important part of interpreting and reusing the knowledge stored in gestalt cases. For instance, in the workflow domain, the fact that an unused input port is more indicative of an error than an unused output port for a processor is not apparent from the analysis of a single case. This procedural knowledge must be encoded and included within the adaptation phase to more properly judge how the recommendations extracted from the case should be altered before being presented to the user.

In Phala, procedural knowledge is encoded in what are called Highly Procedural Adaptation Rules (HPAR's). These rules are actually programmatic components of the hybrid adaptation framework for Phala. The advantage of using these components within a hybrid framework, rather than hard-coding a means of applying these rules, is three-fold. First, the body of research regarding hybridization can be leveraged to ensure each HPAR is contributing optimally to the adaptation process. Second, encapsulating each rule makes it easy to perform ablation studies to determine the value (or lack thereof) of employing the rule in the system. Lastly, HPAR's can be written with varying degrees of domain-dependence. For instance, the rule regarding workflow inputs that was mentioned earlier may apply equally to various workflow domains and thus can be re-used by other systems employing the same framework. The HPAR's associated with Phala are explained in Chapter 5.

HPAR's are used within Phala to support the goal of achieving knowledge-light adaptation. The rules used by Phala are relatively domain independent, and can be applied to many workflow domains. However, the hybrid framework also affords inclusion of knowledge-rich recommender components, which may complement the strengths of the knowledge-light components by providing more support in situations for which no relevant prior case exists, and increasing Phala's performance within the constraints of a more limited domain. In this sense, by using the hybrid framework, Phala allows engineering and inclusion of ontological knowledge which can support, rather than replace, its existing knowledge base to increase the value of the recommendations it produces.

3.7.4 Framework Requirements

This subsection details requirements of a framework seeking to facilitate hybridization. These requirements identify unique issues to this task that may not be present in single-component systems or in ensemble systems. Each of these requirements are additionally addressed in the following section describing the implementation of the Hybrid Recommendation Framework.

3.7.4.1 Combining Recommendations

Recommendations that partially or fully overlap may be combined to produce consistent and less confusing advice for the user. Likewise, if identical recommendations are generated from multiple recommenders, the overall confidence that the system has in those recommendations should reflect the individual confidence values, requiring combination methods. The combination problem also presents requirements such as a method for dealing with the confidence of mutually exclusive recommendations and how contra-recommendations (discussed later in this section) should be combined with other recommendations.

3.7.4.2 Facilitation of Specialization

In Chapter 2 we identified two separate tasks for a recommender for a synthesis task to address: recommending problems and solutions. Components of the framework can specialize in one or the other. For instance, analyzing the frequency of re-used words can identify some potential word-replacement problems, while a thesaurus can identify potential word-replacement solutions. The framework must provide a clear separation of these tasks and facilitate communication between such components.

If recommendations are to be combined, it is not necessary that every recommendation be complete. Some recommendations with abstract problem definitions may lend support to recommendations with answerable problems. These recommendations are called problem recommendations and need not carry any solution at all. They may not necessarily be applicable for the user, but they may help a hybrid system to develop a more accurately ordered set of recommendations.

3.7.4.3 Confidence Estimates

Components tasked with identifying problems can err both in falsely identifying a non-problem (false-positive), or by failing to identify an actual, existing problem (false-negative). Adding additional components will reduce the number of false-negatives (increase coverage) but will also increase the number of false-positives, potentially at a greater rate. Adding components to the ensemble should improve performance rather than degrade it, therefore an additional requirement is considered: recommendations must include a confidence estimate. By ranking recommendations according to confidence estimate, the user can selectively view only the most viable problems and avoid scanning through high numbers of false-positives. The framework is then also responsible for regression over confidence estimates generated by each component to insure comparability and proper weighting.

3.7.4.4 Normalization

There is no universal unit signifying degrees of confidence. Thus, confidence estimates produced by independent components will not necessarily be aligned with one another. This poses problems for combining recommendations since the combined confidence may be skewed by the different interpretations of the values held by each component.

One effort to address this problem is to first ensure there is a maximum and minimum confidence estimate, which allows confidence values to be normalized to the [0, 1] interval. This is an important step to developing translatable confidence estimates but does not yet account for the fact that interpretations of confidence may be non-linear. Confidence estimates must be evaluated alongside the measurable results they estimate, such as accuracy or relevance. Once a correlation can be created between estimates and actual judgements, regression can be performed to produce a more translatable confidence estimate useful for comparison during the recommendation combination process.

3.7.4.5 Contra-recommendation

Another requirement reducing false-positives is allowing components to recommend *against* a problem or solution. Allowing *contra-recommendation* increases the range of strategies that can be incorporated into an ensemble, provides for further decomposition of complex strategies, and increases the benefit of confidence thresholding by reducing the confidence in likely erroneous recommendations. One important concern is the complication this adds to combining confidence estimates, though we have begun to address this issue [61].

3.8 The Hybrid Recommendation Framework (HRF) Components

The Hybrid Recommendation Framework was developed to address each of these requirements in a domain-independent manner. HRF provides the necessary tools for implementing a hybrid recommendation system and supports the complexity of synthesis tasks by combining recommendations of both problems and solutions.

We note that a similar framework was proposed in the context of information extraction by Frietag, who developed a scheme for combining multiple specialized classifiers in the similarly complex task of text mining [31]. Frietag transforms confidence values into correctness probabilities and uses these scores in one of three combination schemes (two of which are the basis for schemes described later in this section). However, in addition to bringing such an approach to the adaptation task for CBR, HRF differs in that it focuses on the issue of recommendation overlap and how this should be handled by general combination methods, in particular, through the new contributions of recommendation review prior to combination and the consideration of contra-recommendations, as well as the two new combination strategies described in the next section.

3.8.1 Recommendations

Recommendations in HRF are viewed as composite, consisting of recommendations of a problem and a solution. Each of these components are addressed independently but are grouped together in the code for HRF for organizational purposes, since the user will require both simultaneously in order to act on a recommendation.

The development of an ensemble framework for this thesis work is motivated by early work performed towards building an intelligent assistant for workflow authorship by combining four independent components [61]. Through this study, it was discovered that the task of identifying and solving problems in a partially-completed workflow can be somewhat cumbersome, and components could be further specialized if this task were decomposed into two independent tasks: recommending problems and solutions. This distinction was adopted for the HRF framework in order to address the requirement for facilitation of specialization, identified in the previous section.

3.8.2 Recommenders

Within the framework, a recommender is a component that is capable of producing or critiquing recommendations. Critiques are performed by generating additional recommendations, which either support a problem or solution by repeating it with a positive confidence or reject a problem or solution by repeating it with a negative confidence value. Not every recommender is responsible for both of these tasks; some will generate the initial recommendations considered by the system, and others will wait until some recommendations have been generated to add critical recommendations to the collection.

3.8.3 Combiners

Combiners are components that reduce redundancy in the recommendations produced by recommenders by combining confidence estimates and eliminating all but one recommendation within a group of recommendations with identical problems and solutions. Combiners in HRF implement a weighted voting scheme between the constituent recommenders to produce a final set of recommendations, which can be presented to the user in order of overall confidence.

If two independent recommenders generate the same recommendation and each has high confidence in that recommendation, the system should consider those recommendations to be independent confirmations and increase their overall confidence. However, if both recommenders base their recommendations on overlapping knowledge, the effect of the repeated recommendations on confidence is not as clear. Furthermore, two recommenders may identify the same problem but different solutions. In such cases, the system should increase its confidence that the problem is relevant. Next are a series of descriptions of combination strategies which address these issues, including how to incorporate contra-recommendations.

3.8.3.1 The MAXIMUM Combiner

The recommendation with the highest confidence value is chosen. If this is a contra-recommendation, all recommendations with matching solutions are removed from the set and the process is repeated. If no recommendations remain, none are returned. This is based on the CMax method used by Freitag [31].

3.8.3.2 The AVERAGE Combiner

This method first partitions recommendations by shared solution. The confidence values for each of these partitions are averaged (confidence scores for contra-recommendations are negative). If the average is negative, the recommendation is switched to a contra-recommendation. The resulting recommendations are then passed through the Maximum combination method.

3.8.3.3 The INDEPENDENCE Combiner

This method also first partitions recommendations by shared solution. Under the assumption that each confidence value is independent and estimates the probability of that recommendation being correct, the overall confidence that the solution is correct is determined by calculating the probability that all matching positive recommendations are not wrong and that all matching contra-recommendations are wrong. For each partition, one solution is returned with this recalculated probability score. These results are then passed through the Maximum combination method. This is derived from the CProb method used by Freitag [31].

Note that a problem with the INDEPENDENCE combination method is that confidence values are not always independent. Although, ideally, recommenders will complement each other, in practice, some knowledge maybe shared by the recommenders.

3.8.4 Recommendation Process

Many independent recommenders can be integrated with HRF. The recommendation process begins when each recommender is issued the query. Recommenders may or may not produce recommendations in response to the query. Once the first round of recommendations is complete, the generated recommendations are re-presented to each recommender, and any new recommendations are collected. This process is repeated until a round in which no recommenders produce a recommendation. At that point, the entire collection of recommendations is forwarded to the selected combiner, which produces a set of recommendations that are delivered to the presenter of the query.

Phala employs HRF to adapt and improve its recommendations through use of multiple recommenders built to address specific gaps in Phala's reasoning. An evaluation of the impact of HRF in Phala is presented at the end of Chapter 5.

Chapter 4

Comparison and Indexing of Structured Cases

4.1 Introduction

Developing a *structure-aware* reasoning system requires attention to a variety of circumstances that do not arise when dealing with flat, unstructured data. The previous chapter identified how structured data complicates the recommendation task by requiring richer recommendation representations. This chapter explores another angle: how to represent, store, and retrieve structured cases.

Essential to retrieval of structures similar to a query structure is also the ability to efficiently compare structures. This issue alone could be the topic of several dissertations (and, in fact, is: e.g. [108], [16]) and is indeed a topic receiving a growing amount of attention. As such, this chapter will not cover the topic exhaustively but will focus on qualities of structured data important to case-based reasoning applications. This implies a focus on design decisions within an approximate-matching retrieval system, in contrast to more popular exact-match retrieval systems.

This chapter provides three main contributions. First, it will identify the role of structured data in case-based reasoning by analyzing desiderata of a structured case storage and retrieval system. Next, it will identify how existing work in graph storage and retrieval relate to structure-aware CBR. This chapter will also introduce The Structure Access Interface (SAI): a software tool for representing, storing, and retrieving structured data.

4.2 Desiderata for Structured Data in Case-Based Reasoning Systems

Structured cases are not a new phenomenon in case-based reasoning. Many CBR projects have used structured case representations, some employing structure-aware similarity assessment, others using the structural aspect solely to assist in problem solving. Since one aim of this dissertation work is to produce a software tool that can assist with representation, storage, and retrieval of structured cases in a variety of domains, we must first examine what the desiderata for such a tool would be. This section identifies these desiderata by examining CBR projects utilizing structured cases and their representational needs. Throughout this section, the term *global* will refer to properties that apply to a case as a whole, whereas *local* will refer to case properties that are more limited in scope. For example, consider a graph representation of a molecular structure. The name of the molecule is a global feature, whereas the type of bond between two carbon atoms is a local feature.

4.2.1 Case Representation

Structured cases possess information that is not global in the scope of the case; rather, it can be constrained to individual components of the case. Graphs are a very common and powerful formalization frequently used to provide structure within data, particularly labeled graphs, which combine both data and structure. In this case, nodes are labeled with the information constrained to a particular component (an object or a concept). This information may be more complex than a simple classification. For example, ShapeCBR is a system that assists in the design of metal castings [74]. It represents casting components as values of which there are eight possible classifications. However, other details exist for individual instances of these components, such as scale. Lately, the cooking domain has been popular with CBR researchers, wherein recipes comprise cases [106]. In these cases, nodes may represent cooking tasks such as baking, which may be further parameterized by details such as temperature and duration.

Edges represent relationships between these entities. If only one type of relationship is modeled, edge labels are not necessary. However, this is often not the case. For instance, an unbounded number of relationship types must be modeled in the Concept Map CBR domain [18]. ShapeCBR is another example. Edges in ShapeCBR represent physical linkage between casting components. These edges are not all equivalent; many components do not have an axis of symmetry between each joining location. Furthermore, different kinds of coupling objects are used to link larger components.

This information must also be captured by the edge's labels.

Structured cases also have global attributes, similar to traditional feature-vectors. In this sense they are a generalization of feature-vectors, but more importantly, the global attributes provide a means to store unstructured data. For instance, in workflow domains, a number of global features exist, including semantic tags, contextual features such as author data, and data describing the structured components, such as the workflow language used.

4.2.2 Similarity Assessment Desiderata

A key issue for retrieval and similarity assessment of structured data is that graph processing algorithms can be computationally expensive (for instance, subgraph isomorphism is NP-Complete [22]). A general retrieval tool must address the inherent complexity in these domains in order to handle similarity queries but it does not impose any single algorithm for performing a task that involves a potential trade-off between computational desiderata, expressiveness, and domain dependence. Numerous methods have been developed to address these computational constraints, such as greedy algorithms, anytime algorithms, and heuristic search. For instance, Bergmann and Gil have adopted a knowledge intensive A* approach to mapping structured cases to queries in a workflow processing domain [12]. Reichherzer and Leake have used topological features of concept maps to weight the significance of local features [93]. Minor et al. have addressed the complexity in the workflow domain by converting graph structures to flat strings for comparison [75]. Additionally, Genter and Forbus recognized the need for structural similarity retrieval systems to use a two-phase retrieval algorithm (what they call MAC/FAC, which stands for “Many are called but few are chosen”), which first retrieves a large and imprecise collection of structures through a computationally inexpensive analysis of their association with relevant indices and then refines the collection using a more expensive but more accurate structure-mapping approach [33]. This idea of retrieving structures in two phases was adopted at an early stage by CBR researchers working with structured data [81, 17].

4.2.3 Indexing and Retrieval Desiderata

Indexing is important for efficient retrieval in large case-bases. Indices can be used to search for the most important and relevant cases without requiring a similarity assessment for each case in the case-base, allowing for sub-linear retrieval complexity. Indexing is related to similarity assessment in that index selection should yield cases likely to be assessed as more similar. The representational power of a system’s indices bears directly on how accurately index-based retrieval can estimate the outcome of a full similarity assessment. This section takes a closer look at the desiderata of an indexing representation by examining representations used in existing index-based graph retrieval systems.

Indices should capture global case properties in order to allow for retrieval methods, such as conversational retrieval, retrieval by semantic tag, context-sensitive retrieval, and others. For example, Weber et al. retrieve related workflow instances through an interactive dialogue with the user [117], answering questions associated with cases in the case-base and retrieving cases related to the user’s responses (also known as conversational case-based reasoning [2]). Also, Allampalli-Nagaraj et al. use semantic metadata tags to improve retrieval of medical imaging cases in their CBR application [4].

Global case properties are also often referred to as surface features. This is in contrast to deep features, which must be derived from the case data through potentially computationally expensive means [81]. In order to index cases by deep features, indices must be able to contain feature types and values that are not present in the cases they index.

Structural features can be seen as surface features, since they are readily apparent in the case data, but they are more aptly deep features, since they are chosen from a very large feature-space. Since, as has been mentioned, similarity assessment often relies on structural comparison, indices should also capture structural information. In order to satisfy this constraint, Yan et al. identify that even simple path indices (such as those used in the graphgrep project [37]) do not sufficiently represent the structure of a graph and that indices must represent arbitrary substructures in order to be most effective [123].

4.2.4 Summary of Desiderata

In summary, an ideal general retrieval tool must provide for a graph representation, allowing annotation of nodes, edges, and graphs as a whole. Annotations must be more rich than simple singular classifications, such as a set of key-value pairs. Also, annotations must allow for multiple values for a given key.

Such a retrieval tool must also provide an abstract means of similarity assessment while maintaining the common goal of performing similarity assessment quickly enough to support real-time user assistance scenarios. In order to maintain flexibility and support multiple applications of the same case-base, the abstraction must allow for selective consideration of individual structural features, allowing other features to be ignored in select contexts. It should also be adaptable to many domains by incorporating common abstract techniques, such as search.

The tool should allow for a multi-phase retrieval process utilizing case indexing. Indices should be able to approximately represent the structural details that cases hold. Indices should not be constrained to using the same data-types as used in the indexed structures to allow for indexing of deep features. Indexing should be abstract within the tool to allow for domain-specific implementations which balance the costs of index selection and retrieval (during queries), generation and mining (during case storage), maintenance (during system idle time), and the expected correspondence to similarity assessment and distinguishing power (the degree to which they limit the number of cases to be retrieved) of indices.

4.3 Existing Work in Graph Storage and Retrieval

The SAI framework was developed through this dissertation to meet the requirements identified in the previous section. The development of SAI was initially motivated by the structured retrieval needs of process-oriented case-based reasoning, but it belongs to the broad area of graph database software. This section discusses its relationship to such software, focusing especially on similarity-based retrieval, to delineate SAI's purpose and contributions with respect to such systems and to identify the extent to which existing work has met the requirements laid out in the previous section.

4.3.1 Graph Database Software and the NoSQL Movement

In recent years, relational databases have seen their dominance reduced in the arena of *structured storage*. Some database efforts, referred to as *NoSQL* databases [107], have rejected the relational model and offer storage access without an SQL interface. By offering fast access to storage of structured data, SAI joins a subgroup of these projects known as *graph databases*.

Graph databases have recently gained popularity primarily for working with emerging linked data sources, such as social networks, biological data, and the world-wide web. These data sources involve very large graphs, spawning projects that focus on retrieving one graph component at a time (typically a node and its incident edges) for traversal algorithms (e.g., Infogrid [34], Neo4j, AllegroGraph, DEX [68], VertexDB, HypergraphDB [48], and InfiniteGraph [114]).

These tools, while useful for many domains, are not ideal for case-based reasoning systems since they lack structural indexing and thus cannot benefit from fast approximate similarity queries. There are, however, some systems that have this capability, which are reviewed in the next section.

4.3.2 Graph Indexing Systems

Several graph indexing systems that provide structural indexing for similarity queries exist. These include graphgrep [37], grafil [123], PIS [124], and Tree+Delta [128].

While each of these projects is a powerful tool for storing and performing similarity queries within a case-base, each prescribes an algorithm that may not be optimal for arbitrary domains. Graphgrep uses only path-based indices that are of limited utility in domains such as chemical compounds in which paths are not highly distinguishing but structures are (as noted by Yan et al. in [123]). Grafil and PIS do not allow for multiple attribute types in graphs and mine frequently occurring structures from the database which increases the time to introduce a new graph to the database and requires a more complicated index retrieval procedure for arbitrary queries, both of which can be avoided in domains where simple paths are sufficient for indexing purposes. Tree+Delta has attributes of both approaches in that it indexes graphs by mining frequently occurring trees as indices.

Each of the previously listed projects are fully implemented retrieval algorithms with accompanying evaluations that identify situations in which the algorithm performs favorably. The iGraph framework was developed to provide a common implementation in which many of these algorithms could be compared to one another in a consistent manner [40]. The iGraph creators noted in their evaluations that there is no single winner amongst the algorithms they tested, and in fact, most algorithms they tested were dominant in at least one test scenario. This reinforces the fundamental requirement for a graph retrieval system identified in this chapter – that it is able to support a variety of algorithmic approaches in order to best fit the data in the target domain.

4.3.3 Similarity-Based Retrieval in Graph Databases

All the systems mentioned until now perform a search for graphs in a database containing the query graph as a subgraph. If a graph in the database doesn't contain the query graph, it is not returned in the search results, regardless of how close it may come to containing the query graph. In case-based reasoning, we rely on the case-base to provide solutions to problems that have not been considered a priori by the reasoning system. In these cases, we require both the identification of approximate matches in the database as well as the ordering of results by their similarity to the query (or by their utility in solving the problem). Exact-match graph retrieval systems cannot assist in solving these problems.

Approximate supergraph retrieval is a different problem than exact supergraph retrieval. The definition of correctness is more flexible, since it is no longer a binary decision, allowing for increased benefit from approaches such as any-time algorithms that provide a trade-off between computation time and accuracy of results.

Some existing work addresses approximate graph matching. For example, the TALE [110] and SAPPER [127] systems provide approximate supergraph retrieval within a database of very large graphs.

Of greater interest to CBR researchers is a tool that can retrieve graphs within a very large database of small to medium sized graphs. There are also many projects that fulfill this task. The SAGA algorithm incrementally extends an initial mapping of nodes to estimate the best possible mapping of a query onto a graph in the database [109]. The C-Tree algorithm uses a novel hierarchical indexing strategy [43]. The Gstring algorithm uses domain-specific index templates to optimize its indexing strategy [49]. Another example is the G-Hash algorithm, which uses graph kernel functions instead of substructure indices [115].

The goal of SAI differs from that of such projects in that SAI is not a retrieval algorithm. Rather, it is an abstract framework for developing, comparing, and applying retrieval algorithms suitable for CBR tasks. In this sense it is more like the iGraph architecture discussed in the previous section, except that it is engineered for approximate supergraph retrieval rather than exact supergraph retrieval.

The SAGA algorithm has been packaged as part of a similarity-based retrieval system called Periscope/GQ [111]. The purpose of Periscope/GQ, like that of SAI, is to provide both a basic means for representing structured cases (in the form of attributed multi-graphs) and an infrastructure for case indexing. However, SAI differs in its focus on providing an extensible framework for the indexing and similarity assessment tasks, whereas Periscope/GQ focuses on supporting a wider range of graph sizes. Many algorithms for indexing and comparing graphs exist, and SAI is designed to allow for adoption of varying techniques to best fit the problem domain. Furthermore, by providing an abstract retrieval framework, SAI is intended for use in analysis and refinement of indexing and comparison techniques.

In addition to supporting abstract indexing/retrieval tasks, SAI supports maintenance tasks that can remove some of the computational burden of indexing from the query-time task of memorizing new cases. Another unique feature of SAI is that it allows any structure to index any case. The features present in indices do not also need to be present in the cases they index. This enables auxiliary and retroactive indexing abstractions, which may not have been considered at the time the case was committed to the case-base.

4.4 The Structure Access Interface (SAI)

SAI was developed as part of this dissertation in order to provide a graph database that supports all of the desiderata in section 4.2 for case-based reasoning systems relying on structured cases. SAI provides solutions for many common tasks for CBR developers (e.g., representation, retrieval, storage, indexing, and similarity assessment) but maintains flexibility through its customizability. SAI's representation makes it easily applicable to a variety of specialized structural domains, including workflows, provenance, state machines, planning, concept maps, CAD models, and scene graphs. SAI does not impose any one algorithm for handling retrieval in these domains. Rather, SAI provides a framework for retrieval and index maintenance, using code that may be applied as is or replaced with custom implementations. SAI also does not impose syntactic rules or semantic constraints: SAI's representation is as general as a labeled, directed multi-graph. SAI can be used to store and perform similarity-based retrieval with powerful graph-based knowledge representations such as conceptual graphs [105], but SAI does not itself provide a means for inference or error checking over these structures. SAI is intended to provide a basis for creating graph-based knowledge representations and to solve storage and retrieval needs as a component of a larger, encompassing reasoning system.

Throughout this section, examples will refer to Fig. 4.1, which depicts an SAI representation of a workflow from the myExperiment website[38]. This workflow is used to report gene alignment information retrieved from the Ensembl gene database for three different species. [1]

[1]This workflow can be viewed from the myExperiment website at the following URL:

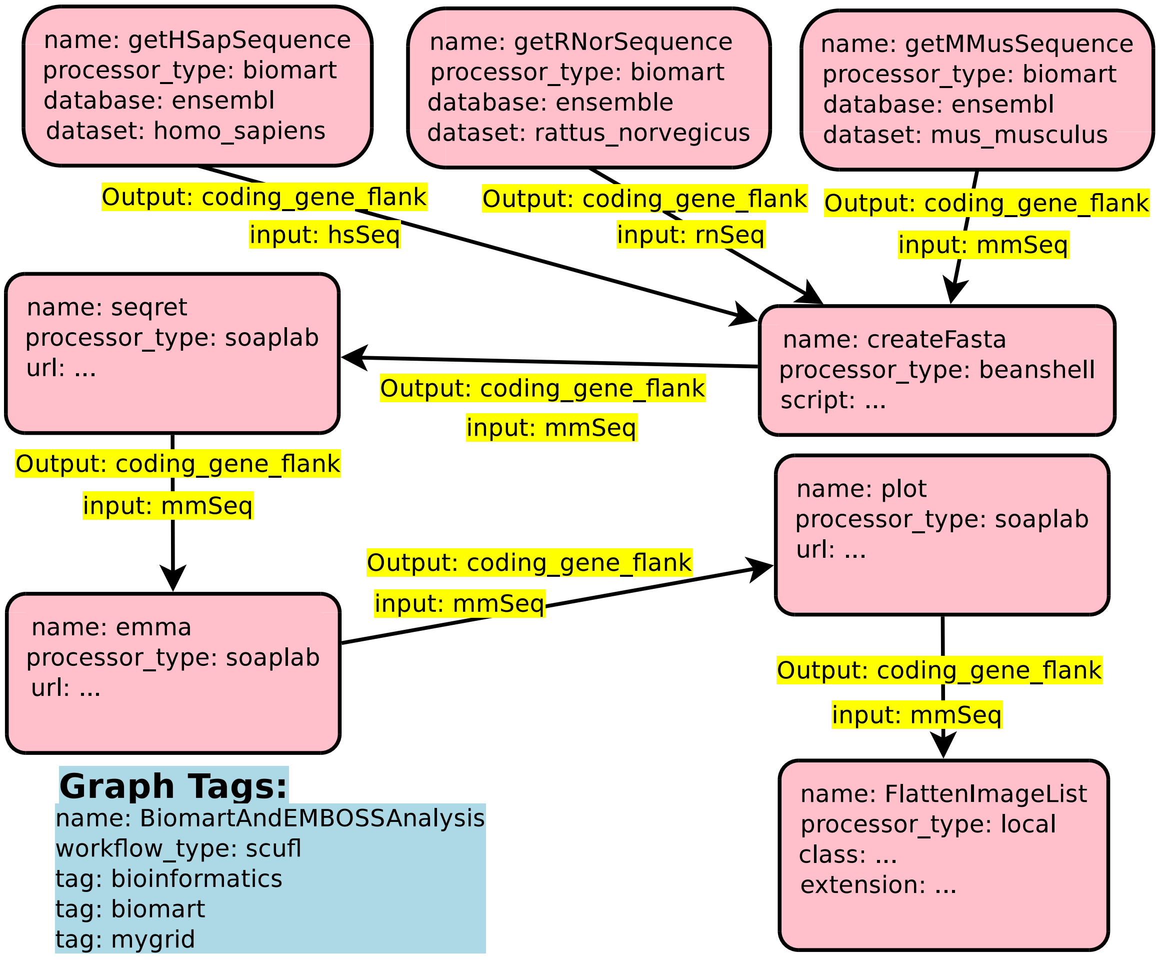


Figure 4.1: An example structure in SAI representing a bioinformatics workflow.

4.4.1 Graph Representation

SAI uses a directed, multi-attribute, multigraph representation. These terms are defined as follows:

- **Directed**: Edges are represented by ordered pairs. This order, however, need not be respected by comparison algorithms.

- **Multi-attribute**: Any graph entity (graphs, nodes, and edges) can be at-

http://www.myexperiment.org/workflows/158.html tributed with any number of *features*. A feature is an atomic value which is a combination of a string value and a class name. Examples of features are shown in Figure 4.1. In this example, the graph features of type *tag* are multi-valued.

- **Multigraph**: Multiple unique and non-unique edges can exist between the same two nodes and be seen as distinct. That is, if an identical edge is recorded twice, it will appear twice in the case representation. This accommodates information such as multiple data links of the same type in a workflow or double bonds in a molecular structure.

Note that using a multi-attribute graph does not increase the representational power of SAI beyond using a basic labeled graph. In a basic labeled graph, the same information captured by multi-attribute graphs could be captured by making labels composite, or by adding nodes which together comprise a set of attributes. However, a multi-attribute representation was chosen because it allows for more concise structures and faster retrieval times, because simple string comparison can be used to match features and fewer nodes must be matched when comparing graphs.

4.4.2 Background Tasks

SAI mitigates the inherent complexity in structured cases by removing as much complexity from query-time tasks as possible and offloading processing to continuous background tasks that do not impinge on query time. Developers may use this for indexing newly stored graphs, a task that may require frequent-structure mining or, at the very least, structural comparison to determine if a newly created index already exists within the database. SAI provides two background tasks: one tests each index against each other index for compatibility, in order to build an indexing hierarchy to support index retrieval. The second task, which relies on the first, judges whether mutually compatible indices are equivalent and collapses them into a single index if appropriate. This assists lazy indexing strategies that do not insure the uniqueness of indices at the time a graph is indexed.

4.4.3 Storage

SAI's back-end is mySQL. SAI uses a typical relational database graph representation in which edges are rows in an edge table containing references to nodes, nodes are rows in a node table containing references to graphs, and graphs are rows in a graph table which indicates whether the graph is an index or not, and also contains a cache of otherwise derivable information, such as the number of nodes in the graph. Feature information, such as the class and value, are stored in another table. Features are associated with graphs, edges, and nodes using three auxiliary tables. Each of these tables (except for the feature information table) also contains a graph ID in each row. Once a graph ID is known, its content can be retrieved from the database by accessing rows in each of these tables associated with the graph ID.

To preserve the integrity of case content and ensure that the stored records associated with any cases loaded in memory will remain unchanged within the database, SAI does not allow updates to existing graphs. Instead, SAI performs updates by re-storing graphs using a new identifier and deleting the original graph. This insures integrity of the indices; an indexed case will always retain the content for which it was indexed. In addition, it simplifies some maintenance tasks discussed in the following section.

SAI could use a key-value store, such as BerklyDB, as a back-end, as many graph databases do (such as HyperGraphDB), but mySQL is used instead for three reasons:

- SAI provides an extensible schema that can be exploited by applications requiring a more complex storage strategy.
- SAI takes advantage of a number of small joins to enhance retrieval for similarity queries and index creation
- mySQL reduces the optimization requirements of SAI by providing features such as caching.

SAI provides two representations of graphs to the client software: ID's and full graphs. The initial retrieval phase does not load structural details of graphs into memory. Instead, they are referred to only by their ID. This avoids the need to unnecessarily load into memory graphs which may ultimately prove unimportant. Once a number of graphs have been eliminated from contention through preliminary processing, remaining graphs are fully loaded into memory.

4.5 Retrieval Task Delineation in SAI

For attributed graphs, similarity is often computed in terms of how close one graph is to being equal to or a subgraph of the other (often in terms of the number of edits required to form this relationship [69]). This is also referred to as inexact graph matching [10]. Note that inexact graph matching has also been used to describe matches of labeled graphs in which the labels do not exactly match [85]. For the purposes of this paper, inexact matching will refer to the tolerance of incompleteness in the match – not all edges and nodes between the two graphs must be matched, but rather the number of matches is taken to indicate the degree of similarity between the graphs.

Inexact matching of attributed graphs is additionally complicated by the constraints that attributes place on graph components. For instance, it is possible that a node from the query has a particular attribute that cannot be matched with a node in a retrieved case, due to a differing attribute value. To reflect the requirements for different nodes to match, we introduce a new property, which we call *compatibility.* Compatibility checks will be applied to graphs and their components to reflect whether they can be matched to form a partial subgraph isomorphism. For instance, using the default procedures provided by SAI, attributes with the same class and value are considered compatible. Depending on their needs, users of SAI may define their own compatibility criteria to be applied by the system.

Because the CBR retrieval task is to retrieve similar cases, not to locate existing identical cases, imperfect matches must be considered. Consequently, it is not possible to definitively establish whether the most similar graph has been found, without in some way considering all potentially related graphs. This issue is unique to the problem of inexact retrieval, which has not received as much attention as the problem of exact retrieval. This is further discussed in the related work section.

SAI was designed principally to deal with indexing and retrieval. However, for large case-bases, in which retrieval efficiency is crucial, retrieval and similarity assessment are inexorably intertwined: methods are required for indexing cases relative to their expected similarity, using sub-structures or other related structures as indices.

By identifying indices associated with a case, it is possible to compute a partial structural similarity assessment with the index's associated cases without performing structure matching.

Phala's structure retrieval is performed in two phases. The first is a coarse-grained process to eliminate most potential cases by only considering cases indexed by relevant features. Finer-grained comparisons are only performed for cases judged sufficiently similar by index-based comparisons. Two-step retrieval processes are common solutions to retrieval tasks wrought with complexity (e.g., for the structure matching of MAC/FAC [33]). While the two steps may be intertwined, we have identified a number of subtasks that can be implemented and extended independently. The following sections identify these tasks and provide detail.

4.5.1 Summary of Tasks in SAI

Figure 4.2 summarizes the tasks performed by SAI (termed *final tasks*) and those which can be altered or individually selected by developers (termed *abstract tasks*).[2] These tasks are not necessarily independent; many of them are either required by or potentially used by other tasks in these lists. These tasks are referred to throughout the next two subsections.

4.5.2 Structural Similarity Basics

Similarity assessment in SAI rests on a number of basic sub-tasks, described here:

[2]The terms *final* and *abstract* are borrowed from the Java programming language and are intended, respectively, to refer to tasks with complete implementations that cannot be altered and tasks whose implementations must be created or selected.

Final Tasks	Abstract Tasks
Node Compatibility	Feature Compatibility
Edge Compatibility	Feature-Set Compatibility
Index Compatibility	Node Map Utility Assessment
Index Equivalence	Node Map Generation
Full Graph Retrieval	Graph Similarity
Graph Storage	Graph Compatibility
Index Storage	Graph ID Retrieval
	Index Generation
	Index Retrieval

Figure 4.2: Tasks in SAI

Feature Compatibility: To support a rich range of attributed-graph comparison algorithms, SAI is designed to enable domain-specific similarity assessment methods to be plugged into the framework. Feature classes—instances of which can be applied to nodes, edges, or graphs—must be developed by those applying SAI. The feature class definitions must include a definition of compatibility specific to the feature class, to determine when two graph elements may be matched. By default, features are compatible if and only if they have identical class and value. This default is symmetric, but compatibility criteria are not required to be symmetric. Preserving symmetry in compatibility, if desired, is left to the implementers as well, should they choose to extend the default behavior.

Through this mechanism, a more knowledge-rich approach could be utilised within

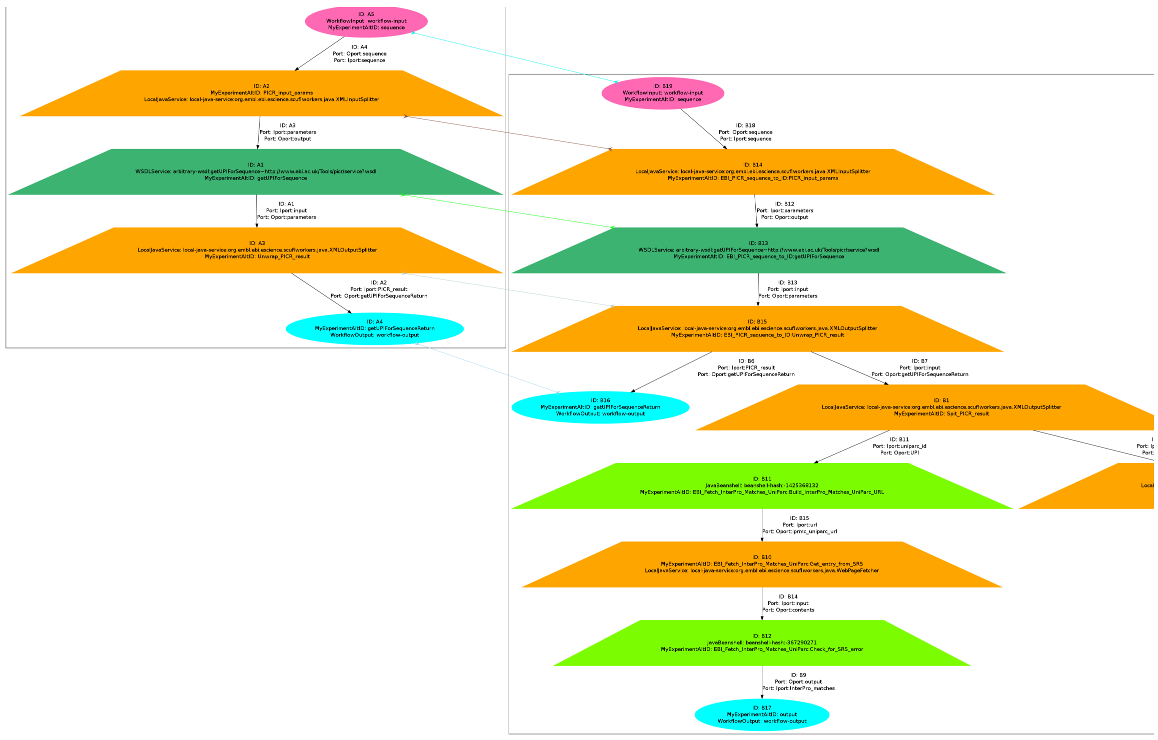

ID: A5
WorkflowInput: workflow-input
MyExperimentAltID: sequence
ID: A4
Port: Oport:sequence
Port: Iport:sequence
ID: A2
MyExperimentAltID: PICR_input_params
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.XMLInputSplitter
ID: A3
Port: Iport:parameters
Port: Oport:output
ID: A1
WSDLService: arbitrary-wsdl:getUPIForSequence~http://www.ebi.ac.uk/Tools/picr/service?wsdl
MyExperimentAltID: getUPIForSequence
ID: A1
Port: Iport:input
Port: Oport:parameters
ID: A3
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.XMLOutputSplitter
MyExperimentAltID: Unwrap_PICR_result
ID: A2
Port: Iport:PICR_result
Port: Oport:getUPIForSequenceReturn
ID: A4
MyExperimentAltID: getUPIForSequenceReturn
WorkflowOutput: workflow-output
ID: B19
WorkflowInput: workflow-input
MyExperimentAltID: sequence
ID: B18
Port: Oport:sequence
Port: Iport:sequence
ID: B14
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.XMLInputSplitter
MyExperimentAltID: EBI_PICR_sequence_to_ID:PICR_input_params
ID: B12
Port: Iport:parameters
Port: Oport:output
ID: B13
WSDLService: arbitrary-wsdl:getUPIForSequence~http://www.ebi.ac.uk/Tools/picr/service?wsdl
MyExperimentAltID: EBI_PICR_sequence_to_ID:getUPIForSequence
ID: B13
Port: Iport:input
Port: Oport:parameters
ID: B15
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.XMLOutputSplitter
MyExperimentAltID: EBI_PICR_sequence_to_ID:Unwrap_PICR_result
ID: B6
Port: Iport:PICR_result
Port: Oport:getUPIForSequenceReturn
ID: B7
Port: Iport:input
Port: Oport:getUPIForSequenceReturn
ID: B16
MyExperimentAltID: getUPIForSequenceReturn
WorkflowOutput: workflow-output
ID: B1
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.XMLOutputSplitter
MyExperimentAltID: Spit_PICR_result
ID: B11
Port: Iport:uniparc_id
Port: Oport:UPI
ID: B11
JavaBeanshell: beanshell-hash:-1425368132
MyExperimentAltID: EBI_Fetch_InterPro_Matches_UniParc:Build_InterPro_Matches_UniParc_URL
ID: B15
Port: Iport:url
Port: Oport:iprmc_uniparc_url
ID: B10
MyExperimentAltID: EBI_Fetch_InterPro_Matches_UniParc:Get_entry_from_SRS
LocalJavaService: local-java-service:org.embl.ebi.escience.scuflworkers.java.WebPageFetcher
ID: B14
Port: Iport:input
Port: Oport:contents
ID: B12
JavaBeanshell: beanshell-hash:-367290271
MyExperimentAltID: EBI_Fetch_InterPro_Matches_UniParc:Check_for_SRS_error
ID: B9
Port: Oport:output
Port: Iport:InterPro_matches
ID: B17
MyExperimentAltID: output
WorkflowOutput: workflow-output

Figure 4.3: Two structures from the myExperiment dataset (structures 126 and 127) in which the first is compatible with the latter.

SAI, rather than the syntactic comparison SAI uses by default to compare nodes. Phala uses this result to compare service types between processor nodes, however, a more knowledge-rich approach may improve the compatibility assessment by considering partial compatibility for two processors sharing a common abstract type or role, such as varied XML processing utilities.

Feature-Set Compatibility: Because structural components in SAI are multi-attributed, SAI requires a means of determining compatibility between sets of features. A central question when comparing sets of features is whether one feature can be matched with multiple features or not, and whether this distinction should be symmetric or not. SAI supports the ability to define compatibility checks reflecting each of these types of relationships.

For example, if graphs represented floor plans (such as with the a.SCatch system [119]), a feature may be a *sink*, indicating a sink is present in a room. When retrieving similar floor plans, a set of two sink features may or may not be compatible with a set only containing one sink feature, depending on the significance of the distinction between having at least one sink or multiple sinks in a room for the similarity measure.

Additionally, implementors may desire that some feature types be ignored (directly supported in SAI and discussed further in section 4.5.4), require context-sensitive constraints, or asymmetric compatibility. For this reason, feature-set compatibility can be selected amongst the listed defaults, or implemented to a custom specification.

Node Compatibility: SAI determines node compatibility by checking compatibility of the feature sets of each node. Beyond providing custom implementations of feature set compatibility, this process cannot be altered.

Edge Compatibility: SAI determines edge compatibility both by checking compatibility of the feature-sets of each edge and by checking the compatibility of the pairs of source and target nodes for the edge.

Graph Compatibility: By default, two graphs are deemed compatible if each of the components of the first graph can be matched with unique compatible components of the second graph. Figure 4.3 depicts two structures within SAI for which the first is compatible with the second. Note that this is analogous to subgraph isomorphism and is not symmetric.

This task may be replaced by custom implementations for the purposes of domain-specific algorithmic optimization. However, this is also an opportunity to expand the compatibility assessment task to include knowledge-rich techniques which examine semantic properties of the SAI graphs, and weigh them with the structural compatibility (which Phala solely relies on). For example, myExperiment allows its users to tag workflows. Comparison of these tags between graphs could potentially be used to adjust the compatibility assessment determined from structural alignment.

4.5.3 Structural Indexing

SAI is designed to provide fast similarity-based retrieval for very large case-bases of structured cases by enabling the use of indices more expressive than the atomic features often used in CBR, and thus providing increased opportunity to combine structural matching with more efficient indexed retrieval. SAI accommodates a variety of indexing styles by implementing indices themselves as graphs. This approach not only accommodates indexing by frequently occurring substructures, but also alternative retrieval strategies for handling situations in which analogous or easily adaptable cases may be desired over cases that are, symbolically, more similar (examples of this are illustrated in section 4.5.4).

All graphs in SAI are first indexed by their atomic features. SAI additionally indexes cases by other graphs that are not identified as cases. These structural indices, which will be referred to simply as *indices* in the remaining discussion, are intended to be small, making them easier to retrieve than full cases by system-specific index similarity criteria. They are stored in the database in the same way other graphs are. Each graph in the database has a flag indicating whether or not it is an index. Following is a description of each of SAI's indexing subtasks:

Index Compatibility: Compatibility between structural indices and other graphs is determined in the same way graph compatibility is. Compatibility is used to determine if an index should be associated with another graph for retrieval purposes. Such relationships between existing indices and graphs are not guaranteed to be known by SAI, however an optional maintenance task updates the list of associated indices for graphs in the database when a compatible index is discovered. Since SAI does not allow edits to existing graphs, SAI can guarantee that all indices associated with a graph will forever be compatible with that graph.

Index Equivalence: SAI additionally does not ensure that indices are unique. Two indices are determined to be equivalent if they are compatible in both directions. Again, SAI provides an optional background task which tests each pair of indices to determine equivalence, and deletes one of the pairs from the database, updating each indexed structure's list of indices to only include the index which was not deleted.

Index Retrieval: SAI allows for custom algorithms for index retrieval, in order to allow a balance between computational efficiency, expressiveness of indices, and completeness of retrieval. Index retrieval is the first step of a similarity query in which indices compatible with the query are retrieved. As the structure of indices becomes more complex, the process of retrieving indices for a query also becomes more complex. For instance, in general, determining index compatibility is NP-Complete. In addition to indexing graphs, indices can also index other indices, allowing for hierarchical retrieval in situations for which larger indices become more difficult to retrieve by atomic features.

Index Generation: At the time a case is committed to the case-base, SAI will apply any available indexing algorithms to the newly stored case. These algorithms may retrieve compatible indices (sidestepping the need for some of the listed main-

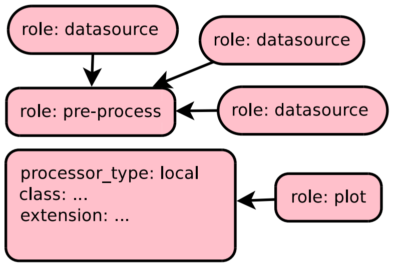


Figure 4.4: A structural index used to form an analogy.

tenance tasks), generate new indices (improving response time), or some mixture of both.

4.5.4 Multiple Views over Feature Sets for Typing

Each of the available similarity assessments can be restricted to consider only a subset of available feature classes. Rather than typing nodes and edges, as similar systems often do (e.g., [122]), SAI allows for variation in the type of features considered for comparison, thus forming types which cater to the developer's specific comparison desiderata on the fly. SAI's approach has several advantages:

- Comparison granularity can be increased or decreased. For example, in Fig. 4.1, if only "processor type" and "database" are considered, each of the top three nodes becomes interchangeable with one another or with other Ensembl sequences. Considering "dataset" features in addition to these provides tighter constraints, requiring that the same species is indicated.
- Similarity can be judged relative to a particular context. For example, in Fig. 4.4, "processor type", "url", "class", and "script" features could be used to

compare workflows for similar processes, whereas "name" features could potentially be used to find re-engineered versions of this workflow that use newer or more appropriate services.

- Abstract views can aid in analogical processing. For example, the graph in Fig. 4.1 could be indexed by the graph shown in Fig. 4.4. This index uses features that identify the roles of various processors in the original workflow and can also index analogous workflows that may fit this role template.

4.5.5 Inexact Structural Similarity Assessment

Node Map Utility Assessment: As mentioned in section 4.5.2, determining graph compatibility relies on a mapping between the nodes of two graphs. Determining similarity for inexact compatibility likewise relies on this mapping. Algorithms for this task must determine how closely a mapping approximates full compatibility. This may be done in multiple ways, but by default, SAI simply counts the number of matched compatible edges induced by a particular mapping to determine its utility.

Node Map Generation: This subtask generates the maps judged by the map utility heuristic. Implementations of this task vary widely in related work. SAI includes a greedy (quadratic time) and exhaustive (exponential time) map generator, as well as a search framework for developing search-based generators, such as the A* approach adopted by Bergmann and Gil [12].

Graph Similarity: Multiple maps may be generated by multiple algorithms to determine similarity between two graphs. Similarity is determined from the map produced with the greatest utility.

4.5.6 Inexact Structural Similarity Based Retrieval

Index Retrieval: The first step in SAI's retrieval process is to determine which indices within the database are compatible with a structural query, and to load these structures into memory. Components for this step are intended to be developed alongside index generators.

Graph ID Retrieval: Indices identified by the index retriever are propagated to the graph ID retriever which ranks graphs in the database according to their association with the set of retrieved indices. SAI includes an implementation of this step which simply orders cases by the number of indices they're associated with, though this can be replaced by a custom implementation. A pre-defined number of ID's of these cases (called Window 1) are then propagated to the full graph retrieval step.

Full Graph Retrieval: In this task, structural details of the propagated case ID's are loaded for full structural comparison. Maps between the query and these cases are generated and used to re-rank the cases. A predefined number of the highest scoring cases and maps (called Window 2) is returned in response to the query.

4.6 Evaluation of SAI's Retrieval in the e-Science Domain

SAI currently doesn't include an implementation of a highly sophisticated indexing algorithm such as those mentioned in section 4.3. However, it does include a simple strategy that enumerates all paths of length 1 (a single edge), including select attributes from the edge and connected nodes, similar to the approach used by graphgrep. This indexing option was developed because the single-edge indices hold the advantage of $O(n)$ index generation while maintaining sufficient distinguishing power within the e-Science workflow domain for cases. This section will demonstrate that these short path indices are sufficient for the e-science domain but not for others, giving evidence to the importance of SAI's modular approach, and also that the result from the iGraph experiment, which determined that no single algorithm is best for every dataset, carries over into the realm of approximate supergraph retrieval.

4.6.1 Terminology

The following terminology is used through the remainder of this section:

- *Phase 1* - an initial, index-based retrieval process
- *Phase 2* - a fully structurally aware, fine-grained similarity assessment
- *Phase 1 Window* - the maximum number of cases Phase 1 provides to Phase 2 (shortened to *Window 1*)
- *Phase 2 Window* - the number of cases returned from the Phase 2 process for further processing by the CBR system (shortened to *Window 2*).
- *Path1* - the indexing strategy discussed in the previous paragraph

4.6.2 Experimental Datasets

Multiple sources of data were used to form datasets for the experiments in the remainder of this section. Both of these datasets were mined from sources on the web.

These include:

4.6.2.1 The myExperiment Dataset

To determine how effective SAI's retrieval algorithm is for the e-Science domain, a collection of cases was created by tracing the data-flow through 341 workflows gathered from the myExperiment website [3]. This database has the following characteristics:

- Size (cases): 341
- Average case size: 11.8 nodes
- Standard deviation of case size: 12.88
- Average pairwise similarity: 1.1
- Features: 3883
- Features per case: 11.3
- Indices: 1627
- Index-to-case ratio: 4.77
- Index-to-node ratio: 0.40
- Index-to-edge ratio: 0.41

Each case in the myExperiment dataset is very unique from each other. The diversity in these cases forces a large number of unique indices, which would be much larger if an indexing technique other than Path1 were chosen, considering Path1's

[3]http://www.myexperiment.org

very small index space. The large number of indices per case in this dataset makes the indices more distinguishing; relatively few cases are associated with any one index.

The low pairwise similarity can also be noted in Figure 4.5, a heat map of the similarity between pairs of cases from the myExperiment dataset. Note that there are often clusters of similar cases along the diagonal. Since the times at which these cases were created are somewhat similar (cases are ordered by the date they were added to the myExperiment website), it's likely that these small clusters represent revisions of a workflow by the same author or team of authors.

Also, notice that many of the bright spots are reflected across the same diagonal axis. This symmetry results from small clusters of cases having a symmetric similarity relationship. Recall that this is not necessarily always the case when similarity is assessed as distance to subgraph isomorphism, since subgraph isomorphism itself is not symmetric. For example, a graph of 40 edges could be at most 50% similar to a graph of 20 edges, and a graph of 20 edges could be 100% similar to a graph of 40 edges.

4.6.2.2 The PubChem Dataset

To explore the impact of properties of the dataset, a different data source was sought with significantly different characteristics from the myExperiment dataset. The PubChem website [4] contains a large library of chemical structures. These structures have been used to evaluate many of the graph databases mentioned in this chapter. The first 100 molecular structures were downloaded from this website to form the dataset. This dataset has the following characteristics:

[4]http://www.pubchem.org

- Size (cases): 100
- Average case size: 17.8 nodes
- Standard deviation of case size: 14.58
- Average pairwise similarity: 29.30%
- Indices: 8
- Index-to-case ratio: 0.08
- Index-to-node ratio: 0.0045
- Index-to-edge ratio: 0.0040

The PubChem dataset exposes the limitations of the Path1 indexing strategy and why more computationally expensive indexing strategies can be better choices in other domains. Path1 generates very few indices in the PubChem dataset, and the index ratios are much lower than they were for the myExperiment dataset. This indicates that there is little variation in the labels, which is true. Most of the compounds in the PubChem dataset are organic, meaning that the majority of bonds will be between hydrogen, carbon, oxygen, and nitrogen atoms, though hydrogen atoms were left out of the structures for simplicity. This leads to a total of only eight indices.

The lack of diversity also leads to much greater similarity between cases. While most pairs of cases in the myExperiment dataset shared no similarity at all, most pairs of cases in the PubChem dataset share some similarity, leading to a 37.3% average similarity, more than 30 times that of the myExperiment dataset. This is also easily recognizable in the heat map in Figure 4.6, as there is very little pure black, indicating most case pairs are related.

Another unique feature of the heat map is the vertical and horizontal bars, which indicate there are some cases that are similar to most other cases or there are cases to which most other cases are similar. Across the diagonal axis of symmetry, the mirrored images of some of these lines are significantly brighter or darker than their counterparts. This is due to the fact that small structures will have high similarity to most larger structures, whereas most larger structures will have relatively low similarity when compared to the smaller structures. For instance, the dark vertical bars near the center of the diagram (structures 52 through 54) are also part of fairly bright horizontal bars. Each of these structures contains more than 50 nodes, far beyond the average for the database. This means that other structures are likely to be mappable to these structures, creating the bright horizontal bars. Also, these structures are unlikely to be completely mappable to other structures, though some portion of these structures will likely be mapped. This forms the dark, but brighter than black, vertical bars.

4.6.3 Experiment 1: Evaluating Each Phase of Two-Phase Retrieval Algorithms

Evaluation of a retrieval algorithm implementation through SAI is not straightforward, due to the fact that it is composed of two independent retrieval phases. This subsection explores the issue and provides justification for the choice of a computationally inexpensive yet functionally limited phase 1 strategy for use in the e-Science domain.

4.6.3.1 Comparing Phase 1 and Phase 2 Rankings

In two-phase retrieval, the first phase can be thought of as a filter. In a sense, the order of the cases returned from the first phase is irrelevant, since they will be re-ordered in phase 2. Since each case will be compared to the query in phase 2, there is no benefit to having phase 1 results ordered similarly to the order phase 2 will generate.

There is a benefit, however, in viewing phase 1 results as a ranking. Cases can be ranked by phase 1 by considering progressively longer window 1 values, beginning with a window 1 value of 1. In this case, the most similar case according to phase 1 will be returned. Increasing window 1 to the value 2 will retrieve the two most similar cases, and by comparing this result to the previous result, the second most similar case according to phase 1 can be determined. This process can be repeated to form a ranking.

Even though ranking the results from phase 1 provides no benefit in the retrieval process, it can be useful when analyzing the retrieval process to determine whether phase 1 or phase 2 is most to blame for poor retrieval performance. This can be determined by comparing the ranking produced by phase 1 to the ranking produced by phase 2, in a manner similar to that used by Boegarts and Leake [15].

Within the evaluation in this chapter, the method for determining which cases phase 1 will return is performed by counting the number of indices it shares with the query. Since this is an integer, there are frequently ties between candidate cases: more than one candidate case may share the same number of indices with the query.

Boegarts and Leake discuss how to effectively compare rankings complicated by ties as well as other issues related to the relative lengths of the rankings. A ranking based on these principles was developed for comparing phase 1 and phase 2 rankings for SAI.

4.6.3.2 Assuring the Soundness of Phase 2 Rankings

Judging the effectiveness of the phase 1 ranking relies on having an ideal ranking to compare it to. The phase 2 ranking should offer such a standard, but greedy algorithms aren't always capable of producing an ideal ranking within a reasonable amount of time for answering a query. The processing time for phase 2 ranking was increased to 10 seconds per case to develop an ideal ranking. Each case was compared to every other case. This process was then repeated, and in some instances, a larger map was found between some pairs of cases, indicating a higher similarity. This is owing to the fact that a greedy algorithm was used, which doesn't guarantee that a maximal mapping will be discovered. This process was repeated until no further improvements on prior mappings were discovered.

To verify that this was a sufficient amount of time, each case was compared to itself with the expectation that a 100% match would be derived. This result was achieved after running three iterations the greedy algorithm. The results showed a 99.8% match for PubChem and 100% for myExperiment after the first iteration.

4.6.3.3 Experimental Setup

In order to determine the effectiveness of the Path1 indexing strategy, rankings from both phases were compared after applying a variety of values for Window 1 and

myExperiment					
Window 1/2	1	2	3	5	10
5	99.70	96.29	93.61	90.74	N/A
8	99.71	98.04	96.68	95.34	N/A
10	99.71	98.44	97.52	96.41	95.91
15	99.71	99.12	98.78	98.31	98.04
20	99.71	99.41	99.24	98.94	98.72
25	99.71	99.41	99.24	99.04	98.88
30	99.71	99.61	99.54	99.31	99.17

Table 4.1: Average Rank Quality for the myExperiment dataset.

Window 2. Each case was compared to every other case for both the myExperiment and PubChem datasets to form these rankings.

4.6.4 Results

Rank quality for the myExperiment dataset (Table 4.1 and Figure 4.7) and for the PubChem dataset (Table 4.2 and Figure 4.8) were recorded to show how well Path1 produced an ideal ranking of the cases relevant to each query. Additionally, a modified recall measure dubbed *containment* was recorded for the myExperiment (Table 4.3 and Figure 4.9) and PubChem (Table 4.4 and Figure 4.10) datasets. The difference between this measure and recall is that not all *relevant* structures are given equal weight. Each successive structure within an ideal ranking of the relevant structures is given half the weight of the preceding structure.

PubChem					
Window 1/2	1	2	3	5	10
5	82.82	63.29	56.25	51.65	N/A
8	86.86	70.7	64.42	60.05	N/A
10	88.88	74.74	68.88	64.63	63.28
15	89.89	79.12	74.03	70.56	69.45
20	94.94	86.19	82.95	79.5	78.44
25	94.94	86.86	84.31	81.24	80.3
30	95.95	89.56	87.38	84.49	83.62

Table 4.2: Average Rank Quality for the PubChem dataset.

myExperiment					
Window 1/2	1	2	3	5	10
5	100.0	96.58	93.93	91.13	N/A
8	100.0	98.34	96.98	95.70	N/A
10	100.0	98.73	97.82	96.73	96.25
15	100.0	99.41	99.08	98.62	98.35
20	100.0	99.71	99.54	99.24	99.02
25	100.0	99.71	99.54	99.34	99.19
30	100.0	99.90	99.83	99.60	99.46

Table 4.3: Average Containment for the myExperiment Dataset.

PubChem					
Window 1/2	1	2	3	5	10
5	84.84	64.98	58.29	53.86	N/A
8	87.87	71.38	65.36	61.15	N/A
10	89.89	75.42	69.84	65.81	64.64
15	89.89	79.12	74.17	70.93	69.98
20	94.94	86.19	83.11	79.83	78.86
25	94.94	86.86	84.41	81.45	80.59
30	95.95	89.56	87.44	84.62	83.81

Table 4.4: Average Containment for the PubChem dataset.

4.6.4.1 Analysis

The results show a stark difference between the performance of the Path1 indexing strategy on each of the two datasets. This verifies the expected result from analyzing the properties of each dataset. Path1 performs relatively poorly for small Window 1 values (at times only retrieving approximately half of the relevant structures), whereas the performance of Path1 on the myExperiment dataset never dips below 90% for rank quality or containment.

Path1 on myExperiment performs well with relatively small values for Window 1. When only retrieving the most relevant structure, only slight deviations are seen from the ideal ranking (less than 1%) and containment is perfect (the most relevant structure is always within the top five ranked structures according to Path1).

Given that Path1 performs below desired benchmarks for the PubChem dataset, it is not universally suitable as an indexing strategy. However, given also that Path1 is less computationally complex than most indexing strategies and that it works well with the myExperiment dataset, it is not only sufficient for use in the e-Science domain but also desirable. This justifies the creation of the SAI framework through demonstration of its ability to facilitate development of two-phase retrieval algorithms and analyze their performance and applicability to different domains.

4.6.5 Scalability of Two-Phase Retrieval

A central goal of SAI is to facilitate the retrieval of large datasets of structured data. The two datasets used in previous sections (myExperiment and PubChem) use a medium number of cases. This works well to examine issues related to similarity assessment, but it doesn't show how well SAI scales with large or very large databases. Large case-bases are likely to occur when employing a case-mining technique, such as that used by the Phala system, making it important to observe how well SAI scales up to performing retrieval within large databases of graphs. This sub-section examines these issues.

4.6.5.1 Experimental Setup

Tests were run by generating random graph databases with 2.5 and 10 unique feature values per graph on a system using an AMD Phenom II X4 955 Processor with 2 Gigabytes of RAM and mySQL 5.5.13 and SAI running in a single process. Graphs (trees) were randomly generated according to a normal distribution with an average size of 14.2 nodes and a standard deviation of 16.9 nodes (no graphs with fewer than 2 nodes were generated).

4.6.5.2 Results

Figure 4.11 shows index and case retrieval results for 100 random case retrievals. Recorded times are for phase 1 tasks only. Index retrieval time refers to the amount of time spent retrieving indices. Since phase 2 similarity assessment is not affected by database size, it was not included in the experiment. Figure 4.12 illustrates the sub-linear growth of retrieval time for the data generated with 0.5 features per case. Blue lines connect data-points from Figure 4.11. Each of the red dotted lines indicate the trajectory of a straight line through the preceding point. Note that, in each case, the slope is much more severe for the linear model than the actual retrieval time. Similar results can be generated for each of the other three feature to case ratios.

4.6.5.3 Analysis

This data shows that Path1 indexing is very effective for large datasets with properties similar to the myExperiment domain. The small index space prevents the database from growing quickly, which keeps index retrieval times sub-linear with respect to the size of the case-base as the case-base grows to 100,000 cases. Retrieval times remain in the order of a few seconds with databases as large as 100,000 cases.

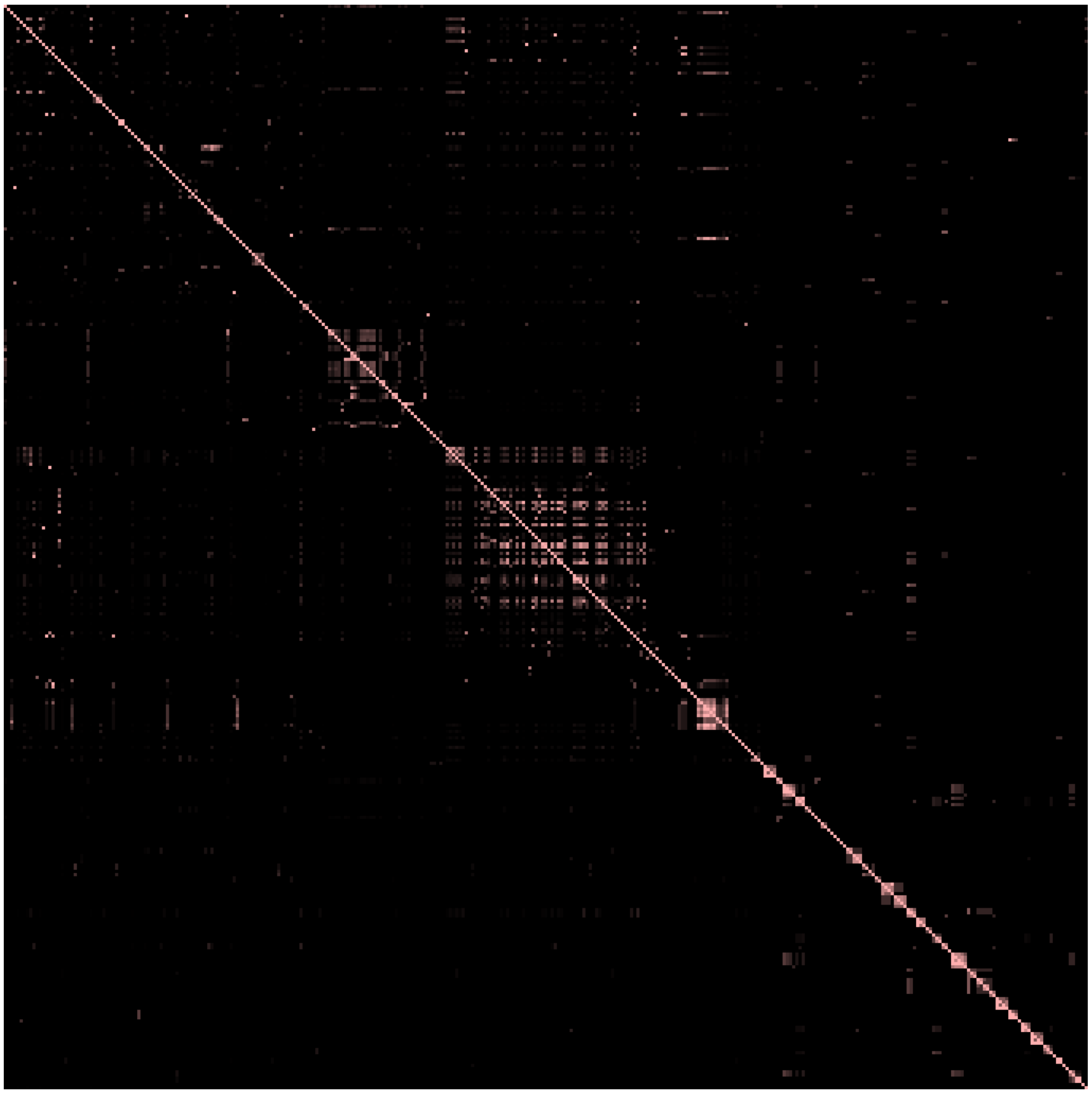


Figure 4.5: Distance to subgraph isomorphism for each pair of cases from the myExperiment Dataset.

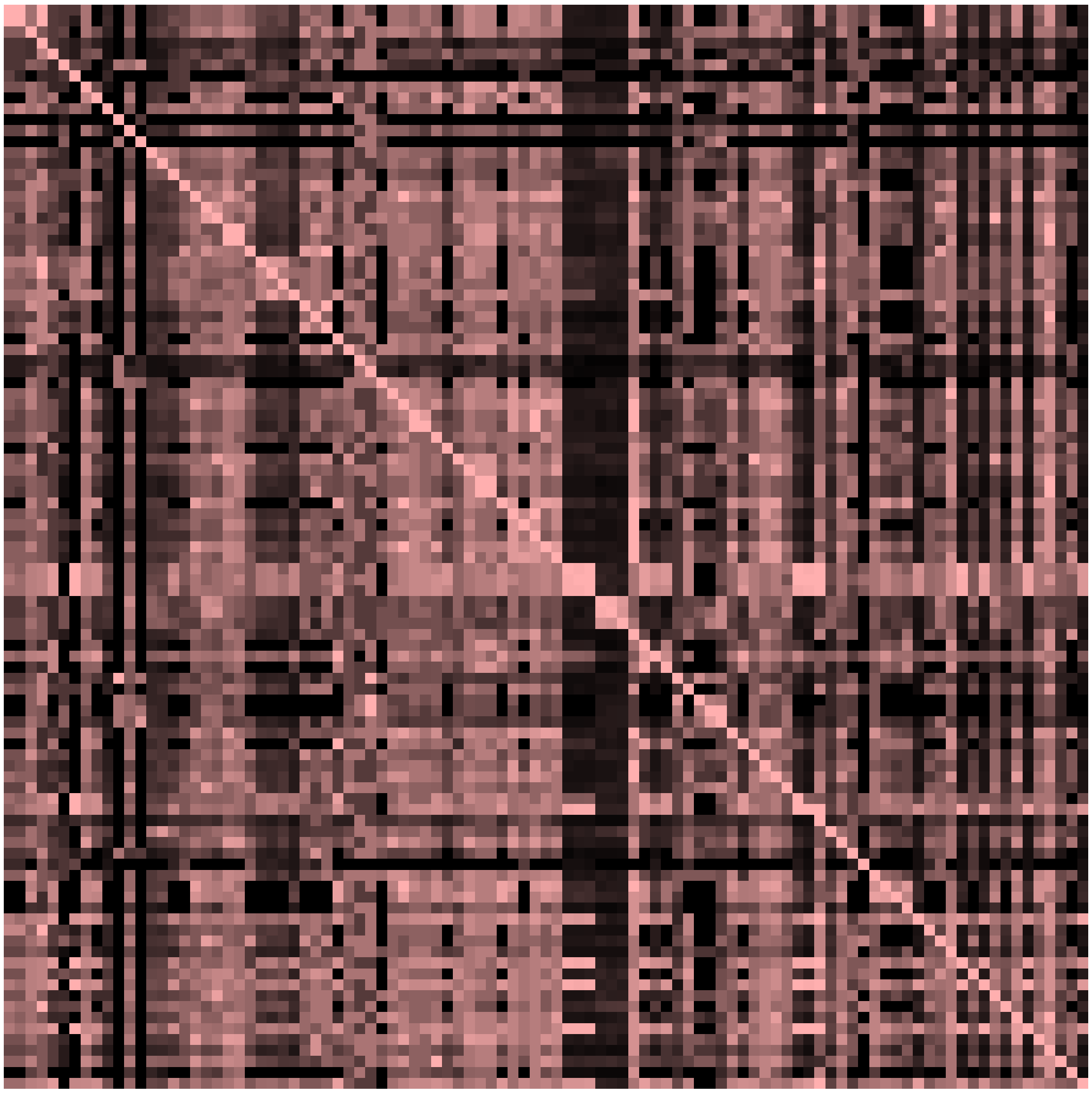


Figure 4.6: Distance to subgraph isomorphism for each pair of cases from the PubChem Dataset.

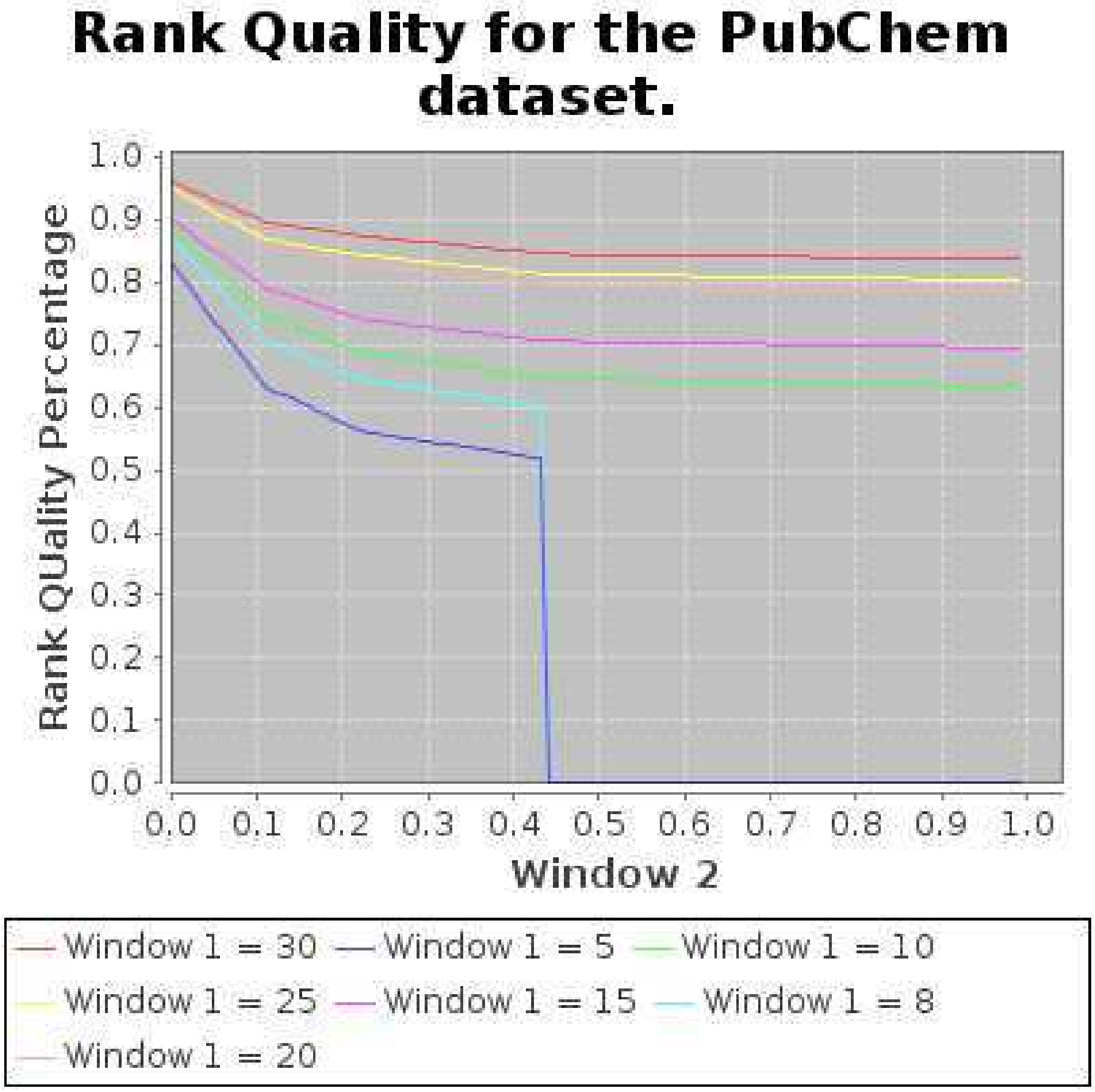


Figure 4.7: Phase 1 Rank Quality with the myExperiment Dataset

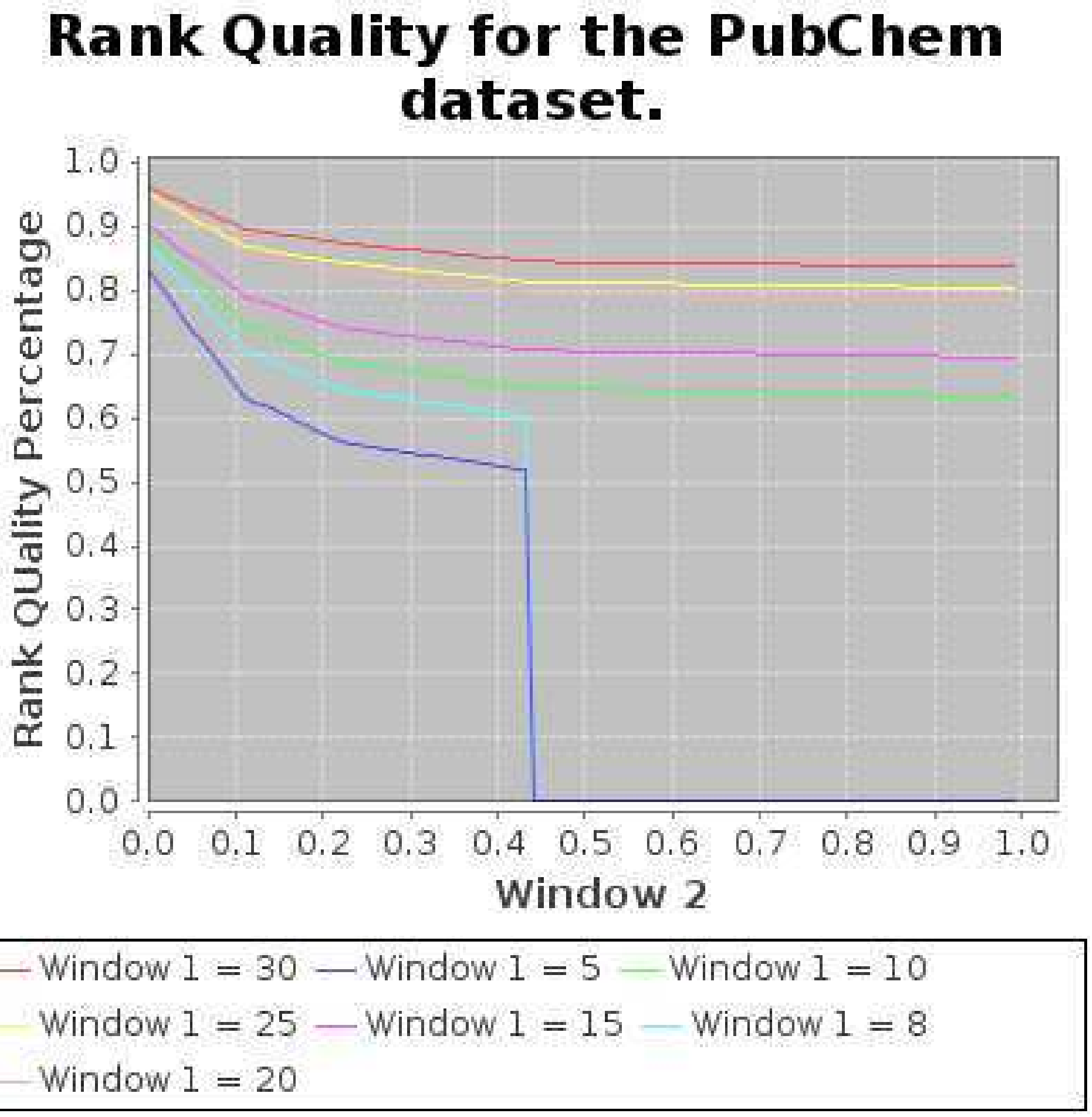


Figure 4.8: Phase 1 Rank Quality with the PubChem Dataset

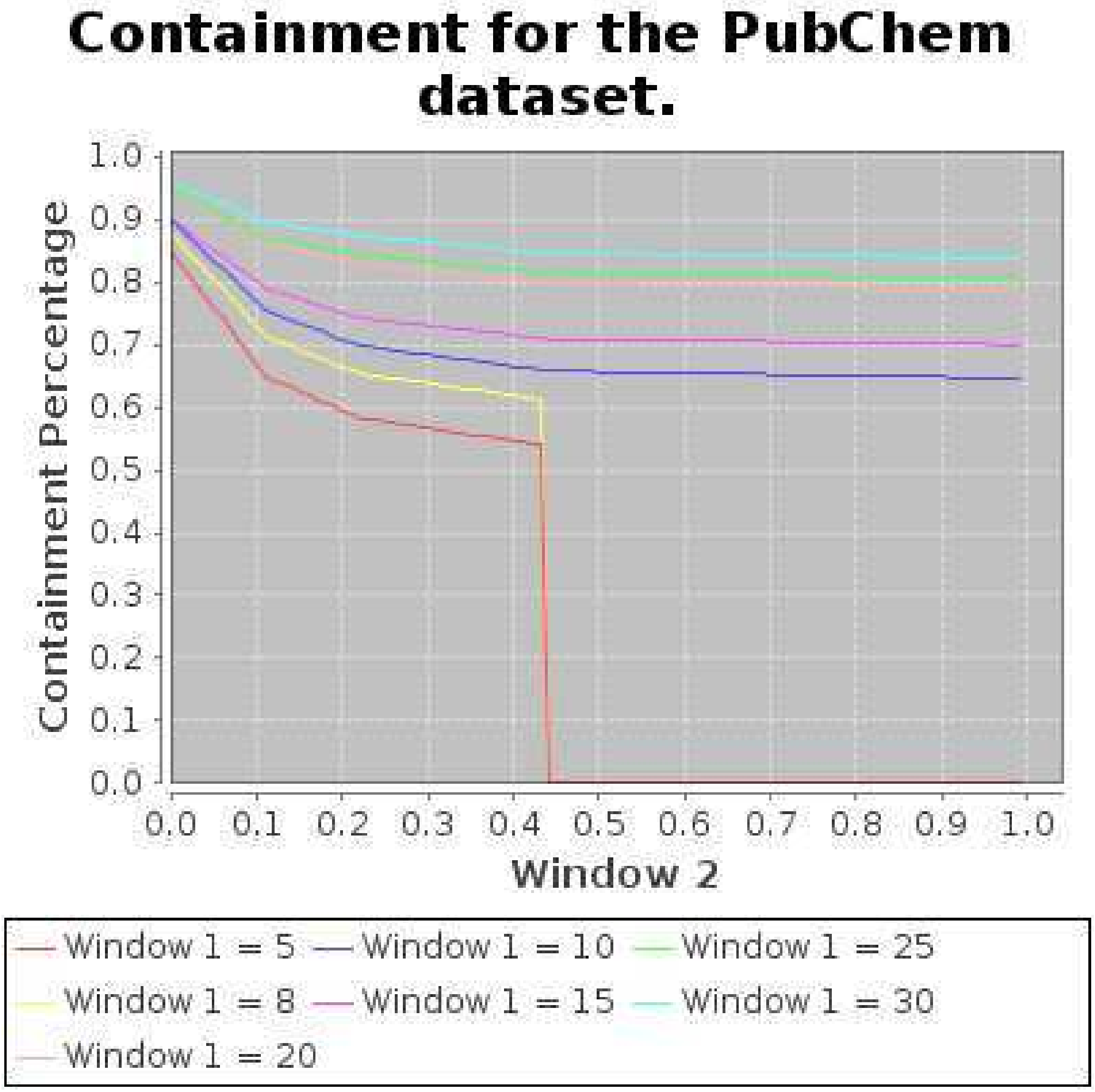


Figure 4.9: Phase 1 Containment with the myExperiment Dataset

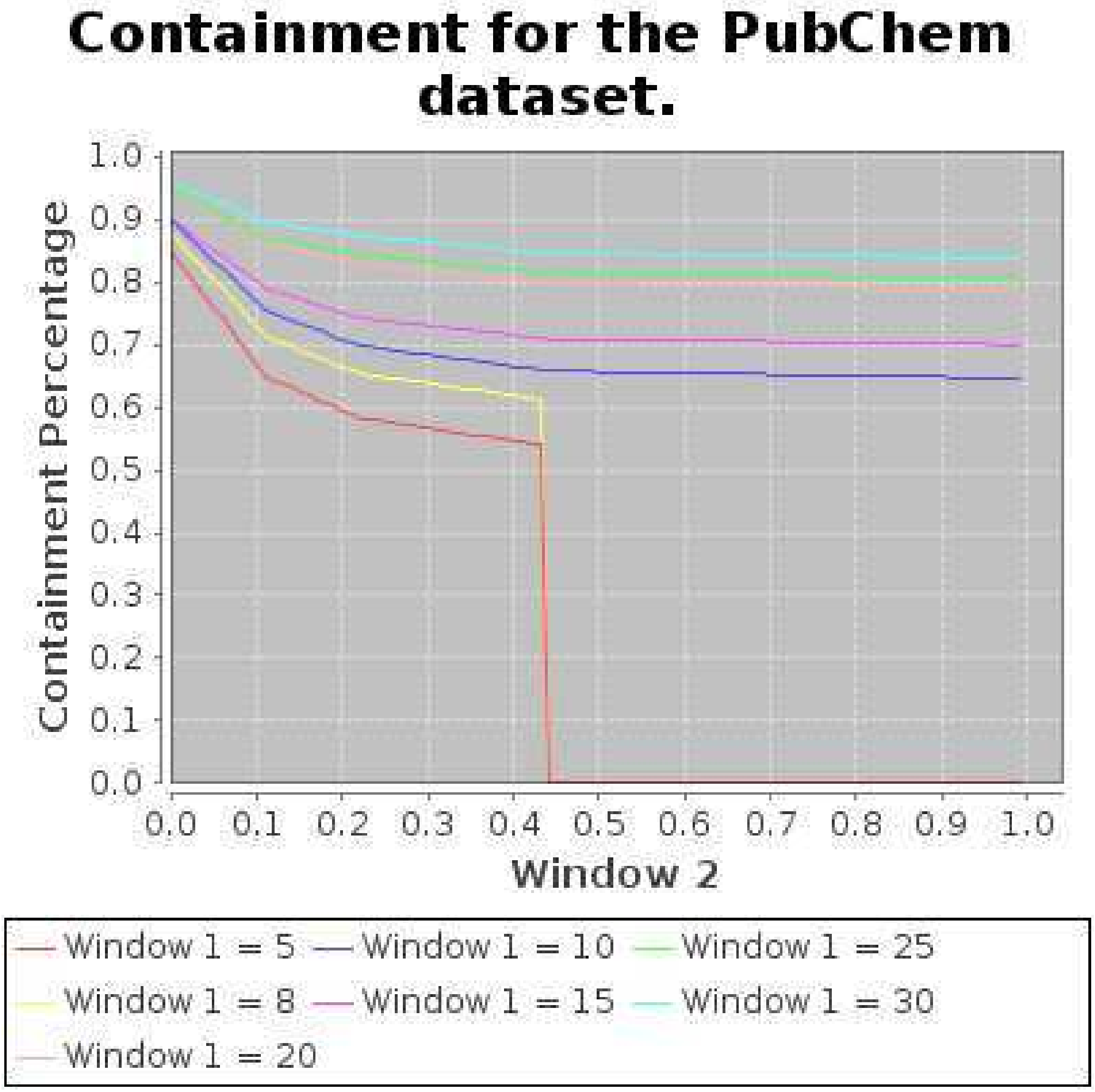


Figure 4.10: Phase 1 Containment with the PubChem Dataset

Features/Graph	Case-Base Size	Index Retrieval Time
0.5	100	29 milliseconds
0.5	1000	97 milliseconds
0.5	10000	346 milliseconds
0.5	100000	2591 milliseconds
2.5	100	77 milliseconds
2.5	1000	155 milliseconds
2.5	10000	378 milliseconds
2.5	100000	2251 milliseconds
10	100	128 milliseconds
10	1000	203 milliseconds
10	10000	432 milliseconds
10	100000	2483 milliseconds
15	100	186 milliseconds
15	1000	256 milliseconds
15	10000	495 milliseconds
15	100000	2664 milliseconds
50	100	233 milliseconds
50	1000	323 milliseconds
50	10000	549 milliseconds
50	100000	2826 milliseconds

Figure 4.11: Performance of SAI on random graph retrievals.

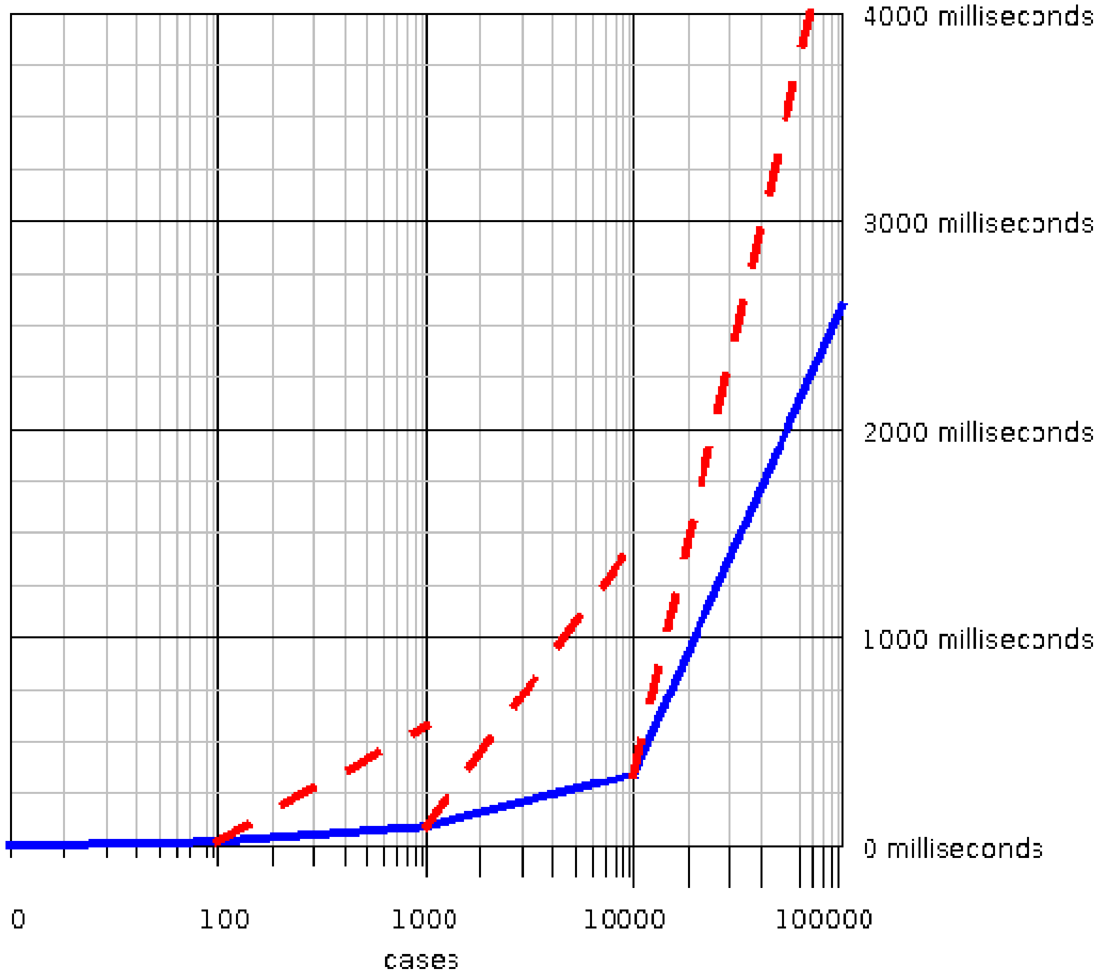


Figure 4.12: An exponential plot of retrieval times for 0.5 features per graph.

Chapter 5

Evaluation of the Phala Workflow Edit Recommender System

5.1 Introduction

Throughout this dissertation, Phala has exemplified the principles put forth and served as a *proof of concept* that case-based reasoning can facilitate knowledge-light, recommendation-based assistance for a synthesis task. This chapter provides a thorough evaluation of Phala and shows that Phala provides recommendations that are useful and save users time. This is done through an empirical, objective study of the recommendations generated by Phala and their internal components performed by forming tests with a collection of expected recommendations associated with them and comparing the content of the recommendations produced by Phala to those expected recommendations. Beyond justifying the value of Phala, this chapter also provides important lessons in evaluating intelligent assistants, including a novel metric that considers important aspects of the user's experience and directly models the value obtained through use of an intelligent assistant, as opposed to merely judging the value of the content of the recommendations.

This chapter begins by presenting a series of performance metrics for a recommendation-based assistant employing principles from the Hybrid Recommendation Framework. Next, the formulation of tests for Phala is detailed. Following this, each of the components of Phala's hybrid architecture are discussed. The chapter concludes with a report on the methodology and results of the tests of Phala.

5.2 Performance Metrics for the Recommendation Task

Helpfulness and diversity are both important performance metrics, but presently are difficult to use for evaluation of Phala. Helpfulness is a subjective measure, and while Phala could do much to estimate helpfulness, the accuracy of these estimations cannot be computationally measured in the same way others can be. The automated evaluation methods presented in this chapter cannot be applied to measuring helpfulness.

Diversity can be measured, but relates to sets of recommendations more than to a single recommendation. Recommendations for workflow edits are complex, and inundating the user with a number of recommendations would likely cause more problems than it would solve. With this in mind, Phala was designed to present one recommendation at a time, decreasing the importance of diversity.

Instead, both correctness and relevance will be extensively studied. Recommendations will be deemed *relevant* if their problem components match an expected problem within a test. If they do not, they will be deemed *irrelevant.* Relevant problems will further be deemed *accurate* if their solutions match that of an expected solution. Though the term *accurate* is used, it should be noted that the solution derived from the original workflow may not be the only *correct* solution, in that alternatives may exist. It is possible that Phala will produce a perfectly valid recommendation that does not match the expected solution, which returns the workflow to its original state. In these cases and those in which Phala recommends a solution that doesn't match the expected solution, recommendations will be deemed *inaccurate*, which can be read as an upper bound on true incorrectness. The rate of *accurate* recommendations thus serves as a lower bound on true correctness.

Figure 5.2 illustrates an example scenario for judging recommendations. In this example, three service addition recommendations are made, each indicated by the dotted lines protruding from the top two nodes. It is expected that a recommendation will be made to add a service from the processor with service B to a new processor with service C. Therefore, the recommendation matching this expected recommendation is labelled relevant and accurate. The recommendation adding service D is not accurate because D was not the expected service type; however, it is relevant since the problem component matches. A third recommendation is to create a processor with the correct service C, but this is judged as irrelevant since it is added off of the processor with service A instead of the processor with service B.

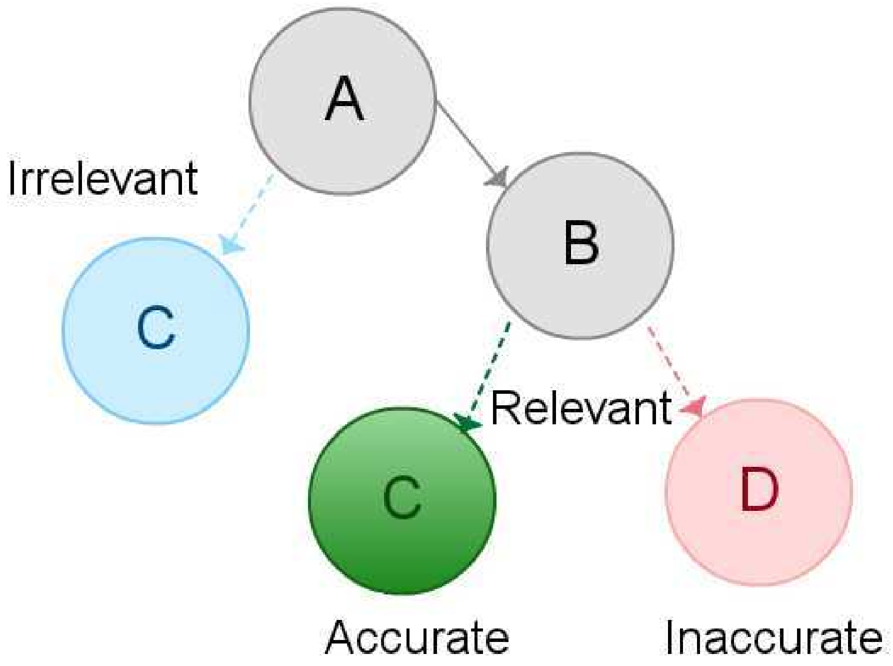


Figure 5.1: Example recommendations for Phala

5.2.1 Basic Metric Definitions

Each expected recommendation for a test will be referred to as an *opportunity*. There may be multiple opportunities generated for each test. For example, a completion test may involve two opportunities if the deleted processor had an input edge connected to another processor and an output edge connected to a third processor. Each of the following four metrics are plotted against a minimum confidence threshold where only recommendations estimated to be above the indicated threshold are considered.

5.2.1.1 Recommendation Rate

The *recommendation rate* is defined as being the proportion of opportunities for which a relevant recommendation was produced. That is, two relevant recommendations that match the same problem component of an expected recommendation are only counted once.

5.2.1.2 Satisfaction Rate

The *satisfaction rate* is similar to the recommendation rate metric but applies only to opportunities for which a relevant recommendation has been produced. Thus, it is not dominated by the recommendation rate; rather, it measures how well the system develops accurate solutions, independent of its ability to generate relevant problems. The number of expected recommendations for which a recommendation with a matching problem and solution has been generated is divided by the total number of expected recommendations for which a relevant recommendation has been generated.

5.2.1.3 Relevance Rate

The *relevance rate* is defined as being the proportion of relevant recommendations generated compared to the total number of recommendations generated. Independent of *opportunities* this metric provides a good measure of the average relevance of each recommendation produced were the user to look through the entire set of returned recommendations. The value of this metric is limited by the fact that the user is unlikely to pore through a large number of recommendations, especially if they are ordered by confidence and few of the top recommendations are deemed useful. This metric is plotted against a minimum relevance confidence threshold where only recommendations estimated to be above the indicated threshold are considered.

5.2.1.4 Accuracy Rate

The *accuracy rate* is similar to the relevance rate, but as with the satisfaction rate, only relevant recommendations are considered.

5.2.2 The Impact of User-Specified Preferences

An effective case-based reasoner must be more than a *black-box*. User interaction can have a significant role in each stage of the CBR process. In Phala, the user affects more than just the retrieval and retention of cases. Confidence values, derived from estimates of performance metrics discussed in section 2.6, play a role in what the user sees and does once recommendations are delivered.

In section 2.5.2.3, it was noted that if the user were aware of the problem within the workflow, the recommender could simply recommend solutions for that problem, eliminating the need to first generate and consider problems within the query structure. The user can specify what the problem component of the recommendations should be when the user is aware of what portions of the workflow s/he need assistance with. In the case where the user is not immediately aware of this, knowledge of the likelihood of a potential problem being relevant allows the recommender system to choose whether or when to display a particular recommendation to the user. A threshold could be set for estimated recommendation relevance, where only recommendations likely to be relevant are presented to the user.

5.2.2.1 Satisfaction vs Recommendation Plot

As the minimum threshold for confidence increases, the number of matched problems decreases, owing to the fact that the number of total recommendations is decreasing. This shows that there is a range of possible recommendation rates for a system with an upper and lower bound. Any of these recommendation rates can be achieved by the system by adjusting the minimum confidence threshold appropriately. Given a selection of a confidence threshold, there will also be a satisfaction rate associated with that confidence threshold. Given this perspective, we could view satisfaction as a function of the recommendation rate. A *satisfaction vs recommendation plot* graphs this relationship.

Such plots are an important means of understanding the complexity of recommendation within a synthesis task, and specifically the role of confidence estimates, by illustrating the impact a minimum confidence threshold has in balancing between the recommendation and satisfaction rates. Ultimately, the only importance of the confidence threshold is in balancing these desired properties, so this chart lays bare the range of values for these properties achievable by the system.

A similar plot was developed by Frietag in his work in text mining [31]. Text, such as a product description, is mined for potential fields, such as price, and their accompanying values, such as $3.50. The former is analogous to the problem components of Phala's recommendations, and the latter is analogous to the solution component.

5.2.3 Presenting One Recommendation at a Time

Users of product recommendation software are often presented with a collection of recommendations, rather than single recommendations one at a time. This fits well with the user's role of browsing in a scenario such as shopping. However, in a synthesis task, the user has a different role, and more directed recommendations may be preferred. As mentioned, users are unlikely to pore through myriad irrelevant or inaccurate recommendations.

To address this issue, Phala presents recommendations one at a time, ordered by confidence. This provides a new perspective on analysis of the system, since only the most highly ranked recommendations are consumed. Specifically, the only irrelevant and incorrect recommendations that would be considered would be those with higher confidence values than the first recommendations that satisfy the user's needs. All other recommendations would be unused, regardless of their relevance or correctness. Two more metrics are presented to explore this aspect of Phala's performance.

5.2.3.1 Initial Relevance Rate

The *initial relevance rate* is similar to the recommendation rate metric; however, only the recommendation with maximum confidence is considered. Additionally, since recommendations are issued by problem, not by opportunity, this metric will track the proportion of tests for which the highest ranked recommendation was relevant. That is, a recommender could receive a 100% score if the top rated recommendation for each test matched at least one of the problem components of an expected recommendation.

5.2.3.2 Initial Accuracy Rate

The *initial accuracy rate* is similar to the accuracy Rate metric but differs in the same way that the initial relevance rate differs from the relevance rate.

5.2.4 Determining Optimal Preference

The previous two subsections outline how user preferences for a recommendation quality threshold and interaction through examining one recommendation at a time can impact the value of the assistance offered by a system such as Phala. This preference has been left to the user at this point since different users will have differing cost functions for independently solving problems, solving problems with the assistance of relevant and accurate recommendations, or recovering from inaccurate or irrelevant recommendations. In fact, this is the reason helpfulness is difficult to measure, since helpfulness itself could be derived from these unknown functions.

This section will consider that, though these functions can't be known, and are not even constant between users, they can be bounded and sampled. Were these costs functions fixed, it would be possible to calculate the optimal quality threshold for the minimum cost for assistance over a set of test data. Exploring the domain of these cost functions for users then shows the true benefit of a system like Phala by laying bare the conditions under which using Phala improves the user experience.

5.2.4.1 Ideal Performance Ratio

Considering the cost of independently solving a problem as i, the cost of solving a problem with the aid of a recommendation as h, and the cost of recovering from an

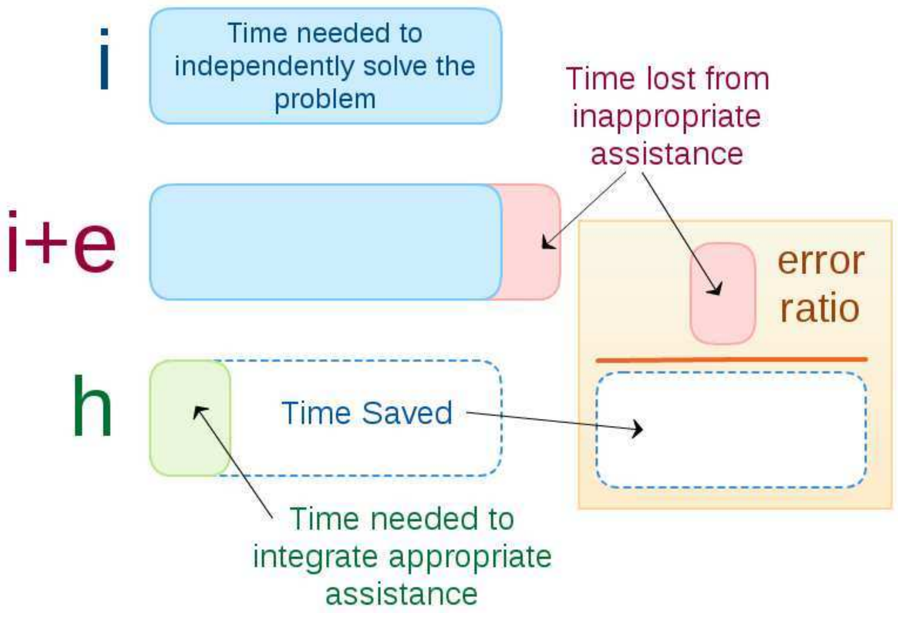

i
Time needed to independently solve the problem
Time lost from inappropriate assistance
i+e
error ratio
h
Time Saved
Time needed to integrate appropriate assistance

Figure 5.2: Measurements used to determine ideal performance

inaccurate or irrelevant recommendation as e, we can calculate the relative benefit of using Phala's top-rated recommendation in terms of these variables. These three values are diagrammed in figure 5.2.4.1. The value of h should be less than that of i, otherwise there would not be a viable option for user assistance via recommendation. Thus, the quantity $i - h$ represents the amount of time saved when an accurate recommendation is issued by the system (this will be referred to as *ideal time saved*).

If p_g is the proportion of episodes in which the user was presented with an accurate recommendation, p_b is the proportion of episodes in which the user was presented with an irrelevant or inaccurate recommendation, and $p_\emptyset$ is the proportion of episodes in which the user was not presented with a recommendation, where $p_g + p_b + p_\emptyset = 1$, $p_g \geq 0$, $p_b \geq 0$, and $p_\emptyset \geq 0$, then $p_g h + p_b(i+e) + p_\emptyset i$ would be the expected cost of using the system, as opposed to the cost of working independently, which is i. If the cost of using the system is less than i, then the system is improving the user experience by decreasing the time needed to complete the task. Thus, $i - p_g h - p_b(i + e) - p_\emptyset i$ is the expected time saved by using the system, which simplifies to $p_g(i - h) - p_b e$.

Rather than analysing this metric with three independent variables, these three variables can be simplified to two quantities, one of which can be graphed as an independent variable and the other of which is a scaling factor ($i - h$, ideal time saved). All three of the original variables can be combined to form a new independent variable in the following manner: Let $f = \frac{e}{i-h}$. This will be called the *error ratio* which represents the ratio of the time taken up by recovery from irrelevant or inaccurate recommendations vs the time gained from accurate recommendations. Using f, the expected time saved from using the system can be re-written as $(p_g - p_b f)(i - h)$; thus, it is written in terms of ideal time saved. The formula $p_g - p_b f$ denotes what will be referred to as the *ideal performance ratio*.

Now that the system can be measured in terms of its benefit to the user, if the cost functions for a user are known, then the settings for the minimum confidence threshold and relevance vs accuracy can be directly calculated. In this plot, they are set in such a way that the benefit factor is maximized, thus representing the true potential for user assistance.

The ideal performance ratio plot explores the error ratio as the independent variable with the benefit factor as the plotted dependent variable. Positive values indicate conditions under which the system provides benefit, whereas a value of zero indicates areas in which the system does not help. Note that it is impossible to reach negative values with the ideal performance ratio, owing to the fact that it is maximized in terms of the selection of a minimum confidence threshold. This threshold could be set so high as to not allow a single recommendation, resulting in an ideal performance ratio value of zero. Since zero is always a possible value, no value lower than zero will ever be the maximum value of the ideal performance ratio.

The ideal performance ratio depends on the ideal time saved, but it identifies the degree to which this ideal is reached in terms of a proportion of the ideal time saved. Thus, positive values represent possible error ratios for which the system will successfully aid the user in his or her task. The degree to which this is true cannot be determined without knowing the cost functions identified, but the degree to which an ideal benefit can be reached is recognized by values approaching 1 for the ideal performance ratio.

5.3 Evaluating Recommendation Extraction

To assess Phala's performance and the potential to exploit available provenance data for case-based support, the evaluation should seek to answer three questions:

1. How often will Phala produce recommendations that aid the user?
2. How well can Phala cater its recommendations to user preferences?
3. How scalable are Phala's methods?

As noted in the prior subsection, usefulness can take on several meanings. Beyond meeting these performance metrics individually, it's important that Phala also be able to maximize those most important to the user. Scalability has been addressed in Chapter 4, though the other criteria will be evaluated by the tests below.

5.3.1 Forming a Test

Phala was tested using cross-validation. This involves reserving a portion of the cases in Phala's case-base to use for forming queries that Phala may employ the remainder of its case-base to solve. Specifically, a variant called *leave-one-out testing* was used. This is an iterative process in which each case in the case base is removed, one at a time, in order to generate a set of tests that are performed in that case's absence. The total number of tests is then a factor of the size of the case-base.

The rest of this section will describe the different types of leave-one-out tests which were produced. For each test description, the term *test case* will refer to the case removed from the case-base for a round of leave-one-out testing.

5.3.1.1 Directed vs Undirected Tests

Directed tests are the most simple of the tests. In a directed recommendation test, information about the location within a workflow of where the user would like to see a recommendation is provided to the recommender. The only remaining job for the recommender is then to determine what the proper remediation is. Undirected tests differ in that no problem context is provided to the recommender, and it must then determine where likely problems are within the query. These types of tests are more difficult for the recommender owing to the added responsibilities and added opportunity for incorrect recommendations, but they also provide greater assistance to the user by alleviating additional burden. For these reasons, undirected tests are the focus of the evaluation in this chapter.

5.3.1.2 Generative Tests

In a generative recommendation test, the recommender is expected to recommend adding a component to a partially completed workflow. This may consist of adding a new processor or an edge between existing processors. Such tests are formed by removing an edge or processor from the test case and presenting the resultant structure as a query. In the case of a directed test, the processors incident to the removed component would also be identified to the recommender. Passing the test involves recommending an edit to the query that returns it to the original state of the test case. Recommendations must identify how to return the query to the original state of the test case, including where in the structure the change needs to be made in the case of an undirected test.

Another generative test that will be used is what is called a construction test. This type of test is intended to model the process of authoring a workflow. All elements of the test case are removed, leaving only the inputs and outputs of the workflow. This is presented as a query, and recommendations that restore all of the processors linked to these inputs and outputs are expected. Then another test is performed in which processors previously linked to the newly added processors are expected. This process continues until the query is restored to the original state of the test case. This is the most difficult test for Phala's extraction method of recommendation, since the extraction method depends on sufficient detail in the query to find a strong match within the case-base. Recommending earlier in the process is more difficult because little of this information is available.

5.3.1.3 Restraint Tests

Recommenders are designed to solve problems, but an issue often encountered is that there isn't always a problem to solve! In these cases, a well designed intelligent assistant should identify that there isn't need for a recommendation and withhold any from the user. Since this is, in a sense, contrary to the goal of a recommender, it can be a difficult problem to approach for an intelligent assistant. The restraint test simply presents the test case, unaltered, as the query. Passing the test involves restraining from making any edit recommendations.

5.3.1.4 Validation Test

An important use-case for a recommender to assist scientific workflow authors is refinement and validation. This use-case has yet to be addressed by the prior tests other than the restraint test which tests for the absence of corrective recommendations as well as the absence of extension recommendations, thus confirming that a workflow is valid. To address this use-case, error correction tests will be performed. To form such a test, a random error is introduced to the workflow from the following list:

- The service for a processor is replaced with a random service.
- An extraneous edge is added to the workflow.

A correct response will include deletion or replacement recommendations for the affected portions of the workflow.

5.3.2 Provenance Data Sources

Data sources for mining provenance for users of Phala would come from a provenance management system, such as karma, capable of exporting provenance records in the OPM format. Cheah et al. have developed a large, 10 GB database of synthetic provenance, derived from the synthetic simulation of six different workflows [21]. Such a database can be important for evaluating systems ingesting provenance, particularly when studying scale.

However, as a test data set for this dissertation, cases have been extracted from a different source: workflows from the myExperiment website. As noted in section 2.7.2, workflows are not the same type of data as provenance, and there are important differences to keep in mind. Specifically, provenance traces may contain unintended gaps which wouldn't otherwise exist in a published workflow. Additionally, workflows may be run multiple times, and although each successive execution may legitimately produce differing provenance traces, a case-base mined from provenance traces will likely contain more repitition than a case-base mined from published worklfows. However, asside from these differences, the data flow traces mined from the Taverna-based myExperiment workflows, specifically, are likely to be very similar to the data flow traces mined from resulting provenance traces due to the tight coupling of provenance and workflows in Taverna [78], and the fact that these workflows contain very little control structure (see Section 2.4.0.4) which is entirely unused by Phala.

The myExperiment dataset includes over 2000 workflows – 349 of which, all Taverna-based, were mined for use in evaluating Phala. This dataset was chosen due to the fact that, as of this writing, and to the best of the author's knowledge, the myExperiment collection is the largest and most diverse collection of scientific workflows publicly available on the Internet.

5.4 Adapting Recommendations Formulated from Extraction in Phala

This section will explore how the HRF framework can be applied to improve performance in Phala and some of the design decisions inherent in its application.

5.4.1 The Phala Workflow Authorship Assistant Revisited

When a user builds a workflow, Phala uses the stored cases to incrementally recommend additions to the workflow based on choices reflected in previous workflows. We referred to this process in Chapter 3 as case-based recommendation extraction.

In addition to generating recommendations from past cases, early work on Phala also generated recommendations based on a baseline recommender, *Link Frequency*, introduced in section 5.4.2.1. These methods were combined into a hybrid method that outperformed each method individually [60]. The success of this combination method suggested investigating a more general combination approach, which led to the research on hybridization presented in this chapter.

5.4.2 HPARs

The success of integrating the baseline led to examining other components that could be added to the hybrid framework to further improve results. Each of these components comprises a somewhat complex rule, most of which are based on statistics that must be mined from the case base. These rules are modelled by Java programs and are encapsulated within the system as components referred to as Highly Procedural Adaptation Rules (HPARs). These rules embody procedural knowledge augmented onto the case-based reasoner of varying domain independence, but they do not embody specific and highly domain-dependent ontological knowledge. These HPARs add a degree of domain sensitivity without the need for a hand-crafted ontology, instead relying on case-mining to produce the requisite domain-dependent knowledge. Each HPAR used in Phala is fairly general and could potentially be used in domains other than scientific workflow authorship.

Phala employs four HPARs, described in the following subsections.

5.4.2.1 Link Frequency

The first HPAR will be called *Link Frequency*: a method of recommending the addition of new processors, which does not rely on the structure of individual cases. This method will only make recommendations after examining the problem context of an existing recommendation. Therefore, without other recommenders, such as extraction from cases, this HPAR will not make recommendations.

For recommendation of new services, this method identifies the service to which the given node's service is most often linked and recommends the node to a new processor instance of that service. In the case of ties, one of them is selected at random. This method is similar to that proposed by Xiang et al. [121], which was briefly discussed in Section 2.5.2.3.

5.4.2.2 Input Port Usage

This HPAR makes recommendations by examining the expected input port utilization of each processor by observing what named input ports were used for other processors using the same service. The frequency with which each port is used is considered in calculating the confidence for any recommendations generated.

If a port is frequently used in other processors with the same service, a problem context recommendation will be generated recommending that a new processor or edge be linked with the service using the specified input port. Output ports are not examined by this HPAR since they are much less predictive of a potentially relevant problem context.

5.4.2.3 Input Profile Matching

This HPAR is similar to Input Port Usage; however, instead of examining each input port individually, this HPAR examines the collection of input ports used as a whole (called the input profile) and compares it to input profiles of other processors with the same service. If the currently examined processor's input profile is a strict subset of most other profiles, a problem context recommendation is produced recommending that an additional service or edge be linked to this processor using the specified port. Confidence is calculated by the proportion of input profiles the currently examined processor's input profile is a strict subset of.

Additionally, if the processor's input profile doesn't match any of those of other instances of the processor, a problem context recommendation to replace the service with a different service is produced.

5.4.2.4 Flow Analysis

This HPAR helps detect errors in the edges connecting nodes by searching for unexpected loops in the data flow of the workflow. Recommendations for removing any such edges are generated by this HPAR. This rule does not rely on the case-base at all and was added to complement the extraction recommender by identifying some basic errors that might be missed by the case-based method.

5.4.3 Confidence Regression

One solution to the problem of confidence comparability is to view confidence as a probability estimate. Frietag adopts such a strategy aiming to estimate the probability that a particular text fragment represents a field [31]. In order to produce these estimated probabilities, Frietag produces transformations for the confidence values, mapping them to probabilities by producing a regression model from a set of mapped data points. Phala applies a similar approach, using an estimated probability that a recommendation will be correct or the probability that a contra-recommendation will be incorrect.

5.5 Evaluating Phala

5.5.1 Incorporating a Baseline

The goal of producing such a hybrid system was to improve the performance of Phala through adaptation of the recommendations it generated through extraction, while maintaining its knowledge-light properties. Thus, HPARs may make a good base-line

Test	Performance Metric	Bound Type	Bound
Service Addition Test	Recommendation Rate	Upper Bound	77.9
Service Addition Test	Satisfaction Rate	Upper Bound	81.2
Service Addition Test	Satisfaction Rate	Lower Bound	16.8
Edge Addition Test	Recommendation Rate	Upper Bound	61.12
Construction Test	Recommendation Rate	Upper Bound	69.89
Construction Test	Satisfaction Rate	Upper Bound	81.06
Construction Test	Satisfaction Rate	Lower Bound	0.06
Edge Removal Test	Recommendation Rate	Upper Bound	76.02

Table 5.1: Ideal limits of knowledge-light assistance in the myExperiment dataset.

for comparison, particularly the Link Frequency recommender, since it produces full recommendations rather than simply problem context recommendations. However, Link Frequency cannot recommend problems, so each of the other problem context recommending HPARs was grouped with Link Frequency to produce the *link frequency* baseline appearing in the evaluation results later in this section.

5.5.1.1 Limits of Knowledge in the Case-Base

While discussing potential baselines for evaluating a knowledge-light recommender, it is important to know the limits within which any recommender can perform. It is possible to provide an upper and lower bound on the possible performance of a knowledge-light reasoner over a given dataset. For example, consider a service recommendation test is presented to Phala, and service A within that test must be linked to a new service B. An obvious limit on the maximum response rate is whether service A exists or not within the case base. Table 5.1 summarizes various ideal limits of knowledge-light assistance for the myExperiment dataset. Limits on satisfaction assume that the recommender has reached the maximum recommendation rate. Limits not listed are either impossible to determine, given the nature of the test, or are already 0% or 100% for lower and upper bounds, respectively.

5.5.2 Experimental Setup

The dataflow within 339 workflows from myExperiment was mined to form simulated provenance traces to seed the case-base. Leave-one-out testing was then employed with each of the tests described in section 5.3, generating each of the plots described in section 5.2. In addition to cases available for retrieval, each of the case-base mining methods in the HPARs as well as the regression method are restricted from using data from the case selected as the test case in each test.

In the service addition and construction tests, the Link Frequency base-line was incorporated but was not included with the other tests since Link Frequency only produces service addition recommendations. In every test but the restraint test, extraction and Phala as a whole are also both graphed. In the restraint test, the number of recommendations produced by the entire Phala system was graphed according to the minimum confidence threshold and compared to the number of recommendations produced in the service addition test.

Each test examines a particular purpose for using Phala without mixing tasks. This was intentional in the modelling of the tests owing to the fact that the degree to which a user will encounter generative vs validation tasks will differ from user to user.

Confidence values in Phala for error correcting recommendations (removing edges and replacing services) tend to be lower than confidence values for construction (adding services or edges), causing Phala to perform poorly on validation tests. In response, Phala can be set to one of two modes: generation or validation. The user can request that recommendations come from either mode. Service addition, edge addition, and construction tests were performed in generation mode, and edge deletion and service replacement tests were performed in validation mode. It is possible to further divide modes for Phala and further increase performance, but this simple distinction was chosen since it retains most of the benefit in the results and its specification would not be as much of a burden on the user.

5.5.3 Results

The results of the service addition test are indicated in figures 5.3 through 5.11. Figure 5.3 indicates what percentage of the problems were addressed with a relevant recommendation. The recommendations from the extraction component are used by both Phala (the combined recommendations from all sources) and link frequency (combined problem context recommendations from all sources, but solutions only come from the link frequency recommender). Extraction only produces one segment of the recommendations considered by the combiner and by link frequency; there are additional HPARs that produce other problem context recommendations, such as input port usage and input profile matching. Though extraction, at best, only matches just over 40% of the recommendations, along with HPARs, it contributes to the success of Phala as a whole, which matches just under 45% of the problem

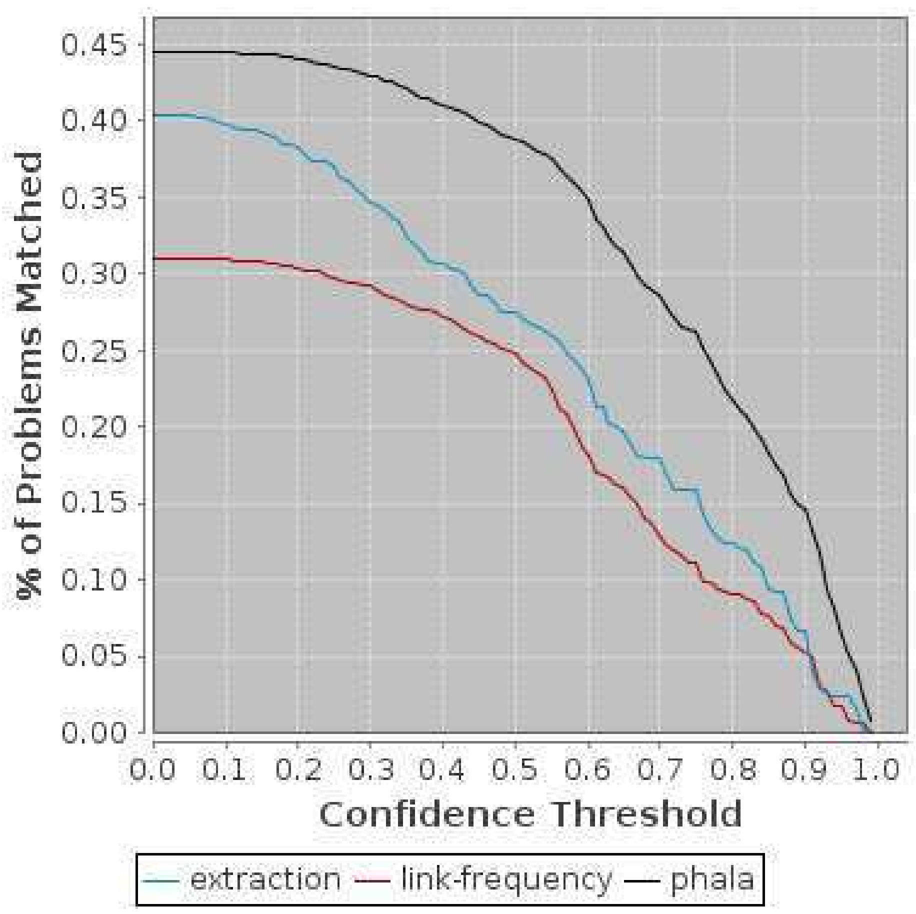


Figure 5.3: A plot of the recommendation rate for the service addition test.

components of the expected recommendations.

> **Key Result:** Phala covers the union of the expected recommendations covered by its components, which results in a greater coverage than any one of its components.

Figure 5.4 depicts the ratio of expected recommendations whose solution components have been matched by a recommendation to the recommendation rate at each minimum confidence threshold. For extraction, this begins just under 80% and climbs as the similarity of the cases used increases to just over 95%. Link frequency

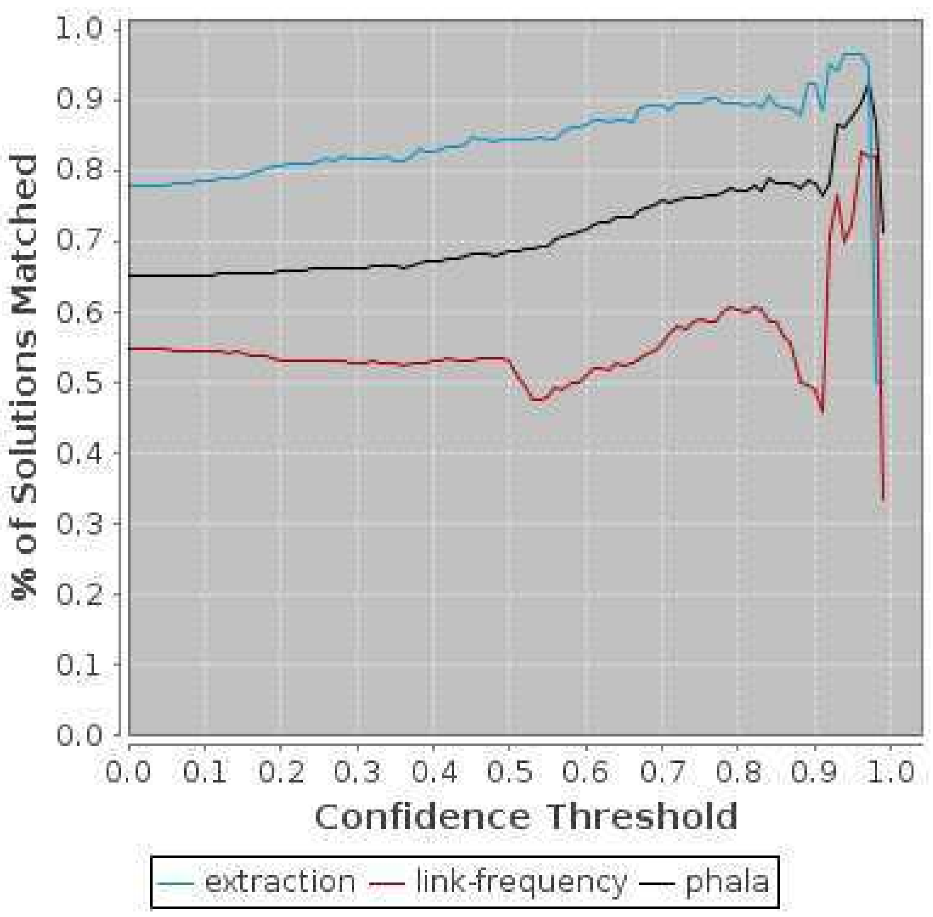


Figure 5.4: A plot of the satisfaction rate for the service addition test.

performs significantly worse[1], beginning just under 55% and reaching its ceiling under 85%. Phala's satisfaction rate is significantly lower than extraction but significantly higher than link frequency since its recommendation rate is much higher, largely because it incorporates recommendations from both of these sources. Figure 5.5 depicts satisfaction reached when also reaching a particular recommendation value, integrating results from each of the previous two plots. In this case, Phala again appears in-between the other two components. This doesn't necessarily have to be the case,

[1]Each significance test used in this section is a Z test with $p < 0.01$.

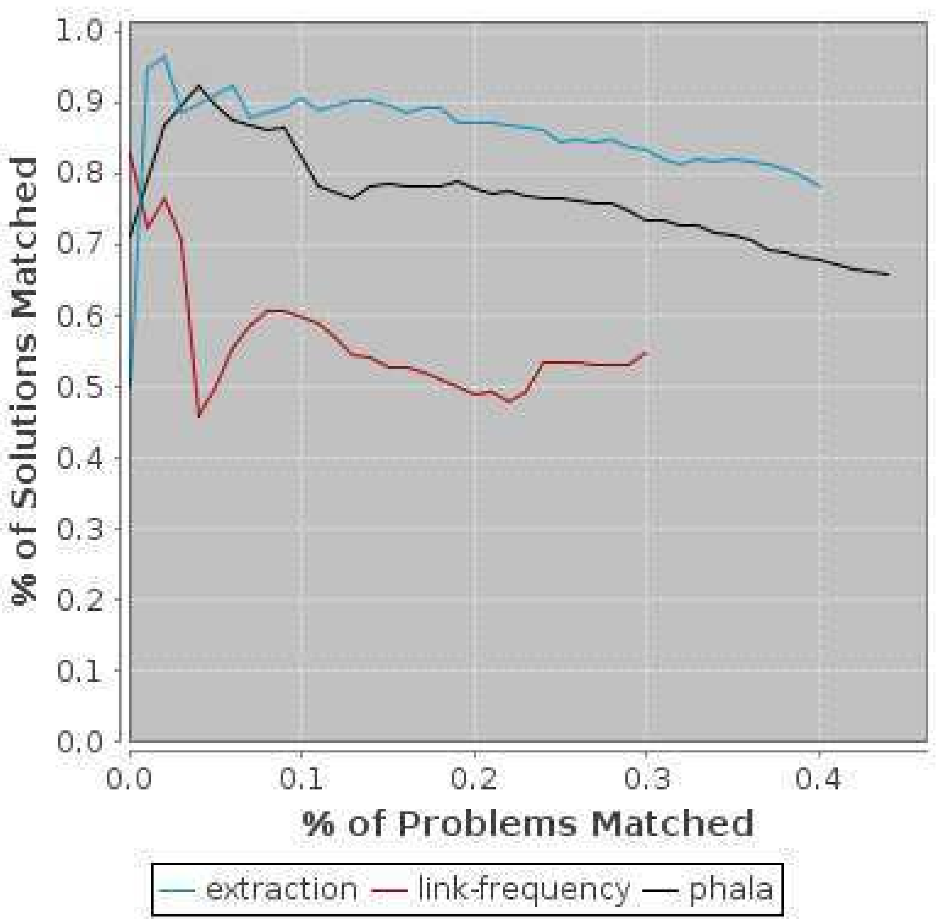


Figure 5.5: A plot of the satisfaction vs recommendation rates for the service addition test.

however. Phala includes HPARs that degrade its performance overall but provide a more important benefit identified later in this section.

> **Key Result:** When considering all recommendations produced, the case-based extraction component produces the best over-all recommendations.

The metrics mentioned thus far relate to the percentage of expected recommendations that are matched in some way. Figures 5.6 and 5.7 depict the relevance and

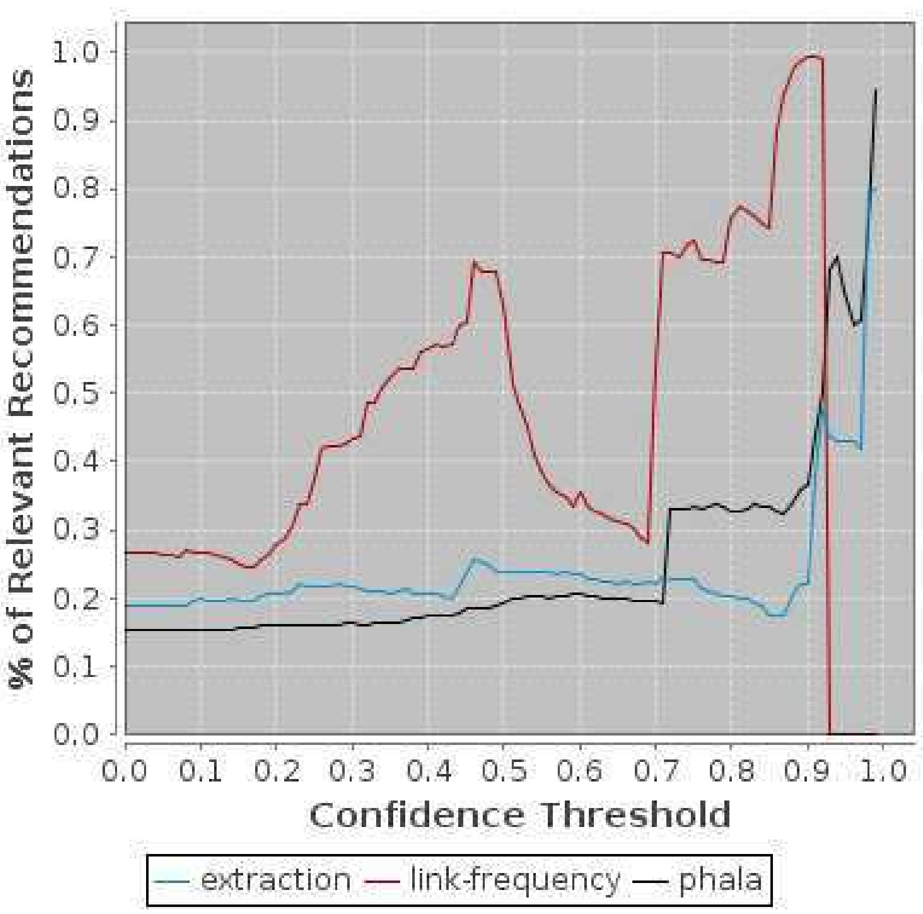


Figure 5.6: A plot of the relevance rate for the service addition test.

accuracy rates for the entire set of generated recommendations. Phala's numbers are fairly low for the relevance rate, compared to the others, for the same reason that Phala's recommendation rate was high: Phala includes all relevant as well as irrelevant recommendations from all methods, and the relevant recommendations Phala includes overlap more so than the irrelevant recommendations. Irrelevant recommendations were not considered in Figure 5.3.

Link frequency provides the best relevance since its recommendations are generated from problem context recommendations produced by HPARs designed to boost

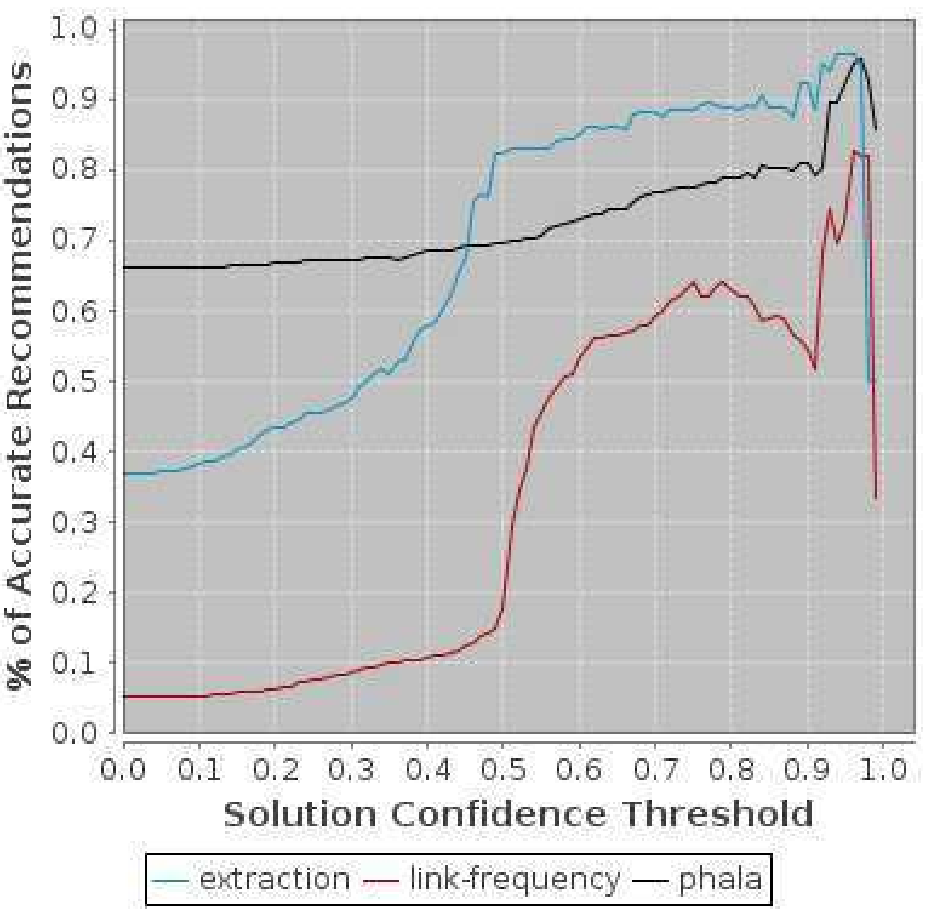


Figure 5.7: A plot of the accuracy rate for the service addition test.

the coverage of Phala. The two spikes in the relevance rate for link frequency coincide with spikes in the input profile matching and input port usage HPARs, respectively. These spikes can be seen to a lesser extent within the Phala plot as well, but the combined approach utilized by Phala mostly flattens them out. Though Phala's combination procedure invites many irrelevant recommendations, it also removes many inaccurate recommendations, resulting in Phala keeping a relatively high accuracy rate, which is either dominant or dominated by extraction at different intervals in figure 5.7. Of note, when using a minimum confidence threshold (thus considering all recommendations), Phala provides significantly more accurate recommendations than either of the others.

> **Key Result:** Overall, HPARs produce recommendations that are more often relevant, while recommendations produced through extraction can be the most accurate, and for other confidence thresholds, Phala produces the most accurate recommendations.

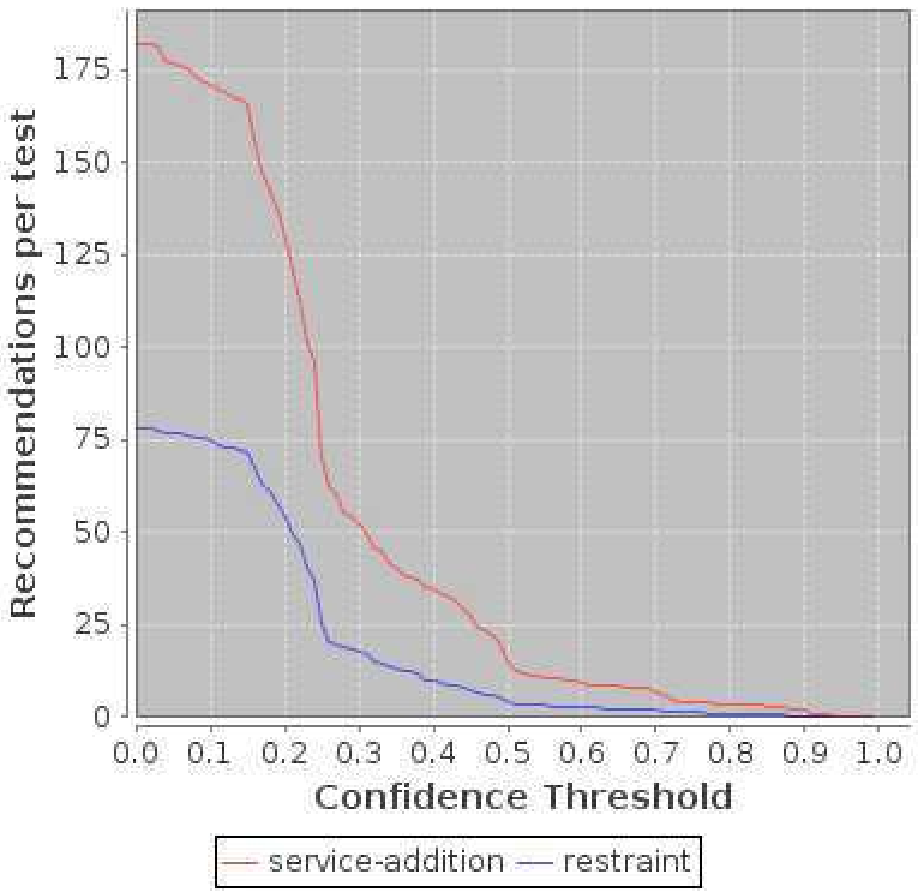


Figure 5.8: A plot of the number of recommendations produced for service addition and restraint tests.

Figure 5.8 plots the number of recommendations produced by Phala during the service addition test against the number of recommendations produced during a restraint test, in which a complete workflow is presented to Phala with no expected recommendations. Phala produces less than half the number of recommendations during the restraint test, showing that the components of Phala are significantly less likely to make recommendations in the cases where all recommendations are irrelevant. Although Phala still produces a large number of recommendations in these cases, the number of recommendations produced may be a basis for a confidence value that indicates to the user the degree to which Phala believes the user's workflow is complete.

> **Key Result:** Phala doesn't stop producing recommendations when none are expected, but the number of recommendations produced by Phala is a good indicator of the likely completeness of the query workflow.

The metrics, to this point, examine the entire collection of recommendations, though as noted in section 3, this is not an ideal means of evaluating an intelligent assistant that ranks its recommendations by a confidence value. Figures 5.9 and 5.10 depict the relevance and accuracy of the top recommendation for each test for each recommender. In both of these plots, the user preference for relevance vs accuracy is set to 0.5, so that both are considered equally important. In figure 5.9, the value of Phala's combiner in filtering out the best recommendations is seen. There is no significant difference between the relevance of the first recommendation produced by Phala and the first produced by Phala's HPARs, though both are significantly more relevant than the first recommendation produced by extraction.

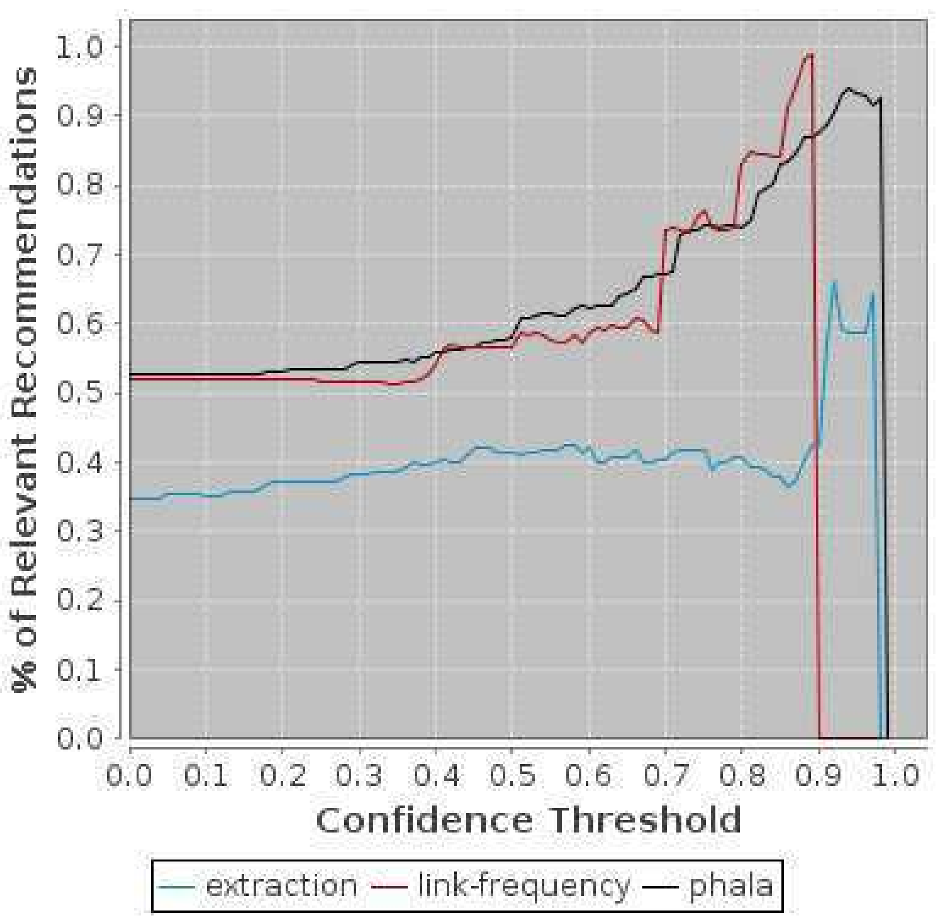


Figure 5.9: A plot of the initial relevance rate for the service addition test.

In figure 5.10, the accuracy of Phala's initial recommendations is shown to be lower than that of extraction and above that of link frequency. The fact that extraction provides accurate recommendations is the reason most of Phala's HPARs focus on boosting the relevance of Phala's recommendations, as evidenced in these two figures.

Key Result: Top recommendations produced from extraction tend to be more accurate, whereas top recommendations produced from the HPARs tend to be more relevant. Phala is able to combine recommendations from each of these sources to provide a more balanced set of final

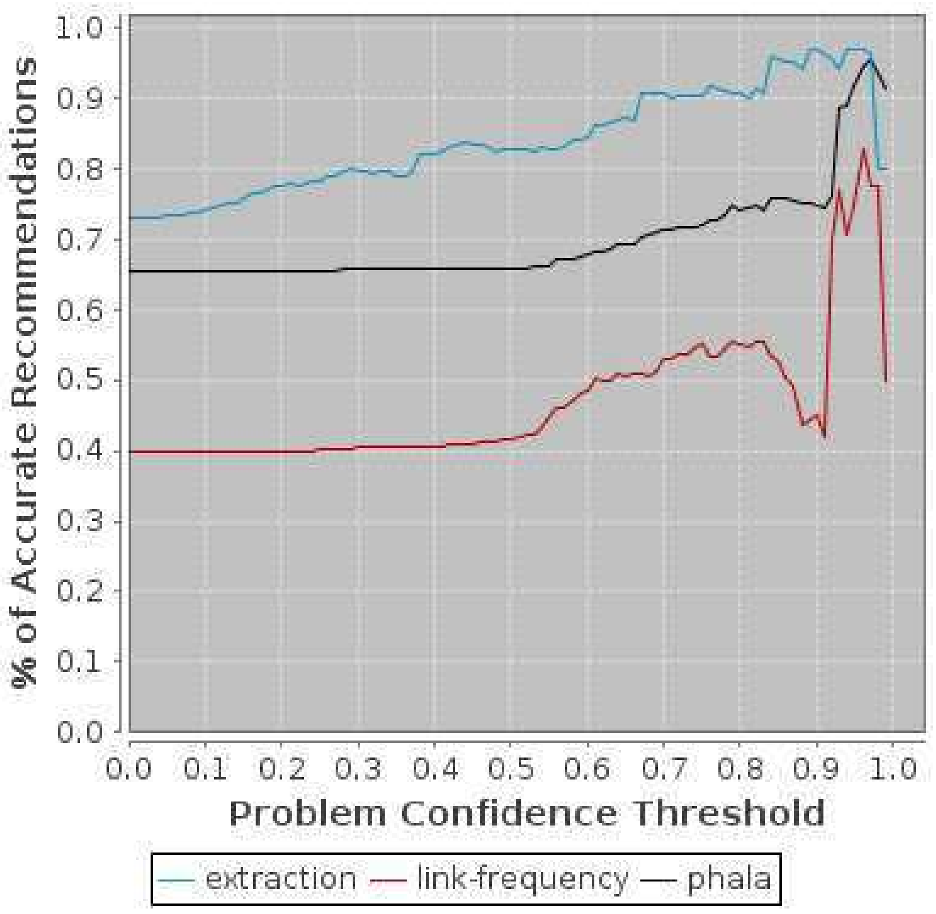


Figure 5.10: A plot of the initial accuracy rate for the service addition test.

recommendations.

Though the utility of Phala may not be entirely clear from the previous two graphs (Phala was out-performed by one of the base-lines in each of the graphs), when considering the impact of these metrics on a user, and also considering the user's choice in optimal confidence thresholds and relevance vs accuracy trade-offs, Phala is consistently able to save the user significantly more time. Figure 5.11 depicts the percentage of the ideal time an intelligent assistant could save the user, in terms of the error ratio (explained in Section 5.2.4). For each recommender's plot, at each point

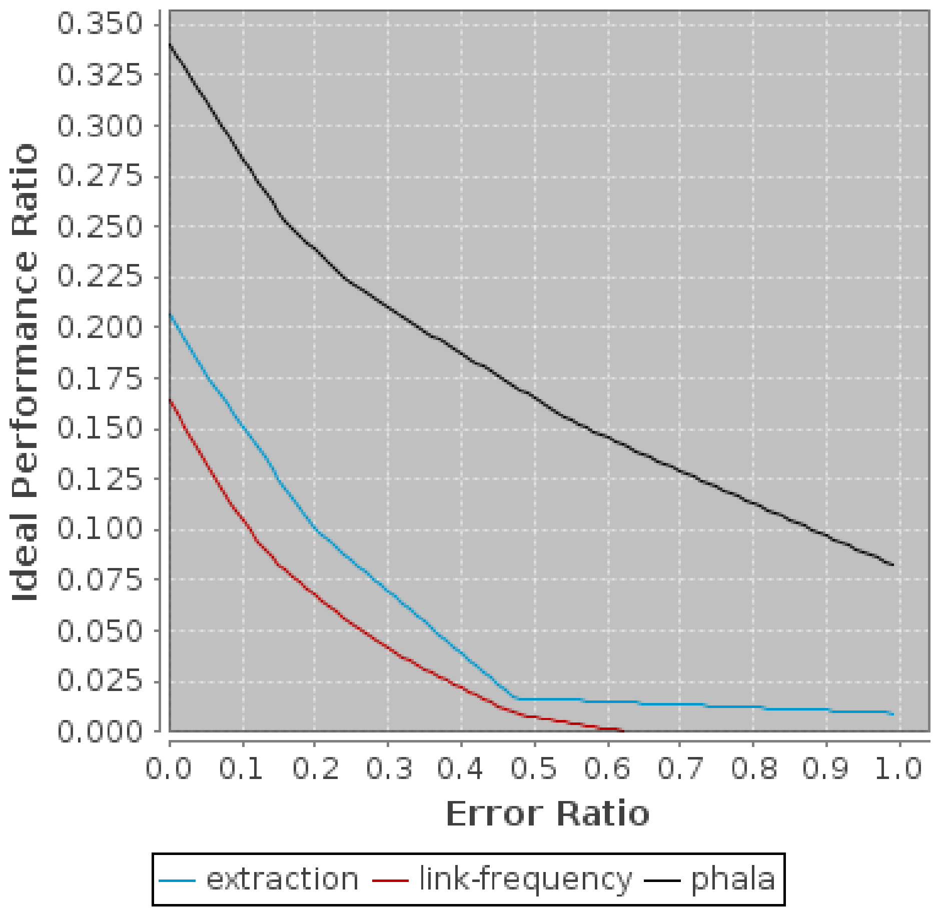


Figure 5.11: A plot of the ideal performance ratio for the service addition test.

where a joint can be seen in this graph, the optimal settings for that recommender changed. In order to provide a benefit under a wide space of user cost functions, one of many different configurations for the recommenders must be used. Phala can provide more than a third of the ideal benefit for a recommender system when the error ratio approaches zero, and continues to provide a high level of benefit, several times that of the base-lines at times, all the way beyond a reasonable value for the error ratio in a system where users are in a position to benefit from recommendation-based assistance (at error ratio 1.0, the time it takes to recover from a bad recommendation is equal to the maximum time saved by incorporating a good recommendation).

Key Result: In the test for adding new services to a mostly-complete workflow, Phala provides significantly more benefit than the baselines and provides benefits to the user under a wide set of circumstances.

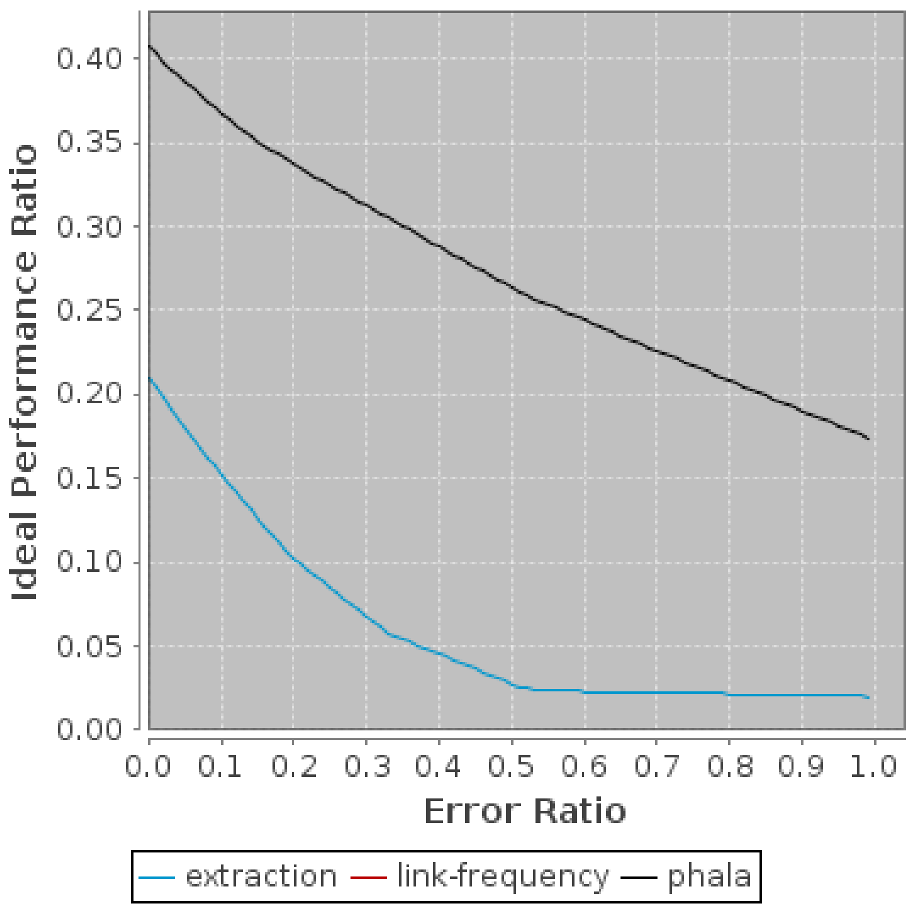


Figure 5.12: A plot of the benefit factor for the edge addition test.

The ideal performance ratio plot is depicted for each of the other tests in Figures 5.12, 5.13, 5.14, and 5.15. Phala continues to provide a large advantage over baseline methods in the edge addition test depicted in Figure 5.12, which would be expected

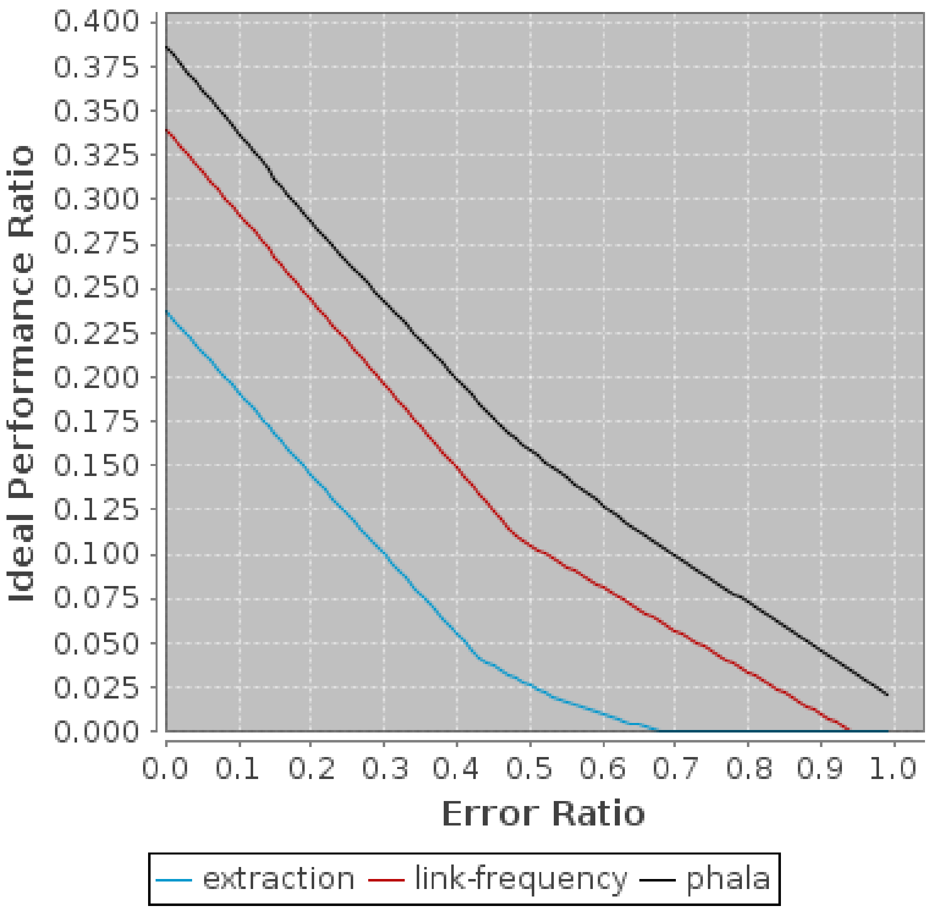


Figure 5.13: A plot of the ideal perforamce ratio for the construction test.

from the similarity of this test to the service addition test. The construction test depicted in Figure 5.13 is more difficult for a knowledge-light approach since there is little context from which to retrieve a useful case early in constructing a workflow. Phala is still dominant over the baseline methods in this test, but provides less benefit over the HPAR's than in previous tests since many of the HPARs provide recommendations independently of the structure as a whole, unlike the case-based extraction method, and are thus immune to the low amount of detail in many of the queries in these tests.

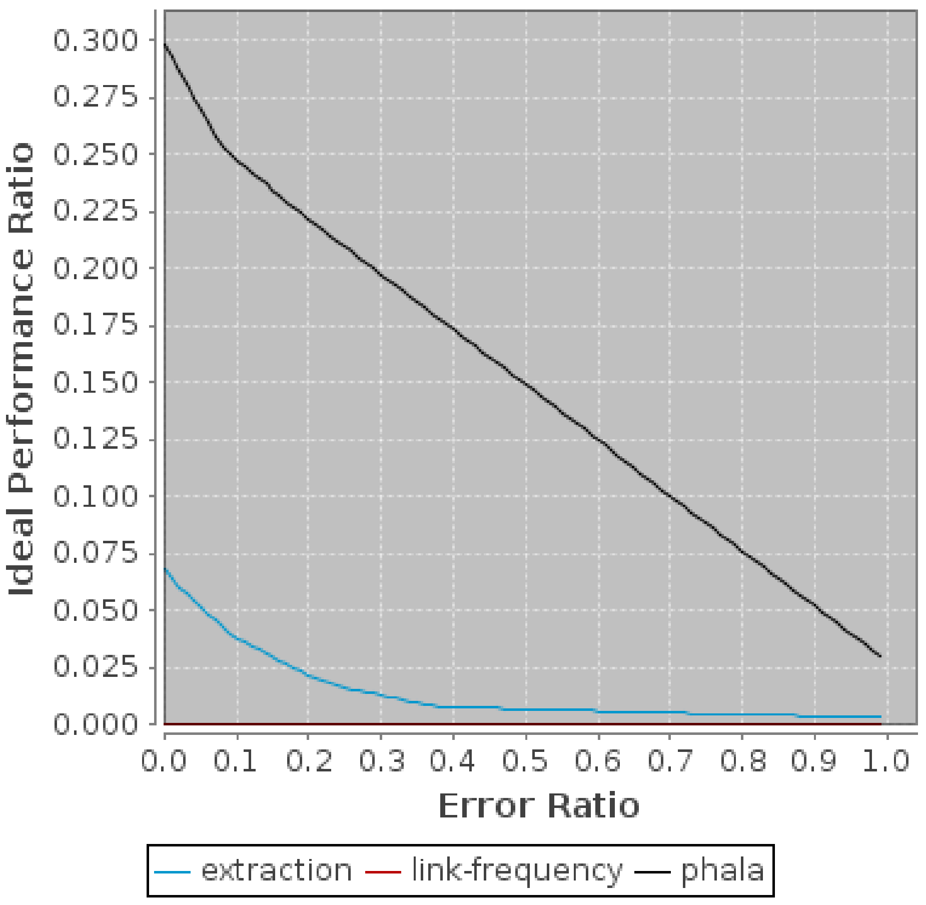


Figure 5.14: A plot of the ideal perforamce ratio for the edge removal test.

Phala provides a substantial benefit over its case-based component in the first of the two validation tests, in which erroneous edges must be detected. Phala also provides benefit in the second test, replacing erroneous services with correct ones, but to a lesser extent.

Though Phala was shown to be somewhat less effective in validation tasks than it was for generative tasks, the positive results are remarkable considering that the work discussed in Chapter 3, which addresses validation, utilizes a knowledge-heavy approach, whereas knowledge-light approaches and those in-between were typically

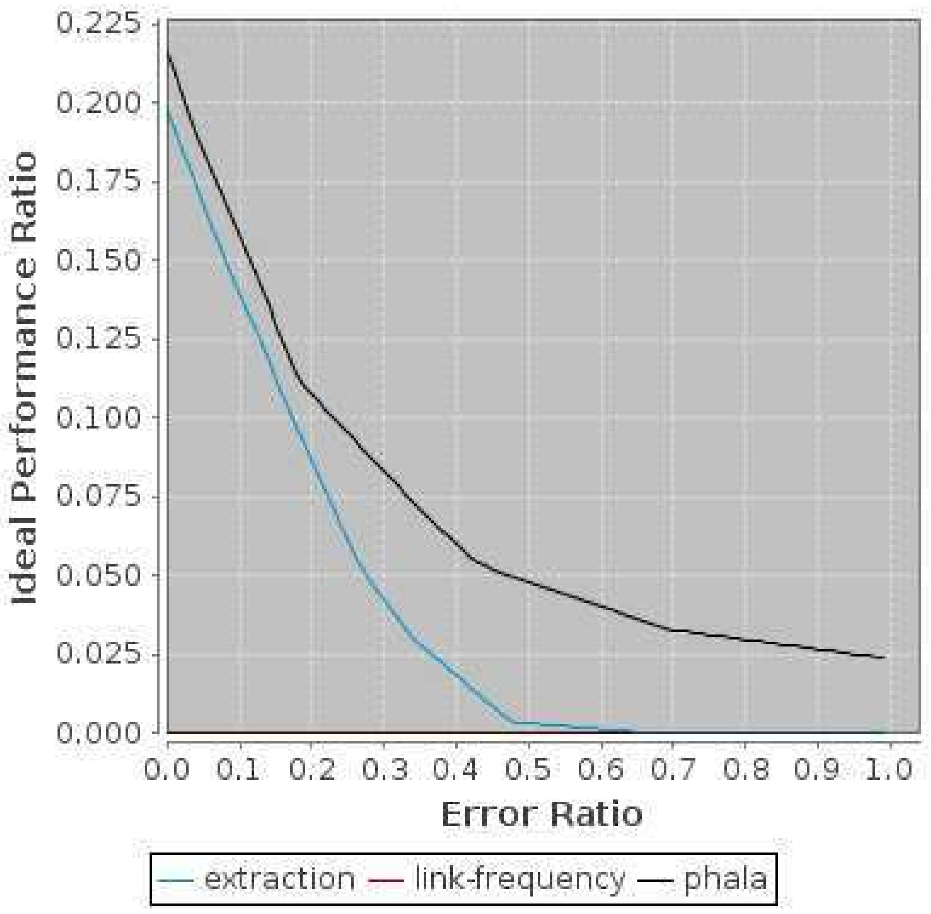


Figure 5.15: A plot of the ideal perforamce ratio for the service replacement test.

used for generative tasks.

> **Key Result:** Phala provides superior benefits in each of a variety of tests, including both validation and generative tests.

Chapter 6

Conclusions

6.1 Claims and Evidence

Work in this dissertation has uncovered and explored a number of key questions and evidence, stemming both from the literature review and the empirical development and evaluation of Phala. This section will summarize the key points argued in this dissertation and review the evidence it brings forth.

6.1.1 Main Thesis

The primary thesis of this dissertation is:

- Recommendation-based assistance for authors of structured data can leverage case-based reasoning to greatly reduce knowledge acquisition, computational resource, and maintenance requirements while maintaining effectiveness.

Phala embodies knowledge-light principles throughout its design, from its knowledge-light approach to mining cases from provenance, through its adaptation procedures that mine the case-base for requisite knowledge for improving solutions generated from the case-based recommendation extraction process.

The case-based recommendation extraction process is shown to be a vital component of the system, contributing largely to its success as it produces strong enough results that allow it to stand alone as a viable source of assistance. Structured gestalt cases mined from provenance provide a rich datasource for generating these recommendations, and graph indexing techniques are able to adequately mitigate the computational costs of using such complex case representations.

The recommendation-based assistance offered by Phala provides a middle-ground between information retrieval systems that require the user to analyse and re-purpose cases and knowledge intensive systems that reduce an expert user's role in the authorship process and additionally require significant initial and continuing investment in knowledge engineering for the creation and maintenance of the systems. Recommendations also make a thorough analysis of the components of a system like Phala possible, further easing the burden of system designers seeking a versatile solution within a rapidly developing domain, such as workflow authorship in e-Science.

6.1.2 Sub-theses

Two sub-theses were additionally specified, which are related to the main thesis. Each of these are restated and justified below.

- Provenance can be mined, without significant domain knowledge, to build a case-base that can support authorship of scientific workflows.

Phala was designed with minimal knowledge of processor types within these provenance traces and is mostly data-driven in its provenance mining process, meeting the goal of utilizing provenance for knowledge-light assistance. The benefit of Phala outlined in Chapter 5 and the specific advantages identified in Section 2.7.4 (despite the challenges also mentioned in this section) justify the value of this knowledge source.

- Ensemble frameworks can be extended to support knowledge-light adaptation within a case-based reasoner.

Phala's usage of HRF for adaptation exemplifies the effectiveness of such frameworks in supporting knowledge-light adaptation. Each of the adaptation methods used by phala are data-driven, much like Phala's case-mining process, furthering the knowledge-light goal. In Chapter 5, Phala is shown not only to provide effective assistance but also to outperform each of its components, were the components to be used to independently assist a user. This justifies the inclusion of the hybrid adaptation scheme as a means of providing more effective assistance through a case-based approach.

6.1.3 Summary of evidence

The first important point to consider about the thesis is its novelty. Chapter 2 surveys existing systems that assist in synthesis tasks and identifies several that employ a case-based approach, recommendation-based approach recommending specific content to be added, or knowledge-light approach, and it contrasts the approach taken by Phala with these systems.

Next is the feasibility of the recommendation-based approach to assistance in synthesis tasks. Chapter 2 begins to lay out how recommendation-based assistance can be realized for synthesis tasks, outlining content requirements and evaluation criteria. Chapter 3 outlines specific content for such recommendations in the synthesis domain of assisting authors of scientific workflows.

Reducing knowledge acquisition requirements and maintaining this knowledge is another key concern for this thesis. The issue of knowledge-light assistance is explored in Chapter 3 and specifically outlined for Phala. Section 3.2 and those following it specifically address case-based reasoning's role in reducing the knowledge acquisition burden. Provenance is identified as a source for automatically mining cases in section 3.3.1, which avoids the need for hand-coding most domain knowledge used by the system. Section 3.5.3 identifies how provenance also affords a means of maintaining knowledge within the case-base by recapturing users' choices about the design of new workflows incorporating Phala's feedback. Additionally, section 3.7.2 explains how adaptation knowledge requirements can also be reduced by employing a hybrid approach to adaptation, thus addressing each of the knowledge acquisition concerns for a system such as Phala.

Chapter 4 covers Phala's approach to managing computational resources through SAI: its domain-independent graph database. This chapter explains how two-phase retrieval can be used by Phala to speed up retrieval of stored cases and how Phala's use of this technique compares with other uses of two-phase retrieval. Experiments with another graph-based dataset and the dataset used by SAI, detailed in section 4.6.3, justify the importance of choosing an indexing strategy that fits the dataset and specifically the decisions made in Phala for the indexing phase of the retrieval strategy by comparing the quality of rankings produced by Phala's indexing strategy between both datasets. Experiments in section 4.6.5 show that Phala's approach scales well to significantly larger databases, beyond those cases mined from the myExperiment dataset.

Chapter 5 completes the specification of Phala's approach and identifies unique and important criteria for evaluating the claim that Phala is an effective system. Phala is tested in several ways mirroring its expected usage, which involves the creation of new elements of a workflow and the validation of existing elements within the workflow to determine whether or not they are error free. Experiments show that in each of these tests, Phala provides valuable assistance to the user through a wide range of potential circumstances.

6.2 Important Lessons

Beyond the theses, several other important lessons emerging from this dissertation work have been identified. They are explained in the following sub-sections.

6.2.1 Two-Phase Retrieval

One of the more important lessons from the work presented in this dissertation is the general importance of two-phase retrieval algorithms in process-oriented case-based reasoning. This subject has garnered the most published and currently submitted work from this dissertation [60, 51, 53] and also the most impact on the field, as others are integrating two-phased retrieval into their work [13].

6.2.2 Dataset Diversity

Phala's extraction technique relies on the existence of stored cases which are similar to the query. This may lead one to suspect that Phala would perform better when using a case-base lacking diversity in a leave-one-out test. However, in Chapter 4, particularly Figure 4.5, it was shown that the cases used with Phala's evaluation are actually quite diverse, yet have local similarity within clusters of similar cases. It is this local similarity which turns out to be more important to Phala's production of useful recommendations, particularly in light of another result from Chapter 4, that a high degree of global similarity between cases actually complicates the retrieval process.

6.2.3 Hybrid Systems

Another lesson is the existence of untapped potential in hybrid systems. As noted in Chapter 3, there has been a lot of attention paid to ensemble systems but not as much to hybrid systems such as Phala, in which components of the hybrid bear little similarity to each other in terms of procedure. Phala's improvement over its component methods identifies the utility of hybrid approaches to multi-faceted problems such as those existing in intelligent assistance tasks.

6.2.4 Evaluating Intelligent Assistants

Lastly, there have been several important lessons in proper evaluation of intelligent assistants in the absence of a direct user study. Evaluation work in this dissertation began with Freitag's work involving plots relating what is referred to in this dissertation as relevance and accuracy. However, Chapter 5 shows that, for intelligent assistants, a proper evaluation involves much more than judgements on individual recommendations, since such judgements do not take into account the fact that few will actually be processed or acted upon by the user. Initial recommendation plots identify how judgements can shift when only the top recommendation is considered, how several user-specified settings have bearing, and how different kinds of recommendations, such as problem context recommendations, can also make an impact. The ideal performance ratio metric additionally shows how these user-specified settings can be dealt with, again without a user study, by reformulating the problem in terms of actual costs and benefits to the user, and maintaining a single independent variable in the process.

6.3 Future Work

6.3.1 New Research Avenues for the Structure Access Interface

Given SAI's ability to support many different case representations and graph indexing and comparison algorithms, SAI can be exploited to develop a comparison of various existing algorithms over various synthetic and real-world datasets to aid in the selection of indexing/comparison strategies. Work is currently under way to expand the statistics SAI tracks about the graphs it stores under the hypothesis that some of these statistics on graph structure may be predictive of which indexing and comparison algorithms are most appropriate for a particular domain.

Work is also currently under way to expand the algorithmic offerings of SAI. A graphgrep-like indexing strategy is currently in development, and a frequent structure mining approach is planned. Beyond indexing algorithms, additional mapping/aliment algorithms can also be explored within the framework.

Work is also planned towards developing representations for new domains of structured data. Supporting the popular AIDS database used by many graph database systems for evaluation is crucial to position SAI as a test-bed for modern graph indexing and retrieval algorithms [123]. Additionally, knowledge structures and ontologies are important domains to examine to further entrench SAI as an important tool in Case-Based Reasoning and other areas of Artificial Intelligence.

6.3.2 New Research Avenues for the Hybrid Recommendation Framework

As mentioned in the lessons section, hybrids such as Phala may have a larger role to play in applied machine learning, particularly in domains for which ensemble techniques have been successful. HRF could be further explored as an option for adaptation for case-based reasoners in domains other than that of Phala. Additionally, HRF could be applied to recommender systems that are not case-based and used to improve the results of an existing system by incorporating it in as a component within a new hybrid system, similar to the design of Phala.

6.3.3 New Research Avenues for Phala

Helpfulness has been mentioned as a particularly difficult quality of recommendations to explore, however Phala may be able to approach this issue. A recommendation's helpfulness could be judged by how much content it is recommending, and there are several opportunities for Phala to make recommendations as an aggregate of several recommendations in Phala's current format. In order to find plausible recommendations of a larger subsequence of processors, SAI could be used to implement common subgraph mining within the provenance database. SAI could also be extended to support multiple views of provenance traces, supported within the OPM framework, which could be used to facilitate these recommendations as well.

To further the research on CBR for e-Science provenance, the next application of Phala will be to explore the problem of repairing incomplete provenance traces within a 10 gigabyte database of provenance traces [21]. This is an important step in insuring the maximum benefit from re-using provenance traces for assisting workflow authors (see Section 2.7.4).

To apply Phala to this task, Phala's recommendation system must be altered to generalize more, as these collections of data will be more redundant than the datasets Phala has been tested with through this dissertation work. SAI will again be used to provide the framework for this system and to aid in the development of its core indexing and similarity assessment algorithms. HRF may be applicable to the problem as well as a means of joining together several independent solutions to again provide a superior approach through combination strategies.

6.3.4 Other Research Avenues

The ideal performance ratio metric provides a useful means of connecting automated evaluations of recommendations to the expected outcome for users of the system in a measurable way, but it is not tied specifically to the unique qualities of Phala as a recommender. Thus, this metric may be useful in evaluating other recommendation systems as well. An important avenue of new research will be to identify existing projects that may benefit from an evaluation using this metric and also to seek opportunities to tie this metric directly to a user study to observe the feasibility of measuring the three cost functions associated with users that this metric relies on.

As discussed in Chapter 2, a great deal of existing work similar to Phala is dedicated to the planning domain. Techniques used in Phala may be more widely applicable to these types of domains and can be particularly useful in the growing sub-field of game AI. In addition to developing plans for game-playing [86], Phala and its components could be applied to developing state machines for specifying the behavior of such game-playing agents and also *non-playable characters* (NPC's) [30].

Bibliography

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

[2] D. Aha, L. Breslow, and H. Munoz-Avila. Conversational case-based reasoning. *Applied Intelligence*, 14:9–32, 2001.

[3] D. W. Aha and D. Wettschereck. Case-based learning: Beyond classification of feature vectors. In *ECML '97: Proceedings of the 9th European Conference on Machine Learning*, pages 329–336, 1997.

[4] G. Allampalli-Nagaraj and I. Bichindaritz. Automatic semantic indexing of medical images using a web ontology language for case-based image retrieval. *Engineering Applications of Artificial Intelligence*, 22(1):18 – 25, 2009.

[5] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, Washington, DC, USA, 2004. IEEE Computer Society.

[6] J. L. Ambite and D. Kapoor. Automatically composing data workflows with relational descriptions and shim services. In *Proceedings of the 6th international The semantic web and the 2nd Asian semantic web conference*, ISWC'07/ASWC'07, pages 15–29, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] P. Andrews, M. Margo, A. Chourasia, and J. Towns. Enabling HPC e-science via integrated grid infrastructure. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pages 156–, Washington, DC, USA, 2006. IEEE Computer Society.

[8] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan. The trident scientific workflow workbench. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 317–318, Washington, DC, USA, 2008. IEEE Computer Society.

[9] P. Baudisch. Proceedings of the CHI 1999 workshop: Interacting with recommender systems, 1999.

[10] E. Bengoetxea. *Inexact Graph Matching Using Estimation of Distribution Algorithms*. PhD thesis, Ecole Nationale Supérieure des Télécommunications, Paris (France), December 2002.

[11] J. Bennett, S. Lanning, and N. Netflix. The netflix prize. 2007.

[12] R. Bergmann and Y. Gil. Retrieval of semantic workfows with knowledge intensive similarity measures. In *Proceedings of the Nineteenth International Conference on Case-Based Reasoning*, page In Press, Berlin, 2011. Springer-Verlag.

[13] R. Bergmann, M. Minor, M. S. Islam, P. Schumacher, and A. Stromer. Scaling similarity-based retrieval of semantic workflows. In *Proceedings of the ICCBR-12 Workshop on Process-Oriented Case-Based Reasoning*, 2012.

[14] J. Blythe. Integrating expectations from different sources to help end users acquire procedural knowledge. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, IJCAI'01, pages 943–949, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[15] S. Bogaerts and D. Leake. Formal and experimental foundations of a new rank quality measure. In *Proceedings of the 9th European conference on Advances in Case-Based Reasoning*, ECCBR '08, pages 74–88, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] K. M. Borgwardt. *Graph Kernels*. PhD thesis, Munich, Germany, 2007.

[17] K. Börner. Structural similarity as guidance in case-based design. In *Selected papers from the First European Workshop on Topics in Case-Based Reasoning*, EWCBR '93, pages 197–208, London, UK, UK, 1994. Springer-Verlag.

[18] A. J. Cañas, D. B. Leake, and A. G. Maguitman. Combining concept mapping with CBR: Towards experience-based support for knowledge modeling. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 286–290. AAAI Press, 2001.

[19] A. J. Cañas, G. Hill, R. Carff, N. Suri, J. Lott, T. Eskridge, G. Gómez, M. Arroyo, and R. Carvajal. CmapTools: A knowledge modeling and sharing environment. In A. J. Cañas, J. D. Novak, and F. González, editors, *Concept*

Maps: Theory, Methodology, Technology. Proceedings of the First International Conference on Concept Mapping, Pamplona, Spain, 2004. Universidad Pública de Navarra.

[20] B. Cao, B. Plale, G. Subramanian, P. Missier, C. A. Goble, and Y. L. Simmhan. Semantically annotated provenance in the life science grid. In J. Freire, P. Missier, and S. S. Sahoo, editors, *SWPM*, volume 526 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[21] Y.-W. Cheah, B. Plale, J. Kendall-Morwick, D. B. Leake, and L. Ramakrishnan. A noisy 10GB provenance database. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops (2)*, volume 100 of *Lecture Notes in Business Information Processing*, pages 370–381. Springer, 2011.

[22] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[23] M. T. Cox and C. Zhang. Planning as mixed-initiative goal manipulation. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, pages 282–291. AAAI Press, 2005.

[24] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, May 2009.

[25] E. Deelman and Y. Gil. Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In *Proceedings of the Second IEEE*

International Conference on e-Science and Grid Computing, E-SCIENCE '06, pages 144–, Washington, DC, USA, 2006. IEEE Computer Society.

[26] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, July 2005.

[27] T. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2000.

[28] K. Droegemeier. Linked Environments for Atmospheric Discovery (LEAD): A Cyberinfrastructure for Mesoscale Meteorology Research and Education. *AGU Fall Meeting Abstracts*, Dec. 2004.

[29] A. Felfernig and M. Zanker. Proceedings of the ECAI 2006 workshop on recommender systems, 2006.

[30] G. Flórez Puga, B. Díaz-Agudo, and P. González-Calero. Experience-based design of behaviors in videogames. In *Proceedings of the 9th European conference on Advances in Case-Based Reasoning*, ECCBR '08, pages 180–194, Berlin, Heidelberg, 2008. Springer-Verlag.

[31] D. Freitag. *Machine Learning for Information Extraction in Informal Domains.* PhD thesis, Carnegie Mellon University, 1998.

[32] J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 20(5):485–496, Apr. 2008.

[33] D. Gentner and K. Forbus. MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 504–509, Chicago, IL, August 1991. Cognitive Science Society.

[34] N. Giannadakis, A. Rowe, M. Ghanem, and Y.-k. Guo. Infogrid: providing information integration for knowledge discovery. *Information Sciences–Informatics and Computer Science: An International Journal - special issue: Knowledge discovery from distributed information sources*, 155:199–226, October 2003.

[35] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, Dec. 2007.

[36] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for pegasus: creating large-scale scientific applications using semantic representations of computational workflows. In *Proceedings of the 19th national conference on Innovative applications of artificial intelligence - Volume 2*, IAAI'07, pages 1767–1774. AAAI Press, 2007.

[37] R. Giugno and D. Shasha. Graphgrep: A fast and universal method for querying graphs. *16th International Conference on Pattern Recognition*, 2:112–115, 2002.

[38] C. A. Goble and D. C. De Roure. myExperiment: social networking for workflow-using e-scientists. In *Proceedings of the 2nd workshop on Workflows*

in support of large-scale science, WORKS '07, pages 1–2, New York, NY, USA, 2007. ACM.

[39] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[40] W.-S. Han, M.-D. Pham, J. Lee, R. Kasperovics, and J. X. Yu. iGraph in action: performance analysis of disk-based graph indexing techniques. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 1241–1242, New York, NY, USA, 2011. ACM.

[41] K. Hanney, M. T. Keane, P. Cunningham, and B. Smyth. What kind of adaptation do CBR systems need?: A review of current practice. In *In Adaptation of Knowledge for Reuse, Proc. 1995 AAAI Fall Symposium.* AAAI, 1995.

[42] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *StorageSS '07: Proceedings of the 2007 ACM workshop on Storage security and survivability*, pages 13–18, New York, NY, USA, 2007. ACM.

[43] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 38–, Washington, DC, USA, 2006. IEEE Computer Society.

[44] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information and System Security*, 22(1):5–53, Jan. 2004.

[45] T. Hey and A. Trefethen. e-science and its implications. *Philosophical Transactions of the Royal Society*, 361(1809):1809–1825, May 2003.

[46] H. Hoang, S. Lee-urban, and H. Muoz-avila. Hierarchical plan representations for encoding strategic game AI. In *In Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05*. AAAI Press, 2005.

[47] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 159–166, New York, NY, USA, 1999. ACM.

[48] B. Iordanov. Hypergraphdb: a generalized graph database. In *Proceedings of the 2010 international conference on Web-age information management*, WAIM'10, pages 25–36, Berlin, Heidelberg, 2010. Springer-Verlag.

[49] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *In ICDE*, pages 566–575, 2007.

[50] J. Kendall-Morwick. Towards an ensemble framework for assisting in synthesis tasks. In H. W. Guesgen and R. C. Murray, editors, *FLAIRS Conference*. AAAI Press, 2010.

[51] J. Kendall-Morwick and D. Leake. A toolkit for representation and retrieval of structured cases. In *Proceedings of the ICCBR-11 Workshop on Process-Oriented Case-Based Reasoning*, 2011.

[52] J. Kendall-Morwick and D. Leake. Facilitating representation and retrieval of structured cases: Principles and toolkit. *Information Systems*, 2012.

[53] J. Kendall-Morwick and D. Leake. On tuning two-phase retrieval for structured cases. In *Proceedings of the ICCBR-12 Workshop on Process-Oriented Case-Based Reasoning*, 2012.

[54] J. Kim and J. Blythe. Supporting plan authoring and analysis. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 109–116, 2003.

[55] J. Kim, M. Spraragen, and Y. Gil. An intelligent assistant for interactive workflow composition. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, pages 125–131, New York, NY, 2004. ACM.

[56] J. H. Kim, W. Suh, and H. Lee. Document-based workflow modeling: a case-based reasoning approach. *Expert Systems with Applications*, 23(2):77–93, 2002.

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

[58] J. Kolodner and D. Leake. A tutorial introduction to case-based reasoning. In D. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 31–65. AAAI Press, Menlo Park, CA, 1996.

[59] D. Leake. ACCEPTER: Evaluating explanations. In R. Schank, C. Riesbeck, and A. Kass, editors, *Inside Case-Based Explanation*, chapter 6, pages 167–206. Lawrence Erlbaum, 1994.

[60] D. Leake and J. Kendall-Morwick. Towards case-based support for e-science workflow generation by mining provenance. In *Proceedings of the 9th European*

conference on Advances in Case-Based Reasoning, ECCBR '08, pages 269–283, Berlin, Heidelberg, 2008. Springer-Verlag.

[61] D. Leake and J. Kendall-Morwick. Four heads are better than one: Combining suggestions for case adaptation. In *ICCBR '09: Proceedings of the 8th International Conference on Case-Based Reasoning*, pages 165–179, 2009.

[62] D. Leake and J. Kendall-Morwick. External provenance, internal provenance, and case-based reasoning. Technical report, Provenance-Aware CBR: Applications to Reasoning, Metareasoning, Maintenance and Explanation, 2010.

[63] D. Lenat, M. Prakash, and M. Shepherd. Cyc: Using common sense knowledge to overcome brittleness and knowledge acquistion bottlenecks. *AI Magazine*, 6(4):65–85, Jan. 1986.

[64] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[65] J. Liu and S. Ram. Who does what: Collaboration patterns in the wikipedia and their impact on data quality. *19th Workshop on Information Technologies and Systems, pp. 175-180, December 2009*, 2009.

[66] T. Madhusudan, J. L. Zhao, and B. Marshall. A case-based reasoning framework for workflow model management. *Data & Knowledge Engineering*, 50(1):87–115, 2004.

[67] R. Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson. Retrieval,

reuse, revision, and retention in CBR. *Knowledge Engineering Review*, 20(3), 2005.

[68] N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 573–582, New York, NY, USA, 2007. ACM.

[69] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:926–932, 1993.

[70] O. Matan. On voting ensembles of classifiers (extended abstract). In *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, pages 84–88, 1996.

[71] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. Communitycommands: command recommendations for software applications. In *UIST '09: Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 193–202, New York, NY, USA, 2009. ACM.

[72] S. A. McIlraith and D. L. Martin. Bringing semantics to web services. *IEEE Intelligent Systems*, 18(1):90–93, Jan. 2003.

[73] S. M. Mcnee. *Meeting user information needs in recommender systems*. PhD thesis, University of Minnesota, Minneapolis, MN, USA, 2006. AAI3230139.

[74] T. Mileman, B. Knight, M. Petridis, K. Preddy, and P. Mejasson. Maintenance of a case-base for the retrieval of rotationally symmetric shapes for the design of metal castings. In *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, EWCBR '00, pages 418–430, London, UK, UK, 2000. Springer-Verlag.

[75] M. Minor, R. Bergmann, S. Grg, and K. Walter. Adaptation of cooking instructions following the workflow paradigm. In C. Marling, editor, *ICCBR 2010 Workshop Proceedings*, 2010.

[76] M. Minor, D. Schmalen, A. Koldehoff, and R. Bergmann. Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. In *Proceedings of the 16th IEEE Internazional Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'07)*, pages 370 – 375. IEEE Computer Society, Los Alamitos, CA, 2007.

[77] M. Minor, A. Tartakovski, D. Schmalen, and R. Bergmann. Agile workflow technology and case-based change reuse for long-term processes. *International Journal of Intelligent Information Technologies*, 2007.

[78] P. Missier and C. Goble. Workflows to open provenance graphs, round-trip. *Future Generation Computing Systems*, 27(6):812–819, June 2011.

[79] A. M. Mulvehill. Building, remembering, and revising force deployment plans. In A. Tate, editor, *Advanced Planning Technology Technological Achievements of the ARPA/ROME Laboratory Planning Initiative*. AAAI, 1996.

[80] K. Myers, P. Berry, J. Blythe, K. Conley, M. Gervasio, D. McGuinness, D. Morley, A. Pfeffer, M. Pollack, and M. Tambe. An intelligent personal assistant for task and time management. *AI Magazine*, 28(2):47–61, Summer 2007.

[81] R. L. D. Mntaras, D. Bridge, and D. Mcsherry. Case-based reasoning: an overview. *AI Communications*, 10:21–29, 1997.

[82] D. Novick and S. Sutton. What is mixed-initiative interaction? In *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, 1997.

[83] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, Nov. 2004.

[84] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[85] I. Olmos, J. A. Gonzalez, and M. Osorio. Inexact graph matching: A case of study. In G. Sutcliffe and R. Goebel, editors, *FLAIRS Conference*, pages 586–591. AAAI Press, 2006.

[86] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. In *Proceedings of the 7th international con-*

ference on Case-Based Reasoning: Case-Based Reasoning Research and Development, ICCBR '07, pages 164–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[87] D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim. A literature review and classification of recommender systems research. *Expert Systems with Applications*, 39(11):10059–10072, Sept. 2012.

[88] D. Patterson, N. Rooney, and M. Galushka. A regression based adaptation strategy for case-based reasoning. In *Proceedings of the Eighteenth Annual National Conference on Artificial Intelligence*, pages 87–92. AAAI Press, 2002.

[89] A. Perini and F. Ricci. An interactive planning architecture: the forest fire fighting case. In M. Ghallab and A. Milani, editors, *New directions in AI planning*, pages 273–283. IOS Press, Amsterdam, The Netherlands, 1996.

[90] E. Plaza and S. Ontañón. Ensemble case-based reasoning: Collaboration policies for multiagent cooperative CBR. In *Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning, ICCBR-01*, Berlin, 2001. Springer-Verlag.

[91] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[92] M. Redmond. *Learning by Observing and Understanding Expert Problem Solving.* PhD thesis, Georgia Institute of Technology, 1992.

[93] T. Reichherzer and D. Leake. Towards automatic support for augmenting concept maps with documents. In *Second International Conference on Concept Mapping*, 2006.

[94] H. A. Reijers. *Design and control of workflow processes: business process management for the service industry.* Springer-Verlag, Berlin, Heidelberg, 2003.

[95] J. Reilly, B. Smyth, L. McGinty, and K. McCarthy. Critiquing with confidence. In *Proceedings of the 6th international conference on Case-Based Reasoning Research and Development*, ICCBR'05, pages 436–450, Berlin, Heidelberg, 2005. Springer-Verlag.

[96] A. J. Rembert. Comprehensive workflow mining. In *Proceedings of the 44th annual Southeast regional conference*, ACM-SE 44, pages 222–227, New York, NY, USA, 2006. ACM.

[97] L. Rokach. Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography. *Computational Statistics & Data Analysis*, 53(12):4046–4072, October 2009.

[98] D. D. Roure and C. Goble. Software design for empowering scientists. *IEEE Software*, 26(1):88–95, January 2009.

[99] S. Shirasuna. *A Dynamic Scientific Workflow System for the Web Services Architecture.* PhD thesis, Indiana University, September 2007.

[100] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):31–36, 2005.

[101] Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance techniques. Technical Report 612, Computer Science Department, Indiana University, 2005.

[102] Y. L. Simmhan, B. Plale, and D. Gannon. Karma2: Provenance management for data-driven workflows. *International Journal of Web Services Research, Idea Group Publishing*, 5(2):1–22, 2008.

[103] B. Smyth and P. McClave. Similarity vs. diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ICCBR '01, pages 347–361, London, UK, UK, 2001. Springer-Verlag.

[104] I. Soboroff, C. Nicholas, and M. Pazzani. Workshop on recommender systems: algorithms and evaluation. *SIGIR Forum*, 33(1):36–43, Sept. 1999.

[105] J. F. Sowa. Conceptual graphs as a universal knowledge representation. *Computers and Mathematics with Applications*, 23(25):75 – 93, 1992.

[106] A. Stahl, M. Minor, and R. Traphöner. Preface: Computer cooking contest. In M. Schaaf, editor, *ECCBR Workshops*, pages 197–198, 2008.

[107] M. Stonebraker. SQL databases v. NoSQL databases. *Communications of the ACM*, 53:10–11, April 2010.

[108] Y. Tian. *Querying graph databases.* PhD thesis, Ann Arbor, MI, USA, 2008. Adviser-Patel, Jignesh M.

[109] Y. Tian, R. C. Mceachin, C. Santos, D. J. States, and J. M. Patel. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics*, 23:232–239, January 2007.

[110] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 963–972, Washington, DC, USA, 2008. IEEE Computer Society.

[111] Y. Tian, J. M. Patel, V. Nair, S. Martini, and M. Kretzler. Periscope/GQ: a graph querying toolkit. *Proceedings of the VLDB Endowment*, 1:1404–1407, August 2008.

[112] N. Tintarev. Explanations of recommendations. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 203–206, New York, NY, USA, 2007. ACM.

[113] van der Aalst W.M.P., ter Hofstede A.H.M., K. B., and B. A.P. Workflow patterns. *Distributed and Parallel Databases*, 14:5–51(47), July 2003.

[114] R. F. van der Lans. Infinitegraph: Extending business, social and government intelligence with graph analytics. Whitepaper, R20/Consultancy, September 2010.

[115] X. Wang, A. Smalter, J. Huan, and G. H. Lushington. G-hash: towards fast kernel-based similarity search in large graph databases. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 472–480, New York, NY, USA, 2009. ACM.

[116] A. Waugh and D. Bridge. An evaluation of the ghostwriter system for case-based content suggestions. In L. Coyle and J. Freyne, editors, *Artificial Intelligence and Cognitive Science (Procs. of the 20th Irish Conference on Artificial Intelligence and Cognitive Science, 2009)*, LNCS 6206, pages 262–272. Springer, 2010.

[117] B. Weber, M. Reichert, and W. Wild. Case-base maintenance for CCBR-based process evolution. In T. Roth-Berghofer, M. H. Göker, and H. A. Güvenir, editors, *ECCBR*, volume 4106 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2006.

[118] B. Weber, W. Wild, and R. Breu. Cbrflow: Enabling adaptive workflow management through conversational case-based reasoning. In P. Funk and P. A. Gonzlez-Calero, editors, *ECCBR*, volume 3155 of *Lecture Notes in Computer Science*, pages 434–448. Springer, 2004.

[119] M. Weber, M. Liwicki, and A. Dengel. a.scatch - a sketch-based retrieval for architectural floor plans. In *ICFHR*, pages 289–294. IEEE Computer Society, 2010.

[120] W. Wilke, I. Vollrath, K.-D. Althoff, and R. Bergmann. A framework for learning adaptation knowledge based on knowledge light approaches. In *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, pages 235–242, 1997.

[121] X. Xiang and G. R. Madey. Improving the reuse of scientific workflows and their by-products. In *ICWS*, pages 792–799. IEEE Computer Society, 2007.

[122] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 335–346, New York, NY, USA, 2004. ACM.

[123] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 766–777, New York, NY, USA, 2005. ACM.

[124] X. Yan, F. Zhu, J. Han, and P. S. Yu. Searching substructures with superimposed distance. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 88–, Washington, DC, USA, 2006. IEEE Computer Society.

[125] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[126] R. Zeng, X. He, J. Le, Z. Liu, and W. van der Aalst. A method to build and analyze scientific workflows from provenance through process mining. Technical report, The Third USENIX Workshop on the Theory and Practice of Provenance, 2011.

[127] S. Zhang, J. Yang, and W. Jin. Sapper: subgraph indexing and approximate matching in large graphs. *Proceedings of the VLDB Endowment*, 3(1-2):1185–1194, Sept. 2010.

[128] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta $\leq$ graph. In *Proceedings of the 33rd international conference on Very large data bases*, Proceedings of the VLDB Endowment, pages 938–949. Proceedings of the VLDB Endowment, 2007.

[129] Y. Zhao, I. Raicu, and I. Foster. Scientific workflow systems for 21st century, new bottle or new wine? In *Proceedings of the 2008 IEEE Congress on Services - Part I*, SERVICES '08, pages 467–471, Washington, DC, USA, 2008. IEEE Computer Society.

[130] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM.

Appendix A

Guide to the Phala Workflow Assistant

A.1 Architecture of Phala

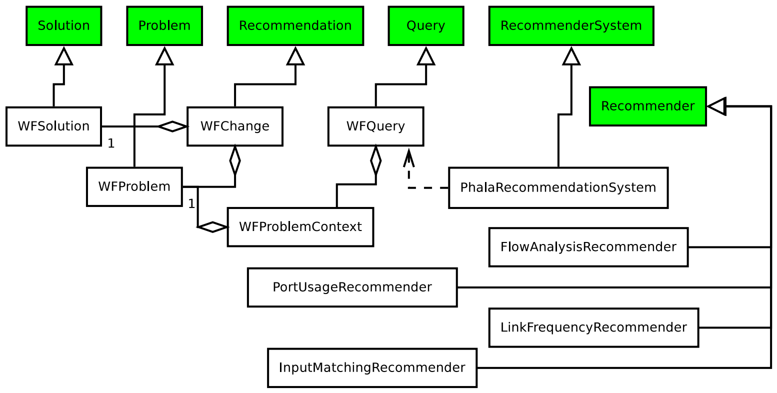


Figure A.1: A UML diagram of Phala's implementation of HRF classes

A.1.1 Recommendation Components

A.1.1.1 WFProblem

Problem contexts for Phala identify a set of processors and, optionally, a recommendation type for which a solution must be found. WFProblems are considered answerable only if they include the expected recommendation type and identify only a single processor. WFProblems may also optionally specify the port on the service to which the solution should be applied, where applicable.

A.1.1.2 WFSolution

Solutions identify the remaining information necessary to solve a problem posted by a WFProblem object. These requirements vary depending on the recommendation type, and are summarized below:

- **ADD_INPUT_SERVICE** - The service type of the processor to be added, and, optionally, the port.
- **ADD_OUTPUT_SERVICE** - The service type of the processor to be added, and, optionally, the port.
- **ADD_INPUT_EDGE** - The processor to be linked, and, optionally, the port.
- **ADD_OUTPUT_EDGE** - The processor to be linked, and, optionally, the port.
- **DELETE_INPUT_EDGE** - The processor to be unlinked, and, optionally, the port.

- **DELETE_OUTPUT_EDGE** - The processor to be unlinked, and, optionally, the port.
- **REPLACE_SERVICE** - The new service type to be used.

A.1.1.3 WFProblemContext

A recommendation that includes only a WFProblem and not a WFSolution. These recommendations serve as a vote to support other recommendations with a matching WFProblem.

A.1.1.4 WFChange

A recommendation to update a workflow in some way, which includes both a WF-Problem and a WFSolution.

A.1.1.5 WFQuery

A WFQuery represents a query to the Phala system. It includes an optional set of WFProblemContext recommendations, which indicate the context in which the user would like to receive a recommendation. Phala will raise its confidence in the relevance of any recommendation that matches the user-specified context to 100%.

A.1.1.6 PhalaRecommendationSystem

This class combines each of the other components, including a Combiner and a SAI database, and allows the user to issue queries and receive response recommendations.

A.1.2 Recommenders

Implementations of the extraction recommender and each of the HPAR's are included within Phala. These recommenders are described in sections 3.5 and 5.4.2, respectively.

A.2 Downloading and Using Phala

Currently, the most up-to-date version of Phala can be found at *http://www.cs.indiana.edu/ jmorwick/Phala*. This URL may or may not be the permanent location for Phala updates.

A.2.1 Service Types and Case Mining

The OPMUtil class contains a static method (ingest OPM) that converts the standard Open Provenance Model Java representation of a provenance trace to an SAI graph representation and stores it within the SAI database. This utility implements Phala's recapture phase and can be used to enact it within a Phala installation.

The code used for mining myExperiment cases can be made available upon request.

A.2.1.1 myExperiment Service Types

A library of processor types has been developed for workflows stored on myExperiment. This collection includes classes for representing the following processor types:

- Biomart
- Biomoby
- constant data

- Java Beanshell scripts
- Java arbitrary class
- KnowARC
- Soaplab
- WSDL

A hash of all properties is taken to identify service types that are not included in the list above, though Phala can easily be extended to include additional types.

A.2.2 Phala Plugins

Both Phala plug-ins that have been developed are intended as a minimal interface for using and testing Phala. Each plug-in allows the user to set the minimum confidence threshold for displaying recommendations and also allows the user to explicitly query the Phala system for a recommendation, which it then displays.

A.2.2.1 XBaya

A modified version of XBaya workflow editor that includes hooks for Phala has been developed. The source code can be made available upon request.

A.2.2.2 Taverna

A plug-in has been developed for the Taverna workflow editor working within its integrated plug-in API. This plugin is pictured in Figure 1.2. The source code can be made available upon request.

Appendix B

Guide to the Hybrid Recommendation Framework

B.1 Architecture of HRF

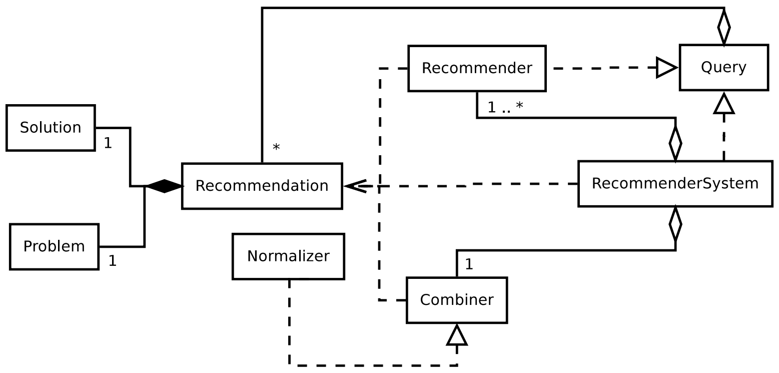


Figure B.1: A UML diagram of HRF's main classes

B.1.1 Coordinating Classes

Figure B.1 outlines the basic class structure of HRF. Each class in the diagram is intended to be extended and tailored to a particular task. Recommendation, Problem, and Solution subclasses are intended to be tailored to the specific domains to which HRF is being applied. This can also be true of the Recommender and Combiner subclasses, but frequently such implementations can be applied to a wider set of domains, such as those included with HRF and described in section B.1.7.1.

B.1.2 Recommendation

Each Recommendation instance identifies two recommendations: the context within which a recommendation to correct or add content could be made and the accompanying recommendation to correct or add content. Each of these components is described in detail below. Additionally, some Recommendations may have only a Problem component.

Recommendation instances will also track the confidence associated with each of their components, and also whether or not the recommendation is a contra-recommendation.

B.1.2.1 Problem

A Problem subclass will provide necessary details to identify where, and for what purpose, a recommendation should be made to correct a problem or add new content.

Problem instances can be answerable or not answerable. An answerable problem is one that fully specifies a problem for which a Solution recommendation can be made. An unanswerable problem is one that is too abstract to be answerable and is missing some information required to determine whether or not a Solution is relevant. Unanswerable Problem recommendations can be used to provide support for other, answerable Problem recommendations when combined by a Combiner.

Subclasses of Problem must be able to determine whether a Problem recommendation is answerable and to what degree one Problem recommendation matches another. The latter function need not be symmetric. For instance, a very general and unanswerable Problem recommendation may match a more specific and answerable recommendation that contains information not included in the first recommendation. However, when reversed, the more specific recommendation may not match the general one, since the general recommendation is missing some of the information that the more specific recommendation contains.

B.1.2.2 Solution

A Solution recommendation contains all necessary information to take action to correct or address a Problem. Solution subclasses, like Problem subclasses, also must be able to determine the degree to which another Solution recommendation matches.

B.1.3 Query

A Query subclass models all user-provided information the system needs to generate recommendations. A Query may optionally include a set of problem recommendations, specified in some way by the user, to take initiative in identifying relevant problems for recommendation.

B.1.4 Recommender

Recommender instances are responsible for generating Recommendation instances throughout the recommendation process. Recommenders may generate initial recommendations when presented with a Query instance. Each Recommender will be given opportunities to examine the pool of existing Recommendations and to make additional Recommendations

B.1.5 Combiner

The Combiner role is to refine a set of recommendations. Combiners act to transform and filter Recommendation instances made by all of the Recommenders before they are issued to the user. Combiners may produce new recommendations that combine components of the recommendations they consume, or they may simply alter the confidence values of recommendations.

B.1.6 Normalizer

Normalizer subclasses are Combiners that implement regression algorithms for transforming confidence values of recommendations to better match the likelihood that the recommendation will be relevant or accurate. Normalizers work with a PerformanceRecord (described in section B.2) to estimate the probability of a problem recommendation's relevance or a solution recommendation's accuracy based on measurements of the metrics associated with confidence values available in the PerformanceRecord.

B.1.7 RecommendationSystem

RecommendationSystem is extended to work with specific Recommendation types. RecommendationSystems house both Recommenders and a Combiner. A RecommenderSystems ingests Query instances, produces Recommendation instances from its Recommenders, and then refines them with its Combiner before issuing them to the user.

B.1.7.1 Algorithm Options Shipping with HRF

Below is a list of implementations of combiners that ship with HRF for the abstract tasks identified in the previous section.

- **Combiner**:
  - **GeneralCombiner** - Implements each of the combining algorithms discussed in section 3.8.3.
  - **UnionCombiner** - Combines the results from two or more other Combiner instances, and returns them as a single set.
  - **CompositionCombiner** - Filters the results from one Combiner through a second Combiner and returns the results from the second Combiner.
- **Normalizer**:
  - **Binning** - A regression algorithm that forms a series of confidence intervals which are then associated with a new confidence estimate computed from the average relevance or accuracy of recommendations previously made by a recommender.

– **WeightedAverage** - A regression algorithm that computes a new confidence value based on the average relevance or accuracy of recommendations within a PerformanceRecord, weighted by their inverse difference in confidence.

B.2 Evaluation Framework for HRF

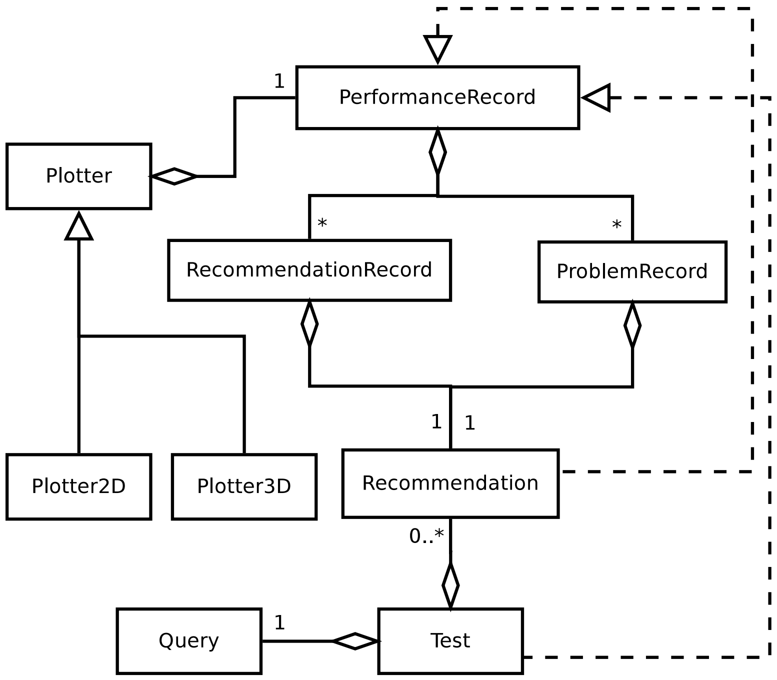


Figure B.2: A UML diagram of the Evaluation Framework for HRF

B.2.1 Coordinating Classes

Figure B.1 outlines the evaluation class structure of HRF. These classes are used to write programs that test an HRF system by producing a series of Test instances that can be used with a RecommendationSystem to produce a PerformanceRecord instance.

B.2.2 Test

Each test consists of a Query and a set of expected Recommendations. During a test, the Query is presented to the RecommendationSystem and the resulting Recommendations are compared with the expected set contained by the Test instance.

B.2.3 ProblemRecord

During any test, there may be several problems, each identified by the expected recommendations. One of these recommendations is held by a ProblemRecord instance to model a single problem within a test.

B.2.4 RecommendationRecord

During a test, the RecommendationSystem may produce many Recommendations, each of which is tracked by a RecommendationRecord instance. These instances may be associated with multiple ProblemRecord instances, but can be associated with only a single Test instance.

B.2.5 PerformanceRecord

A PerformanceRecord records an aggregate of the records for a complete set of tests. PerformanceRecord instances can be used to generate plots or to perform regression transformations on new recommendations.

B.2.6 Plotter

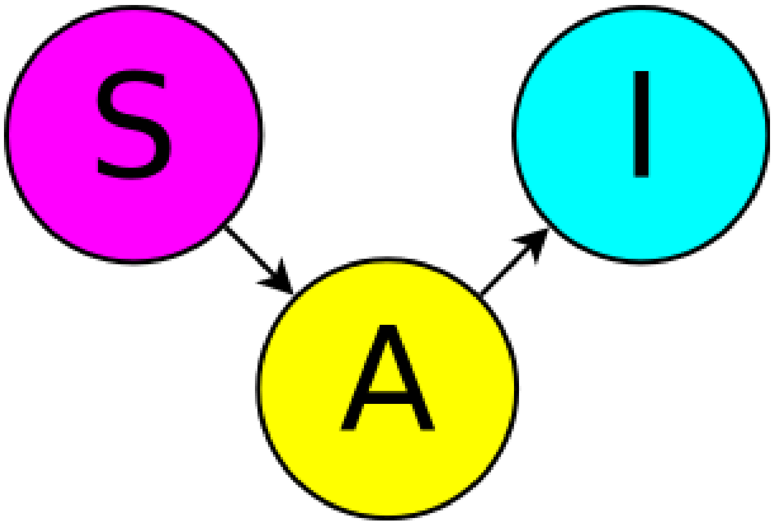
Each Plotter subclass is responsible for generating a dataset for a metric (including those discussed in section 5.2). Typically, Plotters will be subclasses of the Plotter2D and Plotter3D classes, which provide structure for developing 2D and 3D plots, respectively, and are themselves subclasses of the Plotter class.

B.3 Downloading and Using HRF

Currently, the most up-to-date version of HRF can be found at *http://www.cs.indiana.edu/ jmorwick/HRF*. This URL may or may not be the permanent location for HRF updates.

Appendix C

Guide to the Structure Access Interface



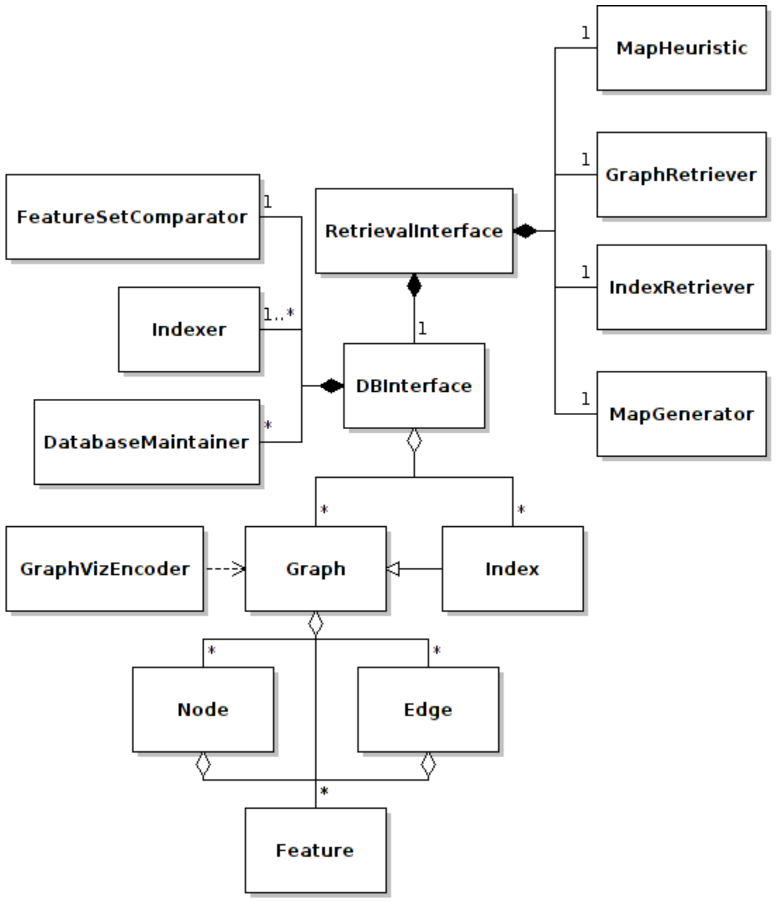


Figure C.1: A UML diagram of SAI's main classes

C.1 Architecture of SAI

C.1.0.1 Coordinating Classes

Figure C.1 outlines the basic class structure of SAI. Two classes serve to coordinate SAI's functionality: **DBInterface** and **RetrievalInterface**. The classes in bold-face in the following paragraphs are abstract classes that perform a task, the implementation of which is intended to be customizable for usage in specific domains.

C.1.0.2 DBInterface

DBInterface handles the low-level loading and saving of structures and indices, including methods for caching and direct database access. **DBInterface** also contains a **FeatureSetComparator**, which is used to check compatibility between feature sets. This is stored in DBInterface to allow for DBInterface to perform compatibility checks between graphs and indices stored in the database, without any dependence on retrieval related classes.

DBInterface contains **Indexer** instances, which are used to index newly saved structures. DBInterface also houses various **DatabaseMaintainer** instances, which embody background tasks that lazily keep the databases indexing consistent and efficient.

C.1.0.3 RetrievalInterface

RetrievalInterface contains objects that handle each step of a similarity query. An **IndexRetriever** first retrieves a set of indices from the database that are compatible with a query structure.

Next, a **GraphRetriever** ranks the candidate cases in the database that are compatible with any of the same indices as the query. A **GraphRetriever** implements the details of how to determine which candidate is more likely to be desired in the retrieval according to how well it covers the indices; structural details of candidate cases are not yet loaded.

Once an ordered list of candidate cases is retrieved, the RetrievalInterface uses a **MapGenerator** to create a mapping between the query and each of the top N candidate cases. Then the cases are re-ordered according to the value a **MapHeuristic** applies to the mapping generated by the **MapGenerator**. The top K of these cases are then returned by the RetrievalInterface.

C.1.0.4 Algorithm Options Shipping with SAI

Below is a list of implementations that ship with SAI for the abstract tasks identified in the previous section. These algorithms were developed either for their simplicity, or for their relatedness to the problem of using provenance traces to assist workflow authors.

- **Indexer**:
  - **Path1** - Creates an index consisting of two nodes and a single edge, retaining features from specified feature classes.
  - **Path1Lookup** Creates the same indices as Path1 but attempts to find matching indices in the database rather than creating repeated instances. This is advantageous because it both eliminates the need for combining indices during system idle time and also insures that there are no repeated indices in the database at any time. The disadvantage, however minor, is that the check for existing indices increases index generation time.
- **IndexRetriever**:
  - **Path1Retriever** Retrieves indices created by the Path1 indexers in one step with a single SQL query.
- **GraphRetriever**:

  - **SimpleCountRetriever** Retrieves IDs of all graphs indexed by the specified indices. Graphs are ordered by the number of specified indices associated with them.

- **MapGenerator**:

  - **SearchMapGenerator** This is an abstract generator that is completed by including a SearchQueue implementation. SearchQueue implementations are a queue of node mappings and dictate how the mappings are ordered and extended in the search process. HeuristicPriorityQueue is an implementation that orders maps according to a provided MapHeuristic and expands maps by creating all maps extended by one additional node mapping.

- **MapHeuristic**:

  - **BasicEdgeCounter**: Counts the number of edges that are compatible according to the map.
  - **WeightedSum**: Counts edges as BasicEdgeCounter does, but assigns them a specified score. Additionally, compatible node and edge features are scored according to a score assigned to each Feature class and are added to the total score for a map.

- **FeatureSetComparator**: FeatureSetComparator implementations are used to judge whether or not one set of features is compatible with another set of features. In the following descriptions, consider that the comparator is deciding whether feature set A is compatible with feature set B.

  - **ManyTo1**: Indicates compatibility if, for each feature x in set A, there is a feature in set B that is compatible with x.
  - **Complete1To1**: Similar to ManyTo1, but doesn't allow for a single feature in set B to be compatible with more than one feature in set A. If such a compatibility relationship exists between the features of set A and the features of set B, this comparator will return true; otherwise it will return false.

- **DatabaseMaintainer**: These are the available idle-time tasks that can be run continuously to maintain SAI databases.
  - **IndexCompatibilityChecker**: Checks each index in the database against each other index and records which indices are compatible with which other indices. This facilitates hierarchical index retrieval and also assists in the next maintenance task.
  - **IndexConsolidator**: Checks each index against each other index to see if they are equivalent by checking compatibility relationships recorded by IndexCompatibilityChecker or another source. If they are, one index is deleted and the other gains the associations of the deleted index.

C.1.1 Database Structure

SAI is built on top of the mySQL database server. This subsection describes the various tables used within an SAI database in mySQL:

- **graph_instances** Entries in this table provide basic information for each instance of a graph in a SAI database. Thus far, these entries contain only an ID number for the graph and an indication of whether or not it is intended to be used as an index.

- **node_instances** Entries in this table provide information for each node instance within a graph in SAI. Node instances have IDs that are only unique within the context of a graph.

- **edge_instances** Entries in this table provide information for each edge instance within a graph in SAI. IDs are again only locally unique. These entries reference, in order, the nodes they connect and the parent graph containing the nodes.

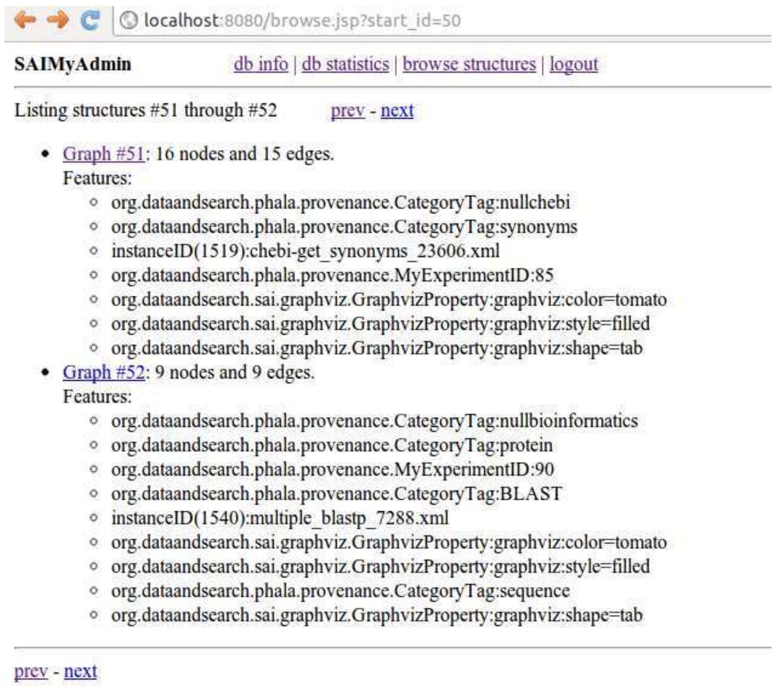
- **feature_instances** Entries in this table describe instances of features within SAI. This includes the full name of the Java class that these features instantiate. Feature instances are always unique. There is no distinction between two different features that contain the same data and instantiate the same class. They are both treated as the same instance within the SAI database.

- **graph_features** Entries in this table relate feature instances to graphs within the database.

- **node_features** Entries in this table relate feature instances to single nodes within graphs within the database.

- **edge_features** Entries in this table relate feature instances to single edges within graphs within the database.

- **graph_indices** Entries in this table determine which indices (graphs) are asso-



```
localhost:8080/browse.jsp?start_id=50

SAIMyAdmin          db info | db statistics | browse structures | logout

Listing structures #51 through #52          prev - next

  • Graph #51: 16 nodes and 15 edges.
    Features:
      ○ org.dataandsearch.phala.provenance.CategoryTag:nullchebi
      ○ org.dataandsearch.phala.provenance.CategoryTag:synonyms
      ○ instanceID(1519):chebi-get_synonyms_23606.xml
      ○ org.dataandsearch.phala.provenance.MyExperimentID:85
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:color=tomato
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:style=filled
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:shape=tab
  • Graph #52: 9 nodes and 9 edges.
    Features:
      ○ org.dataandsearch.phala.provenance.CategoryTag:nullbioinformatics
      ○ org.dataandsearch.phala.provenance.CategoryTag:protein
      ○ org.dataandsearch.phala.provenance.MyExperimentID:90
      ○ org.dataandsearch.phala.provenance.CategoryTag:BLAST
      ○ instanceID(1540):multiple_blastp_7288.xml
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:color=tomato
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:style=filled
      ○ org.dataandsearch.phala.provenance.CategoryTag:sequence
      ○ org.dataandsearch.sai.graphviz.GraphvizProperty:graphviz:shape=tab

prev - next
```

Figure C.2: A Screenshot of the SAIMyadmin Web Application

ciated with other graphs (which may or may not also be indices). Each entry denotes one such relationship between an index and a graph.

C.2 Viewing and Managing SAI Databases

C.2.1 SAIMyAdmin

SAIMyAdmin is a web application developed through this dissertation to afford easy viewing of structures and database statistics for a SAI database. This application was written using JSPs (Java Server Pages) and must be run on a Servlet Container such as Apache Tomcat[1]. Logging in to view an SAI database requires the mySQL

[1]Available at *http://tomcat.apache.org/*

database credentials used by SAI to access the database. At that point, the user can view database statistics or browse structures in the database. Detailed information about each structure can be viewed, including which structures index a structure. A screenshot of this application appears in Figure C.2.

In addition to viewing structure details, SAIMyAdmin allows for basic management through deletion of structures and breaking and forming index relationships. Currently, there is no functionality for performing queries or creating and modifying structures[2]. Future versions are also planned to facilitate management of background processes.

Additionally, SAIMyAdmin is intended to be a launching point for experiments involving the SAI database. Future versions are planned to allow execution and monitoring of scripts that perform some of the experiments used in this dissertation, and to allow for inclusion of new experiments designed by third parties, much in the same way that SAI already allows for third parties to develop new retrieval algorithms.

C.3 Running SAI Experiments

Until SAIMyAdmin has the experimental feature added to it, experiments must be run manually. Experiments in Chapter 3 used a logging tool to generate linked HTML reports that organize results by structure and, further, by each structure compared to that structure. Graphical representations of maps, such as that shown in Figure 4.3, are drawn for each map developed between a pair of cases. These reports are useful

[2]Although modifying structures is not allowed in SAI, a copy-on-write feature is planned to allow modifications through a modify/copy/delete process.

for browsing for examples (such as those used in this dissertation), finding unique outliers, and debugging. This report-generating tool is part of the HRF framework.

C.3.1 Obtaining SAI and Experimental Code

Currently, the most up-to-date version of SAI and SAIMyAdmin can be found at *http://www.cs.indiana.edu/˜jmorwick/SAI*. This URL may or may not be the permanent location for SAI updates. Code for running the experiments can be made available upon request. The code on the SAI page may not continue to be compatible with the experimental code, but a snapshot of SAI at the time it was used for this dissertation will also be made available. Additionally, snapshots of the mySQL databases used for the experiments can also be available, though these databases can be reformed from the code, as well (described in the next section).

C.3.2 Running the SAI Experiments

If you wish to re-run some of the experiments from this dissertation, you must follow these steps:

C.3.2.1 Obtain the Code

Follow the instructions provided in the previous subsection.

C.3.2.2 Prepare the Database

Download and install the latest version of the mySQL database. Next, you may load a snapshot of the databases provided or re-form the databases from scratch.

Database images are stored in SQL files, which can be loaded through most mySQL management clients.

It is recommended that you use the stored database images for experiments rather than re-forming the database from scratch. This is due to the fact that re-forming the database puts a significant strain on the webservers hosting the data and also takes a significant amount of time. Furthermore, there is no guarantee that the code used for re-forming the databases will continue to work, as the myExperiment and pubchem websites may alter their interfaces in the future, breaking this code. However, if you wish to reform the database, you may run the main method of the *MyExperimentUtil* class to create the myExperiment database, or run the main method of the *PubChemUtil* class to create the PubChem database.

Database settings can be changed by altering the constants in the PubChemUtil class or, for myExperiment, the *Constants* class. **DO NOT MODIFY THE CODE THAT FORCES THE LOADER TO WAIT BETWEEN CASES!** It is important for a web-mining application to behave properly, and in order to do so, it must not inundate a server with requests. The database re-forming software was programmed to wait 10 seconds between queries to prevent disruption of service for the host website, but also to prevent the user of this software from being banned from accessing the website!

C.3.2.3 Running the Experiments

To run the tests for insuring the integrity of the retrieval process and for finding the average similarity of cases in the case base and developing the heat maps in experiment 1, run the main method of the class *FullComparison*. Constants at the top of this class can be changed to compare cases only to themselves or to choose the retrieval interface. The retrieval interface (PHALA or PUBCHEM) can also be chosen by providing either of these as a command-line argument. A detailed HTML report will be written to the directory specified by the **OUTPUT_DIRECTORY** field in the *DetailedTestLog class*, including the heat map.

To run the tests for comparing phase 1 to phase 2 in experiment 1, run the main method in the *WKSimilarityTest*. The same rules about constants and command-line arguments apply to this class as well. However, in this case, report details will be printed to standard output.

To run the test for scalability in experiment 2, execute the *SimpleGenerator* class's main method. If you wish to alter the parameters of this test, you must modify the code at the beginning of this class's main method. Test results will be printed to standard output.

Joseph Kendall-Morwick **December 25, 2012**
Instructor - DePauw University josephkendallmorwick@depauw.edu
Greencastle, Indiana, USA http://www.csc.depauw.edu/~jmorwick

Education

- **Ohio State University** Columbus, OH
  B.S. Mathematics / Computer and Information Science *1999 - 2003*
- **Indiana University** Bloomington, IN
  M.S. Computer Science *2005 - 2007*
- **Indiana University** Bloomington, IN
  Ph.D. Computer Science *2007 - 2012*

Teaching Experience

- **DePauw University** Greencastle, IN
  Instructor *Aug. 2011 - may. 2013*
  - Courses: Computer Science 1 (3 sections), Object-Oriented Software Development (CSC121) (3 sections), Operating Systems (CSC428), Senior Project (CSC498)
- **Indiana University** Boomington, IN
  Instructor *Jan. 2010 - May. 2010*
  - Course: Fundamentals of Computing Theory (B401) (2 sections)
- **Indiana University** Boomington, IN
  Co-Instructor *Jan. 2009 - May. 2009*
  - Course: Introduction to Software Systems (C212)
- **Indiana University** Boomington, IN
  Peer Tutoring Supervisor *Aug. 2008 - Dec. 2008*
  - Course: Introduction to Software Systems (C212)
- **Indiana University** Boomington, IN
  Associate Instructor *Aug. 2005 - Dec. 2007*
  - Courses: Introduction to Programming I (A201), Introduction to Computer Science (C211), Introduction to Software Systems (C212), Theory of Computing (B501), Elements of Artificial Intelligence (B551)
- **Ohio State University** Columbus, OH
  Grader / Tutor *Aug. 2000 - Jun. 2004*
  - Courses: Elementary Computer Programming (201), Introduction to Computer Systems (360), Introduction to the Principles of Programming Languages (655), Introduction to Operating Systems (660), Survey of Artificial Intelligence II: Advanced Topics (730), Knowledge-Based Systems (731)
- **Explorer Post 891** Columbus, OH
  Volunteer Instructor *Aug. 2001 - Jun. 2004*

  - Course: Java Programming

Peer-Reviewed Publications and Technical Reports

- Kendall-Morwick, J. and Leake, D. 2012 *Facilitating representation and retrieval of structured cases: Principles and toolkit.* Information Systems, 2012.
- Kendall-Morwick, J. and Leake, D. 2012 *On Tuning Two-Phase Retrieval for Structured Cases.* In Proceedings of Process-Oriented Case-Based Reasoning (Lyon, France, September 3-6, 2012). L. Lamontagne and J. A. Recio-Garcia, Ed.
- Kendall-Morwick, J. and Leake, D. 2011 *A Toolkit for Representation and Retrieval of Structured Cases.* In Proceedings of Process-Oriented Case-Based Reasoning (Greenwich, England, September 12-15, 2011). C. Marling, Ed.
- Leake, D. and Kendall-Morwick, J. 2010 *External Provenance, Internal Provenance, and Case-Based Reasoning.* In Proceedings of Provenance-Aware CBR: Applications to Reasoning, Metareasoning, Maintenance and Explanation (Alessandria, Italy, July 19-22, 2010). C. Marling, Ed.
- Kendall-Morwick, J. 2010 *Towards an Ensemble Framework for Assisting in Synthesis Tasks.* In Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference (Daytona Beach, FL, May 19-21, 2010). H. Guesgen and R. Murray, Eds. AAAI Press 2010.
- Leake, D. and Kendall-Morwick, J. 2009. *Four Heads Are Better than One: Combining Suggestions for Case Adaptation.* In Proceedings of the 8th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development (Seattle, WA, July 20 - 23, 2009). L. Mcginty and D. Wilson, Eds. Lecture Notes In Artificial Intelligence, vol. 5239. Springer-Verlag, Berlin, Heidelberg, 165-179.
- Leake, D. and Kendall-Morwick, J. 2008. *Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance.* In Proceedings of the 9th European Conference on Advances in Case-Based Reasoning (Trier, Germany, September 01 - 04, 2008). K. Althoff, R. Bergmann, M. Minor, and A. Hanft, Eds. Lecture Notes In Artificial Intelligence, vol. 5239. Springer-Verlag, Berlin, Heidelberg, 269-283.

Presentations

- *On Tuning Two-Phase Retrieval for Structured Cases* The 2nd Workshop on Process-Oriented Case-Based Reasoning, Sep. 4, 2012. Lyon, France
- *Move Towards the Light* Invited Talk at The Doctoral Consortium at the 11th International Conference on Case-Based Reasoning, Sep. 3, 2012. Lyon, France
- *Four Heads Are Better than One: Combining Suggestions for Case Adaptation* The 8th International Conference on Case-Based Reasoning, Jul. 20 - Jul. 23, 2009. Seattle, WA

- Invited Guest Lecture on applying CBR to workflow authoring in the course *Topics in Data and Search Informatics* at Indiana Universty, Nov. 17, 2008. Bloomington, IN
- *Leveraging Provenance for Case-Based Support of e-Science Experimentation.* Invited talk at Data and Search Institute Seminar Series, Indiana University, Oct. 22, 2008. Bloomington, IN
- *Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance* The 9th European Conference on Advances in Case-Based Reasoning, Sep. 1 - Sep. 4, 2008. Trier, Germany
- *DJMIRI: a Dynamic Java-based Music Information Retrieval Interface.* Invited Guest Lecture in the course *Organization of and Searching in Musical Information* at Indiana Universty, Apr. 24, 2006. Bloomington, IN
- *Internet Trolls as a Consequence of a 'Broken' Communication Medium.* Time / Passages: A National Interdisciplinary Graduate Student Conference, Mar. 22 - Mar. 24, 2007. Bloomington, IN

Professional Activites

- **Twenty-Sixth International FLAIRS Conference** — St. Pete Beach, FL
  Program Committee Member — *2013*
- **TRUE: Traces for Reusing Users' Experience Workshop at ICCBR 2012** — Lyon, France
  Workshop Program Committee Member — *2012*
- **ACM-ICPC East Central North America Regional Programming Contest** — Cincinnati, OH
  Coach for the DePauw University programming teams — *2011*
- **Consortium for Computing Sciences in Colleges : Midwest** — Huntingon, IN
  Coach for the DePauw University programming teams — *2011*
- **Twenty-Fifth International FLAIRS Conference** — Marco Island, FL
  Program Committee Member — *2012*
- **Indiana University Undergraduate Research Competition** — Bloomington, IN
  Mentor — *2011*
- **Twenty-Fourth International FLAIRS Conference** — Palm Beach, FL
  Program Committee Member — *2011*
- **Provenance-Aware Case-Based Reasoning at ICCBR 2010** — Alessandria, Italy
  Workshop Organizing Committee Member — *2010*
- **Consortium for Computing Sciences in Colleges : Midwest** — Chicago, IL
  Coach for the Indiana University programming teams — *2009*
- **Just Be** — Bloomington, IN
  Presenter — *Aug. 2008 - Dec. 2008*
- **Bring IT On! Reunion Workshop** — Indianapolis, IN
  Assistant Organizer — *2008*
- **Bring IT On! Workshop** — Bloomington, IN
  Assistant Organizer — *2007*

Service

- **IEEE 4th International Conference on eScience** Indianapolis, IN
  Conference Volunteer *2008*
  Just Be is an outreach program which introduces K-12 students to Computer Science
- **IU Computer Science and Informatics Graduate Research Symposium** Bloomington, IN
  Co-Coordinator *2008*
- **Computer Science Graduate Student Association** Indiana University - Bloomington, IN
  President *May. 2007 - May. 2008*

Research Experience

- **Indiana University** Boomington, IN
  Research Assistant *Jun. 2006 - Aug. 2006; Jan. 2008 - Present*
  - Created a case-based reasoning system for intelligent assistance of scientific workflow composition
  - Used the SAL model checker to prove timing properties of clock protocols for serial communication
  - Developed software to determine the usefulness of a trusted platform module in securing bluetooth services
- **Indiana University** Boomington, IN
  Mentor for Undergraduate Research *Jun. 2006 - Aug. 2006; Jan. 2008 - Present*
  - Mentored 3 undergraduates in a semester-long research project to develop a game system for teaching CS2
  - My students won first prize in the poster competition at the end of the semester

Work Experience

- **Scalable Network Technologies** Los Angeles, CA
  Programmer / Analyst *May. 2009 - Aug. 2009*
  - Developed code for statistics tracking and other purposes for the ”Communications Effects Server”, a large-scale C++ networking simulator.
  - Worked closely with a team of developers to engineer evaluative simulation scenarios for emerging tactical military radio technologies.
  - Analyzed results from test scenarios and produced reports and presentations for clients.
- **Moodle** Google Summer of Code
  Developer *Jun. 2008 - Aug. 2008*
  - Worked remotely with an international team of open-source developers on the Moodle learning management system (LMS) through a Google sponsored program.

  - Designed and integrated a system for organizing blog posts among the particular classes, groups, or projects a student is involved in, among other improvements to the blog system.

- **eSchool Consultants** Reynoldsburg, OH
  Programmer / Analyst *Jul. 2004 - Jul. 2005*
  - Developed web applications with PHP to maintain a mySQL database of student records and provide instructors with intuitive interfaces to access data and reports.
  - Transferred the schools 3rd party LMS (Learn.com) to an in-house open-source LMS (Moodle), and integrated this system into the schools existing management software.
  - Integrated the use of barcode scanners for the purpose of equipment tracking.

Awards, Grants & Honors

Travel Grant - Career Mentoring Workshop at SIGCSE 2012	2012
Faculty Development Grant - DePauw University 2012	2012
Student Travel Grant - The 9th European Conference on Advances in Case-Based Reasoning	2008
Trustees Scholarship - Ohio State University	1999-2002
Faculty Award - Indiana University	1998

Affiliations

- Association for the Advancement of Artificial Intelligence
- Association for Computing Machinery