A Phased Ranking Model for Information Systems

Rui Liu

September 8, 2016

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Eric Nyberg (Carnegie Mellon University), Chair
Teruko Mitamura (Carnegie Mellon University)
Jaime Carbonell (Carnegie Mellon University)
Bowen Zhou (IBM T. J. Watson Research Center)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

To effectively sort and present relevant information pieces (e.g., answers, passages, documents) to human users, information systems rely on ranking models. Existing ranking models are typically designed for a specific task and therefore are not effective for complex information systems that require component changes or domain adaptations. For example, in the final stage of question answering, information systems such as IBM Watson DeepQA rank all results according to their evidence scores and judge the likelihood that each is correct or relevant. However, as information systems become more complex, determining effective ranking approaches becomes much more challenging.

Prior work includes heuristic ranking models that focus on a particular type of information object (e.g. a retrieved document, a factoid answer) using manually designed features specific to that information type. These models, however, do not use other, non-local features (e.g. features of the upstream/downstream information source) to locate relevant information. To address this gap, my research seeks to define a ranking approach that should easily and rapidly adapt to any version of system pipelines with an arbitrary number of phases.

We describe a general ranking approach for multi-phase and multi-strategy information systems, which produce and rank significantly more candidate results than the single phase and single strategy information systems to achieve acceptable robustness and overall performance. Our approach allows each phase in a system to leverage information propagated from preceding phases to inform the ranking decision. By collecting ranking features from the derivation paths that generate candidate results, the particular derivation path chosen can be used to predict result correctness or relevance. Those ranking features can be detected from an abstracted system object graph which represents all of the objects created during system execution (e.g. provenance) and object dependencies. This ranking approach has been applied to different domains including question answering and biomedical information retrieval. Experimental results showed that our proposed approach significantly outperforms comparable answer ranking models on the two domains.

Contents

List of Figures

List of Tables

Chapter 1

Research Overview

In this thesis, we define the problem of phased ranking and propose a solution for information systems. The three major research stages include 1) initializing phased ranking hypothesis and rationale (2010-2013), 2) developing strong baseline approaches such as Generalized Watson Rank (2011-2014), and 3) applying the phased ranking model to different domains (2012-2016).

1.1 Phased Ranking Hypothesis and Rationale (2010 - 2013)

The phased ranking model we study in the thesis is originated from two courses at Carnegie Mellon Universitys Language Technologies Institute: Question Answering Lab [1] and Software Engineering for Information Systems [2]. Our ranking model was initially designed and implemented for an experimental version of Ephyra offered by Question Answering Lab, which focused on answering open domain factoid questions. An answer ranking model estimates a confidence score by considering various features of each answer candidate. Answer ranking models can be roughly separated into two approaches: exploiting answer redundancies and combining relevance scores for single answers. The former approach is favored by factoid question answering research, whereas the latter is preferred by community-based question answering research. Exploiting answer redundancy in this stage is reported to be critical, related research contains the topic of finding answer duplicates from a Web search engine [53] or Wikipedia [45], expanding text corpora with external sources [76], grouping similar answers [40] or sampling a representative set from a large corpus [16, 94]. Another important view of answer ranking models is to explore answer relevance through textual clues by searching similarly answered questions from QA communities [20, 85, 93], using textual entailment approaches [36], using translation based methods [72, 85], or applying structured search in natural language [9].

Leveraging both factors is nontrivial. The standard answer ranking approach [45] proposed a simple classification approach by combining relevance scores and similarity scores.

[1] http://lti.cs.cmu.edu/intranet/lti-course-listings#11796
[2] http://lti.cs.cmu.edu/intranet/lti-course-listings#11791

An answer relevance score is produced by merging scores computed from a single formula (e.g., the TF.IDF based approach). Assume a candidate "Vesuvius" has two redundant instances retrieved from different data sources with the TF.IDF scores 0.1 and 0.6 separately. The two scores are merged as the answer relevance score 0.7 for the candidate "Vesuvius".

However, standard ranking approaches that use heuristic merging cannot easily adapt to new feature sets designed to better estimate relevance scores. For example, consider TREC11 question 1396 ("What is the name of the volcano that destroyed the ancient city of Pompeii?"); we retrieved the following passage "Pompeii, along with Herculaneum and many villas in the surrounding area, was mostly destroyed and buried under 4 to 6 m (13 to 20 ft) of volcanic ash and pumice in the eruption of Mount Vesuvius in AD 79." Using a TF.IDF-like score assigns close relevance scores to the candidate instances such as "Mount Vesuvius" and "Herculaneum." However, it is not clear how the standard ranking approach leverages the other feature types (e.g., shallow, semantic, or syntactic features) for answer merging.

Motivated by this problem, our initial research question is that should we learn the relevance score before the merging step or after it. Ideally, if we have gold standard data for each candidate instance, a learning step before the merging step would combine those features as a relevance score. In many cases, training data only contains answer keys to those questions and the judgment to each candidate instance is unavailable. Therefore, a learning step after merging step can be a useful option.

In our 2011 VLIS Capstone project [3], as a natural extension to the standard answer ranking approaches, we defined a ranking framework consisting of three stages: extract features for each candidate instance, clustering/merging instances, and ranking answer candidates. We performed clustering across the set of answer candidates to group similar candidates. Then, within each cluster, the instances are merged into answer candidates by using customized merging functions. Finally, the answer candidates are ranked by using features extracted from merging steps. In the end, we select the answer candidate in the top ranked cluster as the final answer.

Around the same time, our research was greatly influenced by the research of information processing architecture frameworks [25, 26, 27], offered by the course of Software Engineering for Information Systems. The Unstructured Information Management Architecture (UIMA) [27] is a framework that standardizes information systems by managing the workflows, type systems, resources, etc. UIMA was originally developed by IBM to support the processing and analysis of unstructured information (e.g., text, speech, images or videos). An information system is built via defining a set of Analysis Engines (AEs). For example, a question answering system can be built via defining four analysis components: 1) question analysis, 2) document or passage retrieval, 3) answer extraction and 4) answer selection. The question analysis component classifies the input question into one of the predefined categories by deriving lexical, syntactic and semantic information of the given question. Queries composed by this step are then fed into a document or passage retrieval component to find relevant documents or passages. Those results are supplied to an

[3]The VLIS program has changed its name to MCDS, the capstone project description can be found at https://mcds.cs.cmu.edu/learn-us-curriculum

answer extractor component to generate answer candidates. Finally, the answer selection component distinguishes correct answers from incorrect candidates by reasoning or using supporting evidence with machine learning algorithms.

Motived by this previous works on information system frameworks, we attempt to leverage those analysis components, so that information propagated from preceding components can inform the final answer ranking decision and reduce those biases caused by "local" features designed heuristically around answer extraction components. To support this hypothesis, we implemented an answer ranking approach [49] for a multiphase, multi-strategy question answering system that produced and ranked a large number of answer candidates. The proposed ranking approach is accomplished by a system object graph which represents all of the objects created during system execution, as well as object dependencies (e.g. provenance). We evaluate the effectiveness of the proposed ranking approach in a multi-phase question answering system built by recombining pre-existing software modules. Experimental results show that our proposed approach significantly outperforms comparable answer ranking models.

1.2 Developing Generalized Watson Rank (2011 - 2014)

When IBM's Watson won TV's Jeopardy! game against two world-class champions in February 2011, it demonstrated the ability of the system and represented a crowning achievement in the field of open-domain question answering, to answer natural language questions. Watson showed the power of artificial intelligence and natural language understanding to answer humans' questions from providing daily factoid questions to making sophisticated decisions. Technological advances (e.g., artificial intelligence and natural language understanding) inspired by question answering will continue to improve mobile devices, personal and industrial computers, in the area of healthcare, finance, manufacturing, and many other domains.

However, because building a complex question answering system like Watson takes many years, manually creating and training a new restricted-domain QA agent requires significant resources and human effort. One of the main problems is that expertise is often needed for adapting ranking algorithms. For example, given a system pipeline, typically features are manually and heuristically defined and configured by domain experts. This approach can be inefficient and generate inconsistent results because domain experts may be required for each "local" system component. Consequently, adaptation to a new domain is limited. For example, existing knowledge sources may not cover sufficient information to answer new domain questions; the ontologies may not appropriately represent the semantics of the new domain; and the strategy (e.g., machine learning algorithms and features) used to distinguish answers from noisy candidates can vary significantly from the current QA system. It is, therefore, vital to train and evolve the methods of the current system to adapt to a new specific domain.

During my research internship in IBM's DeepQA group at 2014, I was especially interested in the adaptation challenges of Watson ranking framework [28, 32].

In contrast to building a restricted-domain QA system, we propose a pipelined pro-

cedure from a generalization perspective to meet the new requirement of both research and development. The basic idea is that new domain QA agent can be created and optimized rapidly by learning from the pipelined system of a new domain with existing system components and outputs.

1.3 Applications to Information Systems (2012 - 2016)

During my thesis proposal, a phased ranking model, which is extended from our answer ranking model, has been introduced for modeling information systems with multiple sequential phases. The method mainly includes a system object graph representation that captures information about every processing option and intermediate data object produced by a pipelined system. Based on the representation, a phased ranking model leverages information propagated from preceding phases to inform the ranking decision. We demonstrated that our approach is statistically significantly better than or indistinguishable from baseline factoid QA approaches for TREC datasets 9-12. Although tested on factoid question answering, we believe that our approach is generalizable to domain adaptation in many other NLP applications with sequential phases and multiple options per phase.

During my thesis work, we focus on two directions. The first direction presents the generality of our phased ranking model by applying it to another high-impact domain: biomedical information retrieval. A state-of-the-art biomedical information system, TREC Genomics, was chosen as a new case study to empirically evaluate the phased ranking approach. The biomedical question answering pipeline first detects synonyms, acronyms, and lexical variants for query expansion. Then a standard retrieval method is applied to find relevant documents (or passages), which are partitioned into important sentences. Finally, a ranking component is responsible for ranking. Unlike TRECQA, the task of TREC Genomics is to retrieve passages from biomedical journals to answer the structured questions from real biologists. Instead of top-1 binary relevance judgments corresponding to the right or wrong answer, answers can be multi-graded in the biomedical retrieval problem. in most QA systems the desired goal is to find the top-ranked answer right [28]. In contrast to question answering systems where Precision@1 is the primary metric, the biomedical question answering system is evaluated on an entire ranked list, using the metric of mean average precision (MAP).

The second direction attempts to deepen our comparisons with an additional state-of-the-art approach, Generalized Watson Baseline (GWB) [28, 32], for both applications, question answering and biomedical information retrieval. We also perform overlap analysis with detailed examples to better explain our proposed phased ranking model.

An overview of the completed and future work is presented in Table 1.1 with respect to two tasks with different domains and pipelined systems. The cells in yellow show the tasks finished in my thesis proposal. The cells in green represent the newly completed tasks for the thesis defense. For the cells without color coding, no extra action need be taken, i.e., the dataset, system, and evaluation metric are ready to use. For each task, the experimental evaluation is presented in its corresponding sections.

Table 1.1: Overview of the completed work for thesis proposal (yellow cells) and the newly added work for thesis defense (dark green cells).

	Experimental Evaluation			Approaches		
Task	Dataset	System	Metric	State of the Art Models		Proposed Model
TREC QA	TREC9 to TREC12	OpenEphyra-based QA system [77]	TOP@1, MRR@5, Recall@10	Independent Prediction Model [45]	Generalized Watson Rank [32]	Phased Ranking Model
TREC Genomics	TREC 2006	BioQA system [102]	PsgMAP, DocMAP	CSE framework [102]	Generalized Watson Rank [32]	Phased Ranking Model

Chapter 2

Introduction

2.1 Information Systems

An information system transforms and transmits system input from one form to another. More specifically, information retrieval systems narrow the search space from large data collection to small data span. To accomplish this task, a typical question answering system has three main steps: question analysis, passage retrieval and candidate extraction.

Many information processing solutions are designed as sequential pipelines of individual components, where the output of one component provides the input for the next component. An essential weakness of such "simple" information systems is that a failure in one component (e.g., an incorrect output) is likely to lead to a failure in one or more subsequent components. One way to improve robustness and overall task performance is to implement each phase in a pipeline with multiple component algorithms; the input is processed by all component algorithms in a phase, and their outputs are merged as the final output of the phase. For example, a QA system can be sensitive to cascading failures caused by an incorrect output from some phase which results in failures in all subsequent phases. Assume that in answering TREC11 question 1396 ("What is the name of the volcano that destroyed the ancient city of Pompeii?"), the expected answer type is mistakenly detected as "City" in the question analysis phase due to an imprecise question analyzer. In the following passage retrieval phase, the system will filter out those passages that do not contain any "City" named entity; and in the answer extraction phase, candidates that do not match the "City" type will be removed. Because the question analyzer failed to select the correct answer type, all following phases will be misguided and therefore the correct answer "Vesuvius" will not be included in the answer candidate list.

To improve chances of detecting the correct answer type during the question analysis phase, one commonly used approach [67] is to employ multiple strategies instead of only relying on one strategy. An example is shown in Figure 2.1. Although the classification-based question analyzer mistakenly identifies the type of the TREC11 question 1396 as "City", the rule-based question analyzer correctly identifies the type as "Mountain", which enables the correct answer "Vesuvius" to be extracted from a relevant passage node c_{k+1}.

We defined two main characteristics of the complex information systems as follows:

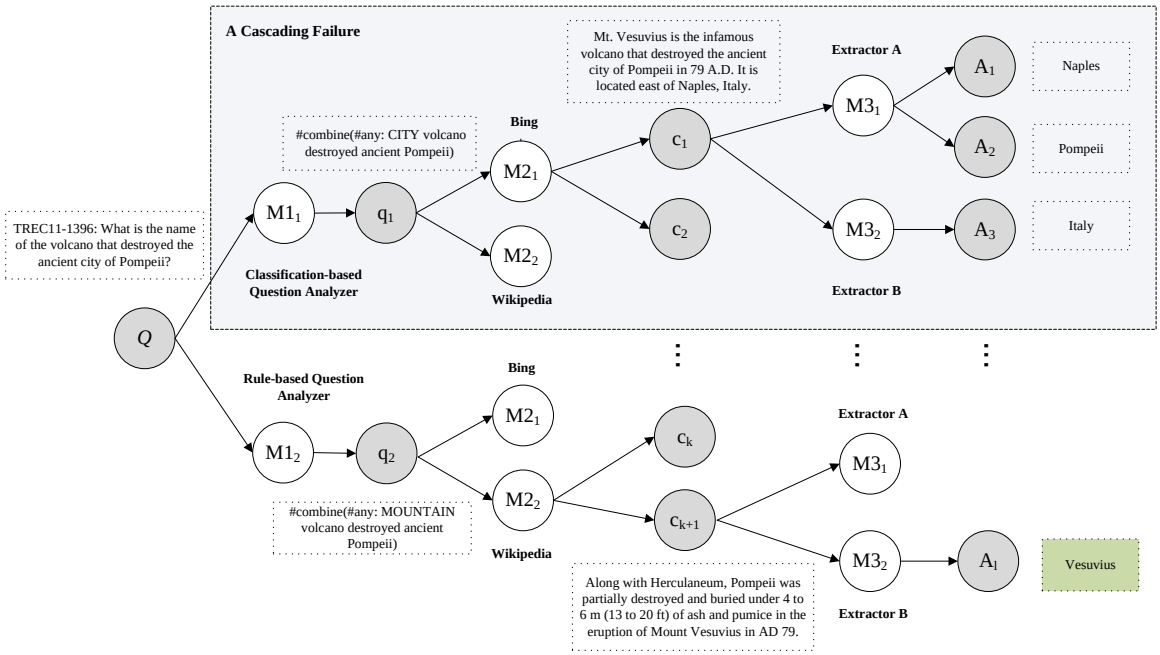

Figure 2.1: Illustration of data flow for a multi-strategy QA system.

1. **Increasing number of phases**. An information processing task is typically decomposed into subtasks processed as sequential phases where each phase takes the output of the preceding phase as its input. As an information system becomes more complex, its number of phases grows. For example, a QA system traditionally contains three sequential phases: question analysis, passage retrieval, and information (answer) extraction [67, 77]. The recent DeepQA system for the Jeopardy Challenge [28] contained a large number of phases with more than 100 different approaches for analyzing natural language, identifying sources, finding and generating answer candidates, detecting and scoring evidence, and merging and ranking answer candidates.
2. **Increasing number of parallel components**. It is a fundamental idea in computing that information systems may have various analysis phases in a processing pipeline, where each phase may admit multiple implementations that can produce alternative results [28]. This is especially relevant for complex information processing tasks where no one approach is likely to produce a satisfactory output for all inputs. For example, the development of question answering systems requires combining multiple knowledge sources from both structured databases and unstructured sources to validate the correctness of answer candidates [77], integrating multi-strategy approaches for answer extraction ranging from simple factoid database retrieval to complex machine learning-based named entity extraction [39, 67], and supporting a mixture of experts for question and content analytic [17, 28].

2.2 The Role of Ranking in Information System

Ranking is a key task common to computer-based information processing and cognitive systems (e.g., recommendation systems [1, 10, 48], information retrieval systems [5, 51, 55], Bioinformatics [74], etc.) that are required to present information pieces (e.g., answers, passages, documents) to human users in a sorted order, such that the cognitive system can present the most relevant and useful information to the users. Complex information systems are usually built as a solution to a specific information processing task for a specific domain. Those information systems adopt a pipeline architecture that incorporates multiple phases to produce analysis results. They also employ a final ranking and selection phase to distinguish correct outputs from erroneous outputs. For example, to answer questions [28], a common approach is to first analyze the question and then retrieve relevant documents and extract candidate answers. For each answer candidate, the individual pieces of evidence are scored by answer scorers by deriving evidence over its supporting resources. Finally, answer candidates are ranked according to their score.

2.3 Ranking Challenges

Traditionally, a ranking model is any function to find the sorted order for a set of outputs. In the domain of information retrieval, the ranking task is performed by using a ranking model $f(q, d)$ to sort documents, where q denotes a query and d denotes a document. For example, the BM25 model uses a bag-of-words retrieval function that assumes the independence between the query terms within a document. Similarly, the query likelihood model, a form of the conditional probability distribution $P(q|d)$, calculates rankings with the terms appearing in the query and document [18]. More recently, researchers studied learning to rank approaches under the domains of Information Retrieval [50] and Natural Language Processing [47]. Learning to rank is a supervised learning task and is often classified into three main categories: pointwise methods, pairwise methods, and listwise methods.

One of the new challenges is that information systems often suffer from the "local optimum" problem which is caused by system components designed and optimized for sub-tasks. Consider the situation that information systems typically contain many system components designed and developed by independent contributors, without awareness of the dependencies and the overall performance of the system.

Although it is possible to adapt existing algorithms, deriving the optimal ranking function or collecting evidence to support answer ranking for such complex systems can be particularly challenging. It has been noticed that the success of machine learning algorithms largely depends on other factors such as data representation (or features) [7]. As information systems become more complex, it has been more challenging than ever to efficiently design a ranking approach, in order to achieve a desired or optimal level of performance for a specific task. Thus, information systems from different domains often lack a clear and efficient way to perform ranking. This makes it difficult to adapt and tune ranking algorithms for new tasks and therefore the goal of rapid domain adaptation cannot

be achieved without a well-defined ranking framework.

As a result, solving the ranking problems for complex information systems requires more than just merely applying single ranking functions or learning to rank algorithms, instead, it is necessary to also consider both high-level system designs and low-level process execution.

2.4 Hypothesis

A multi-phase, multi-module system has the capacity to produce a A significant number of both correct and incorrect candidate results. Previous ranking approaches include a) ranking via answer-specific features (answer validation and similarity) [45], b) logic reasoning by verifying semantic relationships between the question and its candidate answers [63], and c) heuristic-based ranking approaches [54, 101]. These approaches can be less effective when multiple strategies are combined in each phase, especially when recall and precision varies greatly across the combined strategies for any given input question. Localized ranking based on heuristic features is sub-optimal when upstream sources are better sources of answer candidates, for some discernible subset of the inputs. In order to effectively merge and rank outputs from multiple components at a particular processing step, a statistically-trained ranking model must consider features that propagate information earlier phases in the pipeline. Such a ranking model must learn which combination of components produces the best output, given a particular input.

To address this problem, our proposed ranking model focuses on correlating ranking features are drawn from every phase in a pipelined system. Intuitively, an approach which attempts to leverage "global" information from every step in the answer generation pipeline has the potential to learn how to rank answers more effectively than the approach that is limited to just "local" features which are calculated over candidate answers. Our proposed approach uses a general form of feature propagation from prior phases when ranking candidate answers, and is not prone to the weakness described above. Specifically, a multi-phase, multi-strategy is represented by a system object graph, which offers a unified representation for propagation of ranking information, especially useful when training ranking functions for each phase of the system.

We hypothesize that our phased approach, using the system object graph to create new features and feature propagation for ranking, can improve answer accuracy when compared to common answer ranking approaches, which typically ignore information from prior phases or dependencies among system objects (with the possible exception of local context drawn from the answer bearing passage) [49]. In our ranking experiments, we validate our claims using multiple data sets on two different domains [1]. Results show that our approach outperforms several strong baseline models significantly.

[1]Since we used a QA system that is not state-of-the-art as the baseline system for the experiment, we measure significant improvement from that baseline, rather than absolute comparison to published TREC results. Put more simply, the goal of our research is to evaluate ranking approaches, not QA systems.

Chapter 3

Background and Related Work

In this thesis, we address the problem of phased ranking, where information systems collect and combine evidence based on a given input (question) and score each answer candidate. Phased ranking is complicated by 1) ever more complex information systems and 2) ranking challenges associated with complex information systems. Phased ranking is directly related to information system architectures in Section 3.1 and ranking frameworks in Section 3.2. For example, Watson's DeepQA project [28] built a complex information system providing answers that are accurate to the users question by harnessing the technologies of natural language processing, information retrieval, information extraction and machine learning. As the complexity and scaling of information systems become ever greater, ranking solutions are becoming more sophisticated (e.g., from some heuristic ranking functions to learning to rank frameworks) in order to achieve a desired or optimal level of performance on a given task.

3.1 Information System Architectures

Previous research [28] observed that information system architectures enabled rapid integration and evaluation during development. In this section, we study the related work of analysis stages, federated approaches used in those stages, and information system frameworks as a tool to manage system components.

3.1.1 Analysis Phases

Most factoid QA systems [36, 37, 66, 97] employ a pipeline that incorporates major phases such as question analysis, document/passage retrieval, answer extraction and answer selection. For example, in JAVELIN [66], the question analysis phase produces question metadata and representations such as keywords and relations; the document/passage retrieval phase searches for relevant documents or passages; the answer extraction phase extracts a list of answer candidates from the retrieved results; and the answer selection is a process which distinguishes the correct answer from the incorrect ones.

More complex systems such as Watson's DeepQA [26], for example, defines in a processing pipeline various analysis phases. In the question analysis phase, Watson takes a question as an input and attempts to determine the lexical answer type it is asking for. Next, it discovers candidate answers by integrating and expanding Watsons content, including selecting raw textual content, expanding the sources and analyzing the raw content for building a derived resource. In this phase Watson provides the function of the search and candidate generation to find possible answers. Then, it takes the question analysis results as input to form a variety of different queries in multiple interpretations of the question, which are sent into a variety of structured and unstructured sources using search mechanisms. From those search results, a set of answer candidates are generated. Finally, Watson collects and scores evidence for those answer candidates.

Some early QA systems are typically conceived as a "query match" approach to databases [33, 95]. Question Answering over structured data has been traditionally addressed through a deep analysis of the question in order to reconstruct a logical form, which is then translated into the structured query language of the target data [4, 69]. This approach implies a complex mapping between linguistic objects (e.g. lexical items, syntactic structures) and precompiled data objects (e.g. concepts and relations in a knowledge base). An example is the WolframAlpha "Computational Knowledge Engine" [1], which is designed to perform searches and calculations over manually curated structured data. The system has a natural-language interface, and many answers include dynamically generated data often involving mathematical computations.

Logic reasoning or textual entailment can be viewed as an additional phase in a QA's system. Moldovan et al. [62] employed a pipeline with three major phases including an answer logic form converter, a passage logic form converter, and a logic prover. In their approach, question and passages are transformed into logical representations based on their syntactic parses. The logic prover, COGEX, is leveraged to match the two logical representations. The prover performs matching between the question and answer logical representations with the aid of a large body of world knowledge axioms automatically derived from eXtended WordNet glosses.

Similarly, structural matching [43] is another add-on phase for QA systems. Structural matching approaches parse the question answer sentences from the sources and try to align their syntactic or semantic structures. The similarity of the question and sentence structures can be used as confidence estimates. Punyakanok et al. [70] represented both questions and candidate passages using dependency trees, and incorporated semantic information such as named entities in this representation. The sentence that best answers a question is determined to be the one that minimizes the generalized edit distance between it and the question tree, computed via an approximate tree matching algorithm. Cui et al. [19] proposed an approach to approximate match syntactic dependency paths. They applied dependency parser to both the question and answer sentences. A relation triple is the smallest representation of a dependency path embedded in the parsing tree of a sentence. Each triple consists of two slots and one path of relations between them: (Slot1, Path, Slot2), where slots are nouns and verbs, or named entities. To obtain similarity measures

[1] WolframAlpha, http://www.wolframalpha.com

between two paths, a statistical method is used to learn the similarity of relations from training data.

Previous work also added a phase by learning question to answer transformations using a translation model [3, 8, 23, 81]. For example, Agichtein et al. [3] learned lexical transformations from the original question (e.g., "what is a") to a set of common search queries (e.g., "the term" or "stands for"). Those transformed results were likely to retrieve good candidate documents in commercial web search engines. Murdock et al. [65] studied the problem of candidate sentence retrieval for QA and showed that a lexical translation model can be exploited to improve factoid QA. Xue et al. [99] showed that a linear interpolation of translation models and a query likelihood language model outperformed each individual model. Riezler et al. [72] developed SMT-based query expansion methods and used them for retrieval from FAQ pages.

Most existing phases are developed independently focusing on local optimization criteria, without considering the end-to-end system performance. For example, in the OpenEphyra QA system [77], the answer type is determined by a classification approach based on a set of predefined ontology (e.g., countries, animals, and food) in the phase of question analysis that only outputs one best answer type. The generated candidates that do not match the answer type are discarded. As a result, system performance suffers greatly suffered from recall problems.

3.1.2 Federated Frameworks

Merging the output of multiple component algorithms to provide a single output in a processing phase is especially relevant for complex information processing tasks where no one approach or combination of approaches is likely to produce a satisfactory output for all inputs [68].

The JAVELIN systems [67] brought together a collection of system components such as question analysis, document and passage retrieval, answer candidate extraction, answer selection, answer justification, and planning. Its Planner module was used to select and order the execution of individual components, allowing the system to dynamically generate multiple processing strategies and re-plan when the system is performing various question-answering tasks.

The architecture of Watson's DeepQA [26] defines in a processing pipeline various analysis stages where each stage admits multiple and parallel implementations that are independent and can produce alternative results. For example, Watson employs several search strategies to exploit the relationship between titles and content in title-oriented documents (e.g., encyclopedia articles) to improve search quality. When the correct answer is the title of the document that answers the question, Watson adopts a document-oriented search strategy to find documents that, as a whole, best match the question. A Document search rank and a search score are associated with each result, which is used as features for scoring candidate answers. However, for documents whose titles appear in the question, Watson adopts a passage search strategy including adapting both Indri's passage search and a Lucene's retrieval approach. The passage rank is used as a feature for scoring candidate answers extracted from that passage. The query-dependent similarity score is

combined with the query-independent scores such as sentence offsets, sentence length and number of named entities to determine the overall search score for each passage. In the last case, Watson used traditional passage search and scoring strategies [88] for the documents whose title is neither the answer nor in the question.

The idea of federated systems has been particularly relevant for the development of information systems. In NLP, such frameworks have been applied to tasks such as information retrieval [79], question answering [67], word sense disambiguation [73], parsing [75], and machine translation [91]. For example, in the area of question answering, the JAVELIN system [67] is a multi-source and multi-strategy approach question answering system that adopted a modular and extensible architecture that allows for multiple answering agents to process an input question. Some other systems combine multiple knowledge sources from both structured databases and unstructured sources to validate the correctness of answer candidates [11, 77, 78, 96], or to implement multi-strategy approaches for answer extraction, ranging from simple factoid database lookup to complex machine learning-based named entity extraction (JAVELIN [67] and CHAUCER [39]).

Although many systems allow for multiple and parallel implementations of phases, traditional approaches to multi-strategy QA systems only focus on a few phases, and do not adequately attend to the dependencies between implementations from different phases. For example, the JAVELIN system [67] focuses on multiple answering agents with multiple knowledge sources but ignores relations between phases. In a QA system, a passage retrieval implementation with geographic knowledge sources can be a better fit for the question analysis implementation with geographic ontology.

3.1.3 Information System Frameworks

Software frameworks which support integration and scaling of analysis algorithms make it possible to build complex, high-performance information systems [34]. JAVELIN systems [67] adopt a utility-based planner with a modular architecture to the control of the question-answering process. Its planner begins after an initial analysis of the input question and then solves a planning problem describing the initial information state regarding the type of the question. Finally it invokes components to maximize the expected utility of the information produced.

IBM Watson built the Unstructured Information Management Architecture (UIMA) [28] to facilitate the task of developing analytic components for complex information systems. UIMA is a component software architecture that provides a general platform for integrating system components for text, speech, and image processing. The Apache UIMA framework is an Apache licensed, open source implementation of the UIMA Architecture that supports developers to run their UIMA component implementations. There are several essential UIMA concepts listed as follows.

1. *Analysis Engines* (AEs) are UIMA's basic building blocks and are composed to perform analysis tasks. The outputs of AEs are called analysis results, typically representing meta-data about the analysis content.
2. *Annotators* are components that contain analysis used to construct Analysis Engines

by analyzing contents (e.g., web pages, images, or video streams) to create analysis results. Annotators produce their analysis results in the form of typed Feature Structures, simple data structures that have a type and a set of (attribute, value) pairs.

3. *Common Analysis Structure* (CAS) are used to represent all feature structures (e.g., use the CAS to represent a parse tree for a document), which serve as the central data structure through which all UIMA components communicate.

IBMs Watson DeepQA [28] was built based on the UIMA architecture that supports the system-level experiments that brought together multiple processing steps and analysis components.

The CSE framework [103] is another framework built as an extension to the UIMA framework. Unlike the UIMA framework, the CSE framework is designed to automatically explore the space of system configurations and determine the optimal combination of tools and parameter settings for a given task. CSE defines and manages the entire software development process with the help of Extended Configuration Descriptor (ECD), which is a YAML-based execution process description language that extends the description languages in the UIMA framework. ECDs multi-layered design separates the component-level descriptors from the pipeline-level descriptors. It provides a high level, configurable representation of information systems and supports the execution of all possible component configurations and tuning of parameters. The CSE framework is to enable automatic construction and optimization of execution processes based on ECD framework. The CSE framework with the built-in exploration strategy was first used to optimize biomedical information retrieval system for the TREC Genomics tasks over configurations of components and parameter values.

Different from this proposed work, existing information system frameworks mainly focus on system component management in terms of software engineering, or searching configuration spaces, rather than end-to-end overall performance that requires combining all the possible components of each phase in the pipeline. For example, UIMA framework is a suite of toolkits that standardize information systems in terms of a set of basic elements such as types, analysis components, and workflows; however, it does not directly facilitate the analytics task by collecting evidence from its phases to support the ranking decision. CSE framework optimizes component configurations by selecting components to form the one best pipeline rather than combining different components that have their own strengths and weaknesses.

3.2 Ranking Frameworks

A ranking framework can adopt heuristic ranking functions, learning to rank approaches or multi-stage re-rank approaches.

3.2.1 Heuristic Ranking functions

Early ranking models used heuristic ranking functions such as filtering or answer validation. Filtering [17] is often used in numerical questions or questions asking for geographic locations. The obvious wrong answers are typically filtered out from the answer candidate list. Magnini et al. [53] proposed two answer ranking functions: a statistical approach and a content-based approach. Those two approaches form a query from question keywords and an answer candidate and send it to Web Search. The former approach computes answer relevance scores by using the number of hits from the question keywords, the answer candidate, and the combination of the question keywords and the answer candidate. The latter approach counts the word distance between an answer candidate and a question keyword in each text snippet, then calculates an answer relevance score using the distance.

3.2.2 Learning to Rank

In factoid question answering, the problem of answer ranking and selection can be cast into either a classification problem or a ranking problem. Ittycheriah et al. [56] used a learning approach to find answers by maximizing the conditional probability $p(c|a, q)$, where c takes on 1 or 0 as value and denotes being correct or incorrect, and q denotes a question and a denotes an answer candidate. The distribution is used to measure the correctness c of the answer a and question q. A hidden variable was introduced to represent the class of the answer e (answer types) as follows,

$$p(c|a, q) = \sum_{e} p(c|e, a, q)p(e|a, q) \tag{3.1}$$

Where the terms $p(e|a, q)$ and $p(c|e, a, q)$ are the answer type problem and the answer selection problem. The distribution $p(c|e, a, q)$ is estimated by using the maximum entropy approach. Thirty one features are included in this approach, such as a search engine's document rank feature, sentence features and answer candidate string features.

Echihabi and Marcu [22] adapted a noisy-channel approach (IBM model 4) for the task of factoid QA. In this model, a given sentence which may contain an answer candidate a, is rewritten into the question q through a sequence of stochastic operations. Given a corpus of question-answer pairs, a probabilistic model is trained for estimating the conditional probability $p(q|a)$. Once the parameters of this model are learned, given a question and the set of sentences $\mathbf{S}$ returned by an IR engine, this approach tries to find a sentence $s_i \in \mathbf{S}$ and an answer a_j in it by searching for the (s_i, a_j) that maximizes the conditional probability $p(q|s_i, a_j)$.

Ko et al. [45] proposed an independent prediction model which estimates the probability of an individual answer candidate using multiple answer relevance and similarity features. The model is implemented with logistic regression, which is a discriminative method that directly models $P(c|q, a)$ by learning parameters from training data as follows.

$$p(c|q, a) = \frac{exp(\alpha + \sum_{k=1}^{K1} \beta_k rel_k(a) + \sum_{k=1}^{K2} \lambda_k sim_k(a))}{1 + exp(\alpha + \sum_{k=1}^{K1} \beta_k rel_k(a) + \sum_{k=1}^{K2} \lambda_k sim_k(a))} \tag{3.2}$$

Where $K1$ and $K2$ are the number of feature functions for answer relevance and answer similarity scores respectively, $rel_k(a)$ is a feature function used to produce an answer relevance score for an individual answer candidate a, and $sim_k(a)$ is a scoring function used to calculate an answer similarity between a and all the other answer candidates.

An undirected graphical model was developed later by Ko et al. [44] to estimate the joint probability of the correctness of all answer candidates, given the pairwise similarity between each pair of answer candidates and the relevance score for each candidate. A Boltzmann machine was adopted for this approach, where each answer candidate A_i is a node S_i associated with its binary value represents answer correctness. As each node has a binary value (either 0 or 1), this model uses the answer relevance scores only when an answer candidate is correct and uses the answer similarity scores only when both answer candidates are correct. This prevents the biased influence of incorrect similar answers.

Learning to rank methods have been effectively applied to information retrieval tasks. These methods focus on learning a model to find the appropriate document ranking according to their relevancy. Some recent work in learning to rank have also been applied to QA tasks. For example, Agarwal et al. [2] applied different learning to rank approaches including three main categories: pointwise methods, pairwise methods, and listwise methods, as follows.

1. Pointwise methods treat the ranking problem as a standard classification or a regression task.
2. Pairwise methods like FRank [89], SVMRank [42], RankNet [13], RankBoost [31] aim to learn the pairwise preference of candidate answers rather than their absolute rank.
3. Listwise methods operate on the entire list of candidate answers. In listwise methods, a direct loss (an appropriate evaluation measure defined by the user) is minimized between the true ranks of the list and the estimated ranks of the list. Examples of listwise methods are LambdaRank [14], Coordinate-Ascent [57], AdaRank [98], ListNet [15] and [104].

Existing learning to rank approaches mainly focus on learning the mapping of an input vector to a member of an ordered set of numerical ranks, rather than learning feature representations about information systems. The key difference from previous work is that our proposed approach concentrates on linking the ranking problem to the phases of a pipeline. In this way, dependencies of system objects (inputs, models, and outputs) can be explored for any information processing pipeline.

3.2.3 Multi-stage Re-Rank

Re-ranking is a popular approach in multi-strategy QA systems. As answer candidates come from different agents with different score distributions, simple confidence-based voting has been used to merge the top five answers returned by multiple QA agents [17]. As a more advanced approach, a maximum-entropy model has been used to rerank the top 50 answer candidates returned from three different answering strategies [24]. This model improved the performance of answer selection by combining complementary results provided by different answering strategies.

Burger et al. [12] applied ensemble methods to combine the 67 runs submitted to the TREC 11 QA track, using an equal-weighted centroid method for selecting among the 67 proposed answers for each question. Agarwal et al. [2] used two classical rank aggregation techniques including Borda and Kemeny to aggregating rankings of rankers. In Borda aggregation, each ranker assigns a score to each candidate, which is the number of candidates below it in each rankers preferences. The Borda aggregation is the descending order arrangement of the average Borda score for each candidate averaged across all ranker preferences. A Kemeny optimal aggregation is an aggregation that has the minimum number of pairwise disagreements with all rankers, i.e., a choice of τ that minimizes

$$K(\tau, \tau_1, ..., \tau_r) = \frac{1}{r} \sum_{i=1} rk(\tau, \tau_i) \tag{3.3}$$

where the function $k(\tau, \tau_i)$ is the Kendall tau distance measured by the agreements between any two rankers.

Watson [32] used a seven-stage ranking framework for selecting the best answer candidates. Those phases in Watsons for Jeopardy! are summarized as follows:

1. *Hitlist Normalization* Answers begin at the Hitlist Normalization phase. At the start of this phase, answers whose string values are identical are merged. After merging, the standardized feature values are computed. The answers are then sent to one of a set of routes based on how the question was classified.
2. *Base* At the start of the base phase, answers are ranked according to the scores assigned by the Hitlist Normalization phase, and only the 100 highest scoring answers from that phase are retained. With at most 100 answers per question, the standardized features are recomputed. The base phase provides the same set of routes as the previous phase.
3. *Transfer Learning* The score assigned to each answer by the model in the base phase is provided as an input feature to the Transfer Learning phase. In the transfer learning phase, last phases output is used as a feature into a specialized model. In the case of logistic regression, which uses a linear combination of weights, the weights that are learned in the transfer phase can be roughly interpreted as an update to the parameters learned from the general task.
4. *Answer Merging* The Answer Merging phase combines equivalent answers. It uses a variety of resources to determine equivalence. It uses the results of the previous phase to determine which of the various forms being merged is canonical. In other words, Among the identical answers, a canonical form is selected by considering the evidence score.
5. *Elite* The elite phase works similar as the base phase but employs only the top five answers from the previous phase.
6. *Evidence Diffusion* The Evidence Diffusion phase is similar to the Answer Merging phase but combines evidence from related answers instead of equivalent ones.
7. *Multi-Answers* The Multi-Answer phase provides one additional type of merging that is only invoked for questions that ask for a list of answers; it combines multiple

answers from earlier phases into a final answer.

So far as we know, there has been little research on an object ranking model for a general information processing framework that allows any algorithm to be easily incorporated into any phase of an information processing system. The Watson system [28, 32] incorporated a multi-strategy, phased approach to ranking candidate answers, but did not present a general approach to object ranking that can be applied to any set of information processing steps. Our ranking approach models the information propagated from preceding phases to inform the ranking decision. Our approach is motivated by existing multi-strategy QA systems [17, 67, 77] and proposes a general framework that can be used to integrate an arbitrary number of phases, each phase integrating multiple strategies.

Chapter 4

Representing an Information System as a System Object Graph

Given an information system for a specific domain, how do we perform ranking of a set of outputs such that "good" ones come before "bad" ones? Similar to the idea "You shall know a word by the company it keeps", we hypothesize that we shall know an output by the derivation paths that produced it. In other words, evidence from the derivation paths can provide important ranking information to compare various outputs. Consider the simple phase which has two modules M1 and M2, shown in Figure 4.1. Given the same input I, the output O1 is derived from the path I→M1→O1 and output O2 is derived from a different path I→M2→O2. It is possible that O1 is to be ranked higher than O2 because M1 generally produces more accurate results than M2 does. Since each output can have its unique derivation path, it is natural and intuitive to define a *system object graph* to represent the derivation path space. To perform ranking, we introduce the derivation path to capture the important characteristics of a output to distinguish from the other outputs.

This chapter gives the formal definition of a system object graph, a generalized, unifying representation for the information system. Typically, an information system consists of a number of processing steps arranged in series, and each component is described by its input(s) and output(s). All complex information systems that consist of multiple processing steps can be reduced to the system object graph representation described here. What is perhaps most distinctive about the system object graph is its naturalness in formulating models of complex phenomena in information systems.

4.1 What is a System Object Graph?

The objects contained in a system object graph can be roughly distributed into five groups: system input (the initial form of information need such as a question or a query), system phases (critical steps of data processing and analysis), system modules (various concrete implementations of system phases), internal outputs (the output of each phase); system output (the desired information such as correct answers to a question or relevant documents to a query). An output can be associated with a combination of original input,

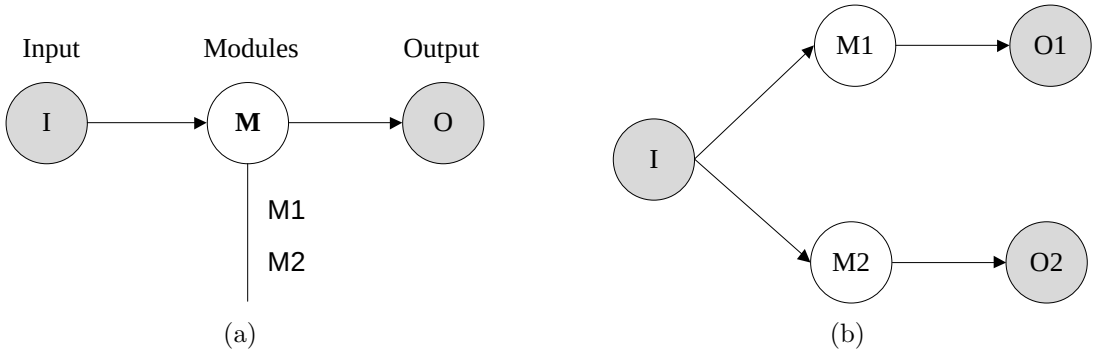

Figure 4.1: (a) is a simple example phase, which has a set of input I, a set of output O and two modules M1 and M2. (b) shows two derivation paths for outputs O1 and O2 produced by (a).

internal outputs, and modules of phases. These variables and their relations can be further abstracted as a derivation path in a pipeline.

Since various outputs can be generated, the set of all derivation paths comprise the system object graph. We formally define the system object graph and the derivation path as follows.

Definition 1 *A **system object graph** $\mathcal{G} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ is a graph that is constructed by a set of sub-graphs or phases from the first phase $\mathcal{P}_1$ to the n-th phase $\mathcal{P}_n$. From a system object graph, a **derivation path** for an output O is an instantiated execution path γ: $[\mathcal{P}_1(O), \mathcal{P}_n(O)]$ where $\mathcal{P}_1(O)$ is the initial phase, $\mathcal{P}_n(O)$ is the final point.*

In general, a system object graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a graph, where $\mathcal{V}$ are the nodes and $\mathcal{E}$ are the edges of the graph. Each node $v \in \mathcal{V}$ or each edge $e \in \mathcal{E}$ belongs to a particular sub-graph named a phase.

In Figure 4.2, we present the system object graph which describes a sequential-phased information system. Although parallel phases are entirely possible in a graph, the special distinguishing feature of this system object graph example is that all its nodes are simplified to be arranged in sequential order from left to right. The input object **I** can be a question or a query. Each phase takes the output from the previous phase as input. The input will be processed though multiple modules **M** per phase, in parallel, to generate a candidate set of results **R**. This process ends at the T-th phase which generates the final output **O**. A phased ranking model may be used at the end of a phase to collect and merge all the results generated by different modules.

For example, consider a QA system which has a pipelined architecture that contains sequential processing phases. Question analysis, passage retrieval, candidate extraction, answer ranking comprise the system object graph for a typical four-phase answer generation task in a QA system. The first phase in the pipeline, question analysis, classifies the input question into one of the predefined categories, and represents the information need of the given question. The information need is then passed to the phase of passage retrieval to retrieve a set of relevant passages. Those retrieved passages are supplied to

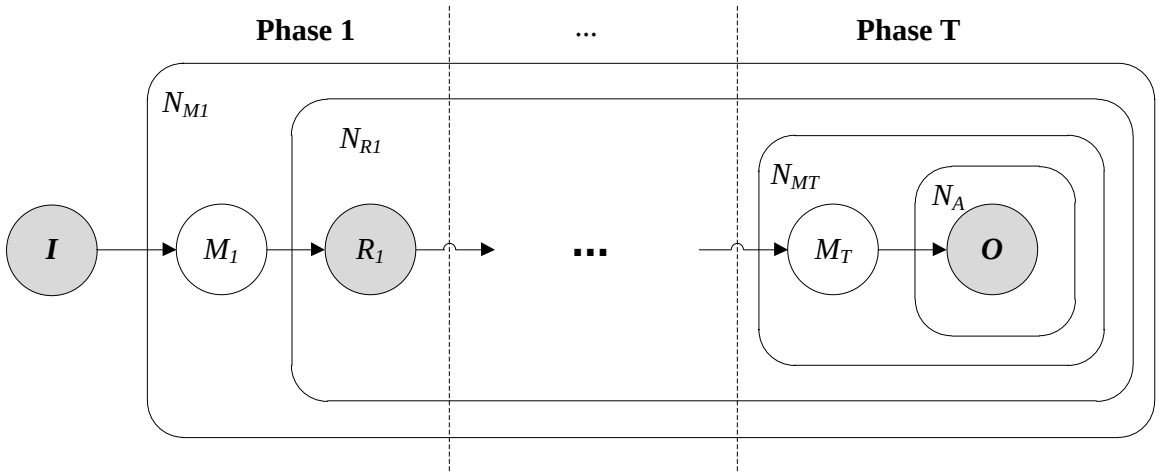

Figure 4.2: A system object graph for a general information system. Nodes are linearly arranged by processing order.

the candidate extraction phase to generate possible answers, namely answer candidates. In the third phase, the system detects named entities as answer candidates by using multiple information extractors, and finally those answer candidates are merged together and ranked as the final answer list.

4.1.1 Phases

We introduce the term *phase* to capture each important execution step of an information system. The initial phase starts with processing the system input, which will be processed by multiple modules in parallel to generate outputs. The following phase takes the outputs from the previous phase as its own input, and then process the input to produce the output of this phase. The similar process continues until the final phase ends with producing the system output as the final results.

Definition 2 *A **phase** $\mathcal{P} = (\mathcal{V}_I, \mathcal{V}_M, \mathcal{V}_O, \mathcal{E})$ is a graph representing a processing step of an information system, where $\mathcal{V}_I$ is a set of input nodes, $\mathcal{V}_M$ is a set of module nodes, $\mathcal{V}_O$ is a set of output nodes, $\mathcal{E}$ is a set of direct edges denoting dependencies for the phase $\mathcal{P}$.*

The idea of a phase is shown in Figure 4.3. Specifically, we let a node O be the input of the phase. The input node is sent into the parallel modules from M_1 to M_N. Those modules process the data and generate the output nodes from O_1 to O_K. In this graph, we denote the input and the output by shading the corresponding nodes to distinguish them from module nodes. A plate is used to capture the replication of the nodes for the same phase as an alternative representation, as shown in Figure 4.4.

As an example, consider the task of Named Entity Recognition as a t-th phase of an information system, where the input I is a tuple (entity type, document) and the output $\mathbf{O} = O_1, ..., O_K$ is a list of named entities that match the entity type. A module M_1 could be a rule-based named entity extractor, M_2 could be a CRF-based named entity extractor,

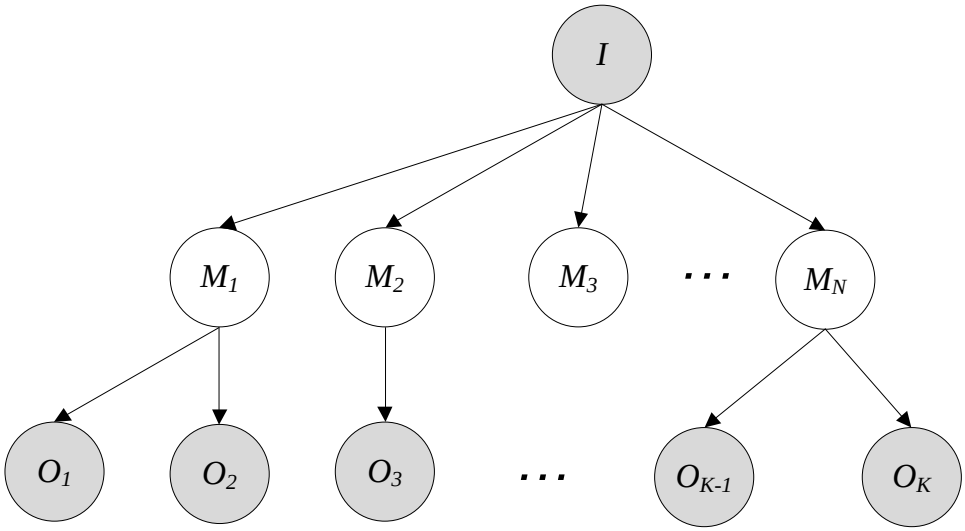

Figure 4.3: A graphical representation of a phase.

and M_3 could be a dictionary lookup based named entity extractor. In this way, it is straightforward to represent a task or a step of an information system as a phase.

4.1.2 Nodes

One of the basic elements of a phase is called a *node*. There are three different types of nodes $\mathcal{V} = \{\mathcal{V}_I, \mathcal{V}_M, \mathcal{V}_O\}$ according to the concept of a phase.

Definition 3 *A node v can be an input node $v \in \mathcal{V}_I$ denoting an input unit of a phase, a module node $v \in \mathcal{V}_M$ representing a module of a phase, or an output node $v \in \mathcal{V}_O$ denoting a produced result unit of a phase.*

Each node can be associated with a variable in order to model the pipelined system. For example, the variable of an output node can be described as $y = f_i^t(x, \omega)$, where x is a variable of an input node produced by previous phase, f_i^t is the i-th processing function of the t-th phase, and ω is a set of input parameters. A module node can be associated with a variable that simply indicating the status of the module.

4.1.3 Edges

Another basic element of a phase is an *edge*. We define an edge as follows.

Definition 4 *An edge is a dependency between two related nodes.*

Various dependencies can be applied to a phase and to a system object graph. We list some possible dependencies for a phase as follows.

1. Input-Module Dependency (I→M), the modules are dependent on the input.
2. Module-Output Dependency (M→O), the outputs are dependent on the module.

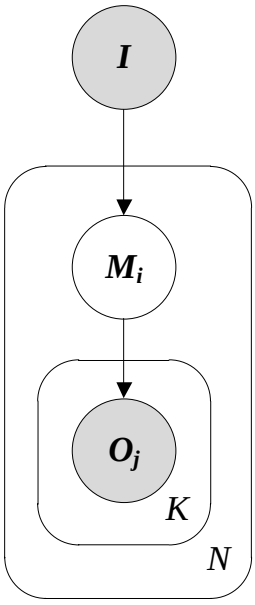

Figure 4.4: An alternative and more compact representation of a phase.

As an example, consider the phase of Named Entity Recognition. Modules are usually developed for annotating entities of the different types. It can be more appropriate to use a particular named entity extractor to annotate entities of "number" type than to use the other extractors. And therefore, a module's performance can be dependent on the input. For the same entity type, a rule-based named entity extractor often has better precision but lower recall compared with a CRF-based named entity extractor. Due to this fact, the output entities are dependent on the modules that generate them.

Chapter 5

Modeling the System Object Graph for Phased Ranking

This chapter introduces a ranking approach for an information system based on modeling a given system object graph described in Section 4.

5.1 What is Phased Ranking?

A phased ranking model is a general ranking approach for multi-phase, multi-module information systems. Many information systems incorporate multiple phases and each phase may have multiple modules with various algorithms or methods to achieve acceptable robustness and overall performance. Such systems may produce and rank a large number of candidate results. Prior work [45] includes models that rank a particular type of information object (e.g. a retrieved document, a factoid answer) using features specific to that information type, without attempting to make use of other non-local features (e.g. features of the upstream information source), and therefore may not be effective enough for an information system. A phased ranking model is a supervised classification that allows each phase in a system to leverage information propagated from preceding phases to inform the ranking decision. This is accomplished by modeling a system object graph which represents all of the objects created during system execution, object dependencies (e.g. provenance), and ranking feature values extracted for a specific object.

5.2 Ranking Framework

As described in Section 4, the system object graph consists of a sequence of phases from $\mathcal{P}_1$ to $\mathcal{P}_T$, where each phase $\mathcal{P}_t$ contains module nodes $\mathbf{M}$ which produce a collection of output nodes $\mathbf{O}$. In this way, it is easy to trace the derivation path of an output node at any phase, and those different output nodes in the same phase can be compared by collecting evidence from their derivation paths. As well, those similar output nodes produced in the same phase provide additional evidence, for the reason that relevant information may be presented in different ways and tend to be redundant [52]. To rank output nodes, the

above notions are captured by three steps. First, we apply a cascade model to increasingly collect ranking features of each phase. Second, we use a voting model to merge and score similar output nodes. Finally, we learn a ranking model by jointly considering the cascade model and the voting model. We now describe each aspect of the proposed phased model approach in more detail.

5.2.1 Problem Definition

We formulate the ranking problem through the definitions given below.

Definition 5 *The ranking problem for the t-th phase of a system object graph is to find an ordering ω of a set of output nodes $\boldsymbol{O}$ produced by the t-th phase such that any k-length prefix of $\omega(\boldsymbol{O})$ is the best possible k output nodes.*

Since our goal is to generate an ordering per phase, a ranking function should be defined to map the output nodes to the ordering.

Definition 6 *A phased ranking model for the t-th phase is defined as a function*

$$\omega = f(\boldsymbol{O}, \mathcal{P}_1, ..., \mathcal{P}_t, \boldsymbol{I}) \tag{5.1}$$

where $\boldsymbol{I}$ is the input set and $\mathcal{P}_i$ is the i-th phase, $1 \leq i \leq t$.

The above ranking function f is used to map a set of output nodes $\mathbf{O}$ to the position vector $1, 2, ..., N$. In other words, the proposed model is a ranking model f: $\mathcal{O}^d \mapsto \mathcal{O}$ such that the ranking order of a set of output is specified by the real value that f takes, specifically, $f(O_i) > f(O_j)$ means that the model asserts that the i-th output node O_i is more relevant than the j-th output node O_j. Usually, the input I and phases $\mathcal{P}$ can be ignored from Equation 5.1, if the output nodes are produced by the same phase.

Definition 7 *The phased ranking's learning problem is defined as a process of learning the phased ranking model f from the given training instances $\{\omega^i, O^i, \mathcal{P}_1^i, ..., \mathcal{P}_t^i, I^i, \omega^i\}$, where O^i is the output nodes, $\mathcal{P}^i$ is a phase and ω^i is the ground-truth label corresponding to the i-th input I^i. The learning process can be formulated as the process of minimizing the loss function*

$$f^* = \underset{f}{\operatorname{argmin}} \sum_i \Delta(\omega_i, f(O^i, \mathcal{P}_1^i, ..., \mathcal{P}_t^i, I^i)) \tag{5.2}$$

where f^* is the ranking function estimated from training instances by minimizing the loss function, Δ means the loss between the ground-truth ω^i and the prediction $\hat{\omega}^i = f(O^i, \mathcal{P}_1^i, ..., \mathcal{P}_t^i, I^i)$.

The phased ranking problem description leaves several issues to be addressed more explicitly for resolving the cost minimization problem in Equation 5.2. First, a function definition of the ranking model f in Equation 5.1 is needed. Second, the loss function needs to be designed carefully. Third, the learning algorithm should be specified and the optimization problem should be solved efficiently. In this chapter, we will provide a function definition of the ranking model $f(O^i, \mathcal{P}_1^i, ..., \mathcal{P}_t^i, I^i)$. In next chapter, we will provide a general framework to resolve the rest implementation issues in order to apply the phased ranking model to an information system.

5.2.2 Cascade Model

We describe a cascade model that uses features derived from each phase of the system object graph to produce a normalized score for comparing different output nodes.

In general, the cascade model consists of a sequence of feature sets $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_T\}$, where each feature set $\boldsymbol{\phi}_t$ is extracted from the t-th phase of a system object graph. The cascade model for an output node O from the T-th phase therefore has the following form:

$$r(O_T) = \sum_{t=1}^{T} \Lambda_t \boldsymbol{\phi}_t(O_T) \tag{5.3}$$

where $\boldsymbol{\phi}_t(O_T)$ is a set of feature functions scoring O_T at the t-th phase, and Λ_k is the weight vector to be learned.

The cascade model includes three kinds of features: a) features computed for a module in a current phase (Model Object Features), b) features computed for an output of the current phase (Output Object Features), and c) features which propagate information from preceding phases which can be useful for ranking a current phase's output objects (Dependency Features). The most important innovation of the proposed model is the inclusion of Dependency Features, along with a means to automatically compute their weights from training data; these features make it possible to learn when performance of a module in the current phase depends on the nature of of the original input, a preceding module or a prior output object. When applied in a solution where multiple modules are used in a given phase, these features make it possible to learn when one module's output should be preferred over that of another, given their derivation graphs and features extracted from them.

5.2.3 Relevance Voting

Previous work [44] shows that it is advantageous to estimate the probability of answers in a joint manner; however, the empirical results reported in [44] were limited to the case where only 10 answer candidates were considered per question (to limit complexity of the required computations over a joint graphical model). Such a constraint is too limiting for complex systems that produce thousands of candidate results. To overcome this limitation, we describe a relevance voting approach to model the relationship between output objects. The relevance voting approach is based on two hypotheses. First, the probability that an output object is correct is dependent on the output object's relevance and the relevance of its similar neighbors. Second, the relevance derived from two text contexts for an output object is independent (e.g., the fact that "Vesuvius destroyed the ancient city of Pompeii" can be found in many documents, but those documents are unlikely to be connected).

We assume the graph $\mathcal{G}$ consists of n output objects $\mathbf{O}_T$ from the T-th phase. The relevance of all the output objects from $r(O_{t1})$ to $r(O_{tn})$ which are known variables modeled by Equation 5.3. We define the conditional probability over the random variables in $\mathcal{G}$ as

$$P(O_{ti}|r(O_{t1}), ..., r(O_{tn})) = \frac{1}{Z}\exp\left\{\sum_{k=0}^{n} w_{ki} r(O_{tk})\right\} \tag{5.4}$$

where n is the total number of the output objects produced in the t-th phase, $Z = \sum_{i=0}^{n} \exp \sum_{k=0}^{n} w_{ki} r(O_{tk})$ normalizes the distribution, and w_{ki} is the weight given from O_{tk} to O_{ti}, indicating how much relevance from node O_{tk} can be transferred to node O_{ti} in the current phase.

Output similarity functions can be defined to decide the weights $\vec{w}$ (Equation 5.4). For example, in the answer ranking stage, we define the weight of relevance transferred from A_i to itself as 1. If two answer candidates A_i and A_j represent different concepts, then they are mutually exclusive answer candidates and their transfer weights to each other should be 0. If we detect that they may represent the same concept or related concepts, their weights can be proportional to the similarity. Based on a similarity function, we can replace w_{ki} with $\mathrm{Sim}(O_k, O_i)$ in Equation 5.4. In this paper, we define the similarity function as

$$\mathrm{Sim}_{k \neq i}(O_k, O_i) = \begin{cases} 1 & \text{if } O_k = O_i, \\ 0 & \text{Otherwise.} \end{cases} \tag{5.5}$$

where O_k equals O_i implies identical surface text. There are two advantages when using this similarity function: a) near-redundant information can provide stronger evidence when identifying the most likely output object, and b) by merging redundant output objects, the final ranked output object has higher value when there are multiple correct answers.

5.2.4 Output Ranking

Given both the cascade model in Section 5.2.2 and the propagation model in Section 5.2.3, we now turn to the problem of answer ranking by integrating the two models.

The cascade model defined in Equation 5.3 is a linear model consisting of m features from the three perspectives: the module objects, the output objects, and the dependencies. We rewrite the Equation 5.3 as

$$r_t(O_t) = \sum_{j=1}^{m} \lambda_j f_j(O_t) \tag{5.6}$$

where f_j is the feature function extracted for ranking output objects and λ_i is the weight given to that particular feature function. Substituting this back into Equation 5.4, we have the following ranking function,

$$\begin{aligned} \mathrm{In}P(O_{ti} | r(O_{t1}), ..., r(O_{tn})) &\propto \sum_{k=1}^{n} w_{ki} \sum_{j=1}^{m} \lambda_j f_j(O_{ti}) \\ &= \sum_{j=1}^{m} \lambda_j \left\{ \sum_{k=1}^{n} w_{ki} f_j(O_{ti}) \right\} \end{aligned} \tag{5.7}$$

Equation 5.7 suggests that instead of directly estimating relevance $r_t(O_t)$, we can learn the ranking model by merging output objects through aggregating the features of similar output objects: for real valued features, we sum up the values; for binary features, we

define this sum as the "and" operator. In this way, we are able to jointly optimize output object relevance and output object similarity by using a learning to rank approach.

For the T-th phase, the output object set $\mathbf{O}_T$ is ranked according to the ranking score of each output object, and the top ranked output objects are selected as the final output. In particular, during final answer selection, the top ranked answer candidate would be selected as the final answer (and final system output).

5.3 Feature Models

If an output object O_t is produced at the t-th phase, its generation path can be represented as $I \rightarrow M_1^{O_t} \rightarrow O_1^{O_t} \rightarrow ... \rightarrow M_T^{O_t} \rightarrow O_t$. Using this path, we can estimate the trustworthiness of a module, and also collect evidence for the correctness of an output object. These dependencies can be used to model the relationship between the generation path and the likelihood of a correct output object.

Based on those intuitions, features are extracted from the three types of objects associated with each phase: the module objects, the output objects, and their dependencies. We combine the module object feature $\boldsymbol{\phi}_M$, the output object features $\boldsymbol{\phi}_O$, and the dependency features $\boldsymbol{\phi}_D$ into a single feature vector $\boldsymbol{\phi}_t = [\boldsymbol{\phi}_M|\boldsymbol{\phi}_O|\boldsymbol{\phi}_D]$ at each phase S_t for the cascade model. Now we describe the details of features derived from the different types of objects in the generation path.

5.3.1 The Module Object Perspective

From the module perspective, features use evidence from the module objects in a phase to model the likelihood of a correct output. The model features are binary-valued features which indicate which model(s) an output object is generated from. For example, given the TREC11 question 1496 "What country is Berlin in?", assume a Wikipedia-based search agent retrieves the passage "1936 Summer Olympics held in Berlin, Germany", and a Twitter-based agent returns the passage "McDonalds in Berlin sells McRib all year and serves beer! I thought America was the greatest country!". In this case, we may expect a ranking model to learn that passages retrieved from Wikipedia are more highly weighted than passages retrieved from Twitter, when used as a source of answers for geographical location questions.

5.3.2 The Output Object Perspective

From the output object perspective, features are derived from output objects with prior knowledge, reflecting how likely a decision (or output object) in a phase can lead to a correct output object. Two types of features can be extracted depending on whether the output objects are drawn from predefined categories. If the feature values are drawn from predefined categories (e.g., detected answer types), we use a binary vector to indicate the category value(s). For example, consider TREC11 question 1396 ("What is the name of the volcano that destroyed the ancient city of Pompeii?"); during question analysis,

the answer types "Location", "Mountain" or "Volcano" might be detected. If an output object is not drawn from predefined categories, we represent an output object via suitable relevance features. For example, when retrieving documents from a web search engine, useful information about each document returned (bag-of-words, rank position) can be normalized and represented as a feature value.

5.3.3 The Dependency Perspective

We have defined the features from both independent module object and output object perspectives based on the assumption that objects in the system object graph are conditionally independent. However, the relations of those objects are not considered. In previous research, relations between nodes are heuristically selected and modeled. For example, in a factoid QA system, the correct answer text is likely to appear in close proximity to at least some of the question keyterms. The correctness of an answer is estimated using a question keyterm matching approach based on web search [45, 52]. The relationship between question and an answer text is specific to QA.

In order to provide a more general solution for information systems, we model the relationships between nodes as dependencies. Features are extracted from those dependencies of a system object graph. We refer to this problem as *dependency feature extraction*, and address two main topics:

1. How can we formally define a dependency to capture the various relations of module objects and output objects?
2. Given those relations, is it possible to formally define the feature types that can easily create a standard set of features for ranking?

To address the first issue, we define eight relations between various types of objects (I, M, O) in a system object graph. An overview of the dependency structures is shown in Table 5.1. Each dependency structure is represented as $A - B$, which means that the variable A and B are related. Although it is entirely possible to define some other structures, in this paper we only consider length-2 dependency structures (in the rest of the paper we will use the term "dependency" as an alternative of "dependency structure").

To address the second issue, we define three possible types of feature functions for each dependency, according to the type of variables of a dependency.

Type I. We extract a binary valued vector if both the variables within a dependency are drawn from predefined categories. One such example is the dependency $M' - M$, where variable M represents a module object drawn from current phase and variable M' represents a module object drawn from previous phase, the total number of features is $|N_M| \times |N_{M'}|$.

Type II. We can estimate the similarity between two objects as a ranking feature, if neither of the objects is drawn from predefined categories, or there are dependencies between input and output objects that we want to model. One example is the dependency $I - O$. The similarity can be estimated between the input object (I) and the document or passage (O) retrieved. Similarity functions can be defined or chosen from standard approaches.

Table 5.1: Dependency structures of a general system object graph.

Name	Description
$I - M$	Dependency between current phase module objects (M) and input question (I)
$M' - M$	Dependency between current phase module objects (M) and module objects (M') from a preceding phase
$O' - M$	Dependency between current phase module objects (M) and output objects (O') from a preceding phase
$I - O$	Dependency between current phase output objects (O) and input question (I)
$M - O$	Dependency between current phase output objects (O) and its module objects (M)
$M' - O$	Dependency between current phase output objects (O) and module objects (M') from a preceding phase
$O' - O$	Dependency between current phase output objects (O) and output objects (O') from a preceding phase
$O - O$	Dependency between output objects of current phase

Type III. Dependency features can be a real valued vector if only one of the two variables within a dependency is drawn from predefined categories. One such example is the dependency $M - O$, an relevance score (e.g., tf.idf) could be estimated as a variable for an object O. Those relevance scores are required to be calculated in advance to associate with the object O. Choices of those scores can be the output object features of O and dependency features $I - O$. In this case, the number of features for the relationship $O - M$ should be $|N_M|$.

Chapter 6

Case Study: Question Answering

In this chapter we describe how to apply the phased ranking model to the task of question answering for answer ranking. Among question answering tasks in language processing, answer ranking has received much attention, with the development of standard evaluation datasets and extensive comparison among methods. The phased answer ranking approach allows answer ranking in a QA system to leverage information propagated from preceding phases, specifically, question analysis, passage retrieval and candidate extraction, to inform the ranking decision. Experimental results show that our proposed approach significantly outperforms comparable answer ranking models.

6.1 Task Description

The factoid question answering task requires systems to return exact answers drawn from a large corpus of newspaper articles in response to open-domain questions in a natural language such as TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?" We follow the standard evaluation [92] which is in an automatic way that each response was judged by the answer key "Vesuvius"; a response is marked correct if an answer to the question matches the answer key exactly. In contrast, returned text that cannot match the string pattern of the answer key is simply judged to be incorrect. The possible outcome of a judgment may include the following two cases:

1. **Correct**: the returned answer string matches exactly the answer key in the form of regular expression.
2. **Incorrect**: the returned answer string does not contain a right answer, the answer contains a right answer but more than just the answer.

6.1.1 Evaluation metrics

All our results are measured in TOP@1 [26], MRR@5 [92], and Recall@10 [26] based on answer keys. Our TOP@1 measure is the number of top 1 correct answers divided by the total number of questions. MRR5@ is the average of mean reciprocal rank of the top 5 answers. Recall@10 or "candidate binary recall at top 10" is computed as the percentage of

Table 6.1: The TREC questions used in the application.

TREC 9	TREC 10	TREC 11	TREC 12
693	500	500	413

questions for which the correct answer is generated as a candidate at top 10. In particular, they are defined as follows,

$$\text{TOP@1} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\mathcal{I}(\text{Correct answer @ 1})$$

$$\text{MRR@N} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

$$\text{Recall@N} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\mathcal{I}(\text{Correct answer @ N})$$

6.1.2 Dataset

The proposed method was evaluated on factoid questions from TREC 9 through TREC 12. The questions and corresponding answer keys can be obtained from the NIST website. We use the same experimental settings presented in [76], in that NIL questions without answer keys are removed. Often there is more than one acceptable answer for a question. Since our goal is to measure and compare the learning capacity of different models, we only use the answer keys found in the NIST website as the acceptable answers to train and test our models. The number of questions for those TREC tasks is summarized in Table 6.1.

6.2 System Overview

A question answering system provides specific answers to questions posed by users in human language. For example, consider the TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?" asked by a user who may be interested in history. The QA system may reply Vesuvius, Italy, Naples, etc. There is no doubt that existing search engines, with Google at the top, have remarkable capabilities to answer the question. In terms of input, a QA system supports natural language questions as opposed to search engines simple and short queries. With respect to output, QA systems answer users question by using concise pieces of text that are accurate to the point. In contrast, traditional search engines typically return ranked lists of relevant documents, which the user must then read to locate and justify the information he or she was looking for.

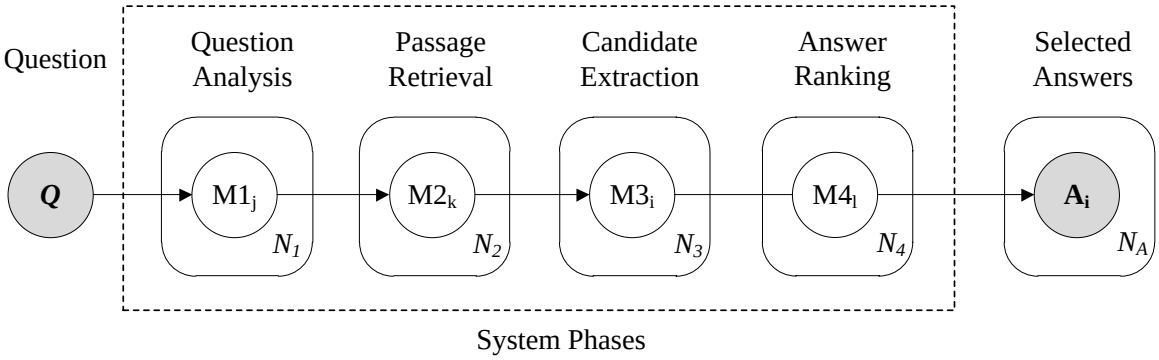

Figure 6.1: A typical QA system's architecture.

Pipelined QA systems have similar architectures according to previous work [67]. A QA system typically contains four main components consisting of question analysis, document or passage retrieval, candidate extraction and answer ranking. Figure 6.1 shows such a simplified QA system pipeline. To represent the information need of the given question, question analysis derives lexical, syntactic and semantic information. The information need is then fed into the phase of passage retrieval, which composes queries to perform passage search. Those retrieved passages are supplied to the candidate extraction phase to generate answer candidates. Finally, the answer ranking phase ranks and selects correct answers from the set of answer candidates with machine learning algorithms.

To compare the performance of our model with previous work on answer ranking, we built a baseline QA system using existing software (OpenEphyra [1] and some other available resources). In this section, we provide details of the baseline QA system.

6.2.1 Question Analysis

In the question analysis phase, a variety of core approaches are applied to the question to represent information needs. First, the question string key terms and phrases are identified. Those key terms and phrases are then used to compose queries for search engines. Function words or stop words that carry little semantic information (e.g., auxiliary verbs) are removed from the original question. Common types of phases, such as person, location, and date, are detected by using named entity recognizers. Second, questions are analyzed and represented by syntactic and semantic information. Features extracted from syntactic and semantic information are used to perform the answer type classification task.

A key function of the question analysis phase is answer type classification, which assigns the expected type of the answer based on a predefined set of types. For example, the TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?" seeks an answer that is a name, a location, more specifically a volcano. Answer type information is further used to filter out the answer candidates that do not match the type. Similar to the above example, many questions can be assigned to a set of types, from general to specific. And therefore, ontologies are constructed with hierarchy

[1] OpenEphyra, http://www.ephyra.info/

Table 6.2: Queries for the simple query "orange juice".

Query Type	Search Type	Query Example
BOW	Web Search	orange juice
Dependency	Indri Search	#weight(0.8 #combine(orange juice) 0.15 #combine(#1(orange juice)) 0.05 #combine(#uw8(orange juice)))

types to cover the requirements of the open domain. It is common to manually create such ontologies to support TREC QA tasks [64].

We adopted three different answer type classifiers. The first one is regular expression based approach, where each regular expression rule is associated with a type in the predefined ontology set and a confidence score for the rule. Another classifier is a statistical learning based approach, where a set of features are extracted and a model is learned to combine those features for classification.

6.2.2 Passage Retrieval

Given the question analysis results, the passage retrieval phase generates queries and retrieval relevant passages from indexes of unstructured documents and web search agents. Those retrieved passages are used to identify answer candidates.

The query generation step transforms a natural language question into a query that can be understood by search engines. For local indexes, we adopt the sequential dependency model [59] as our query formulation method by considering the trade-off between effectiveness and efficiency. The sequential dependency model considers both the unigrams and bigrams in the given question. Using Indri query language, the unigrams and bigrams from the initial question and proximity matches are assigned different weights. Table 6.2 shows an example for the TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?". The above parameters can be turned by using training set.

For a local document index, Indri [84] is used to perform passage retrieval [2]. For a web search agent, top K results are preserved and the rest results are discarded.

6.2.3 Candidate Extraction

In our system, the search results are passages, which are typically shorter than documents. Different approaches have been proposed for extracting factoid answers from the retrieved passages. For the approach of type based candidate extraction, expected answer types detected in the question analysis phase are used to guide the candidate extraction phase.

[2]Indri uses a query likelihood function with Dirichlet prior smoothing to weight terms by default. Query language model is a generative model which rank documents by $P(d|q)$, the probability that a document is relevant to the query q. Typically the retrieval ranking for a query q under the language model is based on Unigram assumption with the maximum likelihood estimator. This model is smoothed by background collections with Bayesian smoothing using Dirichlet priors.

Those answer types are determined based on a predefined ontology. We use multiple named entity recognizers that can identify instances of the expected answer type from the retrieved passages. For example, the expected answer type for the TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?" can be "Location". And therefore, given the retrieved passage, "Mount Vesuvius is best known for its eruption in AD 79 that led to the burying and destruction of the Roman cities of Pompeii and Herculaneum.", possible named entities or answer candidates that match the expected answer type can be "Mount Vesuvius", "Pompeii" and "Herculaneum". Since different named entity recognizers may have different answer type coverage, we also use regular expressions for answer types such as numbers, dates, and quantities.

6.3 Features for Answer Ranking

In Section 5.3, we described the general feature templates of the phased ranking model from three perspectives with nine feature types (one for module object, one for output object and seven for object dependencies). In this section, we will detail the implemented features per phase by using those feature types. Those phases are question analysis (S_1), passage retrieval (S_2), and answer extraction (S_3). We will use M to denote a module object of the current phase and M' to denote the one from the previous phase. An output object is denoted by O and the original question is denoted by Q.

6.3.1 Question Analysis Phase Features

The module object features (M_1) We use binary-valued features to indicate if an output is generated by a particular module. Because the question analysis phase outputs analysis results such as question types, we can assign weights to these features during supervised learning indicating how likely a module is going to produce a correct question type. Suppose there are two modules: statistic learning based question analyzer and rule-based question analyzer. Given the TREC11 question 1396 "What is the name of the volcano that destroyed the ancient city of Pompeii?" two question types are returned. One is detected as "city" by the statistic learning based question analyzer, and the second is detected as "volcano" by the rule based question analyzer. Answer candidates are generated separately according to the type "city" and the type "volcano". For those candidates of "city", we assign value 0 to the feature for the statistic learning based question analyzer and give value 1 to the feature for the rule based question analyzer. In this case, we might expect a ranking model to learn that answer types from the rule-based question analyzer are more highly weighted than answer types from the statistic learning based question analyzer.

The output object features (O_1) We add features for prior knowledge of an output object. The motivation is that some output objects are more likely to be correct or relevant than the others. For example, a specific type "actor" often provides more information than a general type "person". Binary features are created for each type. Moreover, three binary features are added because question types can also be roughly categorized three different

classes, "general", "coarse" and "fine".

The question-output dependency features ($Q - O_1$) Given a question, it is expected to show how likely an output object to be correct or relevant. How do we know if a question type matches the original question? One of our features is using the Naive Bayes model to estimate $P(O|Q)$, the probability that a type is generated by the question Q and model M. The estimation is based on unigram assumption with the maximum likelihood estimator,

$$P(O|Q) \propto P(O) \prod_{t \in Q} P(t|O) \tag{6.1}$$

where O is the question type and $P(O)$ is the question type prior. Those question types are labeled manually as training data.

The question-module dependency features ($Q - M_1$) Intuitively, some modules may perform more accurately for some specific questions. For example, a statistical learning based question analyzer may have high accuracy for a question asking for a person, simply because we have sufficient training questions. A rule based question analyzer may have high accuracy for the numerical types such as length or latitude questions. We add binary features according to pairs combined from question analysis modules (M_1) and the question (Q). The question is represented as unigrams, question focus [77], and extracted named entity types. For example, one such feature can be "rule-based analyzer and question focus volcano".

The module-output dependency features ($M_1 - O_1$) Typically, module objects score and generate their output objects by inference. For example, document retrieval modules may score and return the search results by using language models. However, many modules integrated into the system do not return any score for this purpose or the returned score may not be reliable (e.g., manually assigned score in the rule based question classifier). We need to calculate the probability or correctness of a question type give its module, that is the number of the correct question type divided by the number of both correct and incorrect question type produced by a module. For example, let $c_{person,M}$ be the correct number, and $ic_{person,M}$ be the incorrect number, the score for the feature "person-M" is then $\frac{c_{person,M}}{c_{person,M}+ic_{person,M}}$.

Features of the question analysis phase are summarized in Table 6.3.

6.3.2 Passage Retrieval Phase Features

The module object (M_2) Passages are extracted from structured, unstructured and semi-structured documents or databases. We add binary-valued features to show if a passage is retrieved by a particular search module, one feature per module.

The output object features (O_2) We add passage prior features that are similar to document prior features, such as Pagerank and URL depth, which can improve the retrieval effectiveness of Web Information Retrieval (IR) systems. Given the labeled relevant passages and its URL, we estimate the likelihood a passage is relevant to the question by using the bag-of-words model. Such passage prior can be used to give spams lower rank or to assign a higher rank to relevant passages.

Table 6.3: Features extracted from the phase of question analysis (S_1).

Name	**Structure**	**Feature Type**	**Description**
M_1	M	-	Binary features, one feature per question analysis module.
$Q - M_1$	$Q - M$	Type I	Dependency between question analysis modules and the original question
-	$M' - M$	-	Features unavailable due to the unavailable previous modules M'
-	$O' - M$	-	Features unavailable due to the unavailable previous outputs O'
O_1	O	-	Binary features, one feature per question type
$Q - O_1$	$Q - O$	Type II	Dependency between question types and the original question
$M_1 - O_1$	$M - O$	Type I	Dependency between question types and question analysis modules
-	$M' - O$	-	Features unavailable due to the unavailable previous modules M'
-	$O' - O$	-	Features unavailable due to the unavailable previous output O'

The question-module dependency features ($Q - M_2$) Similar to $Q - M_1$ features, we create potential functions to indicate how likely a search module will retrieve relevant passages.

The question-output dependency features ($Q - O_2$) In the phase of passage retrieval, we applied the query language model to calculate the relevance of a passage retrieved from a search agent to a question,

$$P(O|Q) \propto P(O) \prod_{t \in O} P(t|O) \tag{6.2}$$

where $P(O)$ is the prior estimated as a constant term for passage retrieval. This model is smoothed by background collections with Bayesian smoothing using Dirichlet priors,

$$P(t|O) = \frac{tf_{t,O} + \mu P(t|B)}{L_O + \mu} \tag{6.3}$$

where $tf_{t,O}$ is the frequency count of word t in passage O, L_O is the total number of word in passage O, μ is a prior for smoothing and B is a indexed Wikipedia corpus.

The module'-module dependency features ($M_1 - M_2$) Often modules of different phases are developed together so that their combination can perform well. Binary features are created from modules of question analysis (M_1) and passage retrieval (M_2). One module is selected from the question analysis phase and the other module is selected from the passage retrieval phase. Each feature represents one of the module combinations. And

Table 6.4: Features extracted from the phase of passage retrieval (S_2).

Name	**Structure**	**Feature Type**	**Description**
M_2	M	-	Binary features, indicating which passage retrieval module the passage was retrieved from. One feature per passage retrieval modules.
$Q - M_2$	$Q - M$	Type I	Dependency between passage retrieval modules and the input question
$M_1 - M_2$	$M' - M$	Type I	Dependency between passage retrieval modules and question analysis modules
$O_1 - M_2$	$O' - M$	Type I	Dependency between passage retrieval modules and question types
O_2	O	-	Bag-of-words
$Q - O_2$	$Q - O$	Type II	Calculate passage relevance scores
$M_2 - O_2$	$M - O$	Type III	Dependency between passages and passage retrieval modules
$M_1 - O_2$	$M' - O$	Type III	Dependency between passages and question analysis modules
$O_1 - O_2$	$O' - O$	Type II	Dependency between passages and question types

therefore, the total number of features is the number of first phase's modules multiply the number of second phase's modules.

The output'-module dependency features ($O_1 - M_2$) It is possible that some search modules may perform more accurately with some specific question types. Intuitively, if a question asks for a country's population, it is better to search through electronic gazetteers providing geographic information. We create binary features by combining the current module set and the common question types. The total number of features is then the size of the module set multiply the size of question types.

The module-output dependency features ($M_2 - O_2$) Multiple search modules vary in their domains, documents, retrieval models, and parameters. Usually, the passages' scores cannot be compared directly. Feature scores should be normalized for the purpose of ranking. Our solution is to add features for each of the scoring methods such as tf.idf, BM25, and query likelihood. The number of features for a scoring method is the number of modules. For example, a passage retrieved from Wikipedia has the tf.idf-Wikipedia score, and all the other tf.idf-module scores are 0.

The output'-output dependency features ($O_1 - O_2$) Given a question type (O_1), we create features to estimate how likely a passage (O_2) contains the correct type by analyzing the passage content. We calculate $p(O_2|O_1)$ for this purpose so that the passage content with higher question type relevance will have a better ranking.

Features of the passage retrieval phase are summarized in Table 6.4.

6.3.3 Answer Extraction Phase Features

The module object features (M_3) We add binary-valued features to show if an answer candidate is retrieved by a particular answer extraction, one feature per module.

The output object features (O_3) We add answer candidate (O_3) prior features. It is possible that the answer extractor creates named entities with many commonly used high-frequency concepts. For example, there may be a tendency to answer a question with the year "2012" because some of the web documents are crawled in 2012. We create a dictionary containing stop words and high-frequency words or named entities. An output object "2012" will have a binary feature set to 1 if "2012" is contained in the dictionary.

The question-module dependency features ($Q - M_3$) Similar to $Q - M_1$ features, we create features by using potential functions to indicate how likely an answer extractor will extract correct answers.

The module'-module dependency features ($M_2 - M_3$ & $M_1 - M_3$) We add features that capture the relationship between answer extraction modules (M_3) and question analysis modules (M_1), as well as answer extraction modules (M_3) and passage retrieval modules (M_2). For example, if a candidate is extracted by StanfordNER based answer extraction module and a web search module, the corresponding feature will be set as 1 and all other features will be 0.

The output'-module dependency features ($O_1 - M_3$) The performance of some modules may depend on previous output objects. For example, the answer extractor based on StanfordNLP (M_3) may be proper to extract named entities of person, organization, and location types, but may not be used to extract named entities of many other types (O_1) generated by the question analysis modules. We create binary features by combining the current module set and the common question types. The total number of features is then the size of the module set multiply the size of question types.

The question-output dependency features ($Q - O_3$) We estimate the probability $p(O_3|Q)$. Similar to the web validations approaches [45], a query consisting of an answer candidate (O_3) and the original question (Q) is sent to Wikipedia to retrieve relevant passages. The probability is computed as:

$$P(O_3|Q) \propto P(Q|O_3)P(O_3) = P(O_3) \sum_{q \in Q} P(q|O_3) \tag{6.4}$$

$$P(q|O_3) = (1 - \lambda)P(q|O_3) + \lambda P(q|C) \tag{6.5}$$

where λ is the smoothing parameter, the probability $P(q|O_3)$ is smoothed using the prior probability that the term q is generated from the entire collection of answers C, $P(q|C)$ is computed using the maximum likelihood estimator.

The output'-output dependency features ($O_1 - O_3$) One of the O'-O features estimates the likelihood that the answer candidate matches the detected question type. Similar to answer validation [45], a query consisting of an answer candidate and its target question type is sent to Wikipedia. With those retrieved results, we estimate the probability $p(O_3|O_1)$ as the feature.

Table 6.5: Features extracted from the phase of answer extraction (S_3).

Name	Structure	Feature Type	Description
M_3	M	-	Binary features, indicating which answer extractor found the answer candidate. One feature per answer extractor.
$Q - M_3$	$Q - M$	Type I	Dependency between answer extraction modules and the input question
$M_1 - M_3$	$M' - M$	Type I	Dependency between answer extraction modules and question analysis modules
$M_2 - M_3$	$M' - M$	Type I	Dependency between answer extraction modules and passage retrieval modules
$O_1 - M_3$	$O' - M$	Type I	Dependency between answer extraction modules and question types
$O_2 - M_3$	$O' - M$	Type III	Dependency between answer extraction modules and retrieved passages
O_3	O	-	Match an answer candidate to commonly used high frequent words
$Q - O_3$	$Q - O$	Type II	Calculate answer validation score
$M_3 - O_3$	$M - O$	Type III	Dependency between answer candidates and answer extraction modules
$M_1 - O_3$	$M' - O$	Type III	Dependency between answer candidates and question analysis modules
$M_2 - O_3$	$M' - O$	Type III	Dependency between answer candidates and passage retrieval modules
$O_1 - O_3$	$O' - O$	Type II	Dependency between answer candidates and question types
$O_2 - O_3$	$O' - O$	Type II	Dependency between answer candidates and passages

The output'-output dependency features ($O_2 - O_3$) Given the passage (O_2) and the extracted answer candidate (O_3), $p(O_3|O_2)$ is computed using the maximum likelihood estimator.

Features of the answer extraction phase are summarized in Table 6.5.

6.4 Experiments

We evaluated the phased ranking model on TREC QA datasets, and compared the performance of different answer ranking models. Specifically, we sought to answer the following questions. First, what is the measurable impact of using the proposed method versus a traditional QA ranking approach with final answer ranking? Second, why does the fully implemented phased ranking model perform significantly better than the other models? To

answer these research questions, we first compared the proposed phased ranking method with three baseline answer ranking models in Section 6.4.1, 6.4.2, 6.4.3, and 6.4.4, and then analyzed the effects of the proposed feature types in Section 6.4.5, finally we conduct overlap analysis of answer ranking models with detailed examples in Section 6.4.6.

6.4.1 Baseline I: Comparison to Independent Prediction Model (IPM)

The first baseline model we implemented is the independent prediction model (IPM) [45], an approach that optimizes both answer validation and answer similarity. There are two reasons for us to choose this baseline. The first reason is that it employs answer validation, which is commonly used by previous QA systems in TREC QA task. Answer validation searches the web via different search agents to find relevant answer information. The second reason is that previous successful QA systems [61] reported similar answer ranking methods as IPM. We selected the IPM as a representative of the previous answer ranking methods, which relied on heuristic evidence features to score answer candidates without leveraging non-local information about upstream objects.

6.4.2 Baseline II: Hypothesis Selection Model(HSM)

The second method for comparison is a simplified form of our phased ranking model, which we call the hypothesis selection model (HSM). This approach collects features from prior phases and makes them available for ranking in the current phase, by using only feature types M, O, and $Q - O$. This approach represents a general form of IPM extended with shallow ranking features, such as those used in CHAUCER [39] (e.g., the strategy used to extract the answer, the question type of the original question, and similarity between the original question and an answer candidate).

In this baseline approach, we used the same ranking framework (cascade model, relevance voting, parameters, learning algorithms, training and testing set, and the number of training instances) as the phased ranking model. The only difference is in the feature types employed. All the ranking models were trained by the ranking SVM method based on the cutting-plane method [41]. We applied the ranking models to each dataset separately, and 5-fold cross validation was used to evaluate the performance. In 5-fold cross-validation, we randomly partitioned the original questions into 5 equal size subsets. Among the subsets, a single subset was retained for testing the model, and the remaining 4 subsets were used as training data. The 5 tested subsets are finally combined together to provide an overall evaluation score.

6.4.3 Baseline III: Generalized Watson Rank (GWR)

The Watson DeepQA machine learning framework [32] contains seven sequential phases. This answer ranking framework with multiple ranking phases provides capabilities for manipulating the data and applying machine learning in successive applications. In each phase, the input is a set of candidate answers and their evidence scores. A phase trains

a model for ranking them and for estimating a confidence score indicating correctness for each answer. The number and function of phases are configurable and, those phases in Watson's instantiation for Jeopardy! are summarized as follows.

1. *Hitlist Normalization* Answers begin processing at the Hitlist Normalization phase. At the start of this phase, answers whose string values are identical are merged. After merging, the standardized feature values are computed. The answers are then sent to one of a set of routes based on how the question was classified.
2. *Base* At the start of the base phase, answers are ranked according to the scores assigned by the Hitlist Normalization phase, and only the 100 highest scoring answers from that phase are retained. With at most 100 answers per question, the standardized features are recomputed. The base phase provides the same set of routes as the previous phase.
3. *Transfer Learning* The score assigned to each answer by the model in the base phase is provided as an input feature to a specialized model of the transfer Learning phase. In the case of logistic regression, which uses a linear combination of weights, the weights that are learned in the transfer phase can be roughly interpreted as an update to the parameters learned from the general task.
4. *Answer Merging* The Answer Merging phase combines equivalent answers. It uses a variety of resources to determine equivalence. It uses the results of the previous phase to determine which of the various forms being merged is canonical. In other words, among the identical answers, a canonical form is selected by considering the evidence score.
5. *Elite* The elite phase is similar to the base phase but employs only the top five answers from the previous phase.
6. *Evidence Diffusion* The Evidence Diffusion phase is similar to the Answer Merging phase but combines evidence from related answers instead of equivalent ones.
7. *Multi-Answers* The Multi-Answer phase provides one additional type of merging that is only invoked for questions that ask for a list of answers; it combines multiple answers from earlier phases into a final answer.

6.4.4 Implementation of Generalized Watson Rank (GWR)

We implemented the Generalized Watson Rank for a direct comparison with our proposed phased ranking model. There are two reasons to modify the original Watson DeepQA's ranking framework. First, the original framework only applies to final answer ranking phase. In an information system, it is necessary to extend this framework to all the phases that require the step of ranking. Second, some of the ranking stages are specialized for the Jeopardy! task, and therefore we created the following common ranking stages for any ranking task:

1. *Hitlist Normalization* Standardized features are implemented and added to capture

the relative values of feature in a system. Those features are standardized per query,

$$x_{ij}^{std} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{6.6}$$

where x_{ij} stands for a feature from the i-th instance from the j-th input set I, the standardized feature x_{ij}^{std} is calculated from feature x_{ij} by centering (subtracting the mean μ_j) and scaling to unit variance (dividing by the standard deviation σ_j) over all instance for that input. For each feature provided as an input, we add an additional new std feature that is the standardized version of that feature. Both mean and standard deviation are computed within a single question since the purpose of standardization is to provide a contrast with the other instance that the candidate is being ranked against.

2. *Base* At the start of the base phase, candidates are ranked according to the scores assigned by the hitlist normalization stage, and only the 100 highest scoring answers from that phase are retained. With those per question, the standardized features are recomputed.
3. *Answer Merging* The Answer Merging phase merges equivalent output objects or combines evidence from similar output objects by using customized similarity functions.
4. *Elite* The elite phase works similar as the base phase but employs only the top five answers from the previous phase. All the ranking stages will use the same ranking algorithm (e.g., Logistic Regression), object functions, and cross validation approaches.

6.4.5 Experimental Results and Discussion

Table 6.6 shows the end-to-end system performance for our proposed models and the baselines. Statistical significance was calculated using a one-sided sign test on questions.

From the performance comparison, we notice two trends. First, HSM either significantly outperforms or is statistically indistinguishable from IPM in all cases. This highlights the advantage of the phased ranking approach: by propagating features from every phase of a system object graph to the final phase ranking, it allows for the use of richer features to form more complex ranking functions for higher accuracy than extracting local features heuristically. Second, in all datasets, PhRank yields consistent and significant ($p < 0.001$) improvements over the baseline IPM, and outperforms the HSM and the GWR in terms of TOP@1, MRR@5 and Recall@10 in all cases. This illustrates that exploring system object graph dependencies between phases can significantly improve ranking performance.

We observed that performance numbers vary across different datasets. The main reason is that most of the original development efforts on OpenEphyra focused on TREC11 questions and the components we used don't have specific strategies to answer the large number of definition questions in the earlier TREC datasets (definition questions were not present in the TREC11 dataset).

We conducted a set of experiments to verify the effectiveness of propagating features from every phase to answer ranking. The results are summarized in Table 6.7, and the features grouped by phase are denoted as S_1, S_2 and S_3 separately. It can be clearly observed that none of the individual set of features performed better than the combination of all the features. The features derived from the question analysis phase performed the worst among all the three phases. This is reasonable because most of the answer candidates share the same question evidence, which provides the least information to distinguish the correct answers from a large number of noisy answer candidates. The features derived from the passage retrieval phase will only capture passage relevance (e.g., a keyword match between a passage and the original question), and therefore are not effective enough to train a satisfactory model for answer ranking by themselves. Features derived from the answer extraction phase provided the best performance gain, indicating that direct evidence for answers (e.g., answer validation features, answer similarity features) provided the best source of ranking information.

To further study the effectiveness of the proposed features, we trained ranking models by leaving out features per phase. Based on results, we expect the PhRank approach to achieve the best results by combining features from all the phases. The results confirmed our hypothesis. None of the ranking models trained by leaving one feature type out is better than the model using all of the features. In the experiment, omitting S_3 features (no.S_3) leads to the most significant drop in performance. Although the features from the question analysis phase (S_1) were not effective when used in isolation, a ranking model that uses them in combination with the other features worked better than one leaving them.

The PhRank approach integrates nine types of features. We analyzed each feature set's contribution and the results are shown in Table 6.8. The PhRank approach combining all features significantly outperforms the one with the best single-type features. Surprisingly, both the independent features M and O did not perform well. In contrast with their performance in isolation, they are the building blocks for the best performing dependency feature type $M(+M') - O$ (a combination of type $M - O$ and type $M' - O$, for better illustration of the feature effectiveness).

In the feature ablation studies, we do a "leave one feature type out" analysis. Our motivation is to identify the key factors of the dependency features in PhRank. Results are presented in Table 6.9. We observed that omitting $Q - M$ features leads to the most significant drop in performance. This illustrates that models developed in a system can strongly depend on features of the original question. One might have expected that as long as the question type is detected, the original question words are less important. But these results show that original question provides additional vital evidence for ranking, since classified question types have already been modeled in the $O' - M$ relationship. Ablating the feature type $M(+M') - O$ also leads to a significant drop in performance. This implies that an output object and its preceding model object are strongly connected, and this relationship helps to inform the ranking decision. However, dropping feature type M leads to a performance improvement, suggesting that the dependency feature types can be an effective replacement. The comparative advantage of the other features is less pronounced than that of $Q - M$ features, but they still contribute to the overall performance.

We wish to further understand when and why the datasets favor one approach over the

other. Two concrete examples are shown to explain why HSM outperforms IPM, and how PhRank reinforces HSM in terms of answer accuracy. HSM clearly performs well at its intended purpose, learning a cascade model with three-level features extracted from each phase to rank the candidates. Consider the TREC11 question 1847 "What is Tina Turner's real name?" In the phase of question analysis, the first model detects the expected answer type as "person"; the second model generates an interrogative word "what" strategy as an alternative. In the answer extraction phase, the former model may find the candidate "Anna Mae Bullock" (the correct answer) and the latter may lead to an incomplete answer "Bullock". IPM favors the incomplete answer because both of the candidates have similar text context and the candidate "Bullock" has a higher redundancy than the other candidate "Anna Mae Bullock". As opposed to the independent prediction model, HSM progressively collects the supporting evidence from each phase, and hence it can more correctly estimate whether a candidate answer is a correct one. By learning from the training data, HSM assigns a system object graph feature of 0.9 to the candidate "Anna Mae Bullock" extracted by the first model, and assigns the same feature a value of 0.1 for the candidate "Bullock" extracted by the second model.

In order to answer why PhRank outperforms HSM, we show an example in Figure 6.2 where dependency features are able to estimate supporting evidence scores of candidate answers more accurately. The candidates "Siam" and "Bhumibol Adulyadej" are derived from the component algorithms M_a and M_b separately. In this example, HSM only uses features f_1 and f_2, while PhRank employs dependency features f_3 to f_8 in addition to f_1 and f_2. Because M_b derives correct answers more often than M_a, the ranking approach gives higher confidence score to M_b ($f_1 < f_2$), leading to the incorrect answer for HSM. By adding dependency features, we can see that if a named entity of "LOC" ("Thailand" in this case) appears in the question, component M_a is preferred than component M_b to derive the candidate answers. This observation explains that those dependency features form more complex ranking functions and can provide higher answer accuracy than HSM.

Table 6.6: Performance comparison of three models: the independent predication model (IPM), the hypothesis selection model (HSM), the Generalized Watson Rank (GWR) and the fully implemented phased ranking model (PhRank). Significance, using a one-sided sign test, is denoted with a † at the $p < 0.05$ level and a ‡ at the $p < 0.005$ level over the independent prediction model.

TOP@1

	TREC9	TREC10	TREC11	TREC12
IPM	0.251	0.245	0.315	0.227
HSM	0.307	0.289	0.419	0.335
GWR	0.291	0.296	0.491	0.346
PhRank	**0.362** (+0.055)‡	**0.356** (+0.06)†	**0.514** (+0.023)	**0.348** (+0.02)

MRR@5

	TREC9	TREC10	TREC11	TREC12
IPM	0.341	0.335	0.405	0.296
HSM	0.395	0.387	0.500	0.414
GWR	0.389	0.375	0.553	0.406
PhRank	**0.469** (+0.074)‡	**0.455** (+0.068)‡	**0.606** (+0.053)‡	**0.451** (+0.045)†

Recall@10

	TREC9	TREC10	TREC11	TREC12
IPM	0.594	0.557	0.651	0.509
HSM	0.617	0.594	0.709	0.620
GWR	**0.737**	0.670	0.757	0.641
PhRank	0.714 (-0.023)	**0.677** (+0.007)	**0.797** (+0.04)†	**0.694** (0.053)‡

Table 6.7: Feature analysis per phase: question analysis (S_1), passage retrieval (S_2), and answer extraction (S_3), evaluated on TREC11.

TOP@1

	S_1	S_2	S_3	no.S_1	no.S_2	no.S_3	All
PhRank	0.0248	0.2725	0.4437	0.4977	0.4662	0.4189	**0.5135**

MRR@5

	S_1	S_2	S_3	no.S_1	no.S_2	no.S_3	All
PhRank	0.0589	0.3746	0.5411	0.5923	0.5564	0.5103	**0.6056**

Recall@10

	S_1	S_2	S_3	no.S_1	no.S_2	no.S_3	All
PhRank	0.1779	0.6554	0.7162	0.7905	0.7320	0.7455	**0.7973**

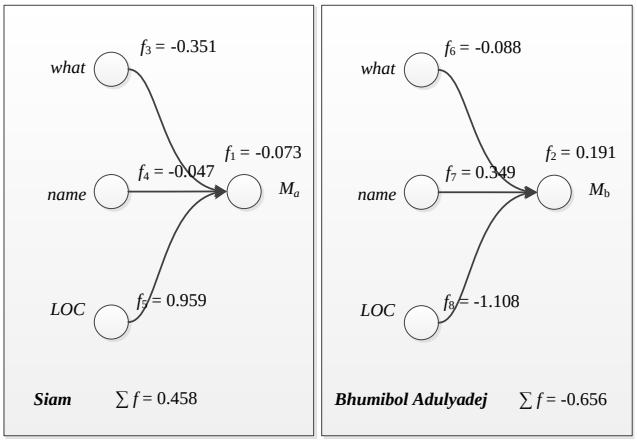

Figure 6.2: Example of the proposed dependency features $Q - M_1$ in PhRank for scoring two candidates "Siam" and "Bhumibol Adulyadej" to answer the TREC11 question 1828 "What was Thailand's original name?" In this example, the aggregated evidence scores for "Siam" and "Bhumibol Adulyadej" are 0.458 and -0.656 separately, suggesting that "Siam" is more likely to be the correct answer.

Table 6.8: Contribution of feature types evaluated on TREC11.

TOP@1

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	0.2320	0.0293	0.3018	0.2995	0.2703	0.3896	0.4572	0.4414	**0.5135**

MRR@5

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	0.3412	0.0680	0.4268	0.4123	0.3779	0.4789	0.5401	0.5274	**0.6056**

Recall@10

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	0.6284	0.1847	0.7320	0.6712	0.6667	0.6847	0.7320	0.6982	**0.7973**

Table 6.9: Feature type ablation study evaluated on TREC11.

TOP@1

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	**0.5248**	0.5090	0.4617	0.5113	0.5113	0.4932	0.4752	0.5180	0.5135

MRR@5

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	**0.6157**	0.6038	0.5560	0.5979	0.5979	0.5955	0.5729	0.6101	0.6056

Recall@10

	M	O	$Q-M$	$M'-M$	$O'-M$	$Q-O$	$M(+M')-O$	$O'-O$	All
PhRank	**0.8018**	0.8040	0.75	0.7928	0.7973	0.7995	0.7838	0.8018	0.7973

6.4.6 Overlap Analysis

We perform overlap analysis between the proposed model and the strongest baseline Generalized Watson Rank to show the differences between the two approaches and to further characterize the effectiveness of the proposed model.

To direct compare the Phased Ranking model with the Generalized Watson Rank, we use the same answer generation pipeline to evaluate the impact of the Phased Ranking model on end-to-end QA performance in these experiments. For scoring candidate answers, we use the same features in the Generalized Watson Rank as those features included in the baseline HSM. There are two main reasons that we do not re-engineering Watson DeepQA's ranking features. First, our goal is to compare the performance of ranking frameworks, not to compare end-to-end QA performances between our QA system and Watson DeepQA system. Second, the implementation details of features in Watson DeepQA are not published and therefore it is difficult to re-implement those features.

In the experiments, the proposed Phased Ranking model leverages the features extracted from the system object graph to score the candidates and select the final answer. Compared with a strong baseline containing features from every individual phase, the Phased Ranking model has the advantage of learning phase dependencies. Those dependencies can be important to model the connections between system objects for ranking candidate answers. Without exploring those dependencies, a ranking framework could risk of not optimizing the system performance for the analytics tasks, especially when a system gets complicated. To illustrate, those dependencies in our QA system are categorized as the following,

1. **Question Analysis → Passage Retrieval**: the dependency between question analysis and passage retrieval, for example, some passage retrieval modules can be better fit for some types of questions (question analysis output).
    - Wikipedia title and first sentence can be a good choice to answer questions asking for a name (Type: $O_1 \rightarrow M_2$).
        - TREC10 question 1044 "*What is another name for vitamin B1?*"
        - Answer: Thiamine
        - Wikipedia title "*Thiamine - Wikipedia, the free encyclopedia*",
        - Wikipedia first sentence "*Thiamine, thiamin or vitamin B1, named as the "thio-vitamine" ("sulfur-containing vitamin") is a vitamin of the B complex.*"
    - Wikipedia title and first sentence may not be a good choice to answer questions asking for date (Type: $O_1 \rightarrow M_2$).
        - TREC10 question 1047 "*When was Algeria colonized?*"
        - Answer: 1830
        - Wikipedia title "*Algeria - Wikipedia, the free encyclopedia*",
        - Wikipedia first sentence "*Algeria, officially People's Democratic Republic of Algeria, is a sovereign state in North Africa on the Mediterranean coast.*"

2. **Question Analysis → Answer Generation**: the dependency between question analysis and answer generation.
    - A correct answer candidate (answer generation output) should have the right answer type to the given question (Type: $O_1 \rightarrow O_3$).
        - TREC11 question 1638 "*What city in Australia has rain forests?*"
        - Candidate answer 1: Queensland (a state, incorrect)
        - Candidate answer 2: Daintree (a city, correct)
3. **Passage Retrieval → Answer Generation**: the dependency between passage retrieval and answer generation, for example, the relationship of a candidate answer with the other words in a passage.
    - An answer candidate (answer generation output) close to the keywords in a passage could be more correct than the candidates that are far away from those keywords (Type: $O_2 \rightarrow O_3$).
        - TREC11 question 1613 "*When was Benjamin Disraeli prime minister?*"
        - Answer: 1868
        - A relevant passage: "Benjamin Disraeli served twice as Prime Minister, the first time from 27 February to 1 December 1868 and the second, 20 February 1874 to 21 April 1880. Disraeli was born on 21 December 1804 at Bedford Row, London, ..."
        - The correct answer candidate: 1868 (within the same sentence with several keywords such as "Benjamin Disraeli" and "Prime Minister")
        - The incorrect answer candidate: 1804 (far away from those keywords)

To perform overlap analysis, the Generalized Watson Rank uses a multi-step ranking framework described in Section 6.4.4. In contrast, Phased Ranking model has only learning layer but integrates more phased dependency features. We call those features "Salient Missing Features" which appear in the Phased Ranking model but are missing from the setting of features in the Generalized Watson Rank. The dependency features include three categories, shown in Table 6.10

Table 6.11 shows the results of overlap analysis between our implementation of Generalized Watson Rank, and our proposed phased ranking framework. The Gain/Loss percentage is higher on the two categories of dependency features, which are Question Analysis → Passage Retrieval and Passage Retrieval → Answer Generation, than the category of Question Analysis → Answer Generation. The reason is that the former two categories contained dependency features from the passage retrieval phase, where answer candidates are scored by dominant features (e.g., answer relevance features).

There is more Loss percentage on the category of Question Analysis → Passage Retrieval than the other two categories. This is because that the baseline model employed evidence scores from search engines and knowledge sources by taking a keyword query and returns a list of documents. Adding similar features extracted by question and passage retrieval could cause the imbalance of various answer evidence (e.g., answer search relevance

Table 6.10: Salient Missing Features.

Feature Categories I: Question Analysis → Passage Retrieval	
$Q \rightarrow O_2$	Question-passage dependency features
$O_1 \rightarrow O_2$	Type-passage dependency features
$Q \rightarrow M_2$	Question-passage retrieval module dependency features
$O_1 \rightarrow M_2$	Type-passage retrieval module dependency features
$M_1 \rightarrow M_2$	Question analysis module-passage retrieval module dependency features
$M_1 \rightarrow O_2$	Question analysis module-passage dependency features
Feature Categories II: Question Analysis → Answer Generation	
$Q \rightarrow O_3$	Question-passage dependency features
$Q \rightarrow M_3$	Question-answer generation module dependency features
$Q \rightarrow M_3$	Question-answer generation module dependency features
$O_1 \rightarrow M_3$	Type-answer generation module dependency features
$O_1 \rightarrow O_3$	Type-answer dependency features
$M_1 \rightarrow M_3$	Type-answer generation module dependency features
$M_1 \rightarrow O_3$	Type-answer generation module dependency features
Feature Categories III: Question Analysis → Answer Generation	
$O_2 \rightarrow O_3$	Passage-answer dependency features
$O_2 \rightarrow M_3$	Passage-answer generation module dependency features
$M_2 \rightarrow M_3$	Passage retrieval module-answer generation module dependency features
$M_2 \rightarrow O_3$	Passage retrieval module-answer dependency features

vs. semantic typing relevance).

The large difference between Gain and Loss percentage on the category of Question Analysis → Answer Generation shows that it contributed better Gain/Loss ratio than the other categories. The reason is that the proposed phased ranking approach could model a broader range of answer types. In contrast, the baseline model did not have features to give preference to answers having the right type. For example, consider the TREC9 question 743 "CNN is owned by whom?". The baseline model cannot have the correct evidence when the pipelined system failed to detect the correct question type. The proposed phased ranking approach can solve the problem by using dependency features; details are shown in Section 6.4.6.

Table 6.11: Overlapping analysis results between the proposed phased ranking model (PhRank) and the strongest baseline Generalized Watson Rank (GWR).

Dataset \ Type		**Salient Missing Features**		
		Question Analysis → Passage Retrieval	Question Analysis → Answer Generation	Passage Retrieval → Answer Generation
TREC9	PhRank>GWR	31 (34.8%)	27 (30.3%)	31 (34.8%)
	PhRank<GWR	25 (51.0%)	4 (8.1%)	20 (40.8%)
TREC10	PhRank>GWR	19 (35.8%)	16 (30.1%)	18 (33.9%)
	PhRank<GWR	17 (62.9%)	3 (11.1%)	7 (25.9%)
TREC11	PhRank>GWR	21 (42.8%)	12(24.4%)	16 (32.6%)
	PhRank<GWRline	28 (73.6%)	3 (7.8%)	7 (14.8%)
TREC12	PhRank>GWR	10 (28.5%)	9 (25.7%)	16 (45.7%)
	PhRank<GWR	25 (73.5%)	3 (8.8%)	6 (17.6%)

Question Analysis → Passage Retrieval (Gain)

An example of gain on features on Question Analysis → Passage Retrieval is shown in Figure 6.3. Consider the following question,

- TREC12 Question 2110: What is Speedy Claxton's real name?
- Answer: Craig

The two ranking approaches have the following returned answers:

- PhRank: Craig (correct)
- GWR: Meeka Claxton (incorrect)

A derivation path in the phases of question analysis and passage retrieval for an instance of "Craig" is,

- Question: What is Speedy Claxton's real name?
- Passage retrieval module: Wikipedia-FirstPassage
- Retrieved passage: ***Craig*** *"Speedy" Claxton (born May 8, 1978) ...*

Similarly, a derivation path for an instance of "Meeka Claxton" is,

- Question: What is Speedy Claxton's real name?
- Passage retrieval module: Web Search
- Retrieved passage: ***Meeka Claxton*** *And Husband Speedy Seen On The Scene In NYC ...*

Figure 6.12 and Figure 6.13 are the simplified ranking features of the two ranking approaches showing that the dependency features (Question Analysis → Passage Retrieval) help PhRank improve answer quality over GWR. Independent feature f2 has higher ranking feature score than f1 indicating that it is more confident to detect correct answer from Web search than from Wikipedia-FirstPassage based module. Dependent feature f3/f4 jointly models the situation that the question is asking for a name of a person and the answer candidate is (or is not) detected from the Wikipedia-FirstPassage based module. From independent feature f1, the answer candidate "Craig" gets lower score 0.3 than the score 1.1 of "Meeka Claxton", but gets better estimation from the dependency feature f3. In that case, the Wikipedia-FirstPassage based module works well for some question but produces incorrect candidates to some other questions. Dependency feature f4 gives penalty score -0.2 to the candidate "Meeka Claxton", simply because it doesn't show up in the first passage of Wikipedia.

Question Analysis → Passage Retrieval (Loss)

An example of loss on features on Question Analysis → Passage Retrieval is shown in Figure 6.4. Consider the following question,

Table 6.12: Performance of feature sets in GWR, f1 represents for M-Feature: Wikipedia-FirstPassage, f2 represents for M-Feature: WebSearch, rank score is the final ranking score.

Answer	f1	f2	Rank Score
Craig	0.3	0.1	0.4
Meeka Claxton	0	1.1	1.1

Table 6.13: Performance of feature sets in PhRank, f1 represents for M-Feature: Wikipedia-FirstPassage, f2 represents for M-Feature: WebSearch, f3 represents for the dependency feature name-f1-1, and f4 represents for the dependency feature name-f1-0, and rank score is the final ranking score.

Answer	f1	f2	f3	f4	Rank Score
Craig	0.1	0.07	0.7	0	0.87
Meeka Claxton	0	0.8	0	-0.2	0.6

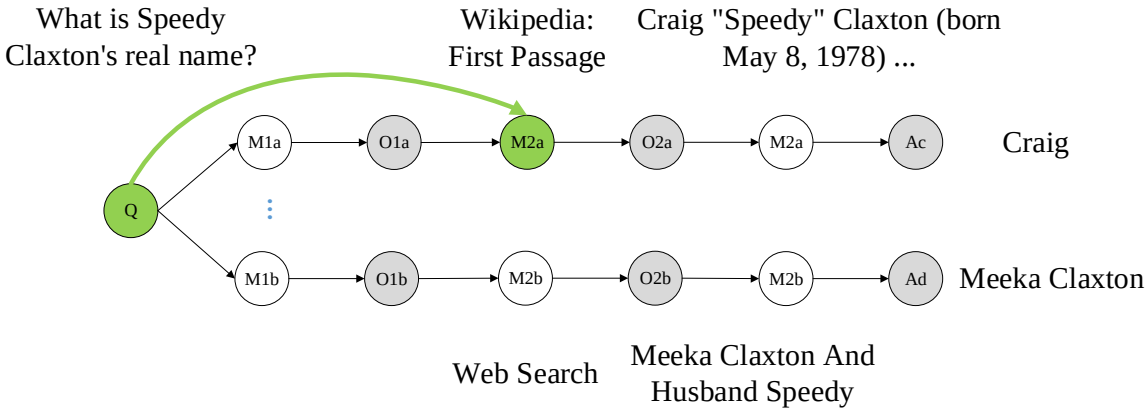

Figure 6.3: Example of Question Analysis → Passage Retrieval Features (Gain).

- TREC9 Question 289: What are John C. Calhoun and Henry Clay known as?
- Answer: Senators

The two ranking approaches have the following returned answers:

- PhRank: Webster (incorrect)
- GWR: Senators (correct)

A derivation path in the phases of question analysis and passage retrieval for an instance of "Webster" is,

- Question: What are John C. Calhoun and Henry Clay known as?
- Retrieved passage: *Three Senatorial Giants: Clay, Calhoun and* ***Webster****... HENRY CLAY of Kentucky, JOHN C. CALHOUN of South Carolina, and DANIEL* ***WEBSTER*** *of Massachusetts dominated national politics from the end of the War of 1812 until their deaths in the early 1850s..*

Similarly, a derivation path for an instance of "Senators" is,

- Question: What are John C. Calhoun and Henry Clay known as?
- Retrieved passage: *Three powerful* ***senators****, Clay, Webster, and Calhoun, each representing a particular region of the nation, defined American politics for decades in the 1800s.*

The question analysis phase detected that the question is likely asking for types "Organization" or "Profession". Two strong candidates extracted from the passage retrieval phase were"Webster" and "Senators". The candidate "Webster" is more like a type "Person" than a type of "Organization". The candidate "Senator" matches the type of "Profession". As a result, the Generalized Watson Rank assigned the two candidates "Webster" and "Senators" with scores 1.2 and 1.5 respectively, so the candidate "Senators" was chosen as the final answer.

We found that errors are introduced when an approach overly emphasizes on shallow dependency features of question analysis and passage retrieval phases. Table 6.14 shows the results of adding additional features of $\varphi(D, Q)$ to estimate the passage relevance. Dependency features f2 include the count of key terms, coverage of key terms, the score of the joint distribution over question Q and document D with various sequential term dependence settings [58], etc. It can be seen that the three names, John C. Calhoun, Henry Clay and Daniel Webster, often co-occur together in the retrieved passages, which causes the high relevance score for the incorrect candidate "Webster". Adding more features of $\varphi(D, Q)$ leads to the results that PhRank approach prefers "Webster" than "Senators".

Question Analysis → Answer Generation (Gain)

An example of loss on features on Question Analysis → Answer Generation is shown in Figure 6.5. Consider the following question,

Table 6.14: Performance of feature sets in PhRank, f1 represents the original features, f2 represents the additional dependency features $\varphi(D, Q)$, and rank score is the final ranking score.

Answer	f1	f2	Rank Score
Webster	1.1	0.8	1.9
Senators	1.3	0.3	1.6

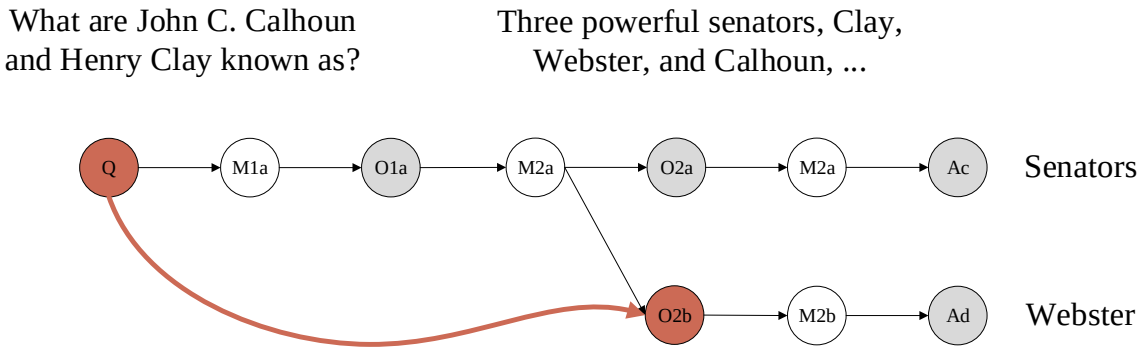

Figure 6.4: Example of Question Analysis → Passage Retrieval Features (Loss).

Table 6.15: Performance of feature sets in PhRank, f1 represents the original features, f2 represent the dependency features "whom → CANDIDATE (TYPE)", "CNN → CANDIDATE (TYPE)", "owned by → CANDIDATE (TYPE)" respectively, and rank score is the final ranking score.

Answer	f1	f2	F3	f4	Rank Score
Turner	2.7	0.4	0	0	3.1
News	2.9	0	0	0	2.9

- TREC9 Question 743: CNN is owned by whom?
- Answer: Turner

The two ranking approaches have the following returned answers:

- PhRank: Turner (correct)
- GWR: News (incorrect)

A derivation path in the phases of question analysis and answer generation for an instance of "Turner" is,

- Question: CNN is owned by whom?
- Answer candidate: Turner

Similarly, a derivation path for an instance of "News" is,

- Question: CNN is owned by whom?
- Answer candidate: News

The question analysis phase fails to detect that the question is expecting candidates with types "Organization" or "Person". A backup plan for the system to detect candidates is using the type "Proper Name". Two strong candidates extracted from the passage retrieval phase were"Turner" and "News". The candidate "News" is highly relevant to the question and has more redundant candidates than the candidate "Turner". As a result, the candidate "News" was chosen as the correct answer in the Generalized Watson Rank.

Table 6.15 shows the results of adding additional features of $\varphi(Q, A)$ for answer ranking. Dependency features from f2 to f4 are the shallow counting features "whom (who) → CANDIDATE (TYPE)", "CNN → CANDIDATE (TYPE)", and "owned by → CANDIDATE (TYPE)". From the training questions, there are many cases matching the feature "whom (who) → CANDIDATE (TYPE)", but it can be seen that the other shallow counting features (f3 and f4) are sparse. Adding dependent features of question analysis and answer generation helps PhRank approach chose the correct answer candidate "Turner" than "News".

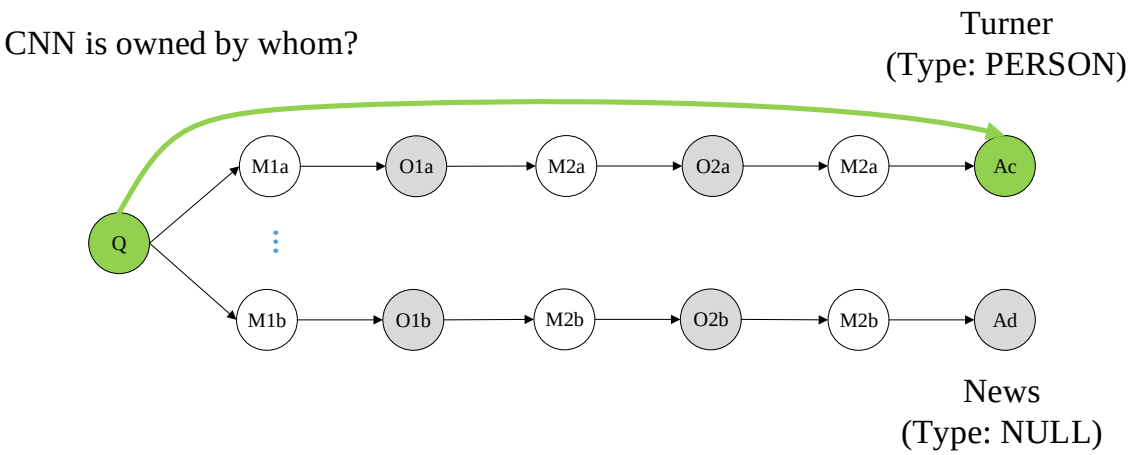

Figure 6.5: Example of Question Analysis → Answer Generation Features (Gain).

Question Analysis → Answer Generation (Loss)

An example of loss on features on Question Analysis → Answer Generation is shown in Figure 6.6. Consider the following question,

- TREC11 Question 1443: When did Bob Marley die?
- Answer: 1981

The two ranking approaches have the following returned answers:

- PhRank: 1977 (incorrect)
- GWR: 1981 (correct)

A derivation path in the phases of question analysis and answer generation for an instance of "1977" is,

- Question: When did Bob Marley die?
- Retrieved passage: *Illness and death Marley in concert in 1980, Zrich, Switzerland In July 1977, Marley was found to have a type of malignant melanoma under the nail of a toe ...*

Similarly, a derivation path for an instance of "1981" is,

- Question: When did Bob Marley die?
- Retrieved passage: *Bob Marley died on 11 May 1981 at Cedars ...*

The question analysis phase correctly detected that the question type is "Date". Two strong candidates extracted from the passage retrieval phase were"1977" and "1981". Both of the candidates have the correct answer type. The candidate "1981" is slightly more

Table 6.16: Performance of feature sets in PhRank, f1 represents the original features and rank score is the final ranking score.

Answer	f1	Rank Score
1977	1.9	1.9
1981	2.1	2.1

Table 6.17: Performance of feature sets in PhRank, f1 represents the original features, f2 represents the dependency features $\varphi(O_1, A)$ to verify type correctness, and rank score is the final ranking score.

Answer	f1	f2	Rank Score
1977	2.8	1.1	3.9
1981	2.9	0.7	3.6

popular than the candidate "1977". In the end, the Generalized Watson Rank chosen "1981" as the final answer.

In the PhRank approach, verifying the type of an answer candidate introduces an unexpected error. Table 6.17 shows the results of adding additional features of $\varphi(O_1, A)$ to estimate the type correctness, where O_1 is the detected question type "Date". To verify the correctness, we use predefined String patterns as well as sending the type "Date" and answer candidates into search components and count the co-occurrence. Those features lead to a result that candidate "1977" is more likely to be a "Date" type than the candidate "1981" to be a "Date" type. Finally, the incorrect candidate "1977" was chosen as the final answer in the proposed RhRank approach.

Passage Retrieval → Answer Generation (Gain)

An example of gain on features on Passage Retrieval → Answer Generation is shown in Figure 6.3. Consider the following question,

- TREC10 Question 963: What strait separates North America from Asia?
- Answer: Bering

The two ranking approaches have the following returned answers:

- PhRank: Bering (correct)
- GWR: Bering Sea (incorrect)

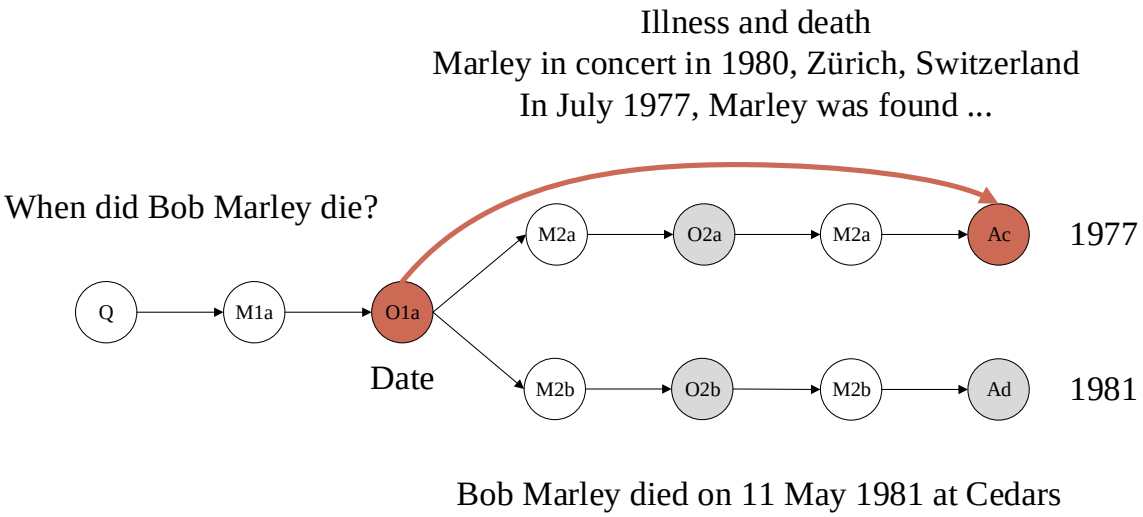

Figure 6.6: Example of Question Analysis → Answer Generation Features (Loss).

A derivation path in the phases of question analysis and passage retrieval for an instance of "Bering" is,

- Retrieved passage: *Although Greenland is the closest land to them, they are much closer to Europe than to the North American mainland. Asia and North America. The Bering Strait and Bering Sea separate the landmasses of Asia and North America, as well as forming the international boundaries between Russia and the United States, respectively. This national and continental boundary separates the Diomede Islands in the Bering Strait, with Big Diomede in Russia and Little Diomede in the US.*
- Answer generator: a third party candidate extractor (ThirdPartyNER)
- Answer candidate: Bering

Similarly, a derivation path for an instance of "Bering Sea" is,

- Retrieved passage: the same as "Bering"
- Answer generator: An extension component of named entity extractor (NERExtension)

The question analysis phase classified the question type as "Location.Water". The ThirdPartyNER component recognized "Bering Sea" but did not recognize "Bering Strait" or "Bering" as a type of "Location.Water". The candidate "Bering Strait" was detected by using the type "Location", and "Bering" is detected by a "NEWhat" type (the question analysis phase often doesn't understand the question type, in this case "NEWhat" is assigned to a question). Because the main question type is one of the key factors for ranking, the Generalized Watson Rank preferred "Bering Sea" to "Bering Strait" or "Bering".

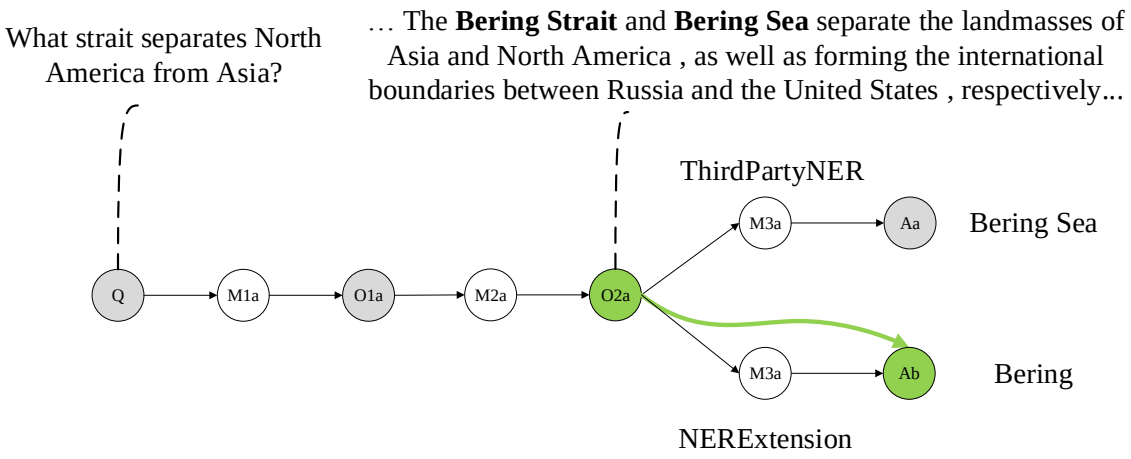

Figure 6.7: Example of Passage Retrieval → Answer Generation Features (Gain).

We found that dependency ranking features (Passage Retrieval → Answer Generation) helped PhRank improve answer quality over the Generalized Watson Rank. A part of dependency ranking features (Passage Retrieval → Answer Generation) that helped PhRank improve answer quality over the Generalized Watson Rank is shown as the following,

- Punctuations, words and their lemmas in a n-word-window surrounding the current answer candidate, (e.g., window size is 1, Candidate is "Bering", surrounding word is "The" and "Strait").
- If words match the keywords or question focus in a n-word-window surrounding the current answer candidate (e.g., window size is 1, Candidate is "Bering", "Strait" is a question focus)
- A word position flag for each candidate (e.g., kStartSentence is true for "Bering", kEndSentence is false).
- A word character flag for each candidate (e.g., firstCap is true for "Bering", allCaps is false).

Passage Retrieval → Answer Generation (Loss)

An example of loss on features on Passage Retrieval → Answer Generation is shown in Figure 6.3. Consider the following question,

- TREC12 Question 2156: How fast does Randy Johnson throw?
- Answer: 100 mph

The two ranking approaches have the following returned answers:

- PhRank: ChaCha (incorrect)
- GWR: 100 mph (correct)

A derivation path in the phases of passage retrieval and answer generation for an instance of "ChaCha" is,

- Retrieved passage: *How hard could Randy Johnson throw a baseball? —* ***ChaCha***
- Answer candidate: ChaCha

Similarly, a derivation path for an instance of "100 mph" is,

- Retrieved passage: *From there, Mantei got a first-hand look at Randy Johnson. Johnson has been known to throw* ***100 mph*** *pitches after nine innings.*
- Answer candidate: 100 mph

The question analysis phase correctly classified the question type as "NESpeed". A ThirdPartyNER component recognized "100 mph" and ranked as top 1 answer candidate by the Generalized Watson Rank. The candidate "ChaCha" was also detected by a "NEHow" type (we used a backup plan because the question analysis phase often doesn't understand the question type). It is ranked lower than the correct answer because its type doesn't match the expected question type.

In the PhRank approach, we found some of the features had negative effects over this question asking for "NESpeed". The main reason is that "ChaCha" is directly followed by a similar question "How hard could Randy Johnson throw a baseball?". Typically as a pattern, answers can be found directly before/after their questions. As a result, the dependency features contributed answer evidence incorrectly for the candidate "ChaCha". For example, those features mainly include word character flags (e.g., firstCap), word position flags (e.g., partOfTitle, ansAfterQuestion), and surrounding words (e.g., n-window keywords) of Passage Retrieval $\rightarrow$ Answer Generation.

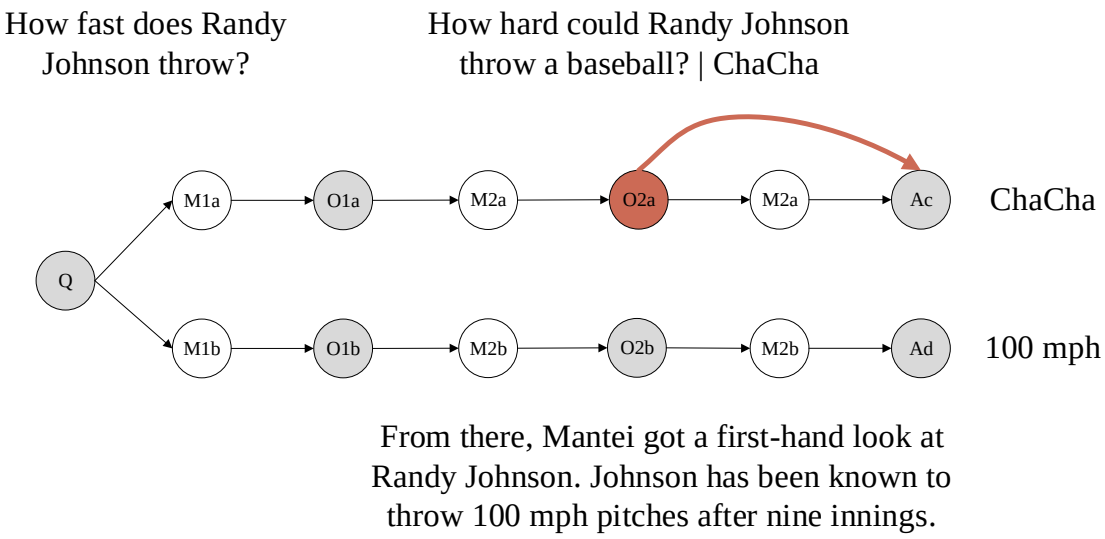

Figure 6.8: Example of Passage Retrieval → Answer Generation Features (Loss).

Chapter 7

Discussion: General Application Principles

We present in this chapter some discussions in order to specify a system object graph for a given pipelined system. Those discussions include a phase identification step (Section 7.1) to classify phases, a phase merging step to simplify the graph (Section 7.2), and applying those principles to information systems (Section 7.3).

7.1 Identifying Phases

The user may define an arbitrary number of phases for an IS, because each of the defined phases can be decomposed into a series of sub-phases. For example, a question answering system may have four main phases: question analysis, document retrieval, information extraction and answer generation. The task of question analysis may has four sub-phases: keyword extraction, dependency parsing, question focus detection, and question classification. Keyword extraction makes use of part of speech tagging and phrase chunking to extract terminological candidates. Those candidates can be further filtered from the candidate list using statistical and machine learning methods. In general, we formulate the process of generating candidates through the definitions given below.

Definition 8 *A candidate generation model is defined as a set of functions*

$$\boldsymbol{O} = g_n(...g_2(g_1(\boldsymbol{I}))...) \tag{7.1}$$

where g_i is the function associated with the i-th phase in a system object graph.

The candidate generation model consists of an additive sequence of functions $\{g_1, ..., g_n\}$, where each function g_i is associated with its phase $\mathcal{P}_i$.

Applying a phase model for every step of a deeply decomposed IS may not be very efficient because a large number of candidate generation functions are likely to be trivial and/or will not help much in the final ranking model. Given an IS, how can we capture the phases that are critical to ranking? Our approach is to apply different criteria to the problem of identifying phases.

Since a processing unit is a possible phase, it must have three basic elements including input, output, and modules. The task is to assign each processing unit with a label indicating whether the processing unit is a phase to be added into the system object graph. For example, in our system the task of keyword extraction is not treated a phase, so we ignore this processing unit. In contrast, passage retrieval step is labeled as a phase and therefore it is added to the phase pool for further processing. Our approach to label the processing unit is rule based. For a processing unit, if any rule is satisfied we label it a phase; otherwise it is not a phase. In order to label phases, we use two criteria, non-shared modules, non-shared output, to help make the decision.

7.1.1 Competing modules

Are there competing modules in the target phase? If there are multiple parallel modules, it is very likely that those modules would generate competing strategies for ranking. As an example, consider Question Named Entity Recognition with multiple modules. Module M_1 can be a rule-based named entity extractor, module M_2 could be a CRF-based named entity extractor, and module M_3 could be a dictionary look-up based named entity extractor. Those named entity extractors will generate multiple and parallel named entities.

7.1.2 Non-shared modules

Are modules in a phase non-shared by all the competing results? Modules are shared if they are designed to use forms of analysis or resources that apply to every input. Those modules can hardly provide addition information if all the results are generated or related to the same set of modules. In contrast, modules are non-shared, if they apply to input depending on certain input types. Consider the example that those modules in the feature extraction step of question classification. A feature extractor scoring temporal information may require that a temporal expression occur in the clue of the question. It is common that the feature vector may have missing values. And therefore we hope to add a response indicator, called a "missing" flag, for each feature indicating whether it is missing. This "missing" flag may provide important additional information in estimating ranking for the candidates, providing a different signal from a zero feature score [32].

7.1.3 Competing output

Does a module or a phase generate multiple competing outputs? If it does, it can be important to evaluate which output is more likely to be the correct result. For example, consider a passage retrieval module that typically retrieves a large number of results. Amongst those returned results, some are more relevant than the others to the given query. If output is produced with uncertainty we hope to mathematically model this character in the system object graph. However, if all the derivation path share the same output, modelling this factor is not beneficial for ranking. For example, during the key term extraction step, it is possible to remove some stop words that are important to infer the answer or stem some words inappropriately. It is difficult to collect information to inform

better ranking decision from such must-do process unit in a system, for every output is produced by the same key term extraction method.

7.1.4 Output as Supporting evidence

Does a module or a phase derive evidence to support ranking candidates? For example, answer scorer modules have feature functions. Those modules show how answer relevance scores are generated for ranking answers.

7.2 Merging Phases

After a set of possible phases is detected, some adjacent phases can be merged into a single phase. The main reason for us to merge phases is to simplify the system object graph to avoid large number of dependency features between phases with a relatively small number of training instances. This task can be difficult because it can be subjective to determine which phases should be merged. We use one rule to merge co-related phases. To further compress the system object graph, we merge phases empirically.

The rule for merging co-related phases is that we can combine the two phases together if the modules of the former phase provide output exclusively for the later phase. In this case we do not explore the dependency features between phases. For example, consider the two phases of query formulation and passage retrieval. The two sequential phases are highly co-related: the Indri query formulator only provide search queries for local passage search while the web query formulator only provide search queries for web passage search. Those two phases provide redundant information for ranking in terms of the output derivation path. And therefore, we can safely merge them as one phase.

7.3 An Example of Designing System Object Graph

In this section, we provide details to show how a system object graph can be derived from the QA system we described at Section refsec:systemoverview. According the approach proposed in section 4, it requires three key steps to achieve the above goal: a phase detection step, an element definition step and a phase merging step. For the phase detection step, first a checklist of processing units is identified and then we use a rule based approach to label the possible phases. For the element definition step, we need to specify input nodes, output nodes, module nodes and dependencies. Finally, we simplify the system object graph by merging the phases that can be grouped together.

7.3.1 Identifying Phases

According to the description defined in Section 6.2, our system pipeline for open domain factoid question answering contains three logical phases: question analysis, passage retrieval, and candidate extraction. However, it is not a good option to start with building the system object graph from such logical phases. There are several reasons. The first

Table 7.1: Summary of the processing units, phase rules and our decisions.

Id	**Processing unit**	**Alternative modules**	**Multiple output**	**Uncertainty& non-sharing output**	**Phase Decision**
1	Key Term Extraction	No	No	No	No
2	Syntactic Analysis	No	Yes	Yes	Yes
3	Question Focus Detection	No	Yes	Yes	Yes
4	Feature Extraction	No	Yes	No	Yes
5	Answer Type Classification	Yes	Yes	Yes	Yes
6	Query Formulation	Yes	Yes	No	Yes
7	Passage Retrieval	Yes	Yes	Yes	Yes
8	Passage Normalization	No	No	No	No
9	Named Entity Recognition	Yes	Yes	Yes	Yes
10	Answer Candidate Scoring	Yes	Yes	Yes	Yes
11	Answer Merging	No	No	No	No

reason is that it can be subjective to define the modules since each logical phase may have more concrete sub-phases. The second reason is that those sub-phase can also have the potential to provide critical ranking information. The third reason is that we may not be able to identify correct dependencies for the system object graph with logical phases. Due to the above reasons, we decompose those logical phases into smaller processing units for more detailed analysis. Phases are selected from those processing units to form our system object graph by using the rules defined in Section 4.1. Table 7.1 shows those processing units abstracted from our QA system and our phase decisions. We provide the detailed analysis as follows.

1. Key term extraction
   (a) There are no alternative modules for key term extraction in our system. Every query are based on the output of key term extraction.
   (b) Our key term extractor is responsible for removing stop words and normalizing the key terms, so there is a single output generated from this processing unit.
   (c) The key term extractor works well for most questions. But it is not sure if the processing can be harmful to some questions. All the outputs shares the same processing step, so this processing unit has sharing output.
   (d) Based on those observations, we do not label the key term extractor as a phase.
2. Syntactic analysis
   (a) There are no alternative modules for syntactic analysis. Stanford Parser [1] is used to detect a phrase structure tree for a given question.
   (b) There is a single output tuple (unigrams, bigrams, a structured tree, question focus) generated from this processing unit.
   (c) This processing unit generates a grammatical structure of a question by using probabilistic algorithms, so it is possible to have parsing errors. The output is not shared by all derivation path, because the output tree structure is mainly used for type-based question classification, and is not used in interrogative word based question classification approach.
   (d) We label the processing unit of syntactic analysis as a phase.
3. Question focus detection
   (a) There are no alternative modules.
   (b) There is a single output generated from this processing unit.
   (c) A rule-based focus detector is employed to detect the question focus, so it is possible to have detection errors. The output is not shared by all derivation path for the same reason of syntactic analysis.
   (d) We label the processing unit of question focus detection as a phase.
4. Feature extraction
   (a) There are no alternative modules for the purpose of feature extraction.

[1]Stanford Parser, http://nlp.stanford.edu/software/lex-parser.shtml

(b) There are multiple features generated from this processing unit. The features we used for answer type classification fall into three categories: lexical, syntactic, and semantic.

(c) No uncertain output is generated. The output is not shared by all derivation path for the same reason of syntactic analysis.

(d) We label the processing unit as a phase.

5. Answer type classification

(a) There are alternative modules for answer type classification.

(b) There are multiple outputs generated from the classification task.

(c) Question classification tasks are not accurate, and both the modules and outputs are not shared by all the derivation paths.

(d) We label the processing unit as a phase.

6. Query formulation

(a) There are alternative modules for the query formulation task. The passage retrieval phase generates the sequential dependency query for local search and key term based query for web search.

(b) Each module only generate one query per question. Since there are two different modules, there are multiple queries produced for retrieval task.

(c) Query formulation does not introduce any uncertain information. Both the modules and outputs are not shared by all the derivation paths.

(d) We label the processing unit as a phase.

7. Passage retrieval

(a) There are alternative modules for the passage retrieval task. We use Indri to perform passage retrieval on local indexes. We also employ various web search APIs to retrieve passages online.

(b) Each module often returns multiple passages.

(c) Relevance is used denote how well a retrieved passage or set of documents meets the information need of the question. Since each answer candidate is generated from a different passage, both the modules and outputs are not shared by all the derivation paths.

(d) We label the processing unit as a phase.

8. Passage normalization

(a) Passage normalization is used to remove punctuations, tokenize the text and stem words. There is no alternative normalization approaches.

(b) Text normalization is the process of transforming text into a single canonical form, so there are no multiple output.

(c) There is little uncertainty for the normalization process, though text normalization requires being aware of what type of text is to be normalized. All the

derivation paths share the same processing step, though the outputs are different.

(d) The processing unit is not labelled as a phase.

9. Named entity recognition
   (a) Multiple NER modules are employed in our QA system to detect answer candidates.
   (b) Given an input passage and the answer type, each NER module often generates multiple outputs.
   (c) NERs can introduce errors. For example, one third party software is Stanford NER, which is also known as CRFClassifier. The software provides a general implementation of linear chain Conditional Random Field (CRF) sequence models. Its produced results are accurate to some categories but not to all the question types. Both the modules and outputs are not shared by all the derivation paths.
   (d) The processing unit is labelled as a phase.
10. Answer candidate scoring
   (a) There are alternative modules for the purpose of scoring answer candidates.
   (b) There are multiple scores produced from this processing unit (e.g., passage relevance scores and proximity scores).
   (c) Many scores are estimated so they are uncertain. Different answer candidates have different output scores that are not shared by all derivation path.
   (d) We label the processing unit as a phase.
11. Answer merging
   (a) There are no alternative modules for merging answer candidates.
   (b) Because we only merge redundant answer candidates, there are no multiple outputs.
   (c) Merging scores are still estimates so they are uncertain. Different answer candidates have different output scores that are not shared by all derivation path.
   (d) We label the processing unit as a phase.

7.3.2 Defining Elements

Give those possible phases, we need to identify input nodes, output nodes and module nodes for each phase to construct the system object graph with dependencies. A dependency declares the relation between two nodes. Given the detected phases, Figure 7.1 shows the system object graph with dependencies. Those basic elements for each phase are described as follows.

1. Syntactic analysis
   (a) Input: the given question
   (b) Output: the grammatical structure of the given question

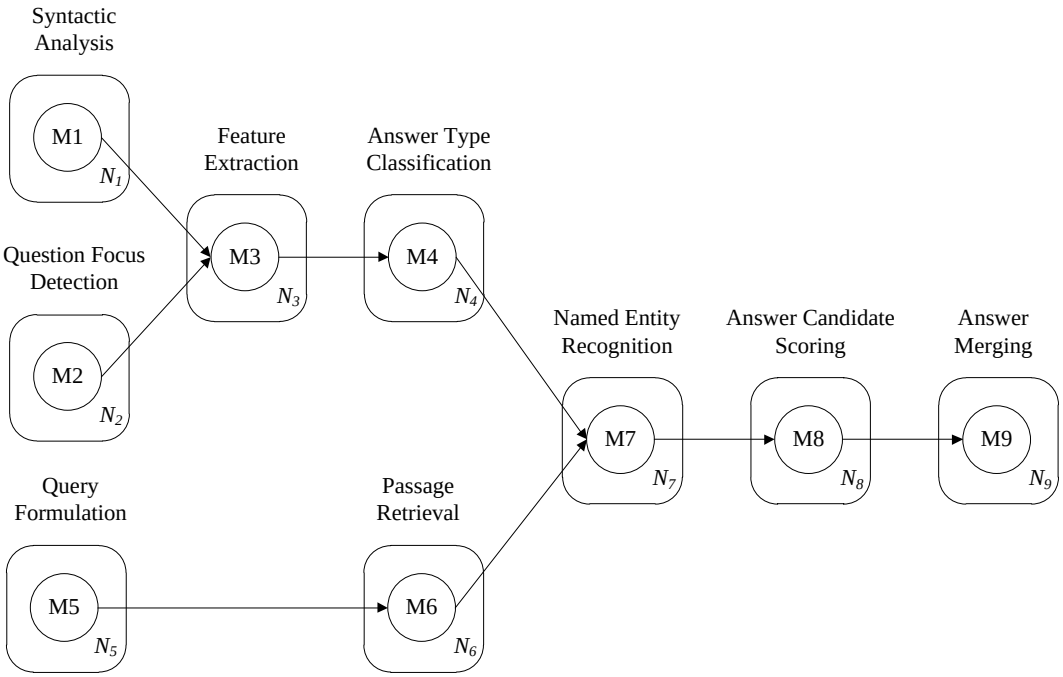

Figure 7.1: The initial system object graph for our QA system, input nodes and output nodes are omitted.

(c) Module: the Stanford Parser

2. Question Focus Detection
   (a) Input: the given question
   (b) Output: the question focus
   (c) Module: the question focus detector
3. Feature extraction
   (a) Input: unigrams, bigrams, the grammatical structure, and the question focus
   (b) Output: a set of features
   (c) Module: the feature extractor
4. Answer type classification
   (a) Input: the feature set
   (b) Output: a set of answer types
   (c) Module 1: a statistical learning based question analyzer, as presented in [77]. This approach classifies each question into 44 top-level categories (i.e., location, date, proper name, number) and 110 subcategories (i.e., airport, day, person, zip code, etc.). For instance, for the question "What is the name of the volcano that destroyed the ancient city of Pompeii?" the expected question type would be "location" as the top-level category, and "mountain" for the subcategory, represented together as "location:mountain".

(d) Module 2: a rule-based question analyzer [77] that uses the same answer type hierarchy as the first model. The rules determine the type of a question from features such as focus word and question word. For example, the question "What are the colors of the Italian flag?" is classified as the "color" type because its focus word is "colors".

(e) Module 2: a default question analyzer that uses a question's interrogative word to determine the answer type. The interrogative word (e.g., "what" or "how much") can indicate the general answer type, but cannot provide more fine-grained answer type information. We extend the two-level classification scheme to include 7 general interrogative types, yielding a a three-level answer type hierarchy: 7 general, 44 coarse, and 110 subcategories.

5. Query formulation
   (a) Input: the given question
   (b) Output: the query for the given question
   (c) Module 1: For local indexes, we adopt the sequential dependency model.
   (d) Module 2: For web searches, we use the simplest bag-of-words model without functional words and stop words.
6. Passage retrieval
   (a) Input: the generated query
   (b) Output: the retrieved passage list
   (c) Module 1: The AQUAINT corpus originally used for the TREC evaluations was indexed by using the open-source IR system Indri [2]. When searching through local indexes, we composed queries by using the method proposed by Metzler and Croft [58]. This method linearly expands the original query with subqueries that add both ordered and unordered proximity constraints. The parameters were tuned empirically to combine the original query model and the sequential dependency model.
   (d) Module 2: An encyclopedia corpus, a Wikipedia [3] data dump (a version downloaded in 2008, which contained 2,114,707 articles), was indexed to support local search.
   (e) Module 3&4: Two search agents were used to retrieve relevant short snippets from a web search engine (Bing Search [4] API), with different periods and locations. For each question, a query consisting of question keywords was sent to the search engine, and the results were cached locally.
   (f) Module 5&6: Two local indexes were built by using documents retrieved from Bing Search API and Wikipedia Search API separately. To do that, we first composed a query and sent it to search APIs, and then downloaded and indexed

[2]Indri, http://www.lemurproject.org/indri/
[3]Wikipedia, http://www.wikipedia.org/
[4]Bing Search, http://www.bing.com/

the text content of the top retrieved documsents. We repeated the process for each question contained in our dataset.

(g) Module 7: Similar to web search engine based agents, we retrieved relevant users' answers from a cQA website, Yahoo! Answers [5] API.

(h) Module 8: The microblog website Twitter [6] Search API was consulted to retrieve top relevant tweets.

7. Named entity recognition
   (a) Input: the retrieved passage list
   (b) Output: the named entity list
   (c) Module 1: List based NER (Named Entity Recognizer) was a part of OpenEphyra's information extractor. It used user-defined dictionary for named entities look-up.
   (d) Module 2: Pattern based NER is also a part of OpenEphyra's information extractor. It used regular expression patterns that match different types of named entities. Two additional pattern based NERs, numeric based and date based, were developed to reinforce the main pattern based NER.
   (e) Module 3: The OpenNLP [7] NER has a number of pre-trained name finder models including date, location, money, organization, percentage, person, and time. Those models are trained on various freely available corpora.
   (f) Module 4: Stanford NER contains 3 class named entity recognizers (person, organization, and location), trained by a linear chain CRF model [30].
   (g) Module 5: An item based NER and a proper name based NER were developed to exhaustively extract single items as answer candidates. Heuristic rules (e.g., stop words, unsuitable POS tagging and a particular set of named entities under some question types) were applied to filter out obvious wrong answer candidates.
8. Answer candidate scoring
   (a) Input: the named entity list
   (b) Output: the answer candidate list
   (c) Modules:
9. Answer merging
   (a) Input: the answer candidate list
   (b) Output: the merged answer candidate list
   (c) Modules:

[5]Yahoo! Answers, http://answers.yahoo.com/
[6]Twitter, http://twitter.com/
[7]OpenNLP, http://opennlp.apache.org/

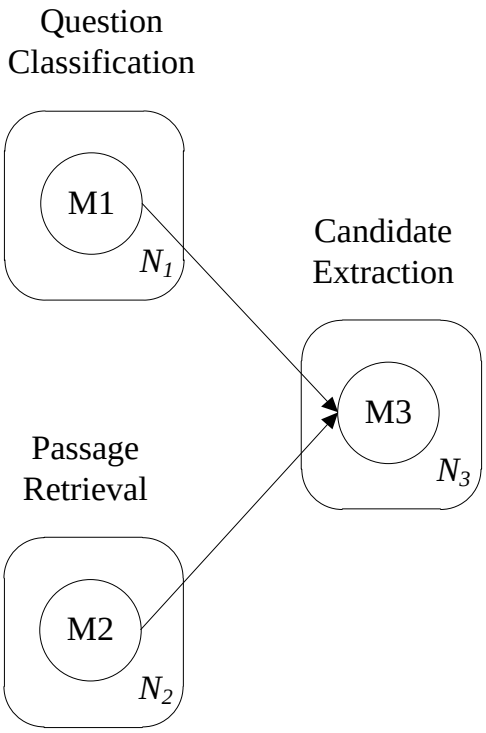

Figure 7.2: The simplified system object graph, input nodes and output nodes are omitted.

7.3.3 Merging Phases

In many cases, system object graphs can be simplified by merging co-related phases to prevent a complex graph with a large number of phases and complex dependencies amongst those phases. For example, consider the two phases of query formulation and passage retrieval. The two sequential phases are highly co-related: the indri query formulator is defined for local passage search while the web query formulator is defined for web passage search. Those two phases provide redundant information for ranking in terms of the output derivation path. And therefore, we can safely merge them as one phase. Here we heuristically merge co-related phases, as a result our system object graph is simplified from Figure 7.1 to Figure 7.2.

Chapter 8

Case Study: Biomedical Information Retrieval

In the previous sections, we introduced a general method for modeling information systems with multiple sequential phases. The method mainly includes a system object graph representation and a phased ranking model. The former part captures information about every processing option and intermediate data object produced by processing a single input. Based on the representation, a phased ranking model leverages information propagated from preceding phases to inform the ranking decision. We applied our proposed approach to the task of factoid question answering, and demonstrated that our approach is statistically significantly better than or indistinguishable from standard baseline approaches for TREC datasets 9-12. Although tested on factoid question answering, we believe that our approach is general and can be used for domain adaptation in many other NLP applications with sequential phases and multiple options per phase.

This chapter presents the application of the proposed phased ranking method to support result ranking for a state-of-the-art system, BioQA [103], in the domain of biomedical information retrieval. In Section 8.1, we describe the biomedical information retrieval task. In Section 8.2, we describe the background and prior work about the previous ranking approaches in biomedical information systems. In Section 8.3 and Section 8.4, we describe the steps of building the system object graph and applying the proposed phased ranking framework for the BioQA system by following the general solution procedure proposed in Chapter 4 and Chapter 5. In Section 8.5, we report the results and compare the Phased Ranking model to several baselines. In Section 8.6, we perform overlap analysis from the two approaches, our proposed Phased Ranking model and the Generalized Watson Rank.

8.1 Task Description

In the biomedical domain, questions contain topics related to biological objects (e.g., genes, proteins, gene mutations, etc), biological process (e.g., physiological processes or diseases), and relationships (e.g., causes, contributes to, effects, associated with, or regulates).

The task of the TREC Genomics Track is to develop an information system that focused

on retrieval of documents or short passages that specifically addressed an information need in the biomedical domain. Those retrieved results usually contain several sentences that are longer than phrases but shorter than a paragraph in length. For example, consider the following question,

- TREC 2006 Genomics Question 160: What is the role of PrnP in mad cow disease?

The goal is to find articles and short passages describing the role of the gene PrnP in the mad cow disease from a full-text biomedical corpus [38]. And therefore, to answer the above question, a system should return relevant PubMed articles (document id, short passage start and end position) such as,

- PMID: 10364247 START:2479 END:2694
  - PrP^{Sc}, an abnormal isoform of PrP^C, is the only known component of the prion, an agent causing fatal neurodegenerative disorders such as bovine spongiform encephalopathy (BSE</A>)
- PMID: 10466827 START:4827 END:5869
  - Transmissible spongiform encephalopathies (TSEs) or prion diseases combine the molecular genetics and epidemiology associated with non-infectious, genetic disorders (gene mutations, familial forms of disease) with the characteristics of virus diseases (transmissibility, strain variation)...
- PMID: 10573173 START:3282 END:356
  - The PrP gene encodes the putative causative agent of the transmissible spongiform encephalopathies (TSEs), a heterogeneous group of fatal, neurodegenerative disorders including human Creutzfeldt ♯ 150;Jakob disease, bovine spongiform encephalopathy

8.1.1 Evaluation metrics

The TREC 2006 Genomics QA task involves retrieval of short passages that specifically address an information need expressed as a question, and the answers are provided along with a reference to the location of the answer in the original source document. A test collection of 162,259 full-text documents and 28 topics expressed as questions was assembled.

In order to automate the evaluation process, we need to define metrics to estimate the retrieval result quality. We employed both the document collection and the topic set from the official evaluation and focused on two task-specific metrics: DocMAP and PsgMAP. Intuitively, DocMAP measures the relevance of the documents retrieved by the system, regardless of the relevance and conciseness of the passages extracted. The PsgMAP metric also considers the relevance of extracted passage spans.

For the purposes of DocMAP, a PMID is considered a relevant document for that question if and only if it is contained in the set of gold standard PMIDs. All other documents were considered not relevant for that question. The DocMAP measure was the mean of the average precisions across all the questions, where average precision was measured across recall levels for a question. In particular, they are defined as follows,

$$Precision@K = \frac{\text{true positive}@K}{K}$$
$$AveragePrecision@N = \frac{1}{m}\sum_{i=1}^{m} Precision(R_i)$$
$$MAP@N = \frac{1}{|Q|}\sum_{i=1}^{|Q|} AveragePrecision@N_i$$

For a single question, the measure $Precision@K$ sets a rank threshold K, and computes the percentage of relevant documents in the set of top K documents. The measure $AveragePrecision@K$ considers rank position of each relevant document R_i, it first computes $Precision@K$ for each R_i and then average those scores.

PsgMAP measure is a variation of MAP, computing precision scores for short passages based on character-level precision. For each retrieved passage, $Precision@K$ was computed as the fraction of characters overlapping with the gold standard passages divided by the total number of characters included in all the K passages. Then the mean of these average precisions over all questions was calculated to compute the MAP for passages [38].

8.2 Previous Work

We focus on summarizing biomedical information retrieval systems and their ranking approaches that have participated in tasks or related to TREC Genomics[38], or have published results by employing the task benchmarks outside the formal competition.

A biomedical information retrieval pipeline typically consists of three major analysis components: query or question analysis, document (or passage) retrieval, retrieved result extraction, and result ranking [71, 87, 103, 105]. Specifically, they are functioned as follows,

1. Query or question analysis components take a query or question as an input and analyze the query or question text to generate an object containing a list of keywords, alternative terms for query expansion, etc.
2. Document or passage retrieval components take the objects from the previous phase as inputs and produce a list of relevant documents or passages retrieved from the corpus.
3. Extraction components produce a list of target candidates by extracting from the documents or passages.
4. Result ranking components generate a list of ranked answers as an output.

To rank the retrieval results, W. Zhou, et al. [105] employed a variation of Okapi relevance scoring formula that not only considered relevance based upon term occurrences within text documents and the length of the documents, but also considered the case when a query contains multiple concepts. In particular, they used a concept retrieval model by combining concept similarity and word similarity.

1. Concept similarity is defined as $sim_{concept}(q, d)$, given a query q and a document d,

$$sim_{concept}(q, d) = \sum_{c} log(\frac{N}{n}) \tag{8.1}$$

   where N is the size of the document collection and n is the document frequency of the concept c.

2. Word similarity is defined as $sim_{word}(q, d)$, computed by using Okapi,

$$sim_{word}(q, d) = \sum_{w} log(\frac{N - n + 0.5}{n + 0.5}) \frac{(k_1 + 1)tf}{K + tf} \tag{8.2}$$

   where N is the size of the document collection, n is the number of documents containing word w, and tf is the term frequency within a document.

Finally, this concept retrieval model used a heuristic rule to combine the two similarity models by emphasizing the concept similarity more than the word similarity.

8.3 System Object Graph Definition

As the initial work of the proposed phased ranking approach, we will build the system object graph for the state-of-the-art BioQA system by using the approach proposed in Section 4 with the following steps: 1) listing all the processing units and determine the phases and defining all the elements per phase in Section 8.3.1, and 2) simplifying the phases for the final system object graph in Section 8.3.2.

To facilitate easy modeling with existing components, we require a system that can be converted to or has well-defined phases to shift through the phased ranking model to make a ranking decision. The BioQA system [103] is such an example that has a standalone workflow representation with the YAML based process modeling language which is independent of implementation details (e.g. the programming language). Due to the accessibility, the components of the system can be easily converted to phases.

8.3.1 Phase Definition

In this section, we detail the steps of converting components from the BioQA system to phases of the system object graph.

The BioQA system is composed of biomedical analysis components, including the basic input/output definition of analytics engine, data processing components (e.g. keyterm extraction, passage retrieval component, etc.), intermediate data objects (such as keyterms, relevant passages, etc.), and analytics task evaluation components. We focus on analysis

components of the pipeline including the Keyterm Extraction component, POS Tagging component, Named Entity Extraction component, Lexical Variant Extraction component, Synonyms Detection component, Keyterm Refining component, Document Retrieval component, LegalSpan Passage Retrieval component, Important Sentence Extraction component, Proximity-based Result Scoring component, and Final Result Ranking component.

Keyterm Extraction

This component extracts Keyterms (or keywords) as metadata representing the information need of a query or question. We define the Keyterm Extraction phase, including an input node (a question), a module node (a keyterm extractor), and an output node (keyterms), shown as Figure 8.1.

For example, consider TREC Genomics 2006 question 160 "What is the role of PrnP in mad cow disease?". First the question is sent into the keyterm extractor and then keyterms were produced such as "What", "is", "the", "role", "of", "PrnP", "in", "mad", "cow", and "disease".

POS Tagging

A Part-Of-Speech Tagger (POS Tagger) is used in this component that reads text and assigns parts of speech to each word (and other token). We define the POS Tagging phase, including an input node (a question), a module node (a POS Tagger), and an output node (tagged keyterms), as shown in Figure 8.2.

For example, first the question is sent into the POS Tagger and then the part of speech tags were assigned to each word such as "What:.", "is:NN", "the:DD", "role:NN", "of:II", "PrnP:NN", "in:II", "mad:VVN", "cow:NN", and "disease:NN".

Named Entity Extraction

A named entity extractor is used to extract named entities from a query or question. The goal of this component is to detect named entities such as gene names and diseases. Those named entities are added into the metadata keyterm set. We define the Named Entity Extraction phase, including an input node (a question), a module node (a name entity recognizer), and an output node (named entities), as shown in Figure 8.3. For example, named entities such as "PrnP" and "mad cow disease" can be detected from the input question.

Lexical Variant Extraction

This component detects gene names and their lexical variants by using different rule-based approaches. The lexical variants of a gene may come from two ways: 1) rules according to human gene nomenclature; 2) retrieved from an abbreviation database. Those named entities are added into the metadata keyterm set. The lexical variants plays the same role as the synonyms of keyterms. We define the Lexical Variant Extraction phase, including an input node (a question), a module node (a lexical variant resolver), and an output node

(lexical variants), as shown in Figure 8.4. In the example question, there is no keyterm that can be matched to our rules so it returns "Null" for each keyterm.

Synonyms Detection

Domain knowledge is leveraged at this component for keyterm expansion. The phenomenon is very common in the biomedical domain that a biomedical term can have many different surface forms. For example, "mad cow disease", "Bovine Spongiform Encephalopathy" and "BSEs" refer to the same disease. This component relies on three resources to expand a keyterm: UMLS, EntrezGene, and MeSH [103]. We define the Synonyms Detection phase, including an input node (the metadata keyterms), a module node (a synonym expander with three resources UMLS, EntrezGene, and MeSH), and an output node (the expanded keyterms), as shown in Figure 8.5.

Keyterm Refining

This component is to remove duplicated synonyms and assign weights to keyterms according to different types. We define the Keyterm Refining phase, including an input node (the metadata keyterms), a module node (a keyterm refiner), and an output node (refined and weighted keyterms), as shown in Figure 8.6.

Document Retrieval

A commonly used retrieval toolkit, Indri [84], is used to index a biomedical corpus using title, abstract and keywords fields. During the query time, all stopwords from the query are removed. The query is formulated by the metadata keyterms and is expanded by those keyterms' lexical variants and synonyms. In the implementation, top 1,000 ranked documents are returned by using Indri's retrieval model with Dirichlet smoothing. We define the Document Retrieval phase, including an input node (the metadata keyterms), a module node (a Indri-based retrieval strategist), and an output node (ranked documents), as shown in Figure 8.7.

LegalSpan Passage Retrieval

A LegalSpan passage is defined as a contiguous list of sentences from a paragraph. LegalSpan passage retrieval component retrieves LegalSpan passages from an annotated corpus index. During the corpus annotation and indexing step, the TREC Genomics corpus from the organizer was annotated by the LegalSpan annotations. The retrieval model searches and returns LegalSpan passages instead of documents. We define the LegalSpan Passage Retrieval phase, including an input node (the metadata keyterms), a module node (a Indri-based retrieval strategist), and an output node (LegalSpan passages), as shown in Figure 8.8.

Important Sentence Extraction

Important sentences are defined as a small number of sentences with a passage. In this component, the maximum number of sentences in a passage is limited to be k. The Important sentences are extracted from returned LegalSpan passages by measuring the similarity between each sentence and the metedata keyterms, as well as the neighboring sentence. We define the Important Sentence Extraction phase, including an input node (the returned LegalSpan passages), a module node (an important sentence extractor), and an output node (important sentences), as shown in Figure 8.9.

Proximity-based Result Scoring

This component scores the important sentences by using a proximity-based ranking formula. We define the Proximity-based Result Scoring phase, including an input node (the extracted important sentences), a module node (a scorer), and an output node (the scored important sentences), as shown in Figure 8.10.

Final Passage Ranking

This component scores and ranks the important sentences by combining several ranking scores, including document retrieval scores, passage retrieval scores, and term proximity based scores. We define the Final Passage Ranking phase, including an input node (the extracted important sentences), a module node (a score combiner), and an output node (the scored important sentences), as shown in Figure 8.11.

8.3.2 Phase Merging

After identifying the system components as the processing units, we will describe the merging steps that produce the final phases to form the system object graph. On one hand, we are interested in those processing units generating competing strategies and output objects, because the relationship between the phases can be modeled to help make ranking decision. On the other hand, some other processing units should be merged into one phase.

We construct those merged phases sequentially to form the system object graph. Three steps are used to merge phases in a top-down manner. First, those 11 processing units are divided into three groups according to their module object similarity and output object similarity. The processing units from 1 to 6 employed natural language processing (NLP) of the questions and/or target texts, using algorithms, toolkits, and pre-trained models for sentence segmentation, tokenization, part-of-speech tagging, named entity recognition, lexical variants, synonym expansion, etc. All their outputs are used to expand the metadata keyterms. The processing units from 7 to 8 is employed to retrieve relevant documents and LegalSpan passages from the unstructured biomedical corpus, relying on a widely-used open-source search engine Indri. Both the outputs are retrieved results. The processing units from 9 to 11 is to segment a retrieved LegalSpan passage into multiple important sentences as the final candidates for ranking.

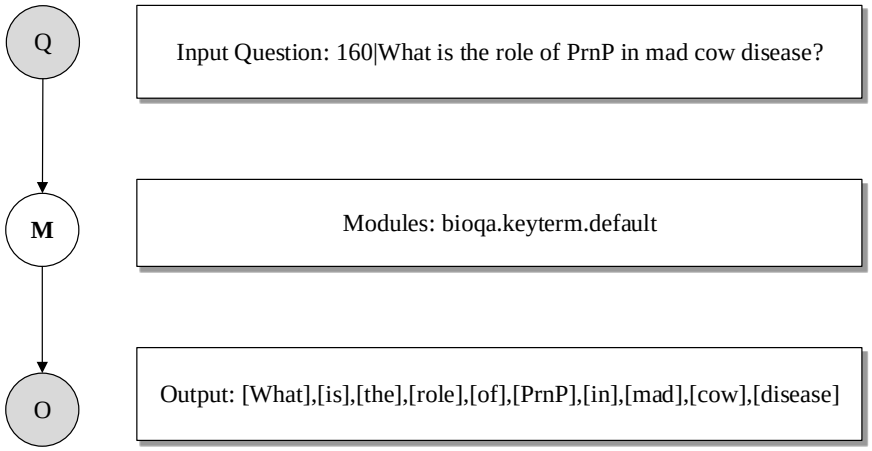

Figure 8.1: Processing unit 1: Keyterm Extraction

Second, representatives are selected from each group. We assigned a processing unit as a representative if it can produce competing strategies or output objects by considering the number of module and output objects. In the first group, the 5th processing unit Synonyms Detection is assigned as a representative because it generates multiple synonyms. In the second group, we assign representatives to Document Retrieval and LegalSpan Passage Retrieval phases, because both of them returned multiple competing outputs. In the third group, the processing unit of Important Sentence Extraction is selected as a representative.

Third, processing units are merged into the nearest representative as a final phase. We merge multiple representatives in a group, if they are independent (e.g., the input of a phase does not depend on the other phase). In the first group, the processing units are merged together to form the Keyterm Expansion phase, as shown in Figure 8.12. In the second group, the two independent processing units are merged together to form the Document/passage Retrieval phase, as shown in Figure 8.13. In the third group, those processing units are merged together to form the Important Sentence Extraction phase, as shown in Figure 8.14.

8.4 Features for Important Sentence Ranking

In this section, we will apply feature models presented in Section 5.3 to those phases defined in Section 8.3.2. In each phase, there are three node types, including input node, module node and output node. Each of the node contains a list of objects, and each of the object can be viewed as a random variable over its representation. Features are defined over their nodes in different ways. The first three feature groups are defined within individual phases, including Keyterm Expansion phase features, Document/Passage Retrieval phase features, Important Sentence Extraction phase features. The rest feature groups are defined between phases including Keyterm Expansion phase $\rightarrow$ Document/Passage Retrieval

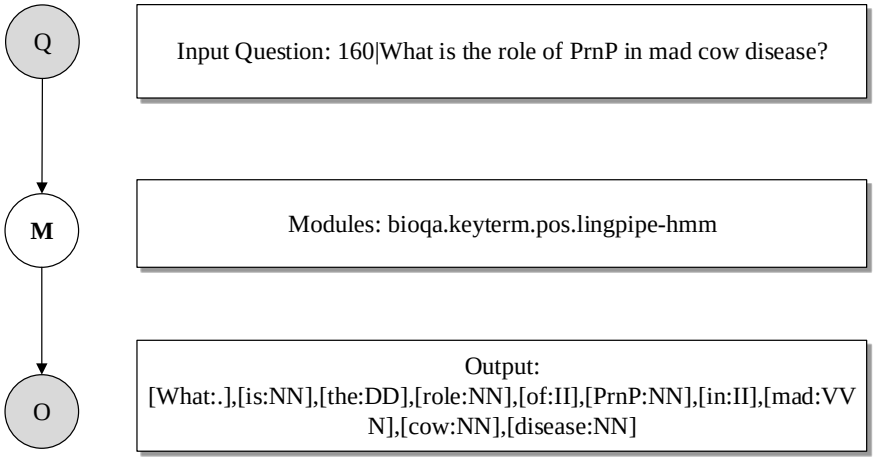

Figure 8.2: Processing unit 2: POS Tagging.

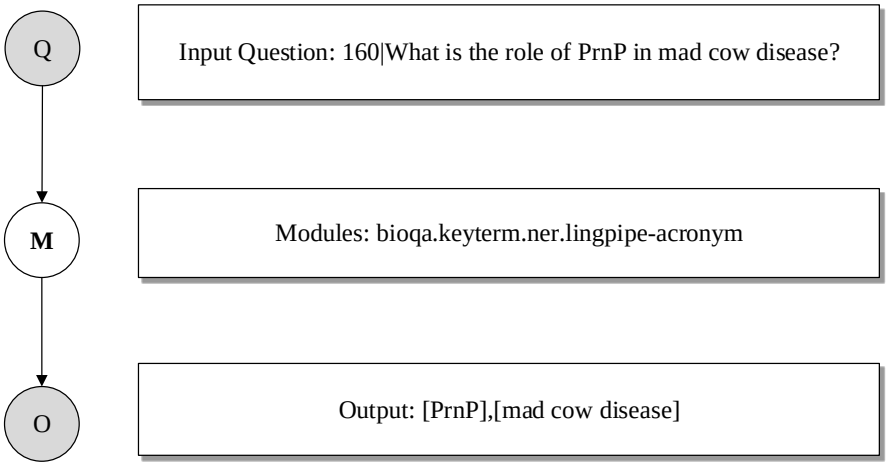

Figure 8.3: Processing unit 3: Named Entity Extraction.

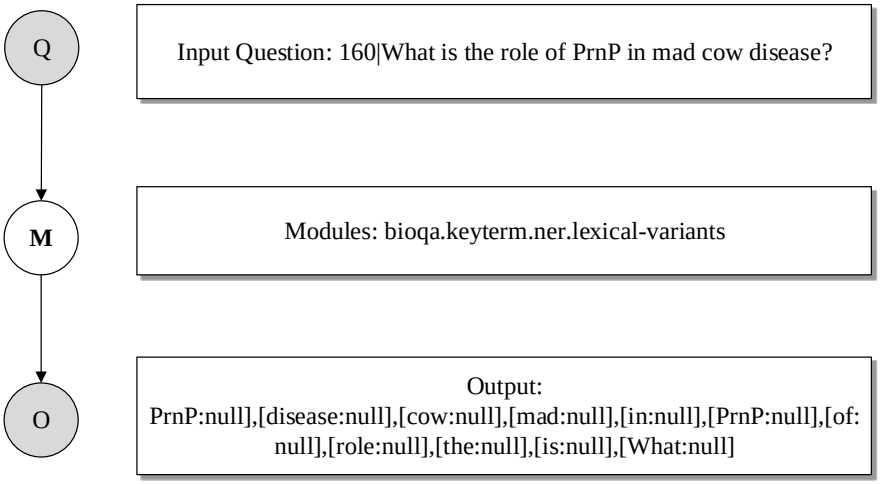

Figure 8.4: Processing unit 4: Lexical Variant Extraction.

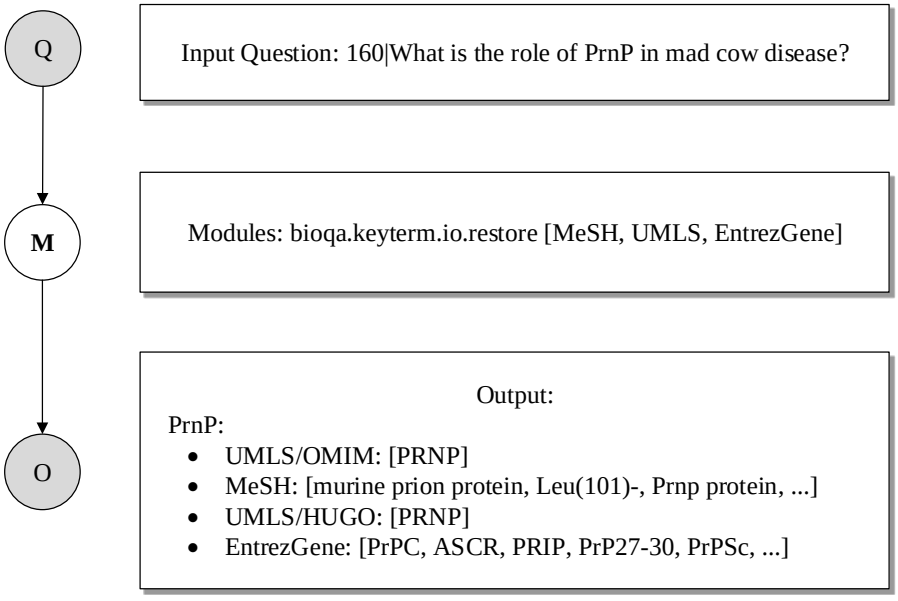

Figure 8.5: Processing unit 5: Synonyms Detection.

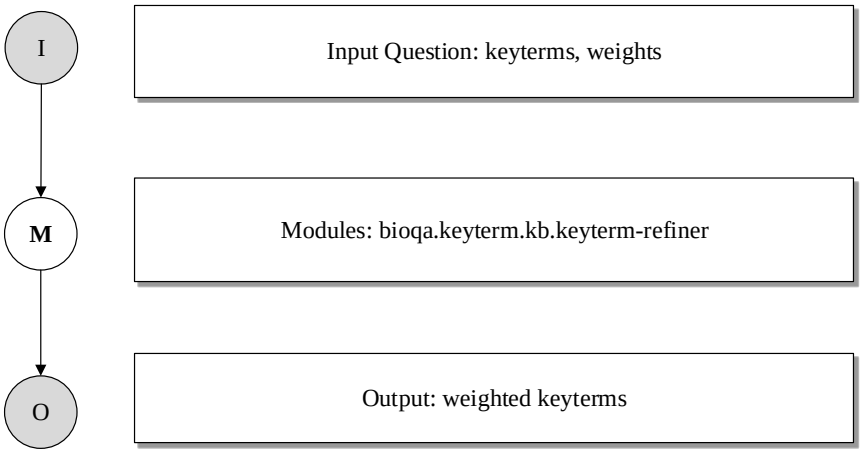

Figure 8.6: Processing unit 6: Keyterm Refining.

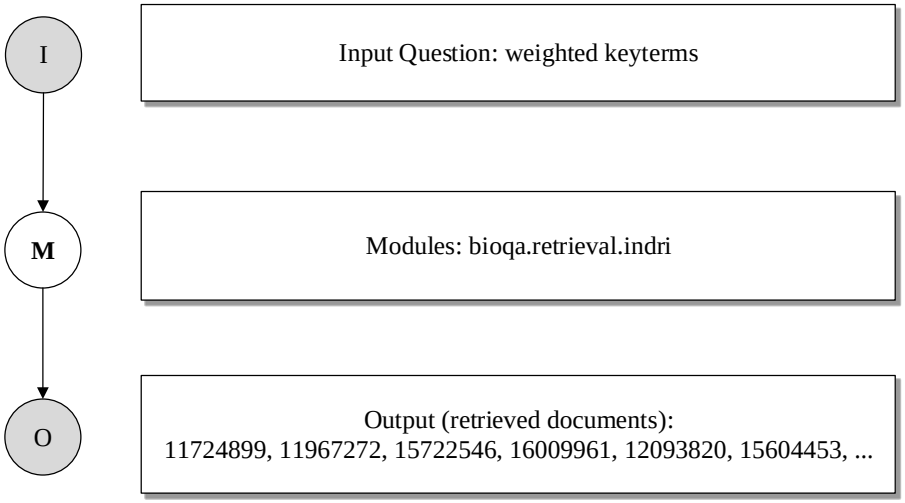

Figure 8.7: Processing unit 7: Document Retrieval.

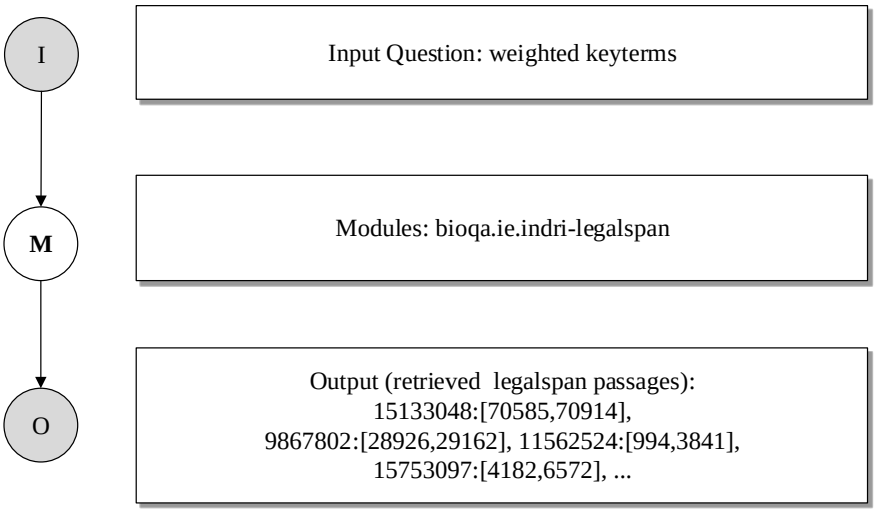

Figure 8.8: Processing unit 8: LegalSpan Passage Retrieval.

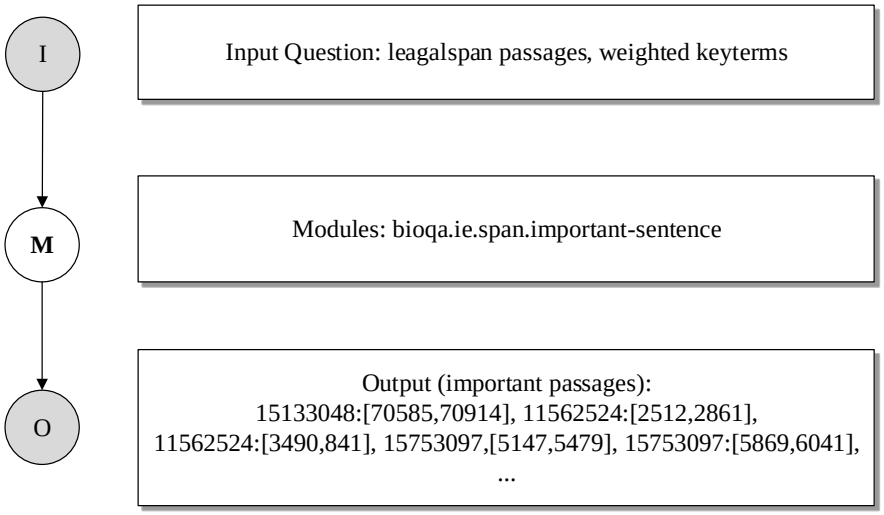

Figure 8.9: Processing unit 9: Important Sentence Extraction.

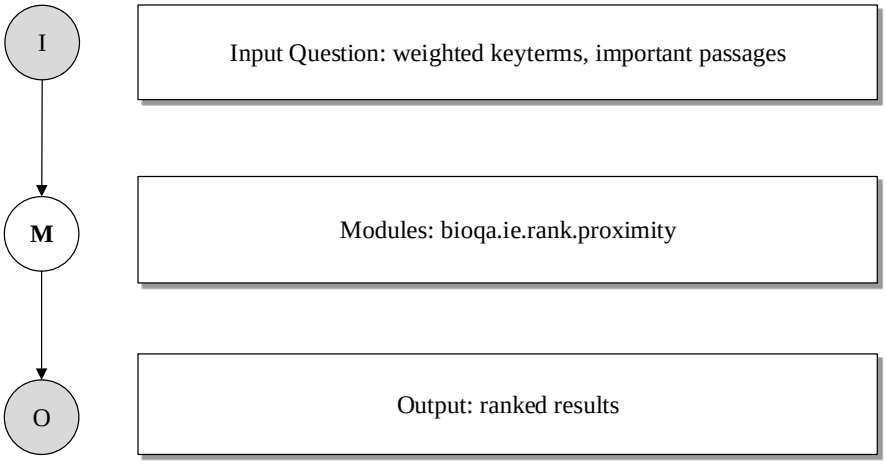

Figure 8.10: Processing unit 10: Proximity-based Result Scoring.

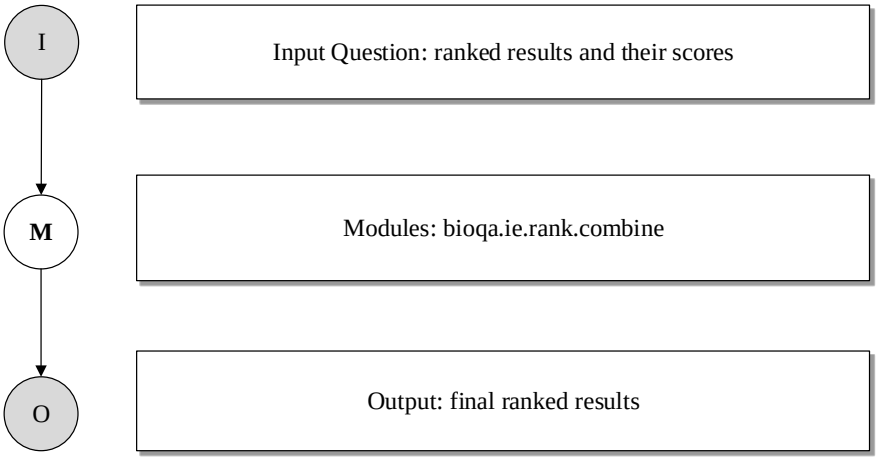

Figure 8.11: Processing unit 11: Final Passage Ranking.

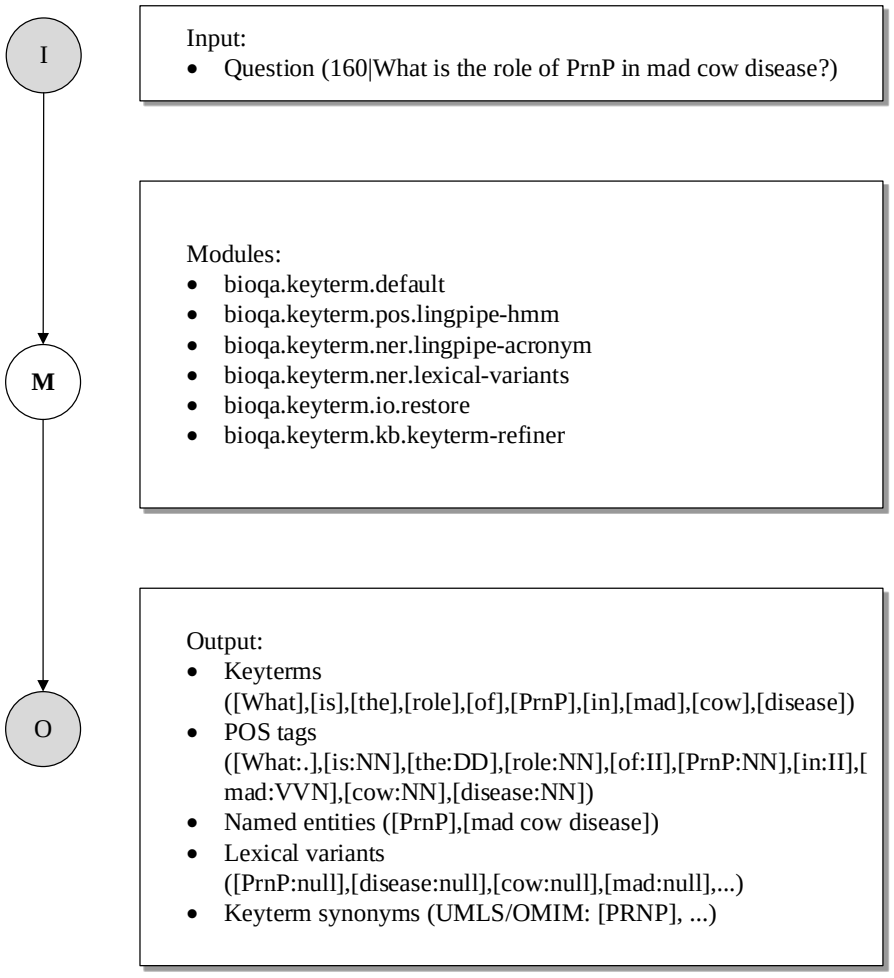

Figure 8.12: Processing unit merging from No.1 to No.6: Keyterm Expansion.

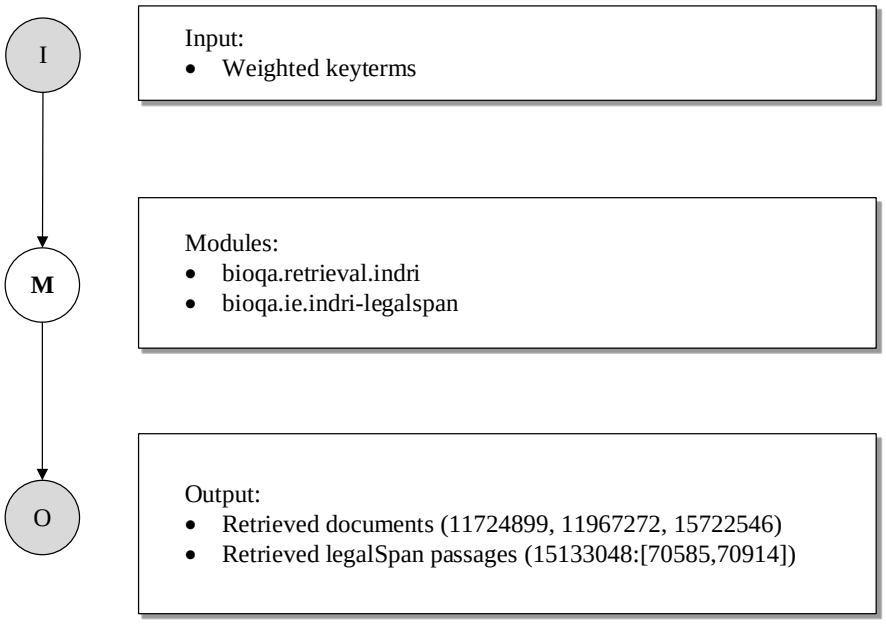

Figure 8.13: Processing unit merging from No.7 to No.8: Document/Passage Retrieval.

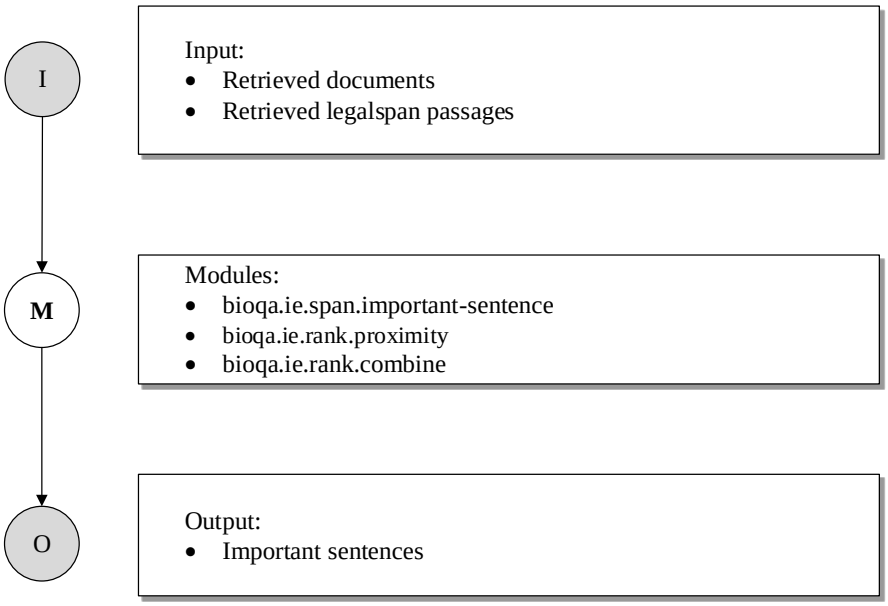

Figure 8.14: Processing unit merging from No.9 to No.11: Important Sentence Extraction.

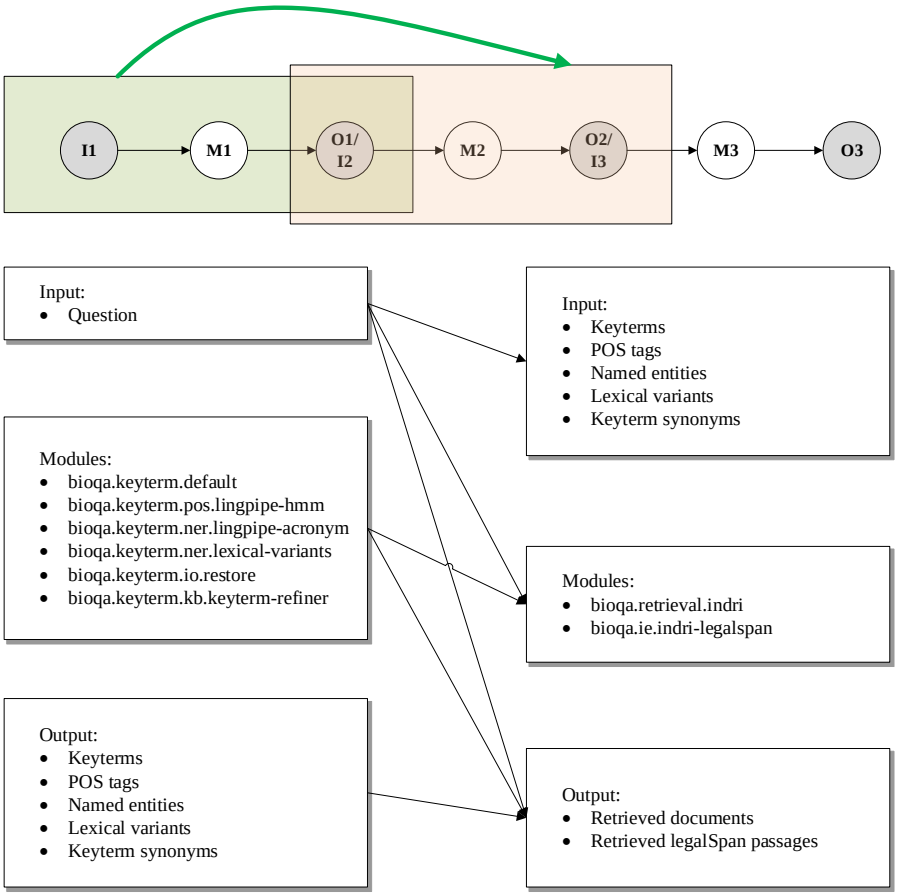

Figure 8.15: The phase dependency I: Keyterm Expansion → Document/Passage Retrieval.

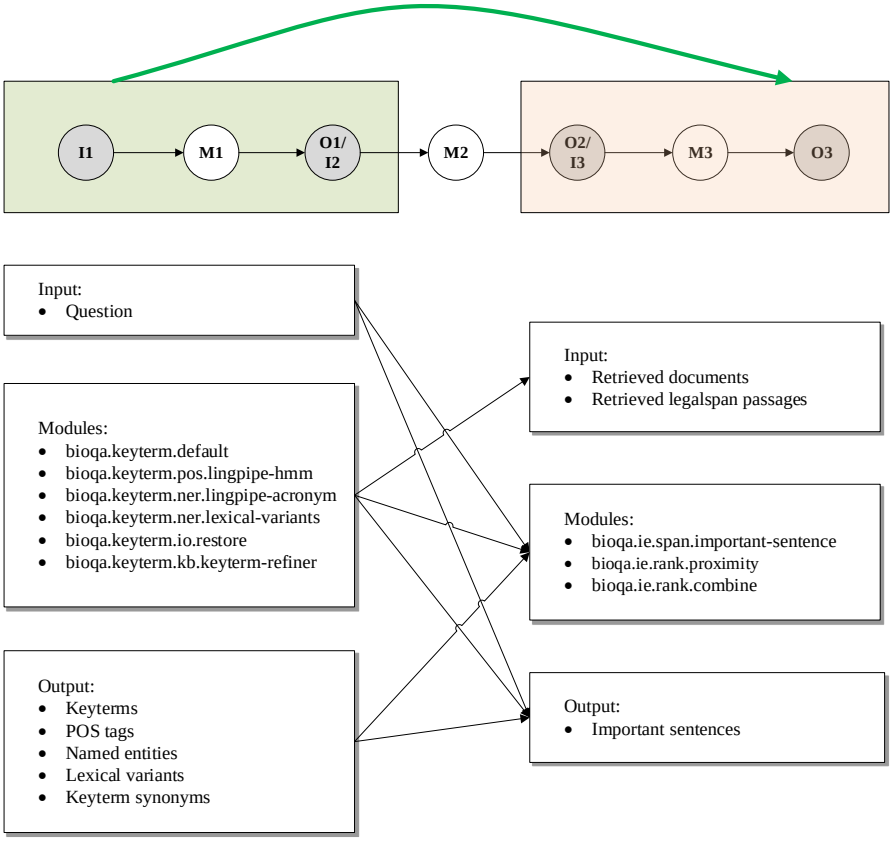

Figure 8.16: The phase dependency II: Keyterm Expansion → Important Sentence Extraction.

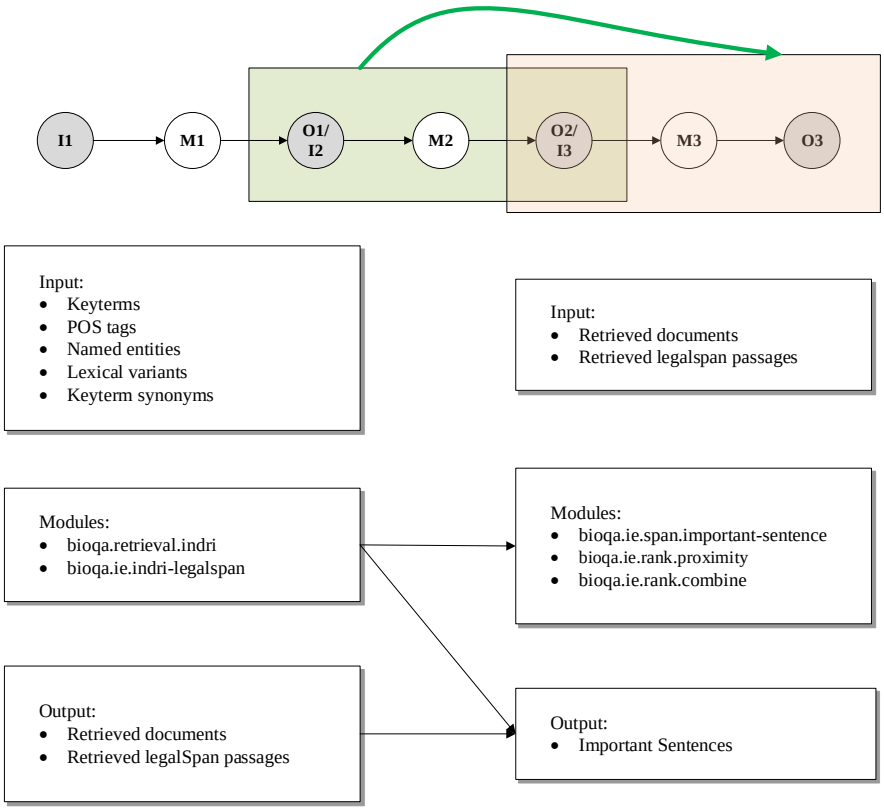

Figure 8.17: The phase dependency III: Document/Passage Retrieval → Important Sentence Extraction.

phase, dependency features: Keyterm Expansion phase → Important Sentence Extraction phase, and dependency features: Document/Passage Retrieval phase → Important Sentence Extraction phase.

8.4.1 Keyterm Expansion Phase Features

In Figure 8.12, the Keyterm Expansion phase is shown to have six module objects and five output objects representing the corresponding system components and their outputs. We use binary feature vectors to represent if an output is generated by a particular module object. Whenever the feature is present, the indicator emits a 1, otherwise it emits a 0. Those output objects can be viewed as metadata to the input, so we assign scores to them instead of directly defining features to the rank model.

The module object features (M_1) We do not assign any features for those modules because they do not generate competing strategies.

The output object features (O_1) We consider those output keyterms as another form of the input query or question. We do not assign any feature as prior knowledge to output objects.

The input → module dependency features ($I - M_1$) We do not assign any feature to module objects, because they do not generate competing strategies.

The module → output dependency features ($M_1 - O_1$) We assign values returned by the module to the corresponding keyterms.

8.4.2 Document/Passage Retrieval Phase Features

In Figure 8.13, the Document/Passage Retrieval phase is shown to have two module objects and two output objects representing the document and LegalSpan passage retrieval components and their outputs. All the final results are generated by the module of LegalSpan passage retrieval. The document retrieval is not responsible for generating final results.

The module object features (M_2) We do not assign any features for those modules because they do not generate competing strategies.

The output object features (O_2) Document prior features can be estimated by counting the number of relevant documents from journals of the training set, e.g.,

$$P(D \in J) = \frac{Num\ of\ relevant\ documents\ of\ journal\ J + m}{Total\ num\ of\ relevant\ documents + |mV|} \tag{8.3}$$

where m is a smoothing parameter, D is a document which can be mapped to a journal by using the PMID of a document, V is the number of journals in TREC 2006 Genomics Track full-text collection.

The output → module dependency features ($O_1 - M_2$) We do not assign any feature over the dependency.

The module → output dependency features ($M_2 - O_2$) We assign document and passage retrieval scores to the returned results separately.

8.4.3 Important Sentence Extraction Phase Features

In Figure 8.13, the Important Sentence Extraction phase is shown to have three module objects and an output object representing the importance sentence extraction components and their outputs. The proximity and combination scorers are used to score those important sentences for ranking.

The module object features (M_3) We do not assign any features for those modules in the merged phase shown in Figure 8.14, because there are no competing modules.

The output object features (O_3) We give penalties to important sentences by using a list of words or particular sections detected from the irrelevant results. For example, some results are irrelevant but have high ranking scores such as "Abbreviations used in this paper: AD, Alzheimer's disease; APP, . . . " or "Keywords: Creutzfeldt-Jakob disease, Prnp, . . . ". The retrieval model prefers those results contained words such as "Abbreviation" or the section "Keyword" that do not provide correct answer to a question.

The input $\rightarrow$ module dependency features ($O_2 - M_3$) We do not assign any feature over the dependency.

The module $\rightarrow$ output dependency features ($M_3 - O_3$) We assign the scores calculated by the proximity scorer and combination scorer to those important sentences.

8.4.4 Phased Dependency Features I

We illustrate the feature model between the Keyterm Expansion phase and the Document/Passage Retrieval phase in Figure 8.15, with the green plate being the Keyterm Expansion phase and the orange plate being the Document/Passage Retrieval phase. We define edges between the two phases, meaning a set of dependence assumptions over the objects.

The input $\rightarrow$ output dependency features ($I - O_1$) We assign similarity scores between the original keyterms with its lexical variants and synonyms. First, the original keyterms are sent into search engine, and then we calculate the ratio of the newly expanded keyterm counts to the original keyterm counts from the top K returned passages.

The input $\rightarrow$ module dependency features ($I - M_2$) We do not assign any feature over the dependency.

The input $\rightarrow$ output dependency features ($I - O_2$) We re-estimate the document and LegalSpan passage scores $P(O_2|I)$ by using Indri's Retrieval Model with only the original input.

The module $\rightarrow$ module dependency features ($M_1 - M_2$) We do not assign any feature over the dependency.

The module $\rightarrow$ output dependency features ($M_1 - O_2$) We do not assign any feature over the dependency.

The output $\rightarrow$ output dependency features ($O_1 - O_2$) We re-estimate the document and LegalSpan passage scores $P(O_2|O_1)$ by using Indri's Retrieval Model with weighted keyterms O_1 (the original keyterms with their expansions). Different retrieval scores are calculated by the weights extracted and estimated from $M_1 - O_1$ and $I - O_1$.

8.4.5 Phased Dependency Features II

We illustrate the feature model between the Document/Passage Retrieval phase and the Important Sentence Extraction phase in Figure 8.17, with the green plate being the Document/Passage Retrieval phase and the orange plate being the Important Sentence Extraction phase.

The input → module dependency features ($I - M_3$) We do not assign any feature over the dependency.

The input → output dependency features ($I - O_3$) We estimate the important sentence relevance scores $P(O_3|I)$ by using Indri's Retrieval Model with only the original input.

The module → output dependency features ($M_1 - O_2$) We do not assign any feature over the dependency.

The module → module dependency features ($M_1 - M_3$) We do not assign any feature over the dependency.

The module → output dependency features ($M_1 - O_3$) We do not assign any feature over the dependency.

The output → module dependency features ($O_1 - M_3$) We do not assign any feature over the dependency.

The output → output dependency features ($O_1 - O_3$) We estimate the important sentence relevance scores $P(O_3|I)$ by using Indri's Retrieval Model with weighted keyterms O_1 (the original keyterms with their expansions).

8.4.6 Phased Dependency Feature Features III

We illustrate the feature model between the Keyterm Expansion phase and the Important Sentence Extraction phase in Figure 8.16, with the green plate being the Keyterm Expansion phase and the orange plate being the Important Sentence Extraction phase.

The module → module dependency features ($M_2 - M_3$) We do not assign any feature over the dependency.

The module → output dependency features ($M_2 - O_3$) We do not assign any feature over the dependency.

The output → output dependency features ($O_2 - O_3$) An important sentence is extracted from a LegalSpan passage object, so we don't know if an important sentence is contained in the retrieved document or not. We use binary feature to indicate if an important sentence is presented in the document set or not.

8.4.7 Applying Phased Ranking Model

Given our feature functions, we apply the rest of the phased ranking framework described in Section 5.2. A cascade model is used to increasingly collect ranking features of each phase. After that, a voting model is defined to merge and score similar important sentences within a LegalSpan passage.

The voting approach is essentially a result merging step to group similar results. Some commonly used String similarity metrics (e.g. Levenshtein distance) cannot model the relationships between those output nodes, simply because random output nodes cannot be merged together. In order to merge important sentences, it is natural to group those sentences within a LegalSpan passage. And therefore, our similarity is determined by the position of the passages in the same LegalSpan passage, based on the assumption that if an important sentence is relevant, then its neighbors should also be relevant. Based on the assumption, we define a similarity function with a max operator among those output nodes to fit and replace the voting equation 8.4,

$$Score(O_i|O_1, \ldots, O_n) = MAX\{r(O_1), \ldots, r(O_n)\} \tag{8.4}$$

where O_1 to O_n are sequential important sentences within a LegalSpan passage.

Given our cascade model and voting model, the final step is to set the parameter values for combining features defined in Section 8.4. We choose to apply grid search to train models by directly maximizing mean average precision due to several reasons. First, directly comparing with the baseline model requires integrating the same learning approach. Second, the training data available is a set of relevance judgments which is relatively small compared to the feature space. For this reason, it is highly unlikely that a maximum likelihood estimate may not be a good fit. Furthermore, that maximizing the likelihood will not maximize the underlying retrieval metric.

8.5 Experimental Results

In this section, we describe experiments using the proposed phased ranking model. Our aim is to analyze and compare the ranking effectiveness of the proposed model.

Our first baseline is the ranking approach of the BioQA System which is a state-of-the-art system on the task of TREC Genomics.All the parameter settings of the system are detailed in the paper [103]. The second baseline concept retrieval model has been introduced in Section 8.2. It reported the PsgMAP metric score 0.174 shortly after the official evaluation. The third baseline is the Generalized Watson Rank introduced in Section 6.4.3.

Table 8.1 gives mean average precision on both document level and passage level. The results given use the optimal parameter values to allow a fair comparison. The phased ranking model outperforms all four baselines on mean average precision of both document and passage level. It significantly improves the BioQA System with 1% passage level MAP score and 2.7% document level MAP score. We also note the proposed phased ranking model outperformed the best participating system (Concept Retrieval Model) and the published results (Best Published Results) in terms of both DocMAP and PsgMAP. Therefore, modeling phases and their dependencies can be done consistently and can result in significant improvements.

Table 8.2 summarizes the outcome of our phased ranking mode features and process. We show the sections for the corresponding feature. The process is initialized with our baseline system (the BioQA System) that replicates the state-of-the-art system (a combination of

Table 8.1: Experimental results and comparisons with published results and baselines on TREC Genomics 2006. Significance, using a one-sided sign test, is denoted with a ‡ at the $p < 0.005$ level over the BioQA baseline.

Approaches	**PsgMAP**	**DocMAP**
The BioQA System	0.169	0.520
Concept Retrieval Model [105]	0.174	0.525
GWB	0.174	0.528
Best Published Results [103]	0.177	0.534
PhRank	**0.179**‡	**0.547**‡

Table 8.2: Summary of phased ranking models.

Approaches	**PsgMAP**	**DocMAP**
The BioQA System	0.169	0.521
+ Important Sentence Similarity 8.4.7	0.173	0.521
+ Important Sentence Extraction Phase Features 8.4.3	0.176	0.529
+ Phased Dependency Features II 8.4.5	0.177	0.529
+ Phased Dependency Feature Features III 8.4.5	**0.179**	**0.547**

the Indri retrieval model scores [1] on both the document level and the passage level, the BM25-based proximity score). The algorithm first builds up the phased ranking model based on the scores from the baseline ranking approach, and then incrementally adds to the feature set the features extracted from each phase and their dependencies. The table shows that the improvements on PsgMAP can be gained from several feature groups: the merging step (Similarity) outperforms the baseline system in terms of PsgMAP; we also gain PsgMAP performance by introducing those new features extracted from Important Sentence Extraction phase and those dependencies between phases. This indicates that, even though the baseline model is strong, it is worth exploring phases of a system object graph to propagate information and combining several strategies for ranking.

Note that if some features do not appear in Table 8.2 it does not necessarily mean that they are useless. In some cases, such features are highly correlated with features previously selected, which already exploited their signal. For example, the Document/Passage Retrieval Phase features 8.4.2 (e.g., estimating the passage retrieval scores to the returned results) are correlated with those features in the baseline ranking approach, because both Indri retrieval model score and BM25-based proximity score are part of the baseline model. In other cases, some features detected from the phased ranking model are used indirectly for ranking. For example, those Keyterm Expansion phase features 8.4.1 are used as weights for retrieval models.

[1] http://www.lemurproject.org/indri/

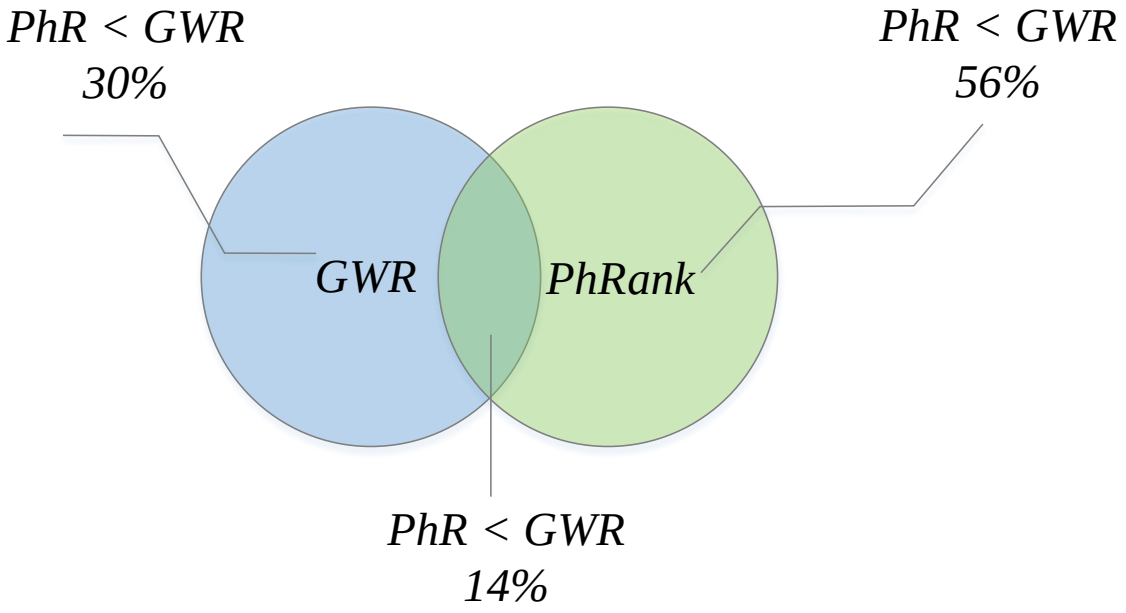

Figure 8.18: Phased ranking model versus Generalized Watson Rank analysis over document retrieval results.

8.6 Overlap Analysis

The absolute performance gaps between the phased ranking model and Generalized Watson Rank are measured as .5 %, 1.9 % in terms of PsgMAP and DocMAP respectively. Our pipelined system generated 1074 relevant documents compared with 1449 total number of gold standard documents, and 1257 relevant passages compared with 3451 total number of gold standard passages to the task of TREC Genomics 2006. Among those total 1074 relevant documents, there are 602 documents that phased ranking model did better than Generalized Watson Rank and 321 documents phased ranking model did worse. Among those total 1257 relevant passages, there are 911 passages that phased ranking model did better than Generalized Watson Rank and 284 documents phased ranking model did worse. We summarize the overlap analysis results of phased ranking model versus Generalized Watson Rank in Figure 8.18 and Figure 8.19.

To further understand why phased ranking model outperforms the baseline models, we use examples of phased ranking features to show what causes the gains and losses, by using the gold standard snippets provided for the TREC Genomics 2006 task, and compare with Generalized Watson Rank. We list the three main feature categories with examples and also show the detailed performance with salient missing features for each category in Table 8.3.

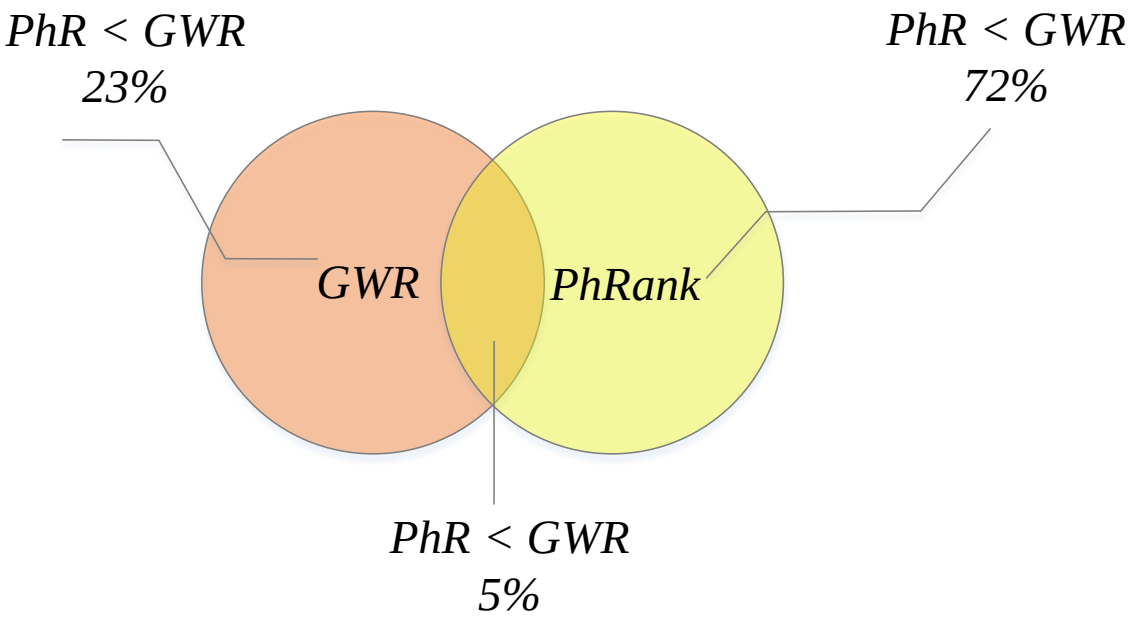

Figure 8.19: Phased ranking model versus Generalized Watson Rank analysis over passage retrieval results.

Table 8.3: Detailed phased ranking model with salient missing features for each main feature category versus Generalized Watson Rank (GWR) analysis.

Ranking Model	**Silent Missing Features**			
	Keyterm Expansion	Document/Passage Retrieval	Important Extraction	Sentence
		PsgMAP		
PhRank>GWR	882	909	626	
PhRank<GWR	298	286	446	
		DocMAP		
PhRank>GWR	602	601	421	
PhRank<GWR	309	324	464	

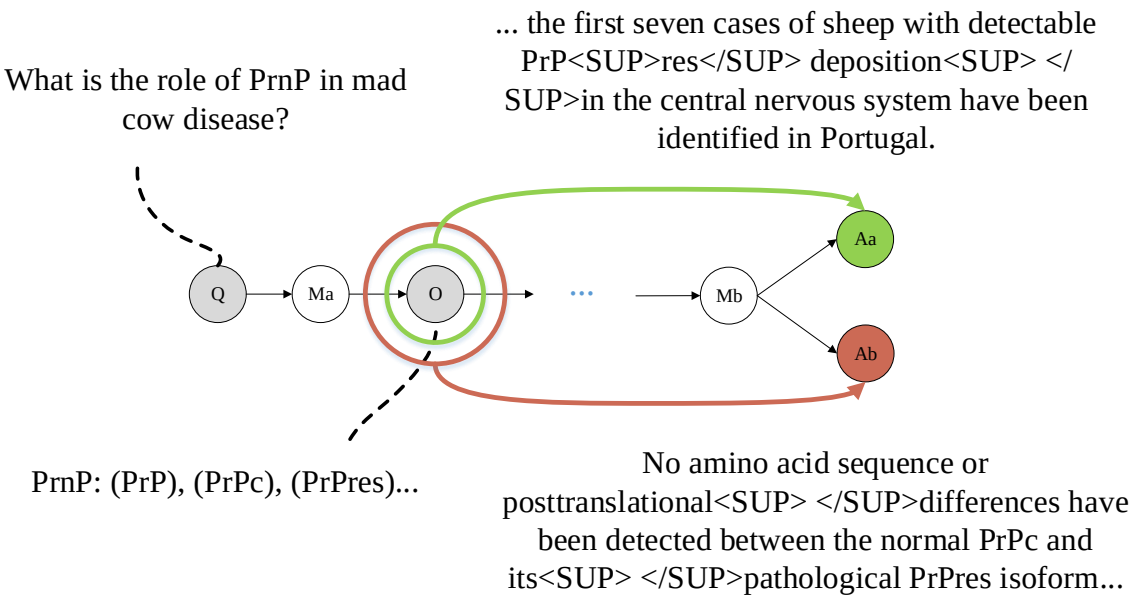

Figure 8.20: Example of Keyterm Expansion → Important Sentence Extraction Features (Gain/Loss).

Keyterm Expansion → Important Sentence Extraction (Gain/Loss)

An example of the Keyterm Expansion → Important Sentence Extraction feature (Gain/Loss) is shown in Figure 8.20. Consider the following question,

- What is the role of prnp in mad cow disease?

The Keyterm Expansion phase detected that "Prnp could have other names or variants,

- PrnP: (PrP), (PrPc), (PrPres)...

Although Generalized Watson Rank used retrieval model scores on document and passage level, the important sentences are not scored directly from the query and its expansions. One of the effective features is to build retrieval model on important sentence level. Consider the following two important sentences,

- ...the first seven cases of sheep with detectable PrP^{res} deposition <SUP></ SUP>in the central nervous system have been identified in Portugal.
- No amino acid sequence or posttranslationaldifferences have been detected between the normal PrPc and itspathological PrPres isoform ...

Using those new features can help improve both their ranks. The first one, which is relevant to the question, can help improve the performance. But the second one is

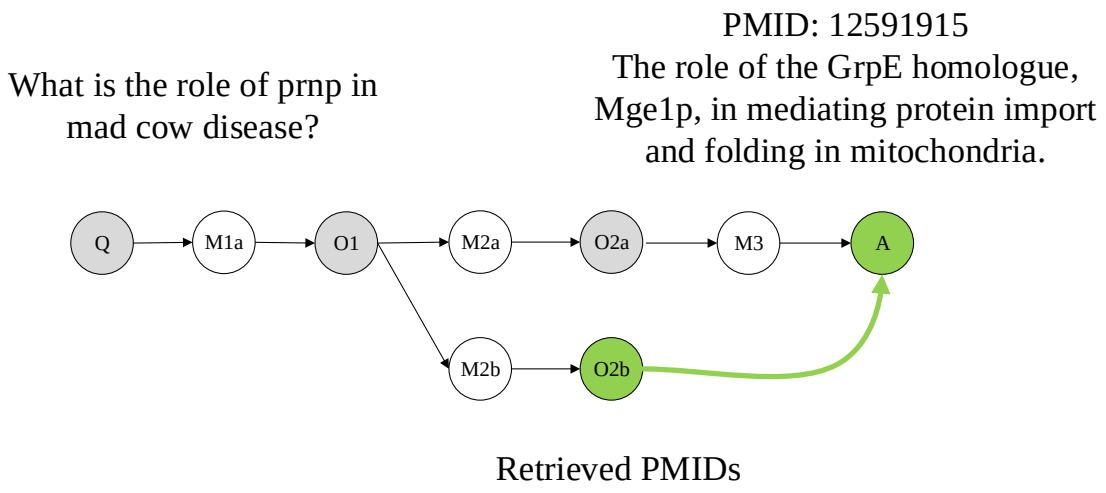

Figure 8.21: Example of Document/Passage Retrieval → Important Sentence Extraction Features (Gain).

irrelevant, and caused the loss of the ranking performance.

Document/Passage Retrieval → Important Sentence Extraction (Gain)

An example of the Document/Passage Retrieval → Important Sentence Extraction feature (Gain) is shown in Figure 8.21. Consider the following question,

- What is the role of prnp in mad cow disease?

The following important sentence is retrieved from the document (PMID: 12591915),

- PMID: 12591915 The role of the GrpE homologue, Mge1p, in mediating protein import and folding in mitochondria.

However, in the Document/Passage Retrieval phase, the document with PMID 12591915 should be viewed as irrelevant as it is not returned in the document retrieval module. The phased ranking model penalizes those important sentences whose PMIDs are not contained in the document sets.

Answer Merging Step (Gain)

An example benefited from answer merging scores is shown in Figure 8.22. Consider the following question,

- What is the role of prnp in mad cow disease?

The following two results are retrieved from the same LegalSpan passage, ranked as 3rd and 15th accordingly in the BioQA system.

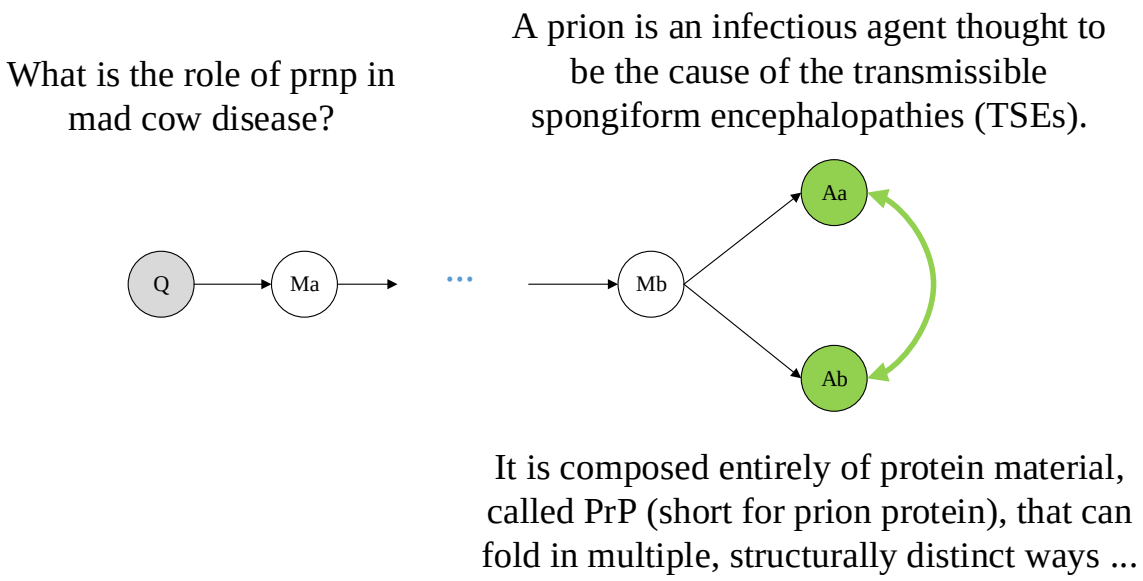

Figure 8.22: Example of the answer merging step (Gain).

- A prion is an infectious agent thought to be the cause of the transmissible spongiform encephalopathies (TSEs).
- It is composed entirely of protein material, called PrP (short for prion protein), that can fold in multiple, structurally distinct ways ...

Both the results are relevant to the question and the first result is followed by the second result in the original LegalSpan passage. After the answer merging step, those two results are merged together ranked as 3rd position.

Answer Merging Step (Loss)

An example caused by answer merging step (loss) is shown in Figure 8.23. Consider the following question,

- What is the role of prnp in mad cow disease?

The following two results are retrieved from the same LegalSpan passage, ranked as 38th and 124th accordingly in the BioQA system.

- They are characterized by the intracerebellar accumulation of PrP<SUP>Sc</ SUP>,¹ an abnormally folded ...
- The molecular mechanisms leading to the structural changes in PrP</A> are still unclear.

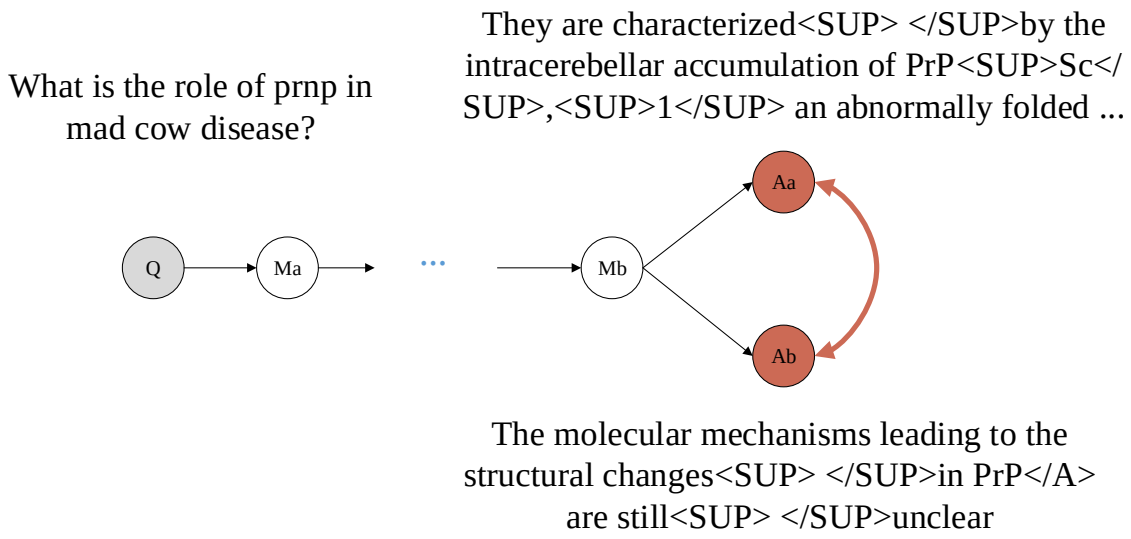

Figure 8.23: Example of the answer merging step (Loss).

The first result is relevant to the question but the second one is irrelevant. Because they are in the same LegalSpan passage, after the answer merging step, those two results are merged together ranked as 35th position.

Chapter 9

Contributions

This thesis studied the phased ranking model for information systems, motived by the observation that standard ranking approaches are not necessarily well-suited to the ranking tasks of information systems. There are two key differences between the proposed phased ranking model and the standard ranking approaches. In terms of novelty, our proposed approach achieved better evidence coverage than the standard ranking approaches by developing a general ranking model capable of propagating information from phases to the final phase ranking stage. In terms of generality, our proposed approach is not specialized to any particular task and can be applied to different applications by following the general solution procedure proposed in Chapter 4 and Chapter 5.. Our results show that the proposed phased ranking model significantly outperforms the standard ranking approaches (e.g., Generalized Watson Rank) and can improve the state-of-the-art system.

System Object Graph

Chapter 4 gives a formal definition of a system object graph which is a generalized and unifying representation for the information system. Typically, an information system consists of a number of processing steps arranged in series, and each component is described by its input(s) and output(s). All complex information systems that consist of multiple processing steps can be reduced to the system object graph representation described here. What is perhaps most distinctive about the system object graph is its naturalness in formulating models of complex phenomena in information systems. We present in Chapter 7 the process to define a system object graph for an information system. The process includes a phase identification step to classify phases and a phase merging step to simplify the graph.

A Phased Ranking Model

A phased ranking model allows each phase in a system to leverage information propagated from preceding phases to inform the ranking decision. This is accomplished by modeling the system object graph which represents all of the objects created during system execution. To perform ranking, our framework in Section 5 defines three main steps. It first applies a feature model and a cascade model to increasingly collect ranking features of each phase and their dependencies. Second, a voting model is used to merge and score similar outputs.

Finally, a ranking model is learned by jointly considering the cascade model and the voting model.

Case Study I: Factoid Question Answering

In Chapter 6, we applied the phased ranking model to the task of question answering for answer ranking. The phased answer ranking approach allows answer ranking in a QA system to leverage information propagated from preceding phases, specifically, question analysis, passage retrieval and candidate extraction, to inform the answer ranking decision. Experimental results showed that our proposed approach is statistically significantly better than or indistinguishable from comparable answer ranking models.

Case Study II: Biomedical Information Retrieval

In Chapter 8, we presented the application of the proposed phased ranking method by following the general solution procedure proposed in Chapter 4 and Chapter 5 to support a state-of-the-art biomedical information retrieval system. It was demonstrated that our proposed approach outperforms several strong baselines significantly. And therefore, our proposed method was shown to be effective for ranking, particularly for complex information systems.

General Principles for the Phased Ranking Model

There are several general application principles and steps for building a phased ranking model, briefly summaries as followings.

1. Define phases: listing all the processing unit and determine the phases and defining all the elements per phase.
2. Merge phases: simplifying the phases for the final system object graph.
3. Define features: apply phased ranking feature models to those phases.
4. Apply ranking model: first applying a cascade model to increasingly collect ranking features of each phase, and then applying a voting model to merge and score similar output nodes.

The phased ranking models for the two tasks, factoid question answering and biomedical information retrieval, are summarized and compared in Table 9.1.

Table 9.1: Comparisons of phased ranking models.

Configurations	**Biomedical Information Retrieval**	**Factoid Question Answering**
System Components	Keyterm Extraction, POS Tagging, Named Entity Extraction, Lexical Variant Extraction, Synonyms Detection, Keyterm Refining, Document Retrieval, LegalSpan Passage Retrieval, Important Sentence Extraction, Proximity-based Result Scoring, Final Passage Ranking	Question Analysis, Passage Retrieval, Answer Extraction and Answer Selection
Merged Phases	Keyterm Expansion, Document/Passage Retrieval, Important Sentence Extraction	Question Analysis, Passage Retrieval, Answer Extraction
Model Objects	One per phase	Multiple per phase
Similarity Functions	LegalSpan passage match	Text surface match
Learning Tools/Methods	Grid Search	Standard learning to rank

Chapter 10

Future Research Directions

This chapter discusses future research in three possible directions.

10.1 Future Research Direction I

In Section 6.4 and Section 8.5, we implemented the Generalized Watson Rank as a strong baseline for direct comparisons with our phased ranking model. The overlap analysis showed that the weight of a feature might vary according to the "ranking context". A "ranking context" is referred to the roughly estimated rank for a candidate object. A feature may have different weights according to the current rank position (e.g., top 5, or top 100). As well, a feature might be good at distinguishing relevant candidate from irrelevant ones, but it might fail to select the best ones. In the next step, our phased ranking model will be extended with Generalized Watson Rank that supports successive models contains more advanced machine learning techniques such as transfer learning, stacking, and successive refinements. We will also need to optimize ranking performance, including feature selection techniques for ranking, and ensemble learning techniques such as bagging and boosting.

10.2 Future Research Direction II

We have shown that our phased ranking model outperform strong baseline models by propagating information from object traces generated by an information system. Essentially, performance improvements come from reducing the "local optimal" problem that is common to information systems constructed by a series of system components that are developed independently.

Some past and growing recent interest shows that there are good reasons to jointly modeling across some specific NLP tasks [80, 100] that allows learning models to gradually reduce errors made earlier in the pipeline using later predictions. For example, parsing and named entity recognition was learned jointly [29] and improvements can be obtaining on both tasks. Entity and relation extraction was modeled together for improvements [60]. Generative models have been proposed to jointly predict co-reference, typing and linking

[21]. These approaches demonstrated the benefits of joint modeling on some individual subtasks, and therefore it is worthy to explore joint modeling for phased ranking.

However, there are challenges to adapt joint modeling approaches to our phased ranking model. First, joint modeling is largely task-specific that depends on task representations joint learning models. Second, previous work mainly focused on similar supervised joint models (e.g., features are similar and tasks are strongly related). Third, designing joint models which actually improve end-to-end systems has proven challenging [86].

To solve those problems, our research will 1) seek for uniform learning representations; 2) analyze joint model performances on various type of tasks (e.g., supervised or unsupervised); 3) report end-to-end system performance with joint modeling.

10.3 Future Research Direction III

The phased ranking model in this thesis assumes that a complex system is accessible so that it can be converted into a system object graph with phases. For example, the systems (e.g., Ephyra system and BioQA system) used in our applications employed standalone workflow representations, such as XML or YAML based process modeling languages. The architecture framework can be easily accessed and interpreted, because such representation language is independent of implementation details.

In practice, it is far from ideal– more often than not, the system architecture may lack standard format and documentation, system components can be difficult to access and analyze, and system components can be updated very quickly or are under development.

One possible extension can be mining through system logs. During the last decade, process mining has been applied in information systems [6]. The process mining does not assume the presence of a workflow management system, instead, it works by collecting workflow logs with event data and it. Previous research have been working on constructing a process based on these workflow logs from a set of real executions [90]. Defining a system object graph for phased ranking is similar to designing a workflow for an information system, i.e., a developer or designer has to construct a detailed model accurately describing the object trace through work. With process mining, it may be easier to construct a phased ranking model to achieve a good balance of resource cost and performance.

Another possible extension is to help developers understanding of software artifacts of the target system. Previous research explored topics on automatically generating natural language comments [82] and describing high-level actions of code fragments [83]. Text retrieval approaches [46] has been to extract words to label source code clusters, where in each cluster, these labels can be considered as (partial) summaries. Also, the summarization of large execution traces was suggested as a tool that can help programmers to understand the main behavioral aspects of a software system [35].

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005. 2.2

[2] Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D. Lawrence, David C. Gondek, and James Fan. Learning to rank for robust question answering. CIKM '12. 3.2.2, 3.2.3

[3] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. Finding high-quality content in social media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 183–194, 2008. 3.1.1

[4] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1(1):29–81, 1995. 3.1.1

[5] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X. 2.2

[6] Boualem Benatallah, Sherif Sakr, Daniela Grigori, Hamid Reza Motahari-Nezhad, Moshe Chai Barukh, Ahmed Gater, Seung Hwan Ryu, et al. Business process data analysis. In *Process Analytics*, pages 107–134. Springer, 2016. 10.3

[7] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. 2.3

[8] Adam Berger, Rich Caruana, David Cohn, Dayne Freitag, and Vibhu Mittal. Bridging the lexical chasm: Statistical approaches to answer-finding. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 192–199, 2000. 3.1.1

[9] Matthew W. Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. Structured retrieval for question answering. SIGIR '07. 1.1

[10] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutirrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. 2.2

[11] Johan Bos, James R. Curran, and Edoardo Guzzetti. The pronto qa system at trec-

2007: harvesting hyponyms, using nominalisation patterns, and computing answer cardinality. TREC 2007. 3.1.2

[12] John D. Burger, Lisa Ferro, Warren R. Greiff, John C. Henderson, Scott A. Mardis, Alexander A. Morgan, and Marc Light. Mitre's qanda at trec-11. In *TREC*, 2002. 3.2.3

[13] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, 2005. ISBN 1-59593-180-5. 2

[14] Christopher J. C. Burges, Robert Ragno, and Quoc V. Le. Learning to Rank with Nonsmooth Cost Functions. In Bernhard Schölkopf, John C. Platt, Thomas Hoffman, Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 193–200. MIT Press, 2006. 3

[15] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, 2007. 3

[16] Asli Celikyilmaz, Marcus Thint, and Zhiheng Huang. A graph-based semi-supervised learning for question-answering. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 719–727, 2009. 1.1

[17] Jennifer Chu-Carroll, Krzysztof Czuba, John Prager, and Abraham Ittycheriah. In question answering, two heads are better than one. NAACL '03. 2, 3.2.1, 3.2.3, 3.2.3

[18] W.B. Croft, D. Metzler, and T. Strohmann. *Search Engines: Information Retrieval in Practice*. Pearson, 2010. 2.3

[19] Hang Cui, Keya Li, Renxu Sun, Tat seng Chua, and Min yen Kan. National university of singapore at the trec 13 question answering main task. In *In Proceedings of the 13 th TREC*, 2005. 3.1.1

[20] Huizhong Duan, Yunbo Cao, Chin yew Lin, and Yong Yu. Searching questions by identifying question topic and question focus. In *In Proceedings of 46th Annual Meeting of the Association for Computational Linguistics: Human Language Tchnologies (ACL:HLT*, 2008. 1.1

[21] Greg Durrett and Dan Klein. A joint model for entity analysis: Coreference, typing, and linking. *TACL*, 2:477–490, 2014. 10.2

[22] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering, 2003. 3.2.2

[23] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 16–23, 2003. 3.1.1

[24] Abdessamad Echihabi, Ulf Hermjakob, Eduard Hovy, Daniel Marcu, Eric Melz, and

Deepak Ravichandran. How to select an answer string? In *Advances in Textual Question Answering*. 2004. 3.2.3

[25] D Ferrucci, E Nyberg, J Allan, K Barker, E Brown, J Chu-Carroll, A Ciccolo, et al. Towards the open advancement of question answer systems. Technical report, IBM Technical Report RC24789, Yorktown Heights, NY, 2009. 1.1

[26] D. A. Ferrucci. Introduction to "this is watson". *IBM J. Res. Dev.*, 56(3):235–249, May 2012. 1.1, 3.1.1, 3.1.2, 6.1.1

[27] David Ferrucci and Adam Lally. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10 (3-4):327–348, 2004. ISSN 1351-3249. 1.1

[28] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010. 1.2, 1.3, 1, 2, 2.2, 3, 3.1, 3.1.3, 3.1.3, 3.2.3

[29] Jenny Rose Finkel and Christopher D. Manning. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 326–334, 2009. ISBN 978-1-932432-41-1. 10.2

[30] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. ACL '05. 7f

[31] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003. ISSN 1532-4435. 2

[32] D. C. Gondek, A Lally, A Kalyanpur, J. W. Murdock, P. A Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty. A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3.4):14:1–14:12, May 2012. 1.2, 1.3, 1.1, 3.2.3, 3.2.3, 6.4.3, 7.1.2

[33] B.F. Green, A.K. Wolf, C. Chomsky, and K. Laugherty. Baseball: An automatic question answerer. In *Proceedings Western Joint IRE-AIEE-ACM Computing Conference*, volume 19, 1961. 3.1.1

[34] M. A. Greenwood. Using pertainyms to improve passage retrieval for questions requesting information about a location. SIGIR 2004. 3.1.3

[35] Abdelwahab Hamou-Lhadj and Timothy Lethbridge. Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system. In *ICPC*, pages 181–190. IEEE Computer Society, 2006. 10.3

[36] Sanda Harabagiu and Andrew Hickl. Methods for using textual entailment in open-domain question answering. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for*

Computational Linguistics, ACL-44, pages 905–912, 2006. 1.1, 3.1.1

[37] Ulf Hermjakob, Eduard Hovy, and Chin-Yew Lin. Automated question answering in webclopedia: A demonstration. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, pages 370–371, 2002. 3.1.1

[38] William R. Hersh, Aaron M. Cohen, Phoebe M. Roberts, and Hari Krishna Rekapalli. TREC 2006 genomics track overview. In *Proceedings of the Fifteenth Text REtrieval Conference, TREC 2006, Gaithersburg, Maryland, USA, November 14-17, 2006*, 2006. 8.1, 8.1.1, 8.2

[39] Andrew Hickl, Kirk Roberts, Bryan Rink, Jeremy Bensley, Tobias Jungen, Ying Shi, and John Williams. Question answering with lcc's chaucer-2 at trec 2007. TREC 2007. 2, 3.1.2, 6.4.2

[40] Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 84–90, 2005. 1.1

[41] Thorsten Joachims. Advances in kernel methods. chapter Making large-scale support vector machine learning practical. MIT Press, 1999. 6.4.2

[42] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, 2006. ISBN 1-59593-339-5. 2

[43] Boris Katz and Jimmy Lin. Selectively using relations to improve precision in question answering. In *In Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering*, pages 43–50, 2003. 3.1.1

[44] Jeongwoo Ko, Eric Nyberg, and Luo Si. A probabilistic graphical model for joint answer ranking in question answering. SIGIR '07, . 3.2.2, 5.2.3

[45] Jeongwoo Ko, Luo Si, and Eric Nyberg. A probabilistic framework for answer selection in question answering. NAACL '07, . 1.1, 1.1, 2.4, 3.2.2, 5.1, 5.3.3, 6.3.3, 6.3.3, 6.4.1

[46] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. Semantic clustering: Identifying topics in source code. *Inf. Softw. Technol.*, 49(3):230–243, March 2007. ISSN 0950-5849. 10.3

[47] Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan & Claypool Publishers, 2011. 2.3

[48] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003. 2.2

[49] Rui Liu and Eric Nyberg. A phased ranking model for question answering. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 79–88,

2013. 1.1, 2.4

[50] Tie-Yan Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3 (3):225–331, March 2009. 2.3

[51] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011. 2.2

[52] Bernardo Magnini, Matteo Negri, Roberto Prevete, and Hristo Tanev. Is it the right answer?: exploiting web redundancy for answer validation. ACL '02. 5.2, 5.3.3

[53] Bernardo Magnini, Matteo Negri, Roberto Prevete, and Hristo Tanev. Is it the right answer?: Exploiting web redundancy for answer validation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 425–432, 2002. 1.1, 3.2.1

[54] A. Fee Majid Razmara and L. Kosseim. Concordia University at the TREC-2007 QA Track. TREC 2007. 2.4

[55] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. 2.2

[56] Abraham Ittycheriah Martin, Martin Franz, and Salim Roukos. Ibm's statistical question answering system-trec-10. TREC 2001. 3.2.2

[57] Donald Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274, June 2007. ISSN 1386-4564. 3

[58] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. SIGIR '05. 6.4.6, 6c

[59] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479, 2005. 6.2.2

[60] Makoto Miwa and Yutaka Sasaki. Modeling joint entity and relation extraction with table representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, October 2014. 10.2

[61] Dan Moldovan, Christine Clark, and Mitchell Bowden. Lymba's PowerAnswer 4 in TREC 2007. TREC 2007. 6.4.1

[62] Dan Moldovan, Christine Clark, Sanda Harabagiu, and Steve Maiorano. Cogex: A logic prover for question answering. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 87–93, 2003. 3.1.1

[63] Dan I. Moldovan, Christine Clark, Sanda M. Harabagiu, and Daniel Hodges. Cogex: A semantically and contextually enriched logic prover for question answering. *J. Applied Logic*, 5(1), 2007. 2.4

[64] J. W. Murdock, A Kalyanpur, C. Welty, J. Fan, D.A Ferrucci, D. C. Gondek, L. Zhang, and H. Kanayama. Typing candidate answers using type coercion. *IBM Journal of Research and Development*, 56(3.4):7:1–7:13, May 2012. 6.2.1

[65] Vanessa Murdock and W. Bruce Croft. A translation model for sentence retrieval. In *in Proceedings of the Conference on Human Language Technologies and Empirical Methods in Natural Language Processing (HLT/EMNLP*, pages 684–691, 2005. 3.1.1

[66] E. Nyberg, T. Mitamura, J. Callan, J. Carbonell, R. Frederking, K. Collins-thompson, L. Hiyakumoto, Y. Huang, C. Huttenhower, S. Judy, J. Ko, L. V. Lita, V. Pedro, D. Svoboda, and B. Van Durme. The javelin question-answering system at trec 2002. In *Proceedings of TREC 12*, 2003. 3.1.1

[67] Eric Nyberg, Robert E. Frederking, Teruko Mitamura, Matthew W. Bilotti, Kerry Hannan, Laurie Hiyakumoto, Jeongwoo Ko, Frank Lin, Lucian Vlad Lita, Vasco Pedro, and Andrew Hazen Schlaikjer. Javelin i and ii systems at trec 2005. TREC 2005. 2.1, 1, 2, 3.1.2, 3.1.3, 3.2.3, 6.2

[68] Eric Nyberg, Teruko Mitamura, James P. Callan, Jaime G. Carbonell, Robert E. Frederking, Kevyn Collins-Thompson, Laurie Hiyakumoto, Yifen Huang, Curtis Huttenhower, Scott Judy, Jeongwoo Ko, Anna Kupsc, Lucian Vlad Lita, Vasco Pedro, David Svoboda, and Benjamin Van Durme. The JAVELIN question-answering system at TREC 2003: A multi-strategh approach with dynamic planning. In *Proceedings of The Twelfth Text REtrieval Conference, TREC 2003, Gaithersburg, Maryland, USA, November 18-21, 2003*, 2003. 3.1.2

[69] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, pages 149–157, 2003. 3.1.1

[70] Vasin Punyakanok, Dan Roth, and Wen tau Yih. Mapping dependencies trees: An application to question answering. In *In Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics, Fort*, 2004. 3.1.1

[71] Hari Krishna Rekapalli, Aaron M. Cohen, and William R. Hersh. A comparative analysis of retrieval features used in the TREC 2006 genomics track passage retrieval task. In *AMIA 2007, American Medical Informatics Association Annual Symposium, Chicago, IL, USA, November 10-14, 2007*, 2007. 8.2

[72] Stefan Riezler, Er Vasserman, Ioannis Tsochantaridis, Vibhu Mittal, and Yi Liu. Statistical machine translation for query expansion in answer retrieval. In *In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL07*, 2007. 1.1, 3.1.1

[73] German Rigau, Jordi Atserias, and Eneko Agirre. Combining unsupervised lexical knowledge methods for word sense disambiguation. ACL '98, pages 48–55, 1997. 3.1.2

[74] Neil J. Risch. Searching for genetic determinants in the new millennium. *Nature*, 405(6788):847–856, Jun 2000. 2.2

[75] Kenji Sagae and Alon Lavie. Parser combination by reparsing. NAACL-Short '06, pages 129–132, 2006. 3.1.2

[76] Nico Schlaefer, Jennifer Chu-Carroll, Eric Nyberg, James Fan, Wlodek Zadrozny,

and David Ferrucci. Statistical source expansion for question answering. CIKM '11, . 1.1, 6.1.2

[77] Nico Schlaefer, Jeongwoo Ko, Justin Betteridge, Guido Sautter, Manas Pathak, and Eric Nyberg. Semantic extensions of the ephyra qa system for trec 2007. TREC 2007, . 1.1, 1, 2, 3.1.1, 3.1.2, 3.2.3, 6.3.1, 4c, 4d

[78] Dan Shen, Michael Wiegand, Andreas Merkel, Stefan Kazalski, Sabine Hunsicker, Jochen L. Leidner, and Dietrich Klakow. The alyssa system at trec qa 2007: Do we need blog06? TREC 2007. 3.1.2

[79] Luo Si and Jamie Callan. Modeling search engine effectiveness for federated search. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 83–90, 2005. 3.1.2

[80] Sameer Singh, Sebastian Riedel, Brian Martin, Jiaping Zheng, and Andrew McCallum. Joint inference of entities, relations, and coreference. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, pages 1–6, 2013. 10.2

[81] Radu Soricut and Eric Brill. Automatic question answering using the web: Beyond the factoid. *Inf. Retr.*, 9(2):191–206, March 2006. 3.1.1

[82] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 43–52, 2010. 10.3

[83] Giriprasad Sridhara, Lori Pollock, and K. Vijay-Shanker. Automatically detecting and describing high level actions within methods. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 101–110, 2011. 10.3

[84] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. Indri: a language-model based search engine for complex queries. Technical report, in Proceedings of the International Conference on Intelligent Analysis, 2005. 6.2.2, 8.3.1

[85] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers on large online qa collections. In *In Proceedings of the 46th Annual Meeting for the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT*, pages 719–727, 2008. 1.1

[86] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08, pages 159–177, 2008. 10.2

[87] Luis Tari, Graciela Gonzalez, Robert Leaman, Shawn Nikkila, Ryan Wendt, and Chitta Baral. Asu at trec 2006 genomics track. In *TREC*, 2006. 8.2

[88] Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *In Proceedings of the 26th Annual International ACM SIGIR Conference on Research*

and Development in Information Retrieval (SIGIR, pages 41–47. ACM Press, 2003. 3.1.2

[89] Ming-Feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: A ranking method with fidelity loss. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 383–390, 2007. ISBN 978-1-59593-597-7. 2

[90] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, September 2004. 10.3

[91] Antti veikko I. Rosti, Necip Fazil Ayan, Bing Xiang, Spyros Matsoukas, Richard Schwartz, and Bonnie J. Dorr. Combining outputs from multiple machine translation systems. In *In Proceedings of the North American Chapter of the Association for Computational Linguistics Human Language Technologies*, pages 228–235, 2007. 3.1.2

[92] Ellen M. Voorhees. Overview of the trec 2002 question answering track. In *TREC*, pages 115–123, 1999. 6.1, 6.1.1

[93] Baoxun Wang, Xiaolong Wang, Chengjie Sun, Bingquan Liu, and Lin Sun. Modeling semantic relevance for question-answer pairs in web social communities. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1230–1238, 2010. 1.1

[94] Xin-Jing Wang, Xudong Tu, Dan Feng, and Lei Zhang. Ranking community answers by modeling question-answer relationships via analogical reasoning. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 179–186, 2009. 1.1

[95] W. A. Woods. Lunar rocks in natural english: Explorations in natural language question answering. In A. Zampolli, editor, *Linguistic Structures Processing*, pages 521–569. North-Holland, Amsterdam, 1977. 3.1.1

[96] Min Wu, Chen Song, Yu Zhan, and Tomek Strzalkowski. Ualbany's ilqua at trec2007. TREC 2007. 3.1.2

[97] Jinxi Xu, Ana Licuanan, Jonathan May, Scott Miller, and Ralph M. Weischedel. Trec 2002 qa at bbn: Answer selection and confidence estimation. In *TREC*, 2002. 3.1.1

[98] Jun Xu and Hang Li. Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 391–398, 2007. 3

[99] Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. Retrieval models for question and answer archives. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 475–482, 2008. ISBN 978-1-60558-164-4. 3.1.1

[100] Bishan Yang and Claire Cardie. Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational*

Linguistics (Volume 1: Long Papers), Sofia, Bulgaria, August 2013. 10.2

[101] Hui Yang and Tat-Seng Chua. The integration of lexical knowledge and external resources for question answering. TREC 2002. 2.4

[102] Zi Yang, Elmer Garduno, Yan Fang, Avner Maiberg, Collin McCormack, and Eric Nyberg. Building optimal information systems automatically: configuration space exploration for biomedical information systems. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, CIKM '13, pages 1421–1430, 2013. 1.1

[103] Zi Yang, Elmer Garduno, Yan Fang, Avner Maiberg, Collin McCormack, and Eric Nyberg. Building optimal information systems automatically: configuration space exploration for biomedical information systems. In *CIKM*. ACM, 2013. 3.1.3, 8, 8.2, 8.3, 8.3.1, 8.5, 8.1

[104] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 271–278, 2007. 3

[105] Wei Zhou, Clement T. Yu, Vetle I. Torvik, and Neil R. Smalheiser. A concept-based framework for passage retrieval at genomics. In *TREC*, 2006. 8.2, 8.2, 8.1