Building Optimal Information Systems Automatically: Configuration Space Exploration for Biomedical Information Systems

Zi Yang
Carnegie Mellon University
ziy@cs.cmu.edu

Elmer Garduno
Sinnia
elmerg@sinnia.com

Yan Fang
Oracle Corporation
yan.fang@oracle.com

Avner Maiberg
Carnegie Mellon University
amaiberg@cs.cmu.edu

Collin McCormack
The Boeing Company
collin.w.mccormack@boeing.com

Eric Nyberg
Carnegie Mellon University
ehn@cs.cmu.edu

ABSTRACT

Software frameworks which support integration and scaling of text analysis algorithms make it possible to build complex, high performance information systems for information extraction, information retrieval, and question answering; IBM's Watson is a prominent example. As the complexity and scaling of information systems become ever greater, it is much more challenging to effectively and efficiently determine which toolkits, algorithms, knowledge bases or other resources should be integrated into an information system in order to achieve a desired or optimal level of performance on a given task. This paper presents a formal representation of the space of possible system configurations, given a set of information processing components and their parameters (*configuration space*) and discusses algorithmic approaches to determine the optimal configuration within a given configuration space (*configuration space exploration* or *CSE*). We introduce the *CSE framework*, an extension to the UIMA framework which provides a general distributed solution for building and exploring configuration spaces for information systems. The CSE framework was used to implement biomedical information systems in case studies involving over a trillion different configuration combinations of components and parameter values operating on question answering tasks from the TREC Genomics. The framework automatically and efficiently evaluated different system configurations, and identified configurations that achieved better results than prior published results.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.4 [**Information Storage and Retrieval**]: Systems and Software; J.3 [**Life and Medical Sciences**]: Medical information systems

Keywords

configuration space exploration; biomedical question answering

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505692.

1. INTRODUCTION

Software frameworks which support integration and scaling of text analysis algorithms make it possible to build complex, high-performance information systems for information extraction, information retrieval, and question answering; IBM's Watson is a prominent example. As the complexity and scaling of information systems become ever greater, it is much more challenging to effectively and efficiently determine which toolkits, algorithms, knowledge bases (KBs) or other resources should be integrated into an information system in order to achieve a desired or optimal level of performance on a given task.

The information retrieval community has considered the formal evaluation of different systems with standard benchmarks and metrics for decades, especially in various evaluation conferences, e.g. TREC[1], NTCIR[2], CLEF[3], etc, which provide researchers opportunities to exchanges ideas and experience on how to select components and evaluate systems. In some evaluations, both features extracted by participants and their end-to-end system outputs are shared [12, 13, 25]; in other cases intermediate subtask outputs are shared for comparative evaluation of one subsystem [21, 22]. To facilitate information system evaluation based on commonly-shared benchmarks, researchers have recently paid considerable attention to open and public evaluation infrastructures [35, 2, 10]. However, due to the lack of a standard task framework, it is difficult to reproduce experimental results and/or evaluate different combinations of system sub-modules across participants.

Complex information systems are usually constructed as a solution to a specific information processing task in a particular domain, and often lack a clear separation of framework, component configuration, and component logic. This in turn makes it more difficult to adapt and tune existing components for new tasks without editing the original component source code, an expense that precludes efficient exploration of a large set of possible configurations. To fully leverage existing components, it must be possible to further automatically explore the space of possible system configurations to determine the optimal combination of tools and parameter settings for a new task. We refer to this problem as *configuration space exploration*, and address three main topics in this paper:

- How can we formally define a *configuration space* to capture the various ways of configuring resources, components, and

[1] http://trec.nist.gov/
[2] http://ntcir.nii.ac.jp/
[3] http://www.clef-initiative.eu/

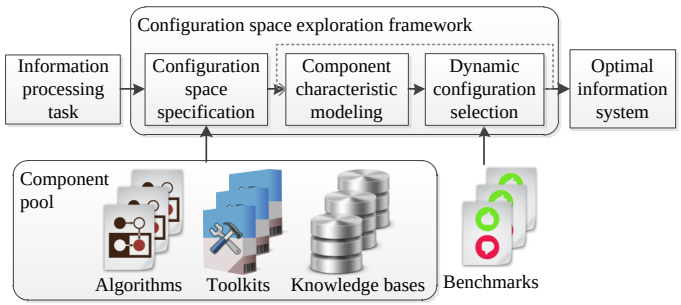

Figure 1: Overview of configuration space exploration framework architecture

parameter values to produce a working solution? Can we give a formal characterization of the problem of finding an optimal configuration from a given configuration space?

- Is it possible to develop task-independent open source software that can easily create a standard task framework and incorporate existing tools and efficiently explore a configuration space using distributed computing?

- Given a real-world information processing task, e.g., biomedical question answering, and a set of available resources, algorithms and toolkits, is it possible to write a descriptor for the configuration space, and then find an optimal configuration in that space using the CSE framework?

To explore these topics, we present a formal representation of the space of possible information system configurations, given a set of information processing components and their parameters (the *configuration space*), and also discuss algorithmic approaches which can be used to determine the optimal configuration within a given configuration space (*configuration space exploration* or *CSE*). Then, we introduce an open-source platform *CSE framework*[4], an extension to the UIMA framework[5], which provides a general distributed solution for building and exploring configuration spaces for information systems. An overview of the architecture is shown in Figure 1. To the best of our knowledge, our model is the first to formalize and empirically measure solutions for configuration space exploration in the rapid domain adaptation of information systems. We propose a novel formulation and solution to the associated constraint optimization problem, which comprises two sub-problems: component characteristics estimation and stochastic scheduling. We also describe open-source software which is the first embodiment of the CSE concept.

The CSE framework was also used to implement a biomedical information system in a case study. For example, responding to a question in the TREC Genomics Track [11] requires a system to answer biomedical questions, by retrieving relevant passages from a collection of journal articles. A very large number of biomedical knowledge bases, algorithms and toolkits for information retrieval, natural language processing and machine learning were cited by the task participants in their notebook papers and subsequent publications. The CSE framework was used to define and explore a configuration space with over a trillion different configurations of components and parameter values. CSE was able to find an optimal configuration of components for TREC Genomics with a passage MAP score better than prior published results for the task. In addition, the CSE framework makes it possible to automatically discover various features of individual components (e.g., impact, preference, speed, etc.) to support adaptive exploration policies during experimentation.

[4] `http://oaqa.github.io`, the portal page for all related projects, including the general CSE framework, base QA framework, biomedical QA system, resource wrappers, user interface, etc. It also features a tutorial in alignment with the examples in the official UIMA tutorial.

[5] `http://uima.apache.org/`

2. RELATED WORK

Retrieval evaluation frameworks. Information retrieval communities have held a series of competitions and conferences to create standard benchmarks and metrics to evaluate retrieval systems, and organizers have attempted to tabulate and analyze the most important features of high-performing systems, based on the information provided by the system reports, e.g. [25]. Due to lack of a standard task framework, organizers rarely reproduce experiments to try different combinations of modules across participants.

Recently, several shared tasks employ standardized data flows for successive subtasks (e.g. IR4QA and CCLQA tasks in the NTCIR Advanced Cross-Lingual Information Access task [21, 22]), and the results of the upstream tasks were collected from each system and shared for processing by all participants in the downstream tasks. This approach was able to discover configurations that combined different task systems to achieve a better result than the originally reported systems, which lends further motivation to configuration space exploration. However, the evaluation achieved interoperability through task-specific protocols, and did not provide a general framework for describing and evaluating configuration spaces that included variation on components and parameter settings.

To facilitate system evaluation based on commonly-shared benchmarks, researchers have recently paid more attention to open and public evaluation infrastructures for information retrieval evaluation and information extraction [35]. More recently, the Workshop on Data infrastructurEs for Supporting Information Retrieval Evaluation [2] was held to focus on organizing benchmark collections and tools into coherent and integrated infrastructures to ensure reproducible and comparable experiments; various retrieval evaluation frameworks were presented [10]. However, systems are usually specific to a single retrieval task, and lack a clear separation of framework, component configuration, and component logic, which makes it hard to fully leverage existing tools through automatic configuration space exploration.

Biomedical information retrieval and question answering systems. We focus here on summarizing systems and approaches that have participated in TREC Genomics QA competition or have leveraged the task benchmarks outside the formal competition.

A typical biomedical question answering pipeline consists of three major components: keyterm extraction and query expansion, document (or paragraph) retrieval, and passage extraction [25, 30, 27]. Synonyms, acronyms, and lexical variants are processed in the first phase; a retrieval model is applied in the second phase; and the similarity between the query and each retrieved passage is considered in the last phase. Researchers have tried to apply existing general retrieval systems [4, 26], relevance feedback to the traditional retrieval model [38, 28, 18], or a fusion or shrinkage of retrieval models [7]. Moreover, linguistic knowledge is also incorporated for the task, e.g., POS tags [19], and SVO structure [30].

Recently, researchers in biomedical information retrieval and question answering continue to leverage the TREC Genomics data set for evaluation [37, 14, 32, 20, 23, 17, 5]. Given the absence of an easy-to-use framework for building baseline systems for new tasks and exploring large parts of the configuration space, iterative research typically focuses on perturbations to a single mod-

ule while keeping modules and parameters elsewhere in the system frozen. Stokes et al [29] follow the same assumption to combine components from task participants to find optimal configurations for a biomedical IR task, using an approach specific to the task and difficult to generalize. In this paper, we use a biomedical information system as a case to theoretically and empirically study the configuration space exploration problem.

3. PROBLEM DEFINITION

In this section, we formally define the concepts used in this paper, and provide a formal definition of the CSE problem.

Most information systems consist of a number of processing units or components arranged in series, and each component is described by its input(s) and output(s). For example, a typical question answering system has four main component types: question analyzer, document retriever, passage extractor, answer generator [31]. A typical ontology-based information extraction pipeline will integrate several preprocessors and aggregators [35]. These processing steps can be abstracted as phases in a pipeline.

Definition 1 (Phase, component, parameter, configuration)**.** *The processing unit as the t-th step in a process can be conceptualized as a **phase** t. A **component** f_t^c in phase t is an instantiated processing unit, which is associated with a set of **parameters**, denoted by $\{\omega_t^{c,p}\}_p$, which constitute a component **configuration** $\boldsymbol{\omega}_t^c$.*

According to the definition of component and configuration, an output produced by a processing unit can be formally described as $y = f_t^c(x|\boldsymbol{\omega}_t^c)$, where x is an input (typed object) from the previous step, and y is the output. Note that parameters are not restricted to numeric values, but can hold a reference to any typed object.

As an example, consider Question Named Entity Recognition, a phase t in a question answering system where the input x_{t-1} is a question sentence, and the output x_t is a list of named entities. Component f_t^1 could be a rule-based named entity extractor, f_t^2 could be a CRF-based named entity extractor, and f_t^3 could be a named entity extractor based on knowledge base lookup. Configuration parameter value $\boldsymbol{\omega}_t^1$ could be the set of rules to use, $\boldsymbol{\omega}_t^2$ could be a weight trained for the CRF model, and $\boldsymbol{\omega}_t^3$ could refer to the knowledge base to be used by the component.

We introduce notations $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ to capture two important characteristics of the configured component $f_t^c|\boldsymbol{\omega}_t^c$: the *cost* of resource required to execute the component on input x and the *benefit* of executing the configured component to performance improvement. Resources used by a component include execution time, storage space, network bandwidth, etc., which can be measured by CPU time, allocated memory size, and data transfers respectively; a resource utilization measure can also be a more specific function of component characteristics (e.g., the cost to execute a configured component on Amazon Web Services[6] is a function of execution time and hardware capacity utilized). The benefit of a single component is relatively difficult to measure without being integrated into a system with other components, where we can leverage commonly-used evaluation metrics for information systems, e.g. F-1 and MAP, etc. We simply assume a configured component shares the benefit with the component that follows, if no direct measurement is available; the true cost or benefit of a configured component is estimated based on the cumulative effect over trillions of pipeline combinations. The scope of benefit is not necessarily limited to the effectiveness of the component, but may also incorporate its efficiency, i.e. execution time, storage space, etc.

[6] http://aws.amazon.com/

A typical information processing task can be described as n processing phases arranged sequentially. A unique series of instantiated components is grouped into a single trace.

Definition 2 (Trace and configuration space)**.** *A **trace** $\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}$ is an execution path that involves a single configured component for each phase, which is formally defined as $(f_1^{c_1}|\boldsymbol{\omega}_1^{c_1}, f_2^{c_2}|\boldsymbol{\omega}_n^{c_2}, \ldots, f_n^{c_n}|\boldsymbol{\omega}_n^{c_n})$. The set of all components with all configurations comprise the **configuration space** $\mathcal{F}|\boldsymbol{\Omega} = \{\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}\}_c$, and a subset $F|\Omega \subseteq \mathcal{F}|\boldsymbol{\Omega}$ is referred to as a **configuration subspace**.*

For example, question analyzers, document retrievers, passage extractors, and answer generators comprise the configuration space for a typical four-phase question answering task. One single execution path would be a unique combination of components (e.g. "Query tokenized by white-space string splitter, document retrieved from Indri repository index with default parameters, sentence extracted based on LingPipe sentence segmenter and Vector Space Model similarity calculator") or a trace in the configuration space.

We extend the concepts of *cost* and *benefit* for a configured component to a trace and a configuration subspace: the cost to execute a trace is the sum of costs to execute each configured component, and the performance of a trace corresponds to the final output from last execution. They can be formally defined as follows:

$$c(\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}, x) = \sum_{t=1}^{n} c(f_t^{c_t}|\boldsymbol{\omega}_t^{c_t}, x(c_1, \ldots, c_{t-1})) \quad (1)$$

$$b(\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}, x) = b(f_n^{c_n}|\boldsymbol{\omega}_n^{c_n}, x(c_1, \ldots, c_{n-1})) \quad (2)$$

where $x(c_1, \ldots, c_{t-1})$ represents the output from a series of executions $(f_1^{c_1}|\boldsymbol{\omega}_1^{c_1}, \ldots, f_{t-1}^{c_{t-1}}|\boldsymbol{\omega}_{t-1}^{c_{t-1}})$, which accumulates information gathered from all previous phases, due to the recursive definition that $x(c_1, \ldots, c_u) = f_u^{c_u}(x(c_1, \ldots, c_{u-1})|\boldsymbol{\omega}_u^{c_u})$ for $u = 1, \ldots, n$. The cost of the entire configuration subspace is defined as the sum of unique executions of configured components on all outputs from previous phases. The benefit of the configuration space is defined as the benefit of the best-performing trace.

$$c(F|\Omega, x) = \sum_{t=1}^{n} \sum_{c_1=1}^{m_1} \cdots \sum_{c_t=1}^{m_t} c(f_t^{c_t}|\boldsymbol{\omega}_t^{c_t}, x(c_1, \ldots, c_{t-1})) \quad (3)$$

$$b(F|\Omega, x) = \max_{\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c} \in F|\Omega} b(\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}, x) \quad (4)$$

We see that $c(F|\Omega, x) \neq \sum_{\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c} \in F|\Omega} c(\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}, x)$, In fact, if a trace $\mathbf{f^c}|\boldsymbol{\omega}^\mathbf{c}$ has been executed, and another trace $\mathbf{f^{c'}}|\boldsymbol{\omega}^{\mathbf{c'}}$ has the same prefix, i.e., $f_1^{c_1}|\boldsymbol{\omega}_1^{c_1}, \ldots, f_t^{c_t}|\boldsymbol{\omega}_t^{c_t} = f_1^{c'_1}|\boldsymbol{\omega}_1^{c'_1}, \ldots, f_t^{c'_t}|\boldsymbol{\omega}_t^{c'_t}$, then we do not need to repeat the executions for the configured components along the prefix, which is one of the key ideas for the problem solution and implementation. We then formally define the problem of configuration space exploration as follows.

Definition 3 (Configuration space exploration)**.** *For a particular information processing task, defined by m_t configured components for each of n phases: $f_t^1|\boldsymbol{\omega}_t^1, f_t^2|\boldsymbol{\omega}_t^2, \ldots, f_t^{m_t}|\boldsymbol{\omega}_t^{m_t}$, given a limited total resource capacity $\mathcal{C}$ and input set $\mathcal{S}$, **configuration space exploration** (CSE) aims to find the trace $\mathbf{f^k}|\boldsymbol{\omega}^\mathbf{k}$ within the configuration space $\mathcal{F}|\boldsymbol{\Omega}$ that achieves the highest expected performance without exceeding $\mathcal{C}$ of total cost.*

The problem can be formulated as a constrained optimization problem as follows:

$$\max_{F|\Omega \subseteq \mathcal{F}|\boldsymbol{\Omega}, X \subseteq \mathcal{S}} \mathbf{E}_\mathbf{x}[b(F|\Omega, \mathbf{x})] \quad \text{s.t. } c(F|\Omega, X) \leq \mathcal{C} \quad (5)$$

where $c(F|\Omega, X)$ is the total cost to consume X, which is defined as $\sum_{x \in X} c(F|\Omega, x)$. Since we intend to find the optimal system configuration generalizable to any unseen input, we use a random variable $\mathbf{x}$ to represent the system input, and the goal is to maximize the expected system performance over the random variable. To satisfy the constraint, the problem also involves selecting a subspace $F|\Omega \subseteq \mathcal{F}|\boldsymbol{\Omega}$ and a subset $X \subseteq \mathcal{S}$.

In this paper and in our initial case study, we adopt the constraint that each parameter must have a finite number of values in the configuration space, in order to support an efficient practical implementation and experimentation[7]. Moreover, we note the CSE problem focuses on determining the globally optimal trace general to all inputs, whereas researchers have observed that some components or configurations might improve the system performance on only some types of inputs while hurt the performance on other types [3]. Accordingly, one can further extend the CSE problem to a variant also incorporating the dependency on the input type[8].

We note the problem is trivial if the total cost to iteratively execute all the components and configurations for all inputs does not exceed available processing capacity. However, this is not the case for most real world problems, since the number of traces and executions grow exponentially as the phase increases. A typical information processing pipeline consisting of 12 components is shown in Section 6; with up to four parameters per component, and up to six options each parameter, there would be an estimated 6.050×10^{13} executions if all the unique traces were evaluated.

The CSE problem description appears isomorphic to constrained optimization. However, in the CSE case, both cost and benefit of a configured component are unknown to the model until the component is executed. In Section 4, we propose a solution framework.

4. MODELING & SOLUTIONS

In this section, we describe a general solution to tackle the objective function (Equation 5). In an ideal case where (1) the exact cost $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ of each configured component on each input is known, and (2) the performance indicator $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ for each configured component $f_t^c|\boldsymbol{\omega}_t^c$ is an *i.i.d.* random variable, the optimal solution to this problem can be yielded by adding configured components and inputs in descending order of *least cumulative cost* (LCC), which is defined as the sum of the component's original cost plus minimal additional cost to execute the components down the pipeline, and formulated as follows:

$$\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x) = c(f_t^c|\boldsymbol{\omega}_t^c, x) + \sum_{u=t+1}^{n} \min_{d=1}^{m_u} c(f_u^d|\boldsymbol{\omega}_u^d, x) \quad (6)$$

When we relax the first assumption used for the simplified case (the cost is revealed until the execution is done), the approaches for deterministic scheduling problems [15] do not provide an adequate solution. Instead, we refer to the stochastic scheduling problem [8] or stochastic knapsack problem [6], which makes a different assumption that job durations are random variables with known probability distributions. These approaches provide a more general formulation suitable for the CSE problem.

[7]To incorporate continuous parameters, one can discretize the feasible region by equally binning the region or sampling values from it. Directly tackling an optimization problem with continuous parameters requires a better understanding of how the behavior of the optimization objective varies as the parameter changes (e.g., linearity, convexity, etc.), and further relies on various continuous optimization techniques. Extending the CSE model for continuous parameter values is left to future work.

[8]As the intermediate data are persisted, a post-performance analysis can be easily conducted to identify the correlation between the input type and the trace performance.

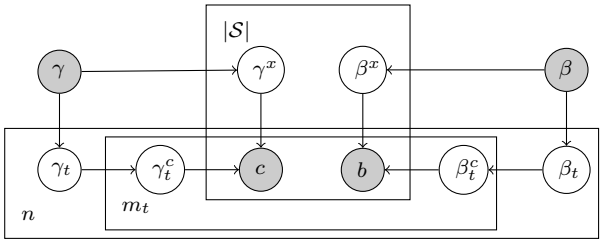

Figure 2: Generative process of cost and benefit distribution

We also relax the second assumption and assume the performance of a component is not *i.i.d.* For example, a weak configured component $f_t^c|\boldsymbol{\omega}_t^c$ (e.g., naive approach, or buggy code) tends to exhibit low performance for every input x; on the other hand, a complex input x (e.g., a very difficult question posed to a question answering system) may have a negative effect on the performance of all configured components. We are inspired to leverage *dependency of characteristics* and *prior knowledge* to dynamically (and more wisely) select which trace to evaluate next from the pool. We can utilize the execution history to estimate $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ by establishing a dependency between them. Intuitively, components that previously showed high benefit (or low cost) tend to achieve again high benefit (or low cost) combined with components from other phases, and components of low benefit (or high cost) tend to be pruned from the pool. In addition, we can also estimate the cost from prior knowledge, e.g., retrieving relevant passages usually requires more resources than tokenizing question texts in a retrieval system.

We develop the solution to the CSE problem based on hierarchical Bayesian modeling, where both prior knowledge and previous execution history are incorporated into the same framework. The basic idea is that we associate each cost $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ or benefit $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ with an unknown distribution, with parameters representing our knowledge and past observations. Starting with some known priors, we select the trace to execute based on the greedy algorithm for the stochastic knapsack problem, and then update the priors each time we finish an execution. The process repeats until $\mathcal{C}$ is reached. We first discuss how to estimate the cost and benefit of each component to solve the stochastic scheduling problem in Sections 4.1 and 4.2, and show an intuitive example in Section 4.3.

4.1 Modeling cost and benefit distributions

We apply a hierarchical Bayesian model to capture the component characteristics hierarchically inherited from global level and phase level. Specifically, we let two hyperparameters γ and β denote the average cost and benefit of a configured component globally across all phases and inputs, and then for each configured component $f_t^c|\boldsymbol{\omega}_t^c$ in a phase t, we introduce phase-level parameters γ_t and β_t to capture the characteristics shared by the components in phase t, and component-level parameters γ_t^c and β_t^c to model the performance of $f_t^c|\boldsymbol{\omega}_t^c$ regardless of the specific input[9]. Analogous to γ_t and β_t, we introduce parameters γ^x and β^x to denote the average cost and benefit of all configured components for each input x, which indicates the difficulty of processing each input. Given all the hyperparameters, the random variable $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ or $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ is the outcome from a series of generative processes from γ or β, which hence defines a joint probability distribution, shown in Figure 2.

[9]Due to the discrete configuration assumption, the same component with a different configuration is treated as a different component. One can easily extend the model to incorporate the dependency between the configured components and their base components.

Algorithm 1: Greedy algorithm for CSE problem

input : Hyperparameters γ, β and capacity $\mathcal{C}$
output : Optimal trace $\mathbf{f^c}^*|\boldsymbol{\omega^c}^*$

1 $X \leftarrow \emptyset, P \leftarrow \emptyset, Q \leftarrow \{(f_t^c|\boldsymbol{\omega}_t^c, x)\}_{t,c,x}$;
2 **while** $c(F|\Omega, X) < \mathcal{C}$ **do**
3 **foreach** $(f_t^c|\boldsymbol{\omega}_t^c, x) \in Q$ **do**
4 predict $b(f_t^c|\boldsymbol{\omega}_t^c, x)$, $c(f_t^c|\boldsymbol{\omega}_t^c, x)$, $\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)$;
5 $(f_t^{c'}|\boldsymbol{\omega}_t^{c'}, x') = \arg\max_{(f_t^c|\boldsymbol{\omega}_t^c, x) \in Q} h(f_t^c|\boldsymbol{\omega}_t^c, x)$;
6 $Q \leftarrow Q \setminus \{(f_t^{c'}|\boldsymbol{\omega}_t^{c'}, x')\}$;
7 $X \leftarrow X \cup \{x'\}$, $P[t] \leftarrow P[t] \cup \{f_t^{c'}|\boldsymbol{\omega}_t^{c'}\}$;
8 **foreach** $x \in X$ *and* $\mathbf{f^c}|\boldsymbol{\omega^c} \in \langle P \rangle$ **do**
9 **if** $\mathbf{f^c}|\boldsymbol{\omega^c}$ *has been executed for* x **then** continue;
10 execute to instantiate $c(\mathbf{f^c}|\boldsymbol{\omega^c}, x)$ and $b(\mathbf{f^c}|\boldsymbol{\omega^c}, x)$;
11 **foreach** t *and* $f_t^c|\boldsymbol{\omega}_t^c \in F|\Omega$ **do** update γ_t^c, γ^x, β_t^c, β^x;
12 $\mathbf{f^c}^*|\boldsymbol{\omega^c}^* = \arg\min_{\mathbf{f^c}|\boldsymbol{\omega^c} \in F|\Omega} \mathbf{E}_{\mathbf{x}}[b(\mathbf{f^c}|\boldsymbol{\omega^c}, \mathbf{x})]$;

Based on prior knowledge of expected cost and benefit, we can assign values for global hyperparameters γ and β, and optionally for some phase-level hyperparameters γ_t and β_t. Subsequently, unspecified phase-level hyperparameters and configured component level parameters γ_t^c and β_t^c can be estimated with *maximum a posteriori* method based on priors γ and β, and all observations (corresponding to previous execution records). For configured components that have not been executed, $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ can be predicted from estimated parameters γ_t^c, γ^x and β_t^c, β^x with Bayesian inference method.

4.2 CSE as a stochastic scheduling problem

Once $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ are estimated for the configured components in the pool, we can rewrite the objective for the deterministic scheduling problem (Equation 5) to a stochastic variant (aka stochastic knapsack problem [8, 6]), which assumes that job durations and/or rewards are random variables with known probability distributions, as follows:

$$\max_{F|\Omega \subseteq \mathcal{F}|\boldsymbol{\Omega}, X \subseteq \mathcal{S}} \mathbf{E}_b[b(F|\Omega, X)] \quad \text{s.t. } \mathbf{E}_c[c(F|\Omega, X)] \leq \mathcal{C} \tag{7}$$

where $\mathbf{E}_b$ and $\mathbf{E}_c$ represent the expected benefit and cost over the random variables $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ respectively.

Among the solutions (or scheduling policies) for the stochastic scheduling problem, *adaptive policies* have been studied extensively in the literature, which dynamically choose which traces to execute next based on previously executed traces. In general, they can achieve a better approximation ratio to the optimal solution [6], and can also be naturally integrated with the CSE problem, since the estimation of all parameters and distributions of $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ are dynamically updated as execution proceeds. A commonly adopted greedy policy for the (stochastic) knapsack problem is to sort the items by decreasing (expected) benefit, increasing (expected) cost, decreasing (expected) benefit density, or $b(f_t^c|\boldsymbol{\omega}_t^c, x)/\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)$ in the context of CSE problem. Finally, based on the benefit evaluated for each trace $\mathbf{f^c}|\boldsymbol{\omega^c}$ and input x, we select the traces by further measuring their generalizability to new data not yet realized, which can be estimated by various model selection methods, e.g. cross-validation, bootstrap, etc.

We present the solution in Algorithm 1, where $\langle P \rangle$ represents the configuration subspace spanned by the component pool P, $h(f_t^c|\boldsymbol{\omega}_t^c, x)$ is a heuristic function defined on $f_t^c|\boldsymbol{\omega}_t^c$ and x, to prioritize the components. Examples of h function can be cost relevant ($-\mathbf{E}_c[\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)]$), benefit relevant ($\mathbf{E}_b[b(f_t^c|\boldsymbol{\omega}_t^c, x)]$), benefit density ($\mathbf{E}_b[b(f_t^c|\boldsymbol{\omega}_t^c, x)]/\mathbf{E}_c[\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)]$), or profit ($\mathbf{E}_b[b(f_t^c|\boldsymbol{\omega}_t^c, x)] - \lambda\mathbf{E}_c[\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)]$), etc. Algorithm 1 applies an adaptive policy to execute the trace (lines 8–10), predict the random variables (lines 3, 4), and reestimate parameters (line 11) inside the loop (lines 2–11). This framework assumes no prior beliefs, nor probability distributions of component characteristics, which can be customized according to specific tasks. One can also consider to change the strategy function h over time, e.g. promoting configured components with greater benefit variance or greater cost once the exploration is stuck in a local minimum. Observing how different strategy functions affect the exploration process is left for future work.

Table 1: Posterior distributions for each of γ_t^c, γ_t and β_t^c, β_t.

	θ	ϕ
γ_t^c	$\tau_t^{c-2} n_t^c \bar{c_t^c} + ({\tau_t}^2 + \tau^2)^{-1}\gamma$	$\tau_t^{c-2} n_t^c + ({\tau_t}^2 + \tau^2)^{-1}$
γ_t	$\sum_c (\tau_t^{c2} + {\tau_t}^2)^{-1} n_t^c \bar{c_t^c} + \tau^{-2}\gamma$	$\sum_c (\tau_t^{c2} + {\tau_t}^2)^{-1} n_t^c + \tau^{-2}$
β_t^c	$\sigma_t^{c-2} n_t^c \bar{b_t^c} + ({\sigma_t}^2 + \sigma^2)^{-1}\beta$	$\sigma_t^{c-2} n_t^c + ({\sigma_t}^2 + \sigma^2)^{-1}$
β_t	$\sum_c (\sigma_t^{c2} + {\sigma_t}^2)^{-1} n_t^c \bar{b_t^c} + \sigma^{-2}\beta$	$\sum_c (\sigma_t^{c2} + {\sigma_t}^2)^{-1} n_t^c + \sigma^{-2}$

4.3 An example: 3-stage linear characteristics

To motivate our solution, we assume that the component characteristics follow a simple and intuitive 3-stage linear model [16], where each observation and hyperparameter follows a Gaussian distribution, with mean drawn from a distribution parameterized by the upper level in the hierarchy and variance predefined and fixed. In the simplified 3-stage linear model, we assume each input x to each configured component $f_t^c|\boldsymbol{\omega}_t^c$ follows uniform distribution over all possible inputs. The generative processes for $c(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(f_t^c|\boldsymbol{\omega}_t^c, x)$ can therefore be instantiated from the general solution framework (Figure 2) and formulated as follows:

$$c(f_t^c|\boldsymbol{\omega}_t^c, x) \overset{\text{iid}}{\sim} \mathcal{N}(\gamma_t^c, {\tau_t^c}^2) \quad b(f_t^c|\boldsymbol{\omega}_t^c, x) \overset{\text{iid}}{\sim} \mathcal{N}(\beta_t^c, {\sigma_t^c}^2)$$
$$\gamma_t^c \overset{\text{iid}}{\sim} \mathcal{N}(\gamma_t, {\tau_t}^2) \quad \beta_t^c \overset{\text{iid}}{\sim} \mathcal{N}(\beta_t, {\sigma_t}^2)$$
$$\gamma_t \overset{\text{iid}}{\sim} \mathcal{N}(\gamma, \tau^2) \quad \beta_t \overset{\text{iid}}{\sim} \mathcal{N}(\beta, \sigma^2)$$

We derive the posterior distributions for γ_t^c, γ_t and β_t^c, β_t, which can be used to update the hyperparameters (line 11 in Algorithm 1). All the posterior probabilities follow Gaussian distribution of the form $\mathcal{N}(\phi^{-1}\theta, \phi^{-1})$, and the values of θ and ϕ are listed in Table 1 for each parameter, where n_t^c represents the number of times $f_t^c|\boldsymbol{\omega}_t^c$ has been executed thus far, and $\bar{c_t^c}$ and $\bar{b_t^c}$ are the average cost and benefit of the component in the history. We see that at the beginning of the experiment, when $n_t^c = 0$ for all t and c, each parameter takes the same value from the user specified hyperparameter (γ or β), and as the experiment proceeds, each parameter is dynamically shifted away from the value of hyperparameter to better model the execution behavior of the phase or component ($\bar{c_t^c}$ and $\bar{b_t^c}$).

Finally, we define function h (line 5 in Algorithm 1) as the profit ($\mathbf{E}_b[b(f_t^c|\boldsymbol{\omega}_t^c, x)] - \lambda\mathbf{E}_c[\text{LCC}(f_t^c|\boldsymbol{\omega}_t^c, x)]$) in the rest of the paper, which makes the CSE framework less expensive to implement and execute. Equivalently, we can dynamically prune the components of either high cost or low benefit. In Section 6, we illustrate how the model was implemented in the framework to tackle a real-world CSE task, using the open-source package presented in Section 5.

5. OPEN SOURCE IMPLEMENTATION

To facilitate easy integration with existing components, we require a system that is simple to configure and powerful enough to sift through thousands to trillions of option combinations to determine which represent the best system configuration. In this section,

we present the CSE framework implementation, a distributed system for a parallel experimentation test bed based on UIMA, configured using declarative descriptors. We highlight the features of the implementation in this section. Source code, examples, and other resources are publicly available. Details of the architecture design and implementation can be found in the extended version [36].

Distributed architecture. In Section 4, we focus on discussion of single-machine solutions to the CSE problem. We have extended the CSE framework to execute the task set $\mathcal{X}$ in parallel on a distributed system based on UIMA-AS[10], where the configured components are deployed into the cluster beforehand; the execution, fault tolerance and bookkeeping are managed by a master server.

Declarative descriptors. To leverage the CSE framework, users specify how the components $f_t^c \in \mathcal{F}$ should be organized into phases in a pipeline, which values need to be specified for each configuration $\boldsymbol{\omega}_t^c$, what is the input set $\mathcal{X}$, and what measurements should be applied. Our implementation introduces *extended configuration descriptors* (ECD) based on YAML[11] format.

Configurable evaluation. CSE supports the evaluation of component performance based on user-specified evaluation metrics and gold-standard outputs at each phase; these measurements are also used to estimate the benefit for each component and trace ($b(f_t^c|\boldsymbol{\omega}_t^c, x)$ and $b(\mathbf{f^c}|\boldsymbol{\omega^c}, x)$). Commonly-used information retrieval metrics are implemented in the open source software.

CSE also provides the capability to calculate statistical significance of the performance difference between traces on a given task. This is important when attempting to disprove the null hypothesis that the best baseline trace is as good as alternative traces, and crucial to understanding each component's contribution (Section 6.4) and prioritizing the traces when adapting to new tasks (Section 6.6).

Automatic data persistence. As mentioned in Section 3, if two traces share the same prefix, it isn't necessary to repeat executions for the configured components along the prefix if we assume that intermediate results are kept in a repository accessible from any trace. Moreover, intermediate data are also useful for error analysis, trace performance analysis, and reproduction of previous experimental results. Therefore, our implementation includes an automatic persistence strategy for data and experiment configuration.

Global resource caching. Online resources are sometimes temporarily unavailable, and are occasionally updated, changing their contents; to support reproducibility of results, we implemented a generic resource caching strategy inside the CSE framework. In addition to ensuring the reproducibility of results when external KBs are queried, global resource caching can also speed up CSE experiments by avoiding replicated service calls across traces.

Configurable configuration selection and pruning. The component selection strategies can be configured by the user; strategies corresponding to the heuristic function described in Section 4.2 are implemented in the open source software. K-fold cross-validation, leave-one-out cross-validation and bootstrap methods are integrated into the open source version to estimate the generalization performance of top traces.

Graphical user interface. A Web GUI has been integrated into the open source release featuring three views: *experiment*, *ad hoc*, and *trend*. The *experiment* view helps users monitor configuration exploration progress and evaluation results, and view current and past persisted intermediate data for error analysis. The *ad hoc* view enables users to submit an *ad-hoc* input and inspect the system output. The *trend* view allows users to visualize performance change across experiments over a selected period of time.

[10] http://uima.apache.org/doc-uimaas-what.html

[11] http://www.yaml.org/

6. CSE FOR TREC GENOMICS QA

In this section, we apply the CSE framework to the problem of building an effective biomedical question answering (BioQA) system from available components and component options. The goal of this work is to demonstrate the effectiveness of the proposed approach and the current open source implementation. Specifically, we employed the topic set and benchmarks from the question answering task of TREC Genomics Track [11], as well as commonly-used tools, resources, and algorithms cited by participants. A set of basic components was selected and adapted to the CSE framework implementation by writing wrapper code where necessary. Then an experiment configuration descriptor was defined for the resulting set of configured components. This configuration space was explored with the CSE algorithm automatically, yielding an optimal and generalizable configuration which outperformed published results of the given components for the same task. Detailed component descriptions and result analysis can be found in the extended technical report [36].

6.1 Task & data set description

The TREC 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 total of 64 genomics-related topics were developed for this task, asking about biological processes or relationships between biomedical entities. 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. Details can be found in the overview paper [11].

To evaluate the generality of the framework in selecting optimal traces for novel data, and further test the performance of the selected traces, we implemented nested K-fold cross-validation (leave-one-out, 5-fold and 10-fold) and nested bootstrap methods, where two subsets of 28 topics used by TREC Genomics 2006 were held out for trace selection (validation set) and performance evaluation (test set) respectively. Furthermore, we employed 36 topics from TREC Genomics 2007 to test the adaptability of traces for a similar but slightly different task, with questions asking for lists of specific biomedical entities (in Section 6.6).

6.2 Resource and algorithm integration

We mainly considered four different aspects when collecting resources and implementing algorithms to build a biomedical question answering system: *NLP tools*, *KBs*, *retrieval tools*, and *reranking algorithms*. We considered popular and successful approaches reported in the TREC Genomics literature to explore with the CSE framework. We summarize these components in Table 2, and briefly describe them in the rest of this subsection.

First, many successful systems 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, etc. To establish a benchmark for NLP in our system, we focused on several rule-based approaches for generating lexical variants, and supervised learning methods provided by LingPipe.

Second, tens of biomedical KBs have been organized and maintained. Some involve knowledge from all areas of biomedical research, while others focus on only a few subareas, e.g., disease, genomics, etc. The implication is that for a particular topic, the qual-

Table 2: Summary of integrated components

Category	Components
NLP tools	LingPipe HMM based tokenizer LingPipe HMM based POS tagger [19] LingPipe HMM based named entity recognizer Rule based lexical variant generator [1, 27, 34, 30]
KBs	UMLS for syn/acronym expansion [27, 7, 38, 33] EntrezGene for syn/acronym expansion [30, 34, 7, 38, 18] MeSH for syn/acronym expansion [30, 38, 34, 9]
Retrieval tools	Indri system [19, 33, 28, 27]
Reranking algorithms	Important sentence identification [4, 33, 32, 34, 38, 9] Term proximity based ranking [4, 33] Score combination of different retrieval units [27, 30, 33] Overlapping passage resolution

ity of the answer generated by the system may vary by the KB(s) that are used. We focused on three resources most popular with TREC Genomics participants: UMLS, EntrezGene, and MeSH.

Next, to retrieve relevant documents from the unstructured biomedical corpus, we need to rely on a widely-used open-source search engine, e.g. Indri[12]. Finding an optimal retrieval configuration requires selecting parameter values specific to each search engine, such as the retrieval model and the parameters in the scoring function, smoothing method, query formulation strategy, etc.

Finally, a number of participants performed postprocessing on their passage output. This was done to refine the boundaries and ranks of retrieved passages, using techniques such as evidence targeting, answer merging and rescoring, etc. The aspect that categorizes these approaches is reranking algorithms.

We followed the basic pipeline phases for a question answering system [31] and implemented a domain-independent QA framework. We integrated benchmarks, task-specific evaluation methods, as well as 12 components, and specified one to several values for each parameter associated with each component, and plugged them into the framework. To support straightforward reproducibility of results, the frameworks, components, and support materials are available online as part of our open source release.

6.3 Experimental results

We designed two experiments to evaluate the CSE framework implementation, given moderate and large-scale configurations respectively. In the first experiment, we limited the number of options and consequently derived 32 configured components ($\mathcal{F}|\Omega$), which could produce a maximum number of 2,700 traces and require 190,680 executions to process all the questions. An experiment of this complexity was expected to execute within a day on our available test hardware (corresponding to $\mathcal{C}$ with resource defined by execution time). The mean and standard deviation for global execution time were initially set as 10 seconds (γ and τ) and the expected benefit for a single configured component was set as 0.1 in terms of PsgMAP (β and σ), and the parameters at phase and component levels were estimated by an empirical Bayesian method. In the second experiment, we planned to test the scalability of the implementation by aggressively setting up to six values for each parameter, and thus yielded a CSE problem with 2,946 configurations and 1.426×10^{12} traces, requiring 6.050×10^{13} executions in total to evaluate the entire space.

We compare the settings for the two experiments with the official TREC 2006 Genomics test results for the participating systems in Table 3. Although many more configured components were implemented by the original participants, only 92 different traces were evaluated. We estimate the number of components, configurations and executions evaluated by the official test, and show the evaluation results [12] in Table 3 for reference. The results reported from the two experiment settings were evaluated with the nested leave-one-out cross-validation method. Different strategies of estimating generalizability of traces are further compared in Section 6.6.

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

Table 3: Overview of experimental settings and comparison with TREC Genomics 2006 participants

		Participants	CSE	Scaled CSE
# Component		∼1,000	12	12
# Configuration		∼1,000	32	2,946
# Trace		92	2,700	$\sim1.426\times10^{12}$
# Execution		∼1,000	190,680	$\sim6.050\times10^{13}$
Capacity (hours)		N/A	24	24
DocMAP	Max	.5439	.5648	.5072
	Median	.3083	.4770	.3509
	Min	.0198	.1087	.2679
PsgMAP	Max	.1486[a]	.1773	.1181
	Median	.0345	.1603	.0713
	Min	.0007	.0311	.0164

[a] 0.174 was reported after the official evaluation [38].

From Table 3, we can see that the best system derived automatically by the proposed CSE framework outperformed the best participating system in terms of both DocMAP and PsgMAP, with fewer, more basic components. Similarly constrained exploration of the much larger Scaled CSE configuration failed to yield a system that performs as well in comparable time, highlighting the importance of *a priori* human selection of appropriate configurations and/or hyperparameter values. Nevertheless, the configuration yielded by exploring the larger space at the same capacity cost still outperformed most of the participating systems reported in the TREC Genomics track paper. The detailed pipeline configuration of the best system derived by the proposed CSE framework is described in Section 6.4 as the baseline to analyze component contributions.

6.4 Component analysis

As the best trace has been discovered, we report the performance of traces with only a single component or configuration different from the best trace, in terms of PsgMAP, to provide a useful comparison. In this section, we investigate in detail how each component or configuration contributes to overall system performance. The results are shown in Table 4 for nominal configurations, e.g. different KBs, score transformation methods, and Figure 3 for real-valued configurations, e.g. smoothing parameters, term weights, etc. We also report the significance test (t-test) results, and label the scores with different significance levels, in Table 4. The configuration of the best system is also shown in Table 4, which uses most available resources (KBs, reranking algorithms, NLP tools).

We see that the performance of leveraging various sources varied for synonym expansion and acronym expansion. Incorporation of MeSH in synonym expansion could make the biggest contribution to the performance, while both UMLS and EntrezGene also benefited the system's performance. However, they also contained more noisy content than MeSH, which hurt the performance when combining with MeSH. As most acronyms were expanded as synonyms, integrating any combination of KBs hardly affected the performance, though we find that EntrezGene is a useful source for acronym expansion compared with others. Unsurprisingly, we confirm that synonyms should be considered in extracting important sentences and reranking, and filtering overlapping passages could better help the task than only filtering identical passages. Different transformation strategies hardly affected the overall performance.

For the real-valued parameters, we can see from Figure 3 that altering weights for concept terms or verbs could greatly change the

Table 4: Variation of nominal configurations for all components, including synonym and acronym expansion, where UMLS (U), MeSH (M), and EntrezGene (E) were integrated, lexical variants (LV), sentence extraction (SE), proximity based ranking (PR), filtering of overlapping (OV) or identical (ID) passages, and combining scores transformed by exponential (EX), normalization (NO), exponential of normalization (EN), normalization of exponential (NE), logarithmic (LG), reciprocal of rank (RR). N and Y represent *not integrated* and *integrated*. S and NS represent *synonym incorporated* and *not incorporated*. Significance levels of 0.1, 0.05, 0.01, and 0.005 are indicated by a sharp symbol (♯) and one to three star symbols (⋆) respectively.

Parameter	Value	DocMAP	PsgMAP	Parameter	Value	DocMAP	PsgMAP	Parameter	Value	DocMAP	PsgMAP
Best baseline system (Acronym: U, Filter: ID, LV: Y, PR w/ S, SE w/ S, Synonym: M+E, Combiner: RR)										.5257	.1744
Synonym expansion with KBs	N	.4169⋆⋆	.1256⋆	Acronym expansion with KBs	N	.5253	.1751	PR	NS	.5444	.1693♯
	U	.4619♯	.1439		M	.5226♯	.1753	Document and passage retrieval score combiner	EN	.5353	.1773
	M	.5050	.1710		E	.5250	.1751		EX	.5362	.1759
	E	.4302⋆	.1233⋆		U+M	.5236	.1746		LG	.5472	.1744
	U+M	.4942	.1609		U+E	.5254	.1744		NE	.5471	.1730
	U+E	.4639♯	.1405		M+E	.5223⋆	.1752		NO	.5473	.1744
	U+M+E	.4959	.1603		U+M+E	.5233♯	.1746		N	.5471	.1730
LV	N	.4696	.1613	SE	NS	.4545⋆⋆⋆	.1259⋆⋆⋆	Filter	OV	.5342	.1773

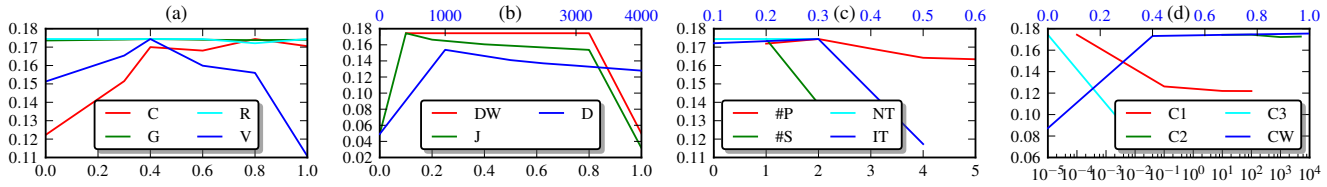

Figure 3: Variation of real-valued configurations for all components. Y-axis corresponds to PsgMAP scores. (a) Weighting for concept terms (C), regular terms (R), gene names (G), and verbs (V). (b) Document relevance score weighting when combined with passage relevance scores (DW), Dirichlet (D) and Jelinek-Mercer (J) smoothing parameter. (c) Sentence extraction parameters: number of sentences in a passage (#S), number of passages in a paragraph (#P), importance and neighboring sentence thresholds (IT and NT). (d) Proximity-based ranking parameters (C1, C2, C3) and document score weighting when combined (CW).

performance. In particular, concept terms favored higher weights and verbs favored moderate weights (~ 0.4). Jelinek-Mercer outperformed Dirichlet smoothing for this task with the best performance achieved when the parameters were set as 0.1 and 1000. We also found the performance was improved with the parameters set to lower values than those reported in their original paper for the important sentence extraction and reranking algorithms [30].

6.5 CSE performance analysis

In this subsection, we demonstrate how the CSE framework discovered the best trace presented in Section 6.3 by analyzing the parameters estimated by the framework at different stages.

We show in Figure 4 the likelihood and posterior distributions of cost estimated by the CSE framework when the experiment completed. First, Figures (a) and (b) represent likelihood estimation of cost for five phases: score combination methods (C), overlapping passage solutions (O), retrieval strategists (R), sentence extraction and reranking (S), and term proximity based reranking (P) respectively. We can clearly see that the framework discovered each phase had a different distribution of execution time, and C and O components usually performed faster than R, S, and P. Figures (c) to (d) represent likelihood estimation of component-level cost for these phases. We see that the configured components in each of C, O, and P had similar cost distributions, while the behavior of different S and R components varied, since the computational complexity of these components tended to be easily affected by different configurations. Figures (e) to (h) are the posterior estimations of γ_t and γ_t^c to predict $c(f_t^c|\boldsymbol{\omega}_t^c)$ for unseen components. Since enough samples were provided at the end, the mean of each γ_t or γ_t^c was close to that of the likelihood estimation, and since more samples were available to estimate the phase-level costs than the component-level costs, the variance of γ_t tended to be smaller than that of γ_t^c.

We further investigate how the likelihood and posterior distributions of cost changed through the execution process by showing both distributions estimated when the experiment completed 1% of total execution tasks in Figure 5. Comparing with Figure 4, we see that the means of both distributions were smaller than those estimated at the end, due to the randomness of initially sampled components and input sets; the variances of the likelihood distributions were smaller due to fewer samples. According to Algorithm 1, components of downstream phases (C and O) tended to be sampled and tested more frequently than those of upstream phases (R, S, and P), due to smaller LCC scores; variances of the estimated posterior distributions of R, S, P components tended to be greater than those of C and O, which is clearly shown in Figures (e) to (h).

Estimating benefit had a similar process as estimating cost. We use various configurations to illustrate the estimated likelihood and posterior distributions of benefit when the 20% of total execution tasks were completed. Figures 6(a) and (b) show the likelihood estimation of the benefit for different configurations of the reranking algorithms and KBs, which exhibited different effects. Since KBs were integrated at the beginning at the pipeline while reranking algorithms were placed at the end, very few combinations of various KBs had been tested thus far, which correspond to the few "sharp" distributions, and the likelihood distributions corresponding to others remain unknown; on the other hand, different reranking methods were tested more equally frequently, as a result, their distributions were more alike. This assumption can also be justified from Figures (c) and (b), which correspond to the posterior estimations of both component-level benefit means, where variances of posterior distributions for reranking algorithms were close, while those corresponding to the untested KBs took the global variance.

6.6 Generalizability analysis

In this subsection, we first demonstrate the performance of trace selection results by various resampling methods. Besides the leave-one-out cross-validation, we also applied 5-fold cross-validation, 10-fold cross-validation (each repeated 100 times) and bootstrap (repeated 300 times) for model selection and performance evaluation. We find that the 10 highest ranked traces produced by all the

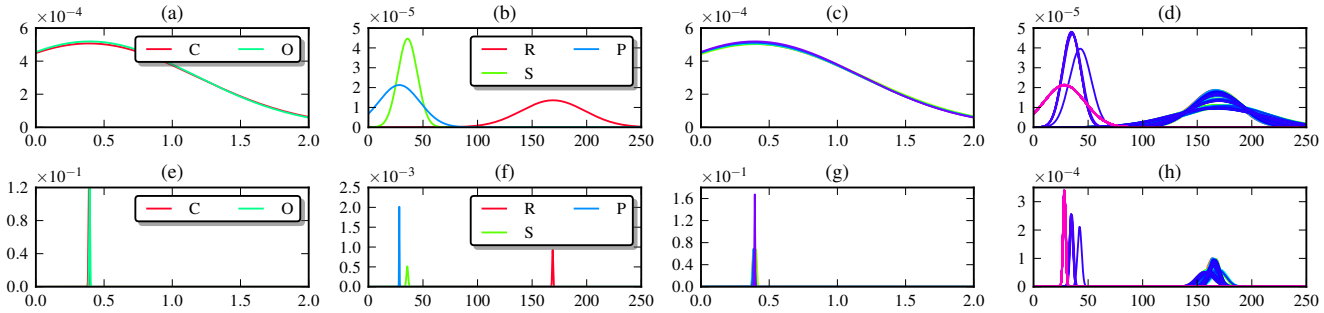

Figure 4: Estimation of likelihood and posterior of cost at the end of the experiment. (a) and (b) represent likelihood estimation of phase-level cost, where C, O, R, S, P correspond to score combination methods, overlapping passage solutions, retrieval strategists, sentence extraction and reranking, term proximity based reranking respectively. (c) and (d) represent likelihood estimation of cost at component level. (e) to (h) are the posterior estimations of cost mean γ_t and γ_t^c. X-axis is the execution time in seconds.

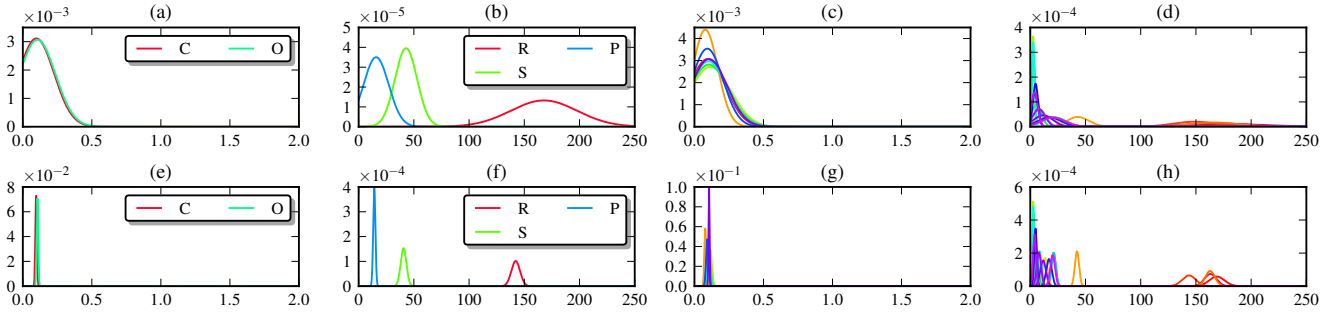

Figure 5: Estimation of likelihood and posterior of cost when the experiment completed 1% of total execution tasks. (a) to (h) correspond to Figure 4 (a) to (h). X-axis is the execution time in seconds in all subfigures.

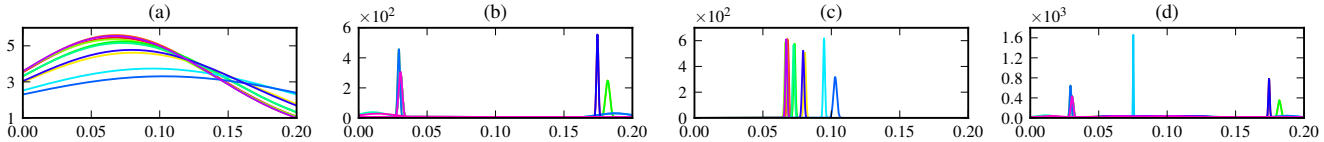

Figure 6: Estimation of likelihood and posterior of component-level benefit when 20% of total execution tasks were completed. (a) shows the likelihood estimation of the benefit for various sentence extraction and reranking methods, and (b) shows the likelihood estimation of the benefit of different KBs. (c) and (b) correspond to the posterior estimations. X-axis is the PsgMAP score.

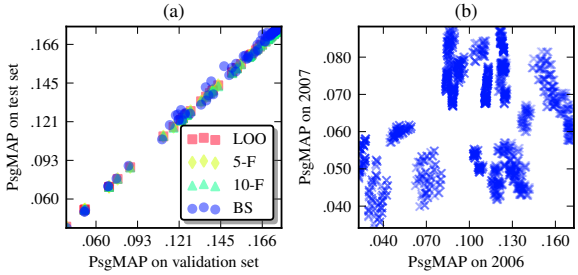

Figure 7: Generalizability of traces with various resampling methods on (a) holdout topic set from 2006 and (b) unseen topic set from 2007. X- and Y-axis correspond to PsgMAP scores.

resampling methods are identical; we further compare the performance of all traces evaluated on the validation set and test set in Figure 7(a). We can see that the traces produced by the proposed framework with a resampling based trace selection method are generalizable to unseen data for the task, and cross-validation methods are less biased than bootstrap method.

We also tested these automatically selected configurations on 36 topics from TREC Genomics 2007 to test their adaptability for a different but similar task. TREC Genomics 2007 topics are in the form of questions asking for lists of specific entities, e.g. "What [GENES] are genetically linked to alcoholism?". For this example question, the expected output would be a ranked list of passages that relate one or more entities of type GENE to alcoholism.

Table 5: Result on TREC Genomics 2007 data set

	DocMAP			PsgMAP		
	Max	Median	Min	Max	Median	Min
Participants	.3286	.1897	.0329	.0976	.0565	.0029
CSE	.3144	.2480	.2067	.0984	.0763	.0412

We used the 2007 topic set to test the configurations selected by CSE for the 2006 data set (Table 5). We can see that the best trace slightly outperformed the best reported in [13]. The difference in performance on the 2006 and 2007 datasets is plotted in Figure 7(b), which is clearly larger than the performance difference between different folds drawn from the 2006 data set. The best configuration is also different, and favors different KBs and weights. Nevertheless, the coefficient of correlation between 2006 and 2007 tests (0.272) still confirms that the best configurations from 2006 tend to perform better for 2007. We also see that data points are clearly clustered into groups, and the performance difference between clusters tends to be more significantly different than that within each cluster. This indicates that traces which statistically significantly differ from the top traces should be explored to better diversify the system outputs and enhance the likelihood of achieving optimal performance.

Finally, we used the CSE framework to automatically configure a different type of biomedical question answering system for the QA4MRE (Question Answering for Machine Reading Evaluation) task[13] at CLEF, where 12 UIMA components were first developed, and then 46 configurations were specified for CSE framework to explore 1,040 different combinations with 1,322 executions. The CSE framework identified a better combination, which achieved 59.6% performance gain over the original pipeline. Details can be found in the working note paper [24].

7. CONCLUSION & FUTURE WORK

In this paper, we investigate the problem of *configuration space exploration* (or *CSE*). We introduce the *CSE framework* as a solution for building and exploring configuration spaces for information systems. The framework was used to implement a biomedical information system for the TREC Genomics QA by automatically and effectively exploring over a trillion system configurations.

In future work we plan to investigate to extend the CSE model for continuous parameter values, and incorporate the dependency on the input type. We also plan to apply CSE to the creation of other information processing pipelines in other task domains.

8. REFERENCES

[1] S. Abdou and J. Savoy. Report on the trec 2006 genomics experiment. In *Proceedings of TREC'06*, 2006.

[2] M. Agosti, N. Ferro, and C. Thanos. Desire 2011: workshop on data infrastructures for supporting information retrieval evaluation. *SIGIR Forum*, 46(1):51–55, May 2012.

[3] S. M. Bailey. Providing question coverage through answer type classification in natural language question answering (nlqa) systems. diploma thesis, Georgetown University, Washington, DC, April 2001.

[4] S. Bergler, J. Schuman, J. Dubuc, and A. Lebedev. Bioki, a general literature navigation system at trec genomics 2006. In *Proceedings of TREC'06*, 2006.

[5] Y. Chen, X. Yin, Z. Li, X. Hu, and J. Huang. A lda-based approach to promoting ranking diversity for genomics information retrieval. *BMC Genomics*, 13(3):1–10, 2012.

[6] B. C. Dean, M. X. Goemans, and J. Vondrak. Approximating the stochastic knapsack problem: The benefit of adaptivity. In *Proceedings of FOCS'04*, pages 208–217, 2004.

[7] D. Demner-Fushman, S. M. Humphrey, N. C. Ide, R. F. Loane, P. Ruch, M. E. Ruiz, L. H. Smith, L. K. Tanabe, W. J. Wilbur, and A. R. Aronson. Finding relevant passages in scientific articles: Fusion of automatic approaches vs. an interactive team effort. In *Proceedings of TREC'06*, 2006.

[8] C. Derman, G. J. Lieberman, and S. M. Ross. A renewal decision problem. *Management Science*, 24(5):pp. 554–561, 1978.

[9] K. C. Dorff, M. J. Wood, and F. Campagne. Twease at trec 2006: Breaking and fixing bm25 scoring with query expansion, a biologically inspired double mutant recovery experiment. In *Proceedings of TREC'06*, 2006.

[10] R. Eckart de Castilho and I. Gurevych. A lightweight framework for reproducible parameter sweeping in information retrieval. In *Proceedings of DESIRE'11*, pages 7–10, 2011.

[11] W. Hersh and E. Voorhees. Trec genomics special issue overview. *Inf. Retr.*, 12(1):1–15, Feb. 2009.

[12] W. R. Hersh, A. M. Cohen, P. M. Roberts, and H. K. Rekapalli. Trec 2006 genomics track overview. In *Proceedings of TREC'06*, 2006.

[13] W. R. Hersh, A. M. Cohen, L. Ruslen, and P. M. Roberts. Trec 2007 genomics track overview. In *Proceedings of TREC'07*, 2007.

[14] X. Huang and Q. Hu. A bayesian learning approach to promoting diversity in ranking for biomedical information retrieval. In *Proceedings of SIGIR'09*, pages 307–314, 2009.

[15] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of SFCS'82*, pages 312–320, 1982.

[13] http://celct.fbk.eu/QA4MRE/

[16] D. V. Lindley and A. F. M. Smith. Bayes estimates for the linear model. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(1):pp. 1–41, 1972.

[17] J. Liu, C. Liu, M. Cole, N. J. Belkin, and X. Zhang. Exploring and predicting search task difficulty. In *Proceedings of CIKM'12*, pages 1313–1322, 2012.

[18] Y. Lu, J. Jiang, X. Ling, X. He, and C. Zhai. Language models for genomics information retrieval: Uiuc at trec 2007 genomics track. In *Proceedings of TREC'07*, 2007.

[19] E. Meij, M. Jansen, and M. de Rijke. Expanding queries using multiple resources. In *Proceedings of TREC'06*, 2006.

[20] E. Meij, D. Trieschnigg, M. de Rijke, and W. Kraaij. Conceptual language models for domain-specific retrieval. *Inf. Process. Manage.*, 46(4):448–469, July 2010.

[21] T. Mitamura, E. Nyberg, H. Shima, T. Kato, T. Mori, C.-Y. Lin, R. Song, C.-J. Lin, T. Sakai, D. Ji, and N. Kando. Overview of the ntcir-7 aclia tasks: Advanced cross-lingual information access. In *Proceedings of NTCIR-7*, pages 11–25, 2008.

[22] T. Mitamura, H. Shima, T. Sakai, N. Kando, T. Mori, K. Takeda, C.-Y. Lin, R. Song, C.-J. Lin, and C.-W. Lee. Overview of the ntcir-8 aclia tasks: Advanced cross-lingual information access. In *Proceedings of NTCIR-8*, pages 15–24, 2010.

[23] X. Mu and K. Lu. Towards effective genomic information retrieval: The impact of query complexity and expansion strategies. *J. Inf. Sci.*, 36(2):194–208, Apr. 2010.

[24] A. Patel, Z. Yang, E. Nyberg, and T. Mitamura. Building an optimal question answering system automatically using configuration space exploration (cse) for qa4mre 2013 tasks. In *Proceedings of CLEF 2013*, 2013.

[25] H. K. Rekapalli, A. M. Cohen, and W. R. Hersh. A comparative analysis of retrieval features used in the trec 2006 genomics track passage retrieval task. In *Proceedings of AMIA'07*, pages 620–624, 2007.

[26] M. E. Ruiz. Ub at trec genomics 2006: Using passage retrieval and pre-retrieval query expansion for genomics ir. In *Proceedings of TREC'06*, 2006.

[27] L. Si, J. Lu, and J. Callan. Combining multiple resources, evidences and criteria for genomic information retrieval. In *Proceedings of TREC'06*, 2006.

[28] M. D. Smucker. Umass genomics 2006: Query-biased pseudo relevance feedback. In *Proceedings of TREC'06*, 2006.

[29] N. Stokes, Y. Li, L. Cavedon, and J. Zobel. Exploring criteria for successful query expansion in the genomic domain. *Inf. Retr.*, 12(1):17–50, Feb. 2009.

[30] L. Tari, G. Gonzalez, R. Leaman, S. Nikkila, R. Wendt, and C. Baral. Asu at trec 2006 genomics track. In *Proceedings of TREC'06*, 2006.

[31] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of SIGIR'03*, pages 41–47, 2003.

[32] D. Trieschnigg, D. Hiemstra, F. de Jong, and W. Kraaij. A cross-lingual framework for monolingual biomedical information retrieval. In *Proceedings of CIKM'10*, pages 169–178, 2010.

[33] J. Urbain, N. Goharian, and O. Frieder. Iit trec 2006: Genomics track. In *Proceedings of TREC'06*, 2006.

[34] R. Wan, V. N. Anh, I. Takigawa, and H. Mamitsuka. Combining vector-space and word-based aspect models for passage retrieval. In *Proceedings of TREC'06*, 2006.

[35] D. C. Wimalasuriya and D. Dou. Components for information extraction: ontology-based information extractors and generic platforms. In *Proceedings of CIKM'10*, pages 9–18, 2010.

[36] Z. Yang, E. Garduno, Y. Fang, A. Maiberg, C. McCormack, and E. Nyberg. Cse: Extending the uima framework for automatically building complex information systems. Technical Report CMU-LTI-13-XXX, Carnegie Mellon University, 2013.

[37] W. Zhou, C. Yu, and W. Meng. A system for finding biological entities that satisfy certain conditions from texts. In *Proceedings of CIKM'08*, pages 1281–1290, 2008.

[38] W. Zhou, C. T. Yu, V. I. Torvik, and N. R. Smalheiser. A concept-based framework for passage retrieval at genomics. In *Proceedings of TREC'06*, 2006.