Technische Universität München

Faculty of Informatics

Bachelor's Thesis in Bioinformatics

PubSeq: Amino Acid-based Search Engine for MEDLINE Abstracts

PubSeq: Aminosäuresequenz basierte Suchmachine für MEDLINE Abstrakten

Author:
Pandu Raharja

Supervisor:
Prof. Dr. Burkhard Rost

Advisors:
Dr. Guy Yachdav
Juan Miguel Cajuela

August 2015

Declaration of Authorship

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Signed: ____

Date: ____

"People usually think that progress consists in the increase of knowledge, in the improvement of life, but that isn't so. Progress consists only in the greater clarification of answers to the basic questions of life. The truth is always accessible to a man. It can't be otherwise, because a man's soul is a divine spark, the truth itself. It's only a matter of removing from this divine spark (the truth) everything that obscures it. Progress consists, not in the increase of truth, but in freeing it from its wrappings. The truth is obtained like gold, not by letting it grow bigger, but by washing off from it everything that isn't gold."

L. N. Tolstoy

Abstract

Background

In genetic research, it is imperative for biomedical researcher to stay updated on the current state of identified proteins. It was hard – and is getting harder, especially after widespread use of Next-Generation Sequencing (NGS) – for researcher to keep updated on the research into protein he/she is currently investigating. This is furthermore exacerbated by the fact that existing search engines only allow querying abstracts using protein names.

Methods

In this project, I present the first search engine that allows user to find all publications mentioning proteins that are similar or identical to the one he/she's interested in. To achieve this, I created a Solr Index that lists down all gene names that were mentioned in each of MEDLINE abstracts and titles. I then populated the index by scanning the whole MEDLINE corpus, tagging protein names found in title and abstract, normalizing those names into UniProt IDs and pushing the ID mentions onto Solr index. Given user's sequence query, the program runs a BLAST on the sequence and normalizes blast results to UniProt IDs. The program then retrieves articles mentioning this ID and return these to user. For the good usability I offer the whole service in a web interface available in following address.

Zusammenfassung

Hintergrund

In genetischer Forschung ist es erzwingend, dass der/die biomedische ForscherIn mit der aktuellen Landschaft von identifizierten Protein sich ständig informiert. Es war schwierig – und wird immer schwieriger sein, vor allem nach dem verbreiteten Ansatz von Next Generation Sequencing (NGS) Technoligien, um der/die Forscherin mit dem Protein von der Interesse in aktuellem Zustand zu halten. Die Tatsache, dass die aktuelle Suchmaschine von den Artikeln nur Namenbasierte Suche unterstützt, hilft leider nicht weiter.

Methoden

In diesem Projekt stellen wir eine Suchmaschine vor, die erlaubt den Benützer, basiert auf Aminosäuresequenz nach den Artikeln suchen, die das Protein oder die Ähnliche erwähnen. Um dies zu erreichen hatten wir einen Solr Index erstellt, der alle erwähnte Proteine innerhalb jedes MEDLINE Artikels auflistet. Wir füllen sich diesen Index in dem wir den gesamten MEDLINE Corpus durchscannen und alle Proteinname mithilfe eines NLP-Programs detektieren. Wir wurden dann diese Namen in UniProt IDs normalisieren. Diese normalisierte Namen wurden schlielich in unserem Solr Index hinzufügen. Um die Benutzbarkeit dieser Dienstleistung zu maximieren hatten wir auch eine Webschnittstelle entwurfen, die in folgender Addresse verfügbar ist.

Acknowledgements

First and mostly, I would like to thank Prof. Burkhard Rost for the holistic supports provided, be it through the lab infrastructures or himself personally. I would also to thank two of my advisors, Dr. Guy Yachdav and Juan Miguel Cajuela, who have in spite of their busy schedules and great distances (and time differences) patiently advised me through the project. Knowing that both are about to finish their PhD programs, I wish them all the best of luck in their future endeavors. I would also like to thank Andre Ofner for helping with the evaluation of the systems.

Also, I would like to thank Tatyana Goldberg for administrative and all-around support during my stay at the lab. Also my gratitude for Tim Karl, our awesome system administrator, who has helped us tremendously in incorporating each of the cogs in our pipeline into one coherent system. While not involved in our project personally, I would like to thank Prof. Lars Juhl Jensen of University of Copenhagen for giving us access to his tagger program.

I would also to thank Andre Ofner for the help in validating the systems. I would also like to express my gratitude towards Robert Leaman and Zhiyong Lu from National Institue of Health. While we ended up using different implementation of normalizer in our system, their contributions during our earlier attempts in the project are not to understate.

Research wouldn't happen without grants and patrons. Therefore I would like to thank grant organizations that have contributed financial supports to the lab and its extension. I am full aware that without sufficient infrastructure and human capital support endowed by several grants, this project would be impossible to kick start and finish.

Finally I would also thank all Rostlab members and its extensions, without whom this work would all but possible.

Contents

List of Figures

List of Tables

Abbreviations

BLAST	**B**asic **A**lignment **S**earch **T**ools
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
MEDLINE	**Med**ical **Li**terature Analysis Retrieval Onli**ne**
NER	**N**amed **E**ntity **R**ecognition
UniProt	**Un**iversal **Prot**ein Resources

Chapter 1

Introduction

1.1 An Easier Biomedical Research

We'll try to present the main idea of this project in following story. Imagine you in the position as a biomedical researcher, are currently investigating some unknown enzymes that somehow were over-expressed in a patient with medical conditions. Upon some more investigating, you managed to get the sequence of several proteins. Without prior knowledge of the proteins, you would naturally BLAST the sequences and wait a little while while the BLAST is searching the sequence against your local database or some online service. Upon the results were coming, you would naturally want to check the resulting proteins one by one, at least the best matching ones. For each protein, you would want to search for articles that have dealt with this protein before.

Imagine that, instead of having to go through blasting the sequence manually and searching for articles one by one, you could just put in a sequence in a website, wait for a while and get the site returns a list of articles that mention the proteins with similar or exact sequence to the one you have. Not only you would save time and resource during the parts that were handled by website itself, you as a researcher could focus more on the substantial part of the research – that is, finding as much essential information about the unknown protein in as little overhead as possible. Therefore, we created a web service that realizes this. In the service, user would only have to put in the sequence of unknown protein, press the search button and receive at the end a list of articles that mention proteins with identical or similar sequence to queried proteins.

With this small contribution, we hope not only to bridge the gap between sequence and knowledge discovery in biomedical research, but also give researcher more flexibility and insights in their literature research. With also ongoing feature extensions and updates, we would also hope that the service would serve more researchers with more conveniences both in medium and long run.

1.2 Overview of This Thesis

In this thesis, we will describe how we came with the idea of creating PubSeq, how we did that and what we planned in the future regarding our implementation.

In **Chapter 2**, we will discuss how bridging the knowledge gap has been attempted in the past and how our contribution would fit in the bigger picture. We also discuss some of the methods that are relevant in our project. Also, we would look into how our project builds upon existing knowledge and technology.

Chapter 3 introduces the system as a whole. How we organize the sub-components together. We would also skim through the technological side of the projects here, while keeping the reader aware of the bigger picture. We would discuss our rationale behind selecting some of technology stacks that we used. All the while,we would also show some the visual examples from our component here. By the end of the chapter it is hoped that the reader would understand how each single component interacts with others within our system.

Chapter 4 explains in detail our Tagging Pipeline. Here the reader would see how the MEDLINE corpus would be processed, annotated and pushed onto Solr Index.

Chapter 5 explains our storage technology, the Solr index. Besides the technology itself, we would explain how we configure and structure our index that would fit the resulting data from Tagging Pipeline.

Chapter 6 investigates our web server component. We would first present the technological stacks that were used in this component and other specifications. We would then present the program from the perspective of end user. We would show how convenient would that be for a researcher to use our application, which would make our case for value proposition of PubSeq search engine.

Chapter 7 covers quantitative measurement of the quality of our website. We would focus mostly on how our system performs, especially with regards to the sensitivity and specificity. We would focus mostly on the the quality of protein tagging within our data (see Chapter 5 for detail). We would also muse on how our system would have an edge over similar UniProt ID-based abstract search service provided by UniProt [1] [2] [3].

Chapter 8 covers our conclusion of the system so far. There we presented our own ideas on how the website could and would be improved. we would again reiterate the the merits of using the PubSeq as the search engine for abstracts based on protein sequence.

Chapter 2

Background

This chapter introduces the concepts and techniques that are relevant throughout this thesis. First, the concept of similarity search, especially the two software suite FASTA and BLAST would open our chapter. And then, we would introduce various contemporary concepts in bioinformatics and bioinformatics-related infrastructure such as UniProt and MEDLINE. Additionally, we would introduce the concept of named entity recognition (NER) within the field of Natural Language Processing and how it would be relevant for us. Finally we would see how our project relates to previous works in similar topics and how it would improve, provide alternative or give additional insight to them.

2.1 FASTA and BLAST

As the title of this thesis already conveyed, the main idea of this project is to bridge the accessibility and knowledge gap between sequence and the main source of knowledge and reference of previous discoveries – a vast corpora of publications in natural sciences – through a modern search engine. Given a sequence of amino acids, it would be impossible for a human to directly identify directly the protein, let alone the characteristics and the functions and the characteristics of the protein.

Several attempts on bridging one component of the gap, specifically between sequence and other known sequences, was done in eighties and earlier nineties. In 1981, Smith and Walterman published the algorithm computing complete local sequence alignment, which was further improved by Gotoh in 1982 [4] and Altschul (Altschul and Erickson, 1986 [5]). This was however deemed too slow, especially if used for the purpose of one-against-all search, which was heavily (and still is) used for sequence-based knowledge discovery in biomedical research.

In 1985, Lipman and Pearson published the first paper mentioning the DNA and protein sequence alignment program FASTA [6]. During the first publication, FASTA was designed and intended to search for similar protein sequences. It takes a sequence of amino acids and searches against entries within a corresponding database by using local sequence alignment to find similar sequences. In general, FASTA takes four steps in computing three scores that characterize sequence similarity [7]:

1. Finding identify regions with high density of sequence identities and pair identities between two sequences. FASTA achieved a fast computation in this step by using a look up table, a map that describes for each character where it appears within sequence. In conjunction with the lookup table, FASTA also uses the diagonal method to find all regions of similarity between the two sequences, counting matches and penalizing for intervening mismatches. This diagonal could be visually seen in two sequence alignment as series of matches ('dots') in match matrix between two sequences.

2. Rescanning of the 10 regions with highest sequence identities using PAM250 matrix. PAM250 matrix refers to assumed point accepted mutation (PAM) matrix after 250 mutations, which is basically the 250-th power of initial PAM matrix. The probability of each entry within PAM matrix was acquired from analysis of phylogenetic trees (Dayhoff, 1978 [8]).

3. Annealing of both ends of alignment and calculating similarity score is the sum of the joined initial regions minus a penalty (usually 20) for each gap [7].

4. Construction of optimal alignment using Needleman-Wunsch Algorthm [9] on the best matching region. The program would then return the similarity score of this alignment along with the best score from step 2 and 3.

In 1988, Pearson and Lipman improved the software by adding support and improvement, among others, for nucleic acid similarity search, translated nucleic acid search [10]. This allowed researchers to do trans-domain search between nucleic and amino acids.

Further down the road, in 1990, Altschul et al. published the Basic Alignment Research Too [11], better known in its acronym as BLAST. The algorithm, like FASTA, is based on heuristics search and is structured in similar manner to BLAST. BLAST takes a sequence to search for and a sequence or a set of sequences to search against. In modern usage, the set of sequences is provided by some database. The algorithm would then run in following main steps [12]:

1. Removal of low complexity regions or sequence repeats from query sequence. Low complexity refers to sequence with few elements.

2. Creation of k-gram sequences from query sequence.

3. For each word from step 2, listing of possible matching words and selection of high scoring words. Matching words are the all possible combinations of words with same length as the k-gram word. For each possible word a score is calculated, which is based on substitution matrix. The best scoring words are then passed onto next step. This differs from FASTA, which focuses more on common words in database.

4. Organization of remaining high scoring words into efficient search three. Both step 3 and 4 would be repeated for each word from step 2.

5. Scanning of database for exact matches with remaining high-scoring words.

6. Extension of database match to high-scoring-segment pair (HSP). This is done by annealing both ends of match until the matching score begins to decrease.

7. Listing of all HSPs that are significant enough.

8. Evaluation of statistical significance of the HSPs. BLAST models statistical significance using Gumbel extreme value distribution [13], in which the probability of observing score S higher than equal to x is defined as

   $$P(S \leq x) = 1 - exp(-e^{-\lambda(x-\mu)})$$

   with

   $$\mu = log(Km'n')/t$$

   The parameters μ and K are fitted from the distribution of results from high scoring pairs. m' and n' are effective length of the query and database sequences.

9. Make two or more HSP regions into one alignment. In a given hit sequence from database, the algorithm would attempt merging the regions into one had the score of combined region is larger than individual score.

10. Computation of sequence alignments using Smith-Walterman Algorithm [14].

Nowadays, both FASTA and BLAST were distributed not only locally but also online by various providers such as National Center of Biotechnology Information (NCBI) [1] and European Bioinformatics Institute (EBI) [2] [3].

2.2 UniProt

UniProt (*Universal Protein Resource* [1]) is platform containing various high quality databases that are essential in bioinformatics research. A joint venture between European Bioinformatics Institute (EBI), Swiss Bioinformatics Institute (SBI) and Protein Information Resources (PIR), it provide four main sets of database:

- **UniProt KnowledgeBase (UniProtKB)**, containing protein database. Our database of interest in this system, there are two main constituents of UniProtKB: the manually annotated, reviewed UniProtKB/SwissProt and automatically entried, unreviewed UniProtKB/TrEMBL. Currently, there are 549,008 reviewed and 50,011,027 unreviewed protein sequences within the UniProtkB environment[4]. We consider the current ever widening over-representation of unreviewed proteins within the KnowledgeBase be a potential challange in our project going forward.

- **UniProt Archieve (UniParc)**, a comprehensive and non-redundant database containing all protein sequences from publicly available protein sequence database[15]. It achieved redundancy by searching for each sequence only once. Multiple matching sequences from various databases would then be merged into one UniParc entry.

[1] `http://blast.ncbi.nlm.nih.gov/Blast.cgi`, accessed 8/19/2015
[2] `http://www.ebi.ac.uk/Tools/sss/wublast/`, accessed 8/19/2015
[3] `http://www.ebi.ac.uk/Tools/sss/fasta/`, accessed 8/19/2015
[4] `http://www.uniprot.org/`, accessed 8/19/2015

- **UniProt Reference Cluster (UniRef)**, a set of three databases of clustered sets of proteins. The databases house clusters sets of proteins with 50% (UniRef50), 80% (UniRef80) and 90% (UniRef90) sequence similarity [16].
- **UniProt Metagenomics and Environmental Data (UniMES)**, a repository for metagenomics and metagenomics sequences. UniMES was initially developed to address issues arising from sequences that are obtained from non-cultured or currently unknown organisms, which are sometimes used in metagenomics studies [1]. The data from UniMES is included in UniParc but not UniProtKB.

Each entry within UniProt main databases possesses a unique identifier. For UniProtKB, every entry within the database is associated with both UniProtKB accession number (UniProtKB-AC) and UniProtKB identifier (UniProtKB-ID). In this project, we use not only UniProtKB-ID as the identifier of choice when it comes to UniProt, but also as our main project identifier protein names. That is, a protein is always identified with UniProtKB-ID within our search engine, each annotation of mentions in text would be represented as UniPritKB-ID and our BLAST search will also return UniProtKB-ID. Consequently, this limits our search and indexing space within the scope that is already defined by UniProtKB. However, considering proteins that are of interest to the scientist are almost always already indexed by UniProtKB[15], we don't really see this issue as potential problem.

2.3 MEDLINE

MEDLINE contains journal citations and abstracts for biomedical literature around the world [17]. It is maintained by U.S. National Library of Medicine and currently houses over 24 millions of journal abstracts and the numbers keep growing along the new publications, which currently clocks at 7 per cent growth per year, which corresponds to over one million new articles in the last years [18]. In MEDLINE environment alone, it adds between 2,000 - 4,000 references each day [17]. Last year alone, it added 750,000 new references into its repository [17]. The subject of journals that are indexed by MEDLINE ranges from life sciences, behavioral sciences, chemical science and bioengineering. While the overwhelming majority of the references indexed are journal articles, there are several newspaper and magazine entries that are deemed useful as reference in biomedical research. Ninety five percent of the articles are written in English and 84% have English abstracts written by the authors of the articles.

For most part, a researcher could access MEDLINE through PubMed [5] [19]. This already covers about about 90% of its indexed abstracts. For complete access, it offers leasing mechanism. It distributes daily the new references in XML format via FTP protocol. We use this leasing mechanism in our project for intial references collection and daily updates.

2.4 Natural Language Processing

The rapid development in sequence similarity search coupled with explosion of genome-wide sequencing, which was even more augmented by the advent of post-Sanger and – recently – New Generation Sequencing (NGS), means that the problem of identifying sequence is more or less explained. There is however one part that is missing from our picture: how to get the information on how the sequence was mentioned in previous publications?

Come Natural Language Processing (NLP). Natural Language Processing is a interdisciplinary field that deals with the interaction between computer and human languages (hence the natural language). The aspects of natural languages such as named entities recognition (NER) [20], morphological segmentation [21], speech recognition and analysis [22] fall into the auspice of natural language processing. The methods used in natural language processing is mostly statistical-based [23] and some of the methods have been known to be used in other fields such as Conditional Random Fields [24] and its special case Hidden Markov Chain, which is one of the more commonly used methods in bioinformatics (e.g. Salzberg, et al. [25] and Burge, et al. [26]).

In this thesis we would make use of one aspect of natural language processing: named entity recognition (NER).

2.4.1 Named Entity Recognition

The term named entity recognition was first coined at the Sixth Message Understanding Conference (MUC-6) in 1996 [20] [27]. The problem statement of named entity recognition roughly goes as follow:

[5] http://www.ncbi.nlm.nih.gov/pubmed, accessed 19/08/2015

Given an input text, locate and classify elements of text according to defined categories of entity (a Named Entity)

The pre-defined set of categories range from unique identifier such as name and location to expression of times and numeric values such as date and percent expression [20]. To take an example, consider the first paragraph of following recent article from the Wall Street Journal [6]:

> Uber Valued at More Than $50 Billion
>
> **Ride-sharing app, which just closed a funding round, reaches mark faster than Facebook**
>
> Uber Technologies Inc. has completed a new round of funding that values the five-year-old ride-hailing company at close to $51 billion, according to people familiar with the matter, equaling Facebook Inc.'s record for a private, venture-backed startup.

A named entity recognition program trained for company names and monetary values would identify and tag following annotations from the text:

> **Uber** `(company)` Valued at More Than **$50 Billion** `(monetary_value)`
>
> **Ride-sharing app, which just closed a funding round, reaches mark faster than Facebook** `(company)`
>
> **Uber Technologies Inc.** has completed a new round of funding that values the five-year-old ride-hailing company at close to **$51 billion** `(monetary_value)`, according to people familiar with the matter, equaling **Facebook Inc.** `(company)`'s record for a private, venture-backed startup.

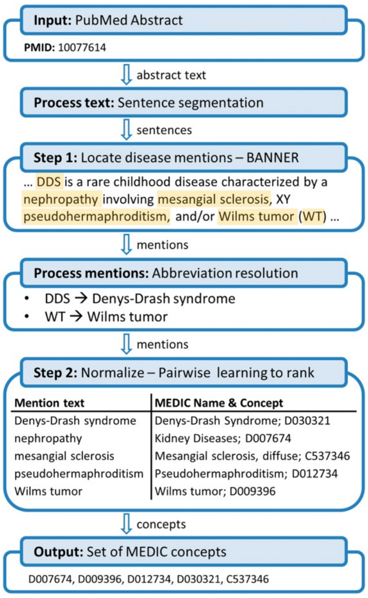
Here we see how various formats of company names could be tagged in this hypothetical case. Indeed, a good named-entity recognition tagger should be able to perform exactly this kind of task.

There are various ways of implementing Named Entity Recognition (NER), all ranging from supervised, semi-supervised to unsupervised learning [20]. Nowadays, most NER

[6] http://www.wsj.com/articles/uber-valued-at-more-than-50-billion-1438367457, accessed 8/19/2015

relies on supervised learning methods [20] such as Hidden Markov Model (HMM) [28], Decission Trees [29], Maximum Entropy Model [30], Support Vector Machines (SVM) [31] and Conditional Random Fields (CRF)[32], which is a discriminative generalization of HMM[33].

In this project, we attempted on two distinct NER implementations: **DNorm**in the earlier phase and non-publicly available NER tagger given to us by Lars Juhl Jensen at University of Copenhagen (which would for our convenience would be called **Lars Tagger** or **PubSeq NER Tagger** from now on) later on. DNorm (Leaman and Lu, 2013 [34]) is a NER designed to tag and normalize disease names within PubMed abstract. Given a PubMed abstract text, it tags the disease entities within text and normalize said entities into standardized form (in this case, MEDIC concepts [35]). The tagging component of DNorm is based on BANNER, which in turn is a CRF-based NER tagger [36]. To normalize the entities, the program utilizes pairwise approach of Learning to Rank [37] (see, for example, Joachims 2002 [38], which is used to rank the results of web search). The idea of ranking as a method of normalization could indeed be explained analogously in light of search engine itself, given a tagged string s, a set of standardized entity names N and a scoring function $M(s, n) \in R$, we would like to find out to which standardized name this string maps best to.


Input: PubMed Abstract
PMID: 10077614
abstract text
Process text: Sentence segmentation
sentences
Step 1: Locate disease mentions – BANNER
... DDS is a rare childhood disease characterized by a nephropathy involving mesangial sclerosis, XY pseudohermaphroditism, and/or Wilms tumor (WT) ...
mentions
Process mentions: Abbreviation resolution
• DDS → Denys-Drash syndrome
• WT → Wilms tumor
mentions
Step 2: Normalize – Pairwise learning to rank
concepts
Output: Set of MEDIC concepts
D007674, D009396, D012734, D030321, C537346

Mention text	MEDIC Name & Concept
Denys-Drash syndrome	Denys-Drash Syndrome; D030321
nephropathy	Kidney Diseases; D007674
mesangial sclerosis	Mesangial sclerosis, diffuse; C537346
pseudohermaphroditism	Pseudohermaphroditism; D012734
Wilms tumor	Wilms tumor; D009396

Figure 2.1: Overview of DNorm pipeline. The figure was taken from Leaman, Dogan and Lu, 2013 [34].

The reason why we initially resorted to DNorm is because of its good performance with regard to precision, recall and F-measure [34]. Also, the underlying tagging mechanism BANNER appeared to perform well in BioCreative benchmarks for tagging various biological entities, most notably gene names [39] [36]. Trying to extend DNorm to support gene names recognition *and normalization*, however, proved to be very difficult. We figured out that there was no publicly available data set that we could use to train our model for gene normalization. Secondly, since the DNorm uses matrix to represents both the normalization and similarity model [34], creating matrix models for the whole set of proteins would have the program create a matrix with millions of row – which wouldn't scale in normal cluster environment. Also, the fact that DNorm was optimized for disease names means that there is no guarantee that the resulting benchmark for protein names would be as good as disease tagging and normalization benchmark.

Later on, we switched to Lars Tagger. Unlike NER methods that are already mentioned in this chapter, our current NER Tagger relies on mapping between normalized entity names and their corresponding alternative names to annotate and normalize named entities within in the text. It checks whether a part of text closely resembles one or more elements in the mapping. It would then calculate the probability of the part of text being the instance of one of the entities (see more on Chapter 3). While there is no publication specifically dedicated for gene annotations, the underlying engine was used to tag and annotate taxonomic names in PubMed text (Pacifilis, et al., 2013 [40]). Moreover, we used part of STRING database (Szklarczyk, et al., 2011 [41]) as the dictionary between normalized entity names and its variation of names that would be used to identify named entities in text and at the same time normalize them.

2.5 Previous Works

There are several publications and tools that attempt on similar goals as we do that have not been reviewed in previous section. While we are not aware of other project that realized end-to-end article search based on sequence, there are several (very successful) attempts on doing parts of it.

In the realm of **gene/protein names tagging and/or normalization**, there are various programs that are known to tag and normalize well such as GNAT (Hackenberg, et al. 2011 [42]), ABNER (Burr, 2015 [43]) and LINNEAUS (Gerner, et al. 2010 [44]).

The latter was originally designed to detect and annotate taxonomy names within text but is now ”intended for general-purpose dictionary matching software”[7].

For searching for protein mentions within PubMed corpus we are for example aware of. Given a UniProtKB-ID, a user can search for the list of articles mentioning the protein. User could, for example, search articles that mention human tumor protein P53 (P53_HUMAN). We are unfortunately not aware how it was done (manually annotated or done using NLP methods) or whether it covers only parts of article that are visible within PubMed corpus or the whole article nor there exists any formal documentation/publication regarding the system.

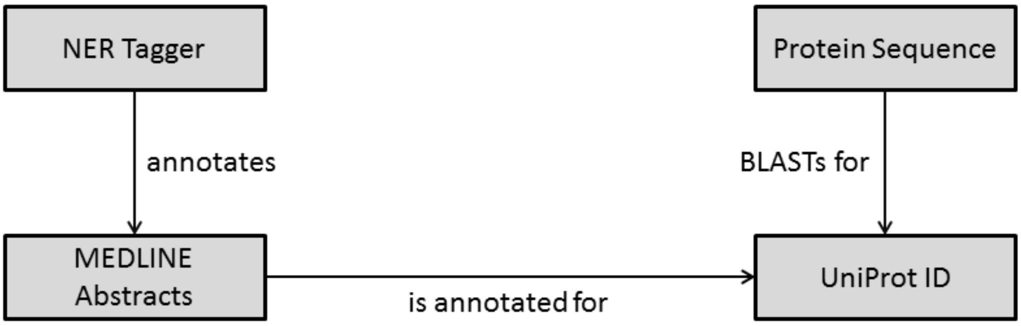


Figure 2.2: Schematic interpretation on how the concepts of BLAST, UniProt, MEDLINE and NER are interrelated.

[7] http://linnaeus.sourceforge.net/, accessed 8/19/2015

Chapter 3

Organizations and Components

This chapter enumerates the components constituting the PubSeq system and explains how each in principle functions. There would also be technicalities of the programs and rationales on how each of the component is used and implemented.

3.1 Introduction

In the most general term, there are three main components that build up the PubSeq environment. The three components could be represented as questions:

- How to *create* the data?
- How the data is going to be *stored*?
- How the user is going to *retreive* the data?

We *create* the data in which we process the raw data from our source into indexable entry data. We then *store* the data in scalable manner for the user to use. Finally, we will facilitate how a user could *retreive* our data.

We formalize this concept further by creating three main components with each represents the answer of the question before:

- **PubSeq Tagging Pipeline** (**Tagging Pipeline** for short) addresses the first question. In this pipeline we would process the data from its main source, the

MEDLINE corpus which is updated daily as XML file, onto indexable input file containing list of UniProt annotations in the abstracts, among others.

- **PubSeq Solr Index** (**Solr Index** for short) addresses storage issue. All processed data would then in an open source enterprise search platform, Solr [45].
- **PubSeq Web Server** (**Solr Server**) addresses the issue of content delivery to end user. Here we would explain the program in more technical manner. We would also convey how the program would see in the perspective of end user – that is how the program runs as user proceeds on using the search engine, later in Chapter 4.

3.2 PubSeq Persistent Components

To understand how the components were structured, I think it is better for the reader to first get to know how the persistent components, that is, the components that are running around-the-clock interact:

Here we can see three main components within the interaction environment: Solr Index, PubSeq Server, PubSeq web page and RostLab Cluster (SunGrid Engine).

- **PubSeq Web Page** is rendered by PubSeq server upon GET request on PubSeq home path and communicates through HTTP protocols to PubSeq server. There are currently POST and GET methods that are launched by this page onto the server. Beside the server, the PubSeq web server, the web page also presents some links to outside world, most notably to PubMed web interface [19]. The linkage to PubMed web interface was embeded in each of the results of the query within PubSeq (see Chapter 4 for details)
- **PubSeq Web Server** is server component of our system. It handles HTTP requests addressed to the web path and renders PubSeq web page. Internally it processes both queries and results from both web page and server reply. The server also communicates to RostLab's clusters for submitting BLAST query for a given sequence.
- **PubSeq Solr Index** contains the whole indexing of MEDLINE and proteins mentions within article. While the main mechanism of updating the index will

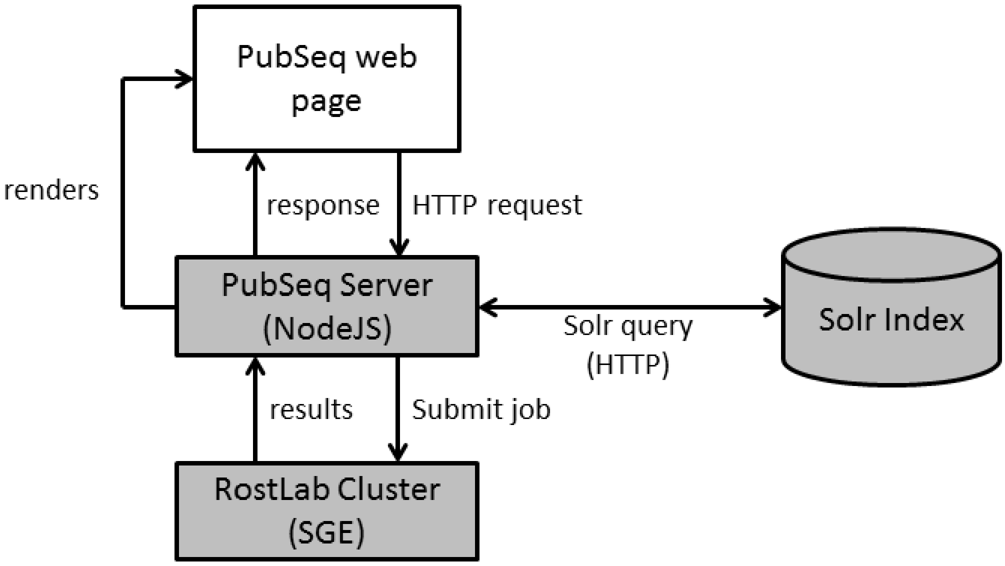


FIGURE 3.1: An overview of how the Persistent Components – the components that are running around the clock – interact with each other. Here we can see two of three main components explained before: Solr Index, Solr Server (and its rendered web app instance, PubSeq Web Page) which is also connected with RostLab Cluster via SunGrid Engine (SGE)

be explained thoroughly in later chapter, it is important to know for reader that the index only communicates via HTTP [45]. As thus, user can check on the machine that the server runs on, the sample content of the Solr server by doing curl followed by the query. For more details see dedicated chapters bellow.

- We utilize **RostLab Cluster** for BLASTing input sequence given by the user.

3.3 Web Interaction

We model our web interaction in following time-dependent sequence diagram [46] on Figure 3.3.

Here we see how four main persistent components of PubSeq interact. First, the user opens the PubSeq page. This will spawn an instance of PubSeq web page. Upon inserting some sequence and pressing query button, the web page would then submit

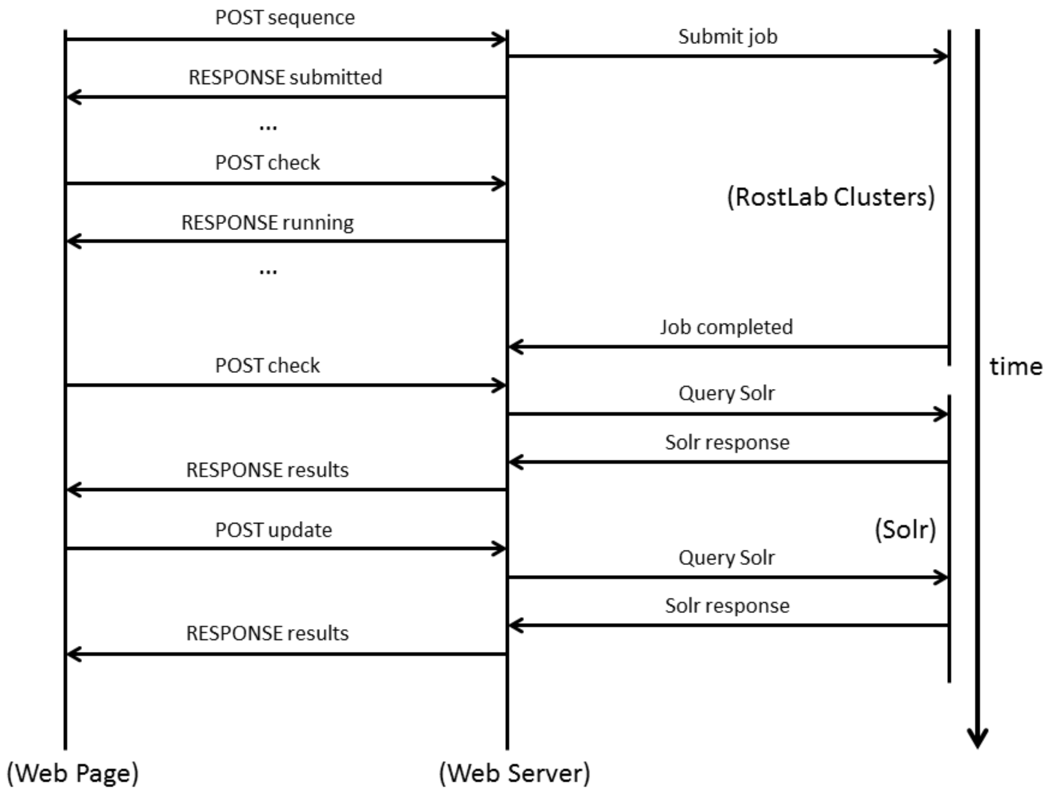


Figure 3.2: Time-dependent sequence diagram of PubSeq web interaction. The y-axis represents the time (from top to bottom) and each of the columns in on the x-axis represents components that interact with each others. Note that the in the third column, there are two entities that interact with Web Server in distinct time spans.

the initial HTTP request [47] onto the server (first **POST sequence**). Our Node.js server [48] would then handle the query and create a script and input file containing aforementioned sequence. It would then submit the query onto RostLab cluster through qsub (**Submit job**) [49] and return first response containing message informing the web client that the job has been submitted (**RESPONSE submitted**). The web client would then re-check the server once in a while (ca. 10 seconds) to find out whether the result of the query has been created (**POST check** and **RESPONSE running**). Once the job has been completed the results would be saved in a pre-defined location using pre-defined names (see later chapter for more details). During the next iteration of checking routine from web app, if the output is already there, the request handler would then parse the output file and prepare a Solr query that would be used for the sequence. This query would then be submitted onto Solr client (**Query Solr**). The query response would then be forwarded to web page in the RESPONSE massage (**RESPONSE results**) and the results would be shown to the user. Had the user have to update the results (mostly by moving between result pages), an update POST would be initiated (**POST sequence**). This update POST contains prepared statement that doesn't require another BLAST, and thus would be directed toward Solr index directly (second **Query Solr**). Just like initial Solr response, the resulting query would be returned to user. As long as user doesn't leave the page or query new sequence, this iteration could be done as many times as possible.

3.4 Tagging Pipeline

Tagging pipeline refers to the process of creating and updating the Solr index that would be used for the server to search for articles containing protein mentions. In this pipeline, a set of documents from MEDLINE database would sequentially processed and finally be pushed into our index. The whole process is done in periodic/one-off basis. The schematic representation with each single step obscured can be seen in Figure 3.3.

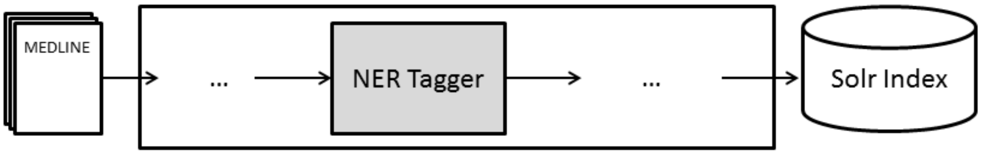


FIGURE 3.3: Schematic diagram representing the broad process of Solr Tagging Pipeline. On the left side the input files, MEDLINE abstracts in XML formats, would be processed sequentially until the data is ready to be pushed onto the Solr index. We deliberately highlighted NER Tagging from within the pipeline to emphasize its importance in our pipeline.

Here we see that the process consists of smaller processes. As already said, there could be two way of running this process: one-off and periodic update. While one-off Tagging Pipeline could be done on RostLab's **jobtest/jobtest2** or similar physical servers within RostLab infrastructure, the periodic update process should be only done via SunGrid Engine call scheduled in crontab [50] (more details later).

We deliberately showed the NER Tagger process within this Pipeline to reader to emphasize its importance. Lars Juhl Jensen was kind enough to give us his program that we use to run on our pipeline. There would be more details on both our NER Tagger and Tagging Pipeline in later chapter(s) but for now we would focus on big picture detail.

Chapter 4

PubSeq tagging Pipeline

4.1 Introduction

In this chapter, we would discuss various steps belonging to Tagging Pipeline. Following main steps belong to the Tagging Pipeline:

1. **Formatting** downloaded MEDLINE abstract into input files that are compliant with Lars' NER Tagger. (1)
2. **Named Entity tagging** done by Lars NER Tagger. (2)
3. **Post-processing** of the results and **preparation** for the entry into Solr index. (3)
4. **Updating** of results onto Solr Index. (4)

The processes are then further divided into several single programs:

- `XMLAbstractsFormatter.java`
- `tagcorpus.cxx`
- `AnnotationBackmapper.java`
- `Annotater.java`
- `StatisticsUtils.java`

- `IndexerNew.java`
- `SolrUpdater.java`

The division of the whole pipeline into smaller tasks are reasoned through following arguments:

- The NER Tagger 2 was developed in C++ while we would implement the rest of pipeline (1, 3 and 4) in Java. This means that pre- and post-tagging procedures would have to be implemented separately. Also since the NER Tagger wasn't written by us and therefore support would be very lacking, we would rather leave the NER Tagger as it is.
- Related to previous argument: Solr is implemented in Java and its most comprehensive API (written by the developers of the Solr themselves) is written in Java [51]. Therefore using Java in 4 would be almost necessary for various convenience reasons.
- The pipeline generally is very memory intensive. During the process, several maps that each would gulp easily teens of gigabyte of memory are utilized. Therefore each step that utilizes such huge maps are all separated into one single routine.
- Dividing the pipeline into smaller components would make it easy to debug, since each routine easily takes several minutes if not hours to run. However this also makes expanding features within the Tagging Pipeline more difficult.

Besides processes mentioned above, there is one process that doesn't exactly belong to Tagging Pipeline but is closely coupled and synchronized with it: downloading and storing of MEDLINE abstracts. Both download and maintenance of MEDLINE corpus and the Tagging pipeline would be covered in following sub-chapters.

4.2 MEDLINE Abstracts

Downloaded MEDLINE abstracts could be found at /mnt/project/rost_db/medline. There are two sub directories found in the path that are relevant to PubSeq: basline and update. Baseline contains all MEDLINE abstracts that were downloaded up until 24 Jan 2015. Update contains all MEDLINE abstracts that were downloaded afterwards. The MEDLINE abstracts are stored in XML files. The XML format for MEDLINE abstracts currently follow the definition set by NLM (Document Type Definitions or DTD for MEDLINE/PubMed, see here or download here). Please refer to the data definition for further property of the corpus.

It had been agreed upon that the MEDLINE will be updated every day at 8 am server time.

Chapter 5

PubSeq Solr Index

Chapter 6

PubSeq Web Service

Chapter 7

Validations and Analyses

Chapter 8

Conclusion and Outlook

Appendix A

PubSeq Paths and Source URLs

A.1 Source Codes

There are several source codes that are relevant to our project:

- pubseq-crawler contains the tagging ('crawler') pipeline of our project. It also contains several scripts that are used in this project (most notably `annotate_new.sh` and `maintenance.sh`.
- pubseq-frontend contains the node.js implementation of our project.

A.2 Project Location

Generally the project will be located at `/mnt/project/pubseq/` within Rostlab server. The directory is further divided into following categories:

```
- /mnt/project/pubseq
|- index_input_docs
|- log
|- named_entity_tagger
|- programdata
|- rundata
|- solr_index
|- scripts
|- stats
```

The overview of each of directories is as follow:

- `index_input_docs` contains the files that would be indexed onto Solr. In other words, this directory contains all files that were created during Tagging Pipeline. During the last step of Tagging Pipelines, the program `SolrUpdater.jar` would point at this directory and index files that match certain pattern of file name.
- `log` contains logs from Tagging Pipeline.
- `named_entity_tagger` contains the **Lars Tagger component** of the Tagging Pipeline.
- `programdata` contains several program-related tab-separated files that would be used and/or produced during the Tagging Pipeline. Note that program-related data is different from run-related data (which are located at `rundata` in which program-related data remain mostly constant across all runs while run-related data were created during one of the steps within the Tagging Pipeline.
- `rundata` contains interim results from Tagging Pipeline and other Tagging-related data. Since Tagging Pipelines consist of multiple programs with each taking input and writing output, it is inevitable that there would be several interim values. The Tagging Pipeline is designed to write standardized in-between values. While the files would not be removed upon completion of one Tagging Pipeline run, they would be overwritten during the next run. The interim values wouldn't be backed up in some consistent environment.
- `solr_index` contains the Solr directory for the project. For readers who are not familiar with Solr, assuming it as ”NoSQL Database” would be sufficient. Solr index is self-contained in the way that the only thing that is needed for it to run properly is its own directory (and JDK).
- `scripts` contain scripts that would be run for various purposes. The most essentials of all scripts are the `annotate_new.sh` and `maintenance.sh`. `annotate_new.sh` wraps the whole Tagging Pipeline process and calls sequentially each component within the pipeline. `maintenance.sh` contains the Maintenance Pipeline and

like `annotate_new.sh` calls each component within Maintenance Pipeline sequentially. Both Tagging and Maintenance Pipelines would be further described in later chapters.

- `stats` is deprecated or might not be necessary for the whole process. It contains some descriptive statistics from the last run of Tagging Pipeline. For each Tagging Pipeline, some set of statistic files were created that more or less describe the nature of the run.

Bibliography

[1] UniProt Consortium et al. The universal protein resource (uniprot). *Nucleic acids research*, 36(suppl 1):D190–D195, 2008.

[2] UniProt Consortium et al. Ongoing and future developments at the universal protein resource. *Nucleic acids research*, 39(suppl 1):D214–D219, 2011.

[3] The UniProt Consortium. Uniprot citation mapping, 2015. URL http://www.uniprot.org/citationmapping/.

[4] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[5] Stephen F Altschul and Bruce W Erickson. Optimal sequence alignment using affine gap costs. *Bulletin of mathematical biology*, 48(5-6):603–616, 1986.

[6] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[7] William R Pearson. Rapid and sensitive sequence comparison with fastp and fasta. *Methods in enzymology*, 183:63–98, 1990.

[8] Margaret O Dayhoff and Robert M Schwartz. A model of evolutionary change in proteins. In *In Atlas of protein sequence and structure*. Citeseer, 1978.

[9] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[10] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[11] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3): 403–410, 1990.

[12] David W Mount and David W Mount. *Bioinformatics: sequence and genome analysis*, volume 2. Cold spring harbor laboratory press New York:, 2001.

[13] Emil Julius Gumbel and Julius Lieblein. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office Washington, 1954.

[14] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[15] Rasko Leinonen, Federico Garcia Diez, David Binns, Wolfgang Fleischmann, Rodrigo Lopez, and Rolf Apweiler. Uniprot archive. *Bioinformatics*, 2004.

[16] Baris E Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, and Cathy H Wu. Uniref: comprehensive and non-redundant uniprot reference clusters. *Bioinformatics*, 23(10):1282–1288, 2007.

[17] U.S. National Library of Medicine. Medline, 2015. URL http://www.nlm.nih.gov/bsd/pmresources.html.

[18] Peder Larsen and Markus Von Ins. The rate of growth in scientific publication and the decline in coverage provided by science citation index. *Scientometrics*, 84 (3):575–603, 2010.

[19] U.S. National Library of Medicine. Pubmed web interface, 2015. URL http://www.ncbi.nlm.nih.gov/pubmed.

[20] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[21] Fernand Meyer and Serge Beucher. Morphological segmentation. *Journal of visual communication and image representation*, 1(1):21–46, 1990.

[22] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. 1993.

[23] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[24] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

[25] Steven L Salzberg, Arthur L Delcher, Simon Kasif, and Owen White. Microbial gene identification using interpolated markov models. *Nucleic acids research*, 26 (2):544–548, 1998.

[26] ChristopherB Burge. Modeling dependencies in pre-mrna splicing signals. *New Comprehensive Biochemistry*, 32:129–164, 1998.

[27] Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *COLING*, volume 96, pages 466–471, 1996.

[28] Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201. Association for Computational Linguistics, 1997.

[29] Satoshi Sekine, Ralph Grishman, and Hiroyuki Shinnou. A decision tree method for finding and classifying names in japanese texts. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 1998.

[30] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proc. of the Sixth Workshop on Very Large Corpora*, volume 182, 1998.

[31] Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 8–15. Association for Computational Linguistics, 2003.

[32] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[33] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[34] Robert Leaman, Rezarta Islamaj Doğan, and Zhiyong Lu. Dnorm: disease name normalization with pairwise learning to rank. *Bioinformatics*, page btt474, 2013.

[35] Allan Peter Davis, Thomas C Wiegers, Michael C Rosenstein, and Carolyn J Mattingly. Medic: a practical disease vocabulary used at the comparative toxicogenomics database. *Database*, 2012:bar065, 2012.

[36] Robert Leaman, Graciela Gonzalez, et al. Banner: an executable survey of advances in biomedical named entity recognition. In *Pacific Symposium on Biocomputing*, volume 13, pages 652–663. World Scientific, 2008.

[37] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[38] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[39] Larry Smith, Lorraine K Tanabe, Rie J Ando, Cheng-Ju Kuo, I-Fang Chung, Chun-Nan Hsu, Yu-Shi Lin, Roman Klinger, Christoph M Friedrich, Kuzman Ganchev, et al. Overview of biocreative ii gene mention recognition. *Genome biology*, 9(Suppl 2):S2, 2008.

[40] Evangelos Pafilis, Sune P Frankild, Lucia Fanini, Sarah Faulwetter, Christina Pavloudi, Aikaterini Vasileiadou, Christos Arvanitidis, and Lars Juhl Jensen. The species and organisms resources for fast and accurate identification of taxonomic names in text. *PloS one*, 8(6):e65390, 2013.

[41] Damian Szklarczyk, Andrea Franceschini, Michael Kuhn, Milan Simonovic, Alexander Roth, Pablo Minguez, Tobias Doerks, Manuel Stark, Jean Muller, Peer Bork, et al. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic acids research*, 39(suppl 1): D561–D568, 2011.

[42] Jörg Hakenberg, Martin Gerner, Maximilian Haeussler, Illés Solt, Conrad Plake, Michael Schroeder, Graciela Gonzalez, Goran Nenadic, and Casey M Bergman. The gnat library for local and remote gene mention normalization. *Bioinformatics*, 27(19):2769–2771, 2011.

[43] Burr Settles. Abner: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[44] Martin Gerner, Goran Nenadic, and Casey M Bergman. Linnaeus: a species name identification system for biomedical literature. *BMC bioinformatics*, 11(1): 85, 2010.

[45] David Smiley, Eric Pugh, Kranti Parisa, and Matt Mitchell. *Apache Solr Enterprise Search Server*. Packt Publishing Ltd, 2015.

[46] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[47] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[48] Stefan Tilkov and Steve Vinoski. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, (6):80–83, 2010.

[49] Wolfgang Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE, 2001.

[50] Michael S Keller. Take command: cron: Job scheduler. *Linux Journal*, 1999 (65es):15, 1999.

[51] Trey Grainger, Timothy Potter, and Yonik Seeley. *Solr in action*. Manning, 2014.