SNPMiner: A Domain-Specific Deep Web Mining Tool

Fan Wang* Gagan Agrawal* Ruoming Jin† Helen Piontkivskaδ
*Department of Computer Science and Engineering
Ohio State University, Columbus OH 43210
{wangfa,agrawal}@cse.osu.edu
† Department of Computer Science
Kent State University, Kent OH 44242
{jin}@cs.kent.edu
δ Department of Biological Sciences
Kent State University, Kent OH 44242
{opiontki}@kent.edu

Abstract—In this paper, we propose a novel query-oriented, mediator-based biological data querying tool, SNPMiner. The system searches and queries Single Nucleotide Polymorphisms (SNPs) data from eight widely used web accessible databases. The system provides a domain-specific search utility, which can access and collect data from the deep web. This is a web-based system, so any user can use the system by accessing our server from their own computers. The system includes three important components, which are the web server interface, the dynamic query planner, and the web page parser. The web server interface can provide end users a unified and friendly interface. The dynamic query planner can automatically schedule an efficient query order on all available databases according to user’s query request. The web page parser analyzes the layout of HTML files and extracts desired data from those files. The final results of the query are organized in a tabular format, which can be reviewed by a biological researcher.

I. INTRODUCTION

Recent advance in genomic technology has been greatly impacting the practice of biological and medical research. A large volume of sequencing and structural data is being made available for further analysis. Further, with the fast development of internet and web, most of the biological data and information is now available for web-access. However, efficient and effective use of growing number of such data sources is becoming a critical problem for biological and medical researchers.

Traditionally, biologists can manually query on all available databases and combine information gathered from these heterogeneous data sources. Unfortunately, the sheer volume and rapid growth of biological data makes this process time-consuming, tedious and error-prone. The biggest challenge here is how to effectively integrate all these heterogeneous databases together and query on them to extract relevant information.

In the past few years, there has been a number of efforts focused on the integration of biological data sources. The key challenges that most of these systems need to address are: variety of data types, representational heterogeneity, autonomous and web-based sources, and differing querying capabilities [1]. In terms of integration approaches, the existing research can be largely divided into three categories, which are warehouse integration, mediator-based integration and navigational integration.

One of the newer challenges in biological data integration, which has not been addressed by the above efforts, is associated with the emergence of deep-web data sources. Many of the data sources are stored in online databases, hidden behind the query forms, forming the deep web [2]. As compared to the surface web, where the HTML pages are static and data is stored as document files, deep web data is stored in databases. Dynamic HTML pages are generated only after a user submits a query by filling an online form. Thus, standard search engines like Google are not able to crawl to these web-sites. At the same time, manually submitting online queries to numerous query forms, keeping track of the obtained results, and combining them together is a tedious and error-prone process. Thus, there exists a growing need for a tool that can extract meaningful and relevant information from deep web biology data sources with minimum human intervention.

This paper presents such a tool for deep-web mining. While the underlying techniques are general, the specific implementation is driven by the problem of searching SNP databases.

A. Specific Motivating Problem: Searching SNP Databases

In the effort to explain the genetic contribution to complex diseases such as cancer and heart disease, Single Nucleotide Polymorphisms (SNPs), that designate sites in the genome that has two or more nucleotide variants segregating in a population [3], seem particularly promising because they are usually biallelic and thus easily assayed [4], [5], [6]. Because over seven million Single Nucleotide Polymorphisms (SNPs) have been reported in public databases, it is desirable to develop methods of sifting through this information for finding likely candidates for disease association. Furthermore, information on human SNPs is also useful for studying questions related to human evolutionary history [7] and the role of population genetic processes such as natural selection in shaping the human genome [8], [9], [10].

There are a number of publicly available databases that provide information on SNPs in humans. At the same time, biomedical
and biological researchers want to be able to access information such as the following, which need a search across multiple different web-sites: 1) allelic and genotype frequencies at each SNP site, 2) the number of individuals on which allelic frequency counts are based and the population origin of these individuals, 3) the location of the SNP site in the genome, 4) if the SNP site is located within a gene, whether it is located in an exon, intron, 5’ UTR, or 3’UTR, 5) if the SNP site is located within an exon, whether the SNP causes an amino acid change (a nonsynonymous SNP) or not (a synonymous SNP), 6) if the SNP is nonsynonymous, the amino acid change involved and an estimate of the likelihood of the change having a radical effect on protein structure, 7) if the SNP is nonsynonymous, the amino acids occurring at the corresponding position in the orthologous gene of non-human mammals, 8) if the SNP is located within a gene, the function and interactions of the protein encoded by that gene, and 9) estimation of linkage disequilibrium with other SNP sites.

B. SNPMiner

To summarize the above discussion, the large volume of biological data and the needs of the biologists imply that we should have a tool which can effectively integrate various deep-web data sources, create a unified user friendly interface, make an efficient query plan, spend the minimum amount of time, and extract meaningful information from the deep web databases.

In this paper, we describe a system SNPMiner that we have developed to integrate several widely used SNP databases and extract relevant information from these databases for end users. SNPMiner is a web-based search tool, and it has a convenient user interface. The system integrates eight biological databases, including general purpose databases and specialized databases for SNPs. The system provides a fixed set of candidate terms which can be queried on, which are refered to as the Query Target Terms. Users select a subset of provided Query Target Terms and specify a gene name or SNP rsID as the Query Key Term. Then, a query planner automatically generates a query order based on user’s required terms. From this point of view, we can view the combination of Query Key Term and Query Target Terms as a kind of virtual query language. As a result, our SNPMiner is not a keyword searching tool, which may produce many false positives and negatives. The query procedure is guided by the query plan. When querying each individual database, the system will reformulate the query into a query on the local data source. Finally, all retrieved information will be integrated into a tabular format. Because the underlying techniques for query planning are quite general, SNPMiner also lays the foundation for a more general-purpose deep-web mining tool for biological information.

The rest of the paper is organized as follows. In Section II, we describe the overall architecture of our system and give examples of its use. In the same section, we also describe eight databases integrated by our system. We show the detail of the dynamic query planner in Section III. In Section IV, we evaluate our system. We compare our work with related efforts in Section V and conclude in Section VI.

II. SYSTEM OVERVIEW

In this section, we give an overview of our SNPMiner tool. First, we show the architecture of the system and give a brief introduction to each component of the architecture. Second, we describe the various databases we collect information from. Finally, we give a running example of our system, showing the input interface and output table.

A. System Architecture

Our current SNPMiner system has three main components, the Web Server Interface, the Query Planner, and the Web Page Parser. The Figure 1 shows the architecture of the system. Web Server Interface: SNPMiner is a web-based querying tool. It allows users to access the system from their own computers. We use Apache Tomcat 6.x to support our web server. Using the interface, users can specify the terms they want to be queried on. Those terms include SNP features, such as alleles of a SNP, SNP frequency, SNP related gene information, SNP related protein information and so on. Currently, based on the eight databases we integrated and all SNP information we had listed earlier in Section I-A, we select 40 terms as the query target terms.

Query planner: Biological databases cover a wide range of information and for a single problem, data from several web sources may be required. Furthermore, information gained from one source may be required to query another source. According to the terms user selected, the query planner can schedule a carefully selected query order among all available web databases to guide the query process. In Sections III-A, we describe the query planner in more details.

Web Page Parser: All results of queries are returned in the form of HTML files. Web page parser uses HTML tags to understand the layout of the HTML files and extract relevant data from them.

B. Data Sources

There are a number of publicly available databases that provide information on SNPs in humans. Each of the available databases
for human SNP data includes a subset of the SNP information that we mentioned earlier in Section I-A. The remaining information is obtainable through linkages to other databases.

Our system integrates the following biological databases: dbSNP, Entrez Gene, Entrez Protein, BLAST, SNP500Cancer, SeattleSNPs, SIFT, and BIND. In this section, we briefly introduce the databases that we integrate.

1) dbSNP: The public dbSNP serves as a repository for all reported SNP data and assigns a unique identifying number to each SNP site [11]. It serves as the central point of search into the resource network of National Center for Biotechnology Information (NCBI), because dbSNP links variations (polymorphisms and clinical mutations) to other NCBI resources, such as Entrez Gene, Entrez Protein, and NCBI BLAST [11]. The linked databases are as follows:

Entrez Gene: Entrez Gene is the gene-specific database at NCBI. It maintains information such as nomenclature, chromosomal localization, gene products, protein interactions, and a wealth of links to sequences, homologs, and protein domain content [12].

Entrez Protein: The protein database contains sequence data from the translated coding regions from DNA sequences in GenBank, EMBL, and DDBJ as well as protein sequences submitted to Protein Information Resource (PIR), SWISS-PROT, Protein Research Foundation (PRF), and Protein Data Bank (PDB).

NCBI BLAST: NCBI BLAST uses the Basic Local Alignment Search Tool (BLAST) to find the region of local similarity between sequences.

2) SNP500Cancer Database: SNP500Cancer is a component of the Cancer Genome Anatomy Project (CGAP) of the National Cancer Institute (NCI) and is specifically designed to generate resources for the identification and characterization of genetic variation in important genes related with cancer [13]. The selection of SNPs in this databases is heavily biased towards non-synonymous SNPs, since the focus of our system is the SNPs, especially non-synonymous SNPs which are responsible for cancer diseases, SNP500Cancer database can provide the database with a suitable set of SNPs for each gene that we are interested in.

3) SeattleSNPs: SeattleSNPs is an effort funded as part of the National Heart Lung and Blood Institute’s (NHLBI) Programs for Genomic Applications (PGA). It is focused on identifying, genotyping, and modeling the associations between Single Nucleotide Polymorphisms (SNPs) in candidate genes and pathways that underlie inflammatory responses in humans.

4) SIFT Database: Identifying substitutions that affect the function of proteins and finally cause diseases is an important issue. In order to estimate the likelihood of the amino acid change having a radical effect on protein structure, we use a tool SIFT, which is a sequence homology-based tool that sorts Intolerant From Tolerant (SIFT) amino acid substitutions and predicts whether an amino acid substitution at a particular position in a protein will have a phenotypic effect. Substitutions at each position with probabilities less than a cutoff value is predicted as damaging, otherwise, it is predicted as tolerated [14].

5) BIND Database: The Biomolecular Interaction Network Database (BIND) has been designed to store information about biomolecular interactions, molecular complexes, and pathways in a computer readable form [15]. This database (BIND) can be used to study networks of interactions, to map pathways across taxonomic branches, and to generate information for kinetic simulations [15].

C. User Interface of Our System

We show the input interface of our SNPMiner system in Figure 2.

The input of the system is composed of three parts. The first one is Query Key Term (QKT). For our current system, a QKT can be a gene name or a SNP ID. It specifies the subject of the query. The second part is a set of Query Parameters (QP). These parameters are required by online databases. The third part is Query Target Terms (QTT). Query Target Terms specify what type of information the user wants to know about the Query Key Term. As introduced in Section II-A, our current system provides 40 Query Target Terms. In the example in Figure 2, we can know that the Query Key Term is a gene name, ERCC6. The selected Query Target Terms are SNP_rsID, Alleles, Genotype_Freq, and Alleletype_Freq, and a few others. It means that the user wants all the SNP’s identifier number, alleles, genotype and alleletype frequency information, and encoded protein information located in gene ERCC6. Figure 3 shows a part of the output of this query.

III. SYSTEM IMPLEMENTATION

In this section, we describe the system implementation, especially focusing on the query planner.

A. Dynamic Query Planner

In Section I-A, we mentioned that several pieces of information need to be extracted from various biological databases. A single
database is not capable of providing us all the needed information. Furthermore, information gained from one source may be required to query another source. Based on the above observation, we need to have a query planning strategy, which can provide us an efficient and correct query plan to query all web databases, in response to the user-provided query terms.

We believe that a query planner should have the following features:

**Correctness**: The query planner must be able to select a query plan such that every database in the plan is accessible and all user required data is covered by these databases.

**Efficiency**: There might be multiple candidate query plans, because several databases may contain similar information. However, different query plans may have different number of databases involved. As a result, our goal is to have a query plan with the least number of databases, while still providing all the needed data.

**Robustness**: Some web database may not be accessible at certain time due to hardware problem at the web server. The query planner should be able to detect this and make another query plan when the optimal one is not feasible. Furthermore, if no other feasible query plan is able to provide all the information needed, the query planner should be able to find the one which can cover the most amount of information that is desired by the user.

We have designed and implemented a dynamic query planner based on a production rule system. Our query planner uses a greedy algorithm. In each iteration, it selects the database which can best satisfy user’s query need.

1) **Production Rule System**: Production rule system is a model of computation that has proved particularly important in AI, both for implementing search algorithm and for modeling human problem solving [16]. Production rules are popular and widely used in knowledge representation and inference reasoning. As an example, MYCIN, one of the early expert systems, uses the idea of production rule system [17]. Since our query planning problem needs the knowledge about online databases and can be viewed as a search problem, we believe that production rule system would be an effective solution for our problem.

A production rule system can be represented by three elements, which are the Goal State (GS), the Current State (CS) or Working Memory and the Production Rules (PR). We map our query planning problem into the three elements of a production rule system as follows:

- **Goal State**: The goal state is the subset of the Query Target Terms selected by the user, as we described in Section II-C.
- **Current State (Working Memory)**: The current state is comprised of all the data which has already been extracted. Our query plan is generated step by step, and when one database is added into our query plan, data that can be obtained from this database is considered as stored in the current state. Initially, the current state is simply the Query Key Term specified by the user.
- **Production Rules**: Each online database has one or more underlying query schema. These schemas specify what the input of the online query form of the database is, and what data can be extracted from the database by using the input terms. The production rules of our system are these database schemas.

2) **Production Rule Representation**: One important issue in our strategy is how to represent database query schemas as production rules. Each online database has a query interface. A user can fill out a query form to perform a query on the database. Each query form corresponds to a query schema, which links a set of input parameters with a set of information stored in the database. Each database $D_i$ has a query schema $QS_i$. For simplicity, for the databases with multiple query schemas, we only consider the query schema which is related with our particular task. The production rule $QS_i$ can be represented as a tuple

$$QS_i = (ID, I_i, D_i, O_i, C_i)$$

in which $ID$ is the unique identifier of this rule, $I_i$ is the set of input elements of this rule, $D_i$ is the identifier of the database, $O_i$ is the corresponding set of output elements of the database, and $C_i$ is a set of additional conditions imposed on the elements in $I_i$ in order to use this rule. For example, if we want to query on SIFT database, we have to require that the SNP being queried must be non-synonymous SNP. The production rule for the database SIFT can be represented as:

$$\{9, \{SNPID\}, \{SIFT\}, \{SIFT\_Info\}, NonSyn(SNPID)\}$$

In our implementation, we assign a unique identifier to each database and each Query Target Term. So, in our system, the above query schema will be represented as

$$\{9, \{0\}, \{6\}, \{36\}, NonSyn(0)\}$$

3) **Rule Selection**: A very important topic in production rule system is rule selection, which means which rule should be executed. Our proposed rule selection algorithm uses a greedy algorithm. First, at each step, we scan each rule and put all the
rules which can be fired into a set called candidate rule set. Then, we compute a benefit score for each of the candidate rules and execute the one with the highest benefit score.

A production rule $QS$ can be executed if the corresponding database is available and the input part $I$ is covered by the current state $CS$, i.e., $I \subseteq CS$. For each rule, we first extract the $D$ part from the $QS$ tuple, and send a test message to the database to see whether the database is available. If the database is not available, we just ignore this rule for the current iteration. If a rule passes the availability test and $I \subseteq CS$, we add this rule to the candidate rule set.

The second step is to compute a benefit score for each rule in the candidate rule set. We use data coverage as our metric. Data coverage measures the percentage of required data that can be provided by a particular rule. For a rule, we can compute the data coverage of the rule with respect to the goal state using the following method. Suppose we are given a rule $QS_k$, the goal state $GS$, and we have already selected $k - 1$ rules before, which are $QS_1, QS_2, \ldots, QS_{k-1}$. Now, we want to compute the data coverage of the current rule $QS_k$ with respect to $GS$. We use the following conditional probability to represent the coverage

$$P(QS_k \cap \neg QS_1 \cap \neg QS_2 \cap \ldots \cap \neg QS_{k-1} | GS) = \frac{P(QS_k, \neg QS_1, \neg QS_2, \ldots, \neg QS_{k-1}, GS)}{P(GS)}$$

We use this formula to make sure that we do not consider the data which can be provided by $QS_k$, but has already been extracted by the previously executed rules. In practice, we use the number of Query Target Terms in $GS$ which are also covered by the output element $O_k$ of rule $QS_k$, but have not been extracted by previous rules to estimate $P(QS_k, \neg QS_1, \neg QS_2, \ldots, \neg QS_{k-1}, GS)$, and use the total number of Query Target Terms in $GS$ to estimate $P(GS)$.

Now each candidate rule is assigned a benefit score, and we always select the rule with the highest score to be executed next.

4) State Update and Termination Condition: When we select a rule $QS$ according to our rule selection strategy, we can consider the corresponding database as queried. We add the element in $O$ component of the rule $QS$ into the current state $CS$. Because we do not want to re-visit the same rule (or database) twice, we mark the selected rule as visited. Our query planning algorithm will terminate in either of the following two cases:

1) The current state has covered all the elements in the goal state, which means that all user requested Query Target Terms have been found. We can represent it as $CS \supseteq GS$.

2) Although some terms in the goal state have not been covered by the current state, but no unvisited rules can cover any more elements in the goal state. This means that it is impossible to retrieve all the request terms from the current available databases. This normally occurs when some databases fail the availability test.

---

**Algorithm III.1: Query_Planer(Key_Term, Target_Terms)**

Initialize Current State CS, Goal State GS
Initialize Production Rule PR
Initialize the Query Chain QC to be empty

while ($\exists x \in GS$ and $x \notin CS$) or ($\exists y \in PR$ and $y \notin visitedPR$ and $\exists z \in output(y)$ and $z \in GS$ and $z \notin CS$

- Initialize an empty set CR for candidate rules
- foreach $p \in unvisitedPR$
  - if (available(p))
    - compute benefit score of p, bs(p)
    - add p to CR
  - Select a rule bp from CR with the highest benefit score
  - Add output(bp) to CS
  - Delete bp from unvisitedPR and add it to visitedPR
  - Add the production rule bp to QC

return $(QC)$

Algorithm III.1 shows the overall pseudo-code of our query planning algorithm. In the algorithm, available(p) is a function which tests whether the corresponding database is available. output(p) returns the output elements of a production rule.

**B. Webpage Parsing**

After filling out an online query form on a web database interface, the query result is returned in form of a web page. In order to extract the information we need, we have to parse the HTML pages and find relevant data. In the current system, we parse HTML web pages by using HTML labels and HTML table tags.

**Using Labels:** Some of the biological data in a HTML web page are managed in the following way: Label: data value. For example, in dbSNP web page for SNP rs7412, the allele information is formatted in this way: Alleles: CT. For this type of presentation, we just seek for the expected label, and then extract the data value followed by the label.

**Using Table tags:** Most of the biological data coming from the above web databases is managed in a table format. For such format, we first scan each column using the <TH> tags, which represent table column header, to obtain the label for each column in the table. By parsing this, we can obtain the number of column which contains the data we need. Then, we use the <TR> tags to go over each row of the table. For each row, we search the <TD> labels by using the number of column has already been extracted before to extract the data we need. Frequency information in dbSNP, linkage disequilibrium values in SeattleSNPs, and many other data are all parsed in this way.

One disadvantage of this parsing strategy is that it is likely to be impacted by a change in the web page layout. Therefore, in our future work, we are planning to use an automatic or semi-automatic web page layout learning strategy, which can learn the layout of a web page and extract desired data items.

**IV. PERFORMANCE**

Our performance evaluation has two components. The first component is the evaluation of our dynamic query planner, and
the second component is the evaluation of the entire system performance.

A. Evaluation of Dynamic Query Planner

In evaluating the quality of generated query plans, the main metric we used was the length of the query plan. Of multiple query plans have the same data coverage, we want to use the shorter one. A shorter query plan involves fewer databases, and therefore, we expect query execution to take a shorter time.

We conducted five experiments, each with different Query Target Terms, and recorded the length of the generated plan. We also manually worked out the optimal query plan for each case. We compare the length of the query plan generated by our dynamic query planner with the desired optimal query length. The results are shown in Figure 4.

We show the details of the five experiments in Table I. Experiment A and B are relatively simple query requests, i.e., all user request Query Target Terms can only be provided by one rule (database) and the Query Target Terms provide enough information on rule selection. For these two relatively simple cases, we can see from Figure 4 that the length of the generated query plan is the same as the optimal length. We also examined each rule (database) that our query planner chooses, and we notice that the query plans for case A and B are exactly the same as the optimal one.

Experiments C, D, and E involve more complicated queries. Some terms in the Query Target Terms can be provided by multiple rules (databases), there are some hidden rules in the query chain. For example, we first execute a rule by using the Query Key Term, and then the extracted data from the rule is added to current working memory. Then, we consider all candidate rules, and we find that the data coverage scores of these candidate rules are zero. But this does not mean that these rule should not be fired. What we need is the intelligence in our query planning to select a potentially useful rule, and that rule can retrieve new data to further fit into other rules. Figure 5 shows one example of this case.

E is the hardest query because it requires 3 hidden rules to be determined out by the query planner, followed by the experiment D, which has 2 hidden rules, and the experiment C (1 hidden rule). From Figure 4 we can have see that for the experiments C, D, and E, the generated query length is longer than the optimal length. With the increase of number of hidden rules, the length of the generated query plan increases as well.

But, overall, we can observe that the actual generated query plan is no more than 40% longer than the optimal length, excluding the rare case E.

B. Overall System Performance

We evaluated the performance of the system focusing on the query execution time. The test is carried on the five experiment cases introduced in Section IV-A. We use the Query Key Term ERCC6 (a gene name) for experiments A, D, and E, and the Query Key Term rs2228528 (a SNP located in gene ERCC6) for the experiments B and C. Table II shows the results.

For all experiments, the time used on query planning is around 3.7 seconds. Therefore, we can say that the scalability of the dynamic query planning module is very good. In terms of the total execution time, the percentage time spent on query planning is only 0.17%. This means that our dynamic query planning module does not cause much overhead to the entire system. Furthermore, most of the time in query planning is spent on database availability test. From Table II we can note that availability test nearly accounts for 99% of all time spent on query planning. It is very hard to improve in this respect, because for availability test, we have to wait for the reply message from online databases.

Network connection time is a very high percentage of the total execution time. From the table we can see that nearly 80% of the execution time is spent on establishing connections with online databases. As expected, with increasing number of terms retrieved, larger time is required for in execution, primarily because we need to connect with more databases. On the average, the time cost on extracting one term is about 2 seconds.

C. Limitations of Current Dynamic Query Planner

One issue that we do not currently address is as follows. For different query plans with the same data coverage and the same length, different results may be extracted. This is because some database may have incomplete data on some terms, but another
database has complete data. In other words, a domain user may have a preference for certain query plans over others. For example, for a particular SNP, dbSNP may contain the SNP frequency information from 10 subpopulations, and SNP500Cancer only contains the SNP frequency information from 5 subpopulations. Our query planner may choose to use SNP500Cancer database. So in this case, we say the extracted result of using dbSNP is better than that of using SNP500Cancer. In this case, the user may have preference to dbSNP. In our future work, we will consider such user preference in query planning.

Note, however, that this issue is different from the situation where different query plans might cause different semantics and some query plans may produce incorrect answers for the user request. For the databases we have currently integrated, this problem does not arise.

V. RELATED WORK

We now compare our work with existing work on bioinformatics data integration and deep web mining.

Integration of Biological Data: A number of current systems on biological sources integration is reviewed in [1]. Integration approaches can be largely divided into three types: warehouse integration, mediator-based integration and navigational integration [1]. GUS [18] is a tool using the warehouse approach. It downloads, cleans, integrates and annotates data from multiple external data sources by using a schema called Genomics Uniformed Schema. Some mediator-based integration systems are TAMBIIS, Discovery-Link, BACIIIS, and KIND. TAMBIIS [19] identifies the appropriate sources to satisfy a user query written in a description logic called GRAIL and rewrites the query to a series of source dependent ordered procedures, or query plan. The query plan is generated in CPL [20]. Discovery-Link [21] needs a wrapper to map the information stored by the data source into DiscoveryLink’s relational data model, and then DiscoveryLink server identifies the relevant data sources and develops a query execution plan, which is optimized in a bottom-up manner, considering various database statistics. BACIIIS [22] and KIND [23] both use wrappers, and they combine ontology with their wrappers. SRS [1], [24] uses local indexes to retrieve data and uses its own parsing component, ICARUS, to recognize the links to other sources. Our current system is a mediator-based system. The main difference in our system is that it focuses on SNP related data, and performs dynamic query planning for extracting information from deep-web sources.

Deep Web Mining: Lately, there has been a lot of work on mining useful information from the deep web. Since for structured web-sources, query schemas are discriminative representatives of their sources, much work has been done in this domain. Chang et.al [25] viewed schema matching as correlation mining, and took advantage of the common observation, which is grouping attributes frequently co-occur, but synonym attributes seldom co-occur, to do schema matching. Chang et.al [26] took an approach consisting of the three phases, hypothesis modeling, hypothesis generation, and hypothesis selection (MGS). They assumed that in a similar domain there must be some underlying model that can describe all the schemas, and they tried to find such a hidden model. Other two types of schema matching approaches, linguistic approach and constraint-based approach, are introduced in [27]. Linguistic approach uses names and text to find semantically similar schema elements, and constraint-based approach uses constraints, such as data types, value ranges, uniqueness to do schema matching. In deep web mining, some other work has been done in query planning. Extracting data

<table>
<thead>
<tr>
<th>Exps</th>
<th>Query Key Term</th>
<th>Query Target Terms</th>
<th>Optimal Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Gene Name</td>
<td>SNP_ID, Alleles, Gene_Summary, GO_Info, Protein_Interaction, SIFT_Info</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>SNP_ID</td>
<td>SNP_General, SIFT_Info, Protein_Interaction, GO_Info, Protein_Seq, Ortho_Protein_Seq</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>SNP_ID</td>
<td>GO_Info, Protein_Seq, Ortho_Protein_Seq, Protein_Interaction</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>Gene Name</td>
<td>SNP_General, SNP_Chromosome, Protein_Seq, Ortho_Protein_Seq</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>Gene Name</td>
<td>SNP_General, Ortho_Blast</td>
<td>6</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Execution Time(s)</td>
<td>33.9</td>
<td>39.9</td>
<td>17.5</td>
<td>99.5</td>
<td>150.9</td>
</tr>
<tr>
<td>Query Planning Time(s)</td>
<td>2.68</td>
<td>4.44</td>
<td>2</td>
<td>4.07</td>
<td>4.23</td>
</tr>
<tr>
<td>Query Planning Time Percentage (%)</td>
<td>0.08</td>
<td>0.14</td>
<td>0.17</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>Database Availability Testing Time(s)</td>
<td>2.67</td>
<td>4.43</td>
<td>2.98</td>
<td>4.06</td>
<td>4.2</td>
</tr>
<tr>
<td>Availability Test Percentage (%)</td>
<td>99.6</td>
<td>99.7</td>
<td>99.3</td>
<td>99.7</td>
<td>99.2</td>
</tr>
<tr>
<td>Network Connection Time (s)</td>
<td>25</td>
<td>23.1</td>
<td>10.43</td>
<td>91.6</td>
<td>144.3</td>
</tr>
<tr>
<td>Connection Time Percentage(%)</td>
<td>75.8</td>
<td>76.7</td>
<td>59.5</td>
<td>92</td>
<td>95.6</td>
</tr>
<tr>
<td>Number of Terms Extracted</td>
<td>26</td>
<td>14</td>
<td>10</td>
<td>38</td>
<td>50</td>
</tr>
<tr>
<td>Execution Time per Term(s)</td>
<td>1.27</td>
<td>2.21</td>
<td>1.75</td>
<td>2.62</td>
<td>3</td>
</tr>
</tbody>
</table>
from web pages is also an important issue in deep web mining. In [28], Arasu et al defined a template, and proposed a model that describes how values are encoded into pages using a template. They presented an algorithm that takes a set of template-generated pages as input, deduces the unknown template used to generate the pages, and extracts the values encoded in the pages as the output.

VI. CONCLUSION

Providing integrated access to multiple data sources on the web is an important issue in information integration. Single Nucleotide Polymorphisms (SNPs) are promising factors in explaining the genetic contribution to complex diseases such as cancer and heart disease. The rapid growth of the volume of SNP data resulted in the need for an advanced tool which can integrate multiple SNP databases and automate the querying process of SNP information.

In the paper, we described our search and integration tool SNPMiner. This system is a query-oriented and mediator-based biological data integration and querying tool. It comprises of three major components, which are web server interface, dynamic query planner, and web page parser. The system integrates eight widely used general-purpose and specialized SNP databases which can provide us all important SNP related data. Our system includes a dynamic query planner which can schedule the query order on different databases according to user’s query request. Our experiments show that for most practical cases, the performance of our dynamic query planner is very good, the length of the query plan generated by our system is no more than 40% longer than the optimal one. We have also evaluated our system by measuring the execution time for various queries. We see that our dynamic query planner takes only a very small fraction of the time as compared to the entire execution time. On the average, our system needs 2 seconds to extract one term from the deep web.

Overall, we believe our SNPMiner can be a valuable tool for biologists. At the same time, our underlying techniques form the foundation for developing deep-web tools for other biological domains, or for enabling a more general tool in other domains.

REFERENCES