A Model for Quality Estimation in Biological Data Sources

TECHNICAL REPORT

Alexandra Martinez
Dr. Joachim Hammer
Dr. Sanjay Ranka

November 2006
Table of Contents

ABSTRACT ................................................................................................................................................ 3
1. INTRODUCTION ......................................................................................................................................... 3
  1.1 Quality Problems of Biological Data Sources ......................................................................................... 3
    1.1.1 Shortage of Quality Metadata ......................................................................................................... 3
    1.1.2 High Data Generation to Curation Ratio ......................................................................................... 4
    1.1.3 Lack of Quality-Driven Query Interfaces ......................................................................................... 4
2. RELATED WORK ....................................................................................................................................... 4
3. A MODEL FOR QUALITY ASSESSMENTS ........................................................................................... 6
  3.1 Framework ............................................................................................................................................ 6
  3.2 Quality Dimensions .............................................................................................................................. 6
    3.2.1 Per-Record Dimensions .................................................................................................................. 6
    3.2.2 Cross-Record Dimensions ............................................................................................................. 8
4. EVALUATION ......................................................................................................................................... 22
  4.1 System Architecture ............................................................................................................................... 22
  4.2 Choice of Data Model ............................................................................................................................ 24
  4.3 Prototype ............................................................................................................................................... 25
    4.3.1 Parameter Optimization .................................................................................................................. 26
  4.4 Data Set ................................................................................................................................................ 26
  4.5 Experiments and Results ....................................................................................................................... 27
5. SIGNIFICANCE, CONTRIBUTIONS AND BROADER IMPACT .......................................................... 37
6. FUTURE WORK ..................................................................................................................................... 38
REFERENCES ............................................................................................................................................. 39
ABSTRACT

We present a new model for estimating the quality of biological data in genomics repositories. The proposed model comprises a set of measurable quality dimensions, and a set of quantitative measures that can be systematically computed to provide a score for each quality dimension. Quality dimensions and measures are integrated into a semi-structured data model, which is suitable for representing both data and quality metadata, and can accommodate a wide variety of data models. We evaluate our model using a large data sample from NCBI (National Center for Biotechnology Information) databases, as well as feedback from domain experts. We believe that users of genomics repositories will benefit from the proposed quality model by being able to quickly discriminate high quality records (based on the records’ quality scores) without conducting much additional background research on the retrieved data.

1. INTRODUCTION

The rapid accumulation of biological information as well as their widespread usage by scientists to carry out research is posing new challenges to monitor and maintain the quality of data in public biological repositories. Genbank [NCG06], RefSeq [NCR06], and Swissprot [SIB06] are prominent examples of public repositories extensively used by biologists, bioinformaticians, and the overall scientific community. In the least, analysis and processing of low-quality (e.g., incorrect, or incomplete) data results in wasted time and resources. In the worst case, the usage of low-quality data may lead scientist to false conclusions or inferences, thus hampering scientific progress.

Although several quality models and assessment methodologies have been proposed in the literature, most are anchored in the context of enterprise data warehousing and are oriented to solve quality problems within the business domain. Hence they do not naturally fit into the genomics context, where the increasing data generation and usage rates impose constraints over the kind of quality assessments that can realistically be performed. A common approach for assessing information quality has been to gather quality appraisals from data users (e.g., in the form of questionnaires), but this approach has two main limitations. First, it is subject-dependent i.e., the results may vary depending on which users are chosen as subjects to carry on the study. Second, it cannot be efficiently applied to a significantly large data set because users can only do certain number of quality appraisals before performance starts to drop. The approach we propose overcomes such limitations by using only quality dimensions that can be objectively (and systematically) measured in an automated way, thus moving away from the subjective path.

1.1 Quality Problems of Biological Data Sources

We studied the quality problems currently existing in public biological repositories; particularly in NCBI’s resources like GenBank and RefSeq. We focused on these data sources because of their widespread use. Next, we describe the three major quality problems found in biological data sources.

1.1.1 Shortage of Quality Metadata

Currently, biological data sources provide minimal information about the quality of the stored data. Some repositories offer base-calling scores, but these quality indicators refer to the sequence data only. Typically, genomic records contain not only sequence data but also annotations about the sequence, which should be taken into account if an evaluation of the entire record is sought.
Several challenges must be overcome when addressing the shortage of quality metadata provided by the data sources. First, comprehensive quality assessments need to be formulated, which consider the entire contents of a record (i.e., sequence data and annotations). Second, different quality aspects of the stored data should be available in order to accommodate the large variation in usage and quality perception by users of the data sources. Consequently, quality must be evaluated from a multidimensional perspective. Third, a mechanism to represent and store quality information about the underlying biological data needs to be devised.

1.1.2 High Data Generation to Curation Ratio
Most public biological repositories have some kind of curation process in place, with the aim of cleaning, standardizing, and annotating the data submitted to the database. Even though this curation process can be (and has been) partially automated, a significant amount of human effort is still required. On the other hand, large amounts of biological data coming from different sequencing centers are loaded into the repositories every day. As a result, the ratio of data generation to data curation is increasing. For this reason, most sources publish their newly acquired data before it is completely curated, thus raising concern over the quality of available data.

One approach for addressing the high data generation to curation ratio problem is to fully automate the curation process. However, until this becomes a viable option, an indication of the quality of the available data (like the amount of annotations) would help users recognize curated versus non-curated data.

1.1.3 Lack of Quality-Driven Query Interfaces
Current query interfaces of biological data sources do not support specification of quality criteria as part of queries. Without such capability, the identification of high-quality records from the query results becomes a time-consuming task even for experienced users. While experienced users can generally glance at a record and roughly estimate its quality level, when a query retrieves a large number of records, examining each record individually is not convenient. Moreover, users who are new to one of these repositories would need to become familiar with the implicit quality indicators embedded in the data, before they can interpret and use them in a quality assessment. Not to mention that criteria used to evaluate of the retrieved records is subjective and depends large on user expertise. Hence, an automated way to present the query results sorted by quality would be preferred.

We believe that biological data sources should provide query interfaces that allow users to (i) selectively request quality metadata over the retrieved records, (ii) filter out data whose quality does not meet the expectations specified by the user, and (iii) order query results with respect to a given quality dimension.

2. RELATED WORK

Numerous models, evaluation methodologies, and improvement techniques have been developed in the area of Information Quality (IQ) [LS03, LSK02, MRV99, SLW97, WRK95]. IQ researchers often regard quality as “fitness for use” [BMW04], so the user’s perception of quality and the intended use of the data prevail in these approaches. Wang et al. [WRK95] proposed an attribute-based model to tag data with quality indicators. They suggest a hierarchy of data quality dimensions with four major dimensions: accessibility, interpretability, usefulness, and believability. These dimensions are in turn split into other factors such as availability, relevancy, accuracy, credibility, consistency, completeness, timeliness, and volatility. Mihaila et al.
[MRV99] identified four Quality of Data parameters: completeness, recency, frequency of updates, and granularity. Lee et al. [LS03] distinguished five dimensions of data quality: accessibility, relevancy, timeliness, completeness, and accuracy; each considered a performance goal of the data production process. Lee et al. [LSK02] developed a methodology for IQ assessments and benchmarks called AIMQ. AIMQ is based on a set of intrinsic, contextual, representational, accessibility IQ dimensions, which are important to information consumers. These dimensions were first devised by Strong et al. [SLW97] as categories for high-quality data. Naumann and Rolker [NR00] proposed an assessment-oriented classification of IQ criteria based on three sources of IQ (the user, the source, and the query process). More recently, Naumann and Roth [NR04] analysed how well modern (relational) DBMS meet user demands based on a set of IQ criteria. All these works offer valuable contributions for better understanding data quality problems and challenges, but they fail to provide quantitative measures for the quality dimensions or indicators proposed.

Data Quality has also been studied in the context of Cooperative Information Systems (CIS), where more pragmatic approaches have emerged [MSV03, MB03, NFL04, SVM04]. Mecella et al. [MSV03] describe a service-based framework for managing data quality in cooperative information systems, based on an XML model for representing and exchanging data and data quality. Scannapieco et al. [SVM04] developed the DaQuinCIS architecture and the D’Q (Data and Data Quality) model for managing data quality in cooperative information systems. They defined four data quality dimensions: accuracy, completeness, currency, and consistency. Naumann et al. [NFL04] presented a model for determining the completeness (i.e., a combination of density and coverage) of a source or combination of sources. Missier et al. [MB03] defined the notions of quality offer and quality demand within cooperative information systems, and modelled quality profiles as multidimensional date cubes. Bouzeghoub and Peralta [BP04] analyzed existing definitions and metrics for data freshness in the context of a data integration system (DIS). All these works deal with quality issues intrinsic to multiple-source systems (CIS, DIS) such as data exchange, data integration, notification services among the sources. Since we are primary concerned with the quality of single-source systems, most of those issues are not applicable to us and hence are not addressed by our model. Yet we believe our model nicely complements works in CIS and DIS because they typically do not provide solutions for measuring the quality at the source level.

Research efforts in the areas of Quality of Service (QoS) and Digital Libraries (DL) have also explored the characteristics and the role of quality [SW97, BH97, SKR03, BEA05]. QoS has mainly been developed to support distributed multimedia applications, which transmit and process audiovisual data streams. QoS comprises the quality specifications, mechanisms, and architecture necessary to ensure that user and/or application requirements are fulfilled [SW97, BH97]. In the context of Digital Libraries, Sumner et al. [SKR03] analysed the dimensions of educators’ perceptions of quality in digital library collections for classroom use. They found that most educators agree in what constitutes quality in digital collections, namely scientific accuracy; and that metadata influences how the quality of the collections is perceived. Beall [BEA05] describes the main types of errors in digital libraries, both in metadata and in actual documents; and offers suggestions for managing digital library data quality.

A few works have been proposed in the context of biological data quality. Particularly, the research by Müller et al. [MNF03] identifies the main errors involved in the process of genome data production as well as their corresponding data cleansing challenges. As a framework for understanding the sources and types of error, it is a valuable work, but it lacks concrete methodologies or assessment methods.

Finally, works on semistructured data modeling are also relevant to us because such data models have been extensively used in the biological domain, and because our quality model uses an
underlying semistructured data model. Most of the models proposed for semistructured data [ABS00, BDH96, CDL99, MAG97] share a common underlying representation, which is either a graph or a tree with labels on the nodes or on the edges. Abiteboul et al. [ABS00] use an edge-labeled graph to represent semistructured data. UnQL and LORE are based on an edge-labeled tree representation [BDH96, MAG97]. Calvanese et al. [CDL99] use the basic data model for semi-structured data (called BDFS) in which both databases and schemas are represented as graphs. The work by Scannapieco et al. [SVM04] provides a good example of the usage of a semistructured data model (in particular, XML) to represent both data and quality metadata.

3. A MODEL FOR QUALITY ASSESSMENTS

Most of the ideas presented in this section were published (with minor changes) in [MH05].

3.1 Framework

We briefly describe the reference framework for our quality model, which defines the important concepts we use throughout the paper.

We define \textit{Data Quality} as a measure of the trustworthiness of the data. Our quality model precisely aims to measure the trustworthiness of data stored in biological data sources. Since trustworthiness is a rather intangible concept, we decompose it along six different quantifiable dimensions.

\textit{Quality dimensions} are aspects of the quality of data which either the user or the data provider is interested in measuring. Since we aim for quantifiable quality dimensions, we need to specify how the quality dimensions will be measured. The particular formula or algorithm by which each dimension is assigned a value is called a \textit{measure}.

The set of quality dimensions of a data item is referred to as its \textit{quality metadata}, and it is represented as a vector where each entry holds the score of a quality dimension, e.g., $Q = [s_1, s_2, \ldots, s_n]$ with $s_1, s_2, \ldots, s_n$ the scores for the $n$ quality dimensions.

3.2 Quality Dimensions

In order to identify appropriate quality dimensions for our model, we looked for dimensions that could be objectively measured, could be computed efficiently, and were biologically-relevant. The relevancy for biology was preliminary judged by the authors, then validated by a field-expert, and lastly confirmed experimentally.

Using the criteria described above, we selected a set of seven measurable quality dimensions: \textit{Stability}, \textit{Density}, \textit{Freshness}, \textit{Correctness}, \textit{Redundancy}, \textit{Usefulness}, and \textit{Linkage}. The first four of these dimensions are \textit{per-record} dimensions and the last three are \textit{cross-record} dimensions. Per-record dimensions consider records on an individual basis. Cross-record dimensions consider the interactions among records.

Next we provide the intuition behind each quality dimension, and later formalize it using mathematical formulae. In what follows, we use the general term “data item” to denominate semantic data units such as records, fields of records, etc.

3.2.1 Per-Record Dimensions

\textit{Stability}

The Stability dimension captures information about fluctuations in the value of a given data item. The most appropriate information to look at for this dimension is the history of changes (updates)
of the data item. Such information is usually available in main public repositories in the form of a “version history” or “revision history”. Given this information, we propose to measure the magnitude of the updates (i.e., changes) applied to a data item, relative to its size, and then weigh this quantity by a function of the time elapsed since the updated occurred. This weighting function diminishes the influence of older updates in favour of more recent ones.

The stability of a data item behaves as follows. Recent updates largely decrease the stability score of a data item (qualifying it as unstable) whereas older updates have less effect over the stability score. If no updates are made to a data item for some period of time, the stability score will keep increasing until it either reaches its maximum value of 1 or until it is decreased due to a new update. Therefore, a period of low update frequency increases the stability of a data value, which is consistent with our expectation that the users are probably “happy” with the contents of the data item and therefore place a higher confidence in its correctness.

Density
This dimension provides an assessment of the amount of information conveyed by a data item. We propose to measure the amount of information as the number of “data units” where data unit refers either to a data value (e.g., string or number) or to a collection of data items. This rather abstract concept of data unit will take a concrete and natural form once the underlying data model is specified.

The intuition of this dimension is clear: a data item containing many data units will be considered denser than a data item containing a few data units. So, the more data units the larger the density.

Freshness
This dimension indicates how up-to-date the contents of a data item are. We are measuring it as a function of the time elapsed since the data item was last updated, using a logarithmic scale. The Freshness dimension is similar to Stability but differs in two significant ways. First, Stability considers all previous updates made to a data item whereas Freshness considers only the last one. Second, Stability accounts for the magnitude of the updates whereas Freshness ignores this information.

The intuition behind the freshness dimension of a data item is as follows. If the data item has remained unchanged for a long period of time, it is considered outdated, so its freshness score would be high. Conversely, if the data item has recently been updated, it is considered up-to-date, and its freshness score would be low.

Correctness
The Correctness dimension provides an estimate of the accuracy of a data item. Devising a way to measure the correctness or accuracy of biological data is not a trivial task. Available measures from other contexts cannot easily be applied to the biological domain. For example, Scannapieco et al. [SVM04] proposed the use of a distance function between the value stored at the database and the true value. In biology, however, such true value cannot be assumed to be available (or even known) due to the uncertainty associated to the data collection process and the shortage of knowledge about many biological interactions. Hence, a different approach is needed to estimate the correctness dimension of data in biological repositories.

We propose to use a combination of the stability and age of the data item in order to estimate its correctness. The rationale for this is described next. We believe stability and correctness are related because stable data items are more likely to have been accepted as correct information by both users and experts, than unstable data items. If a data item is temporary unstable due to an update, its correctness score will decrease; but as the item becomes more stable, its correctness
score will raise. We also believe that the age (defined as the time elapsed since the creation of the data item) and correctness of a data item are related. In general, we expect newly added data items to be less reliable or accurate than data items which have been in the repository for long time, simply because older data have had the chance of being studied, used, and annotated for a longer period of time.

3.2.2 Cross-Record Dimensions

Redundancy
The Redundancy dimension captures the amount of overlap present in a set of data items (more specifically, in a set of records), relative to the total amount of information conveyed by the set. We do not measure the redundancy for data items other than records because we assume that data items within a record do not contain overlapping information. Thus, redundancy is only measured across data items representing records.

The redundancy value of a record with respect to other records from a given set measures the maximum fraction of information contained in the record that overlaps with some other record in the set. For example, if a record has an overlap of 50% with respect to one record, and an overlap of 75% with a different record, we take the maximum of these values (0.75) as the redundancy score of the record.

A key issue to address is what “overlap” means and how to measure it. The most relevant kind of “overlap” for people in biology is the similarity at sequence level (called sequence homology). However, once they obtain records that overlap in their sequences, they also look for overlap at the annotations (or description) level. To measure sequence similarity, most biologists rely on BLAST scores. To measure annotations similarity, they generally look at the records and compare them manually, but deciding when the overlap is biologically significant usually is subjective. Using BLAST as the measure for overlap in our model is not feasible because it is computationally expensive. Hence a more efficient measure is needed, which also accounts for the similarity in annotations. An alternative approach to measure the overlap is to adopt a more conventional approach coming from the database domain, which uses a “distance function” between two records (the inverse of the distance would then estimate the overlap). We opt for this approach since it is more efficient and provides an objective measure that encompasses both the sequence similarity and the annotations similarity at once.

Usefulness
The Usefulness dimension indicates how useful a data item is. Objectively measuring this dimension is difficult since it has normally been the user who decides how useful a particular data item is for the task at hand. Yet we believe the perceived usefulness of a data item is influenced by its density, redundancy, and correctness. We therefore propose to measure usefulness as the fraction of non-redundant correct information conveyed by a data item. The positive effect of density and correctness over the usefulness of a data item is clear. High-density data items provide users with large amounts of information about the sequence at hand, which is considered advantageous. Likewise, highly accurate (correct) data items are deemed beneficial for users. The negative effect of redundancy over usefulness can be argued, as we have found in conflicting opinions expressed by experts.

Linkage
The Linkage dimension provides information about the interaction graph of a data item (record, in this context). This graph consists of a set of nodes representing records, and a set of directed
edges or links between nodes, representing relationships between records e.g., a link between a RefSeq record and an entry in the PubMed database indicates the corresponding PubMed article describes or uses the information in the RefSeq record. Our collaboration with biologists has shown that the occurrence of certain links (e.g., links to PubMed, Conserved Domains, and Entrez Gene databases) in genomics records is used to indicate higher levels of curation and thus more trustworthiness in the data provided. This is analogous to the Web, where the number and type of links is used to infer the importance of a Web site. So the intuition behind our linkage dimension is that a high link count is an indicator of high quality whereas a low link count is indicator of low quality. A comprehensive measure for the linkage of a record should consider both incoming and outgoing edges to/from the node corresponding to that record. However, so far our current measure includes only outgoing links since they are easier to collect. Two different modes of presenting the linkage dimension are proposed: extended mode and aggregated mode.

Extended Mode

In the extended mode, a set of distinct links is first obtained by an off-line scan of the records, and essentially any new URL found is added to the set. Each distinct link defines a link type. Then, for every record, we count the number of occurrences of each link type (effectively creating a histogram of the links found in a record, where each bin represents a link type). Thus, the Linkage dimension is virtually split up into several link types (i.e., sub-dimensions).

So far we have identified over 80 different link types, which have either been found in genomics records that we have analyzed or given to us as relevant links by domain experts. Table 1 lists these link types. It is worth mentioning that six of the links shown in Table 1 had to be removed in our experiments because they introduced an unwanted bias in the classifier. Specifically, those links clearly revealed the database from which each record in our data set was taken (RefSeq, dbEST, or SwissProt), and therefore, its classification label (more details in Section 4.4). The removed link types were: UniProtKB/Swiss-Prot Protein Knowledgebase, Universal Protein Resource (UniProt), Links to other sequences in Entrez (Genbank or RefSeq), Self links, RefSeq Website, and NCBI Expressed Sequence Tags (dbEST). The first two are links to the Swissprot database; the next three link to the RefSeq database; and the last one links to the dbEST database.

Table 1. Link types.

<p>| 1. | Atlas of Genetics and Cytogenetics in Oncology and Haematology |
| 2. | Berkeley Drosophila Genome Project |
| 3. | Caenorhabditis elegans |
| 4. | Cancer Genome Anatomy Project |
| 5. | Drosophila Genome (FlyBase) |
| 6. | ENZYME |
| 7. | Enzyme Website |
| 8. | European Hepatitis C Virus database |
| 9. | Expansins |
| 10. | Functional and Comparative Genomics of Disease Resistance Gene Homologs |
| 11. | Genetic Codes Website |
| 12. | Genome Exploration Research Group |
| 13. | Genomesystems Website |
| 14. | GO |
| 15. | HUGO Gene Nomenclature Committee (HGNC) |
| 16. | Human Protein Reference Database (HPRD) |
| 17. | I.M.A.G.E. Consortium Website |
| 18. | Invitrogen Website |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Japan National Institute of Genetics' Nematode Expression Pattern Database</td>
</tr>
<tr>
<td>20</td>
<td>Japan's National Institute of Radiological Sciences</td>
</tr>
<tr>
<td>21</td>
<td>Kazusa DNA Research Institute</td>
</tr>
<tr>
<td>22</td>
<td>Links to other sequences in Entrez (Genbank or RefSeq)</td>
</tr>
<tr>
<td>23</td>
<td>Malaria Full-Length cDNA DB</td>
</tr>
<tr>
<td>24</td>
<td>MGC Website</td>
</tr>
<tr>
<td>25</td>
<td>Mouse Genome Informatics (MGI)</td>
</tr>
<tr>
<td>26</td>
<td>NCBI 3D Domains</td>
</tr>
<tr>
<td>27</td>
<td>NCBI AceView</td>
</tr>
<tr>
<td>28</td>
<td>NCBI Cancer Chromosomes</td>
</tr>
<tr>
<td>29</td>
<td>NCBI COGs</td>
</tr>
<tr>
<td>30</td>
<td>NCBI Consensus CDS (CCDS)</td>
</tr>
<tr>
<td>31</td>
<td>NCBI Conserved Domains (CDD)</td>
</tr>
<tr>
<td>32</td>
<td>NCBI Documentation</td>
</tr>
<tr>
<td>33</td>
<td>NCBI Evidence Viewer</td>
</tr>
<tr>
<td>34</td>
<td>NCBI Expressed Sequence Tags (dbEST)</td>
</tr>
<tr>
<td>35</td>
<td>NCBI GenBank</td>
</tr>
<tr>
<td>36</td>
<td>NCBI Gene</td>
</tr>
<tr>
<td>37</td>
<td>NCBI Gene Expression Omnibus (GEO) Datasets</td>
</tr>
<tr>
<td>38</td>
<td>NCBI Gene Expression Omnibus (GEO) Profiles</td>
</tr>
<tr>
<td>39</td>
<td>NCBI Genome Project</td>
</tr>
<tr>
<td>40</td>
<td>NCBI Genome Survey Sequences (dbGSS)</td>
</tr>
<tr>
<td>41</td>
<td>NCBI Genomes</td>
</tr>
<tr>
<td>42</td>
<td>NCBI HomoloGene</td>
</tr>
<tr>
<td>43</td>
<td>NCBI Nucleotide</td>
</tr>
<tr>
<td>44</td>
<td>NCBI Online Mendelian Inheritance in Animals (OMIA)</td>
</tr>
<tr>
<td>45</td>
<td>NCBI Online Mendelian Inheritance in Man (OMIM)</td>
</tr>
<tr>
<td>46</td>
<td>NCBI PopSet</td>
</tr>
<tr>
<td>47</td>
<td>NCBI Probe</td>
</tr>
<tr>
<td>48</td>
<td>NCBI Protein DB</td>
</tr>
<tr>
<td>49</td>
<td>NCBI PubChem BioAssay</td>
</tr>
<tr>
<td>50</td>
<td>NCBI PubChem Compound</td>
</tr>
<tr>
<td>51</td>
<td>NCBI PubChem Substance</td>
</tr>
<tr>
<td>52</td>
<td>NCBI PubMed</td>
</tr>
<tr>
<td>53</td>
<td>NCBI Sequence Tagged Sites (dbSTS)</td>
</tr>
<tr>
<td>54</td>
<td>NCBI Serial Analysis of Gene Expression (SAGE)</td>
</tr>
<tr>
<td>55</td>
<td>NCBI Single Nucleotide Polymorphism (dbSNP)</td>
</tr>
<tr>
<td>56</td>
<td>NCBI Structure (MMDB)</td>
</tr>
<tr>
<td>57</td>
<td>NCBI Taxonomy</td>
</tr>
<tr>
<td>58</td>
<td>NCBI Third Party Annotation (TPA)</td>
</tr>
<tr>
<td>59</td>
<td>NCBI UniGene</td>
</tr>
<tr>
<td>60</td>
<td>NCBI UniSTS</td>
</tr>
<tr>
<td>61</td>
<td>NIH's Gene Tests</td>
</tr>
<tr>
<td>62</td>
<td>Plants (Mendel)</td>
</tr>
<tr>
<td>63</td>
<td>Protein Reviews On the Web (PROW)</td>
</tr>
<tr>
<td>64</td>
<td>Rat Genome and Nomenclature Committee (RGNC)</td>
</tr>
<tr>
<td>65</td>
<td>Rat Genome Database RatMap</td>
</tr>
<tr>
<td>66</td>
<td>RefSeq Website</td>
</tr>
<tr>
<td>67</td>
<td>RZPD German Resource Center for Genome Research</td>
</tr>
<tr>
<td>68</td>
<td>Saccharomyces Genome Database (SGD)</td>
</tr>
<tr>
<td>69</td>
<td>Stanford Human Genome Center</td>
</tr>
</tbody>
</table>
Aggregated Mode
In the aggregated mode, the linkage dimension considers all links collectively, and does not differentiate link types. The linkage measure (in aggregated mode) is the total number of links in a record. The aggregated mode thus summarizes the information presented in the extended mode in a single figure.

3.3 Measures for Quality Dimensions
So far we have shown a set of quality dimensions and their intuitive meaning. Our next step is to choose a suitable data (and metadata) representation that serves as a platform for the formulation of the quality dimensions’ measures.

3.3.1 Underlying Data Model
Before we can formulate measures for the quality dimensions, we need to choose a data model in which the underlying biological data will be represented. For this work, we chose the semistructured data model. Semistructured data is commonly described as “schemaless” or “self-describing” [1, 6] because the schema of the data is contained within the data. A semistructured data model generally represents data hierarchically (i.e., in a tree-like structure)\(^2\), with actual data lying at the bottom (i.e., leaf nodes) and schema information encoded in upper layers of the hierarchy (i.e., internal nodes). Here, leaf nodes store \(atomic\) data items, which can be either strings or numbers. Internal nodes represent \(complex\) data items, which are collections of other data items. A common example of this kind of data model is XML (Extensible Markup Language).

Figure 1(a) shows part of a nucleotide record from NCBI’s RefSeq represented in a hierarchical data model. Figure 1(b) sketches the semistructured representation of an example database, where the root of the tree represents the entire database, the nodes immediately below the root (i.e., root’s direct descendants) represent each a record in the database, and nodes below them represent data items within the records. It is worth noting that our quality estimation model can be used with any semistructured data model that adheres to the principles described above. Hence we refrain from using a specific syntax, and rather describe our model at a conceptual level.

---

\(^2\) Strictly, the semistructured data model allows cycles in the data, so a graph representation should be used (instead of a tree). However, when the nature of the data is acyclic, a tree-like structure can be assumed.
Several reasons justify the selection of a semistructured data model in this context. First, a vast amount of biological data is currently available in some form of semistructured data, as a result of public genomic repositories (e.g., GenBank [NCG06], EMBL [EBE06], and DDBJ [NID06]) publishing their data in XML format. Second, semistructured models have proven useful at representing biological data and its intrinsic complexities. This is demonstrated by the increasing number of XML-based languages developed within the biological context (e.g., BioML, BSML, AGAVE, GeneXML, MAGE-ML. Third, a semistructured data model can seamlessly represent both data and metadata, which is a desirable feature in our quality model. Finally, it can accommodate a variety of other data models, thus making it possible to estimate the quality of a wide variety of repositories that use different data representations.

Using the semistructured data model allows us to measure the different quality dimensions in a bottom-up fashion, which is described next.
Figure 1. (a) Fragment of a RefSeq record represented in a hierarchical model. (b) Sketch of the semistructured representation of an example database.
3.3.2 A Measure for Stability

In Section 3.2.1, we suggested to quantify the magnitude of the updates applied to a data item, and use a time-dependent weighting function to reduce the effect of older updates. This is formally defined in formula (1), where $S$ denotes the stability of an atomic data item $d$ (see Section 3.3.1).

$$S = 1 - \sum_{i=1}^{n} [\Delta(d(i-1), d(i))] \times \int_{t_i}^{t_{i+1}} \lambda e^{-\lambda t} dt$$  \hspace{1cm} (1)

Here, $n$ is the number of intervals at which we measure the stability of $d$, $t_i$ is the time elapsed since the $i$th interval (with $t_0 = \infty$), $d(i)$ is the state$^3$ of $d$ at interval $i$, and $\lambda > 0$ is a free parameter. The function $\Delta$ measures the fraction of $d$ that changed between two consecutive intervals. The integral of the exponential function applies a time-decaying weight to the changes applied to $d$ (giving more weight to recent changes than to old ones). Note that $S$ is initially 0 since $\Delta(d(0), d(1)) = 1$ for any data item $d$ (the default type of any data item at time $t_0$ is null, and $\Delta(null, d(1)) = 1$ for any $d(1) \neq null$, so the integral evaluates to 1).

Stability can be iteratively computed by using formulas (2) and (3). The stability score $S$ of a data item $d$ at time $t_k$ can be derived from its ‘instability’ score as in (2). And the instability score $I$ of $d$ at time $t_k$ only depends on the instability at time $t_{k-1}$ as in (3). Hence, we can efficiently compute $S$ if it measured at frequent intervals of time.

$$S_{tk} = 1 - I_{tk}$$  \hspace{1cm} (2)

$$I_{tk} = e^{-\lambda (t_k - t_{k-1})} \times I_{t_{k-1}} + \Delta(d_{k-1}, d_k) \times (e^{-\lambda t_k} - e^{-\lambda t_{k-1}})$$  \hspace{1cm} (3)

The function $\Delta(d_1, d_2)$ for atomic data items $d_1$ and $d_2$ is defined by formula (4). Note that $0 \leq \Delta(d_1, d_2) \leq 1$ for any pair $(d_1, d_2)$. If $d_1$, $d_2$ are numbers, this formula assumes that they are positive.

$$\Delta(d_1, d_2) = \begin{cases} \frac{\text{editDist}(d_1, d_2)}{\max\{\text{length}(d_1), \text{length}(d_2)\}} & \text{if } d_1, d_2 \text{ are strings} \\ \frac{|d_1, d_2|}{\max\{d_1, d_2\}} & \text{if } d_1, d_2 \text{ are numbers} \\ 1 & \text{otherwise} \end{cases}$$  \hspace{1cm} (4)

Once we calculate the stability scores for atomic data items at the bottom (leaves) of the tree, we can recursively compute the stability score for complex data items in upper levels of the tree. The stability $S$ of a complex data item $d$ is defined as the average over the stability score of its components (i.e., direct descendants of $d$ in the tree).

The stability score can only take values in the range $[0,1]$, with 0 meaning minimum stability and 1 meaning maximum stability.

3.3.3 A Measure for Density

In our informal description of density (see Section 3.2.1), we used the general term “data unit” to refer to objects that would count towards the density of a data item. Now we can refine such definition. Under the adopted hierarchically-structured data model, a data unit refers to either the

---

$^3$ The state of a data item corresponds to its type and contents.
atomic data represented by a leaf node, or the complex data represented by an internal node of the tree structure.

The density of an atomic data item $d$ is defined as 1, for any $d$. Hence, every atomic data item contributes in equal amount to the density score, despite the number of bits needed to store it. For example, if leaf node $l_1$ contains a large string $d_1$, and leaf node $l_2$ contains a short string $d_2$, each will have a density of 1 since each leaf node is believed to represent a meaningful semantic unit.

Formula (5) specifies how to measure the density of a complex data item $d$, with $n$ being the number of direct descendants of $d$ in the tree, and $D_i$ being the density score of the $i^{th}$ direct descendant of $d$.

$$D = 1 + \sum_{i=1}^{n} D_i$$  \hspace{1cm} (5)

The density $D$ of a complex data item $d$ is therefore the sum over the density of the elements (i.e., direct descendants) of $d$. This measure is equivalent to the size (i.e., number of nodes) of the subtree whose root node is $d$.

The score of the density dimension can take values on the interval $[1, \infty]$, with 1 meaning minimum density, and no upper limit.

### 3.3.4 A Measure for Freshness

One way of measuring freshness would be to subtract the last update time from the current time. This time-based distance could then be expressed in the preferred time units (e.g., days, seconds, etc). However, this simplistic approach has two shortcomings. First, it does not account for the frequency at which the data source gets updated. This could negatively affect the user perception of the data item’s freshness if the selected time unit is smaller than the frequency of update of the database. For example, if a data item $d$ coming from a monthly-updated database was updated two months ago, it is considered relatively fresh ($d$ is at most 2 times older than the most recent data in the database); but if we choose days as our time unit, $d$ might not seem very fresh to a user (its freshness value would be 60). The second limitation shows up when computing the freshness of complex data items. Since the freshness of a complex data item is determined as the average of the freshness of its components, if most of the components have low freshness scores (e.g., 0 or 1) but one of them have an extremely high score (e.g., 2000), then the average would be largely affected by the single high score, which is usually undesirable. This can be solved by transforming the data into a logarithmic scale.

The measure we propose does not suffer from the problems mentioned above. We start by computing the time-based distance in the preferred time units, as before, but then we make this distance relative to the database update frequency. Next, we apply a logarithmic transformation to circumvent the problem of extremely high values. These steps are condensed in formula (6), which defines the freshness of an atomic data item $d$.

$$T = \log \left(1 + \left[\frac{t-u}{f}\right]\right)$$  \hspace{1cm} (6)

Here, $t$ is the current time, $u$ is the time when $d$ was last updated, and $f$ is the frequency of update of the database (represented in the same time units as the subtraction in the numerator). A value of one is added to the argument of the logarithm to avoid the logarithm of zero.

The freshness of a complex data item $d$ is defined as the average over the freshness score of the elements of $d$. This average is robust to extremely high values due to the logarithmic scale used in formula (6).
The score for the Freshness dimension can take values on the interval \([0, \infty]\), with 0 denoting minimum freshness. There is no upper limit to the freshness score.

### 3.3.5 A Measure for Correctness

In Section 3.2.1, we suggested to measure the correctness using a combination of the stability and age of the data item. Formula (7) formalizes this idea by specifying how to compute the correctness \(C\) of an atomic data item \(d\).

\[
C = w_1 \times S + w_2 \times (1 - e^{-\beta \cdot \text{age}})
\]  

(7)

Here, \(\beta > 0\), \(0 \leq w_1 \leq 1\), and \(w_2 = 1 - w_1\) are free parameters, \(S\) is the stability score of \(d\), and \(\text{age}\) is the time elapsed since the creation of \(d\). \(C\) is therefore a weighted average of the stability and the age, with age being first mapped to the interval \([0,1)\) through an exponential function that maps new data items to 0, and old data items to values close to 1.

We believe that a correctness measure should also include information about the query pattern of the data item (e.g., how many times the data item has been queried, when did those queries occurred, etc). Such query information would be used similarly to the way in which the update history is used in stability. For example, the correctness of a data item that has been recently queried would increase, but if no queries are issued to the data item for a long period of time, its correctness would decrease. We believe that high query frequencies are indicators of high quality since the more a data item is queried, the more it is being used (and scientists normally use data they trust to be correct). Formula (8) exemplifies how we can incorporate the query information into the correctness measure in (11).

\[
C = w_1 \times S + w_2 \times (1 - e^{-\beta \cdot \text{age}}) + w_3 Q
\]  

(8)

Here, \(0 \leq w_1, w_2, w_3 \leq 1\) and \(w_1 + w_2 + w_3 = 1\). \(Q\) represents a function of the query pattern (i.e., its measure). The particular query measure to use depends on the information that is available at the data source. In our experiments, we did not include \(Q\) when computing correctness because the chosen repository did not provide information about the query pattern of each data item in the data source.

The correctness of a complex data item \(d\) is defined as the average over the correctness score of its components. The score of the correctness dimension can take values on the interval \([0,1]\), with 0 meaning minimum correctness and 1 meaning maximum correctness.

### 3.3.6 A Measure for Redundancy

In Section 3.2.2, we stated that we would not measure redundancy for data items other than records, so we express this by giving non-record data items a default score of zero.

Given a set \(A\) of records, formula (9) specifies how to compute the redundancy score of a record \(r \in A\) with respect to the other records in the \(A\).

\[
R = \max_{r \in A} \{1 - \text{dist}(r, r_i)\}
\]  

(9)

Here, \(\text{dist}(r_1, r_2)\) is a function that measures the distance between two records \(r_1\) and \(r_2\). Since we have assumed a hierarchical structure for all data items, our distance function is recursively applied to the descendants of the complex data items \(r_1\) and \(r_2\). For simple data items \(d_1 \in r_1\) and \(d_2 \in r_2\), \(\text{dist}(d_1, d_2) = \Delta(d_1, d_2)\). We ensure that \(0 \leq \text{dist}(r_1, r_2) \leq 1\). The similarity or overlap between two records is then estimated as the inverse of their distance.
Given a set $A$ of records (e.g., a data source), formula (10) specifies how to compute the redundancy of the set $A$ (assuming that the redundancy score of every record in $A$ with respect to the others is known).

$$R = \frac{\sum_{i=1}^{n} R_i \times D_i}{\sum_{i=1}^{n} D_i}$$

(10)

Here, $n$ is the number of records in $A$, and $R_i$ and $D_i$ are the redundancy and density scores of the $i^{th}$ record in $A$, respectively. We believe that a more meaningful measure when the set is the entire data source is the information content. This measure is derived from the redundancy, and indicates the fraction of unique or non-redundant information present in the data source. If there is high redundancy among the records of the data source, then its information content would be close to zero. On the other hand, if the records are non-redundant, then the information content of the data source would be high (i.e., 1). Formula (11) shows how to compute this measure for a set of records $A$.

$$I = \frac{\sum_{i=1}^{n} (1 - R_i) \times D_i}{\sum_{i=1}^{n} D_i}$$

(11)

Here, $n$ is the number of records in set $A$, and $R_i$ and $D_i$ are the redundancy and density scores of the $i^{th}$ record in $A$, respectively.

The redundancy score can take values on the interval $[0,1]$, with 0 meaning minimum redundancy and 1 meaning maximum redundancy.

3.3.7 A Measure for Usefulness

In Section 3.2.2, we described the usefulness measure as the amount of non-redundant correct information conveyed by a data item relative to its size. Formula (12) defines the usefulness $U$ of an atomic data item $d$. $D$, $C$, and $R$ are the density, correctness and redundancy scores of $d$, respectively. Since $0 \leq C \leq 1$, $0 \leq R \leq 1$, and $D = 1$ for atomic data items, $U$ is effectively the fraction of non-redundant correct information provided by $d$.

$$U = D \times C \times (1 - R)$$

(12)

Formula (13) defines the usefulness $U$ of a complex data item $d$. Here, $n$ is the number of direct descendants of $d$ in the tree, and $D_i$, $U_i$, and $R_i$ are the density, usefulness, and redundancy scores of the $i^{th}$ direct descendant of $d$, respectively.

$$U = \frac{\sum_{i=1}^{n} D_i \times U_i \times (1 - R_i)}{\sum_{i=1}^{n} D_i}$$

(13)

The score for the Usefulness dimension can take values on the interval $[0,1]$, with 0 meaning minimum usefulness and 1 meaning maximum usefulness.

3.3.8 A Measure for Linkage

Given that linkage is a cross-record dimension, we only provide the measure for complex data items here. In the aggregated mode of the linkage dimension, the linkage score $L$ of a complex
data item \(d\) that represents record \(r\) is defined as the number of links present in \(r\); so each link contributes with one unit to the total link count. In the extended mode, \(L\) would be a multidimensional value where each entry corresponds to a link type and contains the number of times that particular link type appears in the record \(r\).

Each of the linkage scores can take values on the interval \([0, \infty)\], with 0 meaning that no links exist. There is no upper limit on the value of the linkage scores.

### 3.3.9 Complexity Analysis of the Measures

Based upon the semistructured data model described in Section 3.3.1, we define \(n\) as the total number of nodes in the tree representing the database (see illustration in Figure 1(b)). Also, let \(m_r\) be the number of nodes needed to represent the largest record in the database; \(a_r\) be the average number of nodes needed to represent a record, \(a_c\) be the average number of child nodes of an internal node of the tree (i.e., complex data item), and \(r\) be the total number of records or root’s direct descendants (see illustration in Figure 1(b)). We first present the complexity analysis for the measures of the per-record dimensions and then for the measures of the cross-record dimensions. Our analysis distinguishes between atomic and complex data items; as well as between initialization and update times. Initialization time refers to the time when new biological data is added to the database (usually in the form of a record), so the scores of the quality dimensions should be given an initial value. Update time refers to the time when the biological data is updated (usually parts of a record are modified), so the score of each quality dimension needs to be updated to reflect the change in the underlying data.

**Per-Record Measures**

At initialization time, any of the per-record measures can be computed in constant time i.e., \(O(1)\) for atomic data items. On the other hand, computing the initial per-record measures for a complex data item takes time proportional to the number of direct descendants (i.e., ‘child’ nodes) of the complex data item at hand. On average, this would be \(O(a_c)\). Hence, initializing the scores of the per-record dimensions for the entire database can be done in \(O(n)\) time since a post-order traversal of the tree suffices.

At update time, any of the per-record measures except Stability can be computed in constant time for atomic data items. For Stability, the worst case happens when updating an atomic data item that is a string since the function \(\Delta\) uses edit distance between the old and new values of the string. Thus, the complexity of the stability measure at update time is in the worst case \(O(s_o * s_n)\), where \(s_o\) denotes the length of the old string and \(s_n\) denotes length of the new string. In the best case (when the atomic data item is a number), it takes constant time.

On the other hand, updating the per-record scores of a complex data item merely involve re-computing an average or similar aggregate over the complex item’s child nodes. In the naïve way, this would require \(O(a_c)\) time. However, if we store the sum of the child nodes’ scores rather than the final average, we can update this sum with one subtraction and one addition, hence taking \(O(1)\) time.

**Cross-Record Measures**

Both at initialization and update time, computing any of the cross-record measures takes constant time for atomic data items and for complex data items below the record level. For complex data items representing records, computing the cross-record measures takes \(O(a_r^*r)\) time since interactions among all \(r\) records are considered and each record contains, on average, \(2a_r\) nodes. Initializing and updating the scores of the cross-record dimensions for the entire database can thus be done in \(O(a_r^*r^2)\) time since the sub-trees of every pair of records need to be compared and each comparison takes \(O(a_r)\) time.


3.4 Quality-Aware Operations

Since we are primarily concerned with biological data, we must consider a scenario where data is constantly being updated and queried. Thus, we need to address the issues of how the quality measures described above are affected by data manipulation operations (e.g., insert, delete, update of fields or records), and how the quality measures extend the result of these operations. For this purpose, we will consider a core set of operations over hierarchically-structured data. Such set includes query operations such as selection, and maintenance operations such as insertion, update, and deletion. The way in which these operations are conceptually described here may differ from the way they actually get implemented, in order to meet additional efficiency constraints imposed by usage patterns.

For the discussion in subsequent sections, let \( v_1, v_2, \ldots, v_k \) (with \( v_k = v \)) be the sequence of adjacent vertices (or nodes) from the root of the tree to the node of interest \( v \). Then \( v_1, v_2, \ldots, v_k \) is called the path of \( v \) in the tree, and \( \{v_1, v_2, \ldots, v_{k-1}\} \) is the set of ancestors of \( v \) in the tree.

3.4.1 Query Operations

We only consider here ‘select’ type of queries. In the context of hierarchically structured data, a ‘select’ operation consists of navigating a path given by the user and then returning all or part of the contents of the node located at the end of the path.

Selecting a node and returning its contents

The Select operation takes an input path \( p = v_1, v_2, \ldots, v_k \), navigates this path to its last node \( v_k \), and returns the contents of this node. If \( v_k \) is a leaf node, its contents refers to the atomic data item (string or number) stored at \( v_k \). If \( v_k \) is an internal node, its contents refer to the subtree rooted at \( v_k \).

Typically under a select operation the quality measures of the node involved (\( v_k \) in this case) will not be affected since this is a ‘read’ operation (i.e., no changes are made to the contents of the node). However, if we choose to use formula (12) as our Correctness measure, then the correctness score of node being selected will change to reflect a change in the access pattern. Similarly, the usefulness score of the selected node will change since it depends on the correctness score. Besides updating these two quality measures of \( v_k \), we also need to propagate the change in \( v_k \)’s quality metadata to all its ancestors (i.e., \( v_1, v_2, \ldots, v_{k-1} \)). If we choose to use formula (11) then none of the quality scores of \( v_k \) will be affected by the select operation.

This operation will return both the contents of \( v_k \) and \( v_k \)’s quality metadata (i.e., scores of the quality dimensions).

3.4.2 Maintenance Operations

We consider here three types of maintenance operations: inserts, deletes, and updates. In the context of hierarchically structured data, each of these operations have to navigate a path given by the user, and perform the corresponding insertion, deletion or update at the end of the given path. Figure 2 illustrates an update operation (other operations work in a similar way). This figure shows the state of the database at times \( t_i \) and \( t_{i+1} \) in two equivalent representations: a tree representation and an XML-syntax representation. Two leaf nodes (colored in red) are updated at time \( t_{i+1} \), which causes an update to their quality metadata (only the change in Stability is shown). Then the quality metadata of their ancestors is also updated (colored in blue).
Figure 2. Sample Update Scenario. Two leaf nodes (in red) are updated at time $t_{i+1}$; then the quality metadata of the leaves and their ancestors (in blue) is also updated.

**Inserting a node**

The Insert operation takes a node $v$ (to be inserted) and a path $p=v_1, v_2, ..., v_k$ (where $v$ will be inserted) as inputs. It navigates path $p$ to its last node $v_k$, and inserts node $v$ as a child of $v_k$. Since $v$ is a new node, the scores of all its quality dimensions need to be computed for the first time. These initial scores are obtained using the formulae from Section 3.1.2. Next we sketch the steps involved in updating the quality measures under an insert operation.

- If $v$ is a single node:
  - Compute $v$’s quality scores using formulas (1), (3), (5), (7), (11), and (14).
- If $v$ is the root of a subtree:
  - Recursively compute the quality scores of all descendants of $v$.
  - Compute the quality scores of $v$ using formulas (2), (4), (6), (8), (13), and (15).
- In either case, add $v$ as a child node of $v_k$.
- Propagate the effect of this insertion to the ancestors of $v$ so that their quality metadata gets updated.

This operation will return both the path to the recently inserted node and the quality metadata associated to this new node.

**Deleting a node**
The Delete operation takes as input a path \( p = v_1, v_2, \ldots, v_k \), navigates it to the last node \( v_k \), and deletes this last node. When node \( v_k \) is deleted from the hierarchical data model, the quality dimensions of \( v_k \)’s parent (\( v_{k-1} \)) need to be recomputed to reflect the deletion. Next we sketch the steps involved in updating the quality measures under a delete operation.

- If \( v_k \) is a leaf node:
  - Delete \( v_k \).
- If \( v_k \) is an internal node, then we need to distinguish between two cases: (1) single node deletion case, and (2) subtree deletion case.
  - Single node deletion case:
    - Move \( v_k \)’s child nodes to path \( v_1, v_2, \ldots, v_{k-1} \) so that they become children of \( v_{k-1} \). Keep their quality scores unchanged.
    - Delete \( v_k \).
  - Subtree deletion case:
    - Delete \( v_k \) and all its descendants.
- Recompute the quality scores of node \( v_{k-1} \) and all its ancestors.

This operation will return the updated quality metadata of the just removed node’s parent, \( v_{k-1} \).

### Updating a node

The Update operation takes as input a path \( p = v_1, v_2, \ldots, v_k \), navigates this path to its last node \( v_k \), and updates this last node. Then the scores of \( v_k \)’s quality dimensions need to be recomputed to reflect the update to \( v_k \)’s contents. Next we sketch the steps involved in updating the quality measures under an update operation.

- If \( v_k \) is a leaf node:
  - Update \( v_k \).
  - Recompute \( v_k \)’s quality scores using formulas (1), (3), (5), (7), (11), and (14).
- If \( v_k \) is an internal node:
  - Update \( v_k \). We assume that only its label is being updated (i.e., none of the descendants of \( v_k \) is involved in the update).
  - Keep the scores of \( v_k \) unchanged. A change in the label of a node does not affect its quality measures.
  - Propagate the effect of the update (if any) to the ancestors of \( v_k \).

This operation will return both the path to the recently updated node and the updated quality metadata associated to this node.

#### 3.4.3 Complexity Analysis of the Operations

Here we provide the complexity analysis of the quality operations described previously. Let \( p = v_1, v_2, \ldots, v_k \) be the path to a node \( v \) in the tree, and \( s_p \) be the ‘size’ of path \( p \) where size is measured as the number of nodes in the path sequence (e.g., \( s_p = k \)). Suppose \( v \) is the node (with path \( p \)) on which the operations will be performed. Also, let \( n \) being the total number of nodes in the tree.

The Select operation takes \( O(s_p) \) time since we only need to traverse the path \( p \) once to reach \( v \) and obtain its contents. Our analysis does not include the time required to actually find node \( v \) in
the tree since we assume that all operations are given the path of \( v \) as input. This means that \( v \) has to be searched (and its path found) before any operation can be called. Since this is a common preprocessing step to all operations but is not part of the operations (as defined here), we simply disregard its time complexity (which is \( O(n) \)).

The time needed to perform each Insert, Update, and Delete operation is \( nav + op + prop \), where \( nav \) is the time needed to navigate through \( p \) to node \( v \), \( op \) is the time to perform a given operation (e.g., insert, update, or delete) on node \( v \), and \( prop \) is the time to propagate the changes up the tree to the ancestors of \( v \) (refer to Figure 2 for an illustration of a maintenance operation). Both \( nav \) and \( prop \) are \( O(s_p) \) for all measures except Redundancy. For Redundancy, the propagation phase takes \( O(a_r*r^2) \) time (see discussion in Section 3.1.2.8). The \( op \) time is analyzed next.

The time to perform an insert operation at node \( v \) depends on whether we insert a single node or a subtree. If a single node is inserted, it takes constant time because the initialization of the quality measures of a leaf node can be done in constant time (see Section 3.1.2.8). If a subtree is inserted, the insertion takes time proportional to \( t \), where \( t \) is the number of nodes in the subtree. This is because we have to initialize the quality measures of each node within the subtree. So, for inserts, \( op \) is \( O(1) \) if a single node is inserted and \( O(t) \) if a subtree is inserted.

The time to perform a delete operation at node \( v \) is constant. It does not depend on whether we delete a single node or a subtree (as in the insert operation) because we only consider the extra time needed to update the quality metadata when a data operation is performed. In the case of a subtree deletion operation, it may take \( O(t) \) time to delete all nodes in the subtree (depending on the particular implementation used) but it only takes constant time to update the quality measures of the parent node.

The time to perform an update operation at node \( v \) depends on whether we update a leaf node or an internal node. If a leaf node is updated, it takes \( O(v_o*v_n) \) time where \( v_o \) denotes the old data value and \( v_n \) denotes the new data value (see Section 3.1.2.8). If an internal node is updated, the update takes constant time.

In summary, the time needed to complete an Insert operation is \( O(s_p + t + a_r*r^2) \), the time needed to complete a Delete operation is \( O(s_p + a_r*r^2) \), and the time needed to complete an Update operation is \( O(s_p + v_o*v_n + a_r*r^2) \). All of these are worst-case times.

Note that the size of a path \( p \) is always upper-bounded by the height of the tree. Assuming that the tree is roughly balanced, we have \( s_p \leq \log n \).

4. EVALUATION

This section describes technical details of our testbed such as the system architecture, choice of data model for the test data, current preliminary prototype, and preliminary results.

4.1 System Architecture

Figure 3 depicts the overall architecture of the system we envision. It is a well modularized architecture consisting of (1) a data repository, (2) a quality metadata repository, and (3) a service layer that interacts with both the data and quality metadata repositories, to provide a consistent quality-aware view of the system underneath.
In this architecture, the actual biological data (data source in Figure 3) is stored separately from its corresponding quality metadata (metadata source in Figure 3) to allow more independence at the implementation level. For example, one could replace the way the data source is implemented without affecting the way in which quality metadata is managed, as long as an XML wrapper to the new data source is provided. Since virtually any data and data representation can be mapped to a semistructured data model (or XML), our quality model can be used with a variety of underlying data. Hence the decision of building our model on top of a semistructured data model facilitated the usability and portability of our model. Similarly, the implementation and contents of the quality metadata source can change without affecting the data stored in the data source. This decoupling or modularization is advantageous especially if we are concerned with minimizing the changes that an existing source requires in order to use our quality-aware model.

The Query Processor handles user queries. User queries are restricted to ‘select’ type of queries with ‘condition’ and ‘group by’ predicates. The query engine must be able to support quality specifications given by the user as part of a query (e.g., rank results based on a particular quality dimension, show only a subset of the quality dimensions, and filter out results whose quality score is lower than certain threshold). The Query Processor will forward user requests to the XML wrapper, which in turn interacts with the underlying data source’s native API. At the same time, the Query Processor will issue corresponding requests to the quality metadata source. Once it obtains answers from these two sources, it will combine them and possibly do some post-processing to provide the final results to the user.

The Metadata Manager handles administrator-level operations, which can only be performed by the data source admin. Administrator-level operations are data manipulation operations such as inserts, deletes, and updates. The way the Metadata Manager handles administrator requests is similar to the way the Query Processor handles user queries except for a main constraint: the
Metadata Manager must ensure that changes made to the data source are propagated to the quality metadata source; otherwise the information in the two repositories will not be consistent.

### 4.2 Choice of Data Model

We choose XML as the data model underlying the testbed since it is widely used in the biological community. We need to integrate the abstraction of a multidimensional quality vector \( Q = [S, D, F, R, C, U, L] \) into our XML data model. We devise a simple way of accomplishing this through the use of XML attributes. Since we can attach attributes to any element node in an XML document, our quality dimensions are simply mapped to attributes whose value correspond to the quality score (see Figure 4).

![Figure 4. Sample XML augmented with quality attributes.](image)

One problem with this approach is that attributes cannot be attached to XML text nodes, where actual data values (i.e., strings) reside. Most of the quality measures are calculated for atomic data items (i.e., strings or numbers) first and then for complex data items. Not being able to attach quality attributes to XML text nodes would correspond to not being able to store the quality scores of atomic data items. However, we realize that we can attach the quality attributes of a text node to its parent element node. An example illustrates this idea. Suppose you have the XML data shown in Figure 5(a), which is parsed as the DOM (Document Object Model) structure of Figure 5(b). Since the element node `<Org-ref_taxname>` has only one child text node, the quality scores of this child can be safely passed on to the parent for storage and be recovered at any time.

Another issue we have to deal with is the fact that in XML attributes can actually carry relevant information about the element node to which it belongs. However, our original hierarchical data model did not contemplate the existence of attributes, and rather assumed that all “relevant” data values resided in atomic data items at the leaves of the tree. Therefore, our quality measures have to be extended to account for the value of the attributes. We can also improve the way we compute some of our measures by taking advantage of the fact that in XML the names of element nodes convey meaningful information (in fact, the element names and their structural arrangement define the schema of the XML document). This, for example, can significantly reduce the computation time of the redundancy measure (particularly, the distance function) since only the element nodes (or complex data items) with equal name have to be processed.
4.3 Prototype

We have implemented a prototype of our quality estimation model mainly in Java, using Perl scripts to process the linkage dimension. The Java prototype reads in a set of XML documents corresponding to the biological records of interest, and processes them to compute their quality scores. Since NCBI is updated daily, we update the quality measures every day in our prototype. Figure 6 shows a diagram of the different modules that make up the prototype.

![Diagram of prototype modules](image)

The Load module is used at start up to load the initial set of XML records downloaded from NCBI into our quality-augmented database. When the Load module finishes, the quality database contains an initial set of quality-aware records. Here, “quality-aware records” refer to records that have been augmented with quality measures in the form of XML attributes, as described in Section 4.3.

Both the Update and the Aging modules are time-triggered. In our test setting, the time intervals at which these modules were triggered would theoretically have consisted of one day (due to the daily update rate of the NCBI repository). However, since we downloaded all the sample records together with their “version history” (i.e., all previous versions of the records), we did not really have to wait for an entire day in order to get the next update of each record (as would be the case...
in a real-time system). So, after we finish computing the update and aging of the records for one day, we can immediately continue the update and aging for the next day.

When the Update module is triggered, it looks for updates (new versions) of records that are already in the quality database. If it finds that one or more records were updated with a new version, it proceeds to compare the old and new versions of each of these records in order to find out the differences and update the quality scores accordingly. Once it is done computing the new quality scores, it adds to the database the new version of the record, and stores the old version in an archive database.

When the Aging module is triggered, it ages the records in the quality database for which no new versions were available. Aging a record means updating its quality scores to reflect the fact that the record contents have not been changed (stability, correctness, usefulness, and freshness need to be updated). Once this module is done updating the quality scores, it adds to the database the “aged” version of the record, and stores the old version in an archive database. The archiving feature can be turned off to boost the performance of the system.

4.3.1 Parameter Optimization

Our model has four free parameters, which can be fine-tuned using domain expertise. We suggest applying a method called ‘gradient descent’, which is commonly used in computer science for parameter optimization. This method can be implemented with a back-propagation algorithm. The input required from the experts consists of the “desired” or “target” value for each of the quality dimensions that need to be optimized. Not all quality dimensions have measures with free parameters, so we are only concerned with those that can be optimized through their free parameters. These dimensions are Stability, Correctness, and Usefulness. Although Usefulness does not have any free parameter by itself, it depends on Correctness, which has three free parameters. Correctness also depends on Stability, which has one free parameter.

In order to simplify the task of the domain experts, we can reduce the number of “target” values that they have to supply by taking advantage of the dependencies between the Usefulness, Correctness and Stability measures. Rather than optimizing the three measures concurrently, we can optimize only the Usefulness measure, which will indirectly set the parameters of the other two measures. Although this simplification does not directly optimize Correctness and Stability according to their “target” values, it is probably a good approximation considering the fact that Usefulness is a representative measure of the overall quality of a record. Another reason to adopt this simplification is that it is easier for a domain expert to give an assessment of the usefulness of a record than it is to evaluate its correctness or stability (criteria that they may not even use in their daily quality judgments). Therefore, asking the experts to give target values for correctness or stability and using these values to minimize the overall error across all three measures might actually yield poorer results than using a single usefulness target value.

Details on how the partial derivatives of the error can be computed incrementally through time are not shown here, but are available upon request.

4.4 Data Set

We used a data set consisting of over 3,400 records from two NCBI databases: dbEST and RefSeq. These two databases were chosen after consulting with some domain experts about the overall quality of these repositories. Experts agreed in that dbEST was a low quality repository, and RefSeq a high quality one. The reason why we needed these two databases to have significantly different quality levels was because we wanted to automatically classify a large amount of sample records as either high-quality (HQ) or low-quality (LQ) records. Automatically
assigning these labels to records was the only plausible way of training and testing our model over significantly large data sets for a convincing evaluation.

Records were downloaded in XML and HTML formats using the NCBI’s EUtils tool and Perl. In particular, we searched for dbEST records containing the words “incomplete”, “partial”, or “putative” in the Title field, which were indicators of low quality within the dbEST database.

The data set contains roughly the same amount of records from dbEST and from RefSeq. The data set includes records from 21 popular organisms in NCBI, which are listed in Table 2. Records were downloaded in both XML and HTML formats using the Entrez Programming Utilities (eUtils) from NCBI.

| 1. Arabidopsis thaliana | 11. Mus musculus |
| 2. Bos taurus | 12. Mycoplasma pneumoniae |
| 5. Danio rerio | 15. Pneumocystis carinii |
| 7. Drosophila melanogaster | 17. Saccharomyces cerevisiae |
| 8. Escherichia coli | 18. Schizosaccharomyces pombe |
| 9. Hepatitis C virus | 19. Takifugu rubripes |
| 10. Homo sapiens | 20. Xenopus laevis |
| | 21. Zea mays |

### 4.5 Experiments and Results

We ran our prototype over the data sample described above, and obtained the scores of each quality dimension for every record and data item within (when applicable). Such scores were calculated on a (simulated) daily basis since the time of creation of the oldest record in the data set (1989) until October 2006. Although we have the scores of more than 3,400 records along six quality dimensions over a period of 17 years, showing all of them is certainly not feasible. Hence, the results we present here are based on the latest scores computed for the records (as of October 2006).

Even though we suggested in Section 4.3.1 an optimization technique to achieve better quality estimations, we did not use it in the experiments presented here because of the difficulty of asking experts to provide quality scores for such a big data set. Another caveat of our current experimental evaluation is that it does not include the scores for the Redundancy dimension. The reason being that time complexity of measuring Redundancy is significantly high: $O(n^2)$ where $n$ is the number of records in the data set. Since our data set consists of over 3,400 records spanning over a period of 17 years, adding the ‘daily’ $O(n^2)$-computation for Redundancy was simply impractical. As a matter of fact, finding an efficient measure for Redundancy is a challenge that we need to address as we further develop our quality model. For the purpose of our preliminary experiments, though, we use a default score of zero for the Redundancy dimension of all records (i.e., meaning that there is no overlap among records in the data set). Avoiding the computation for Redundancy effectively transforms Usefulness into a per-record dimension since the interactions among records are not considered. In what follows, Usefulness will hence be considered part of the per-record dimensions, and Linkage (in both extended and aggregated modes) will be the only cross-record dimension.
To illustrate the distribution of the quality dimensions scores for the two data sets LQ and HQ, we plot normalized histograms in Figures 7-14. In all these figures, the distribution of the scores coming from dbEST records is labeled LQ (low quality) and the distribution of scores coming from RefSeq records is labeled HQ (high quality). Figures 7 and 8 show the histogram for two link types in the Extended-Linkage dimension: NCBI Gene and NCBI PubMed, respectively. Figures 9, 10, 11, 12, and 13 show the normalized histogram for the Density, Stability, Freshness, Correctness, and Usefulness dimensions, respectively (Redundancy is not plotted for reasons previously mentioned). Figure 14 shows the histogram for the Aggregated-Linkage dimension. The dimensions plotted in the figures 7, 8, and 9 have been found to be useful for discriminating between the LQ and HQ data sets (see Table 3).

From Figures 7, 8, 9, and 14, we can observe that the distributions of the quality scores over LQ and HQ data sets are different (particularly, their centers\(^4\)). Also, there is minimal overlap among the data points of the two distributions from figures 8, 9, and 14. This is an indication that records from LQ and HQ can be differentiated by setting a threshold on any of these dimensions (Extended-Linkage’s NCBI PubMed, Density, and Aggregated-Linkage). In Figures 10, 11, 12, and 13, however, the histograms for the LQ and HQ data sets significantly overlap, making it hard to find a clean boundary or threshold that separates the two classes. Finding a threshold for each quality dimension manually is not feasible, especially if we consider that there are at least 6 quality dimensions (when the Aggregated-Linkage is used), and possibly up to 86 dimensions (when the Extended-Linkage is used with all 81 link types). Therefore, we decided to use C4.5 \cite{QUI93} instead. C4.5 is a publicly-downloadable classifier that builds decision trees from a set of examples. C4.5 can be given a training set, from which classification rules are learned, and then a testing set, from which a classification error can be obtained.

\(^4\) The center of a distribution commonly refers to its mean, but the median or other location measures can also be used.
Figure 7. Normalized histogram of the link count for the ‘NCBI Gene database’ link type (in the extended-Linkage dimension) over records in HQ and LQ.
Figure 8. Normalized histogram of the link count to the ‘NCBI PubMed’ type (in the extended-Linkage dimension) over records in HQ and LQ (log scale used).
Figure 9. Normalized histogram of the Density score for records in HQ and LQ sets (log scale used).
Figure 10. Normalized histogram of the *Stability* score for records in HQ and LQ sets.
Figure 11. Normalized histogram of the *Freshness* score for records in HQ and LQ sets.
Figure 12. Normalized histogram of the Correctness score for records in HQ and LQ sets.
Figure 13. Normalized histogram of the *Usefulness* score for records in HQ and LQ sets.
Figure 14. Normalized histogram of the *aggregated-Linkage* dimension for records in HQ and LQ sets (log scale used).

In our experimental setting, we used cross-validation [TSK05] for evaluating the performance of the classifier built by C4.5. The entire data set was divided into 10 mutually exclusive subsets, hence resulting in a 10-fold cross-validation. During each fold, the classifier was built (i.e., trained) using 9 subsets and the remaining set was used for evaluating (i.e., testing) the performance of the classifier.

Four different combinations of *per-record* and *cross-record* quality dimensions were explored using C4.5. The purpose of trying different combinations of dimensions when building the decision tree was to discover which dimensions better classified the given test records from LQ and HQ sets. For each of the four combinations of dimensions, an experiment was conducted. The first experiment was a 10-fold cross-validation over the *per-record* quality dimensions. The second experiment was a 10-fold cross-validation over the extended-Linkage (eL) *cross-record* dimension. The third experiment was a 10-fold cross-validation over the set of *per-record* and aggregated-Linkage (aL) *cross-record* dimensions. The last experiment was a 10-fold cross-validation over the set of *per-record* and extended-Linkage *cross-record* dimensions. Results are summarized in Table 3.
Table 3. Classifier performance for different experiments using cross-validation.

<table>
<thead>
<tr>
<th>Per-record dimensions (S,D,F,U)</th>
<th>Average Classification Error</th>
<th>Significant Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-record dimension (eL)</td>
<td>0.4%</td>
<td>D, S, F, U</td>
</tr>
<tr>
<td>Per-record &amp; cross-record dimensions (S,D,F,U,aL)</td>
<td>0.6%</td>
<td>L26, L1</td>
</tr>
<tr>
<td>Per-record &amp; cross-record dimensions (S,D,F,U,eL)</td>
<td>0.2%</td>
<td>D, L, S, U</td>
</tr>
</tbody>
</table>

Table 3 shows in the second column the classification error averaged over the ten folds for each of the four experiments conducted. The last column of Table 3 shows the dimensions that found relevant for classifying the LQ and HQ data sets (called ‘significant dimensions’). An important finding was that most of the significant dimensions (bold font ones) were consistently chosen by C4.5 across all the ten training sets generated during the 10-fold cross-validation. Usefulness (abbreviated U in Table 3) was chosen by C4.5 only six out of ten times in the first experiment, and three out of ten times in the third experiment. Another interesting finding was that C4.5 chose Density (abbreviated D) as the first significant dimension in all the experiments that actually included it. Other dimensions such as Stability (S), Freshness (F), and aggregated-Linkage (aL) were also found to be relevant. It is also worth noting that from the eighty-one link type attributes in the extended-Linkage (eL) dimension, only two were chosen by the classifier to be significant, namely NCBI PubMed (L1) and NCBI Gene (L26) (their distribution is shown in figures 7 and 8). Smaller classification errors were obtained when using a combination of per-record and cross-record dimensions (last two experiments from Table 3).

These results demonstrate the usefulness of the attributes (both per-record as well as cross-record) that we have chosen. The importance of the attributes found was also (a posteriori) confirmed by domain scientists. These results are to an extent dependent on the choice of the datasets that are deemed “good” and “bad”. However, it demonstrates the power of our approach as well as the importance of the per-record and cross-record attributes. We strongly believe that the use of these and other attributes can be leveraged to build an automatic system for classification and then can be extended for scoring (non-binary) the quality of the records. Such a system can also incorporate user feedback on the reasonableness of the estimates of quality and then used to refine the scoring algorithms.

5. SIGNIFICANCE, CONTRIBUTIONS AND BROADER IMPACT

Although several quality models and assessment methodologies have been proposed in the literature, most are anchored in the context of enterprise data warehousing and are oriented to solve quality problems within the business domain (see Section 2). Hence they do not naturally fit into the genomics context, where the increasing data generation and usage rates impose constraints over the kind of quality assessments that can realistically be performed. Instead of relying on subjective appraisals gathered from data users via questionnaires or alike (as in
previous works), our novel approach assesses the quality of data using quantitative measures that can be systematically computed from the data already stored in the database.

It is worth mentioning that although some quality indicators are already provided by a few repositories in the form of base-calling quality scores, for example, these indicators refer solely to the quality of the sequence data. Genomic records also contain annotations about their sequence data, which should be taken into account when evaluating the record’s quality. Our quality assessments are thus comprehensive because they consider the entire contents of the records (i.e., annotations plus sequence data), using estimates for the different aspects (dimensions) of information quality.

The main contributions of this work are:

- The identification of a set of measurable quality dimensions fit for genomic data.
- The formulation of quantitative measures for the quality dimensions, which can be computed in a systematic way.
- The integration of the quality dimensions and associated measures into a data model suitable for representing both data and quality metadata.
- The definition of a set of maintenance and query operations over the quality-augmented data model.

We expect our quality model to have a broad impact on how data stored in public repositories is curated and used. The perceived value and usefulness of existing repositories will be enhanced through a query interface that allows users to selectively request the quality scores of the selected records, and to filter out query results below a given threshold for one or more of the quality dimensions. As a result, users will be able to quickly discriminate high quality records without conducting further background research on the retrieved information. We also believe the data curation process will be facilitated by providing computed estimates for the quality of records initially submitted to the database. Our model can help curators prioritize records for further editing or revision.

Even though genomic data and genomics databases are the target scenario of our work, the quality assessment model we propose can be applied to other scenarios as well. One such scenario where we foresee immediate application is in web content management systems such as wikis [EET06, WIK06] where data undergoes frequent updates by several users. Other application scenarios include databases where the correctness and freshness of the data are of utmost importance, for example the Department of Motor Vehicles (DMV) database, and the Census Bureau databases.

6. FUTURE WORK

We have developed a model for estimating the information quality in biological databases. The novelty of our approach resides in the development of quality dimensions and measures appropriate for genomic data, with an emphasis on quantitative assessments that can be systematically computed.

We have implemented our quality estimation model in a functional prototype. Our experimental evaluation demonstrates that the proposed quality estimation model is capable of providing meaningful and valuable quality information. Experimental results demonstrate the usefulness of the attributes that we have chosen (both per-record and cross-record). We strongly believe that
the use of these and other attributes can be leveraged to build an automatic system for classification and then can be extended for scoring the quality of the records. Such a system can also incorporate user feedback on the reasonableness of the estimates of quality and then used to refine the scoring algorithms.

We are currently in the process of designing a experimental phase with broader samples and participation of subject matter experts.

REFERENCES


[EBE06] European Bioinformatics Institute. EMBL Nucleotide Sequence Database (Feb 2006). Available at http://www.ebi.ac.uk/embl/


