Data Warehouse/OLAP System

Meghana Ananth Gad (50182335)
Ladan Golshanara (50093954)
Duc Thanh Anh Luong (50094977)

October 2016

Abstract

In this report, we describe a typical data warehouse for bioinformatics application which supports various OLAP operations and enables knowledge discovery in bioinformatics.

1 Data warehouse schema (Part I)

The original format of dataset is in star schema which is a canonical way to store data in data warehouse application. In paper by Wang et al. [1], the Biostar schema seems to have few advantages over the canonical star schema. For this reason, in our data warehouse application, instead of using star shcema, we use BioStar schema to store our dataset.

First, we will briefly describe the details of schema we use in our application.
1- Clinincal data space
Central Entity: patient
Dimensions: disease, drug, test
M_tables: diagnosis, druguse, testresult

patient(p_id, ssn, name, gender, DOB,s_id)
disease(ds_id, name, type, description)
drug(dr_id, name, type, description)
test(tt_id, name, type, setting)
diagnosis(p_id, ds_id, symptom, ds_from, ds_to)
druguse(p_id, dr_id, dosage, dr_from, dr_to)
testresult(p_id, tt_id, result, tt_date)

Note that we put *s_id* in the patient table instead of making a separte table that stores (p_id, s_id) because 53 patients (out of 60 patients) have a unique s_id and 7 patients have no sample ids. Therefore, we put s_id in patient table.
2- Sample data space
Central Entity: sample

Dimensions: marker, assay, term
M_tables: sample_marker, sample_assay, sample_term

sample(s_id, source, ampunt, sp_date)
marker(mk_id, name, type, locus, description)
assay(as_id, name, type, setting, description)
term(tm_id, name, type, setting)
sample_marker(s_id, mk_id, mk_result, mk_date)
sample_assay(s_id, as_id, as_result, as_date)
sample_term(s_id, tm_id, tm_description)

3-Microarray and proteomic data space
Central Entity: Probe
Dimension: measureUnit
M_table: microarray_fact

probe(pb_id, uid1, name,desciption, isQC)
measureUnit(mu_id, name, type, description)
microarray_fact(s_id, e_id, pb_id, mu_id, exp)

4- Gene data space
Central Entity: Gene
Dimension: go, cluster, domain, prometer
M_table: gene_go_cluster, gene_domain

gene(uid1, seqType, accession, versio, seqDataset, speciesId, status)
go(go_id, accession, type, name, definition) **cluster**(cl_id, num, pattern, tool, tSetting, description)
domain(dm_id, type, db, accession, title, length, description)
promoter(pm_id, type, sequence, length, description)
gene_go_cluster (uid1,cl_id, go_id)
gene_domain(uid1, dm_id)
Note that dimensions *go* and *cluster* share one M_table (i.e. gene_go_cluster). We do not show the experiment biostar schema because it is not used in the queries of this project.

All the entity identifiers such as p_id, s_id, etc. are converted to integers instead of strings to increase efficiency of joins and other database operators.

Also, note that SQL is a declarative language, meaning that the query optimization part and the type of joins are selected by the query optimizer of Oracle depending on available indexes. We have removed the primary key constraints due to duplicates in some tables. Therefore, we do not have index on any table. The plan of all queries has been checked in ORACLE and the time complexities are given based on the decision of Oracle query optimizer.

2 Basic OLAP operations (Part II)

In this section, we explain basic OLAP operations that can be done to assist knowledge discovery process. We have implemented a GUI for all OLAP operations in python. Details of the GUI program can be obtained in README file inside the Code folder of the submitted zipped folder.

2.1 List the number of patients with particular disease information

For demonstrating this operation, we will perform three specific operations, each corresponds to a particular disease information. These operations include:

- List the number of patients who had "tumor" (disease description)
- List the number of patients who had "leukemia" (disease type)
- List the number of patients who had "ALL" (disease name)

User can further select his/her disease information using our graphic user inferface. The GUI for the aforementioned operations is shown in Figure 1 while the result is shown in Figure 2.

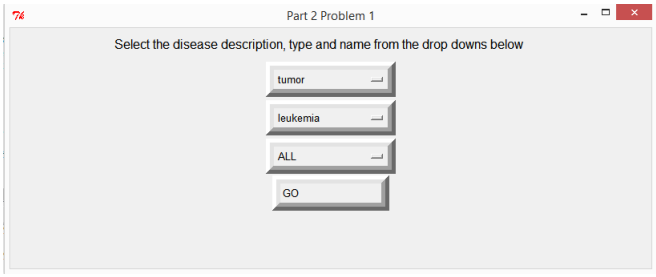

Figure 1: Program interface for listing the number of patients with particular disease information

These queries are combination of *dice* and *roll up*(i.e. summerizing) operations. Oracle query optimizer used *sort-merge join* for these queries. So the complexity is $O(n * logn)$, where n is the maximum number of tuples in the diagnosis and disease tables.

2.2 List the types of drug that patients with specific disease information used

For demonstration purpose, we will write a query to list the types of drug that patients with "tumor" used. Other select options can also be used in our GUI program.

The query for this operation can be written as follow:

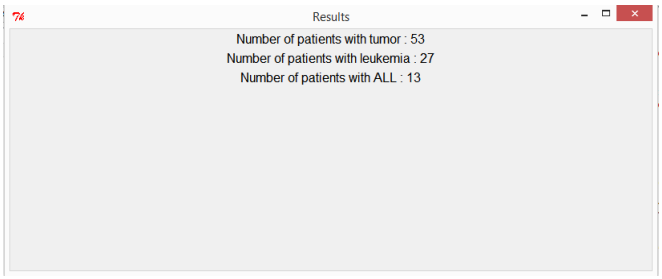

Figure 2: Result for listing the number of patients with particular disease information

```
SELECT distinct dr.TYPE
FROM Drug dr, DRUGUSE du, disease ds, DIAGNOSIS di
WHERE dr.DR_ID = du.DR_ID
and du.P_ID = di.P_ID
and di.DS_ID = ds.DS_ID
and ds.DESCRIPTION = 'tumor';
```

Figure 3 and 4 show the interface of the program and the result of the above operation.

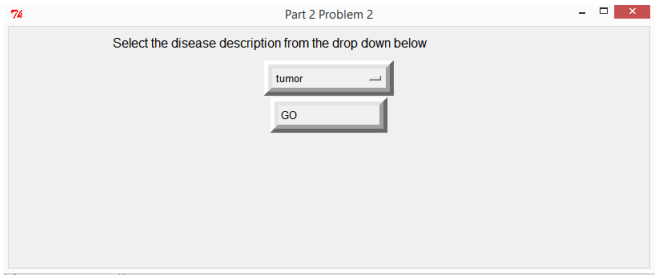

Figure 3: Program interface for listing type of drugs that patients with specific disease information used

In the above query, the oracle query optimizer uses hash join for the joins between tables Drug, Druguse and Diagnosis, and a sort-merge join between Diagnosis and Disease. Therefore the time complexity is dominated by sort-merge join $O(n * log(n))$, where n is the number of tuples.

2.3 List the mRNA values of patients with specific disease information and gene information

In bioinformatics application, the operation to extract mRNA values of patients with specific disease information and gene information appears very often. This

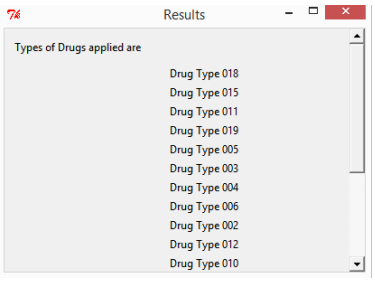

Figure 4: Result for listing type of drugs that patients with specific disease information used

type of operation requires the extraction of both disease information and gene information which may be expensive for large dataset.

We demonstrate in this report one such specific operation: For each sample of patients with "ALL", list the mRNA values (expression) of probes in cluster id "00002" for each experiment with measure unit id = "001".

Figure 5 and 6 show the interface of the program and the result of the above operation. The interface for this query is as follow:

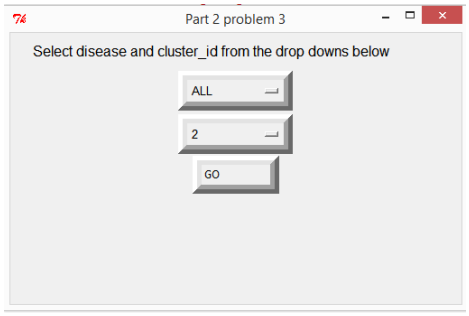

Figure 5: Program interface for listing the mRNA values of patients with specific disease information and gene information

2.4 Understanding the differences in some particular group of genes between two groups of patients

In order to facilitate knowledge discovery of genes that are relevant to an application, we often need to compare the difference in gene info of one particular group of patients with another group of patients. For this reason, in our bioinformatics warehouse application, we also support this operation by using statistical hypothesis testing.

For the sake of demonstration, we will write a query to extract mRNA values

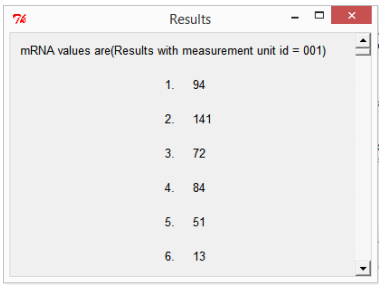

Figure 6: Result for listing the mRNA values of patients with specific disease information and gene information

of a specific group of genes for two groups of patients and perform hypothesis testing to test whether there is a significant difference in gene expression among two groups. In particular, we consider the probes with go_id = "0012502", extract all of mRNA values for two groups of patients (one with disease "ALL" and one without "ALL") and perform t-test to test whether those mRNA values are from the same distribution or from different distribution. In the t-test, we assume that two groups have equal variance. Moreover, since we don't know which group has higher mRNA than the other group, two-sided t-test is used instead of one-sided test.

The SQL query of for this operation can be written as follow:

```
SELECT STATS_T_TEST_INDEP(temp.type, temp.value, 'STATISTIC', 1)
as t_statistic
FROM
(SELECT 1 as type, mf.expression as value
      FROM microarrayfact mf
      where mf.pb_id in
      (SELECT pb_id
      FROM probe pr, GENE_GO_CLUSTER gcc WHERE pr.uid1 = gcc.uid1
            and gcc.GO_ID= 12502 )
      and mf.s_id in
      (SELECT s_id
      FROM  patient1 p, DIAGNOSIS di,DISEASE ds
      WHERE p.P_ID=di.P_ID and di.DS_ID = ds.DS_ID
      and ds.NAME = 'ALL')
      UNION ALL
      SELECT 2 as type, mf.expression as value
      FROM microarrayfact mf
      WHERE mf.pb_id in
      (SELECT pb_id
      FROM probe pr, GENE_GO_CLUSTER gcc WHERE pr.uid1 = gcc.uid1
      and gcc.GO_ID= 12502
      )
```

```
    and mf.s_id in
    (SELECT s_id
    FROM patient1 p, DIAGNOSIS di,DISEASE ds
    WHERE p.P_ID=di.P_ID and di.DS_ID = ds.DS_ID
          and ds.NAME <> 'ALL')
) temp;
```

Alternatively, instead of purely using SQL query to perform operation, one may want to extract the mRNA values for two groups from data warehouse to their computer and perform t-test locally. That approach is what we have done in our GUI program with python t-test. Figure 7 and 8 shows the interface of our program and the result of the analysis.

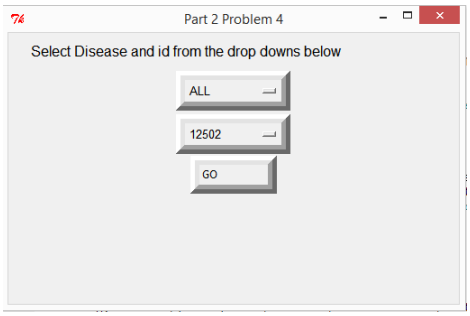

Figure 7: Program interface for comparing multiple groups using ANOVA

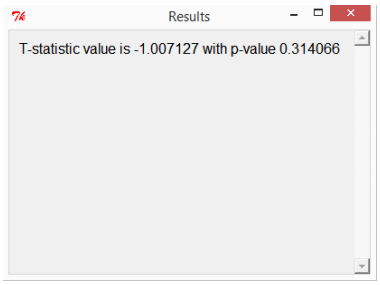

Figure 8: Result of comparing multiple groups of patients using ANOVA on a specific set of genes corresponding to go_id="0007154"

The execution plan of this query according to Oracle query optimizer is hash joins betweeb microarray, patient, probe and gene_go_cluster tables, and a sort merge join on diagnosis and disease tables. Therefore, the complexity is dominated by sort-merge join $O(n * logn)$.

2.5 Understanding the differences in some particular group of genes between multiple groups of patients

In the previous part, we have mentioned how to compare two groups of patients and understand the differences in gene expressions. However, in practice, it is quite often that more than two groups need to be compare against each other. For this reason, instead of using t-test for hypothesis testing, we use analysis of variance (ANOVA) to compare the differences in multiple group.

In our demonstration, we want to understand whether the mRNA values for probe with go_id="0007154" are significantly different among groups of patients having different diseases (in our case the diseases include: ALL, AML, colon tumor and breast tumor). For this reason, we perform ANOVA on the mRNAs obtained from each group of patients.

Similar to Section 2.4, this operation can be performed using either SQL query (using function STATS_ONE_WAY_ANOVA) or download extracted data and perform ANOVA internally. In our application, we use the second approach. Figure 9 and 10 shows the interface of our program and the result of the analysis.

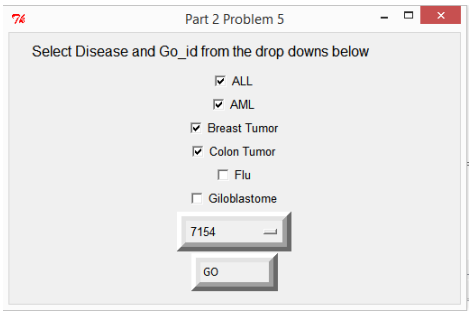

Figure 9: Program interface for comparing multiple groups using ANOVA

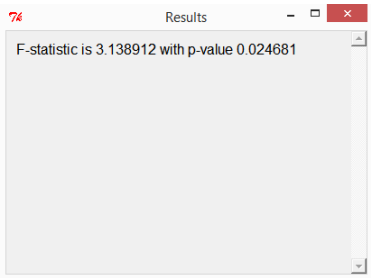

Figure 10: Result of comparing multiple groups of patients using ANOVA on a specific set of genes corresponding to go_id="0007154"

2.6 Correlation on the basis of gene expressions between two patients

Another popular operation that analysts usually perform in bioinformatics application is computing correlation on gene expressions between two patients. We demonstrate this operation through an example as follow: For probes belonging to GO with id="0007154", calculate the average correlation of the expression values between two patients with "ALL", and calculate the average correlation of the expression values between one "ALL" patient and one "AML" patient.

In our program, user needs to specify which group of patients they want to analyze first. Figure 11 shows the interface for choosing it. After choosing the group of patients, the program will calculate the average correlation among patients of this group and show the result as shown in Figure 12.All the calculation was done with mRNA values coresponding to genes with go_id = "0007154".

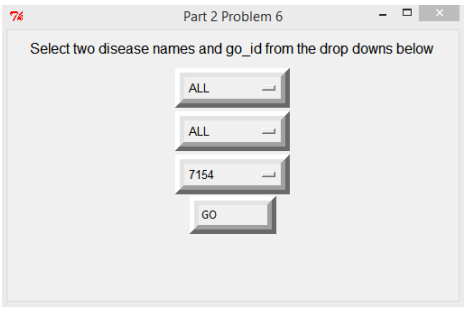

Figure 11: Program interface for choosing group of patients

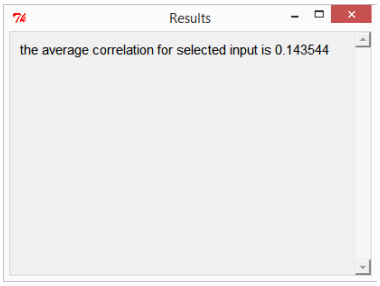

Figure 12: Result of average correlation of the group of patients with "ALL" for genes having go_id = "0007154"

The average correlation can also be computed between two groups of patients. In our program, user first needs to choose the pair of disease that he/she wants to compute correlation accross two groups with respect to genes with go_id = "0007154". Figure 13 shows the interface and Figure 14 shows the result when users choose "ALL" and "AML".

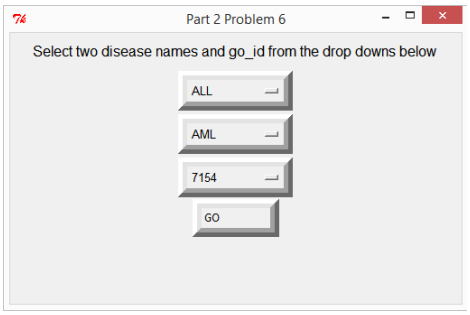

Figure 13: Program interface for choosing a pair groups of patients

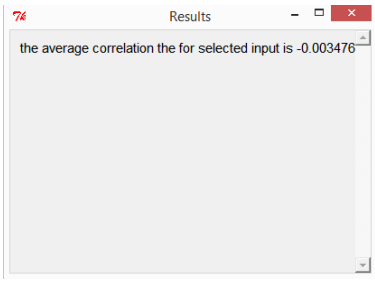

Figure 14: Result of average correlation accross two groups "ALL" and "AML" for genes having go_id = "0007154"

3 Knowledge discovery (Part III)

In this section, we show a use case when user can use our application to perform knowledge discovery.

Analysts who are highly interested in finding informative genes for a specific disease can use our application to do so. In our application, user can choose a specific disease and the program will enumerate all informative gene (UID) corresponding to that particular disease. All the informative genes are computed by performing t-test between groups of patients having disease and group of patients not having disease. Figure 15 shows the interface for choosing a disease. After that, the list of informative genes are shown on the screen as shown in Figure16. The new patients are classified based on those informative genes as shown in Figure17.

Note that in the given dataset each probe id has been associated with one UID. Therefore, we have reduce one redundant join operation between Gene and Probe tables. Instead, we used the UIDs that are given in probe table.
Note: In our exisiting implementation of the knowledge discovery part, we dynamically compute the informative genes to classify new patients. But if the same computations are to be performed on a larger dataset then we suggest that the informative genes should be precomputed and stored in materialized

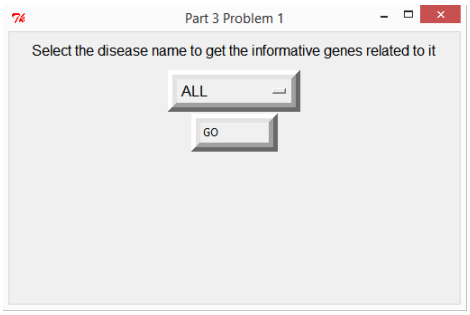

Figure 15: Choosing a disease for enumerating informative genes

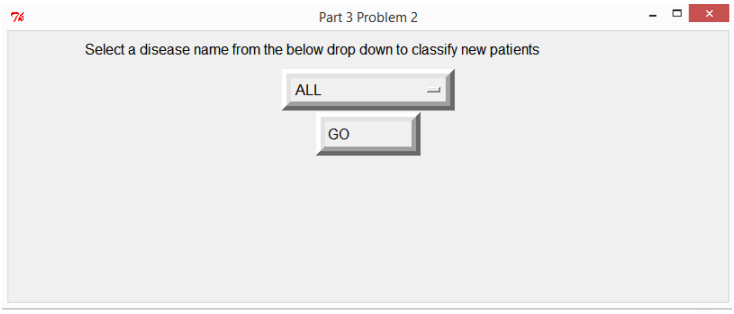

Figure 16: List of informative genes for "ALL"

views or tables for faster computations.

References

[1] Liangjiang Wang, Aidong Zhang, and Murali Ramanathan. "BioStar models of clinical and genomic data for biomedical data warehouse design." International journal of bioinformatics research and applications 1.1 (2005): 63-80.

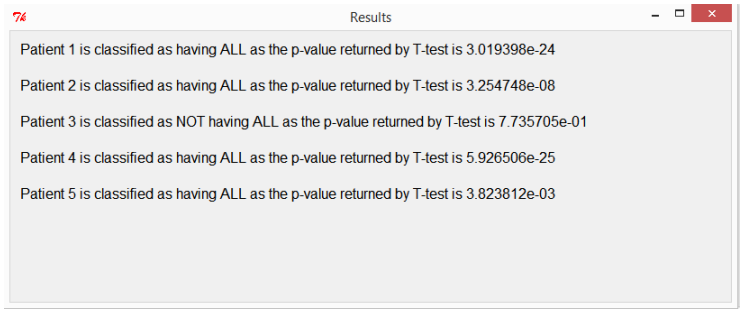

Figure 17: Classifying new patients