Georgia State University

ScholarWorks @ Georgia State University

Computer Science Dissertations | Department of Computer Science

5-2-2008

Design of a Structure Search Engine for Chemical Compound Database

Hao Wang

Follow this and additional works at: https://scholarworks.gsu.edu/cs_diss

 Part of the Computer Sciences Commons

Recommended Citation

Wang, Hao, "Design of a Structure Search Engine for Chemical Compound Database." Dissertation, Georgia State University, 2008.
doi: https://doi.org/10.57709/1059443

This Dissertation is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.

DESIGN OF A STRUCTURE SEARCH ENGINE

FOR CHEMICAL COMPOUND DATABASE

by

Hao Wang

Under the Direction of Robert Harrison

ABSTRACT

The search for structural fragments (substructures) of compounds is very important in medicinal chemistry, QSAR, spectroscopy, and many other fields. In the last decade, with the development of hardware and evolution of database technologies, more and more chemical compound database applications have been developed along with interfaces of searching for targets based on user input. Due to the algorithmic complexity of structure comparison, essentially a graph isomorphism problem, the current applications mainly work by the approximation of the comparison problem based on certain chemical perceptions and their search interfaces are often e-mail based. The procedure of approximation usually invokes subjective assumption. Therefore, the accuracy of the search is undermined, which may not be acceptable for researchers because in a time-consuming drug design, accuracy is always the first priority. In this dissertation, a design of a search engine for chemical compound database is presented.

The design focuses on providing a solution to develop an accurate and fast search engine without sacrificing performance. The solution is comprehensive in a way that a series of related problems were addressed throughout the dissertation with proposed methods. Based on the design, a flexible computing model working for compound search engine can be established and the model can be easily applied to other applications as well. To verify the solution in a practical manner, an implementation based on the presented solution was developed. The implementation clarifies the coupling between theoretic design and technique development. In addition, a workable implementation can be deployed to test the efficiency and effectiveness of the design under variant of experimental data.

INDEX WORDS: Search engine, chemical compound, compound database, substructure comparison, compound representation.

DESIGN OF A STRUCTURE SEARCH ENGINE FOR CHEMICAL COMPOUND DATABASE

by

Hao Wang

A Dissertation Submitted in Partial Fulfillment of Requirements for the Degree of

Doctor of Philosophy

In the College of Arts and Sciences

Georgia Stage University

2007

Copyright by

Hao Wang

2007

DESIGN OF A STRUCTURE SEARCH ENGINE

FOR CHEMICAL COMPOUND DATABASE

by

Hao Wang

Major Professor:	Robert Harrison
Committee:	Raj Sunderraman,
	Yan-Qing Zhang
	Irene Weber

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2007

Acknowledgments

Firstly, my specific thanks go to my advisor, Dr. Robert Harrison, for his guidance and precise advisement during the process of my PhD dissertation. The dissertation would not have been possible without his help.

Secondly, I would like to thank my committee members, Dr. Raj Sunderraman, Dr. Yan-Qing Zhang, and Dr. Irene Weber for their well-appreciated support and assistance.

Finally, I want to thank my family and friends for their support and beliefs.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Acronyms

AGPM	Atom Group Partition Mapping
LOB	Large Object Byte
BFS	Breath First Search
CGDS	Compound-Group-Distribution-String
QSAR	Quantitative structure-activity relationship
JDBC	Java Database Connectivity
JSP	Java Server Page
JSTL	Java Server Page Standard Tag Library

Chapter 1 Introduction

1.1. Chemoinformatics, chemical compounds and database queries

Chemoinformatics is the study of the use of databases in handling chemical knowledge. Chemoinformatics, unlike bioinformatics, focuses more on small molecules and a wider range of molecules rather than genes and gene products. It serves a critical role in the development of new materials and new pharmaceuticals by aiding in the selection of starting points for experimental development [2, 24, 59] . As in bioinformatics, many new structures along with their chemical properties are published annually resulting in a huge mass of data that has to be organized into a database for efficient search and recall [8-10, 12, 13]. Traditional relational database engines like Oracle are required for performance because of the volume of data. However, the properties of the data do not map directly into the purely numerical and string based data types the relational database engines are designed to handle. Therefore one important problem in chemoinformatics is the development of efficient representations of the chemical and physical properties as well as the structures of molecules. Intimately related to the development of the representation of molecular properties is the ability to compare molecules and extract which ones are most similar in some sense [1, 2, 5-8, 10, 48]. The ideal representation of chemical and structural data would allow for the rapid and highly specific recall of molecules which are similar in structure and properties. Current approaches tend to be either rapid and imprecise or precise and relatively slow [8, 14, 15, 24, 28, 33, 34, 39, 48, 53, 54]. Therefore the more accurately the chemical information

can be represented in the native representation for the database engine the more the overall system meets this ideal.

Typically there are three kinds of queries that are applied in chemoinformatics: shared substructure, similar subset, and molecular property. In a shared substructure query, molecules are selected that share a chemical group or structural framework but differ in other features. For example, aspirin and benzoic acid share a benzene ring and carboxylic acid group but do not share the phenol oxygen and acetyl group of aspirin. In a similar subset query, features that are in common among a set of molecules are extracted and then used to find similar molecules. Superimposing HIV protease inhibitors, for example, would reveal that they share many structural features that would not be readily apparent on casual inspection.[20, 35, 58] Finally, with molecular property queries, molecules are selected based on a desired chemical feature or property. An example of this would be the selection of hydrophobic monomers for the design of a novel water repelling polymer.

1.2. Design goal

The search for structural fragments (substructures) of compound is very important in medicinal chemistry, QSAR, spectroscopy, and many other fields [8, 17, 18, 51]. In the last decade, with the development of hardware and evolution of database technologies, more and more chemical compound database applications have been developed along with interfaces of searching for targets based on user inputs[4, 8, 11, 17, 18, 21-23, 25, 27, 30, 36-38, 40, 41, 44, 51]. Due to the computational complexity for structural comparison, essential a graph isomorphism problem, the current applications mainly

work on the approximation of compound structure problem [12, 21, 23, 40, 43]. The procedure of approximation usually invokes subjective knowledge and consequently there exist many different approximations and their accuracies are usually closely associated with the inputs [38, 41, 44].

In this dissertation, a search engine for chemical compounds is designed and developed with a focus on accuracy. The design is devoted to providing a computing model of developing an approximation free substructure search engine for chemical compound database but maintaining a high speed processing capability at the same time. The design goal is comprehensive in the sense that such a goal has to deal with a series of subproblems. Most of them are interesting research topics in themselves. In this dissertation, however, those problems are explored under one goal: to design an efficient and accurate compound search engine. As a result, the strategies and approaches for different problems are inherently connected and consistent, they as a whole contribute to the achievement of the final goal.

1.3. Problem definitions

The goal involves a series of subproblems in which thoughtful consideration and efficient solution is demanded. The description and definition of problems using computer science language helps to make them easy to understand and discuss.

The critical problems that are explored in the dissertation are listed as follows. The coverage and resolution of those problems can directly lead to a design of backbone computing model of a compound substructure search engine and any further problems proved to be essential can be appended to the list when they are encountered.

- Computer data representation of compound structure
- The efficient substructure comparison algorithm
- The structure comparison algorithm under main-frame database system
- Logic preprocessing (screening) for compound structure search engine
- Interactivity in search engine and its application in web interface

The dissertation is intended to present practical and efficient solutions for the above problems. The solution for one problem may not be the best in an isolated situation, but its connectivity with other solutions may prove its advantage in the overall application design.

1.4. Organization

The dissertation is organized as follows: In the chapter 2, we propose a bond partition based compound data representation. Based on the data representation, a new substructure comparison algorithm can be developed to efficiently process compound structure comparison problem. Chapter 3 focuses on the database design, it includes data representation persistence in a database, domain specific query layer, logic screening, and the integration of data, algorithm and implementation under database infrastructure. In the next chapter, a system implementation of search engine is demonstrated. The implementation is designed to make a workable version and also addresses some realistic problems under system level development. The implementations are based on the design model and are used to test its efficiency and effectiveness. By feeding experimental data and analyzing the result of output, the model can be refined and some conclusions of the

design can be drawn. In last chapter, a general conclusion and future work of development are provided.

Chapter 2 Data Structure and Algorithm for Structure Comparison

2.1. Introduction

Chemoinformatics is the study of the use of databases in handling chemical knowledge. Chemoinformatics, unlike bioinformatics focuses more on small molecules and a wider range of molecules rather than genes and gene products. It serves a critical role in the development of new materials and pharmaceuticals by aiding in the selection of starting points for experimental development. One important problem in chemoinformatics is the development of efficient representations of the chemical and physical properties as well as the structures of molecules [46, 47, 55]. Intimately related to the development of the representation of molecular properties is the ability to compare molecules and extract those that are most similar in some sense. One way of doing this is to calculate the structure fragment relationships among molecules. The ideal representation of chemical and structural data would allow for the rapid and highly specific recall of molecules which are similar in structure and properties [49, 50, 52-54, 56, 57].

In this chapter, a new graph representation of molecules is presented. The representation is designed by taking close consideration of special features of chemical compounds. The resulting chemical-context based graph representation hence contains more information in its data representation. Based on the proposed representation, a new substructure comparison algorithm is also presented, the introduction and analysis of algorithm demonstrates its strong efficiency in solving structure comparison problem.

2.2. Theoretical background and related work

It is widely agreed by chemists and biologists that a chemical molecule can be efficiently supported by a topological graph and a great range of hidden chemical properties of the molecule can be further derived from that graph representation. Therefore the problem of determining structure fragment relationship between two molecules can be solved by using graph-based algorithms. In graph theory, it is officially named as graph isomorphism which is believed to a NP complete problem. A mathematical description of graph isomorphism problem can be illustrated as follows [1]:

Input Description: Two graphs, *g* and *h*.

Problem: Find a (all) mappings *f* of the vertices of *g* to the vertices of *h* such that *g* and *h* are identical, *i.e*. *(x,y)* is an edge of *g* iff *(f(x),f(y))* is an edge of *h*.

Given graph *g(1, 2, 3)* and graph *f(a, b, c)*, where *1, 2, 3, a, b, c* are the vertices of two matching graphs. The search of mapping(s) is equivalent to exploring the permutation tree of Fig.2-1. Without prior knowledge, the worst case search needs to visit all the paths among root and leaves. The total number of paths amounts to a function of the permutation of *m* and *n*, where *m* and *n* are the number of vertices of graph *g* and *h*. Obviously, the number of paths goes beyond the polynomial range.

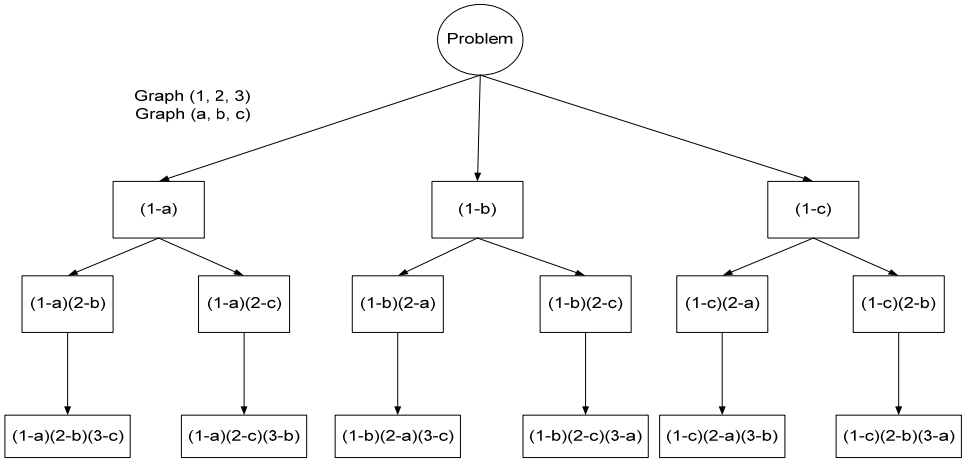


Figure 2-1 A permutation tree for solving graph isomorphism problem

In practical drug design, the application is considerably more complex than only one pairwise comparison as an input chemical molecule needs to be compared over a large collection of potential candidates in the database. The whole operation performs the pairwise comparison *n* times if the search has to be repeated against *n* potential candidates.

The interest to solve isomorphism problems efficiently started in the 60's when it was discovered that a chemical compound can be represented by a graph [1]. A back-track mechanism was proposed to compare two structures by a searching route [8, 11, 14, 51]. In the early 70's, Ullman proposed a backtrack algorithm for generalized isomorphism, which even today proves to be one of most efficient algorithms targeting the problem [44]. The algorithm can be applied to both substructure and maximum substructure problems and subsequent research demonstrates the average running time of this algorithm falls into an acceptable polynomial time complexity range. The general graph isomorphism problem involves, by its very nature, heavy computation.

Consequently, in spite of tremendous research efforts directed towards this area, the Ullman algorithm still remains one of the best graph-based isomorphism algorithms.

Later on, it was found that although a molecule can be represented as a complete graph, fragments or substructures of the molecules may also carry chemical connotations which may not be fully defined in graph representation [39, 49, 57]. Therefore, defining a series of chemically sensitive fragments, followed by a graph representation with existence indication of various fragments becomes another viable trend. As the number of fragments and their connections is much less than the number of atoms and bonds, a set of chemical sensitive fragments and their relationships can be used as molecular fingerprints instead of defining molecules using complete graphs thereby simplifying the computation. However, the selection of sensitive chemical fingerprints working for this purpose is never easy to establish even for a particular case. The wide application of this approach is thus restricted by the availability of valid fingerprints targeting various cases. In addition, the fingerprint does not record complete connectivity of chemical molecule and, as a result, the search result is not as accurate as graph-based approach. A basic rule for these applications is that the experimental use of these molecules in a real chemistry lab may take months or even years so it is important not to be overly aggressive at trimming processing speed at the cost of missing critical leads.

Another trend for substructure searching is to use those soft modeling methods [19, 30], such as genetic algorithm (GA) [45]. GA has widely demonstrated its potential in the attempts to solve NP problems and has been applied toward this problem for a long

time. However, the approximate and nondeterministic nature of GAs means that they should not be used when a feasible, conventional algorithm is available [51].

2.3. Graph-based data representation of molecules

A small molecule structure is mostly described by a topological graph. A graph is called topological when it shows only the linkages between atoms and the type of bonds between them. It is already proven and practically applied that graph representation is a powerful tool for studying chemical structure problem. However, when working on graph data and using subgraphs as patterns, the computations are very expensive due to absence of any polynomial algorithm to solve the graph isomorphism problem. Current algorithms for subgraph isomorphism feature exponential time complexity [1, 34].

The structure of a chemical molecule involves atoms and connection between pairs of atoms. In the graph representation of molecular structure, the node is used for atom and the edge is set by their connection. Although, the connectivity information is conserved in the graph representation, it carries no classification information. As a result the search routine can not speed up by taking that information into account.

Unlike general graph, node and edge in the context of a chemical structure graph are much more restrictive and have special meanings: A node presents an atom which in theory comes from an element in the periodic table, and an edge is defined to present a numeric connection value bounded by a constant value. In chemical parlance, the pairwise atoms and their connections together are called bonds and a molecule structure graph is actually constructed by a collection of bonds. The chemical definition of molecular bonds can be used to classify them into limited number of groups/partitions

2.3.1 Bond-partition based molecular representations

A new way of representing a molecule can be made based on bond partition. In this method, instead of representing atoms by their chemical symbols, atomic numbers are used. The goal here is to classify the chemical bonds in a molecule into different bond types. The significance of this design is that these atomic values can be further used in calculations that are specific for each type of atom. In this design, a pair of atoms and its bond order (collectively referred to as a bond type) are represented by 7 digits (Figure 2-2). The first and the second 3 digits are the atomic numbers of the first and the second atoms, respectively. The last digit represents the bond order between the two atoms. In practice, several issues must be considered when converting the molecule into a graph. An example is the representations of the ring system. Aspirin, for instance, can have three different graphs that are chemically equivalent (Fig. 2-3). One way of representing an aromaticity within a ring is to define the bond value of aromatic system to be a specially fixed value. In the numeric representation this value is defined as 9. This rule is enforced in all the inputs and data in the chemical database to ensure structural compatibility during structure comparison.

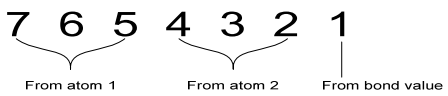


Figure 2-2 Numeric representation of bond type

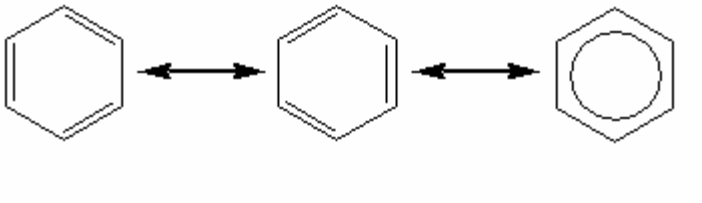
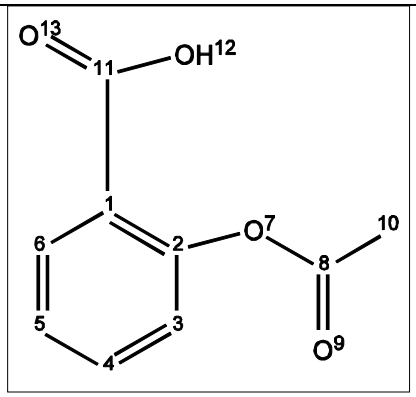


Figure 2-3 Common representation of aromatic ring

This design is based on the fact that each of the atoms has a unique atomic number. The largest atom in the periodic table is 118; thus, it is more than enough to have a 3-digit to present that numeric value. In this descriptor, two atoms are arranged in an ascending order of their atomic numbers. For example, a bond type of a double bond between a carbon and an oxygen can be defined as 0060082. In this representation, the numerical value 6 is the atomic number of the carbon, 7 is the atomic number of the oxygen, and the last digit 2 represents the double bond between the atoms. In a given molecule, a bond type is determined for each of the bonds in the molecule. The bond types are then collected into a ***bond group***, based on the identities of the atoms that are involved in the bonding and the bond order. Each ***bond group***, therefore, consists of bonds with identical pair-wise atoms and connection value. The atoms in the group are represented by their positions *(x, y)*. *x* represents the position of the atom with a lower atom number, and *y* represents the position of the atom with a higher atomic number. A bond partition of the aspirin graph is shown in Figure 2-4.



6-6-1

(1,11) (8,10)

6-6-9

(1,2)(2,3)(3,4)(4,5) (5,6)(1,6)

6-8-1

(2,7)

6-8-2

(8,9)(11,13)

Figure 2-4 Revised graph representation of aspirin

The group appearing times of each atom in correspondence with each bond group can be further derived and we have named that as Atom-Group-Partition-Map (AGPM). AGPM is described as a two-dimensional table. In the table, each column designates a bond group and each row represents an atom. A cell of the table holds an integer value defined as the frequency at which the atom (defined by the row index) appears in a bond belonging to a particular bond group (defined by the column index). A sample of the AGPM table for chemical molecule of figure 2-4 is illustrated as follows:

Table 2-1 AGPM table for sample chemical molecule

	6-6-1	6-6-9	6-8-1	6-8-2
1	1	2	1	
2	1	1		
3	1	1		
4	1	1		1
5	1	1		
6	1	1		
7			1	
8	1			1
9				1
10	1			

2.4. Algorithm

The proposed new representation suggests that molecule bonds of a molecule can be logically partitioned into different groups (partitions). As a result, graph representation of molecule can be revised to reflect more chemical background.

The introduction of bond partition provides more chemical context information that can be used in solving molecule isomorphism problems. For a substructure comparison problem: which is, given graph *g* and *h*, determine if *g* is a substructure of *h*. Some statements can be derived after logic partition.

- After the partition, original permutation problem is equivalent to several small-size permutation problems while boundary conditions are satisfied
- A valid permutation mapping exists **only if** at least a valid permutation mapping exists between any two matching partitions of molecules
- The exploration of permutation tree and pruning of unqualified branches can be simplified by taking account of additional partition context.

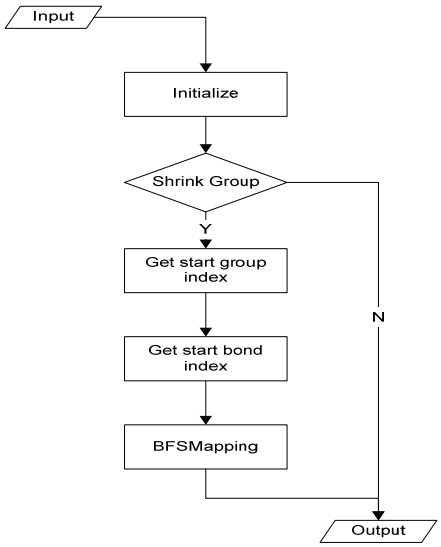


Figure 2-5 Flowchart of the algorithm

Consequently, a new graph algorithm for molecule substructure problem is presented as follows. Without loss of generality, the input of algorithms is two chemical molecules and they are represented by the format introduced in previous section. The entire workflow of algorithm can be illustrated by a flowchart (fig. 2-5) and the algorithm takes five steps:

2.4.1 Initialize

The algorithm not only tells if there is a substructure relationship between two molecules, but also shows the matching atom mapping between two molecules. If more than one mapping exists, all of them will be returned. This step is designed to define necessary variables and specify their initial value.

The algorithm defines a mapping by the format of an integer array. The array stores the matching atom positions of molecule2. For an element in the array, assume the index of element as x, and the value of element as y, where both x and y are integers and array index starts from 1. The element indicates a mapping relationship that the atom of

xth position in molecule1 is mapped with the atom of *yth* position in molecule2. The length of array is determined by the minimum logic of atom size of molecule1 and molecule2. The array is initialized to -1 for every element, which means mapping has not been established yet.

2.4.2 Shrinking of partitions

The step directly comes from the statement "For substructure comparison, a valid permutation mapping exists for two molecules **only if** a valid permutation mapping exists between any two matching bond groups of two molecules". The statement implies that if there is a mapping between two molecules, the partitions of the smaller molecule must be a subset of those of the bigger one. Any partitions in the bigger molecule which are not in the smaller molecule play no role in the final mapping and thus are safe to discard.

As a result, assuming molecule1 is the smaller molecule, the first activity of this algorithm is to determine if partitions of molecule1 are a subset of partitions in molecule2. Negative "no" means no further processing is necessary and the algorithm returns with the output of no mapping. For the positive answer, the algorithm continues with elimination of those partitions in molecule2 that are not found in molecule1.

After this step, two input molecules are the same in terms of their partitions and their indices in the two molecule representations.

2.4.3 Finding the starting partition index

The algorithm needs to determine which bond to start mapping with and it takes two steps. In step 3, the partition index of starting bond is resolved. The basic rule dictates that the starting partition be the one that has least permutation of bond mapping.

Assume, for *x*th partition, molecule1 contains *n* bonds while molecule2 has *m* bonds. The possible permutation among bonds of two molecules is defined as follows:

$$permutation_x = (m) * (m-1) * ... (m-n+1) * (isTwoAtomsSame ? 2 : 1) \quad \textbf{(2-1)}$$

Thus the search for the partition index of least bond mapping permutation is done by calculating possible bond permutations for all index partitions. The index of the first partition producing the lowest permutation value is the output for this step.

2.4.4 Finding the starting bond index

The mapping of the algorithm starts with a chosen bond from a partition generated by the previous step and the starting bond index of that bond is calculated in this step. The basic rule for the starting bond is to locate a bond with the most complicated bond partition distribution. In previous section, the concept of Atom-Graphic-Partition-Mapping is introduced. Based on AGPM, a numeric value, judging the complexity level of the atom, can be calculated as follows: where *x* and *y* refer the row and column index of AGPM.

$$atom(x,y) = AGPM(x,y) == 0 ? -1 : AGPM(x,y) \quad \textbf{(2-2)}$$

$$atom(x) = \sum_{y=1}^{column(AGPM)} atom(x,y) \quad \textbf{(2-3)}$$

The algorithm conveys a proposed complexity definition: the more diverse neighbors an atom connects, the more complicated the atom is.

The complexity definition can be extended to a bond by considering both atoms.

$$bond = atom(x) + atom(y) \quad \textbf{(2-4)}$$

Consequently, the search for the most complicated bond can be taken by calculating complexity values for all the bonds in given partition, the first one with highest value is the final output.

2.4.5 Breadth first search mapping (BFS mapping)

This step is the core part of the whole algorithm. The name BFS Mapping comes from the fact that the whole mapping procedure can be visually represented by the generation of a mapping forest with a breath first search order (BFS).

Some definitions are given first to make it easy to illustrate the algorithm.

2.4.5.1 Definitions and terms

- **Mapping forest**

A mapping forest embodies the mapping abstraction between two molecules. The node of the forest stands for an atom mapping and is valued as a position pair *(x, y)*, where *x* and *y* are mapping atom positions of molecule1 and molecule2. A sample of mapping forest is depicted as follows (fig. 2-6). Note here, due to possible existence of more than one substructure mapping between two molecules, the BFS mapping may generate more than one mapping forest.

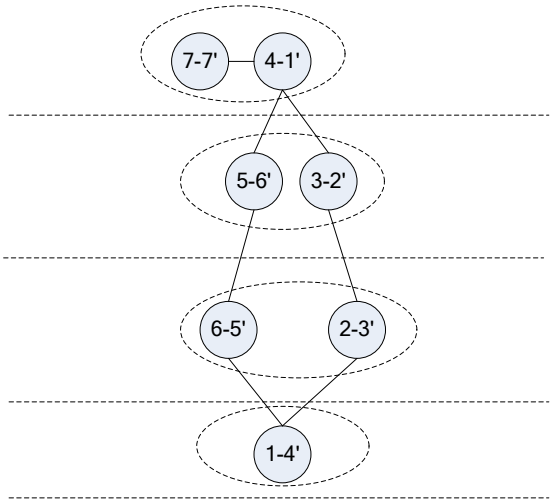


Figure 2-6 A sample mapping forest

♦Unmapped nodes and mapped nodes

The nodes in the mapping forest have two stages: mapped and unmapped. The left node in figure 2-7 is a mapped node and right node is an unmapped node. The difference between these two lies in the fact that the mapped node has already located position of the mapping atom in molecule2 while unmapped node has not.

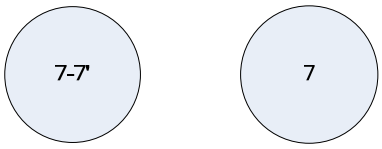


Figure 2-7 Mapped node and unmapped node

♦**Active nodes**

Active nodes are a set of mapped nodes which are currently acting as roots for the next round of mapping forest generation. The active nodes involve both operations that mapping forest may have, which will be discussed in the following section (Fig. 2-8).

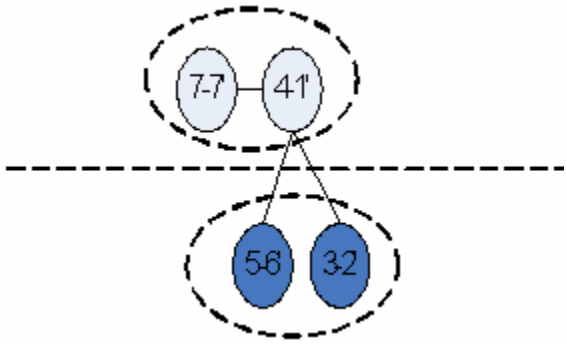


Figure 2-8 Active nodes (colored dark blue in the figure)

2.4.5.2 Operations

The generation of mapping forest involves two operations, *bond locating* and *bond mapping*.

♦**Bond locating**

The purpose of *Bond locating* is to expand the current mapping forest into next level. In the design, the input for the *Bond locating* is the set of current active nodes in the mapping forest and those atoms of the input molecule (molecule1) are extracted from active nodes to work as the root nodes. *Bond locating* searches each of those atoms for its existing direct neighbors through the support of AGPM table and bond-partition structure. Those neighbor atoms will be constructed into the unmapped nodes and appended into the mapping forest as the direct children of initial active node. If *bond*

locating can not expand any active nodes, the whole algorithm stops with a success for establishment of substructure mapping.

The whole operation can be illustrated by following example, assuming there is one active node 4-1, as shown in figure 2-9. The atom 4 comes from the input molecule and is used to search for unexplored direct neighbor atoms. From the current AGPM table, atom 4 appears once in an unexplored bond residing in bond group 6-6-1. Then by a search against bond group 6-6-1, it is observed that bond (4, 5), which contains atom 4, is the bond we are looking for.

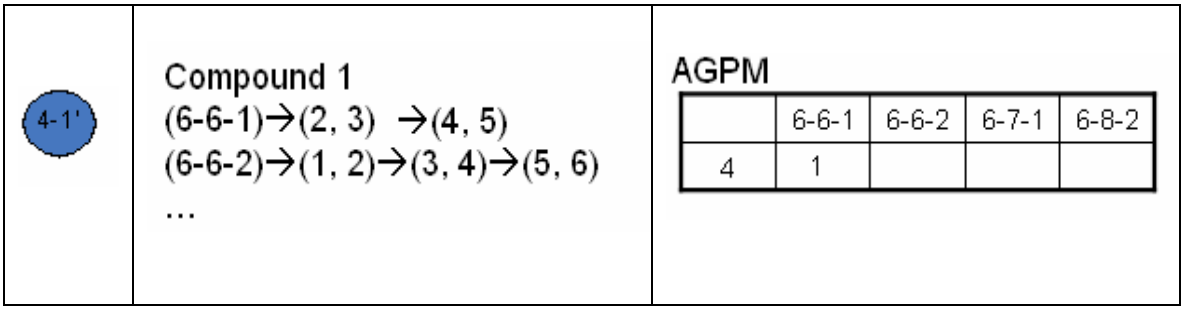


Figure 2-9 An example of bond locating operation (a)

The new explored bond (4, 5) will be added into mapping forest and the bond group and AGPM table are updated to reflect the progress, as illustrated in figure 2-10.

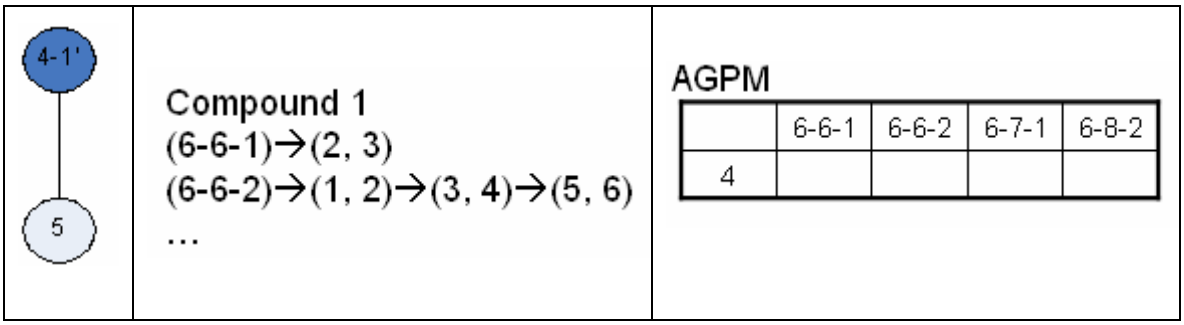


Figure 2-10 An example of bond locating operation (b)

♦**Bond mapping**

Bond mapping is designed to fill the mapping of the atoms. However, instead of searching for a mapping atom based only on atom types, the operation should function under a bond based context.

The *bond mapping* normally follows *bond locating* operation. There exists one exception, however, which is the very first *bond mapping*. As introduced before, mapping forest starts with a starting bond which is constructed by two unmapped nodes. The *bond mapping* of these two unmapped nodes is made by applying the same selection rule onto target molecule: 1) the mapping bond in target molecule has to come from the bond group that has the same group index of starting bond. 2) The complex value of mapping bond has to be at least the same as that of starting bond. By calculating complex value for each bond in candidate bond group, the bond with satisfied value is chosen to be the mapping bond for the starting bond. It is possible there is more than one bond that is qualified for *bond mapping*. The bond mapping responds to that by spawning more mapping forests to reflect those additional mapping choices. Although in practice, that possibility is fairly rare due to the designed selection.

Bond mapping mainly operates just after *bond locating*. Here *bond mapping* works towards those bonds that have a mapped node and an unmapped node, and then the operation settles the mapping atom for that unmapped node. (Fig.2-11). The *bond mapping* thus works under a restriction: The mapping bond in target molecule has to be in a bond group with same group index as the bond in input molecule and one atom end of that bond is already determined. The operation can be best illustrated by a sample as

in Figure 2-12 (the bond (4, 5) of input molecule is in bond group 6-6-1, which is not identified in the figure). Bond mapping locates the AGPM table with the cell that corresponds to the group 6-6-1 and atom 1. The cell has value 1 which assures there is one qualified mapping bond, as in the *bond locating*, the mapping atom for that bond in mapping forest can be determined by searching against bond group 6-6-1 and both APGM and bond group are updated as shown in figure 2-13.

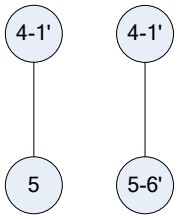


Figure 2-11 Bond mapping operation

The lookup cell may have value bigger than 1. The *bond mapping* may be ambiguous when a new mapping forest has to come into existence. The new mapping forest will be the exact same except for one node difference which amounts to the other mapping choice for the *last bond mapping* operation. The algorithm has a stack to hold all newly generated mapped forests for future processing. The generation of new mapping forest can happen in any bond mapping operation. The intention to use a first-in-last-out stack is to make those more matured mapping forests processed before the others.

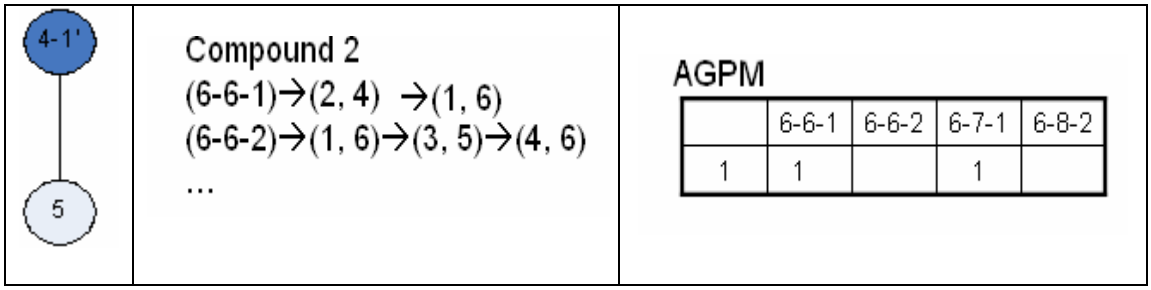


Figure 2-12 A sample of Bond mapping (a)

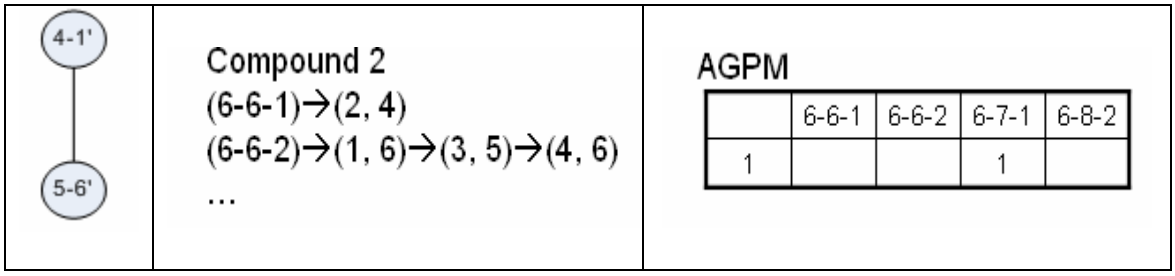


Figure 2-13 A sample of Bond mapping (b)

The general bond mapping operation can be defined as following pseudo code:

```
/*   BondMapping:
Input:
A mapped node (inputNode1), an unmapped node inputNode2), a global stack (forestStack) for new generated   mapping forests, and current AGPM for molecule2    (AGPM2)
Output:
    One mapped node which is generated from unmapped           node (inputNode2). If more than one possibility    exists.   New   mapping forest will be generated based          on    the    current    mapping forest. AGPM2 will be       modified during the procedure to reflect the generation of new mapping
*/
       1. Atom parentAtom1 = atomInMolecule1(inputNode1)
       2.     Atom childAtom1 = atomInMolecule1(inputNode2)
       3.     int groupIndex  = groupOf(parentAtom1, parentAtom2)
       4.       Atom parentAtom2 = atomInMolecule2(inputNode1)
       5.       Array groupDistributionParentAtom2[] =
       6.             groupOfAtom(parentAtom2)
       7.       int distributionParentAtom2 =
```

```
8.              groupDistributionParentAtom2[groupIndex]
9.        List bondList = bondListInMolecule2(groupIndex)
10.       for (int i = 0; i < distributionPaneAtom2; i++)
11.               Bond bond = bondList.next()
12.              if (bond.contain(parentAtom2))
13.                  if (i == 0)
14.                        inputNode2.setAtom(
15.           bond.anotherAtom(parentAtom2))
16.       AGPMUpdate(AGPM2)
17.       elseif
18.           MappingTree newTree = clone(currentTree)
19.           newTree.inputNode2.setAtom(
20.               bond.anotherAtom(parentAtom2))
21.           AGPMUpdate(AGPM2)
22.           treeStack.push(newTree)
```

With the introduction of above definitions, the BFS Mapping can be defined as a procedure to generate mapping forest for two input molecules. The procedure works in a loop, which first makes the bond locating based on active nodes, followed by the bond making and reassignment of the active nodes. The BFS Mapping succeeds when no further bond locating can be made for current active nodes, which indicates a substructure mapping between two input molecules, or fails when no bond mapping can be established. The whole operation is illustrated by a sample case illustrated in figure 2-14.

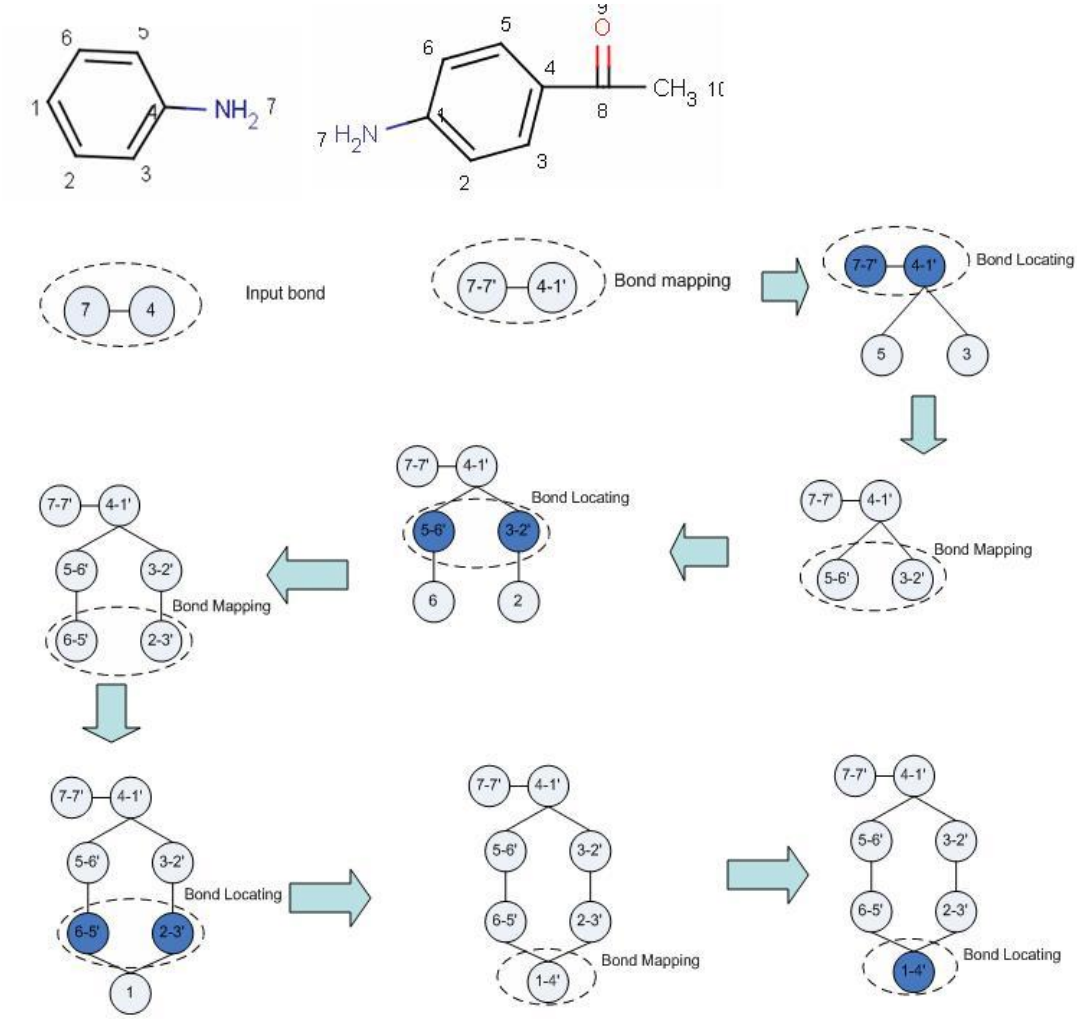


Figure 2-14 Illustration of BFSMapping

2.5. Conclusion

Based on a new bond-partition based graph representation, the proposed algorithm fully explores the specific chemical structure classification of a molecule. The algorithm starts with a pair of mapping bonds by fully exploring bond distributions between two molecules. With the introduction of mapping forest abstraction, the search routine is logically simplified with two repeatedly interweaved operations: bond locating and bond mapping. The two operations are inherently straightforward and involve little

computation. The implementation and experimental results demonstrate that the algorithm yields high performance in most of the test cases and hence is efficient in solving the molecular substructure comparison problem. The basic idea of data representation and algorithm can be applied to other graph isomorphism fields as long as a valid classification can be drawn.

Chapter 3 Search Engine and Database

3.1. Chemical database

Chemical databases store chemical structures and associate information. The sizes of the structures stored in databases can range from those of small molecules, as in the Cambridge Structural Database and Inorganic Crystal Structure Database, to those of macromolecules such as proteins and nucleic acids stored in the Protein Databank (PDB) database, Table 3-1. Other databases, such as Quantum Chemical Literature Database, store information from the literature regarding chemical properties that can be used in the analysis. Other useful chemical databases include PubChem, KEGG LIGAND Database, ChemIDplus, Indiana University Molecular Structure Center, NCI-3D Database, and Chmoogle .

Table 3-1 A list of some of the common chemical databases used in chemoinformatics.

Data Source	Web Sites
Cambridge Structural Database	www.ccdc.cam.ac.uk.
Databases on STN International	*http://www.stn-international.de/stndatabases/c_datab.html*
Protein Database	www.rcsb.org.
NCBI PubChem	http://pubchem.ncbi.nlm.nih.gov
KEGG LIGAND Database	http://www.genome.jp/ligand
National Library Medicine Specialized Information Service	http://chem.sis.nlm.nih.gov/chemidplus/
Indiana University Molecular Structure Center	*http://www.iumsc.indiana.edu/database/index.html*
TOXNET (Toxicology Data Network)	*http://toxnet.nlm.nih.gov/cgi-bin/sis/htmlgen?Multi*
eMolecules Chemical Searching	http://www.chmoogle.com/

3.1.1 Chemical characteristics and indexing of database

Relational databases are the prevailing type of databases used to store chemical information. They are powerful tools for organizing information; however, they are designed to handle numeric and string data rather than chemical structures. Therefore, the characteristics of the chemical structure must first be converted into a representation using strings and digits that is in turn, stored inside the database. Clearly the choice of this transformation will affect the flexibility and accuracy of the recall process. The characteristics can be any of the properties and features of the molecule. This method of labeling, or indexing, of the characteristics allows the information to be stored efficiently in the relational database. Another purpose of indexing the database is to pre-determine the solutions to some of the expected queries to shorten the response time. For instance, the result returned by popular search engines (such as Google, Yahoo, and MSN) is an indexed portion of pre-search pages. This allows the viewers to access the pages without having to wait for tedious calculations.

In chemical databases, indexing can be used to label chemical features. Two methods that have been used to index the chemical databases are a fragment code and a fingerprint.

The fragment code method [10, 49] is sometimes referred to as a structural key method. In this method, the molecule is broken down into fragments with pre-defined patterns. Each of the patterns represents a characteristic (such as atomic content, functional groups, ring systems, etc.) of a molecule. Depending on the molecules, different types of fragment codes can be defined. String representations, like SMILES,

are well suited to this decomposition. Unlike the canonical structures, fragment codes can be ambiguous, and different structures could possess identical fragment codes. This is because the code does not specify the connection orientation. The key in designing a fragment code dictionary is to first determine the type of search that will be performed on the databases. This is to optimize the search performance by eliminating the irrelevant molecules, which, in turn, reduces the searching time. It is also necessary to design the dictionary according to the compounds stored in the database and the queries that might be submitted. Although any type of chemical features and queries maybe used, there are certain types that are frequently encountered.

A fingerprint method [49, 57] describes properties and characteristics of a molecule using a string of binary digits 1 and 0: 1 represents a positive response and 0 representing a negative response. The string can be of any size, which allows as many chemical features and properties of a molecule to be expressed. A fingerprint of a benzoic acid, for example, can be 111 for the presence of a benzene ring, a carbonyl, and a hydroxyl group, respectively. If a second molecule with a fingerprint of 011 is compared with the acid's fingerprint, the difference between the two fingerprints indicates that the second molecule contains a carbonyl and a hydroxyl, but no benzene ring. Because of its flexibility, the fingerprint method is often used as a similarity measurement tool between structures and/or in a substructure searching routine during the screening of the molecule databases.

3.1.2 Prevailing design of search engine for chemical database

Search engine queries the chemical database for potential targets and uses search algorithm to determine their structural relationships. A non-trivial chemical database contains a large volume of chemical compounds thus speed must be a concern for the design of search engine.

To meet the speed requirement, in the prevailing design of search engine, instead of storing exact chemical structures into database, fragment codes or fingerprints of molecules are saved in the chemical database. The search of potential targets in the database only invokes the comparison of string or numeric representation of molecules, for which traditional database engines are designed. The design, however, reduces the accuracy because not all molecular properties were kept in the database and are used for the target filtering. .

3.2. Compound data representation in database

An efficient search engine often requires a close coupling of the search algorithm and database implementation. For a search application based on chemical compound database, the database must process the data representation of chemical compounds based on the search algorithm. Advanced algorithms often introduce data representations which are not directly supported by mainframe databases, which imposes a practical constraint on the implementation of those algorithms onto the database. In addition, the current database only supports relationships on pre-defined data types, which greatly impedes the application of domain specific queries onto database. A general design for seamlessly coupling data representation used by various algorithms and mainframe database systems

is presented in this section. The purpose of the design is to eliminate the algorithm dependency on the database in terms of data representation and definition of relationship

3.2.1 Introduction

One of the critical steps in drug discovery and chemical research is compound identification or compound screening[18, 29]. The compound identification problem is equivalent to an operation of a compound search engine in a real application. The procedure finds a structure, a substructure, or a compound fragment in one or more of the structures in a chemical database. Therefore, the approach to represent, store, and retrieve chemical compounds efficiently in a *de facto* database is a key factor for the successful development of a search engine. However, an approach that applies to the general case is not trivial: a) Currently, search algorithms are not constrained to use primitive data types[17, 18, 23, 30, 40, 44]. Therefore, the attempt to apply those algorithms onto database has to deal with how to represent, and save advanced data types in mainframe database. b) The search of relational database is based on relationship, however, there are no predefined relationships regarding chemical compound fragments. Therefore, a direct statement such as “find all compounds that have equivalent structures to that of the input compound A in a chemical database” is not straightforwardly interpretable by database engine. Structure similarity relationship that allows two chemical structures to be compared must be developed and applied to SQL level so that the relationship query can be made.

Some approaches have been proposed to make structure and sub-structure searching feasible in database. These include the conversion of structural compounds to

strings that are easily recognized by database engine. This approach converts a chemical structure to a unique string (Wiswesser Line Notation (WLN), ROSDAL (Representation of Organic Structures Description Arranged Linearly), and Simplified Molecular Input Line Entry System (SMILES)) [17]; hence, a simple string matching can be performed [40]. However, it is obvious that a complex structure, which can be viewed as a graph in computer science, cannot be converted to a string without losing information. For example, two different structures can have the same topologic index representation. In other words, two different compounds may have the same string representation [17, 18].

To solve these problems, in this part, a general method is presented. The method is designed to eliminate data type constraints for potential search algorithms. Based on that, a model to define and apply complicated relationship on top of current relational database is also introduced, which provides a solid foundation for the application of domain specific queries in the current database.

3.2.2 Method

3.2.2.1 Save/Retrieve General Data Type Instances Into/From Oracle Database

A general instance, such as an object instance in Java, cannot be saved directly into the database. This barrier comes from the fact that there may be no corresponding data type in the database that supports such data. Although some databases allow self-defined types, even object types, wrapping a general language instance using the database language is a difficult task because SQL is not as powerful as a general language in this aspect. However, it is still possible to save a general type instance into a database. This

can be done using two techniques: large object (LOB) for a database and instance serialization for a general language.

♦**LOB:** LOB is a data type that is supported by a large number of database management systems (DBMS). The introduction of the data type is for the convenience of saving data with large size, such as movies and/or pictures. The database does not interpret this data type, and LOB can be saved or retrieved as a binary or character stream.

♦**Instance serialization in a general language:** Serialization is primarily used to transfer data from an unstable medium (such as computer memory) into a stable medium (such as a computer hard drive). A program running in the memory must save certain important information into the hard drive to make it available later. This procedure is defined as serialization, which generally is a built-in component in advanced computer languages. Although most general languages support serialization/persistence, they may differ in the implementations of the mechanism. As a consequence, serializing an instance using one language and deserializing the instance by another language is prone to errors. Furthermore, the serialization can be extended to object instance.

These two techniques can be used collaboratively to make it possible to save general type instances into database. The figure below demonstrates the basic idea.

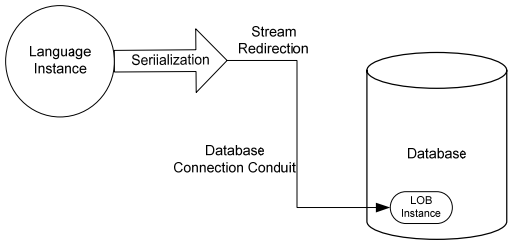


Figure 3-1General type instance serialization as a LOB instance

According to Figure 3-1 the program that has the language instance launches the serialization as usual. However, instead of launching as default that pipes into the hard drive, the instance is redirected into a database from the conduit (some database access interface). The instance flows through the conduit and persists into the destination database as a LOB instance. The retrieval procedure is simply a series of reversed actions, in which the program first obtains language instances as LOB instances. These instances are then deserialized and the cast into the original language type. During the entire execution process, the database does not interpret the language instances. It simply behaves as a unit of storage and provides the mechanism for stream instance to and from the database.

Deserialization and casting must be performed in the same language context. During the serialization steps, the overall procedure may involve a lot of network communication, since serialization and deserialization may occur on different host computers in a client/server model. In addition, a client program must have a prototype definition of the processed instance; otherwise, it will not have the knowledge of how to

make the casting. Oracle database along with Java language encoding offers a perfect platform for the proposed method. The Java language is well known for its capability to represent various objects. Oracle 8i databases, and the later versions, come with an integrated Java Virtual Machine known as the Oracle JVM [31]. The Oracle JVM allows users to deploy and run Java program, known as Java stored procedure/Java stored function (a short Java program), in the Oracle database [31]. With the advent of Java stored procedure, Oracle database provides an alternative to writing business logic in PL/SQL.

We can adapt Java stored procedure to save and retrieve general type instances in Oracle database, as shown below:

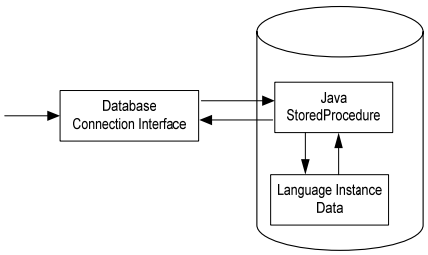


Figure 3-2 using Java stored procedure to save and retrieve general type instance in Oracle database

The Java stored procedure resides inside the database, and has the capability to process all types of instances coded by Java language. For this reason, we choose to use Java language to encode chemical compound representation, which later can be used in a search algorithm. Given the chemical compound prototype, Java stored procedure can serialize and save the representation into a database as a LOB object, as well as retrieve

the LOB object from the database and cast the object into meaningful chemical compound type instance.

3.2.2.2 Building a domain specific language on top of SQL

A chemical compound search engine must know how to handle structural queries, which requires a predefined relationship for comparison of compounds. However, using the traditional SQL commands like “select * from chemicalDatabase where compound = inputCompound” cannot yield the desired results because the traditional SQL can only handle the primitive (string, number, date) datatypes. Therefore, a domain specific language that knows how to process a chemical compounds query will be designed for the search engine.

Because a chemical compound can be represented as a Java object and the search algorithm logic inside the Java object can be easily deployed, Java language can be used to build a domain specific language on top of SQL. A Java class can behave as a domain specific layer on top of the traditional SQL layer. A few member functions of the class, such as isEqual() and isSubStructure(), can be exposed publicly as a Java stored procedure and behave as an interface corresponding to relationships defined in the chemical compound search. The call of Java stored procedure invokes the logic processing wrapped in the Java class, which is coded by Java language and implemented in the class. Java stored procedure can, then, work as a domain specific language parser, converting a domain specific relationship hidden in a basic SQL into a member function call.

A simple pseudocode for accomplishing exact compound match is illustrated as follows:

```
1. Java stored procedure parses the user input and determine it is
   a   exact compound match search
2. Retrieve all compound objects from database, recast them into
   the Java object, and save into an array.
3. for each Compound object
4.     Call member function isEqual(),save the Compound object
       output representations which yield expected result into a
       result array.
5. Return the result array back to end user initiating the query.
```

Other relationships like substructure or similar structure have a similar pseudo code, except for calling different member functions.

The above pseudo code assumes chemical compounds already have been saved into the database as Java objects. This preliminary procedure can be carried out by the following pseudo code:

```
1. Set up a connection with Oracle database
2. for each Compound compound ∈ Mol _ List
3.    do Persistent compound into persistent_compound
4.    save persistent_compound as LOB object in      Oracle
   database
5. end for each Compound
```

Users are allowed to extract as much of the chemical information, such as the functional groups, as they wish from the chemical compounds. This information can then be saved as properties into the compound objects.

3.3. Screening

Search algorithm is used by search engine to determine pairwise compounds' structure relationship. However, in the drug design, which is the realistic background for chemical compound searching, the application is beyond just pairwise comparison of the chemical compounds. Instead, an input chemical compound needs to be compared over a collection of chemical compounds in the database. The practical search is trying to find one or a predefined number of the most consistent chemical compounds. The whole operation expands *n* time longer compared with pairwise comparison if the comparison has to be repeated against *n* candidate compounds.

A straightforward question is do we need to search over all the compound candidates in the database to collect the final targets. The answer is an obvious no. Take substructure search as an example. Given an input chemical compound, the candidate compounds that may contain the input compound have to be at least the same size as the input one in terms of the number of atoms. A search engine for compound substructure can make use of the above logic to speed up the search practice. One design is to associate individual compound table with atom size, a compound table contains only compounds with the same atom size. Therefore, after determining the atom size of a given compound, only compounds in the tables with at least that size need be visited by the search engine.

For a chemical compound substructure search, more logic can be used to improve the search speed. As introduced in chapter 2, a chemical compound can be partitioned into different bond groups. For the chemical compound depicted below, its group distribution is listed as follows:

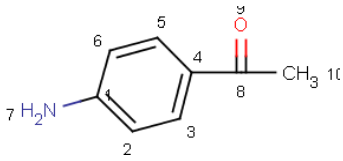


Figure 3-3 A chemical compound structure graph

(Group 1) C-C-1 (60061)→(2, 3)→(4, 5)→(6, 1)→(4, 8)→(8, 10)

(Group 2) C-C-2 (60062)→(1, 2)→(3, 4)→(5, 6)

(Group 3) C-N-1 (60071)→(1, 7)

(Group 4) C-O-2 (60082)→(8, 9)

The group distribution of a chemical compound can be defined as a string with the following syntax:

$$(GroupDescriptor, bondSize)+ \quad \textbf{(3-1)}$$

Where `GroupDescriptor` is the unique integer value associated with each group, and the pairs are sorted descendingly based on their `GroupDescriptor` value.

If we give the name compound-group-distribution-descriptor (CGDS), the CGDS for sample compound is:

(60082,1)(60071,1)(60062,3)(60061,5) **(3-2)**

The CGDS may not be unique for different compounds. However, a compound is a substructure of another **only if** its CGDS satisfies a certain assertion with the other CGDS, and the assertion is:

For two compounds: comound1 and compound2, molecule2 contains molecule1 **only if**: for any matching groups in two molecules,

$$bondSize(x, compound1) \le bondSize(x, compound), \quad \textbf{(3-3)}$$

where x is the group index and two compounds have the same type of groups.

The assertion will apply on searching, which greatly narrows down the potential searching targets. The problem is how to design a data model in the database which is efficiently partitioned to apply the assertion.

In the first example, the data model in database design has a partition which embodies the concept of compound distribution based on atom size. But the partition is not general because a) data model in relational database is limited to represent complicated partitions, b) direct coupling of data model with partitions is not viable since further adjustments or upgrades may be necessary.

Although the concept is clear, the underlying partition which supports that assertion is not easy to implement even without considering database limitations.

3.3.1 Generation and search of distribution tree

Take the CGDS of the above sample as an example, the string describes the group distributions descending from left to right. The assertion needs to be applied to each group type. Therefore, it will be beneficial to have the partitions of underlying structure listed in the same way as the CGDS string. A straightforward but efficient way to describe group information can be made by using tree structure.

If we define the nodes in the tree to be associated with groups, the CGDS string can be represented by a full path from the root to a leaf. The order from left to right is

converted to a parent-child link in the tree (Fig. 3-4). Because the assertion takes two steps, we split the group descriptor and bond size with two adjacent nodes in the tree to simplify the operation. A group descriptor node may have more than one child but the bond size node only has one child node, which is the next level group descriptor node.

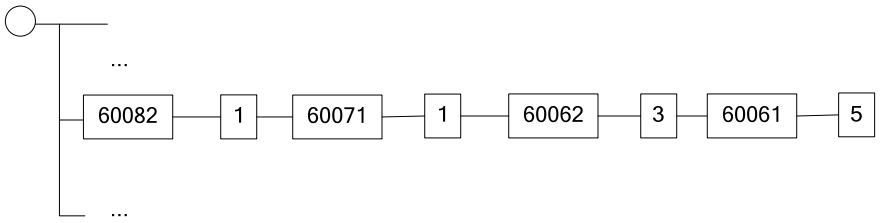


Figure 3-4 The tree representation of sample compound's CGDS

3.3.1.1 Generation of distribution tree

The different CGDS leads to a different path, although two paths may share some common nodes. By our definition, for any two nodes in a tree sharing the same parent, the order of nodes to the parent is determined by their numeric values: the smaller the value is, the lower position the node will be (Fig 3-5). A unique distribution tree can be generated after processing all the compounds in the database. The distribution tree fully represents the compounds distribution based on their CGDS, and it can be further modified if there is any change in the underlying compound data without regenerating the tree.

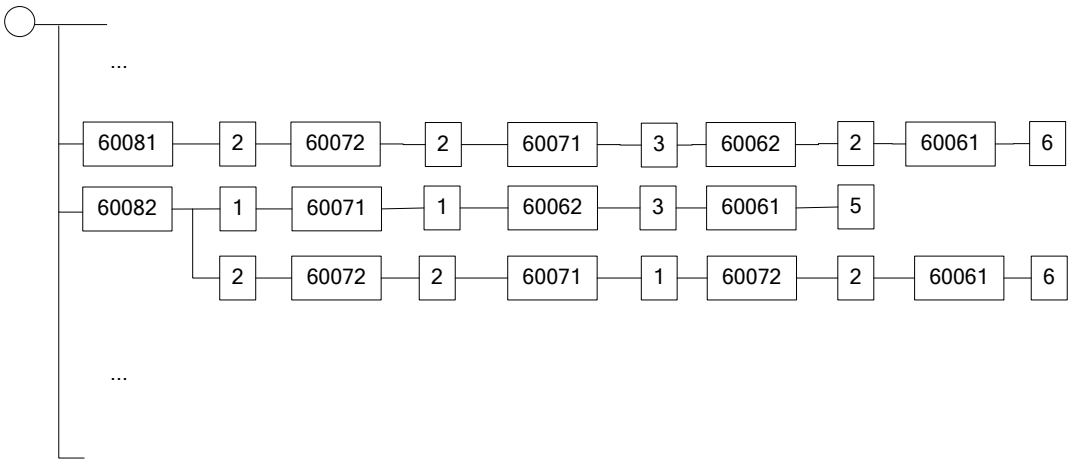


Figure 3-5 The Distribution tree of whole chemical compounds

3.3.1.2 Search of distribution tree

The search of distribution tree for potential targets is based on the input compound CGDS. General speaking, the search involves two activities: finding the collection of group matching nodes and asserting their bond values with input. Due to the structure of distribution tree, the pseudo code for search can be illustrated as follows:

```
/*
* search
*/
1.    rootNodes = root
2.    foreach GroupDescriptor in inputCompound
3.       matchNodes = findMatchingNodes(rootNodes,GroupDescriptor)
4.       validNodes = assert(matchNodes, bondSize)
5.       rootNodes = validNodes
```

In pseudo code, `findMatchingNodes` is a function listing all the matching group nodes which are under tree path staring from parameter `rootNodes`. The function makes use of the tree structure, and its pseudo code is given as follows:

```
/*
*     Function findMatchingNodes
*/
1.     foreach rootNode in rootNodes
2.        int childIndex = findFirstChildEqualOrBigger(rootNode,
                                GroupDescriptor)
3.        childNode = rootNode.childOf(childIndex)
4.        if (childNode.value == rootNode.value)
5.           increment childIndex by 1
6.           foreach bondSize node of childNode
7.               add bondSize node into matchNodes
8.        foreach childNode of rootNode start from childIndex
9.           nodes = findAllGrandChildren(childNode)
10.          findMatchingNodes(nodes, GroupDescriptor)
11.    return matchNodes
```

In the distribution tree, for each CGDS path, the bigger group node appears before the smaller group node. Therefore, when the functions meet those nodes that have bigger group descriptor values, the search needs to go deep into the next level. `findMatchingNodes` is designed to be a recursive function so that it can dig further (line 9, 10 in the pseudo code).

All the return bond nodes have to be asserted to confirm their validity. Function `assert` takes two inputs, and its pseudo code is:

```
/*
*     Function assert
*/
    1.     foreach matchNode in matchNodes
    2.        check assertion with its value and input bondSize
    3.        Success: add child node of matchNode into validNodes
    4.     return validNodes
```

In the distribution tree, group descriptor and its bond size takes two nodes, hence, after assertion, it is the child of the successful bond node that gets added into valid nodes. By design, a bond node has only one child group node, and that group node will be the root for next round search.

3.3.2 Database and distribution tree

Distribution tree provides a representation of compounds' CGDS distribution. However, there is still no direct connection between the distribution tree and the data model in the database. As we mentioned before, the physical mapping of distribution tree into data model is not desirable. The bond between distribution tree and data model therefore prefers logic correlation.

Actually the search of distribution tree results in a set of tree paths, which are equivalent to a set of CGDS values. Only those compounds in database that have CGDS values falling into that set qualify as the potential targets for further comparison. In data model, it is simple to add one more field that saves the CGDS for each compound. Thus,

if we convert output to a set of CGDS values instead of tree paths, the connection between search result and potential targets is set.

One straightforward way to make the conversion is to add one child node to each leaf of current distribution tree. The added node which is now a leaf of distribution tree saves a hashed value of CGDS represented by a tree path from the root to its parent. In addition, besides the hashed value, the leaf node can have an integer counter which indicates how many compounds have that CGDS value. As a result, the search can have not only a set of hashed CGDS values but also the exact number of potential targets.

The same hash function is applied to each compound in the database and their values are saved and indexed. Then the retrieval of potential compounds can be easily done by an SQL statement where the condition is restrained by the set of hash values.

3.3.3 Computing model

Although the above data structure and method are targeted to chemical compound search, the basic idea is more generally applicable. For many search applications, it is not hard to make some logical assertions that may help improve the search speed. In that case, we may design an advanced data structure which fully exploits the essence of that logic. The isolation of logic data structure and physical data model achieves both ease of application and flexibility of future modification. Actually, for physical data, there can be more than one logic associated with it. In other words, physical data may behave as raw data, many search choices can be made by associating different logics with the physical data. The whole idea can be represented as a computing model for the application.

3.3.4 Conclusion

With the aim of applying logical assertion in the preprocessing of compound search, a distribution tree structure is presented to fully exploit the underlying logic. Generation and search of the distribution tree are introduced and demonstrate the effectiveness and efficiency of the structure in the application of compound search. The idea behind the design can be used to develop a computing model which separates the search logic and physical data.

3.4. Algorithm and its Integration with Database

Chemical database stores data that are designed to be used by search algorithm. In general, the data are carefully tailored to be suitable for the algorithm. In that case, there exists an *association* between data and algorithm, in other words, the data in the database are algorithm dependent. It may not raise concerns if the database application always uses one search algorithm. However, in a rapidly-growing field like bioinformatics, it is usually expected that more advanced algorithms would come into play. Moreover, due to the complexity of biological processes, most bioinformatics algorithms apply certain approximations or assumptions that may not work for all the cases. As a result, it is quite natural that there may exist more than one algorithm for one application and they would share the same data.

3.4.1 Flexibility and upgradeability

In the design of search engine, it is preferable that the system is built with flexibility and upgradeability, which can be roughly defined as the capability to smoothly adopt different algorithm implementations and processing logics without changing the underlying database data and system architecture.

The achievement of flexibility and upgradeability directly leads to the following benefits:

- **Upgradeability to new algorithms**. It is not a surprise that more advanced and efficient algorithms may be introduced in the near future, especially in an area not fully developed yet. It is of best interest that the adaptation of new algorithms requires no systematic change of data and architecture.
- **Effective benchmark workstation to test different algorithms**. In the chemoinformatics and bioinformatics research area, for one topic, there usually exist a few algorithms that differ in the focus and approximation of the underlying problem. The algorithms may excel at different cases and the comparison among those algorithms on one statistically significant dataset unquestionably sheds a light on how to efficiently apply them for different targets.
- **User configurable search function**. As stated before, chemoinforamtics is an application science that always involves assumptions and disparate ideas. For chemical similarity, no well accepted definition has been established. It would be unacceptable for a researcher to use a search function that uses an unfavorable algorithm. On the other hand, a search engine that allows its user to configure his desired search function would greatly improve the number of potential users for the system.
- **Heterogeneous system integration.** Search algorithm needs to be implemented. The implementations may use different languages and stay in different platforms.

The capability to smoothly adapt different implementations into system would be a key to heterogeneous system integration in terms of database application.

3.4.2 Association between database data and search algorithm

Currently, the design of mainframe system has an *association* between data in the database and search algorithm. Take chemical compound search engine as an example: Chemical database stores data that are designed to be used by search algorithm. The data is carefully tailored to be suitable for the algorithm. Therefore, a search algorithm that demands an input of compound tree representation expects to see the data in the database to be a tree representation. It could be enforced in the design of database data model; which states that compound data in the model must be a specific data type. Subsequent data processing has to ensure data consistency by bounding data with a data type. A more advanced search algorithm which reads a different type input can not be easily plugged into the system because the data in the database is not directly supported for the algorithm. In a rapidly developing area like bioinformatics, it would be of researchers' great interest and benefit if we can decouple the association between database data and algorithm.

3.4.3 Design of Data Model

3.4.3.1 Decoupling between database data and application

In general designs, the data models reflect tight coupling between data and application. For example, it is expected that the data of a user bank account in the database would be used by some application knowing how to reach and read it. After careful observation, we noticed that the application data in database is composed by two

parts: *dynamic data* and *static data.* The key difference between them is that dynamic data keep status information which change from time to time and static data, on the other hand, stays the same at any time. *Static data* are an application specific description directly from the underlying entity. However, the *static data* may not cover all the aspects of the underlying entity and there is no requirement that underlying entity can be fully redrawn just by *static data*. On the other hand, the underlying entity can be completely described by the data which are application independent, which we name them *raw data*. The definition of both *application data* and *raw data* are given as follows.

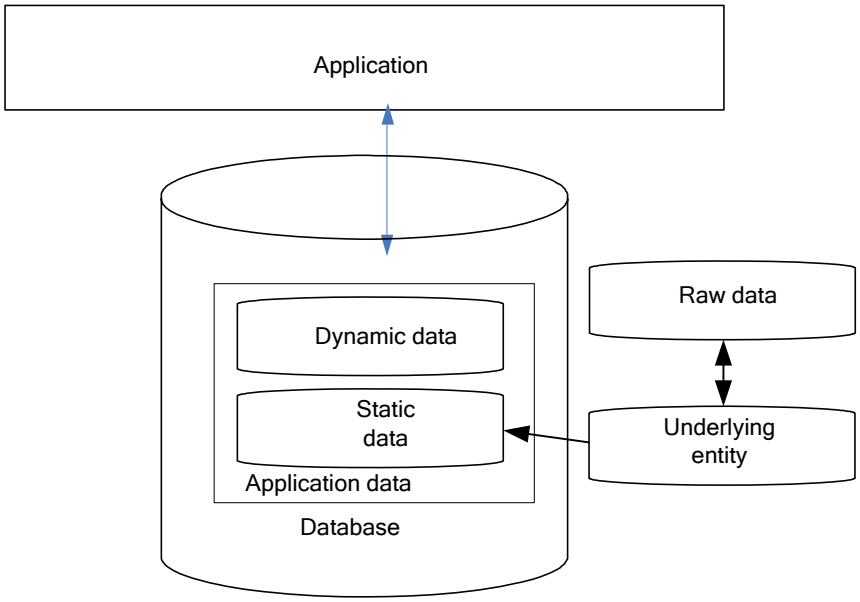


Figure 3-6 The composition of general database data

- ***Raw data***. *Raw data* is a definition which stands for an immutable, non-refined and complete description of underlying entity. When we say immutable, we mean the data are static and does not change at any time.

The data are not refined for any purpose hence they are not application and implementation dependent.

- ***Application data***. Application data partially comes from raw data but is designed and refined to work for the specific application. Application data may also contain dynamic status information that does not exist in the raw data.

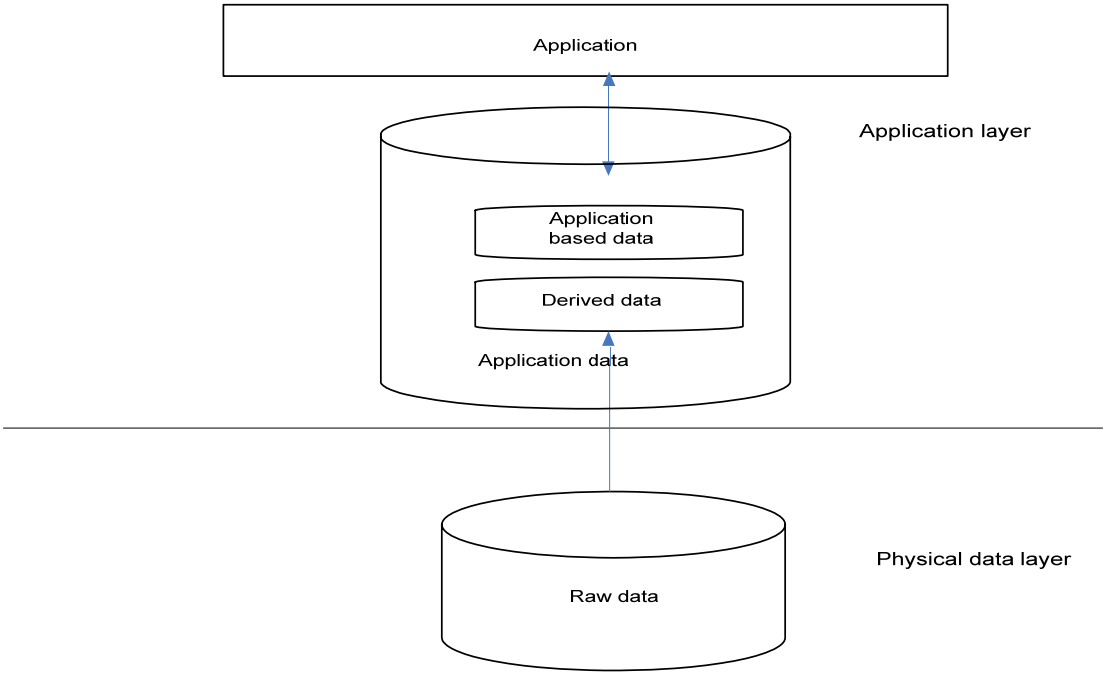


Figure 3-7 The two-layer data model architecture

In the widespread database design, *raw data* are not generated and kept in the database. Instead, only the application data are produced and stored in the database targeting directly to the application, which is the primary reason that database data is dependent on the application. Consequently, a design that aims to break data association could start with separating the database data into two layers: raw data and application

data. The raw data layer is hidden from application and only talks with application data. Application data layer works for application, its data partially comes from *raw data* but are refined to work for application. In addition, application data may contain dynamic status information that does not exist in raw data. The overall picture of this two-layer data architecture is illustrated in the figure 3-7. The two-layer architecture system highlights the independency of raw data. The application only talks with application data, which is directly generated from raw data. The modification of application only affects application data, and it could be regenerated from raw data by certain means.

3.4.3.2 Design pattern of two-layer data architecture

Two-layer data architecture provides one way to attain a separation between application and data. The two-layer data architecture may not work very well to break the data association in some cases. For example, the application status data (*dynamic data*) is dominant and there is very little immutable raw data. However, the status data are usually represented by numbers or strings, which are primitive data type and rarely vary across applications. Therefore, the design can work for decoupling data association in a relatively general manner.

The relationship between derived data and raw data can be best described to be equal to the relationship of model and view in the `MVC` (Model, View, and Control) design pattern. The derived data is an application view of the underlying raw data. Conversely, in our design, there is no controller which behaves as a communication conduit and messenger between two layers. The derived data comes from the immutable raw data and for one application and the view they should not change from time to time.

Thus controller does not need to exist in the situation. Instead, an application-based transformer would be launched in the beginning to convert the raw data into application-based derived data.

The raw data can be used for different applications. Raw data may have more than one derived data which corresponds to different applications. The generation of new application data can be done quite easily after defining new transformer and it incurs no systematic and architectural change of the whole data model. The basic idea can be demonstrated by following figure.

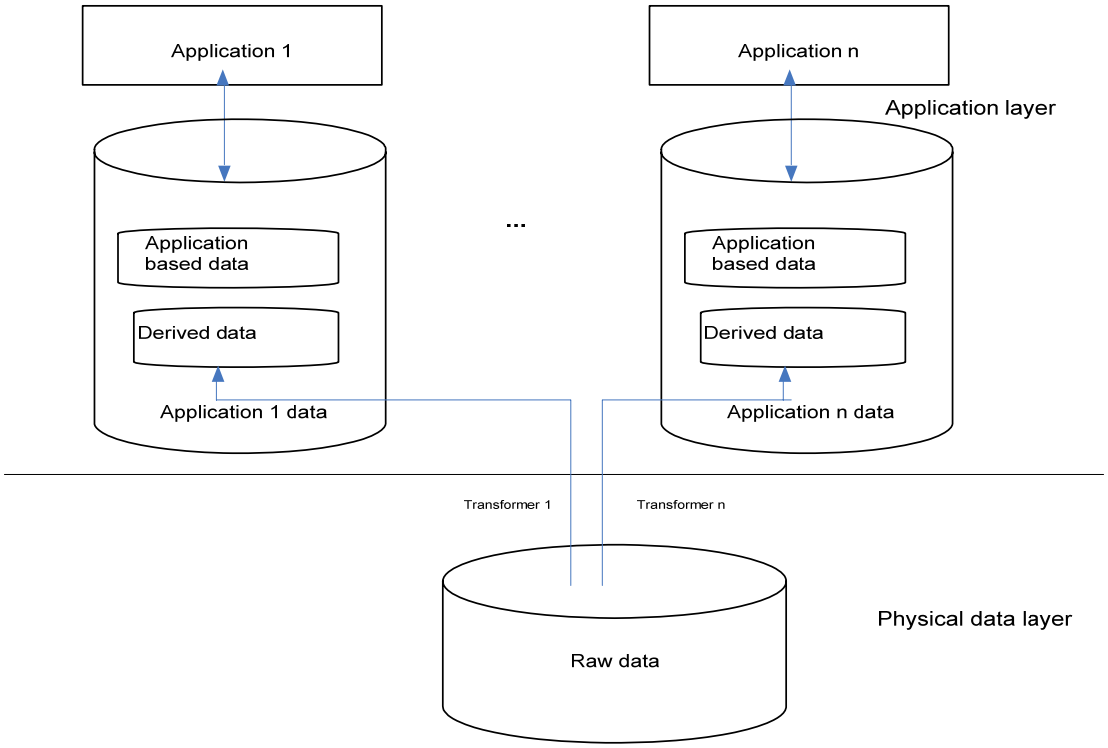


Figure 3-8 The multi-application two-layer data architecture

It is not difficult to see that the two-layer data architecture design lays the foundation to achieve those design goals defined in flexibility and upgradeability,

assuming we can generate application data from raw data based on the architecture. There are still questions, however, of how efficiently and simply we can make the transformation from *raw data* to derived data and what is the design for that.

3.4.4 Transformation from raw data to derived data

3.4.4.1 Representation of raw data

Raw data is defined to be an immutable objective data description for underlying entity, which is application/algorithm independent. From the design point of view, the representation of raw data should take following features:

- **Unambiguous**. One basic feature of raw data is its unambiguous readability for any application. In other words, the raw data should be represented in a computer recognizable way but platform and language independent.
- **Strong Descriptive capability.** The representation of raw data should be capable of describing very complicated data
- **Database support.** Raw data are supposed to stay in the database system. Therefore, the representation of raw data needs to be supported by most main frame database management systems.
- **Transformability.** According to the design, the derived data is directly generated from raw data. It would be beneficial that the representation of raw data can be easily used to make the transformation.

The requirements demand a well-established data representation and XML immediately calls our attention.

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879) [32]. Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. Since its debut, its flexibility and efficiency to carry data has been long proved. Moreover, a great number of tools have been developed to handle XML format.

3.4.4.2 XML transformation

Raw data, an objective description of underlying existence, can be described by some markable language, for instance, XML. However, the XML description of raw data is not supposed to be exposed directly to the application. Instead, the derived data needs to be generated from XML raw data which carries application specific data structure and data. That procedure can be done using XML transformation [26, 42].

XML transformation involves several entities and its procedure can be basically demonstrated by following figure.

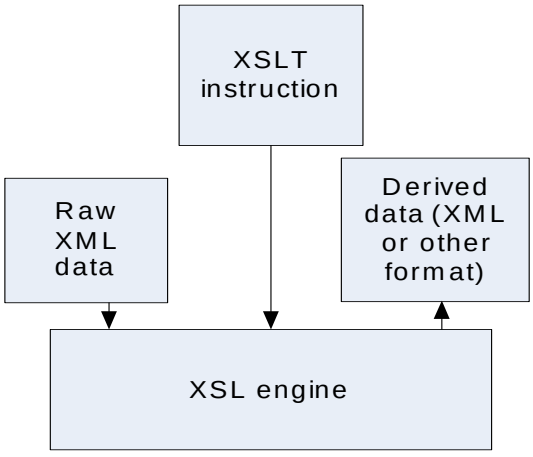


Figure 3-9 XML transformation using XSLT

XSLT is the acronym for Extensible Stylesheet Language Transformation. XML has become a popular means to represent data. One of the fastest growing uses of XML is within various business environments. Business applications use XML to represent data shared within the bounds of a business application, between business applications, and between businesses. A necessity for making use of the data housed in XML documents is the ability to access and manipulate the data to fit the needs of the business application or end user of the data. Extensible Stylesheet Language (XSL) provides facilities to access and manipulate the data in XML documents.

XSL is itself an XML dialect and provides two distinct and useful mechanisms for handling and manipulating XML documents. Many of the same constructs are shared between the two mechanisms, but each plays a distinct role. One is concerned with formatting data, and the other is concerned with data transformation. When XSL is used as a formatting language, the stylesheets consist of formatting objects that prepare an XML document for presentation.

When XSL is used for transformation, XSL takes the form of Extensible Stylesheet Transformation (XSLT). An XSLT stylesheet is composed of template rules that match specific portions of an XML document and allow the transformation of the XML document content. Not only can XSLT transform an XML document from one dialect to another, but it provides many other capabilities for extracting data from an XML document and manipulating the data.

XSLT does not work alone. An XSL processor engine performs the matching between the XML document and stylesheets. The processor performs pattern matching between the various portions of the XML document and the XSLT stylesheet.

A XSLT stylesheet has considerable power: it can create structures of arbitrary complexity. The power ensures us that we can generate any application based derived data from XML raw data. The only concern is how to define the stylesheet based on the application demands.

3.4.5 Application case: compound data in the database

In the data model design of our chemical compound database, it was observed that the major data, compound data, comes with some features which could be described as follows:

- The data size is huge. A compound database on average contains hundreds of thousands of compound data.
- The compound data itself is immutable for any operation. In other words, the compound data has no status and can not be changed.
- The connection table representation of the compound data, although simple, is language and implementation independent.
- The application/algorithm is based on the partition passed compound data representation.

Based on the features, the design of two-layer data architecture is as follows:

1. The raw data of the structural compound information is basically described using canonical connection table and encoded into XML document.

2. The generation of derived application data can be done by constructing a template in the XSLT which classifies original raw data into bond-group based categories. The generated application data still takes XML format. The bond partition based data can be efficiently described by the XML tree architecture.
3. The persistence of two layer data in the database could go two ways: (a) most current main-frame database systems directly support XML data type. Persistence as an XML type has advantages: Most of mainframe database management systems like Oracle 10g have built-in XML functions which can better support the data location and manipulation if data is explicitly saved as XML data. Nevertheless, the XML data can be associated with an XML schema, which provides a method for defining the structure, content and semantics of XML documents [3]. As a result, the validity of XML data can be verified; (b) as stated in the previous section, XML data can also be saved as a BLOB which is a standard primitive database data type. BLOB is straightforward and efficient in terms of data persistence. However, database is not supposed to talk directly with BLOB data, thus the internal information of BLOB can not be searched directly by database system and it has to be type recasted before application use.

3.5. Conclusion

In this chapter, aiming at providing database support for the search engine, several methods are proposed for different design problems. The methods cover the data persistence in the database, logical preprocessing and computing architecture for integration of algorithm and data. The methods in the chapter are

intent to provide a solid database solution to develop the substructure search engine for chemical compound database. However, the methods are general in a way that the application of them in other fields is also straightforward.

Chapter 4 Implementation of the Search Engine

4.1. Introduction

In this chapter, the design and implementation of a workable substructure search engine for chemical compound database are presented. The development and implementation of the search engine are focused on testing the algorithm, architecture and design pattern proposed in the previous chapters. It is always more convincing if positive performance can be reached for a working search engine developed for the proposed ideas. The working search engine is designed to handle large volume compound data efficiently. In contrast to general search engines that use fingerprint or fragment coding for comparison, the determination of structural relationship of search profile is processed by on the fly calculations (based on the search algorithm). The implementation of such a system involves a lot of work. In the development, however, we intend to concentrate on building a viable backbone for the whole system architecture, such as:

- Computation pattern of search engine with large volume data processing.
- Chemical compound graph representation in database
- Efficient algorithm to deal with graph-based compound substructure comparison
- Integration of algorithm and database
- Logic processing in target screen and its connection with database

4.2. System design

4.2.1 System architecture

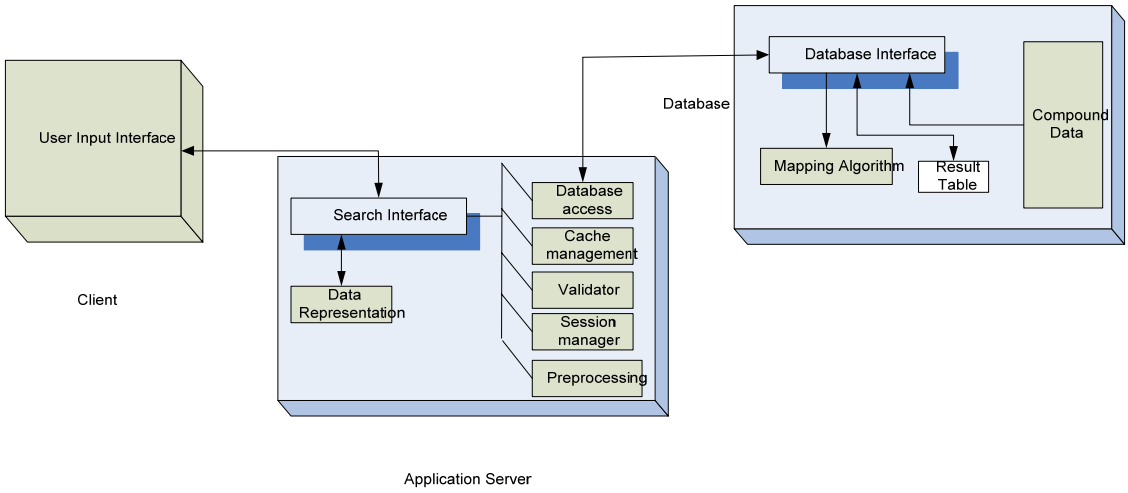


Figure 4-1 System architecture of search engine

The chemical compound search engine is complicated in the sense that the routine operation involves many functions. For the flexibility and best module isolation, we define a multi-layer system architecture as shown in figure 4-1.

The user interface is the first and only layer that is directly exposed to the end user. It is designed to handle user input in a user friendly manner. After receiving and validating user input, user interface layer transfers the search profile to the search interface which is located in the middle layer -- application server. The communication channel between user interface and search interface is alive during all the period of search calculation (see following section for details). The user interface periodically receives the partial result set from search interface and is responsible for rearranging the up-to-date results and properly displaying them to the end user.

The application server is where the business logic comes in. The submitted search profile as well as context information is first constructed into a logical request object which will be recognized by all the function modules in the layer. The search engine is designed to handle simultaneous user requests thus the session concept has to be associated with each request. Therefore, one of the basic business logics in this layer is to efficiently manage different requests. In the design, the session data, which is used to distinguish different requests, is saved in a property of the request object. To speed up the processing, the search profile in the request object is first used to look up against cache table to see if the same search profile has been requested recently. A match of the search profile in the cache table leads to a direct return of previous search result saved in the table. For any new compound search request, the search profile is logically analyzed by preprocessing module to generate the set of hash values of potential calculation targets, as introduced in chapter 3. The set of hash values and search profile data are transferred to the search interface of data module layer.

The database layer is the place where the computation really takes place. The transferred set of hash values is immediately used to locate finite potential compound data objects in the database. For each potential compound data object, the mapping algorithm is called to determine its structural relationship with input search profile target. The result is kept in a temporary result table, which is accessible by application layer through the call of database access module.

4.2.2 Computation model design

The design of search engine is expected to have the highest accuracy of substructure search. In the processing therefore, instead of using traditional fragment coding or fingerprints, which introduces inaccuracy due to the absence of connectivity information among segments, the pairwise structure calculation is undertaken between any potential target and search profile target.

As we know, the search for structural fragments (substructure) may work under a chemical database containing even millions of compound data. A challenging question thus arises for our real time search engine, which is if it still can return result in a timely manner when data size is huge. Traditionally, there are several methods that may be deployed to work on the issue: efficient algorithm of structure comparison, preprocess screening and distribute/parallel computing. Efficient algorithm can reduce individual pairwise structural calculation, preprocessing screening may greatly narrow down the potential targets, and distributed/parallel computing would speed up the whole process time by the factor of parallelism. However, they are not general solutions: When data size reaches a certain level, both efficient algorithm and distributed/parallel computing may fail to guarantee a timely feedback. Although logic preprocessing like in chapter 3 can shrink the potential targets, nevertheless, the effectiveness of logic preprocessing heavily depends on the input. A trivial input may fail logic preprocessing in its performance of screening.

4.2.2.1 Interactivity

The problem comes from the fact that a human being (end user) gets involved in the search activity. A round of search generally starts with a best profile of structural fragment inputted from the end user, the profile is then made against database through search engine and it is supposed that a predefined set of targets will return which are closest to the input. Basically speaking, the search rarely stops at one round of communication. The launcher of the search usually refines or redesigns his/her structural fragment based on current search feedback and starts the next round of search. As multiple round communications are expected between search engine and participating human being in a timely manner, the search activity is full of interactivity.

Interactivity is very important in chemical compound search. Due to the ambiguous nature of similarity definition, the human interpretation is inevitably involved. For the same search result, different users may vary by their determinations because they have different focus or expectations for the result. A search engine has to be based on certain rank algorithm to index and rank its output. However in reality, there is no rank algorithm which is agreed by all people. Although theoretically imperfect, interactivity makes it practicable: People are free to choose result no matter what rank position search engine has for it. The basic methodology was deployed in many search engines such as Google.

Interactivity is challenging if the search is against a huge data set. Human interactivity demands that the result be returned within a limited time. When the size of

data set reaches certain level, no matter what advanced algorithm is deployed under the search engine, the calculation time may be beyond that limit.

4.2.2.2 Nonatomic result return

The dilemma of limited time and possible lengthy calculation comes from one standpoint: The result has to be returned as a unit. The standpoint is based on one assumption that the result is perceptible only when it returns as a whole. It does make sense in certain applications, such as a calculation that involves several steps, where each step generates only an intermediate result. However, in the application of chemical compound search, the result contains a set of hit targets and each target in itself is final and independent. Although there is some ranking algorithm which tries to sort targets, as we stated before, the user is the final judger and ranking is only for reference purposes. In other words, the standpoint of atomic return is not vital in this case.

Nonatomic return, which pops partial result back to the user periodically, can resolve the imbalance of limited waiting time and oversize data set. With the use of efficient algorithm, there is a guarantee that a **sufficient** part of the result can be generated within a time limit. The sufficient part is based on the speed of human perception. A good analogy of film work can be used to explain the concept: anything beyond 30 frames per second may not make a film more alive, 30 frames per second is enough for human being to perceive the animation.

The traditional search is a blocking operation which expects an atomic return. For search that does not require atomic return, asynchronous communication can be deployed. After a search command is transferred, the client side does not wait till result

has been returned. Instead, the client side hooks a callback mechanism with server side. The callback mechanism will be invoked every time if there is a return available.

There is something beyond classical asynchronous communication. In classical asynchronous communication, although the operation is nonblocking, the communication essentially is one round. For our compound search engine application, nonatomic return is likely to have more than one callback. It is achievable though, with add-on self defined protocol, callback function is deployed and called to tell the stage of return and update the client side based on the data and current stage.

4.2.2.3 Interactivity and its application in web service

Web based search interface provides the broadest access to the end user without tedious installation. Web access gains more and more popularity in today's computing service. A search interface based on web access provides large range of accessibility, and it is more of interest to apply interactivity on web-based search interface. Nevertheless, in essence, web application is nothing but a client-server computing model. HTTP, which is the underlying communication protocol for web application, takes the blocking operation. Due to the problem mentioned above, most of the current chemical compound search engines which rely on pairwise structural calculation only provide an e-mail based web interface, which totally eliminates the interactivity of the search activity.

A nonatomic return communication can be applied to web application as well. With the help of some existing techniques, traditional web communication can be converted to be asynchronous. It provides a solid ground for applying an interactive web-base interface for chemical compound search.

As discussed before, classical web application is nothing but a client-server model and that model enforces a synchronized request and response. That model is basically considered to be a thin client model because most of the calculation does take place in the server side, while client side only works as a simple user input interface. The computing brings a heavy load to the server side when requested volume increases and is considered to be not well scalable. To solve that, recently some new techniques like AJAX (Asynchronized Javascript and XML) are introduced to accomplish a thick web client by involving more calculation in client side. The concept practically increases interactivity by reason of some logic being done in client side, and due to the asynchronous communication, it provides a solid foundation for the application of nonatomic return methodology mentioned previously. However, though the communication takes non-blocking mode, essentially the communication requires a full result return, which is not in accordance with the design of nonatomic return.

In the design, something can be added to make the communication more flexible. There might be two ways to add that functionality:

- By repeat client requests

  After receiving a request, server side commits to having a return in a predefined time span along with the current progress ratio of result even if the result is not fully finished. Client side keeps launching another round of request with same ID if the progress ratio has not reached full. Server side can easily identify if there is a new search request just by looking up the ID in local table. The request with ID in table leads to an updated return with new generated result.

- By event status

  AJAX technique uses `XMLHttpRequest` object, a W3C specification under draft. Although not formally documented, The `XMLHttpRequest` object is implemented today, in some form, by many popular Web browsers. The current implementation has an attribute of `onreadystatechange` of type Function, An attribute that represents a function that MUST be invoked when `readyState` changes value. The function MAY be invoked multiple times when `readyState` is 3 (Receiving). If we define any partial return to be associated with an invocation of `onreadystatechange` with `readyState` to be 3, a client function that targets that state knows exactly that this is a partial result and is coded to handle it.

4.2.2.4 Parallel/cluster computing

The operation of compound search involves repetitive pairwise structural calculation. For each pair, the calculation is complete and independent of the others and it provides a solid base to apply parallel/distributed computing.

The key point for parallel/cluster computing for compound search is how to split data sets across the available computing resources evenly so that each resource would have the same load for any task. Assume there are ten computers and a task involves 100 structural pairwise calculations, the best hope is that each computer takes 10 calculations thus all the computers would finish their work in almost the same time.

Given a data set and assume any one of the set would be calculated for any input task, the distribution of task would be straightforward based on the partition of data set. The data set could be partitioned into *n* parts where *n* is the total amount of computer

resources available. Each computer is in charge of the calculation of one part of the data set for input task.

However, for compound search engine, not all the compound data would be calculated after the screening operation. Only a small percentage of potential compound targets would be calculated against the input, as a result, the distribution of potential targets is not fixed with data set and varies case by case, which makes the task allocation complicated for parallel/distributed computing.

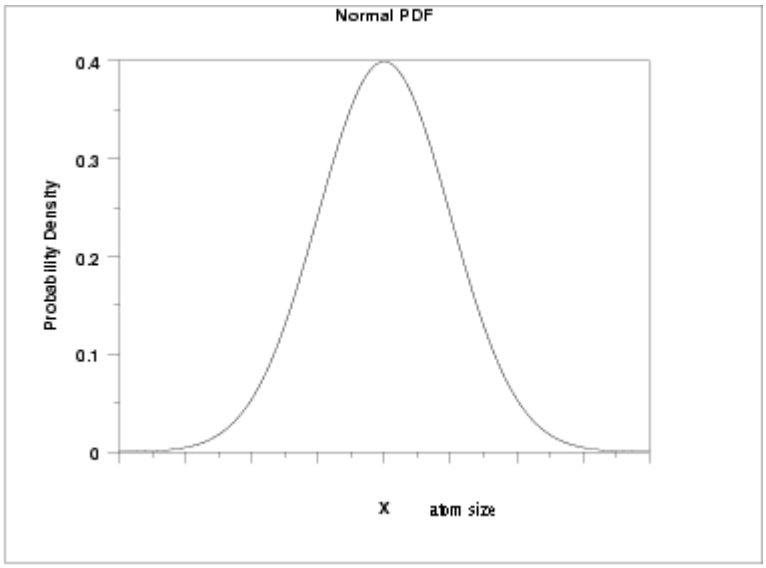


Figure 4-2 Normal distribution of compound data in terms of atom size

In the design, experiment analysis was adopted to have a roughly even distribution. As we know each compound data is composed by atom and bond, if we partition data set based on the size range of atom and bond, we would get non-overlaping sets of compound data with different size range. Further data analysis found that the whole data set is an approximately normal distribution in terms of size range (shown in figure). The compound data cluster heavily around atom size 40 to 100. The observation

tells us that the partition of data set should reflect that distribution. Hence basically, the size range interval jumps quickly at two sides and proceeds slowly in the gathering range of 40 to 100.

Orion cluster is a high performance parallel cluster. In our lab, there is a 12 node Orion cluster and it is designed to host the application server. In 2.4, a distribution tree concept is presented to handle the preprocess logic. Generally, the distribution tree reflects the whole compound data in database but the concept can be applied to a subset as well. In the design, the head node of Orion computer is working as the global organizer of the service in this layer. Each of the rest of the 11 work nodes in cluster holds a unique distribution tree which represents a subset of compound data in the database. The individual distribution tree has no overlap with any other and the union of all distribution trees fully reflects the entire compound data in database. The task of search will be broadcast to all work nodes and be processed parallel. Each work node has access to database and results will be returned to head node.

The communication between head node and work nodes is multiple-round based and based on straightforward socket programming. In head node, a session id will be created to associate with each search request, and if there is any change regarding the search, the head node will notify all work nodes. To speed up the processing, a cache lookup table may be set up for each work node. A hit in the lookup table leads to a direct result, which is a significant improvement in terms of speed.

In this design, we propose a model which is believed to accomplish some interactivity in the compound search. The real interactivity is determined by the

perception of the human and may vary from person to person. While the real implementation has to be fixed in hardware architecture, there is no doubt the design concept can be extended. Parallel/cluster computing can work out as an adjustment factor to reach the design goal if the performance is not favorable.

4.3. Implementation

4.3.1 Data processing

4.3.1.1 Data source and preprocessing

The raw structure data that were used to build the chemical compound database came from NCI open source (http://cactus.nci.nih.gov/ncidb2/download.html). There are 250,251 2D structures calculated with CACTUS. (*Attention*: Stereochemistry is assigned by CACTUS according to default rules due to lack of stereochemical information in the original NCI data. The SMILES string and the CAS RN (where available) are also included for each structure).

NCI offers a downloadable file that is uncompressed to a SDF file of about 982 MB, which contains all the compounds. For the convenience for further processing, we have developed a Java program (`DeprocessPro`) to decompose all the compounds into individual molecule files.

♦**File name and primary key**

The decomposition requires giving a name for each compound file, and that name should work as a primary key in the database to identify each compound. Although for any publicly deposited compound, a unique CAS registry name was already given. However some compounds are proprietary and do not have a CAS registry name. In

NCI system, it defines a unique 6-digit NCI internal number for each compound and for any compound with no public CAS registry, a dummy 999-99-9 number is used for its CAS registry name. In our design, the file name and primary key of compound comes from the combination of CAS registry name and NCI internal number: if the CAS registry name is available, that name will be used as the key; else the file name is given by a leading 999-99-9 and a following parenthesis which has the NCI number inside.

♦Data arrangement (compound tree)

The decomposed molecule files are arranged in file system in such a way to be compatible with their chemical features and thus convenient for the following processing.

- All the molecule files would be put together under one destination folder and that folder is named as root folder.
- The file path between the mol file and the root folder reflects one of its basic chemical characteristics: its atom size and bond size. Each molecule file is under two-folder deep of the root destination folder. The first level folder is named by the number of atom size and the second level folder is followed by having its name from the bond size.
- The compounds with the same atom and bond size would stay in the same folder and the compounds with the same atom size would have same parent folder. The whole arrangement of mol files in the file system constructs a three-layer compound tree.

- The construction of the file system structure was done on the fly by the `DeprocessPro` during the decomposition procedure.

4.3.1.2 Data representation for chemical compound

Mol file is just an ASCII text file. For convenience, a logic data representation which fully records structural information but is easy to process for the algorithm is used to wrap the mol file. In the design of the compound data representation, both the partition concept introduced in chapter two and the object-orientation idea are deployed. From the object's point of view, a chemical compound consists of a set of partitions of bond group, each partition has a set of the bonds with the same type, and a bond is constructed by two atoms with a connection value. Thus an object hierarchy for the chemical compound can be drawn (Fig. 4.3). For each object in the figure, a corresponding class is defined and they are `Atom`, `Bond`, `Partition` and `Compound`. In the Compound class, a parser method is implemented to parse a molecule file into a compound object.

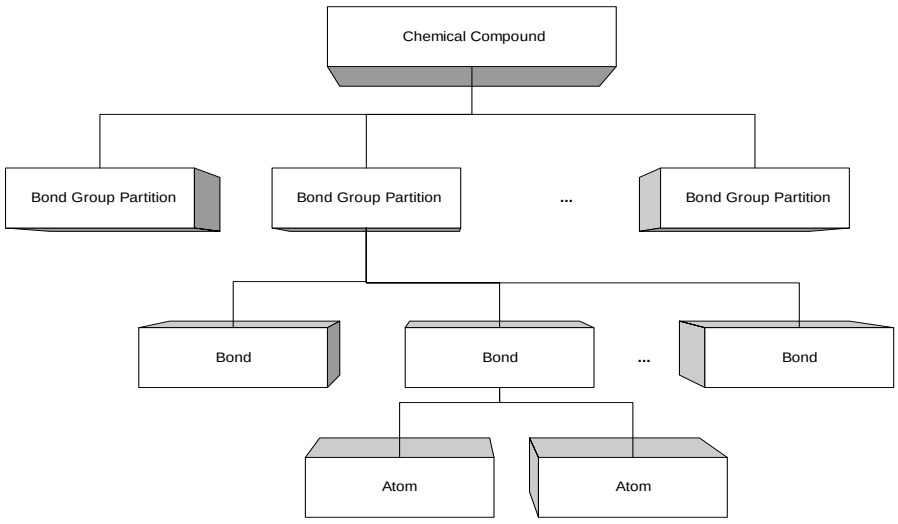


Figure 4-3 Object hierarchy of chemical compound data representation

4.3.2 Implementation of algorithm

The Java language was chosen to develop the algorithm due to its powerful language capability and wide acceptability. The algorithm proposed in chapter two is used for pairwise compound comparison. To separate the algorithm and operating data, an abstract class `Mapping` was first defined. The `Mapping` class defines two basic abstract member functions: `initialize` and `process`, which subclass of the `Mapping` has to implement. The `Mapping` class takes two compound inputs. The inputs are defined as an interface type `Compound`, which is the abstract interface of the compound data representation. The `Mapping` and `Compound` classes specify the basic prototype of the algorithm and the algorithm implementation takes the subclass format of these two. In our case, `SubMapping` and `PartitionCompound` are the two classes that are designed to implement the partition algorithm and their whole relationship is illustrated in figure 4-4. The design provides great flexibility in a manner that several of mapping algorithms could be developed in the same way and the switching of the algorithms in the future would not incur too much hassle.

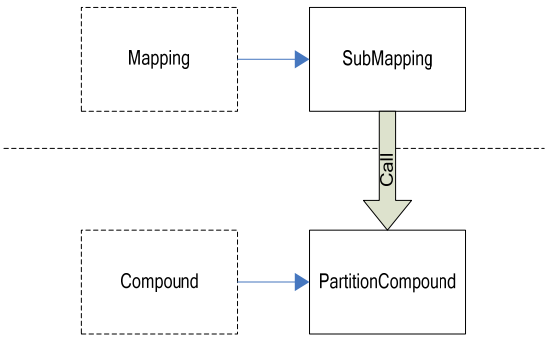


Figure 4-4 The relationship among abstract classes and implementation classes

In the SubMapping class, the implementation of the partition-based algorithm, the constructor of the class takes two input compounds defined as ParitionCompound and the steps described in the algorithm are designed to be its private member functions which are called by either of two public functions: initialize or process. Since the mapping forest is introduced to simulate the mapping procedure, a corresponding class MappingForest is also defined to be responsible for that procedure. MappingForest has field definitions such as active nodes. The member functions bondLocate and bondMapping are used to implement the activities of forest generation. The MappingForest object is cloneable when ambiguity is reached and a new mapping tree is necessary. SubMapping class keeps all the valid MappingForests in a list and if after the processing the size of the list is not equal to zero, it signifies that at least one substructure mapping has been established between two input compounds. The detailed mapping can be further obtained by querying MappingForest.

4.3.3 Implementation of screening

4.3.3.1 Definition of the distribution tree

In the Java language, tree structure is a fully implemented data structure. The basic unit of a tree is the tree node, and Java has a super interface TreeNode for that. TreeNode has a subinterface MutableTreeNode which defines the tree node that can be changed. A general-purpose class DefaultMutableTreeNode is provided by Java to operate on tree structure. For a tree structure, after we instance each tree node into DefaultMutableTreeNode, the tree can be basically represented by a

`DefaultMutableTreeNode` object which stands for the root of the tree. Given that root we can explore the whole tree by calling the built-in functions implemented in `DefaultMutableTreeNode` class.

As described in section 3.3, a distribution tree fully represents the compounds distribution based on their CGDS. In essence, the distribution tree is still a tree structure, and in the distribution tree, each non-leaf tree node keeps a value which is a part of the CGDS strings. The leaf node of the distribution tree has a hash value associated with database, and that hash value is uniquely generated between the root and the leaf (A CGDS string).

In the design, we have a class definition for the tree node in distribution tree, `DistributionTreeNode`. The `DistributionTreeNode` extends `DefaultMutableTreeNode` class hence inherits all its functions. The `DistributionTreeNode` converts its part of `CGDS` string into a numeric value and saves that value in a private field. The basic `DistributionTreeNode` is used to wrap all distribution tree nodes except leaf nodes, where we have a dedicated class `LeafNode` for that. `LeafNode` is a subclass of `DistributionTreeNode`, it is designed to be associated with the backend database by a common `CGDS` hash value. Beside that, the leaf node also records the total number of compounds that share this `CGDS` value. Therefore, after the search of `DistributionTree`, the total number of potential targets could be easily calculated just by summing up that number in all matching leaf nodes.

4.3.3.2 Partition of the distribution tree

The distribution tree is a logical abstraction of the CGDS distribution for a collection of compounds. Given a large amount of compound data, we could have just one distribution tree for all the data. However, it would be of benefit if more granular distribution trees are generated.

In the case of our application, there are more than 250000 compounds. It would be a huge tree if all data are put into one tree. The direct impact of that design is the requirement of large memory and slow processing speed. The distribution tree has to be loaded into memory before any search can be carried out, thus the host of the distribution tree needs to have extra memory resources. In addition, the search against that tree would be time consuming due to its large tree size.

The partition of compound data into different distribution tree is preferable. Actually, CGDS is one of the logical steps in preprocessing we can apply to screen the potential targets. On the other hand, as we introduced in section 3.3, other logics such as screening based on atom size and bond size can also be used to narrow down the targets. The simple logic comes from the fact that a compound may contain another compound as a substructure stands only if both the atom size and the bond size of the first compound is at least the same as the second one. Consequently, if we partition the compound data into different distribution trees based on their atom or bond sizes, after determining the size of the input compound, only those qualified distribution trees need to be searched. Moreover, the concept of partition is fully compatible with the distributed/cluster

computing. More granular partition of distribution tree leads to more flexibility of task allocation if more than one computing resources are available.

In the data processing section of this chapter, we described that individual compound data is decomposed into file system based on its atom size and bond size and all the folders and files thus generated form a compound tree in the file system. Given that as an input, the partition of the distribution tree can be easily made as long as some rules are specified. For example, we can define a fixed range such as an interval of 20 and use that as a splitting range for the partition of distribution tree. As a result, the compounds with atom size range of 1 to 20 are partitioned into the first distribution tree, 21 to 40 into the second distribution tree, and so on. The partition operation is illustrated in the following figure.

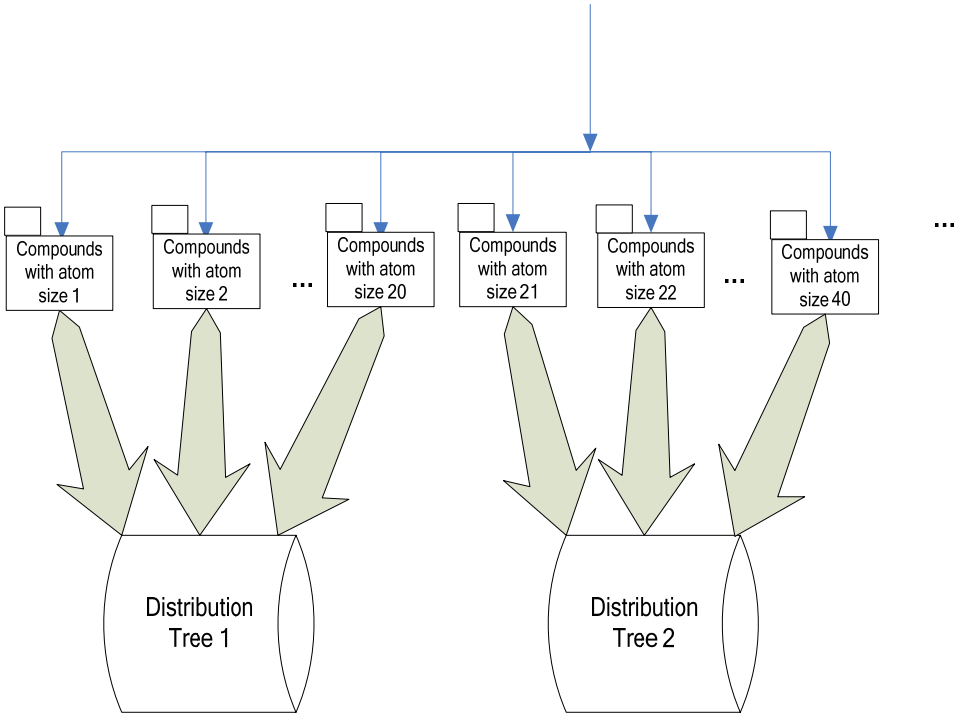


Figure 4-5 The partition of compound data into different distribution trees

For the best performance, each distribution tree should encompass roughly even part of the whole data. According to the data analysis of this chapter, the compound data appears a normal distribution in terms of the containing atom size. Therefore, in the implementation, the partition does not simply follow a fixed range.

4.3.3.3 Generation and persistence of the distribution tree

The generation of the distribution tree is straightforward once the set of compound files are defined. The procedure starts with an instantiation of a root node for the new distribution tree, and then the processing of an individual mol file would become to add a CGDS based tree path if it does not exist before. The pesuocode for the generation of the distribution tree object is given as follows:

```
/*
*   Function DTGeneration
*   Generation of the distribution tree for a set of compound files
*/
1.    Instantiate a new tree node as the root node
2.    foreach compound file
3.    read and wrap it into a Compound object
4.    Declare a variable node root and assign the root node to the
      variable
5.    Sort the groups in the Compound by the descending order of
      their numeric values
6.       foreach group in the Compound
7.          if root does not contain the group as a child
8.             Create a child tree node based on the group
```

```
9.              assign the new child tree node to the root variable
10.           assign that child node to the root variable
11.         increment 1 to the total molecule count in the leaf node
12.     return the root node of the distribution tree
```

The distribution tree object needs to persistent so that it can be reached by the application server afterward. Object serialization is widely supported by most of advanced languages but the internal implementation mechanism varies. The natural way of object serialization suffers some problems. 1) The serialization of one language can not be parsed by the other language and the stream of serialization is not even designed to be read directly by a person, which is the case when you open up the serialization file of a Java object. 2) Even in one language, if there are some changes in the class implementation after you serialize the class object, you may fail to reach it (version issue).

The problems come from the fact that traditionally, the persistence of graphs of objects was made by an approach called **Marshalling**, which basically records all states in an object graph, including non-public states. The simplest schema taking the approach requires the inclusion of all the classes that define the objects. There is a practical alternative approach that is officially called **archiving**. The approach records only all states that can be reconstituted using the public APIs of the objects in the graph. The second approach cannot produce as faithful a copy of the original objects as the first but can store the state of the graph in such a way that any API-compatible implementation of the classes involved will be sufficient to reconstitute it. Since APIs are so much more stable than their private implementations, this single step virtually solves the versioning

issues for most practical purposes. In addition, compared with the first approach, the size of the file generated by the second approach can be reduced dramatically.

The second approach can also be used to generate serialization using some standard format, such as XML document. In Java, since 1.4, `XMLEncoder`, an alternative class to the traditional `ObjectOutputStream`, was provided. The `XMLEncoder` class is exclusively designed for the purpose of archiving graphs of JavaBeans as textual representations of their public properties. Like Java source files, documents written this way have a natural immunity to changes in the implementations of the classes involved. The `XMLEncoder` class provides a default denotation for JavaBeans in which they are represented as XML documents complying with version 1.0 of the XML specification and the UTF-8 character encoding of the Unicode/ISO 10646 character set. The XML documents produced by the `XMLEncoder` class are:

- Portable and version resilient: they have no dependency on the private implementation of any class and so, like Java source files, they may be exchanged between environments which may have different versions of some of the classes and between VMs from different vendors.
- Structurally compact: The XMLEncoder class uses a redundancy elimination algorithm internally so that the default values of a Bean’s properties are not written to the stream.
- Fault tolerant: Non-structural errors in the file, caused either by damage to the file or by API changes made to classes in an archive remain localized so that a

reader can report the error and continue to load the parts of the document which were not affected by the error.

In our design, the distribution tree is represented by connected `DistributionTreeNodes`. Each node contains several public properties that are critical to construct the distribution tree and those properties can be output into a XML document by `XMLEncoder.` A sample `DistributionTree` XML file is shown in the figure 4-6. Given that XML output, we can fully reconstruct the original `DistributionTree` Object by using `XMLDecoder`, the class which handles the deserialization procedure.

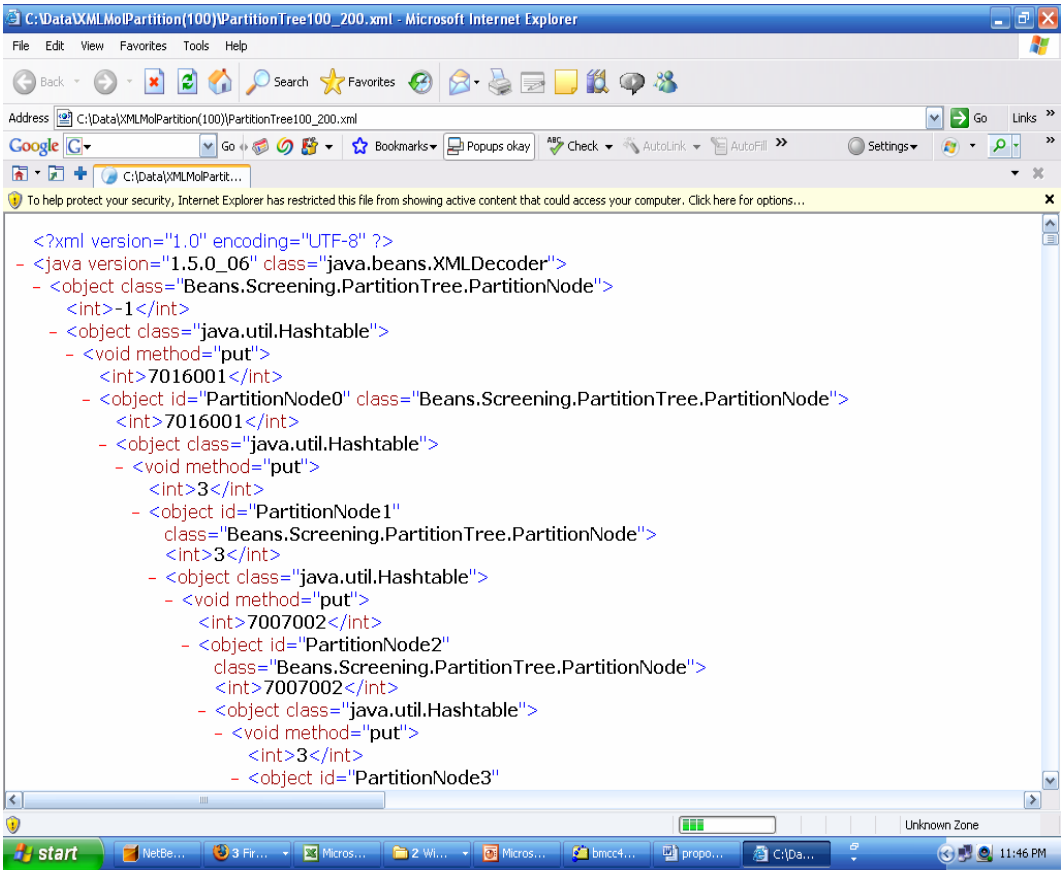


Figure 4-6 A sample distribution tree XML file

4.3.3.4 Search of the distribution tree and database

As indicated in system architecture, distribution trees are supposed to stay in the application server and are used in the process of screening. After receiving the search request from the user, the application server can determine the distribution trees to search based on the search profile. Those trees will be loaded if they are not in memory and then the search profile will be searched against the trees to locate potential set of hash values. The application server will generate SQL query based on retrieved set of hash values and that query will be submitted through data access module for the database operation. In backend database part, the potential compounds will be easily pinpointed as their hash values are already determined.

4.3.4 Implementation of database

In the design, we chose Oracle 9i as our backend database management system. Oracle database system has long been proven its high performance and remarkable stability in terms of data management. In addition, starting from oracle 8, Oracle system has built-in support of Java language, and for our application, the implementation was primarily developed using Java language. As a result, the integration of application with database would be much more convenient and natural if Oracle database is chosen.

4.3.4.1 Data representation for chemical compound in Oracle database

Compound data in the database is encoded by the class representation introduced in section 4.3.1.2. The representation involves a graph of hierarchical classes for the compound definition and they are all saved into Oracle database as a Java class type.

There is a preprocessing step which parses all the compound files and save data into database. During the process, Compound object would be first instantiated from compound file, and then the compound object is saved into the database. The object persistence adapts the **archiving** approach and takes XML format as output. The generated XML document is saved as a BLOB type in the database.

The point of taking **archiving** approach instead of traditional object serialization method is that the compound data in the database would be comparably stable even if we later change the implementation of representation classes. The update of the classes only needs to reload the class code into the database system by calling Oracle support utility, the whole compound data would not be affected provided that the new code does not change the public interface, which is normally the case for the software development.

The compound data stays in the database and so do the operations on them. In the design, the operations of the compound data, like query, are exposed to end users by stored procedures. Those stored procedures are actually Java stored procedures, and they act as a normal SQL stored procedure from outside view. However, owing to their Java essence, they can handle compound object in its natural way. As a result, the operation on compound data can be developed by using powerful Java language. Additionally, because both Java stored procedures and compound data, stay and execute in the database, there is no overhead of network communication in contrast to those applications that require the logic processing in the client side.

4.3.4.2 Integration of search function with Oracle database

The core part of the search function is to determine the two compounds' substructure relationship and as introduced in chapter two, it is accomplished by the search algorithm. The search algorithm would be called many times by feeding different pairs of compounds during the whole search procedure and consequently, it is preferred that the implementation of algorithm is located close to the compound data, in other words, reside in the database. Same as compound data, the algorithm implementation, `SubMapping` class, was developed by Java language. On account of that, it was directly saved into the Oracle database as a Java class for the best performance.

On top of the algorithm and compound data, control logic is required to manipulate the whole search process. That is why another `Java` class, `Search` was developed. Search works as both the control headquarters for the search function and the data access interface for the end user. As indicated before, search function interface was exposed as Java stored procedures and they are actually public member functions of the `Search` class. The object instance of Search class is instantiated once Oracle database system was loaded and it is always in the Oracle system memory waiting for the request. The request call from application server invokes a "search activity" of the Search object. The "search activity" is logical abstraction of the search procedure for one specified request and was threaded implemented. Basically, the "search activity" takes three steps to handle one request:

1) Locate potential compounds. The set of hash values transferred from application server is used to position the potential targets in the database. This step takes

little time and the result, total number of potential targets found in the database, as well as the identification number for this activity, is returned to application server as the immediate output.

2) Repetitive calculation. The second step involves recurring pairwise calculation between search profile and each individual potential target. In the beginning of this step, a temporary table, which saves the final matched targets, is created. For each round of calculation, if submapping can be established, the matched compound information is saved into that table. The table has a special field used to indicate if the table is final or still under processing.
3) Wait and self-destruct. After the second step, the search activity is idle for a while if no terminate instruction is received. It will release all the resources and itself anyway, after a predefined span of time, even if these is no instruction for that.

The communication between application server and Search object is one way. The Search object will not inform application server of current progress of search activity. Instead, another dedicated Java stored procedure is exposed. By passing search activity identification number as parameter, the application server can retrieve the up-to-data information of the result table for that activity. Application server, at any time, can also terminate the server activity by calling the same stored procedure with both identification number and a stop flag set to be true. The whole process is illustrated in the figure 4.6.

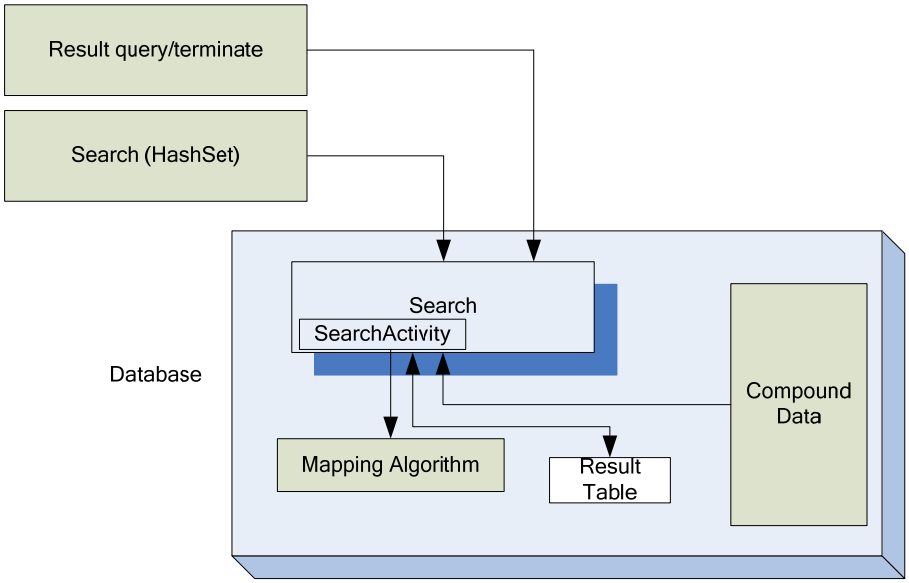


Figure 4-7 Design of search engine

4.3.5 Implementation of web interface

As to the user interface, there is no restriction on how it is implemented. However, web applications provide the broadest access to the end user without tedious installation. In the design, we have implemented a web-based search user interface.

4.3.5.1 Molecular editor

The user interface is designed to provide a way for a user to input and a user friendly molecular editor should be provided. In our web application, JME molecular editor [16], by the courtesy of Dr. Peter Ertl, was deployed to present a web-based graphic user interface for users to draw their input compounds. JME Molecular Editor is a Java applet which allows to draw/edit molecules and reactions and to depict molecules directly within an HTML page. Editor can generate Daylight SMILES or MDL mol file

of created structures, which is the supported input for our implemented compound object. (The web page with molecular edit Java applet loaded is shown in figure 4-8)

4.3.5.2 Design of web interface application

The web interface application was developed under J2EE architecture. The implementation involves JSP, Java Servlet, JSTL, Structs Framework, Javabeans, JDBC and AJAX, and it strictly follows the MVC (Model view and control) design pattern.

The front end web page `input.jsp` is dedicated for user input. It is integrated with molecular editor applet, and JavaScript is used to interact with applet when input is drawn. The input molecule would be retrieved as a string stream of MDL mol format and submitted as a property in `ActionForm`, a class defined in Structs Framework.

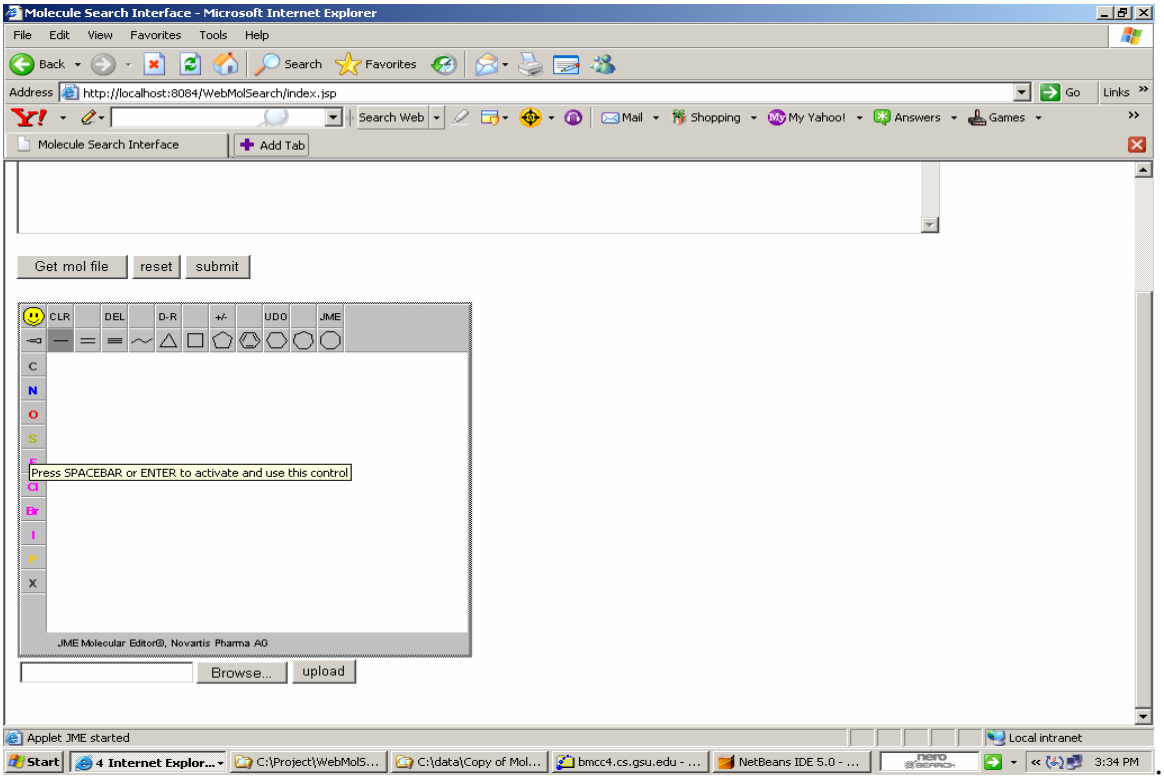


Figure 4-8 A snapshot of web page with molecular editor applet loaded

The input is transferred to a handling Java servlet, which basically handle different `ActionForm` by calling different `Action` class objects, where the `Action` class is also a concept in `Structs` Framework. In our design, the business logic in application layer was developed into a `Search` JavaBean component, our customized `Action` class object only needs to launch search method exposed by the `Search` component, and the result would be given as a `SearchResult` JavaBean component. An `ActionFoward` class object would then be launched, which defines output view to the end user.

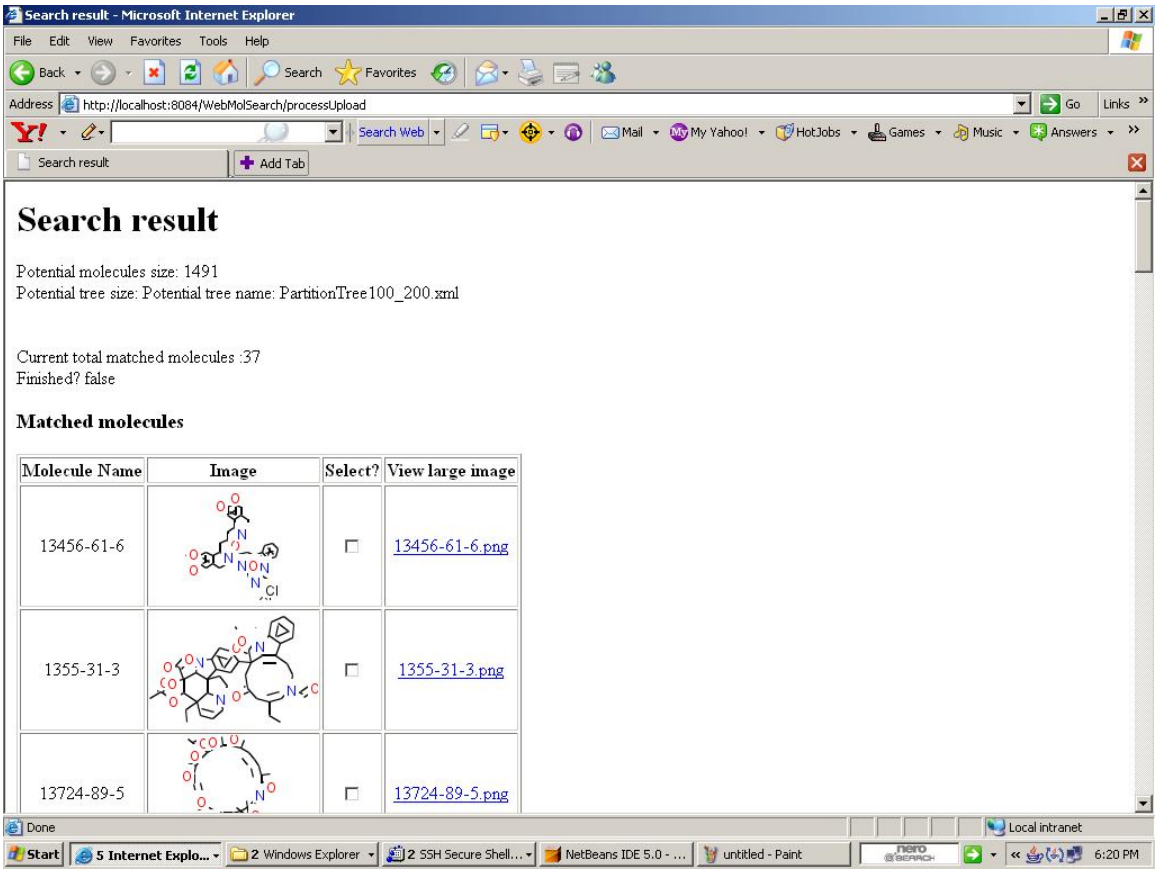


Figure 4-9 A snapshot of search result web page

The final output view is displayed in a JSP page (figure 4-9). It basically lists the current found search targets by exploring `SearchResult` component using JSTL. Due to the nonatomic return nature for our search activity, it is designed to refresh periodically until the result is final.

4.4. Experimental result and performance

The efficiency of search engines involves several factors, and they are all tested in terms of accuracy or performance.

- Algorithm. The accuracy of implementation was test by feeding a collection of pairwise compounds for substructure mapping. The implementation correctly identified all mapping relationships.
- Screening. General speaking, the performance of screening varies case by case with different input search profiles. In the test, a set of non-trivial search profiles were employed against screening operation. On average, the screen logic we use could filter out 85%~90% unqualified compounds. It serves much better if the search profile is kind of special in terms of containing atom or containing bond.
- Nonatomic return. We test the calculation model by searching against a compound dataset featuring total atom size between 100 to 200 in one personal computer. It takes seconds to report the total potentials targets and in every 15 seconds, the result could be refreshed to reflect the up-to-data progress. When dealing with more data, the whole processing may need more time if handled in one computer, but, as we stated before, the design is supposed to assign the task

evenly into different nodes in a distributed/parallel computing environment. Hence, processing speed is still under control.

Chapter 5 Conclusions and Future Works

In the dissertation, a design and implementation of a substructure search engine for chemical compound was presented. The search for structural fragment (substructure) of compounds is very important in medicinal chemistry, QSAR, spectroscopy, and many other fields. Due to the complexity nature of structure comparison and large amount of compound data which is usually involved in search calculation, current search engines mainly works out the speed by applying fingerprint comparison instead of pairwise structure calculation. By defining and encoding chemical structure information into fingerprint, which generally takes primitive computer data representation, the search calculation can be done in a remarkably fast way. However, the definition of fingerprint is subjective and some structure information, such as connectivity between different segments, is not existed in the fingerprint. As a result, inaccuracy is introduced as a tradeoff. However it must be remembered that the experimental use of these compounds in a real chemistry lab may take months or even years so it is important not to be overly aggressive at trimming processing speed at the cost of missing critical leads. It is more important to find good lead compounds that it is to reduce the time of the query from a month to a second.

Taking aim at providing a more accurate search engine with reasonable speed, in the dissertation, a total new design solution was provided. The design starts with a new look at graph isomorphism problem of compound substructure mapping, although essential a NP problem, chemical compound graph has some features that may not exist

in general case and those features can be used to simplify the calculation. The observation of the problem ends with a new partition based algorithm and the corresponding representation of compound data. The new algorithm is logically simple and proved to be effective and efficient in determining pairwise compound structure mapping. Secondly, the design tries to work on the problem under a database view. By proposing a method to save any type of object instances into database, the design provides the way to persist our partition based data representation into the compound database. In addition, the design introduces a relationship logic based on the partition concept, which can be used to apply screening for compound search in the database. The application of the logic for screen processing is under a new proposed computing model, which totally eliminates the coupling between screening logic and data model in the database. In this part, a design how to isolate the data, the algorithm and the implementation is also discussed. To verify the design, in the third part, we implemented a search engine based on the proposed methods. The implementation touches a wide range of practical problems. It first proposes a nonatomic return concept for compound search. Based on that, a calculation model can be developed to handle both accuracy and speed. The implementation takes multi-layer system architecture and was developed mainly in Java architecture. The detailed implementation introduction of each major part of the system is given in this part. The test of developed search engine demonstrates its effectiveness in the application of compound search.

As we stated in the very beginning, a search engine for chemical compound database is very important in a variety of fields. The design and implementation would

not be complete until more knowledge can be applied and more tests can be taken. The current dissertation provides a basic solution to the problem and it is not mature in many ways. However, it leaves a solid base and flexible architure onto which further efforts can be applied.

Bibliography

1. *Chemical Graph Theory*. CRC Press, Boca Raton, 1983.

2. *Chemoinformatics*. John Wiley & Sons, Germany, 2003.

3. XML Schema.

4. Alexander Barmann, H.M., Dirk Walkowiak Substructure searching on very large files by using mutliple storage techniques. *Journal of Chem. Info. Comput. Sci.*, *33*. 539-541.

5. Allinger, N.L. Conformational Analysis 130. MM2. A Hydrocarbon Force Field Utilizing V1 and V2 Torsional Terms. . *J. Am. Chem. Soc.*, *99*. 8127-8134.

6. Allinger, N.L., Yuh, Y.H. and Lii, J.-H. Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 1. *J. Am. Chem. Soc.*, *111*. 8551-8565.

7. An, J., Nakama, T., Kubota, Y. and Sarai, A. 3DinSight: an integrated relational database and search tool for the structure, function and properties of biomolecules. *Bioinformatics, 14* (2). 188-195.

8. Barnard, J.M. Substructure searching methods: Old and new. *J. Chem. Inf. Comput. Sci.*, *33*. 532-538.

9. Barreca, M.L., Rao, A., De Luca, L., Zappala, M., Gurnari, C., Monforte, P., De Clercq, E., Van Maele, B., Debyser, Z., Witvrouw, M., Briggs, J.M. and Chimirri, A. Efficient 3D Database Screening for Novel HIV-1 IN Inhibitors. *J. Chem. Inf. Comput. Sci.*, *44*. 1450-1455.

10. Bayada, D.M., Hamersma, H. and van Geerestein, V.J. Molecular Diversity and Representativity in Chemical Databases. *J. Chem. Inf. Comput. Sci.*, *39*. 1-10.

11. Bradley D. Christie, B.A.L., James G. Nourse Structure searching in chemical database by direct lookup methods. *Journal of inf. comput. sci.*, *33*. 545-547.

12. Camoglu, O., Kahveci, T. and Singh, A.K. Towards index-based similarity search for protein structure databases. *Proc. IEEE Comput. Soc. Bioinform. Conf., 2*. 148-158.

13. Can, T. and Wang, Y.F. Protein structure alignment and fast similarity search using local shape signatures. *J. Bioinform. Comput. Biol., 2* (1). 215-239.

14. Cramer, R.D., Redl, G. and Berkoff, C.E. Substructural analysis. Novel approach to the problem of drug design. *J. Med. Chem.*, *17*. 533-535.

15. Dury, L., Latour, T., Leherte, L., Barberis, F. and Vercauteren, D.P. A New Graph Descriptor for Molecules Containing Cycles. Application as Screening Criterion for Searching Molecular Structures within Large Databases of Organic Compounds. *J. Chem. Inf. Comput. Sci.*, *41*. 1437-1445.

16. http://www.molinspiration.com/jme/.

17. J. Gasteiger, T.E. Chemoinformatics. (Weinheim, WEILEY-VCH Publisher).

18. J. Xu, A.H. Review: Chemoinformatics and drug discovery. *Molecules*, *7*. 566-600.

19. Jenwitheesuk, E. and Samudrala, R. Prediction of HIV-1 protease inhibitor resistance using a protein-inhibitor flexible docking approach. *Antivir. Ther.*, *10* (1). 157-166.

20. Jenwitheesuk, E. and Samudrala, R. Virtual screening of HIV-1 protease inhibitors against human cytomegalovirus protease using docking and molecular dynamics. *Aids*, *19* (5). 529-531.

21. John W. Raymond, E.J.G., Peter Willett RASCAL: calculation of graph similarity using maximum common edge subgraphs. *British computer society*, *45*. 631-644.

22. Jonathan Chen, S.J.S., Yimeng Dou, Jocelyne Bruand, Pierre Baldi ChemDB: a public database of small molecules and related chemoinformatics resources. *Bioinformatics*, *21*. 4133-4139.

23. L. P. Cordella, P.F., C. Sansone, M. Vento Performance evalutation of the VF graph matching algorithm. *Proc. of the 10th ICIAP, IEEE computer socieity press*. 1172-1177.

24. Leach, A.R. and Gillet, V.J. *An Introduction to Chemoinformatics.* Springer, Netherlands, 2003.

25. Lynch, M.F. Introduction of computers in chemical structure information systems, or what is not recorded in the annals. *American society for information science and technology*. 137-148.

26. Mangano, S. XSLT Cookbook, Second Edition (Cookbooks (O'Reilly)) [ILLUSTRATED]

27. McKay, B.D. Practical graph ismorphism. *Congressus Numerantium, 30*. 45-87.

28. Monev, V. Introduction to Similarity Searching in Chemistry. *Match-Communications in Mathematical and in Computer Chemistry 51*. 7-38.

29. NCI NCI. National Cancer Institute.

30. P. Volarath, H.W., H. Fu, R. Harrison Knowledge-based algorithms for chemical structure and property analysis. *EMBS 26th IEEE EMBS Annual International Conference, San Francisco, CA 2004*.

31. Price, J. JDBC programming in Oracle 9i. (McGraw-Hill Publisher, Berkeley, CA).

32. Ray, E.T. Learning XML, Second Edition [ILLUSTRATED].

33. Ray, L.C. and Kirsch, R.A. Finding Chemical Records by Digital Computers. *Science, 126*. 814-819.

34. Read, R.C. and Coreneil, D.G. The Graph Isomorphism Disease. *Journal of Graph Theory 1*. 339-363.

35. Reddy, M.R., Viswanadhan, V.N. and Weinstein, J.N. Relative differences in the binding free energies of human immunodeficiency virus 1 protease inhibitors: a thermodynamic cycle-perturbation approach. *Proc. Natl. Acad. Sci. USA, 88* (22). 10287-10291.

36. Reynolds, X.C.a.C.H. Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients. *Journal of Chem. Info. Comput. Sci., 42*. 1407-1414.

37. Robert D. Brown, G.J., Peter Willett Matching two-dimensional chemical graphs using genetic algorithm. *Journal of inf. comput. sci., 34*. 63-70.

38. Robert D. Brown, G.M.D., Gareth Jones, Peter Willett Hyperstructure model for chemical structure handling: techniques for substructure searching. *Journal of Chem. Info. Comput. Sci., 34*. 47-53.

39. Salim, N., Holliday, J. and Willett, P. Combination of Fingerprint-Based Similarity Coefficients Using Data Fusion. *J. Chem. Inf. Comput. Sci., 43*. 435-442.

40. Savle, R. Improved SMILES substructure searching.

41. Sergey V. Trepalin, A.V.S., Konstantin V. Balakin, Anatoly F. Nasonov, Stanley A. Lang, Andrey A. Ivashchenko, Nikolay P. Savchuk Advanced exact structure searching in large databases of chemical compounds. *Journal of Chem. Info. Comput. Sci., 2003*. 852-860.

42. Tennison, J. Beginning XSLT 2.0: From Novice to Professional (Beginning: from Novice to Professional)

43. Tie, Y., Boross, P.I., Wang, Y.F., Gaddis, L., Liu, F., Chen, X., Tozser, J., Harrison, R.W. and Weber, I.T. Molecular basis for substrate recognition and drug resistance from 1.1 to 1.6 angstroms resolution crystal structures of HIV-1 protease mutants with substrate analogs. *Febs J.*, *272* (20). 5265-5277.

44. Ullman, J.R. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery, 23*. 31-42.

45. Verkhivker, G.M., Rejto, P.A., Gehlhaar, D.K. and Freer, S.T. Exploring the energy landscapes of molecular recognition by a genetic algorithm: analysis of the requirements for robust docking of HIV-1 protease and FKBP-12 complexes. *Proteins, 25* (3). 342-353.

46. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci., 28*. 31-36.

47. Weininger, D., Weininger, A. and Weininger, J.L. SMILES. 2. Algorithm for generation of unique SMILES notation. *J. Chem. Inf. Comput. Sci., 29*. 97-101.

48. Weskamp, N., Kuhn, D., Hullermeier, E. and Klebe, G. Efficient similarity search in protein structure databases by k-clique hashing. *Bioinformatics, 20* (10). 1522-1526.

49. Wild, D.J. and Blankley, C.J. Comparison of 2D Fingerprint Types and Hierarchy Level Selection Methods for Structural Grouping Using Ward's Clustering. *J. Chem. Inf. Comput. Sci., 40*. 155-162.

50. Wildner, G. and Thurau, S.R. Database screening for molecular mimicry. *Immunol. Today, 18* (5). 252, doi:210.1016/S0167-5699(1097)81665-81669

51. Willett, P. Search techniques for database of two and three-dimensional chemical structures. *Journal of Medicinal Chemistry, 48*.

52. Willett, P., Winterman, V. and Bawden, D. Implementation of nearest-neighbor searching in an online chemical structure search system. *J. Chem. Inf. Comput. Sci., 26*. 36-41.

53. Wipke, W.T., Krishnan, S. and Ouchi, G.I. Hash Functions for Rapid Storage and Retrieval of Chemical Structures. *J. Chem. Inf. Comput. Sci., 18*. 32-37.

54. Wiswesser, W.J. *A Chemical Line-Formula Notation.* Crowell Co., New York, 1954.

55. www.stn-international.de.

56. Xu, Y. and Johnson, M. Algorithm for naming molecular equivalence classes represented by labeled pseudographs. *J. Chem. Inf. Comput. Sci., 41* (1). 181-185.

57. Xue, L., Godden, J.W., Stahura, F.L. and Bajorath, J. Design and evaluation of a molecular fingerprint involving the transformation of property descriptor values into a binary classification scheme. *J. Chem. Inf. Comput. Sci., 43*. 1151-1157.

58. Zhu, Z., Schuster, D.I. and Tuckerman, M.E. Molecular dynamics study of the connection between flap closing and binding of fullerene-based inhibitors of the HIV-1 protease. *Biochemistry, 42* (5). 1326-1333.

59. Zupan, J. *Algorithms for chemists*. John Wiley & Sons, New York, 1989.