

Cube Specifications and Other Details from PhD Thesis

Hazel Jane Webb

June 9, 2010

1 Real Data Cubes

Details of the data cubes, and how they were extracted from the raw data, are given in the following sections.

2 Census

Census data were downloaded from <http://kdd.ics.uci.edu/> [?] in comma-separated format. The 27 dimensions used in cubes C1 and C2 were extracted using the Unix awk utility and the commands used to extract cube C1 and to remove duplicates are given in Command 3.1 and Command 3.2. The shaded dimensions of Table 1 are those used in our experiments. A comma-separated file that incorporated the measures, wage per hour and dividends from stock, was loaded into a MySQL database.

Command 2.1 Awk command to extract cube C1

```
awk -F',' '{print $1,"$2","$3","$4","$5","$7","$8",  
"$9","$10","$11","$12","$13","$14", "$15", "$16","$20",  
"$21","$22","$23","$24","$26","$27","$28","$29","$30",  
"$31"," $40","}' census-income.data > census_count.csv
```

Command 2.2 Remove duplicate rows from census data.

```
sort census_count.csv | uniq > census_count_u.csv
```

3 TWEED

The TWEED data set was downloaded from <http://folk.uib.no/sspje/tweed.htm> [?] in SPSS format. It was transformed using SPSS to a comma-separated flat file, which was the required input format for all our implementations. Details of the dimensions used are given in Table 2. The CSV file was loaded into a MySQL database.

4 Forest

Cube F1 was extracted from forestcover.dat, a file created by Liu [?] from Forest CoverType stored in the UCI KDD archive [?] The first eleven columns were copied into ForestCover.csv, which was then sorted and duplicates removed:

We used the command `sed` to remove lines parentheses, ampersands and periods.

The resulting file has 580 950 lines.

Table 1: Census Income data: dimensions and cardinality of dimensions. Shaded dimensions and measures are those retained for cubes C1, C2 and C3. Dimension numbering maps to those described in the file *census-income.names* [?].

dim	dim cardinality	
d0	91	age
d1	9	class of worker
d2	52	detailed industry recode
d3	47	detailed occupation recode
d4	17	education
d6	3	enrol in edu inst last wk
d7	7	marital stat
d8	24	major industry code
d9	15	major occupation code
d10	5	race
d11	10	Hispanic origin
d12	2	sex
d13	3	member of a labour union
d14	6	reason for unemployment
d15	8	full or part time employment stat
d19	6	tax filer stat
d20	6	region of previous residence
d21	51	state of previous residence
d22	39	detailed household and family stat
d23	8	detailed household summary in household
d25	10	migration code-change in msa
d26	9	migration code-change in reg
d27	10	migration code-move within reg
d28	3	live in this house 1 year ago
d29	4	migration prev res in sunbelt
d30	7	num persons worked for employer
d39	53	weeks worked in year
measures used for C2 and C3		
d5	1240	wage per hour
d18	1478	dividends from stocks
unused dimensions		
d39	2	year
d16	132	capital gains
d17	113	capital losses
d24	99800	unknown
d30	5	family members under 18
d31	43	country of birth father
d32	43	country of birth mother
d33	43	country of birth self
d34	5	citizenship
d35	3	own business or self employed
d36	3	fill inc questionnaire for veteran's admin
d37	3	veterans benefits

Table 2: Measures and dimensions of TWEED data. Shaded dimensions are those retained for TW1. All dimensions were retained for cubes TW2 and TW3 (with total people killed as the measure for TW3).

	Dimension	Dimension cardinality
d1	Day	32
d2	Month	13
d3	Year	53
d4	Country	16
d5	Type of agent	3
d6	Acting group	287
d7	Regional context of the agent	34
d8	Type of action	11
d31	State institution	6
d32	Kind of action	4
d33	Type of action by state	7
d34	Group against which the state action is directed	182
d50	Group's attitude towards state	6
d51	Group's ideological character	9
d52	Target of action	11
Measure		
d49	total people killed	
people from the acting group	military	police
civil servants	politicians	business executives
trade union leaders	clergy	other militants
civilians		
total people injured		
acting group	military	police
civil servants	politicians	business
trade union leaders	clergy	other militants
civilians		
total people killed by state institution		
group members	other people	
total people injured by state institution		
group members	other people	
arrests	convictions	executions
total killed by non-state group at which the state directed an action		
people from state institution	others	
total injured by non-state group		
people from state institution	others	

Command 4.1 Sort and remove duplicates.

```
sort -n -t ', ' -u ForestCover.csv > fcover_unique.csv
```

5 King James Bible

Unix utilities were used to extract a fact table from the 4-gram and 10-gram files (kj4d.csv and kj10d.csv) [?] generated by the kingjames script—kingjames.py—described in Section ???. First kingjames4d.csv was sorted and the unique records piped to kj4d.csv (Command 6.1).

Command 5.1 Unique records from kingjames4d.csv sorted and piped to kj4d.csv

```
sort kingjames4d.csv | uniq -c > kj4d.csv
```

The ‘-c’ flag recorded a count for each instance of a line. This count was used as the measure in the sum-diamond experiments on cube K1. Further, it was necessary to move the count value to the final field of the fact table in preparation for loading into the MySQL database—all fact tables were assumed to have the format *attr0,attr1,...,attrd,measure*. All commas were replaced by spaces (Command 6.2).

Then the fields containing data and the measure were written to a new file (Command 6.3).

A similar approach was employed for the 10-dimensional version.

Command 5.2 All commas replaced by spaces

```
sed -i 's/,/, /g' kj4d.csv
```

Command 5.3 Data and measure fields written to kj4dWithMeasures.csv

```
awk -F ' ' '{print $2","$3","$4","$5","$1","}' kj4d.csv  
> kj4dWithMeasures.csv
```

6 Netflix

The Netflix training set was downloaded from <http://www.netflixprize.com> [?]. It comprises 1 770 files in a zipped archive. Each file contains the dates and ratings given by distinct reviewers to one movie. The first line of the file indicates the movie number and the remaining data is in CSV format with three columns (reviewer, rating, date).

The files were unzipped and then concatenated, adding a column containing the file number and moving the rating to the final column. The format of the resulting CSV file was four columns (movie number, reviewer, date, rating).

7 Weather

The weather data set contains 124 million surface synoptic weather reports from land stations for the 10-year period from December 1981 through November 1991, except January 1982. Each report is 56 characters in length. Files with the format <MONTH><YEAR>.DAT.Z were downloaded from <http://cdiac.ornl.gov/ftp/ndp026b/> [?]. In total 119 data files were used. The files were unzipped and amalgamated to a single file (Command 8.1).

Command 7.1 Unzip and concatenate the files

```
zcat *.Z > weather_large.DAT
```

A comma-separated file was created by executing a Java application, which retained the attributes listed in Table 3 and excluded attributes 7, and 12–16. (The code is available at <http://www.hazel-webb.com/archive.htm>.) The resulting file was loaded into a MySQL database. Figure 1 gives a sample input line and its corresponding output line.

8 SQL Queries

Tables census_stocks, tweed and weather_large were created by loading the relevant CSV files into a MySQL database as described in Appendix 2. Cubes C2, TW3 and W2 were extracted from the MySQL database using Listing 1, Listing 2 and Listing 3, respectively. They were also exported in CSV so that they could be used by the other implementations.

Listing 1: Extract C2 from the census income data

```
INSERT INTO temp(attr0,attr1,attr2, attr3, attr4,attr5,attr6,attr7, attr8,  
attr9,attr10,attr11,attr12, attr13, attr14,attr15,attr16,attr17, attr18,  
attr19,attr20,attr21,attr22, attr23, attr24,attr25,attr26, measure)  
SELECT attr0,attr1,attr2, attr3, attr4,attr5,attr6,attr7, attr8,  
attr9,attr10,attr11,attr12, attr13, attr14,attr15,attr16,attr17, attr18,  
attr19,attr20,attr21,attr22, attr23, attr24,attr25,attr26, sum(measure)  
FROM census_stocks  
GROUP BY attr0,attr1,attr2, attr3, attr4,attr5,attr6,attr7, attr8,  
attr9,attr10,attr11,attr12, attr13, attr14,attr15,attr16,attr17, attr18,  
attr19,attr20,attr21,attr22, attr23, attr24,attr25,attr26;
```

82040100,1,8250,29767,71082,71,8,8,5,5,10,-1,800,900,0,0,8,9,4
82040100,1,8250,29767,71082,71,5,10,-1,8,9,4

Figure 1: Sample input line from weather_large.csv and its corresponding output line.

Table 3: Weather data: dimensions and cardinality of dimensions for cubes W1 and W2.

	Dimension	Dimension cardinality
attr0	year,month,day,hour	28 767
attr1	sky brightness	2
attr2	latitude × 100	4 337
attr3	longitude × 100	6 344
attr4	station number	8696
attr5	present weather	101
attr6	low cloud type	13
attr8	high cloud type	14
attr9	change code	11
attr10	solar altitude	10
attr11	relative lunar illuminance	1 799
ignored	land/ocean indicator	226
Measure	total cloud cover	
Unused measures		
	lower cloud amount	
	lower cloud base height	
	middle cloud amount × 100	
	high cloud amount × 100	
	middle cloud amount(non overlapped)	
	high cloud amount (non overlapped)	

Listing 2: Extract TWEED cube

```

INSERT INTO tweed15( d1, d2, d3, d4, d5, d6, d7, d8, d31, d32, d33, d34, d50, d51, d52,
d49 )
SELECT d1, d2, d3, d4, d5, d6, d7, d8, d31, d32, d33, d34, d50, d51, d52, sum( d9 )
FROM tweed
GROUP BY d1, d2, d3, d4, d5, d6, d7, d8,d31, d32, d33, d34, d50, d51, d52

```

Listing 3: Extract weather cube

```

INSERT INTO weather12d(attr0, attr1, attr2, attr3, attr4, attr5, attr6, attr7, attr8,
attr9, attr10, attr11, measure)
SELECT attr0, attr1, attr2, attr3, attr4, attr5, attr9, attr10, attr11, attr16, attr17,
attr18, sum(attr6)
FROM weather_large
GROUP BY attr0, attr1, attr2, attr3, attr4, attr5, attr9, attr10, attr11, attr16, attr17,
attr18;

```

9 Implementation of Algorithm ??

Part of the Java code implementing Algorithm ?? is given in Listing 4. It assumes that a MySQL database table with the column names “attr1, attr2, ... attr*d*, measure, flag” is populated with the cube of interest.

Listing 4: Implementation of Algorithm ??

```

/*****
 * DBSQL4.java
 * implementation of SQLA using jdbc

 * Sets a delete flag to true when cell is deleted from cube and rebuilds whenever
 * 75% of remaining cells are marked for deletion
 *
 *****/

import java.sql.*;
import java.util.*;

public class DBSQL4 {

    private static Statement stmt;
    private static Statement countStmt;
    private static Statement deleteStmt;
    private static String table;
    private static String table_base;
    private static int k; //carats
    private static int d; //number of dimensions
    private static int cells;
    private static int cellsDeleted;
    private static Connection con;
    private static final double FACTOR = .75;

    public static void main(String[] args) {

        // process command line arguments for table name, number of dims, number of carats - code
        // omitted
        // connect to the database - code omitted
        // call process

    }

    /**
     process for a k-carat diamond.
     Sets delete flag to true as it processes.
    */
    public void process() {

        boolean deletedSome = false;
        boolean moreToDelete = false;

        try {
            moreToDelete = true;
            //iterate until we have found the diamond
            while (moreToDelete) {
                moreToDelete = false;
                //loop through dims in single iteration
                for (int dim = 0; dim < d; dim++) {
                    if (countAndDelete("attr"+dim,dim,k))
                        moreToDelete = true;
                }
            }
        } catch (Exception e) {}
        //catch exceptions - code omitted
    }

    /**
     countAndDelete
     @param col dimension name
     @param dim dimension number
     @param k number of carats
     table table name is static
     queries table for the list of unique attributes and stores in table temp
     sets delete flag of attributes with count < k
    */
    public static boolean countAndDelete(String col, int dim, int k) {
        String statement = "";
        ResultSet countsRS;
        boolean deletedSome = false;
    }
}

```

```

try {
    stmt = con.createStatement();
    countStmt = con.createStatement();
    deleteStmt = con.createStatement();

    if (cells > 0 && cellsDeleted > 0 && cellsDeleted > cells*FACTOR) {
        if (table.charAt(table.length()-1) == '2')
            table_base = table.substring(0,table.length()-1);
        else
            table_base = table + "2";
        //rebuild the table and use table2 from now on to process
        deleteStmt.executeUpdate("DELETE FROM " + table_base);
        //clear out any old data then insert the next iteration
        deleteStmt.executeUpdate("INSERT into " + table_base + " SELECT * from " + table
            + " WHERE flag = false");
        System.out.println("rebuild table when " + cellsDeleted + " cells were deleted");
        table = table_base;
        cells = cells - cellsDeleted; //rebuild the table as often as necessary swapping
            between table and table2
        cellsDeleted = 0; //reset the counter
    }

    //ensure we have no leftover data in temp
    stmt.executeUpdate("DELETE FROM temp");

    //get attribute counts and store in temp
    int rows = countStmt.executeUpdate("INSERT INTO temp SELECT "+ col + " as col,
        count(" + col + ") as cnt FROM "+ table + " WHERE flag = false GROUP BY "+ col);
    if (rows > 0) {
        //delete attributes with count < k from table
        int del= deleteStmt.executeUpdate("UPDATE " + table + ",temp SET flag = true
            WHERE temp.attr0 = "+table+"."+col+ " AND temp.attr1 < " +k);
        if (del > 0) {
            deletedSome= true;
            cellsDeleted = cellsDeleted + del;
            System.out.println("cellsDeleted and del " + cellsDeleted + " \t" + del);
        }
    }
} catch (Exception e) {
}
// catch Exceptions - code omitted
return deletedSome;
}
}

```

9.1 Algorithm ?? on Different Database Engines

We were interested in the effect of other database engines on our algorithms, so we implemented SQL4 on a different test machine. The SQL queries were run against MySQL version 5.0.41-community and SQL Server 2008 Enterprise edition version 10.0.1600.22 on a Dell 1750 with a 2.8 GHz Intel processor 80532K and 2 GiB RAM. It was running Windows Server 2003.

We used W3 and NF4, which comprised the first 500 000 cells of NF1. Results are given in Table 4. Times in seconds are averaged over 10 runs. We see, for this particular test, that MySQL is almost twice as fast as SQL Server. Although we cannot make a general statement that all databases are too slow at computing diamonds, we do see that these two popular RDBMSs are both slower than the binary Java implementation.

cube	engine	Preprocessing	Processing
W3	SQL Server	189	192
	MySQL	27	103
	binary	51	25
NF4	SQL Server	31	57
	MySQL	7	28
	binary	12	3

Table 4: Comparison of SQL Server and MySQL. Times (in seconds) are averaged over 10 runs.

Figure 2: Cells remaining in DCLD1 after the duplicates were removed.

```

0, 0, 10, 1, 7, 8, 1, 9, 9, 2, 6, 5, 2, 8, 5, 3, 1, 8, 3, 2, 8, 4, 2, 3,
4, 3, 8, 4, 4, 0, 4, 7, 3, 4, 7, 4, 4, 7, 9, 4, 8, 8, 4, 9, 9, 5, 2, 3,
5, 4, 8, 5, 7, 5, 5, 8, 7, 5, 9, 6, 6, 2, 6, 6, 4, 6, 6, 5, 5, 6, 6, 4,
6, 7, 5, 6, 7, 8, 6, 8, 3, 6, 9, 2, 7, 5, 3, 7, 6, 4, 7, 6, 7, 7, 7, 7,
7, 8, 8, 8, 1, 8, 8, 2, 3, 8, 4, 0, 8, 4, 6, 8, 4, 8, 8, 5, 3, 8, 5, 6,
8, 5, 9, 8, 6, 4, 8, 6, 5, 8, 7, 4, 8, 8, 3, 8, 8, 7, 8, 8, 9, 8, 9, 5,
9, 1, 2, 9, 3, 3, 9, 6, 9, 9, 8, 5, 10, 6, 7, 10, 6, 8, 10, 6, 9, 10, 8, 3,
1, 11, 3, 11, 3, 5, 11, 4, 3, 11, 4, 4, 11, 4, 9, 11, 5, 5, 2, 11, 8, 2, 3, 10,
3, 11, 0, 3, 11, 5, 3, 11, 9, 3, 4, 10, 3, 5, 11, 3, 8, 10, 4, 10, 5, 4, 11, 9,
4, 3, 10, 4, 3, 11, 4, 6, 10, 5, 11, 1, 5, 11, 3, 5, 11, 8, 6, 11, 9, 6, 6, 10,
6, 6, 11, 6, 7, 10, 7, 11, 6, 7, 6, 11, 7, 8, 10, 8, 4, 10, 9, 10, 4, 9, 10, 9,
9, 2, 11, 10, 2, 11, 11, 9, 11, 9, 11, 10,

```

10 Algorithm to Generate DCLD1

As discussed in Section ?? synthetic cubes were generated to test our DCLD heuristic. Algorithm 1 was used to generate cube DCLD1. A similar approach with different weights was used to generate cube DCLD2. Figures 2 and 3 list the cells in cubes DCLD1 and DCLD2, respectively.

```

prob ← 0.1
for i ∈ 1...100 do
  for j ∈ 1...d do
    // generate a uniform random number in the range 0-119
    x ← ran(0 - 119)
    if x < 120 * prob then
      // generate a uniform random number in the range 0-2
      chosenj ← ran(0 - 2)
    else
      // 90% of the time we will generate a uniform random number
      // in the range 3-11
      chosenj ← ran(3 - 11)
    // write chosen to cube DCLD1

```

Algorithm 1: Generates DCLD1: values 0-2 have a 10% chance of being chosen in each dimension.

Figure 3: Cells remaining in DCLD2 after the duplicates were removed.

```

0,0,10, 0,0,2, 0,10,1, 0,1,10, 0,1,11, 0,1,3, 0,1,8, 0,2,3,
0,3,0, 0,5,6, 0,7,1, 0,9,0, 10,0,1, 1,0,1, 1,0,2, 1,0,6,
1,0,7, 1,10,5, 11,0,6, 1,1,1, 1,1,10, 11,1,0, 1,11,10, 11,11,4,
11,1,2, 1,11,6, 1,11,8, 11,3,2, 11,8,2, 11,9,0, 11,9,11, 1,2,1,
1,5,8, 1,6,1, 1,6,10, 1,7,1, 1,9,10, 1,9,9, 2,0,0, 2,0,5,
2,1,0, 2,10,0, 2,10,1, 2,10,2, 2,1,1, 2,1,11, 2,1,7, 2,2,1,
2,2,5, 2,2,8, 2,4,7, 2,5,4, 2,6,11, 2,6,7, 2,8,10, 2,9,2,
2,9,9, 3,0,2, 3,11,11, 3,11,3, 3,5,11, 3,9,8, 4,0,1, 4,1,1,
4,2,0, 4,5,1, 4,9,1, 5,0,3, 5,10,1, 5,1,2, 5,2,0, 5,4,8,
5,5,2, 5,6,11, 6,0,9, 6,10,0, 6,5,1, 7,0,6, 7,0,9, 7,1,1,
7,9,10, 8,0,0, 8,1,11, 8,2,2, 8,4,1, 8,4,4, 8,7,4, 8,9,8,
9,0,0, 9,1,1, 9,2,2, 9,2,7, 9,6,8, 9,7,8,

```

11 Java Code to Implement Algorithm ??

Part of the Java preprocessing and diamond building applications are given in Listing 5 and Listing 6. The preprocessor is common to both string and binary implementations. We list `DBCount_binary.java` and note that the string implementation only differs in the data type and the data structure used to keep track of counts.

Listing 5: Preprocessor.

```

/*****
 * DBCountPreprocessor.java
 * Preprocesses csv file outputs are used by DBCount_binary or DBCount to
 * process for K.
 *****/
import java.util.*;
import java.io.*;

public class DBCountPreprocessor {

    private int d;
    //number of data dims
    private String file;
    //store input file name
    private Vector<Integer>[] data;
    //data structure ragged array of Vectors to store counts for normalized attributes
    private Hashtable[] mappedCoords;
    //ds maps attribute values to their counts
    private Hashtable[] mappedNorms;
    //ds helps to transform data to normalized integers
    private boolean norm = true; //default—preprocess for binary data
    private boolean hash = true; //default—preprocess for string data
    public static void main(String[] args) {

        // parse command line arguments for file name and number of dims — code omitted

        // call createHashTables

        // call writeHashTables: writes hashtables to file for future processing. Java does not
        // guarantee order of elements returned from a Hashtable, so it is necessary to run
        // the (key,value) pairs through Enumerations in order to keep a persistent and
        // accurate record — code omitted
        // call writeArrays: writes binary file of packed integers — code omitted
        // call writeNormFile: writes normalized data to csv file and a file of packed integers
        // for future binary processing — code omitted
    }

    /**
     createHashTables
     @param fname String of the csv fact table to process.
     CSV file should have no headers or metadata.
     creates d hashtables of <key(attribute), value(count)> pairs

```

```

creates d hash tables of <key(attribute)>,value(normvalue)> pairs
creates array of vectors that store counts for the normalized attributes
*/
public void createHashTables(String fname, int d) {
    BufferedReader raw;
    PrintWriter permanent_out;

    // initialize 'em happens regardless of output choice...
    for (int i = 0; i < d; i++) {
        mappedCoords[i]=new Hashtable<String,Integer>();
        mappedNorms[i] = new Hashtable<String,Integer>();
    }
    // map all of the attributes to their counts
    try {
        permanent_out = new PrintWriter(new BufferedWriter(new
            FileWriter(fname+"-string-to-norm")));
        raw = new BufferedReader(new InputStreamReader(new FileInputStream(fname)));
        String line = raw.readLine();
        Integer val = new Integer(1);
        Integer[] normVal = new Integer[d];
        //ds to store the normalized value for the next key in each dimension

        for (int i = 0; i < d; ++i)
            normVal[i] = new Integer(0);

        while (line!=null) {
            //ensure a comma at the end of the line so split will extract all the fields
            if (!(line.charAt(line.length()-1)==','))
                line = line + ",";
            String[] atts = line.split(",");
            for (int k = 0; k < d; ++k) {
                String key = atts[k];
                if (hash) {
                    if (!mappedCoords[k].containsKey(key)) {
                        //if it wasn't mapped yet

                        mappedCoords[k].put(key,val);
                        //add this key to mappedCoords hash
                    } else {
                        //increment count for this attribute
                        Integer intVal = (Integer)mappedCoords[k].get(key);
                        mappedCoords[k].put(key,intVal + val);
                        //increase count of this key in mappedCoords hash
                    }
                }

                if (norm) {
                    if (!mappedNorms[k].containsKey(key)) {
                        //if it wasn't mapped yet

                        mappedNorms[k].put(key,normVal[k]);
                        //add this key to mappedNorms hash and store the relationship
                        permanently
                        permanent_out.println("dim: "+ k + "\t"+ key
                            +"\t"+normVal[k].intValue());

                        data[k].add(val);
                        //add a place holder for counts in the appropriate vector

                        normVal[k] = (Integer)(normVal[k].intValue() + val.intValue());
                        //increase the current norm value for the next key seen
                    } else {
                        //increment count for this attribute
                        Integer count =(Integer) mappedNorms[k].get(key);
                        int accumulator = data[k].get(count);
                        data[k].set(count, accumulator+1);
                        //increase count of this key in the appropriate vector slot
                    }
                }
            }

            line = raw.readLine();
        }
        raw.close();
        permanent_out.close();
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}

```

}
}
}

Listing 6: Diamond builder for binary data.

```

/*****
 * DBCount_binary.java
 * input: 1) preprocessed binary file (output of DBCountPreprocessor.java)
 *         format of input file:
 *         number of dims, cardinalities for each dim, data – all integers
 * 2) k: number of carats
 *****/
import java.util.*;
import java.io.*;

public class DBCount_binary {

    private int d;    //number of data dims
    private int K;    //number of carats
    private int iters = 0; //count number of times the file is processed until convergence
    private String file; //to store input file name from command line
    int[][] data; //ragged array of arrays storing counts of the attrvals
    private int FILE_NAME_LENGTH;

    public static void main(String[] args) {

        // parse command line arguments for file name and number of carats K — code omitted
        // call loadArrays: read in counts for each attribute in each dimension — code omitted
        // call initArrays: sets count to zero for every attribute whose count is less than K —
        // code omitted

        // call process

        // call writeDiamond: writes cells remaining in diamond to csv file — code omitted
        // call writeDiamondCounts: writes all attribute values and their current count to file
        // — code omitted
    }

    /* process
     * counts for dim i are stored in data(i)
     * a count of zero removes this attr from the diamond
     */
    public void process(String inFile, String outFile) {

        File file = null;
        iters++;
        boolean reprocess = false; //assume this will be the last time through the input file

        try {
            FileInputStream file_input = new FileInputStream (new File(inFile));
            DataInputStream data_in = new DataInputStream( new BufferedInputStream(file_input
            ));

            file = new File(outFile);
            DataOutputStream data_out = new DataOutputStream(new BufferedOutputStream(new
            FileOutputStream(file)));

            int[] attrs = new int[d]; //somewhere to store each line of fact table
            boolean del = false;
            try {
                while (true) {
                    for (int i = 0; i < d; ++i) {
                        attrs[i] = data_in.readInt();
                        //read in a line of the fact table
                    }

                    del = false;
                    //we haven't yet deleted anything
                    for (int i = 0; i < d; ++i) {
                        //get counts array for dim i
                        if (data[i][attrs[i]] == 0) {
                            del = true;
                            reprocess = true; //
                            //we deleted something so we will need to reprocess the file
                            //this attr is deleted so update others in the row
                            for (int j = 0; j < d; ++j) {
                                if (j != i) {
                                    if (data[j][attrs[j]] > 0) {
                                        data[j][attrs[j]]--;
                                    }
                                }
                            }
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

        //decrement the count
        if (data[j][attrs[j]] < K)
            data[j][attrs[j]] = 0;
        //this should be deleted on next iter
    }
}
if (del) break;
//only delete this row once
}
if (!del) {
    for (int v: attrs) {
        data_out.writeInt(v);
        //write this row to new diamond file
    }
}
} catch (EOFException eof) {
    //do nothing this terminates the loop
}

data_in.close ();
data_out.close();

if (reprocess) {
    if (inFile.charAt(FILE_NAME_LENGTH)=='_')
        (new File(inFile)).delete();
    //and delete the old file except original data
    inFile = inFile.substring(0,FILE_NAME_LENGTH)+"_"+iters +"B";
    //force a new file to be output each time
    process(outFile,inFile);
    //call process again with newly minted data file
}
} catch (IOException e) {
    System.out.println ( "IO Exception =: " + e );
}
}
}

```

12 Words Remaining in $\kappa(K1)$ Diamond

The κ -diamond for K1 was established as described in Section ?? and the words remaining in the κ -diamond are given in Tables 5, 6, 7 and 8. Words that occur in one dimension, but none of the others are shown in boldface.

Table 5: Words remaining in dimension one of $\kappa(K1)$ diamond.

words remaining in dimension one			
adadah	adithaim	adullam	amam
ashnah	azekah	azem	baalah
bealoth	beersheba	bethpalet	bizjothjah
chesil	children	citi	coast
dimonah	eder	edom	eltolad
enam	engannim	eshtaol	gederah
hadattah	hazargaddah	hazarshu	hazor
heshmon	hezron	hormah	ithnan
jagur	jarmuth	judah	kabzeel
kedesh	kerioth	kinah	lebaoth
madmannah	moladah	nine	rimmon
sansannah	sharaim	shema	shilhim
socoh	southward	tappuah	telem
their	toward	tribe	twenti
uttermost	vallei	villag	were
which	with	zanoah	ziklag
ziph	zoreah		

Table 6: Words remaining in dimension two of $\kappa(K1)$ diamond.

words remaining in dimension two			
achzib	adithaim	adullam	ashan
ashdod	ashnah	azekah	azem
baalah	beersheba	bethdagon	bethpalet
bizjothjah	bozkath	cabbon	chesil
citi	dilean	eglon	ekron
eltolad	enam	engannim	eshtaol
ether	even	fourteen	from
gederah	gederoth	gederothaim	hadashah
hazarshu	hormah	jarmuth	jiphtah
joktheel	keilah	kithlish	lachish
lahmam	lebaoth	libnah	madmannah
makkedah	mareshah	migdalgad	mizpeh
naamah	near	nezib	nine
rimmon	sansannah	sharaim	shilhim
sixteen	socoh	tappuah	that
their	town	twenti	unto
vallei	villag	which	with
zanoah	zenan	ziklag	zoreah

Table 7: Words remaining in dimension three of $\kappa(K1)$ diamond.

words remaining in dimension three			
achzib	anab	anim	aphekah
arab	ashan	ashdod	ashnah
bethdagon	bethtappuah	border	bozkath
cabbon	citi	dannah	debir
dilean	dumah	eglon	egypt
ekron	eleven	eshean	eshtemoh
ether	even	from	gaza
gederoth	giloh	goshen	great
hadashah	hebron	holon	humtah
janum	jattir	jiphtah	joktheel
keilah	kirjatharba	kirjathsannah	kithlish
lachish	lahmam	libnah	makkedah
maon	mareshah	migdalgad	mizpeh
mountain	naamah	near	nezib
nine	river	shamir	sixteen
socoh	that	their	thereof
town	unto	villag	which
with	zanoah	zenan	zior

Table 8: Words remaining in dimension four of $\kappa(K1)$ diamond.

words remaining in dimension four			
anab	anim	aphekah	arab
ashdod	bethanoth	betharabah	bethtappuah
bethzur	border	cain	carmel
citi	dannah	debir	dumah
egypt	eleven	eltekton	engedi
eshean	eshtemoh	gaza	gedor
gibeah	giloh	goshen	great
halhul	hebron	holon	humtah
janum	jattir	jezreel	jokdeam
juttah	kirjatharba	kirjathba	kirjathjeirim
kirjathsannah	maarath	maon	middin
mountain	nibshan	nine	rabbah
river	salt	secacah	shamir
socoh	their	thereof	timnah
town	unto	villag	which
wilder	with	zanoah	zior
ziph			