Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

APPLICATION OF APPROXIMATE GRAPH MATCHING TECHNIQUES FOR SEARCHING DATABASES OF TWO-DIMENSIONAL CHEMICAL STRUCTURES

by
Karen R. Pysniak

This paper proposes the application of approximate graph matching techniques for best-match searching of two-dimensional chemical structure databases based upon topology. Chemical structures are represented as labeled graphs, each atom a node in the graph and each bond an edge. By inserting, deleting and renaming nodes/edges, one structure may be transformed into another. We define similarity as the weighted sum of the costs of these edit operations. An algorithm for approximating the minimum distance between two graphs based on simulated annealing is applied. Best-match searches are performed utilizing this pre-computed distance information and applying the concepts of triangle inequality to prune out structures from the database which could not possibly satisfy the query. We also extend the best-match retrieval by allowing subgraphs of a data graph to be freely removed. The result is a technique which extends existing substructure search techniques by allowing certain distances to exist between the query and a substructure of the data graph.

APPLICATION OF APPROXIMATE GRAPH MATCHING TECHNIQUES FOR SEARCHING DATABASES OF TWO-DIMENSIONAL CHEMICAL STRUCTURES

by
Karen R. Pysniak

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

October 1995

APPROVAL PAGE

APPLICATION OF APPROXIMATE GRAPH MATCHING TECHNIQUES FOR SEARCHING DATABASES OF TWO-DIMENSIONAL CHEMICAL STRUCTURES

Karen R. Pysniak

Dr. Jason T. L. Wang, Thesis Advisor — Date
Assistant Professor of Computer and Information Science, NJIT

Dr. James McHugh, Committee Member — Date
Professor of Computer and Information Science, NJIT

Dr. Peter Ng, Committee Member — Date
Chairman and Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Karen R. Pysniak

Degree: Master of Science in Computer Science

Date: October 1995

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1995

- Bachelor of Science in Chemistry,
  Indiana University of Pennsylvania,
  Indiana, Pennsylvania, 1986

Major: Computer Science

This thesis is dedicated
to my parents,
Anna and Theodore Pysniak.

ACKNOWLEDGMENT

The author wishes to express her sincere gratitude to her supervisor, Professor Jason T. L. Wang, for his guidance, encouragement and support throughout this research. Special thanks to Professors James McHugh and Peter Ng for serving as members of the committee. And finally, a thank you to Dr. James J. Kaminski for his suggestions and expert assistance throughout this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Retrieval techniques currently used in chemical information systems may be divided into three categories: (1) *structure*, (2) *substructure* and (3) *similarity searches* [1,2]. The first two categories, *structure* and *substructure searches*, search a database of 2D and 3D molecular structures for a molecule which exactly matches an input molecular structure or substructure. *Similarity searches* retrieve those molecules from the database which are most similar in structure to an input molecular structure. In this thesis we focus on new techniques for performing searches on a database of 2D molecular structures.

The conventional technique for 2D substructure searches is based upon a screening feature which utilizes characteristic fingerprints of molecules to quickly eliminate candidates during the 2D searching process; then atom-by-atom comparisons are performed to determine whether the desired substructure is present in the remaining molecules. A molecular fingerprint is often represented as a wide bit set where each bit indicates the absence or potential presence of a molecular feature. Molecular features may include fragments of varying length, the presence of functional groups, and also the presence of heteroatoms. Searches are performed on inverted bit maps which contain sets of bit strings of a group of structures, each row indicating the substructural fragments present within the structure.

For a substructure search the bit screen of the hit must contain exactly all of the query bits. Thus, the substructure search may not identify structures with minor deviations from the query structure. It is for this reason that similarity searches have been developed. Similarity is based upon the screens present and absent in the query and in the target molecule, however, no atom-to-atom comparison is needed to eliminate molecules which do not exactly match the query structure. Figure 1 illustrates the difference in results of a substructure search and a similarity search.

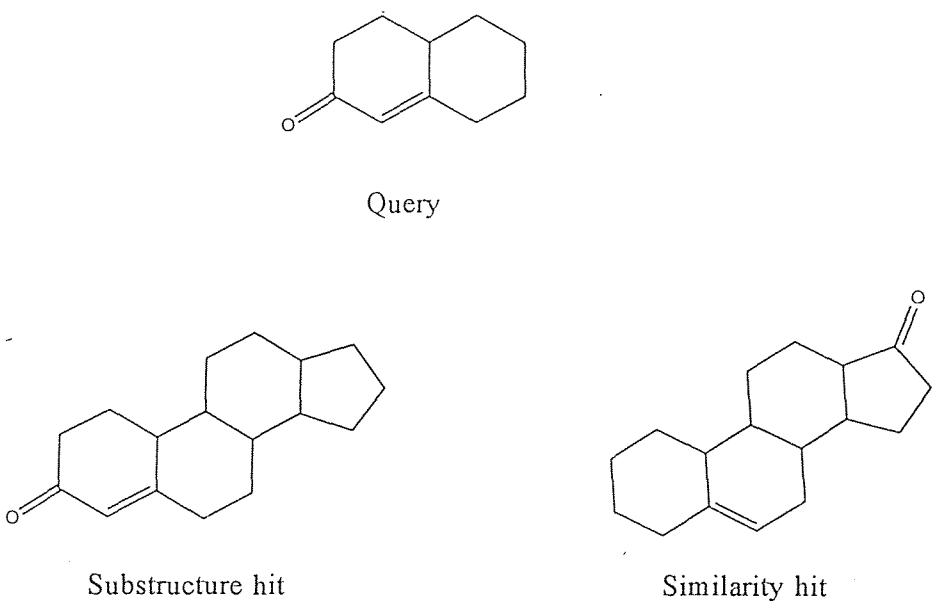


Figure 1. Illustration of results obtained from a substructure search vs. a similarity search.

In this thesis we approach the concept of inter-molecular similarity from the notion of edit distances for labeled graphs. Distance is determined by the minimum cost of sequential edit operations required to transform one graph into the other. Such edit operations consist of inserting, deleting, and relabeling a node or edge in the graph.

Calculation of this distance measure is achieved through simulated annealing, a probabilistic hill climbing algorithm which has been applied in areas such as pattern recognition [3], combinatoric graph theory [4] and query optimization [5,6]. A searching technique based upon triangle inequality is then implemented to perform best-match retrievals on the molecular structures in the database. A best-match query is one which, from a file containing a collection of objects, retrieves the object(s) in the file which are closest to a given target according to some (dis)similarity metric [7]. The best-match technique we present in this thesis makes use of a separate data file which reflects the precomputed intrafile distances between each of the structures in the database based upon topology. It has been shown in previous work, that by utilizing such distance metric information, combined with triangle inequality, the number of comparisons required to achieve the desired results may be significantly reduced [7,8,9,10,11].

In the second part of the thesis, we extend the best-match retrieval by allowing subgraphs of a data graph to be freely removed when matching the target pattern with the data graph. Specifically, given the target and a database of data graphs, we want to find those data graphs approximately containing the target. This type of retrieval extends the substructure search by allowing certain distances to exist between the target and a substructure of the data graph.

CHAPTER 2

TECHNIQUES FOR BEST-MATCH SEARCHING

The techniques presented in this section consist of comparing chemical structures based upon their topology. Chemical structures are compared on a one-to-one basis utilizing a similarity metric based upon edit distances. These distances are then compared to that of an input query to perform best-match searches on the database. This chapter is divided into three sections, each of which outlines a basic concept involved in structure comparison (*similarity metric* and *simulated annealing*) and database searching (*triangle inequality*).

2.1 Similarity Metric

Each molecular structure is represented as a labeled graph in which each atom represents a node in the graph and each inter-atomic bond an edge. Given two graphs, a pattern graph (G) and a data graph (G'), the distance (or similarity) between the two is defined as the minimum cost of all sequences of edit operations needed to transform one graph into the other. We represent an edit operation as $a \rightarrow b$, where a and b are either labeled nodes or labeled edges; Λ denotes the null node. The three types of edit operations that may occur are as follows:

1) node/edge insert

$a \rightarrow b$, where (a = Λ) and (b $\neq \Lambda$)

2) node/edge delete

$a \rightarrow b$, where (a $\neq \Lambda$) and (b = Λ)

3) node/edge relabel

$a \rightarrow b$, where (a $\neq \Lambda$) and (b $\neq \Lambda$)

We impose the following constraints on the edit operations: (1) A node can be deleted only when its degree is zero, *i.e.*, no edge connects to the node; (2) An edge can be inserted only when both of its end nodes are already in the graph. Figure 2 illustrates each of the three edit operations.

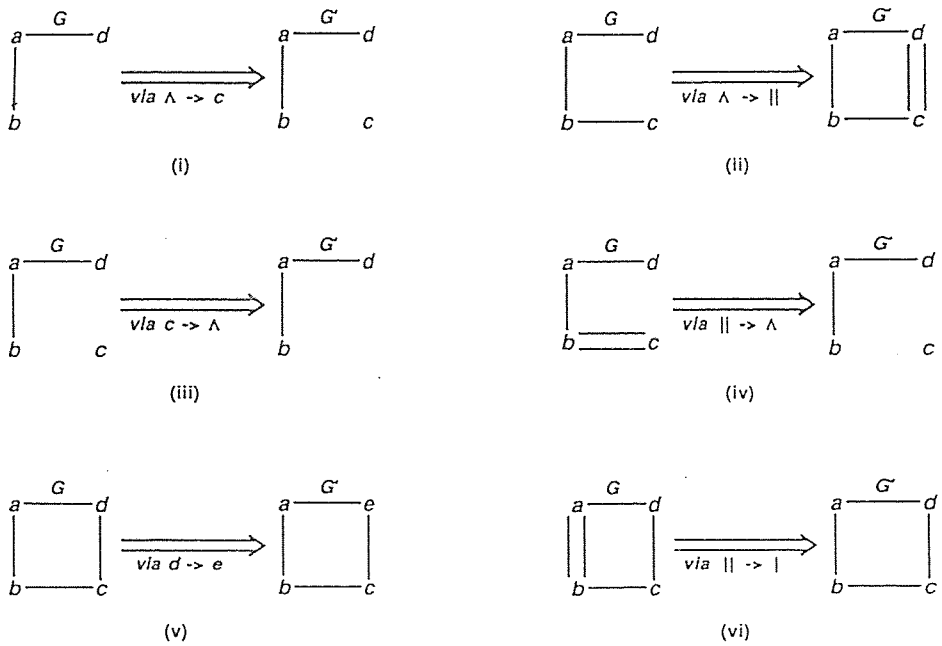


Figure 2. Examples of edit operations. Lower case letters represent node labels; / and // represent edges. (i) and (ii) node/edge insert; (iii) and (iv) node/edge delete; (v) and (vi) node/edge relabel, respectively.

Associated with each edit operation, $a \rightarrow b$, is a cost, $\gamma(a \rightarrow b)$. γ is defined to be a distance metric, that is, it satisfies the following properties:

- non-negative definiteness $\gamma(a \rightarrow b) \geq 0$ and $\gamma(a \rightarrow b) = 0$

- symmetry $\gamma(a \rightarrow b) = \gamma(b \rightarrow a)$

- triangle inequality $\gamma(a \rightarrow c) \leq \gamma(a \rightarrow b) + \gamma(b \rightarrow c)$

Consider the transformation of graph G to G' through a series of edit operations $s_1, s_2, \ldots, s_k$, represented by S. g may be extended to this series of edit operations as follows:

$$\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$$

Thus we define the distance between these two graphs, *dist(G,G')*, as the minimum cost of all sequences of edit operations which transform G to G', i.e.,

$dist(G,G') = \min\{\gamma(S) | S$ is a sequence of edit operations transforming G to $G'\}$

Figure 3 shows the results of a one-to-one comparison of two chemical structures utilizing the edit distance metric.

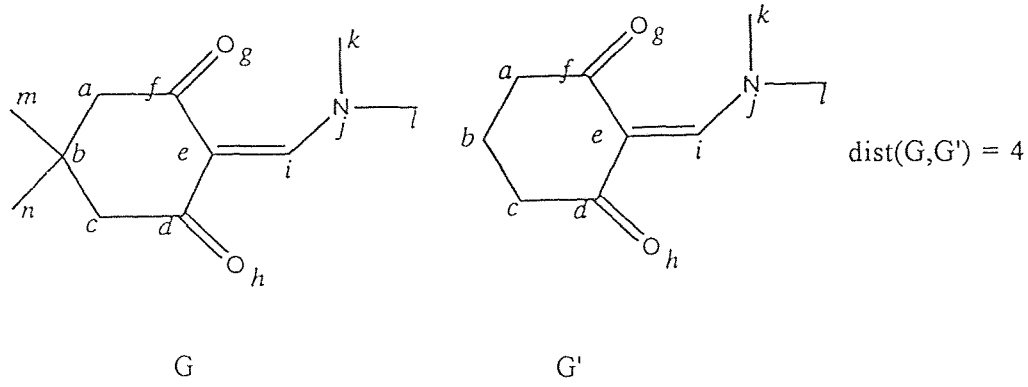
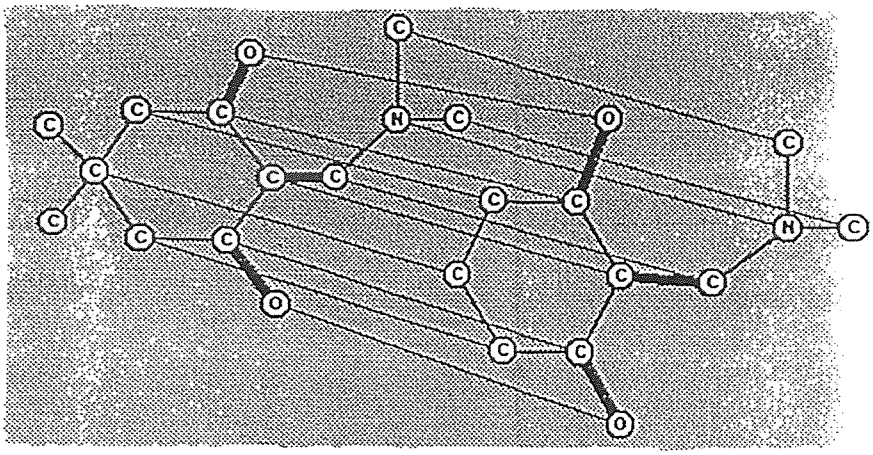


Figure 3. One-to-one graph comparison of chemical structures utilizing the similarity metric based upon edit distances. The four edit operations are as follows: (i) delete node labeled (*m*); (ii) delete edge between nodes (*b*) and (*m*); (iii) delete node labeled (*n*); (iv) delete edge between nodes (*b*) and (*n*).

The edit operations correspond to a *mapping*, a graphical representation of which edit operations apply to each node and edge in the two graphs. Given a sequence of edit operations S, to transform G to G', there exists a mapping, M, such that $g(M) \leq \gamma(S)$ [9]. Conversely, for any mapping M, there exists a sequence of edit operations S such that $\gamma(M) = \gamma(S)$. In [8] it is proven that:

$$dist(G,G') = \min\{\gamma(M) | M \text{ is a mapping from } G \text{ to } G'\}$$

Figure 4 illustrates a mapping that corresponds to the edit operations in Figure 3 (screendump courtesy of Dr. Jason T.L. Wang).



G G'

Figure 4. Graphical mapping to transform chemical structure G to G' through a series of edit operations. The nodes of G not touched by a mapping line are to be deleted and the nodes of G' not touched are to be inserted. The definitions are similar for the edges in the graphs.

2.2 Simulated Annealing

In [8] it is shown that finding dist(G, G') is hard in the sense that it is unlikely that there will be an efficient algorithm for it. We therefore propose to transform the problem into a state search problem. Each mapping, M, is transformed to a state, S_M, which is associated with a cost, where $cost(S_M) = \gamma(M)$. If the cost of the destination state is higher than the cost of the source state, the move is considered *uphill*. Respectively, if the cost of the destination state is lower than that of the source state, the move is considered *downhill*. If in all paths starting at a given state any downhill move comes after at least one uphill move, that state is said to be a *local minimum*. A state is a *global minimum* if it has the lowest cost of all states. It is proven in [8] that the global minimum, r, is equal to $dist(G, G')$.

The simulating annealing optimization algorithm starts at a random state and repeatedly performs downhill moves, as well as uphill moves with some probability, until it reaches a local minimum. The algorithm proceeds in stages under a fixed value of a parameter *Temp*, called *temperature*. This parameter controls the probability of accepting uphill moves; the lower the temperature, the smaller the probability of accepting an uphill move. Each stage exits when *equilibrium* is reached, defined as a certain number of iterations through the loop. The algorithm then reduces the temperature according to some function and another stage begins. The algorithm terminates when it is considered to be *frozen*, *i.e.*, when the temperature is equal to zero. Figure 5 illustrates the simulated annealing optimization algorithm, courtesy of Dr. Jason T.L. Wang, which is outlined below.

```
S := random state;
Temp := initial temperature;
minS := S;
while not (frozen) do
    begin
        while not (equilibrium) do
            begin
                S' := randomly pick a neighbor state of S;
                D(C) := cost(S') - cost(S);
                if D(C) < 0 then S: = S';
                if D(C) ≥ 0 then S := S' with probability e^-Δ(C)/Temp;
                if cost(S) < cost(minS) then minS := S;
            end
        Temp := reduce(Temp);
    end
return(minS, cost(minS));
```

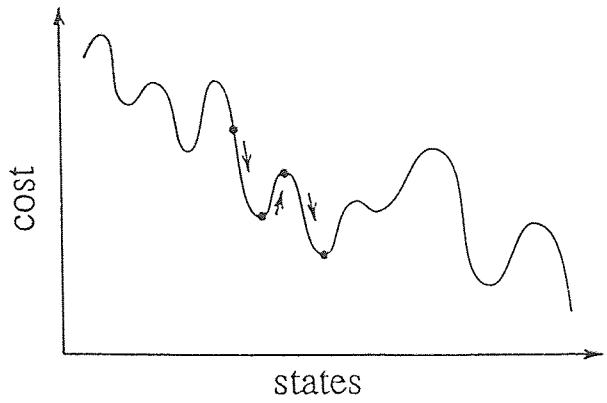


Figure 5. Illustration of simulated annealing technique utilizing uphill and downhill moves to achieve optimization.

2.3 Triangle Inequality

One method of searching the database for a specified input query structure would be to calculate the distance between each chemical structure in the database against that of the query structure and retain that with the least distance as the best-match. This technique, however, can be very consuming of time and resources depending on the size of the database and the individual structures. Instead, we apply the concepts of triangle inequality in searching the database. Structures that could not possibly satisfy a given query are eliminated from consideration based upon precomputed structural distances, eliminating the need to compute the distance between each structure in the database against the query structure.

This approach first precomputes pairwise distances between the structures in the database and stores this information in a separate file. In searching the database for a best-match graph, g_b, to a given input query graphical structure, q, we choose a graph,

g, and compute the distance between the two, which becomes dist(q, g_b). We proceed by choosing another graph, g_i, and comparing dist(q, g_i) with dist(q, g_b). There are two cases which may arise are discussed below.

Case (1): dist(q, g_i) $\geq$ dist(q, g_b). In this first situation we eliminate from consideration any graphs, g, that are farther away from g_i than dist(q, g_i)+dist(q, g_b) or closer to g_i than dist(q, g_i)-dist(q, g_b) since they cannot be a best-match graph. The following equations illustrate the existing conditions to imply that g would never become the best-match graph.

$$\text{dist}(q, g) \geq |\, \text{dist}(g, g_i) - \text{dist}(q, g_i) \,| \qquad \text{(by triangle inequality)}$$

$$> \text{dist}(q, g_i) + \text{dist}(q, g_b) - \text{dist}(q, g_i)$$

$$\text{(if dist}(g, g_i) > \text{dist}(q, g_i) + \text{dist}(q, g_b))$$

- or -

$$\text{dist}(q, g_i) - (\text{dist}(q, g_b)) - \text{dist}(q, g_i)$$

$$\text{(if dist}(g, g_i) < \text{dist}(q, g_i) - \text{dist}(q, g_b))$$

$$= \text{dist}(q, g_b)$$

Case (2): dist(q, g_i) < dist(q, g_b). In the second situation, g_i becomes the current best-match, g_b, and we disregard those graphs which are farther away from g_i than 2×dist(q, g_i). Note the existence of the conditions given in the following equations.

$$\text{dist}(q, g) \geq |\, \text{dist}(g, g_i) - \text{dist}(q, g_i) \,| \qquad \text{(by triangle inequality)}$$

$$2 \times \text{dist}(q, g_i) - \text{dist}(q, g_i)$$

$$= \text{dist}(q, g_i)$$

CHAPTER 3

RESULTS AND DISCUSSION

A series of experiments were performed to evaluate the effectiveness and speed of our algorithms, as well as their performance relative to existing procedures. The algorithms were implemented in the C programming language and run on a Silicon Graphics Indigo2 workstation under the IRIS operating system version 5.2. In simulated annealing we defined *equilibrium* to be ten iterations through the loop as outlined in the previous section. The database consisted of 200 chemical structures randomly chosen from a database of compounds supplied by BIONET [12], each compound containing no more than 50 atoms per structure (or 50 nodes per graph). The original structures were supplied in two-dimensional SD file (Structure-Data file) format. These files were manually converted to two-dimensional MDL (Molecular Design Limited) file format for input to CONCORD 3.0.2 [13], a commercial program for the generation of three-dimensional molecular coordinates. This program generated 3D structure coordinates in MDL format and MOLfile format. A detailed description of the various structure file formats may be found in [14].

3.1 Performance Analysis

In the first set of experiments, we compared the heuristic performance relative to that of a linear exhaustive search. Graphical format structure files were generated manually from the 2D MDL format files. For ease of implementation, the dataset was divided into two subfiles, each containing 100 structures. Intrafile distances were computed for both subfiles and our best-search searching algorithms applied to both sets; the final result is a comparison of the two. All edit operations were assumed to have unit cost.

To begin with, we compare the performance of the two methods in terms of the number of times the simulated annealing algorithm was run (number of comparisons), as well as elapsed CPU time (in seconds). The results are presented in Table 1. Figure 6 illustrates the correlation between the number of atoms in a structure and elapsed CPU time for the two methods.

Table 1. Results of best-match retrieval methods in terms of the number of comparisons and elapsed CPU time.

	Heuristic Method		Exhaustive Method	
Query	Number of Comparisons	CPU Time (Sec.)	Number of Comparisons	CPU Time (Sec.)
A	53	53	200	488
B	96	309	200	1113
C	90	352	200	1441

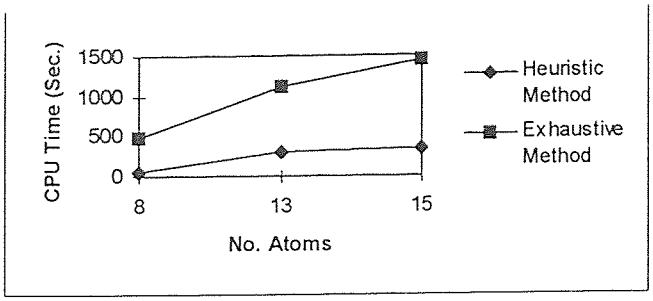


Figure 6. Elapsed CPU time as a function of the number of atoms in the input query.

It can be seen from these results that by utilizing triangle inequality, we significantly reduce the number of comparisons required to perform a search by pruning out structures from the database that could not possibly be best-match candidates. This reduction in comparisons naturally leads to a significant reduction in computation time relative to that required for the exhaustive search. When correlating the size of the input query to that of CPU time the result is not surprising, since the simulated annealing algorithm performs an atom-to-atom comparison when calculating edit distances. Thus, the larger the query, the more time comparisons will consume.

To determine the effectiveness of our algorithms, we compare the actual structures retrieved by the two techniques. Figure 7 presents the chemical structures of the best-match compounds retrieved upon execution of the two algorithms. For queries A and C the results are identical, both techniques retrieving the same structure with the same minimum distance, affirming the effectiveness of our algorithms. The results for query B, however, are inconsistent, leading us to further examine our method.

As mentioned previously, the simulated annealing algorithm proceeds in stages until a certain equilibrium is reached. This equilibrium is based upon the number of iterations specified in the algorithm. By increasing the number of iterations we can fine-tune the algorithm in order to more closely approach the global minimum, leading to more consistent results. As stated above, the number of iterations for these experiments was fixed at ten. To illustrate the effect of this parameter on the results, and also to reaffirm the effectiveness of our algorithms, the number of iterations was increased to 20 and 50. Best-match retrievals were then performed utilizing the exhaustive technique. Figure 8 summarizes the results, which confirm this hypothesis. Unfortunately, this results in a four-fold increase in CPU time in order to obtain the optimal results. This approach is therefore only recommended when questionable results are obtained.

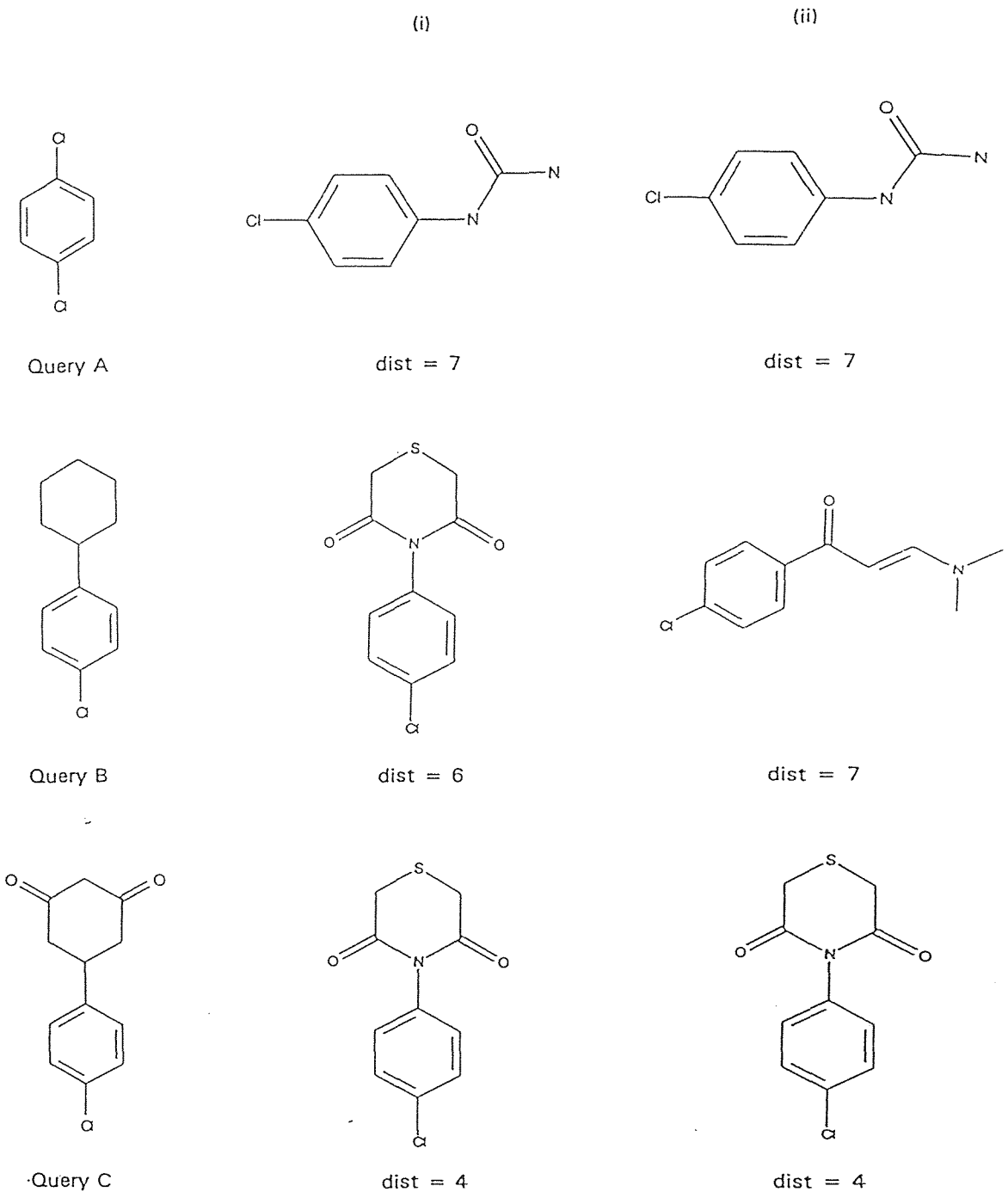
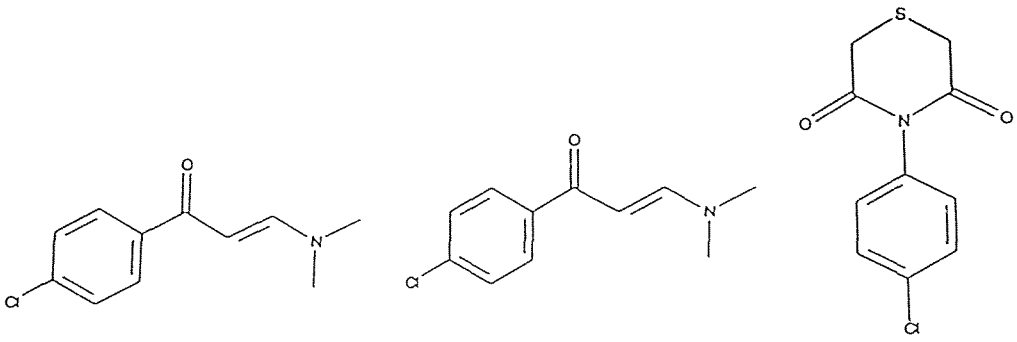


Figure 7. Results of best-match retrievals by heuristic method (i) and exhaustive method (ii).



ite = 10	ite = 20	ite = 50
dist = 7	dist = 7	dist = 6
CPU time(sec) = 1113	CPU time(sec) = 2088	CPU time(sec) = 4735

Figure 8. Summary of best-match retrieval results upon varying the number of iterations (*ite*) in the simulated annealing algorithm using the exhaustive method for query B.

3.2 Comparison to Existing Methods

To obtain a better understanding of the relative performance of our algorithms, we compared our results to those obtained using a commercially available 2-D/3-D database searching package, UNITY 2.3 [15]. The UNITY 2D screening feature utilizes molecular fingerprints in determine similarity among compounds as described in Chapter 1. Two classes of fingerprints are supported: fragment length based and fragment type based. Fragment length based fingerprints characterize a molecule based upon the length of fragments it contains, while fragment based fingerprints characterize a molecule based on the particular fragment types it contains. 2D screens are generated according to default 2D fingerprint definitions once the structure database is loaded. The default 2D fingerprint definitions provided with UNITY include the following characteristics:

- is 256 bits wide;
- includes fragments of length 2 through 7;
- uses both atom and bond type information;
- does not include hydrogens in generated fragments.

3D MOL files were required to build the UNITY database.

Similarity searches were performed using the same three structures for queries as used in the previous chapter. When performing a similarity search in UNITY, the user is able to define the maximum number of hits to be retrieved, and also the degree of similarity that is required for a structure to be considered a hit (*Min. Similarity*). The maximum number of hits to be retrieved was set at one, and *Min. Similarity* was set at 50, 40, 30 and 25%. Setting *Min. Similarity* greater than 50% resulted in zero structures retrieved.

Comparison of the results obtained using the different methods suggest that no significant correlation is observed. In order to better compare the two methods, similarity searches were performed on query C, only, with the same similarity constraints and the maximum number of structures retrieved set at 200. These results were then compared to a ranking of the structures in the database according to the edit distances as computed by the simulated annealing algorithm. Comparison between these results again suggest that no significant correlation was observed. Structures which the heuristic method classified as best-match candidates, *i.e.* least edit distances, were sometimes, but not always, included in the list of those retrieved by UNITY, likewise those which had the largest edit distances. It should be noted that, though the edit distances computed

accurately reflected topological differences in structures, they simply did not correspond to the same criteria used by UNITY. The structures retrieved by UNITY usually contained the basic substructures of the input query while allowing a greater degree of flexibility in their positions within the structure, *i.e.* the overall connectivity.

CHAPTER 4

TECHNIQUES FOR APPROXIMATE SUBSTRUCTURE SEARCHING

As stated in previous chapters, we define the approximate graph matching problem as follows: given two graphs, a pattern graph and a data graph, the distance (or similarity) between the two is defined as the minimum cost of all sequences of edit operations needed to transform one graph into the other. A variant of this problem is to allow the pattern graph to match only a part of the data graph, *i.e.*, subgraphs may be freely removed from the data graph. Again, we consider a pattern graph, P, and a data graph, D. We define K to be an *edge-complete* set which consists of edges and nodes of D, such that if a node u is present in K all the edges connected to u must also be in K. Let *Cut(D,K)* represent the data graph D with all nodes and edges in K removed. Figure 9 illustrates *Cut(D,K)*. Furthermore, let *Subgraphs(D)* to be the set of all possible edge-complete sets in D. In [8] it is shown that:

$$distwithcut(P,D) = \min_{K \in Subgraphs(D)}\{dist(P,Cut(D,K))\}.$$

In calculating *distwithcut(P,D)*, we apply the simulated annealing algorithm for finding the global minimum as outlined in Chapter 2. Once again, a separate file is created which contains these intra-molecular distances and the concepts of triangle inequality are applied in carrying out searches.

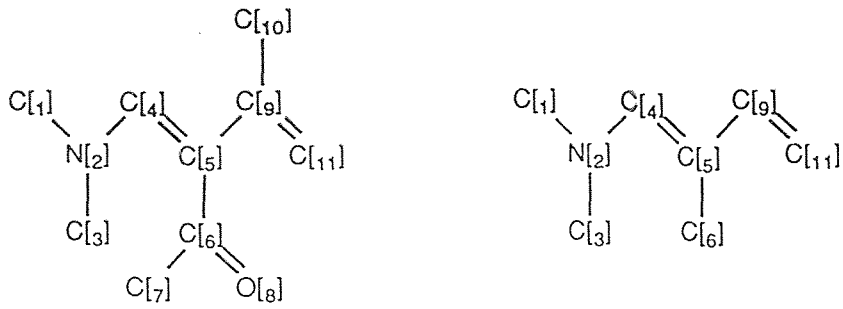


Figure 9. Diagram illustrating *Cut(D,K)*, where *K* contains the nodes C[7], O[8], C[10], and the edges connected to them, *i.e.*, {C[7],C[6]}, {O[8],C[6]} and {C[10],C[9]}.

CHAPTER 5

RESULTS AND DISCUSSION

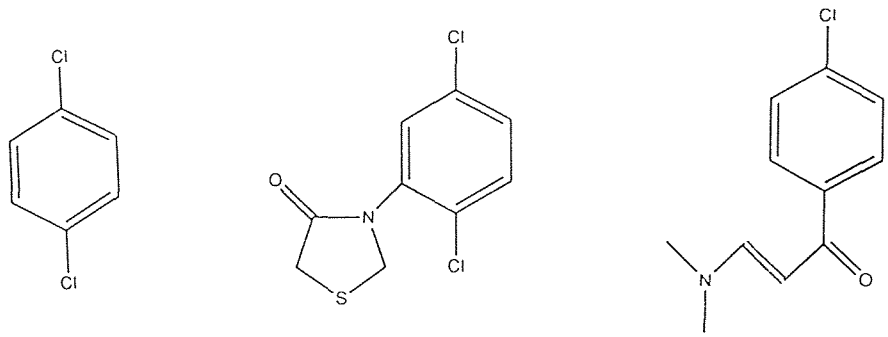
Best-match searches were performed based upon *distwithcut(P,D)* using the same database and the same three queries as in prior chapters. The results were then compared to those obtained from UNITY substructure and similarity searches.

The results obtained using the proposed *distwithcut(P,D)* technique displayed a strong correlation to the results obtained from the UNITY substructure search for each of the three queries. For queries A and C, those structures retrieved by UNITY were found by our proposed technique to have a distance value of zero, indicating that the query structure was a complete substructure of these compounds. Query B, however, showed some slight variations. The UNITY program retrieved a total of 13 hits when the substructure search was performed. Four of these 13 structures were found to have a distance value of zero using the proposed method. The remaining nine structures were found to have distance values ranging from 1 to 4. Upon examination of the molecular structure of those nine compounds, it was evident that they do, in fact, contain the entire query as part of their structure. The reason for such observed distance discrepancies lies in the fact, once again, that the simulated annealing algorithm is merely approximating the global minimum, it is not a precise calculation. As discussed in the previous chapter, it is recommended that the number of iterations be increased in the simulated annealing algorithm to more closely approximate the global minimum if such results are desired.

Here, however, we are focusing on approximate substructure retrievals, therefore, the algorithm provides sufficient results as implemented.

When our results are compared against those obtained from the UNITY similarity searches described in Chapter 4, we find that as the size and complexity of the query increases, the structures retrieved at various similarity percentages become more dispersed among the various distances. Furthermore, there is no significant correlation between structures retrieved by UNITY at 50% similarity and our proposed method with a specific distance or even a distance range.

Overall, if we limit ourselves to a similarity distance range of 0 to 2, we find that our technique retrieves approximately 97% of the structures retrieved by the UNITY substructure search. In addition, we also provide the user with additional structures which approximately match the input query and may have been previously overlooked due to slight variations which are not taken into consideration by the UNITY program (see Figure 10).



Query A (i) (ii)

Figure 10. Illustration of approximate substructure retrievals for query A. (i) *distance = 0*, retrieved by UNITY substructure search; (ii) *distance=1*, not retrieved by UNITY substructure or similarity search.

CHAPTER 6

CONCLUSIONS

In this thesis we have presented new techniques for performing best-match searches on databases consisting of 2D chemical compounds. We have demonstrated a method based upon graph theory which defines structural similarity based upon edit distances. To determine the minimum distance between two graphs we implemented an algorithm based upon simulated annealing and proceeded to apply triangle inequality as a means of performing best-match searches utilizing this distance information. In comparing our results with that of a linear exhaustive method we successfully proved the accuracy and efficiency of our technique as a function of CPU time and the number of comparisons required to perform a search.

By allowing subgraphs to be freely removed when matching a target pattern with a data graph we extended the best-match retrieval. Again, we implemented the concepts of simulated annealing and triangle inequality in determining structural distances and carrying out searches. The result is an approximate substructure search. The advantage of such a technique is that, while providing complementary information to that of a substructure search, it also returns additional structures similar to that of a given query which may be of significance and previously overlooked.

APPENDIX

This appendix contains all of the programs used in the implementation of the various techniques presented throughout this thesis.

```
/* ************************************************************** */
/* Program:   kp.h                                                */
/* Author:    Karen R. Pysniak                                    */
/*                                                                */
/* Type definition for true/false variables.                      */
/* ************************************************************** */


typedef enum kp_enum

{
   false = 0,
   true
} boolean;
```

```
/* ********************************************************************** */
/* Program:    sd2md.c                                                    */
/* Author:     Karen R. Pysniak                                           */
/*                                                                        */
/* This program converts 2D structure files from SD to MDL format for use in  */
/* CONCORD program.                                                       */
/* ********************************************************************** */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "kp.h"

#define MAX_ATOMS 50
#define MAX_BONDS 50
#define IN_PREFIX "sd_files/"
#define OUT_PREFIX "md_files/"

int main(void)
{
   char input_name[20], output_name[20];
   char input_file[30], output_file[30];
   int number_atoms, number_bonds;
   int number_props, number_cmpds;
   char header[25];
   char counts[35];
   char atom[50][65];
   char bond[50][25];
   char prop[10][12];
   char atom_temp[65];
   char bond_temp[25];
   char prop_temp[12];
   char line[35];
   char ch, response;
   int num_a, num_b, num_c, num_d, num_e, num_f, num_g, num_h;
   int i, j;
   boolean cont;
   FILE *file_in, *file_out;

   cont = true;
   number_cmpds = 0;

   printf("\n");
```

```
printf("Input File Name: ");
scanf("%s", input_name);
printf("\n");

strcpy(input_file, IN_PREFIX);
strcat(input_file, input_name);

file_in = fopen(input_file, "r");

if (file_in == NULL)
{
     printf("Can't open file %s", input_name);
     printf("\n\n");
     exit(0);
}

printf("\n");
printf("Output File Name: ");
scanf("%s", output_name);
printf("\n");

strcpy(output_file, OUT_PREFIX);
strcat(output_file, output_name);

file_out = fopen(output_file, "w+");

while (cont)
{
     fgets(line, sizeof(line), file_in);

     while (!feof(file_in))
     {
          if (number_cmpds == 100)
          {
               fclose(file_out);

               printf("%i compounds in output file %s", number_cmpds,
                    output_name);
               printf("\n");
               number_cmpds = 0;
               printf("\n");
               printf("Next Output File Name: ");
               scanf("%s", output_name);
               printf("\n");
```

```
        strcpy(output_file, OUT_PREFIX);
        strcat(output_file, output_name);

        file_out = fopen(output_file, "w+");
}

fgets(header, sizeof(header), file_in);
fgets(line, sizeof(line), file_in);
fgets(counts, sizeof(counts), file_in);
sscanf(counts, "%d %d %d %d %d %d %d %d %d %d %d",
        &number_atoms, &number_bonds, &num_a, &num_b, &num_c,
        &num_d, &num_e, &num_f, &num_g, &num_h, &number_props);

if ((number_atoms>MAX_ATOMS) || (number_bonds>MAX_BONDS))
{
        printf("\n");
        printf("*** Disregarding Compound: %i atoms/%i bonds ***",
                number_atoms, number_bonds);
        printf("\n");

        for (i=0; i<number_atoms; i++)
                fgets(atom_temp, sizeof(atom_temp), file_in);
        for (i=0; i<number_bonds; i++)
                fgets(bond_temp, sizeof(bond_temp), file_in);
        for (i=0; i<number_props; i++)
                fgets(prop_temp, sizeof(prop_temp), file_in);
        fgets(line, sizeof(line), file_in);

        while (line[1] != '$')
        {
                if (line[0] != '>')
                        fgets(line, sizeof(line), file_in);
                else
                {
                        ch = line[4];
                        fgets(line, sizeof(line), file_in);
                        if (ch == 'I')
                                printf("       ID: %s", line);
                        else if (ch == 'L')
                                printf("     List: %s", line);
                        fgets(line, sizeof(line), file_in);
                        fgets(line, sizeof(line), file_in);
                }
        }
```

```
}
else
{
        number_cmpds++;

        for (i=0; i<number_atoms; i++)
        {
                fgets(atom_temp, sizeof(atom_temp), file_in);
                for (j=0; j<65; j++)
                        atom[i][j] = atom_temp[j];
        }

        for (i=0; i<number_bonds; i++)
        {
                fgets(bond_temp, sizeof(bond_temp), file_in);
                for (j=0; j<25; j++)
                        bond[i][j] = bond_temp[j];
        }

        for (i=0; i<number_props; i++)
        {
                fgets(prop_temp, sizeof(prop_temp), file_in);
                for (j=0; j<12; j++)
                        prop[i][j] = prop_temp[j];
        }

        fgets(line, sizeof(line), file_in);

        while (line[1] != '$')
        {
                if (line[0] != '>')
                        fgets(line, sizeof(line), file_in);
                else
                {
                        ch = line[4];
                        fgets(line, sizeof(line), file_in);

                        if (ch == 'I')
                        {
                                ch = line[0];
                                j = 0;
                                while (ch != '\n')
                                {
                                        j++;
```

```
                                fputc(ch, file_out);
                                ch = line[j];
                        }
                        for (i=j-1; i<10; i++)
                                fputc(' ', file_out);
                }
                else if (ch == 'L')
                {
                        ch = line[0];
                        j = 0;
                        while (ch != '\n')
                        {
                                j++;
                                fputc(ch, file_out);
                                ch = line[j];
                        }
                        for (i=j-1; i<5; i++)
                        fputc(' ', file_out);
                        fputc ('\n', file_out);
                }

                fgets(line, sizeof(line), file_in);
                fgets(line, sizeof(line), file_in);
        }
    }
    fputs(header, file_out);
    fputc('\n', file_out);
    fputs(counts, file_out);
    for (i=0; i<number_atoms; i++)
        fputs(atom[i], file_out);
    for (i=0; i<number_bonds; i++)
        fputs(bond[i], file_out);
    for (i=0; i<number_props; i++)
        fputs(prop[i], file_out);
}

fputs(line, file_out);
fgets(line, sizeof(line), file_in);
}

fclose(file_in);

printf("\n");
printf("End of file reached for %s", input_name);
```

```
        fflush(stdin);
        printf("\n");
        printf("Would you like to continue with another input file?(y/n) ");
        scanf("%c", &response);

        if ((response == 'n') || (response == 'N'))
                cont = false;
        else
        {
                printf("\n");
                printf("Input File Name: ");
                scanf("%s", input_name);
                printf("\n");

                strcpy(input_file, IN_PREFIX);
                strcat(input_file, input_name);
                file_in = fopen(input_file, "r");

                if (file_in == NULL)
                {
                        printf("Can't open file %s", input_name);
                        printf("\n\n");
                        exit(0);
                }
        }
    }

    printf("\n");
    printf("%i compounds in output file %s", number_cmpds, output_name);
    printf("\n\n");

    fclose(file_out);

    return(0);
}
```

```
/* ********************************************************************** */
/* Program:    mdl2grph.c                                                 */
/* Author:     Karen R. Pysniak                                           */
/*                                                                        */
/* This program converts MDL format files to graph format files to be used by    */
/* triangle.c and long.c                                                  */
/* ********************************************************************** */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "kp.h"

#define IN_PREFIX "mdl_format/"
#define OUT_PREFIX "grph_format/"
#define IN_EXT ".mdl"
#define OUT_EXT ".grph"

int main(void)
{
   char input_name[20], output_name[20];
   char input_file[30], output_file[30];
   char matrix[50][51];
   int number_atoms, number_bonds, number_props;
   int number_cmpds;
   char str_temp[24];
   char id[20];
   char counts[100];
   char line[100];
   char node_id[50][2];
   char bond_type[1], response;
   int node_a, node_b;
   float num_a, num_b, num_c, num_d;
   int i, j;
   boolean cont;
   FILE *file_in, *file_out;

   cont = true;
   number_cmpds = 0;

   printf("\n");
   printf("Input File Name: ");
   scanf("%s", input_name);
```

```
printf("\n");

strcat(input_name, IN_EXT);
strcpy(input_file, IN_PREFIX);
strcat(input_file, input_name);

file_in = fopen(input_file, "r");

if (file_in == NULL)
{
     printf("Can't open file %s", input_name);
     printf("\n\n");
     exit(0);
}

printf("\n");
printf("Output File Name: ");
scanf("%s", output_name);
printf("\n");

strcat(output_name, OUT_EXT);
strcpy(output_file, OUT_PREFIX);
strcat(output_file, output_name);

file_out = fopen(output_file, "w+");

while (cont)
{
     fgets(line, sizeof(line), file_in);
     strncpy(id, line, 20);

     while (!feof(file_in))
     {
          if (number_cmpds == 100)
          {
               fclose(file_out);

               printf("\n%i compounds in output file %s", number_cmpds,
                    output_name);
               printf("\n");

               number_cmpds = 0;

               printf("\n");
```

```
        printf("Next Output File Name: ");
        scanf("%s", output_name);
        printf("\n");
        strcat(output_name, OUT_EXT);
        strcpy(output_file, OUT_PREFIX);
        strcat(output_file, output_name);

        file_out = fopen(output_file, "w+");
}

number_cmpds++;
fgets(line, sizeof(line), file_in);
fgets(line, sizeof(line), file_in);

fgets(counts, sizeof(counts), file_in);
sscanf(counts, "%d %d %f %f %f %f %d",
        &number_atoms, &number_bonds, &num_a, &num_b, &num_c,
        &num_d, &number_props);
fputs(id, file_out);
fputc('\n', file_out);
fprintf(file_out, "%d", number_atoms);
fputc('\n', file_out);

for (i=0; i<number_atoms; i++)
{
        fgets(line, 35, file_in);
        sscanf(line, "%f %f %f %s", &num_a, &num_b, &num_c,
                node_id[i]);
        fgets(line, sizeof(line), file_in);
}

for (i=0; i<number_atoms; i++)
{
        for (j=0; j<i; j++)
                matrix[i][j] = '-';
}

for (i=0; i<number_bonds; i++)
{
        fgets(line, sizeof(line), file_in);
        sscanf(line, "%d %d %c", &node_a, &node_b, bond_type);

        switch(*bond_type)
        {
```

```
                case '1':       {
                                        *bond_type = 'a';
                                        break;
                                }
                case '2':       {
                                        *bond_type = 'b';
                                        break;
                                }
                case '3':       {
                                        *bond_type = 'c';
                                        break;
                                }
                default:        printf("\nInvalid bond type.\n");
                                exit(1);
        }
        if (node_a < node_b)
                matrix[node_b-1][node_a-1] = *bond_type;
        else
                matrix[node_a-1][node_b-1] = *bond_type;
}

for (i=0; i<number_atoms; i++)
{
        for (j=0; j<i; j++)
        {
                fprintf(file_out, "%c", ' ');
                fprintf(file_out, "%c", matrix[i][j]);
        }
        if (node_id[i][1] == NULL)
        {
                node_id[i][1] = node_id[i][0];
                node_id[i][0] = ' ';
        }
        fprintf(file_out, "%c", node_id[i][0]);
        fprintf(file_out, "%c", node_id[i][1]);
        fprintf(file_out, "%c", '\n');
}

fgets(line, sizeof(line), file_in);
while (line[1] != '$')
        fgets(line, sizeof(line), file_in);
fputs(line, file_out);

fgets(line, sizeof(line), file_in);
```

```
            strncpy(id, line, 20);
        }

        fclose(file_in);

        printf("\n");
        printf("End of file reached for %s", input_name);
        fflush(stdin);
        printf("\n");
        printf("Would you like to continue with another input file?(y/n) ");
        scanf("%c", &response);

        if ((response == 'n') || (response == 'N'))
                cont = false;
        else
        {
                printf("\n");
                printf("Input File Name: ");
                scanf("%s", input_name);
                printf("\n");

                strcat(input_name, IN_EXT);
                strcpy(input_file, IN_PREFIX);
                strcat(input_file, input_name);

                file_in = fopen(input_file, "r");

                if (file_in == NULL)
                {
                        printf("Can't open file %s", input_name);
                        printf("\n\n");
                        exit(0);
                }
        }
}

fclose(file_out);

return(0);
```

```
/* ************************************************************** */
/* Program:    graph.c                                            */
/* Developed at NJIT                                              */
/*                                                                */
/* Modified by K. Pysniak                                         */
/*                                                                */
/* Implementation of simulated annealing algorithm using dist.    */
/* ************************************************************** */


int graph()
{
   int mn;
   int mnmap[MAX], map[MAX];
   int ct, ce;
   int i, j, k;
   double alpha;

   read_dist_mtx();
   input();
   ct=10;
   ce=10;
   alpha=0.5;
   prim(map,1);
   j=SA(ct,ce,alpha,map,cost);
   mn=j;
   for(i=0;i<Pn;i++)
        mnmap[i]=map[i];
   for(i=0;i<ITE;i++)
   {
        rnd_st(map);
        j=II(map,cost);
        j=opt_3(map,cost);
        if(j<mn)
        {
                mn=j;
                for(k=0;k<Pn;k++)
                        mnmap[k]=map[k];
        }
   }
   return mn();
```

```
input()
{
   int i, j;

   for(i=0;i<Pn;i++)
        for(j=0;j<=i;j++)
                if(i==j)
                        P[i][j]=ei(&Px[i][j+j]);
                else
                        P[j][i]=P[i][j]=(Px[i][j+j+1]=='-')?'\0':Px[i][j+j+1]-'a'+1;

   for(i=0;i<Dn;i++)
        for(j=0;j<=i;j++)
                if(i==j)
                        D[i][j]=ei(&Dx[i][j+j]);
                else
                        D[j][i]=D[i][j]=(Dx[i][j+j+1]=='-')?'\0':Dx[i][j+j+1]-'a'+1;
}


read_dist_mtx()
{
   int i, j, k;
   char a[100];
   FILE *in;

   in=fopen("dist.mtx","r");

   do
        fgets(a,100,in);
   while(a[0]=='\n');

   for(i=0;i<27;i++)
   {
        fscanf(in,"%s ",a);
                for(j=0;j<27;j++)
                {
                        k=fscanf(in,"%d ",&N[i][j]);
                        N_cut[i][j]=N[i][j];
                }
   }
```

```
    do
        fgets(a,100,in);
    while(a[0]=='\n');

    for(i=0;i<27;i++)
    {
        fscanf(in,"%s ",a);
        for(j=0;j<27;j++)
        {
                k=fscanf(in,"%d ",&E[i][j]);
                E_cut[i][j]=E[i][j];
        }
    }

    fclose(in);

    for(i=0;i<105;i++)
        for(j=0;j<105;j++)
                N_cut[i][j]=N[i][j]=(i!=j);

    for(i=0;i<105;i++)
        N_cut[i][0]=0;

    for(i=0;i<27;i++)
        E_cut[i][0]=0;
}


set_random()
{
    struct timeval *tp;
    struct timezone *tzp;

    tp=(struct timeval *)malloc(sizeof(struct timeval));
    tzp=(struct timezone *)malloc(sizeof(struct timezone));
    gettimeofday(tp,tzp);
    srandom((int)(tp->tv_usec%30000));
}
```

```
int cost(map)
int map[MAX];
{
   int i, j, k;
   char t[MAX][MAX];

   for(i=0;i<Dn;i++)
        for(j=i;j<Dn;j++)
              t[i][j]='\0';

   for(i=0;i<Pn;i++)
   {
        k=map[i];
              for(j=0;j<Pn;j++)
                    t[k][map[j]]=P[i][j];
   }

   for(k=i=0;i<Dn;i++)
   {
        k+=N[D[i][i]][t[i][i]];
        for(j=i+1;j<Dn;j++)
              k+=E[D[i][j]][t[i][j]];
   }

   return(k);
}


prim(map, way)
int map[MAX], way;
{
   int i,j,n,u,v,r,x,y,done,apn,adn,a,b,c,d,t,mn;
   int ap[MAX],ad[MAX],tem[MAX],pam[MAX],wt[MAX][MAX];

   for(i=0;i<Pn;i++)
        map[i]= -1;
   for(i=0;i<Dn;i++)
        pam[i]= -1;
   if(way!=0)
        goto mx1;
```

```
    for(done=0;done<2;done+=2)
    {
again1:
        i=random()%Pn;

        if(map[i]!= -1)
            goto again1;

again2:
        j=random()%Pn;

        if(map[j]!= -1 || i==j)
            goto again2;
        if(P[i][j]==0)
            goto again1;

        u=Opt[i];
        v=Opt[j];

        if(D[u][v]==0)
            goto again1;

        map[i]=u;
        pam[u]=i;
        map[j]=v;
        pam[v]=j;
    }
    goto start;

mx1:
    i=random()%Pn;
    j=random()%Dn;
    map[i]=j;
    pam[j]=i;
    done=1;

start:
    while(done<Pn)
    {
        for(i=0;i<Pn;i++)
            if(map[i]>=0 && map[i]!=MX)
                for(j=0;j<Pn;j++)
                    if(map[j]<0&&P[i][j]!=0)
                        map[j]=MX;
```

```
for(i=0;i<Dn;i++)
        if(pam[i]>=0 && pam[i]!=MX)
                for(j=0;j<Dn;j++)
                        if(pam[j]<0&&D[i][j]!=0)
                                pam[j]=MX;

for(apn=i=0;i<Pn;i++)
        if(map[i]==MX)
                ap[apn++]=i;

for(adn=i=0;i<Dn;i++)
        if(pam[i]==MX)
                ad[adn++]=i;

n=max(apn,adn);

for(i=0;i<=n;i++)
        for(j=0;j<=n;j++)
                wt[i][j]=0;

for(i=0;i<apn;i++)
        for(j=0;j<adn;j++)
        {
                u=ap[i];
                v=ad[j];
                r=N[P[u][u]][D[v][v]];
                for(x=0;x<Pn;x++)
                        if((y=map[x])>=0&&y!=MX)
                                r+=E[P[u][x]][D[v][y]];
                wt[i+1][j+1]=r;
        }

bi_graph(wt,n,tem);

for(i=0;i<apn;i++)
        if((j=tem[i+1]-1)<adn)
        {
                map[ap[i]]=ad[j];
                pam[ad[j]]=ap[i];
        }

done+=min(apn,adn);

for(i=0;i<Pn;i++)
```

```
                if(map[i]==MX)
                        map[i]= -1;

        for(i=0;i<Dn;i++)
                if(pam[i]==MX)
                        pam[i]= -1;
    }

    for(i=Pn,j=0;i<Dn;i++)
    {
        while(pam[j]>=0)
                j++;
        map[i]=j++;
    }
}


rnd_st(map)
int map[MAX];
{
    int i, j, t[MAX];

    for(i=0;i<Dn;i++)
        t[i]=0;
    for(i=0;i<Dn;i++)
    {
again:
        j=random()%Dn;
        if(t[j]==1)
                goto again;
        t[j]=1;
        map[i]=j;
    }
}


int II(map, fun)
int map[MAX], (*fun)();
{
    int i, j, k, n, l_min;

    l_min=fun(map);
new:
    for(i=0;i<Pn && l_min>0;i++)
```

```
        for(j=i+1;j<Dn;j++)
        {
                n=map[i];
                map[i]=map[j];
                map[j]=n;
                n=fun(map);

                if(n<l_min)
                {
                        l_min=n;
                        goto new;
                }

                n=map[i];
                map[i]=map[j];
                map[j]=n;
        }

   return(l_min);
}


int opt_3(map, fun)
int map[MAX], (*fun)();
{
   int i, j, k, n, l_min;

   l_min=fun(map);
new:
   for(i=0;i<Pn && l_min>0;i++)
        for(j=i+1;j<Dn;j++)
        {
                n=map[i];
                map[i]=map[j];
                map[j]=n;
                n=fun(map);

                if(n<l_min)
                {
                        l_min=n;
                        goto new;
                }

                n=map[i];
```

```
            map[i]=map[j];
            map[j]=n;
        }

    for(i=0;i<Pn;i++)
        for(j=i+1;j<Pn;j++)
            for(k=j+1;k<Dn;k++)
            {
                n=map[i];
                map[i]=map[j];
                map[j]=map[k];
                map[k]=n;
                n=fun(map);

                if(n<l_min)
                {
                    l_min=n;
                    goto new;
                }

                n=map[i];
                map[i]=map[j];
                map[j]=map[k];
                map[k]=n;
                n=fun(map);

                if(n<l_min)
                {
                    l_min=n;
                    goto new;
                }

                n=map[i];
                map[i]=map[j];
                map[j]=map[k];
                map[k]=n;
                n=fun(map);
            }

    return(l_min);
}
```

```
int SA(ct, ce, alpha, s, fun)
int ct, ce, s[MAX], (*fun)();
double alpha;
{
   int i, j, n;
   int loop, unxg, costs, costs2, costmins, delta_c, equ;
   double p, temp;
   char mins[MAX];

   equ=ce*(Pn+Dn);
   costs=fun(s);
   temp=ct*costs;
   costmins=costs;

   for(i=0;i<Pn;i++)
        mins[i]=s[i];

   while(!(temp<1.0 && unxg==4) && costmins>0)
   {
        for(loop=0;loop<equ && costmins>0;loop++)
        {
                i=random()%Pn;
                j=random()%Dn;
                n=s[i];
                s[i]=s[j];
                s[j]=n;

                costs2=fun(s);
                delta_c=costs2-costs;

                if(delta_c<0)
                {
                        costs=costs2;
                        unxg=0;
                }
                else
                {
                        p=exp((-delta_c)/temp);
                        n=random()%32768;
                        if(n<=p*32767)
                        {
                                costs=costs2;
                                unxg=0;
                        }
```

```
                else
                {
                        n=s[i];
                        s[i]=s[j];
                        s[j]=n;
                        unxg++;
                }
        }

        if(costs<costmins)
        {
                costmins=costs;
                for(i=0;i<Pn;i++)
                        mins[i]=s[i];
        }
    }

    temp*=alpha;

}   /*  End while loop.  */

  for(i=0;i<Pn;i++)
      s[i]=mins[i];

  return(costmins);
}


#define     EPS         0.01
int  mate_v[MAX], mate_u[MAX], exposed[MAX], label[MAX], nhbor[MAX];
int  A[2][MAX];
int  Q[MAX];
int  M;
double  slack[MAX], af[MAX], bt[MAX];


int bi_graph(C, n, MATES)
int  C[MAX][MAX], n, MATES[MAX];
{
  int ii, i, j, k, v, u;
  int M_cost;
  double tf;
```

```
M=n;

if(M==1)
{
    MATES[1]=1;
    return C[1][1];
}

for(i=1;i<=M;i++)
{
    mate_v[i]=mate_u[i]=0;
    af[i]=0.0;
    bt[i]=minc(i,C);
}

for(i=0;i<MAX;i++)
    label[i]= -1;

for(ii=1;ii<=M;ii++)
{
    for(i=0;i<MAX;i++)
        A[0][i]=A[1][i]=0;    /* empty A */

    for(v=1;v<=M;v++)
    {
        exposed[v]=0;
        slack[v]=MX;
    }

    for(i=1;i<=M;i++)
        for(j=1;j<=M;j++)
        {
            if(fabs(C[i][j]-af[i]-bt[j])>EPS)
                continue;

            if(mate_u[j]==0)
                exposed[i]=j;
            else
                Union_A(i,mate_u[j]);
        }

    for(i=1;i<=M;i++)
        Q[i]=0;        /* empty Q */
```

```
        for(i=1;i<=M;i++)
        {
                if(mate_v[i]!=0)
                        continue;

                if(exposed[i]!=0)
                {
                        augment(i);
                        goto endstage;
                }

                Q[i]=1;
                label[i]=0;

                for(k=1;k<=M;k++)
                        if((EPS<(tf=C[i][k]-af[i]-bt[k]))&&(tf<(slack[k]-EPS)))
                        {
                                slack[k]=tf;
                                nhbor[k]=i;
                        }
        }

search:
        while(!EmptyQ())
        {
                i=DQ();
                for(j=1;j<=M;j++)
                {
                        if((label[j]!= -1) || In_A(i,j)!=1)
                                continue;

                        label[j]=i;
                        Q[j]=1;

                        if(exposed[j]!=0)
                        {
                                augment(j);
                                goto endstage;
                        }

                        for(k=1;k<=M;k++)
                                if((EPS<(tf=C[j][k]-af[j]-bt[k]))&&(tf<(slack[k]-EPS)))
                                {
                                        slack[k]=tf;
```

```
                                        nhbor[k]=j;
                                }
                        }
                }

                if(modify())
                        goto endstage;
                goto search;

endstage:
                continue;

        }       /* most out for */

        for(i=1;i<=M;i++)
                MATES[i]=mate_v[i];
        for(M_cost=0,i=1;i<=M;i++)
                M_cost += C[i][mate_v[i]];
        return M_cost;
}


int minc(j, C)
int j, C[MAX][MAX];
{
        int i, l;

        for(l=MX,i=1;i<=M;i++)
                if(C[i][j]<l)
                        l=C[i][j];
        return l;
}


double minslack()
{
        int i;
        double temp;

        for(temp=MX,i=1;i<=M;i++)
                if((slack[i]<temp)&&(slack[i]>EPS))
                        temp=slack[i];
        return temp;
}
```

```
int EmptyQ()
{
   int i;

   for(i=1;i<=M;i++)
        if(Q[i]==1)
                return(0);
   return(1);
}


int DQ()
{
   int i;

   for(i=1;i<=M;i++)
        if(Q[i]==1)
        {
                Q[i]=0;
                return(i);
        }
}


int Union_A(x, y)
int x, y;
{
   int i;

   if(x>y)
   {
        i=x;
        x=y;
        y=i;
   }

   for(i=0;i<MAX;i++)
   {
        if(A[0][i]==0)
        {
                A[0][i]=x;
                A[1][i]=y;
                return(1);
        }
```

```
        if(A[0][i]==x && A[1][i]==y)
                return(0);
    }
}


int In_A(x, y)
int x, y;
{
    int i;

    if(x>y)
    {
        i=x;
        x=y;
        y=i;
    }

    for(i=0;i<MAX;i++)
    {
        if(A[0][i]==0)
                return(0);
        if(A[0][i]==x && A[1][i]==y)
                return(1);
    }
}


int modify()
{
    int i, j, u;
    double th1;

    th1 = minslack()/2.0;
    for(i=1;i<=M;i++)
        if(label[i]!= -1)
                af[i]=af[i]+th1;
        else
                af[i]=af[i]-th1;

    for(j=1;j<=M;j++)
        if(fabs(slack[j])<EPS)
                bt[j]=bt[j]-th1;
        else
```

```
                bt[j]+=th1;

    for(u=1;u<=M;u++)
    {
        if(slack[u]<EPS)
                continue;

        slack[u]=slack[u]-2.0*th1;
        if(fabs(slack[u])>=EPS)
                continue;

        if(mate_u[u]==0)
        {
                exposed[nhbor[u]]=u;
                augment(nhbor[u]);
                return 1;
        }
        else
        {
                label[mate_u[u]]=nhbor[u];
                Q[mate_u[u]]=1;
                Union_A(nhbor[u],mate_u[u]);
        }
    }

    return 0;
}


int augment(v)
int v;
{
    if(label[v]==0)
        mate_u[mate_v[v]=exposed[v]]=v;
    else
    {
        exposed[label[v]]=mate_v[v];
        mate_u[mate_v[v]=exposed[v]]=v;
        augment(label[v]);
    }
}
```

```
char ei(e)
char *e;
{
   if(e[0]==' ' && e[1]=='-')return((char)  0);
   if(e[0]==' ' && e[1]=='H')return((char)  1);
   if(e[0]=='H' && e[1]=='e')return((char)  2);
   if(e[0]=='L' && e[1]=='i')return((char)  3);
   if(e[0]=='B' && e[1]=='e')return((char)  4);
   if(e[0]==' ' && e[1]=='B')return((char)  5);
   if(e[0]==' ' && e[1]=='C')return((char)  6);
   if(e[0]==' ' && e[1]=='N')return((char)  7);
   if(e[0]==' ' && e[1]=='O')return((char)  8);
   if(e[0]==' ' && e[1]=='F')return((char)  9);
   if(e[0]=='N' && e[1]=='e')return((char) 10);
   if(e[0]=='N' && e[1]=='a')return((char) 11);
   if(e[0]=='M' && e[1]=='g')return((char) 12);
   if(e[0]=='A' && e[1]=='l')return((char) 13);
   if(e[0]=='S' && e[1]=='i')return((char) 14);
   if(e[0]==' ' && e[1]=='P')return((char) 15);
   if(e[0]==' ' && e[1]=='S')return((char) 16);
   if(e[0]=='C' && e[1]=='l')return((char) 17);
   if(e[0]=='A' && e[1]=='r')return((char) 18);
   if(e[0]==' ' && e[1]=='K')return((char) 19);
   if(e[0]=='C' && e[1]=='a')return((char) 20);
   if(e[0]=='S' && e[1]=='c')return((char) 21);
   if(e[0]=='T' && e[1]=='i')return((char) 22);
   if(e[0]==' ' && e[1]=='V')return((char) 23);
   if(e[0]=='C' && e[1]=='r')return((char) 24);
   if(e[0]=='M' && e[1]=='n')return((char) 25);
   if(e[0]=='F' && e[1]=='e')return((char) 26);
   if(e[0]=='C' && e[1]=='o')return((char) 27);
   if(e[0]=='N' && e[1]=='i')return((char) 28);
   if(e[0]=='C' && e[1]=='u')return((char) 29);
   if(e[0]=='Z' && e[1]=='n')return((char) 30);
   if(e[0]=='G' && e[1]=='a')return((char) 31);
   if(e[0]=='G' && e[1]=='e')return((char) 32);
   if(e[0]=='A' && e[1]=='s')return((char) 33);
   if(e[0]=='S' && e[1]=='e')return((char) 34);
   if(e[0]=='B' && e[1]=='r')return((char) 35);
   if(e[0]=='K' && e[1]=='r')return((char) 36);
   if(e[0]=='R' && e[1]=='b')return((char) 37);
   if(e[0]=='S' && e[1]=='r')return((char) 38);
   if(e[0]==' ' && e[1]=='Y')return((char) 39);
   if(e[0]=='Z' && e[1]=='r')return((char) 40);
```

```
if(e[0]=='N' && e[1]=='b')return((char) 41);
if(e[0]=='M' && e[1]=='o')return((char) 42);
if(e[0]=='T' && e[1]=='c')return((char) 43);
if(e[0]=='R' && e[1]=='u')return((char) 44);
if(e[0]=='R' && e[1]=='h')return((char) 45);
if(e[0]=='P' && e[1]=='d')return((char) 46);
if(e[0]=='A' && e[1]=='g')return((char) 47);
if(e[0]=='C' && e[1]=='d')return((char) 48);
if(e[0]=='I' && e[1]=='n')return((char) 49);
if(e[0]=='S' && e[1]=='n')return((char) 50);
if(e[0]=='S' && e[1]=='b')return((char) 51);
if(e[0]=='T' && e[1]=='e')return((char) 52);
if(e[0]=='I' && e[1]=='I')return((char) 53);
if(e[0]=='X' && e[1]=='e')return((char) 54);
if(e[0]=='C' && e[1]=='s')return((char) 55);
if(e[0]=='B' && e[1]=='a')return((char) 56);
if(e[0]=='L' && e[1]=='a')return((char) 57);
if(e[0]=='C' && e[1]=='e')return((char) 58);
if(e[0]=='P' && e[1]=='r')return((char) 59);
if(e[0]=='N' && e[1]=='d')return((char) 60);
if(e[0]=='P' && e[1]=='m')return((char) 61);
if(e[0]=='S' && e[1]=='m')return((char) 62);
if(e[0]=='E' && e[1]=='u')return((char) 63);
if(e[0]=='G' && e[1]=='d')return((char) 64);
if(e[0]=='T' && e[1]=='b')return((char) 65);
if(e[0]=='D' && e[1]=='y')return((char) 66);
if(e[0]=='H' && e[1]=='o')return((char) 67);
if(e[0]=='E' && e[1]=='r')return((char) 68);
if(e[0]=='T' && e[1]=='m')return((char) 69);
if(e[0]=='Y' && e[1]=='b')return((char) 70);
if(e[0]=='L' && e[1]=='u')return((char) 71);
if(e[0]=='H' && e[1]=='f')return((char) 72);
if(e[0]=='T' && e[1]=='a')return((char) 73);
if(e[0]==' ' && e[1]=='W')return((char) 74);
if(e[0]=='R' && e[1]=='e')return((char) 75);
if(e[0]=='O' && e[1]=='s')return((char) 76);
if(e[0]=='I' && e[1]=='r')return((char) 77);
if(e[0]=='P' && e[1]=='t')return((char) 78);
if(e[0]=='A' && e[1]=='u')return((char) 79);
if(e[0]=='H' && e[1]=='g')return((char) 80);
if(e[0]=='T' && e[1]=='l')return((char) 81);
if(e[0]=='P' && e[1]=='b')return((char) 82);
if(e[0]=='B' && e[1]=='i')return((char) 83);
if(e[0]=='P' && e[1]=='o')return((char) 84);
```

```
    if(e[0]=='A' && e[1]=='t')return((char) 85);
    if(e[0]=='R' && e[1]=='n')return((char) 86);
    if(e[0]=='F' && e[1]=='r')return((char) 87);
    if(e[0]=='R' && e[1]=='a')return((char) 88);
    if(e[0]=='A' && e[1]=='c')return((char) 89);
    if(e[0]=='T' && e[1]=='h')return((char) 90);
    if(e[0]=='P' && e[1]=='a')return((char) 91);
    if(e[0]==' ' && e[1]=='U')return((char) 92);
    if(e[0]=='N' && e[1]=='p')return((char) 93);
    if(e[0]=='P' && e[1]=='u')return((char) 94);
    if(e[0]=='A' && e[1]=='m')return((char) 95);
    if(e[0]=='C' && e[1]=='m')return((char) 96);
    if(e[0]=='B' && e[1]=='k')return((char) 97);
    if(e[0]=='C' && e[1]=='f')return((char) 98);
    if(e[0]=='E' && e[1]=='s')return((char) 99);
    if(e[0]=='F' && e[1]=='m')return((char)100);
    if(e[0]=='M' && e[1]=='d')return((char)101);
    if(e[0]=='N' && e[1]=='o')return((char)102);
    if(e[0]=='L' && e[1]=='r')return((char)103);
    if(e[0]=='K' && e[1]=='h')return((char)104);
    if(e[0]==' ' && e[1]=='I')return((char)105);

    printf("\nUnknown element %c%c, program stop\n",e[0],e[1]); exit(1);
}
```

```
/* ************************************************************** */
/* Program:   graph.c                                             */
/* Developed at NJIT                                              */
/*                                                                */
/* Modified by K. Pysniak                                         */
/*                                                                */
/* Implementation of simulated annealing algorithm using distwithcut.  */
/* ************************************************************** */


int graph()
{
   int mn;
   int mnmap[MAX], map[MAX];
   int ct, ce;
   int i, j, k;
   double alpha;

   read_dist_mtx();
   input();
   mn = 99999;

   for(i=0;i<Pn;i++)
        mnmap[i]=map[i];

   for(i=0;i<ITE;i++)
   {
        rnd_st(map);
        j=II(map,cost_cut);
        j=opt_3(map,cost_cut);

        if(j<mn)
        {
                mn=j;
                for(k=0;k<Pn;k++)
                mnmap[k]=map[k];
        }
   }
   return mn;
}
```

```
input()
{
   int i, j;

   for(i=0;i<Pn;i++)
        for(j=0;j<=i;j++)
                if(i==j)
                        P[i][j]=ei(&Px[i][j+j]);
                else
                        P[j][i]=P[i][j]=(Px[i][j+j+1]=='-')?'\0':Px[i][j+j+1]-'a'+1;

   for(i=0;i<Dn;i++)
        for(j=0;j<=i;j++)
                if(i==j)
                        D[i][j]=ei(&Dx[i][j+j]);
                else
                        D[j][i]=D[i][j]=(Dx[i][j+j+1]=='-')?'\0':Dx[i][j+j+1]-'a'+1;
}


read_dist_mtx()
{
   int i, j, k;
   char a[100];
   FILE *in;

   in=fopen("dist.mtx","r");

   do
        fgets(a,100,in);
   while(a[0]=='\n');

   for(i=0;i<27;i++)
   {
        fscanf(in,"%s ",a);
        for(j=0;j<27;j++)
        {
                k=fscanf(in,"%d ",&N[i][j]);
                N_cut[i][j]=N[i][j];
        }
   }
```

```
    do
        fgets(a,100,in);
    while(a[0]=='\n');

    for(i=0;i<27;i++)
    {
        fscanf(in,"%s ",a);
        for(j=0;j<27;j++)
        {
            k=fscanf(in,"%d ",&E[i][j]);
            E_cut[i][j]=E[i][j];
        }
    }
    fclose(in);

    for(i=0;i<105;i++)
        for(j=0;j<105;j++)
            N_cut[i][j]=N[i][j]=(i!=j);

    for(i=0;i<105;i++)
        N_cut[i][0]=0;

    for(i=0;i<27;i++)
        E_cut[i][0]=0;
}


int cost_cut(map)
int map[MAX];
{
    int  i, j, u, v, w, x, y;
    char t[MAX][MAX];

    for(i=0;i<Dn;i++)
        for(j=i;j<Dn;j++)
            t[i][j]='\0';

    for(i=0;i<Pn;i++)
    {
        x=map[i];
        t[x][x]=P[i][i];

        for(j=i+1;j<Pn;j++)
        {
```

```
                y=map[j];
                u=P[i][j];
                v=D[x][y];

                if(E_cut[v][u] > E_cut[0][u])
                        t[x][y]=t[y][x]='\0';
                else
                        t[x][y]=t[y][x]=u;
        }
    }

again:
    for(i=0;i<Dn;i++)
    {
        x=t[i][i];
        y=D[i][i];
        if((w=N_cut[y][x]-N_cut[0][x])<=0)
                continue;

        for(j=0;j<Dn;j++)
        {
                u=t[i][j];
                if(u==0)
                        continue;
                v=D[i][j];
                w+=E_cut[v][u]-E_cut[0][u];
        }

        w=w-E_cut[y][x]+E_cut[0][x];
        if(w<=0)
                continue;
        for(j=0;j<Dn;j++)
                t[i][j]=t[j][i]='\0';
        goto again;
    }

    for(x=i=0;i<Dn;i++)
    {
        x+=N_cut[D[i][i]][t[i][i]];
        for(j=i+1;j<Dn;j++)
                x+=E_cut[D[i][j]][t[i][j]];
    }
    return(x);
}
```

```
rnd_st(map)
int map[MAX];
{
   int i,j,t[MAX];

   for(i=0;i<Dn;i++)
        t[i]=0;

   for(i=0;i<Dn;i++)
   {
again:
        j=rand()%Dn;

        if(t[j]==1)
                goto again;

        t[j]=1;
        map[i]=j;
   }
}


int exhaustive(map, fun)
int map[MAX], (*fun)();
{
   int i,j,n,k,n_k,dist,c[MAX];
   int m,pm,dm,x[MAX],p[MAX],d[MAX],tmp[MAX];

   n=Dn;
   k=Pn;
   dist=MX;
   c[0]= -1;
   j=1;
   n_k=n-k;
   for(i=1;i<=k;i++)
        c[i]=i;

   while(j!=0)
   {
        for(i=1;i<=k;i++)
        {
                x[i]=p[i]=i;
                d[i]= -1;
        }
```

```
        d[1]=0;
        x[0]=x[k+1]=m=k+1;

        while(m!=1)
        {
                for(i=1;i<=k;i++)
                        tmp[i-1]=c[x[i]]-1;

                i=fun(tmp);
                if(i<dist)
                {
                        dist=i;
                        for(i=0;i<k;i++)
                                map[i]=tmp[i];
                }

                m=k;
                while(x[p[m]+d[m]]>m)
                {
                        d[m]= -d[m];
                        m--;
                }

                pm=p[m];
                dm=d[m];
                i=x[pm];
                x[pm]=x[pm+dm];
                x[pm+dm]=i;
                i=p[x[pm]];
                p[x[pm]]=pm;
                p[m]=i;
        }

        j=k;

        while(c[j]==n_k+j)
                j--;

        c[j]++;
        for(i=j+1;i<=k;i++)
                c[i]=c[i-1]+1;
    }
    return(dist);
}
```

```
int II(map,fun)
int map[MAX], (*fun)();
{
   int i,j,k,n,l_min;

   l_min=fun(map);

new:
   for(i=0;i<Pn && l_min>0;i++)
        for(j=i+1;j<Dn;j++)
        {
                n=map[i];
                map[i]=map[j];
                map[j]=n;
                n=fun(map);

                if(n<l_min)
                {
                        l_min=n;
                        goto new;
                }

                n=map[i];
                map[i]=map[j];
                map[j]=n;
        }

   return(l_min);
}


int opt_3(map,fun)
int map[MAX], (*fun)();
{
   int i,j,k,n,l_min;

   l_min=fun(map);

new:
   for(i=0;i<Pn && l_min>0;i++)
        for(j=i+1;j<Dn;j++)
        {
                n=map[i];
                map[i]=map[j];
```

```
                map[j]=n;
                n=fun(map);

                if(n<l_min)
                {
                        l_min=n;
                        goto new;
                }

                n=map[i];
                map[i]=map[j];
                map[j]=n;
        }

for(i=0;i<Pn;i++)
    for(j=i+1;j<Pn;j++)
                for(k=j+1;k<Dn;k++)
                {
                        n=map[i];
                        map[i]=map[j];
                        map[j]=map[k];
                        map[k]=n;
                        n=fun(map);

                        if(n<l_min)
                        {
                                l_min=n;
                                goto new;
                        }

                        n=map[i];
                        map[i]=map[j];
                        map[j]=map[k];
                        map[k]=n;
                        n=fun(map);

                        if(n<l_min)
                        {
                                l_min=n;
                                goto new;
                        }

                        n=map[i];
                        map[i]=map[j];
```

```
                        map[j]=map[k];
                        map[k]=n;
                        n=fun(map);
                }

    return(l_min);
}


char ei(e) char *e;
{
    if(e[0]==' ' && e[1]=='-')return((char)  0);
    if(e[0]==' ' && e[1]=='H')return((char)  1);
    if(e[0]=='H' && e[1]=='e')return((char)  2);
    if(e[0]=='L' && e[1]=='i')return((char)  3);
    if(e[0]=='B' && e[1]=='e')return((char)  4);
    if(e[0]==' ' && e[1]=='B')return((char)  5);
    if(e[0]==' ' && e[1]=='C')return((char)  6);
    if(e[0]==' ' && e[1]=='N')return((char)  7);
    if(e[0]==' ' && e[1]=='O')return((char)  8);
    if(e[0]==' ' && e[1]=='F')return((char)  9);
    if(e[0]=='N' && e[1]=='e')return((char) 10);
    if(e[0]=='N' && e[1]=='a')return((char) 11);
    if(e[0]=='M' && e[1]=='g')return((char) 12);
    if(e[0]=='A' && e[1]=='l')return((char) 13);
    if(e[0]=='S' && e[1]=='i')return((char) 14);
    if(e[0]==' ' && e[1]=='P')return((char) 15);
    if(e[0]==' ' && e[1]=='S')return((char) 16);
    if(e[0]=='C' && e[1]=='l')return((char) 17);
    if(e[0]=='A' && e[1]=='r')return((char) 18);
    if(e[0]==' ' && e[1]=='K')return((char) 19);
    if(e[0]=='C' && e[1]=='a')return((char) 20);
    if(e[0]=='S' && e[1]=='c')return((char) 21);
    if(e[0]=='T' && e[1]=='i')return((char) 22);
    if(e[0]==' ' && e[1]=='V')return((char) 23);
    if(e[0]=='C' && e[1]=='r')return((char) 24);
    if(e[0]=='M' && e[1]=='n')return((char) 25);
    if(e[0]=='F' && e[1]=='e')return((char) 26);
    if(e[0]=='C' && e[1]=='o')return((char) 27);
    if(e[0]=='N' && e[1]=='i')return((char) 28);
    if(e[0]=='C' && e[1]=='u')return((char) 29);
    if(e[0]=='Z' && e[1]=='n')return((char) 30);
    if(e[0]=='G' && e[1]=='a')return((char) 31);
    if(e[0]=='G' && e[1]=='e')return((char) 32);
```

```
if(e[0]=='A' && e[1]=='s')return((char) 33);
if(e[0]=='S' && e[1]=='e')return((char) 34);
if(e[0]=='B' && e[1]=='r')return((char) 35);
if(e[0]=='K' && e[1]=='r')return((char) 36);
if(e[0]=='R' && e[1]=='b')return((char) 37);
if(e[0]=='S' && e[1]=='r')return((char) 38);
if(e[0]==' ' && e[1]=='Y')return((char) 39);
if(e[0]=='Z' && e[1]=='r')return((char) 40);
if(e[0]=='N' && e[1]=='b')return((char) 41);
if(e[0]=='M' && e[1]=='o')return((char) 42);
if(e[0]=='T' && e[1]=='c')return((char) 43);
if(e[0]=='R' && e[1]=='u')return((char) 44);
if(e[0]=='R' && e[1]=='h')return((char) 45);
if(e[0]=='P' && e[1]=='d')return((char) 46);
if(e[0]=='A' && e[1]=='g')return((char) 47);
if(e[0]=='C' && e[1]=='d')return((char) 48);
if(e[0]=='I' && e[1]=='n')return((char) 49);
if(e[0]=='S' && e[1]=='n')return((char) 50);
if(e[0]=='S' && e[1]=='b')return((char) 51);
if(e[0]=='T' && e[1]=='e')return((char) 52);
if(e[0]=='I' && e[1]=='I')return((char) 53);
if(e[0]=='X' && e[1]=='e')return((char) 54);
if(e[0]=='C' && e[1]=='s')return((char) 55);
if(e[0]=='B' && e[1]=='a')return((char) 56);
if(e[0]=='L' && e[1]=='a')return((char) 57);
if(e[0]=='C' && e[1]=='e')return((char) 58);
if(e[0]=='P' && e[1]=='r')return((char) 59);
if(e[0]=='N' && e[1]=='d')return((char) 60);
if(e[0]=='P' && e[1]=='m')return((char) 61);
if(e[0]=='S' && e[1]=='m')return((char) 62);
if(e[0]=='E' && e[1]=='u')return((char) 63);
if(e[0]=='G' && e[1]=='d')return((char) 64);
if(e[0]=='T' && e[1]=='b')return((char) 65);
if(e[0]=='D' && e[1]=='y')return((char) 66);
if(e[0]=='H' && e[1]=='o')return((char) 67);
if(e[0]=='E' && e[1]=='r')return((char) 68);
if(e[0]=='T' && e[1]=='m')return((char) 69);
if(e[0]=='Y' && e[1]=='b')return((char) 70);
if(e[0]=='L' && e[1]=='u')return((char) 71);
if(e[0]=='H' && e[1]=='f')return((char) 72);
if(e[0]=='T' && e[1]=='a')return((char) 73);
if(e[0]==' ' && e[1]=='W')return((char) 74);
if(e[0]=='R' && e[1]=='e')return((char) 75);
if(e[0]=='O' && e[1]=='s')return((char) 76);
```

```
    if(e[0]=='I' && e[1]=='r')return((char) 77);
    if(e[0]=='P' && e[1]=='t')return((char) 78);
    if(e[0]=='A' && e[1]=='u')return((char) 79);
    if(e[0]=='H' && e[1]=='g')return((char) 80);
    if(e[0]=='T' && e[1]=='l')return((char) 81);
    if(e[0]=='P' && e[1]=='b')return((char) 82);
    if(e[0]=='B' && e[1]=='i')return((char) 83);
    if(e[0]=='P' && e[1]=='o')return((char) 84);
    if(e[0]=='A' && e[1]=='t')return((char) 85);
    if(e[0]=='R' && e[1]=='n')return((char) 86);
    if(e[0]=='F' && e[1]=='r')return((char) 87);
    if(e[0]=='R' && e[1]=='a')return((char) 88);
    if(e[0]=='A' && e[1]=='c')return((char) 89);
    if(e[0]=='T' && e[1]=='h')return((char) 90);
    if(e[0]=='P' && e[1]=='a')return((char) 91);
    if(e[0]==' ' && e[1]=='U')return((char) 92);
    if(e[0]=='N' && e[1]=='p')return((char) 93);
    if(e[0]=='P' && e[1]=='u')return((char) 94);
    if(e[0]=='A' && e[1]=='m')return((char) 95);
    if(e[0]=='C' && e[1]=='m')return((char) 96);
    if(e[0]=='B' && e[1]=='k')return((char) 97);
    if(e[0]=='C' && e[1]=='f')return((char) 98);
    if(e[0]=='E' && e[1]=='s')return((char) 99);
    if(e[0]=='F' && e[1]=='m')return((char)100);
    if(e[0]=='M' && e[1]=='d')return((char)101);
    if(e[0]=='N' && e[1]=='o')return((char)102);
    if(e[0]=='L' && e[1]=='r')return((char)103);
    if(e[0]=='K' && e[1]=='h')return((char)104);
    if(e[0]==' ' && e[1]=='I')return((char)105);

    printf("\nUnknown element %c%c, program stop\n",e[0],e[1]); exit(1);
}
```

```
/* ************************************************************************/
/* Program:    matrix.c                                                   */
/* Author:     Karen R. Pysniak                                           */
/*                                                                        */
/* Build matrix file of intra-molecular distances for structures in database using    */
/* graph.c.                                                               */
/* ************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include "kp.h"

#define ITE    10
#define MAX          100
#define MX    9999
#define min(a,b)       (((a)<(b))?(a):(b))
#define max(a,b)       (((a)>(b))?(a):(b))

char P[MAX][MAX], D[MAX][MAX];
char Px[MAX][MAX], Dx[MAX][MAX];
int Pn, Dn;
int N[105][105], E[27][27], N_cut[105][105], E_cut[27][27];
int Opt[MAX];
int graph();
char ei();
int cost();       /* for caluculating dist; becomes cost_cut() for calculating distwithcut */

main(argc, argv)
int argc; char *argv[];
{
   char input_file[15], output_file[15];
   char str[100];
   char id[MAX][20];
   char table[MAX][MAX][MAX];
   int number_atoms[MAX];
   int matrix[MAX][MAX];
   int number_cmpds;
   int i, j, k, l;
   boolean cont;
   FILE *file_in, *file_out;
```

```
if(argc!=3)
{
     printf("\nusage: %s in_file out_file\n", argv[0]);
     printf("\n");
     exit(0);
}

strcpy(input_file, argv[1]);
strcpy(output_file, argv[2]);

file_in = fopen(input_file, "r");

if (file_in == NULL)
{
     printf("\nCan't open file %s\n", input_file);
     printf("\n");
     exit(0);
}

file_out = fopen(output_file, "w+");
i = 0;
cont = true;
fgets(str, sizeof(str), file_in);

while (!feof(file_in))
{
     strncpy(id[i], str, sizeof(str));
     fprintf(file_out, "%s", id[i]);
     fgets(str, sizeof(str), file_in);
     sscanf(str, "%d", &number_atoms[i]);

     for (j=0; j<number_atoms[i]; j++)
          fgets(table[i][j], sizeof(table[i][j]), file_in);

     fgets(str, sizeof(str), file_in);
     fgets(str, sizeof(str), file_in);
     i++;
}

fputc('\n', file_out);
number_cmpds = i;

fclose(file_in);
```

```
    for (i=0; i<number_cmpds; i++)
        for(j=0; j<number_cmpds; j++)
            matrix[i][j] = -99;

    for (i=0; i<number_cmpds; i++)
    {
        for (j=i+1; j<number_cmpds; j++)
        {
            if (number_atoms[i] <= number_atoms[j])
            {
                Pn = number_atoms[i];
                Dn = number_atoms[j];
                for (k=0; k<number_atoms[i]; k++)
                    strcpy(Px[k], table[i][k]);
                for (k=0; k<number_atoms[j]; k++)
                    strcpy(Dx[k], table[j][k]);
            }
            else
            {
                Pn = number_atoms[j];
                Dn = number_atoms[i];
                for (k=0; k<number_atoms[i]; k++)
                    strcpy(Dx[k], table[i][k]);
                for (k=0; k<number_atoms[j]; k++)
                    strcpy(Px[k], table[j][k]);
            }

            matrix[i][j] = graph();

            fprintf(file_out, "%3d", matrix[i][j]);
                fputc(' ', file_out);
        }

        fputc('\n', file_out);
    }

    close(file_out);
    return(0);
}
```

```
/* ************************************************************************/
/* Program:    triangle.c                                                 */
/* Author:     Karen R. Pysniak                                           */
/*                                                                        */
/* Triangle inequality method for searches implementing simulated annealing */
/* algorithm (graph.c).                                                   */
/* ************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include "kp.h"

#define ITE     10
#define MAX     100
#define MX      9999
#define min(a,b)        (((a)<(b))?(a):(b))
#define max(a,b)        (((a)>(b))?(a):(b))

char P[MAX][MAX], D[MAX][MAX];
char Px[MAX][MAX], Dx[MAX][MAX];
int Pn, Dn;
int N[105][105], E[27][27], N_cut[105][105], E_cut[27][27];
int Opt[MAX];
int graph();
char ei();
int cost();     /* for caluculating dist; becomes cost_cut() for calculating distwithcut */

main()
{
   int dist[MAX][MAX];
   int best_a, dist_a;
   int gr_best, dist_best;
   int gr_guess, dist_guess;
   int atoms_in_q;
   int atoms_a;
   int number_atoms[MAX];
   int pass, number_cand;
   int cnt_grph;
   int i, j, k;
   char input_db[15], input_mtrx[15];
```

```
char id_a[20], id_best[15];
char id[MAX][20];
char q_table[MAX][MAX];
char table_a[MAX][MAX];
char table[MAX][MAX][MAX];
char str[25], line[100];
char ch[3];
boolean ELIM[MAX];
boolean cont_1, cont_2;
FILE *file_in_q, *file_in_db, *file_in_mtrx;
FILE *file_out;

file_in_q = fopen("query", "r");

fgets(line, sizeof(line), file_in_q);
sscanf(line, "%d", &atoms_in_q);

for (i=0; i<atoms_in_q; i++)
     fgets(q_table[i], sizeof(q_table[i]), file_in_q);

fclose(file_in_q);

file_out = fopen("tri.out", "w+");

strcpy(input_db, "grph_db_a");
strcpy(input_mtrx, "mtrx_a");

cnt_grph = 0;

for (k=0; k<2; k++)
{
     file_in_db = fopen(input_db, "r");

     for (i=0; i<MAX; i++)
     {
          fgets(str, sizeof(str), file_in_db);
          strncpy(id[i], str, sizeof(str));

          fgets(line, sizeof(line), file_in_db);
          sscanf(line, "%d", &number_atoms[i]);

          for (j=0; j<number_atoms[i]; j++)
               fgets(table[i][j], sizeof(table[i][j]), file_in_db);
```

```
        fgets(str, sizeof(str), file_in_db);
}

fclose(file_in_db);

file_in_mtrx = fopen(input_mtrx, "r");

for (i=0; i<MAX; i++)
        fgets(str, sizeof(str), file_in_mtrx);

fgetc(file_in_mtrx);

for (i=0; i<MAX; i++)
        for (j=0; j<MAX; j++)
                if (i!=j)
                        if (i<j)
                        {
                                fscanf(file_in_mtrx, "%3s", ch);
                                dist[i][j] = atoi(ch);
                                fgetc(file_in_mtrx);
                        }
                        else
                                dist[i][j] = dist[j][i];

        fclose(file_in_mtrx);

for (i=0; i<MAX; i++)
        ELIM[i] = false;

gr_best = 0;

if (number_atoms[gr_best] <= atoms_in_q)
{
        Pn = number_atoms[gr_best];
        Dn = atoms_in_q;
        for (i=0; i<number_atoms[gr_best]; i++)
                strcpy(Px[i], table[gr_best][i]);
        for (i=0; i<atoms_in_q; i++)
                strcpy(Dx[i], q_table[i]);
}
else
{
        Pn = atoms_in_q;
        Dn = number_atoms[gr_best];
```

```
        for (i=0; i<atoms_in_q; i++)
                strcpy(Px[i], q_table[i]);
        for (i=0; i<number_atoms[gr_best]; i++)
                strcpy(Dx[i], table[gr_best][i]);
}

dist_best = graph();
cnt_grph++;

gr_guess = gr_best + 1;

if (number_atoms[gr_guess] <= atoms_in_q)
{
        Pn = number_atoms[gr_guess];
        Dn = atoms_in_q;
        for (i=0; i<number_atoms[gr_guess]; i++)
                strcpy(Px[i], table[gr_guess][i]);
        for (i=0; i<atoms_in_q; i++)
                strcpy(Dx[i], q_table[i]);
}
else
{
        Pn = atoms_in_q;
        Dn = number_atoms[gr_guess];
        for (i=0; i<atoms_in_q; i++)
                strcpy(Px[i], q_table[i]);
        for (i=0; i<number_atoms[gr_guess]; i++)
                strcpy(Dx[i], table[gr_guess][i]);
}

dist_guess = graph();
cnt_grph++;

cont_1 = true;
pass = 0;

while (cont_1)
{
        ELIM[gr_best] = true;
        ELIM[gr_guess] = true;

        pass++;

        fprintf(file_out, "\n");
```

```
fprintf(file_out, "\nPass #%d\n", pass);
fprintf(file_out, "Best  = %d    Distance = %d\n", gr_best, dist_best);
fprintf(file_out, "Guess = %d   Distance = %d\n", gr_guess, dist_guess);

if (dist_guess >= dist_best)
{
        for (i=0; i<gr_guess; i++)
                if ((i != gr_best) && (ELIM[i]==false))
                        if (dist[i][gr_guess] > dist_guess + dist_best)
                                ELIM[i] = true;
                        if (dist[i][gr_guess] < dist_guess - dist_best)
                                ELIM[i] = true;
        for (i=gr_guess+1; i<MAX; i++)
                if ((i != gr_best) && (ELIM[i]==false))
                        if (dist[gr_guess][i] > dist_guess + dist_best)
                                ELIM[i] = true;
                        if (dist[gr_guess][i] < dist_guess - dist_best)
                                ELIM[i] = true;
}
else
{
        gr_best = gr_guess;
        dist_best = dist_guess;

        for (i=0; i<gr_guess; i++)
                if ((i != gr_best) && (ELIM[i]==false))
                        if (dist[i][gr_guess] > 2 * dist_guess)
                                ELIM[i] = true;
        for (i=gr_guess+1; i<MAX; i++)
                if ((i != gr_best) && (ELIM[i]==false))
                        if (dist[gr_guess][i] > 2 * dist_guess)
                                ELIM[i] = true;
}

i = 0;
cont_2 = true;

while ((cont_2) && (i<MAX))
        if (ELIM[i] == false)
        {
                cont_2 = false;
                gr_guess = i;

                if (number_atoms[gr_guess] <= atoms_in_q)
```

```
                {
                        Pn = number_atoms[gr_guess];
                        Dn = atoms_in_q;
                        for (i=0; i<number_atoms[gr_guess]; i++)
                                strcpy(Px[i], table[gr_guess][i]);
                        for (i=0; i<atoms_in_q; i++)
                                strcpy(Dx[i], q_table[i]);
                }
                else
                {
                        Pn = atoms_in_q;
                        Dn = number_atoms[gr_guess];
                        for (i=0; i<atoms_in_q; i++)
                                strcpy(Px[i], q_table[i]);
                        for (i=0; i<number_atoms[gr_guess]; i++)
                                strcpy(Dx[i], table[gr_guess][i]);
                }

                        dist_guess = graph();
                        cnt_grph++;
                }
                else
                        i++;

        printf("Possible Candidates Pass %d\n", pass);
        number_cand = 0;
        for (i=0; i<MAX; i++)
                if (ELIM[i] == false)
                {
                        number_cand++;
                        printf("%d\n", i);
                }

        fprintf(file_out, "Number Compounds Remaining: %d\n", number_cand);

        if ((cont_2) && (i==MAX))
                cont_1 = false;

}

fprintf(file_out, "\n\n");
fprintf(file_out, "Best match = %d distance = %d\n", gr_best, dist_best);
```

```
        if (k==0)
        {
                best_a = gr_best;
                dist_a = dist_best;
                strncpy(id_a, id[gr_best], 15);
                atoms_a = number_atoms[gr_best];
                for (i=0; i<number_atoms[best_a]; i++)
                        strcpy(table_a[i], table[best_a][i]);

                strcpy(input_db, "grph_db_b");
                strcpy(input_mtrx, "mtrx_b");
        }
        else
                if (dist_a <= dist_best)
                {
                        gr_best = best_a;
                        dist_best = dist_a;
                        strncpy(id_best, id_a, 15);
                        strcpy(input_db, "grph_db_a");
                }
                else
                        strncpy(id_best, id[gr_best], 15);

    }

    fprintf(file_out, "\n\n");
    fprintf(file_out, "Best Match Compound ID: %15s\n", id_best);
    fprintf(file_out, "Input File: %s\n", input_db);
    fprintf(file_out, "Distance: %d\n", dist_best );
    fprintf(file_out, "\nTotal Number of Comparisons: %d\n", cnt_grph);

    fclose(file_out);
    return(0);
}
```

```
/* ********************************************************************** */
/* Program:    long.c                                                     */
/* Author:     Karen R. Pysniak                                           */
/*                                                                        */
/* Linear exhaustive method for searches implementing simulated annealing */
/* algorithm (graph.c).                                                   */
/* ********************************************************************** */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/time.h>
#include "kp.h"

#define ITE     10
#define MAX     100
#define MX      9999
#define min(a,b)        (((a)<(b))?(a):(b))
#define max(a,b)        (((a)>(b))?(a):(b))

char P[MAX][MAX], D[MAX][MAX];
char Px[MAX][MAX], Dx[MAX][MAX];
int Pn, Dn;
int N[105][105], E[27][27], N_cut[105][105], E_cut[27][27];
int Opt[MAX];
int graph();
char ei();
int cost();     /* for caluculating dist; becomes cost_cut() for calculating distwithcut */

main()
{
   int best_a, dist_a;
   int gr_best, dist_best;
   int atoms_in_q;
   int atoms_a;
   int number_atoms[MAX];
   int dist[MAX];
   int cnt_grph;
   int i, j, k;
   char input_db[15];
   char id_a[20], id_best[15];
   char id[MAX][20];
```

```
char q_table[MAX][MAX];
char table_a[MAX][MAX];
char table[MAX][MAX][MAX];
char str[25], line[100];
char ch[3];
boolean ELIM[MAX];
boolean cont_1, cont_2;
FILE *file_in_q, *file_in_db;
FILE *file_out;

file_in_q = fopen("query", "r");

fgets(line, sizeof(line), file_in_q);
sscanf(line, "%d", &atoms_in_q);

for (i=0; i<atoms_in_q; i++)
     fgets(q_table[i], sizeof(q_table[i]), file_in_q);

fclose(file_in_q);

file_out = fopen("long.out", "w+");
strcpy(input_db, "grph_db_a");

cnt_grph = 0;

for (k=0; k<2; k++)
{
     file_in_db = fopen(input_db, "r");

     for (i=0; i<MAX; i++)
     {
          fgets(str, sizeof(str), file_in_db);
          strncpy(id[i], str, sizeof(str));

          fgets(line, sizeof(line), file_in_db);
          sscanf(line, "%d", &number_atoms[i]);

          for (j=0; j<number_atoms[i]; j++)
               fgets(table[i][j], sizeof(table[i][j]), file_in_db);

          fgets(str, sizeof(str), file_in_db);
     }

     fclose(file_in_db);
```

```
for (i=0; i<MAX; i++)
{
        if (number_atoms[i] <= atoms_in_q)
        {
                Pn = number_atoms[i];
                Dn = atoms_in_q;
                for (j=0; j<number_atoms[i]; j++)
                        strcpy(Px[j], table[i][j]);
                for (j=0; j<atoms_in_q; j++)
                        strcpy(Dx[j], q_table[j]);
        }
        else
        {
                Pn = atoms_in_q;
                Dn = number_atoms[i];
                for (j=0; j<atoms_in_q; j++)
                        strcpy(Px[j], q_table[j]);
                for (j=0; j<number_atoms[i]; j++)
                        strcpy(Dx[j], table[i][j]);
        }

        dist[i] = graph();
        printf("dist[%d][%d] = %d\n", k, i, dist[i]);
        cnt_grph++;
}

gr_best = 0;
dist_best = dist[gr_best];
strncpy(id_best, id[gr_best], 15);

for (i=1; i<MAX; i++)
        if (dist_best > dist[i])
        {
                gr_best = i;
                dist_best = dist[i];
                strncpy(id_best, id[gr_best], 15);
        }

fprintf(file_out, "\n\n");
fprintf(file_out, "Best match = %d distance = %d\n", gr_best, dist_best);

if (k==0)
{
        best_a = gr_best;
```

```
                dist_a = dist_best;
                strncpy(id_a, id[gr_best], 15);
                atoms_a = number_atoms[gr_best];
                for (i=0; i<number_atoms[best_a]; i++)
                        strcpy(table_a[i], table[best_a][i]);
                strcpy(input_db, "grph_db_b");
        }
        else
                if (dist_a <= dist_best)
                {
                        gr_best = best_a;
                        dist_best = dist_a;
                        strncpy(id_best, id_a, 15);
                        strcpy(input_db, "grph_db_a");
                }

    }   /*  End of k loop  */

    fprintf(file_out, "\n\n");
    fprintf(file_out, "Best Match Compound ID: %15s\n", id_best);
    fprintf(file_out, "Input File: %s\n", input_db);
    fprintf(file_out, "Distance: %d\n", dist_best );
    fprintf(file_out, "\nTotal Number of Comparisons: %d\n", cnt_grph);

    fclose(file_out);
    return(0);
}
```

REFERENCES

1. Martin, Y.C., Bures, M.G., and Willett, P. Searching databases of three dimensional structures. In *Reviews in Computational Chemistry*(K.B. Lipkowitz and D.B. Boyd, Eds.) VCH, New York, 1990, pp. 213-263

2. Pepperrell, C.A., Taylor, R., and Willett, P. Implementation and use of an atom-mapping procedure for similarity searching in databases of 3-d chemical structures. *Tetrahedron Computer Methodology* 1992, **3,** 575-593

3. Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. A learning algorithm for boltzman machines. *Cognitive Science* 1985, **9,** 147-169

4. Aragon, C.R., Johnson, D.S., Megeoch, L.A., and Schevon, C. Optimization by simulated annealing: an experimental evaluation. An unpublished manuscript 1984

5. Ioannidis, Y.E., and Kang, Y.C. Randomized algorithms for string optimizing large join queries. *Proc. of the ACM-SIGMOD Int'l Conference on the Management of Data* 1990, 312-321

6. Ioannidis, Y.E., and Wong, E. Query optimization by simulated annealing. *Proc. of the ACM-SIGMOD Int'l Conference on the Management of Data* 1987, 9-22

7. Shasha, D., and Wang, T.L. New techniques for best-match retrieval. *ACM Transactions on Information Systems 8* 1990, **4,** 140-158

8. Wang, J.T.L., Chirn, G.W., and Zhang, K. Algorithms for approximate graph matching. *Information Sciences* 1995, **82(1),** 45-74

9. Wang, J.T.L., Zhang, K., Jeong, K., and Shasha, D. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering* 1994, **6(4),** 559-571

10. Burkhard, W.A., and Keller, R.M. Some approaches to best-match file searching. *Commun. ACM 16* 1973, **4,** 230-236

11. Shapiro, M. The choice of reference points in best-match file searching. *Commun. ACM 20* 1977, **5,** 339-343

12. Bionet Research Ltd., 3A Highfield Industrial Estate, Camelford, England

13. CONCORD is distributed by Tripos Associates, St, Louis, Missouri

14. Dalby, A., Nourse, J.G., Hounshell, W.D., Gushurst, A.K.I., Grier, D.L., Leland, B.A., and Laufer, J. Description of several chemical structure file formats used by computer programs developed at molecular design limited. *J. Chem. Inf. Comput. Sci.* 1992, **32**, 244-255

15. UNITY is produced by Tripos Associates, St. Louis, Missouri