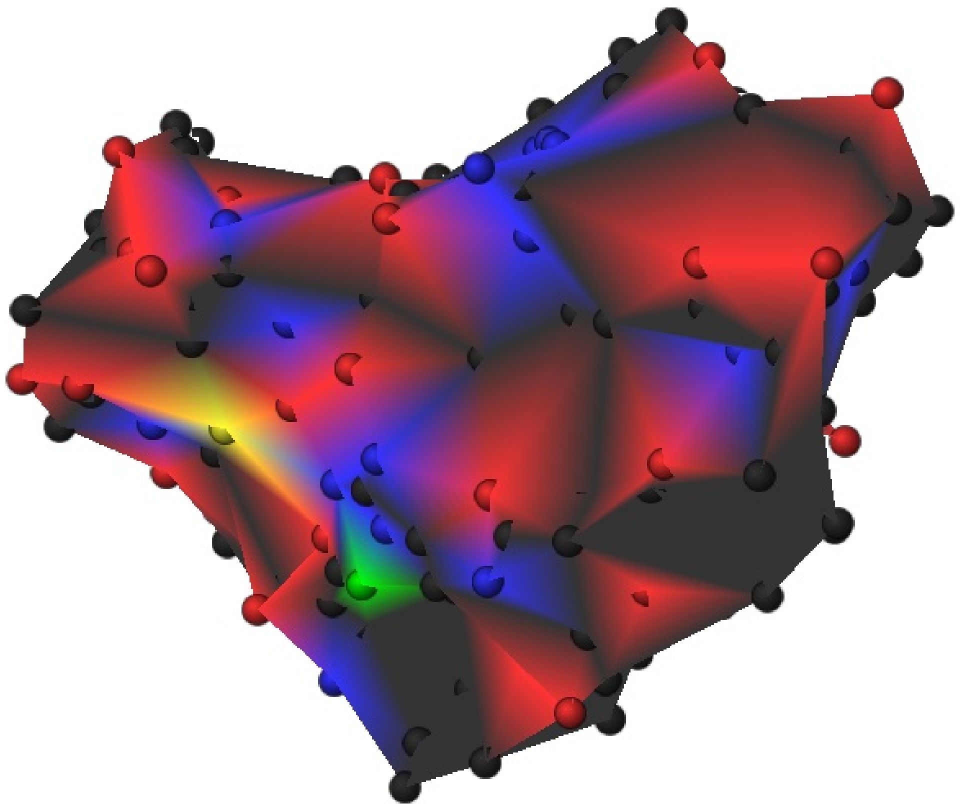
CHALMERS GÖTEBORGS UNIVERSITET



A program for comparing proteins using a graph representation of atom triplet surfaces

Bachelor thesis
Department of Computer Science and Engineering

Daniel Eddeland **Daniel Kvarfordt**
Hjalmar Lindskog **Joakim Sellin**
Christofer Warg **Mikael Weckstén**

The Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2012

Bachelor thesis nr 2012:3

Abstract

This report details the development of software with the capability to find similarities on the surfaces of large biological molecules and display the result graphically in three dimensions. In the field of bioinformatics an approach to represent proteins as triangle surfaces has emerged and it is therefore interesting to have software tools available to examine this structure representation. The software has the ability to identify the largest common surface patch on two proteins as well as letting the user choose a patch to be matched for in the supplementary protein. This is done using an algorithm based on McGregor (1982) with some modifications. The result of the protein comparison is rendered in 3D with the option of rotating the proteins as well as changing several graphical options such as color and the visibility of wireframe and atoms. The resulting program works well for most inputs, but when comparing large, similar proteins the program can fail to terminate within an hour according to performance testing.

Sammandrag

Denna rapport beskriver utvecklingen av ett program som kan användas för att finna likheter på ytan av stora biologiska molekyler och visa resultatet grafiskt i en 3D-modell. Inom bioinformatikområdet finns ett nytt sätt att representera protein som triangelytor, och det kan vara intressant att ha mjukvaruverktyg för att undersöka denna representation. Programmet kan identifiera det största gemensamma området på ytan av två molekyler, eller hitta alla förekomster av ett område som användaren kan välja. Till det används en algorithm baserad på McGregor (1982) som är anpassad till vårt ändamål. Resultatet av sökningen visas i en 3D-modell som användaren kan rotera. Det finns också olika lägen för att visa atomer, färger och kanter. Det färdiga programmet fungerar bra för de flesta molekyler, men för vissa fall där molekylerna stora och likartade kan det ta över en timma för algoritmerna att köra klart enligt våra testkörningar.

Acknowledgements

We would like to thank our supervisors Graham Kemp and Peter Damaschke for their guidance and help during this Bachelor's project.

Contents

Glossary

Bionformatics computer science applied to the fields of biology and medicine.

Heuristics Using experience and knowledge of a problem to improve efficiency in algorithms..

JOGL A software library used to integrate OpenGL functionality into Java..

OpenGL A software library commonly used for rendering of three-dimensional graphics..

PDB The Protein Data Bank, a database containing a large amount of proteins in well-specified formats. The Protein Data Bank is available at http://www.rcsb.org.

SVN a software versioning and revision control tool..

Triominoes a program which takes a protein file in the PDB format and produces a file containing a list of atom triplets which constitute a triplet surface for the protein..

Triplet surface a shell covering a protein, consistent of atom triplets as defined by Mehio et al. (2010)..

1 Introduction

In 1990 the Human Genome Project started, a project whose main goal was to identify and map all of the human genome. In 2003 they succeeded in meeting their goal ahead of time, due in part to considerable cooperation on a world scale as well as significant advances in computer processing speeds (Carroll, 2003). The combination of computer science and information technology with biology and medicine is what is referred to as bioinformatics.

In many disciplines, such as medicine, it might be useful to compare different proteins to find matching areas between them that may share similar biological functions. The use of computers offers many different ways of doing this, and there are also several different representations that can be used for molecule structures; one example being the use of strings to represent proteins (Leskie, Eskin and Noble, 2002).

1.1 The triplet surface format and the Triominoes program

In 2010, Mehio et al. described a new format for representing proteins (Mehio, Kemp, Taylor and Walkinshaw, 2010), which is used to successfully predict biologically significant sites on proteins. The format consists of a surface shell constructed using triplets of atoms in a molecule, and is created by simulating a probe going over the molecule. These atom triplets constitute triangles in a triplet surface model covering the molecule. Fig. 1.1 shows an example of the molecular structure of the protein crambin, as well as a triplet surface representation.

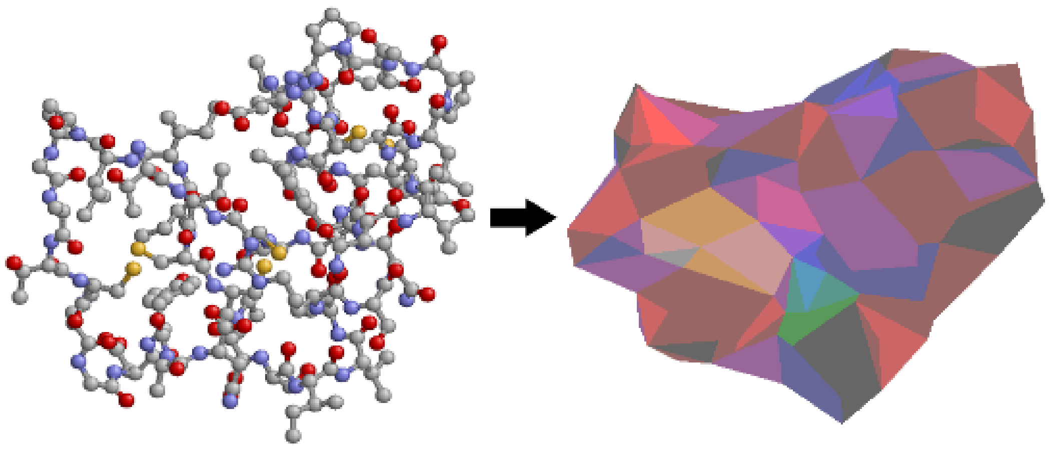


Figure 1.1: The protein crambin (PDB entry 1CRN). The left figure shows crambin's molecular structure, rendered in the program RasMol. The right figure shows the triplet surface, rendered in our program.

Since the format and method is still largely unexplored, it would be interesting to examine the possibilities and limitations of this format. The goal of this Bachelor project is to do so; to help us we are using a program called Triominoes, written by our supervisor Graham Kemp. The program takes a data file in the PDB protein description format and constructs a the triplet surface representation of the molecule.

The triplet surface format describes a closed shell without holes, and models in the format can therefore be described as planar graphs. A graph based approach thus leads to the use of a well established part of mathematics. The significance of using graphs is that the graph algorithms implemented in this project could possibly be applicable to other areas where graphs, planar or otherwise, are used as well.

1.2 Purpose

The purpose of this project is to examine whether the triplet surface format allows for comparison of proteins using standard graph algorithms with reasonable performance. Furthermore, we discuss some of the limitations of using the triplet surface format.

The report documents a basic implementation of a program in which comparison of proteins is tested and evaluated. The program will contain a graphical user interface which will allow for loading proteins in the standard PDB format, comparing them with the implemented algorithms, and presenting a visual representation of the result.

1.3 Previous work

One structure representation of proteins that is in use today in the field of bioinformatics is the string representation (Leskie, Eskin and Noble, 2002). The string representation seems like the natural choice when it comes to some areas of bioinformatics, e.g. when you're representing the structure of DNA. Algorithms for solving the maximum common subgraph problem has several applications, among them the matching of chemical structures (Raymond and Willet, 2002). The maximum common subgraph problem has also been used in a wider range of areas, for example monitoring networks (Vijayalakshmi, Nadarajan, Nirmala and Thilaga, 2011).

1.4 Scope

The program made in this project is expected to work primarily with a certain class of graphs: namely, labeled graphs constructed from triplet surface models. For other graphs the program will work, but for unexpected inputs the program does not need to perform as well. The reason for this is the inherent time constrains present in a Bachelors project.

Although there are highly specialized algorithms for use with certain types of graphs (such as planar graphs) we will use a simple backtracking algorithm for general graphs, and only optimize the algorithm using heuristics. The reason for this is that the focus on the project is on making all the parts work and exploring the basic properties of the triplet surface format, rather than on implementing advanced specialized algorithms. We will not put much emphasis on analysing the algorithms theoretically, but rather on testing our implementations on typical inputs.

The program should be able to read triplet surface models as input from the Triominoes program. The data read should then be used to construct a graph for use with the comparison algorithms, and also combined with the PDB file data in order to make a 3D model of the shell. The program should not focus on reading any other input data than that provided from the Triominoes program, and the corresponding necessary information from the PDB file format.

2 Theoretic background

This chapter briefly describes some terminology that is useful for describing problems within graph theory, as well as some methods for comparing graphs for likeness.

2.1 Simple Graphs and Multigraphs

When discussing graphs it is relevant to distinguish between simple graphs and multigraphs. Simple graphs are graphs such that:

- There is no loop: i.e. an edge that starts and ends in the same node.
- There is a maximum of one edge between any two nodes in the graph.

(Bondy and Murty, 2008)

A multigraph is a graph which allows loops, as well as pairs of nodes with 2 or more edges between them.

2.2 Graph Isomorphism

When comparing graph for likeness, a useful tool is graph isomorphism. A graph isomorphism between two graphs G and H is a bijective function between the nodes of G and the nodes of H, such that; there is an edge between u and v in G if and only if there is an edge between f(u) and f(v) in H (Bondy and Murty, 2008).

If an isomorphism exists between two graphs, the graphs are said to be isomorphic. Graph isomorphism can be more intuitively understood by rearranging graphs: if the nodes of a graph can be moved around so it directly resembles another graph, then the graphs are isomorphic. To illustrate isomorphism, Fig. 2.1 shows two undirected graphs isomorphic to each other. The nodes in the graphs can be moved around so they resemble each other.

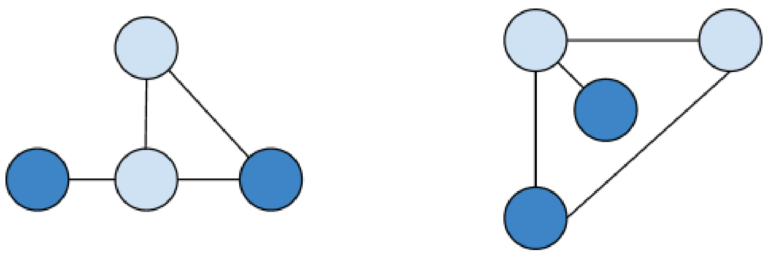


Figure 2.1: Two isomorphic graphs. Each node in the left graph corresponds to the node in the right graph with the same color

2.3 Planar Graphs

A graph is planar if it can be drawn on a paper in such a way that no two edges cross each other (Diestel, 2002). Fig. 2.2 illustrates the difference between planar and non-planar graphs.

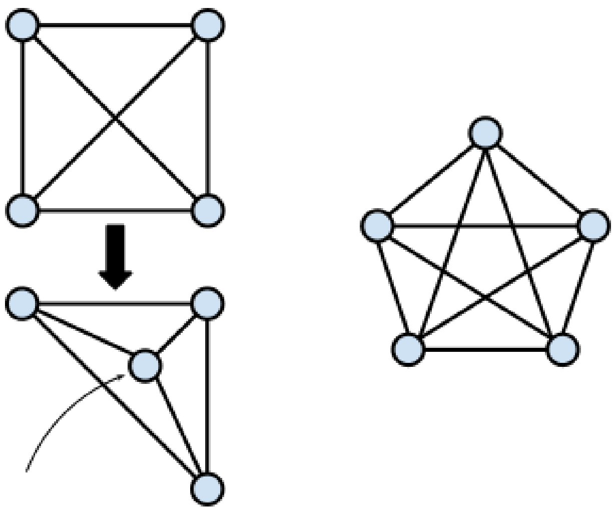


Figure 2.2: The left graphs are planar, but the right is not. Nodes in the upper left graph can be moved so that no two edges cross, while the nodes in the right graph cannot.

Since planar graphs are a subset of all graphs, knowing if a graph is planar can prove to be valuable information; for instance, when constructing search algorithms, this information can reduce the search space.

2.4 Subgraphs

A subgraph F of a graph G, is a graph such that:

1. The nodes in F are a subset of the nodes in G
2. The edges in F are a subset of the edges in G

(Bondy and Murty, 2008)

Any subgraph of a graph G can simply be constructed from G by performing the basic operations:

- Remove any number of edges from G
- Remove any number of nodes from G, as well as removing all edges connected to these nodes.

A special case of a subgraph is an induced subgraph. F is said to be an induced subgraph of G, if:

1. F is a subgraph of G
2. If an edge exists between two nodes in G and those two nodes are present in F, then the edge must be present in F as well.

An induced subgraph can be constructed from a graph G by removing any number of nodes from G, and removing all edges connected to these nodes. The difference between induced subgraphs and non-induced subgraphs can be seen in Fig. 2.3.

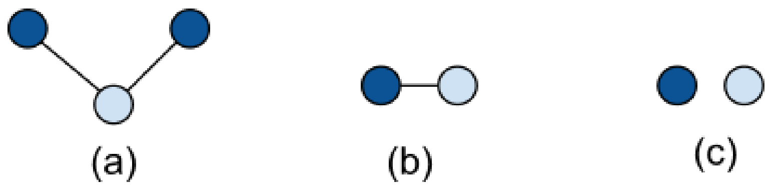


Figure 2.3: The graphs in (b) and (c) are subgraphs of the graph in (a). (b) is an induced subgraph while (c) is not.

2.5 The Maximum Common Subgraph problem

One aspect of interest when comparing two molecules is to find the biggest area which is present in both molecules. The problem of finding the largest graph which is a subgraph of two graphs G and H, is called the maximum common subgraph problem (Marini, Spagnuolo and Falcidieno, 2005). The maximum common subgraph problem can be defined both for disconnected and connected graphs; in this project and report we will focus on connected graphs. Therefore, when the problem is mentioned in the report, it applies to connected graphs unless otherwise is stated. The problem can also be defined for either induced subgraphs only; however, this project will consider the problem for any subgraph, both non-induced and induced ones.

There is no other common subgraph of G and H with more nodes than a maximum common subgraph. However, note that there may be several maximum common subgraphs, with the same number of nodes. Fig. 2.4 illustrates two graphs and one of their two maximum common subgraphs.

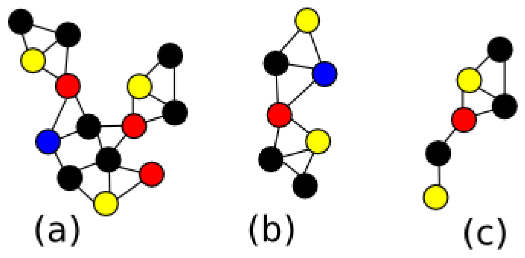


Figure 2.4: Three graphs. The graph (c) is the maximal common subgraph of (a) and (b).

2.6 The Subgraph Isomorphism problem

Given two graphs G and H, finding out whether there exisist a subgraph G' of G, such that G' is isomorphic to H, is known as the subgraph isomorphism problem (Eppstein, 1999). A related problem is finding one or all occurrences of such subgraphs. An example of finding all such occurrences can be seen in Fig. 2.5.

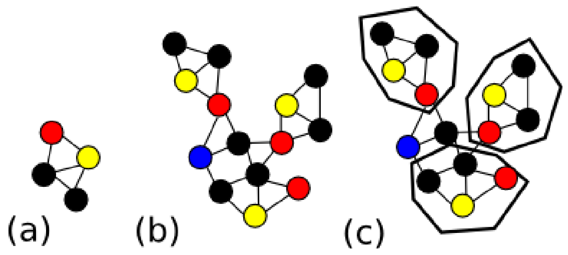


Figure 2.5: Given the query patch shown in (a), and the graph shown in (b), (c) shows all occurrences of subgraphs within (b) that are isomorphic to (a).

In biological structures certain types of regions may be of special interest, such as chemically reactive structures. Therefore the subgraph isomorphism problem, and finding the related subgraphs, is of interest as well in the scope of the project.

3 Method

In this chapter we describe the methodology of the project. We list a number of articles that were part of our initial literary study. Furthermore we list the methods and tools used during the development of the program.

3.1 Literature Review

Early in the project considerable study of the fields of graph theory and algorithms was necessary. Specifically algorithms related to the subgraph isomorphism problem. Only minor research in bioinformatics was necessary in the project. In the later stages of the project the literary study was demarcated to the study of various algorithms. The most significant literature for the project is reviewed below.

The most important literary work that were studied include work done by Raymond and Willet (2002), which provides a good review of many of the Maximum common subgraph algorithms that exists as well as basic explanations of the underlying problems.

Eppstein (1999) describes an advanced algorithm to solve the subgraph isomorphism problem for planar graphs in linear time. Much time was spent on studying this paper although little was implemented in the project.

McGregor (1982) describes an algorithm that solves the maximum common subgraph problem using backtracking. This paper was of great significance for the project as the algorithm described within it forms the basis of the algorithm that handles much of the protein matching in the program.

A paper by Mehio et al. (2010) describes the triplet surface format used in this project. However, the format is only described briefly in the paper. The remaining parts of the paper are unrelated to this project and were thus not part of the literary study.

3.2 Choice of software platform

The programming language used for this project was Java. The main reason for this was that all group members in the project have experience with Java, and it would make development easier to understand and partake in for everyone. Java is the main language for use in many Chalmers and GU courses, and therefore we are familiar with its data structures, and we all know the same coding conventions. As a natural consequence of choosing Java, we also chose to use the Java Swing libraries for creating the graphical user interface in the program.

For the project we needed a framework for 3D graphics, in order to visually render the protein structures loaded into the program. There are two major such frameworks; DirectX and OpenGL. In order for the program to be platform independent we chose to use OpenGL through the JOGL library. This was important since all members of our group do not use the same operating system. Another benefit of using OpenGL with JOGL is that some members in our group have worked with both OpenGL and JOGL before.

3.3 Development process

The development process used in the project is best described as dynamic. The major points in the project such as individual algorithms, graphics and several other tasks were assigned to specific people or groups.

Smaller tasks were not assigned by the group but were done on an individual group member's discretion. There were several meetings every week, online or in person, with feedback and status updates on the work done as well as the possibility of new assignments being delegated. An SVN server was used for version control and to facilitate concurrent and individual programming.

3.4 Representation of proteins

In this project we needed to use protein data files for two reasons, the first being that the Triominoes program takes a protein data file as input to construct the triplet surface for that protein. Second, geometric data about proteins are needed to correctly render triplet surface models to the screen.

3.4.1 Protein Data Bank file format

We decided to use the Protein Data Bank (PDB) file format as input to our program. The reason for this is mainly that this is the format that the Triominoes program uses as input. For simplicity we also use the same format to parse geometric information, the positions of the atoms.

3.4.2 Internal graph format for describing triplet surfaces

The triplet surface format can easily be described using standard graph formats. This, combined with the fact that graph algorithms are well documented, and that we were advised by our supervisor to implement graph algorithms for protein comparison, led us to choosing a simple graph representation in our program. The graph representation we chose was undirected labeled simple graphs using adjacency lists. This was because of the properties of typical triplet surfaces constructed from proteins; the resulting graphs are sparse, meaning that they have few edges compared to the number of nodes. In the triplet surfaces constructed we expected the number of outgoing edges from a node to not be significantly higher than ten on average, while the number of nodes for some graphs would be several thousand.

We developed the graph data structures in the program during the course of the project, rather than using an external graph library. The reason for this was that we wanted the graph system to be customizable to our needs, but not much more complicated than we needed for the scope of the project.

3.5 Evalution and Performance Testing

When running an algorithm the program keeps track of time taken to perform the comparison. This was used as our test time. The test focused on the maximal common patch algorithm since the subpatch search algorithm is just a extension off it.

We picked proteins that we knew shared patches and went from a small number of atoms to a quite large number of atoms. We compared them both to themselves, yielding a 100

One computer were used for all the tests. All other programs that might have interfered (anything that is not essential for running either the OS or the program itself) with the results were turned off from the start. Before taking the 10 test-results we ran two test-runs so that the OS did not interfere if it needed to allocate any resources to the program. Then the 10 comparisons were performed and noted.

The program itself were of the current stable build, Version 355. It used the default settings if not otherwise noted at the test results. This means that the probe was set to size 1.5, the H2O atoms were discarded and triangles were used as nodes.

4 Implementation

In this chapter we describe the problems that our program was faced with, and the main components of the program as well as the implementation choices that were made.

4.1 Construction of graphs from triplet surfaces

To be able to examine the triplet surfaces in an efficient way inside the program, they should first be converted to a graph format.

The most intuitive way of converting a triplet surface to a graph is to simply use all the atoms on the surface as nodes. If two nodes are both in any atom triplet, then there is an edge between them in the graph. Fig. 4.1 details a triplet surface model where atoms are clearly shown as spheres, and edges in the atom graph are displayed as white lines.

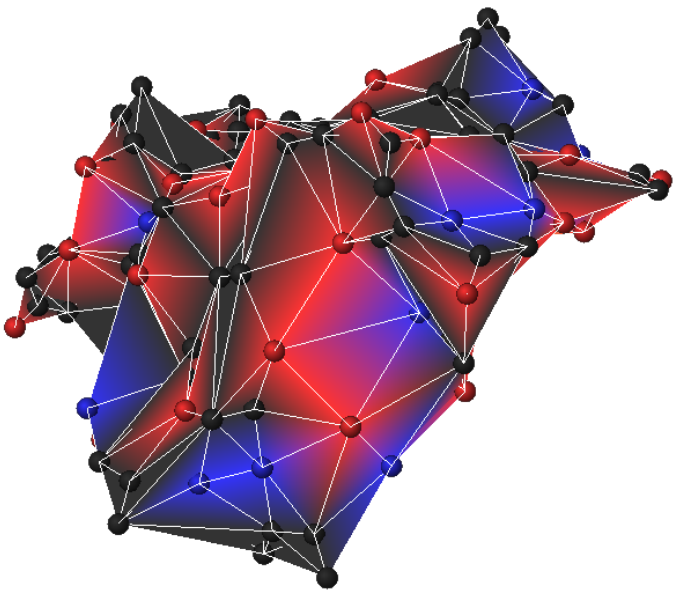


Figure 4.1: A triplet surface model of crambin (PDB entry 1CRN) showing atoms and edges between them.

4.2 Triangle graphs

Constructing graphs using atoms as nodes is simple and intuitive; however, there is another graph representation which might be of interest. The alternative way is to consider the dual of the atom graph, and treat the atom triplets as nodes. Edges in the graph are constructed between two nodes if the corresponding atom triplets are adjacent in the triplet surface. Fig. 4.2 shows the difference in graph structure for the different graph types.

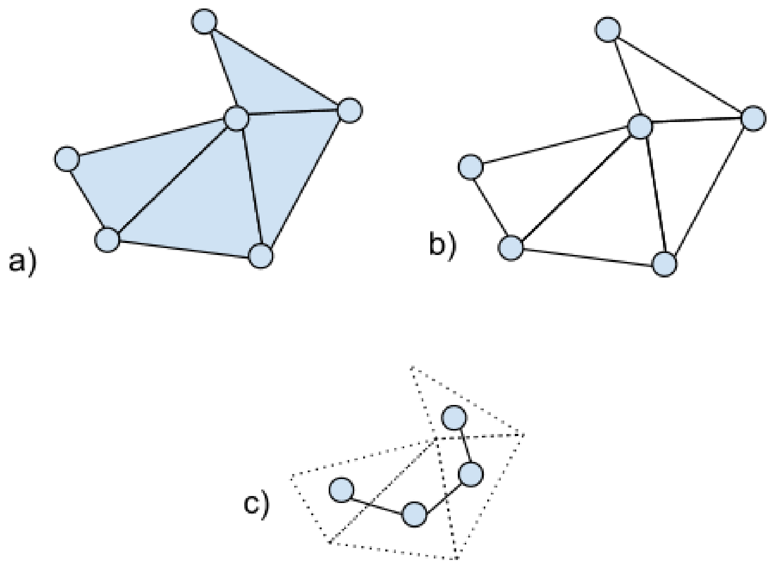


Figure 4.2: a) shows a simplified section of a triplet surface, while b) shows the corresponding graph with atoms as nodes and c) shows the graph with atom triplets as nodes.

An inherent property of the triangle graph is that there are likely to be a larger number of unique label types. There are 13 types of atoms commonly found in proteins; however, when ignoring the orientation of atoms in a triangle, there are 455 different types of atom triplets using the same 13 atom types (Mehio et al., 2010). The larger number of label types mean that there generally are fewer nodes for a given label type. This would potentially make it easier for the algorithm to discard solutions quicker.

One other characteristic of triangle graphs is that they only allow matches which are at least one triangle broad, whereas atom graphs would allow matches that are just one atom thin. This is a limitation, but in our application it can actually be an advantage. Allowing lines of single edge width would make the search space much bigger, and might connect patches that are far apart, which is not desirable as we want to match compact patches.

4.3 Details of triangle graph generation

When constructing the triangle graphs, we had to take precautions when deciding which triangles should be neighbors. It is not enough to just connect every triangle pair that share an edge, since there are a number of cases where more than two triangles share an edge. For two adjacent triangles to be neighbors in the graph, firstly the ordering of the two mutual atoms must be opposed, meaning that the triangles face the same direction across the edge. Triangles B and C in Fig. 4.3 should not be neighbors for instance. Second, for a triangle to be a neighbour to another, the other must be the closest of the suitable triangles around the edge. In other words, the angle in the direction of the normals must be smaller than for any other triangle, which is the case for triangle A and B in Fig. 4.3. If care was not taken, A and C in the figure would be considered neighbours, and the resulting graph would not necessarily be planar. Note that the in the case shown in the figure, triangle B would have a superposed triangle with opposing normal, as the atom triplet shell given by Triominoes is closed, but we have omitted it for clarity.

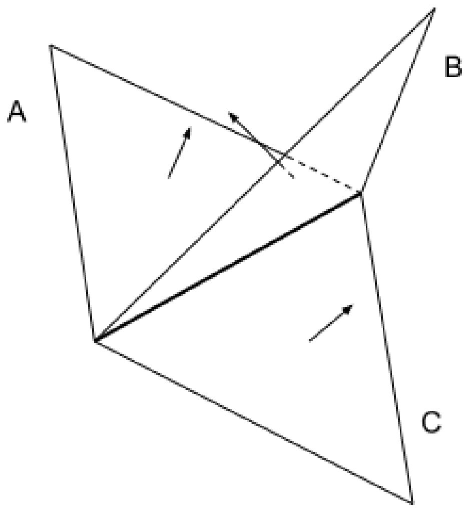


Figure 4.3: Three triangles, of which only two should be neighbours in the graph.

If these two criteria are followed, each node in the resulting graph will have no more than three neighbours. It is however possible for a node to have only two neighbours. This happens where there is a triangle “fin”, two superposed triangles that have opposing normals and share two edges, sticking out of the surface, see Fig. 4.4. This does not pose a problem as the resulting graph is still planar. In our program, all graphs used are simple graphs, which means that even though triangles ‘A’ and ‘a’ in Fig 4.4 are neighbors across two edges, there is only one edge connecting their nodes.

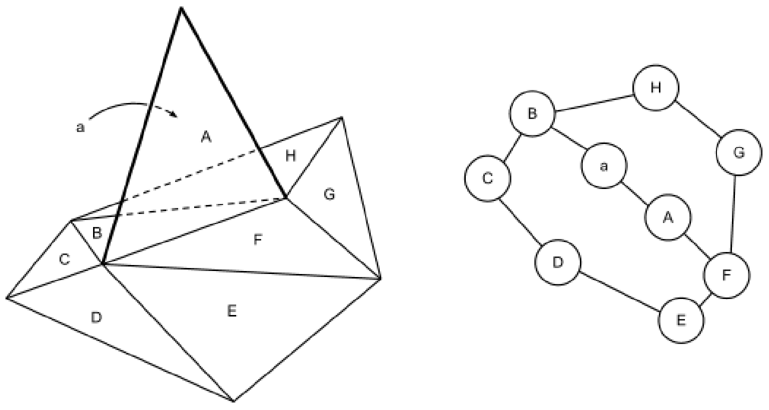


Figure 4.4: A double sided “fin” on the left, and the generated graph on the right. In the left figure, a is the back and A the front face of the protruding triangle.

The labels of the nodes in the triangle graph are constructed by concatenating the labels of its atoms. In order to make the labels of equal triangles equal, we sort the atom labels lexically. But just sorting and concatenating the labels would not preserve the information about the direction (handedness) of the atoms labels. Therefore, we also detect if the lexical direction of the atom labels is the same as the handedness of the triangle. If they are opposite, we invert the sorting order in the combined label. For example, a triangle with the atom labels A-B-C, B-C-A or C-A-B in the direction of the triangles atom order would have a combined label A:B:C. A triangle with the atom labels C-B-A, B-A-C or A-C-B would have a combined label with the inverted order; C:B:A, see Fig. 4.5. The colons in the labels make sure that the atom labels of the combined label are uniquely identifiable, and also make the label more readable by a human.

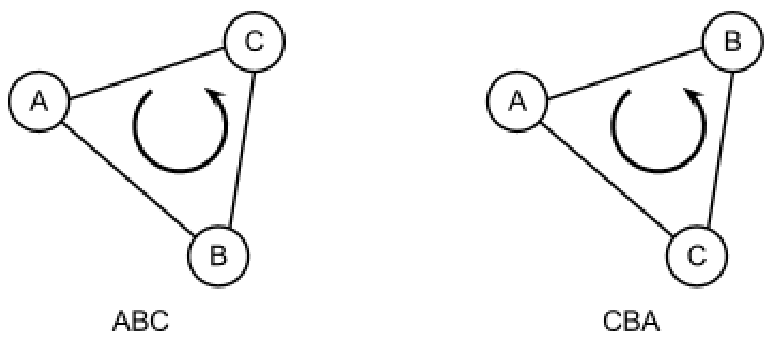


Figure 4.5: Triangle labels with preserved atom label handedness. The left triangle's atom labels are positively ordered, and thus its label is sorted normally. The right triangles atom labels are negatively ordered, so its label is inversely sorted.

4.4 Choice of algorithm

One of the most fundamental decisions in this project was the choice of algorithm for the comparison of proteins. During the initial stages of the project much time was spent researching different algorithms related to the subgraph isomorphism problem, as well as discussing within the group and with our supervisors how to tackle the problem. Below we detail the various algorithms that were studied and considered for the problem, including the ones that we subsequently choose to implement in the program.

4.4.1 Maximum common subgraph of two graphs

The main graph problem we solved during this project was the maximum common subgraph problem, as defined in chapter 2 of this report; the problem was restricted to connected simple graphs, but accepted both induced and non-induced subgraphs.

McGregor (1982) describes a basic backtracking algorithm used to solve the maximum common subgraph problem for unlabeled graphs. Using this algorithm as a base, we developed an algorithm for our graph data structures, specialized for labeled graphs.

McGregor's algorithm is based on a complete backtracking method, where the entire search space is explored. For that reason, the maximum common subgraph search is assumed to be correct; the algorithm is fine-tuned by removing unnecessary searches, without removing any valid solution from the search space. Secondary optimizations were added to our algorithm which were not in McGregor's paper; such as comparing node labels and edge labels. Furthermore, McGregor's algorithm finds common subgraphs which are both connected, as well as disconnected; on the other hand, our algorithm narrows down the search to only connected subgraphs.

Our modified algorithm takes two graphs G and H as input. The algorithm then works by considering all the nodes in G, and either map them to a node in H, or decide that the node in question is not part of the solution. The main steps of the algorithm are described as follows:

1. If G is not partly mapped yet, pick any node u from G which is not marked as "not in the solution". Otherwise pick a node u from G that fulfills:
    - u must not be mapped already, and must not be marked as "not in the solution"
    - u must be adjacent to a node which is already mapped
2. Pick a node v from H, which fulfill the conditions:
    - v must not be mapped already
    - the mapping (u,v) has not been tried for this recursion level.

- u and v must have the same node label.
- u must be adjacent to an already-mapped node u', and v must be adjacent to an already-mapped node v'
- The edge label for edge (u,u') must match the edge label for edge (v,v').

3. If two nodes u and v were found in step 2, then tentatively map u to v. Then recursively map the rest of graph G, from step 1. After the recursive search is done, mark the mapping (u,v) as tried for this recursion level, and remove the mapping (u,v). Then repeat step 2 again to find new nodes which may match u.
4. All mappings for the node u has been tried, so mark u as "not in the solution". Then recursively try to map the rest of graph G from step 1.
5. Before heading to the previous recursion level, remove u from the "not in solution" list. This is because u may be mapped in another part of the search tree.

When choosing what nodes to explore first in steps 1 and 2, the algorithm uses heuristics to value the possible candidates. Candidates for u are preferred with as many mapped neighbours as possible; this reduces the search space, and compact patches are more likely to be explored early than elongated ones. When picking candidates for node v, nodes are preferred whose surrounding nodes match those surrounding u. Nodes are also preferred with labels that have as few occurrences as possible in the graphs.

Pseudo-code for the algorithm can be found in Appendix C.

4.4.2 Subgraph Isomorphism algorithm

For this problem, we used a modified version of the algorithm used for the maximum common subgraph problem described in the previous section. The modifications are comprised of a number of preprocessing steps and heuristics to minimise the search space, with improved performance as a result.

The preprocessing steps unique to this algorithm work as follows:

1. Remove all nodes from the graph that have labels that are not in the patch. Also remove the edges which have endpoints in these nodes. This might result in a disconnected graph. Normally, doing this reduces the search space by a great deal, since there are many different labels in the graphs. An example of the result of this is shown in Fig. 4.5c.
2. Consider all the connected components of the graph. If such a component has fewer nodes than the patch, we can remove the component, since it trivially can't include a matching. If it has less nodes of a certain label than the patch, we can also safely remove it, as it can't possibly match the patch's nodes with this label. In Fig. 4.5c, the connected component g1 does not have a blue node which the patch does. Furthermore, it has only two nodes while the patch has three. For these reasons it should be removed.
3. If there are any connected components left, use the maximum common subgraph alogorithm to see if there is a match. If the biggest match found is smaller than the patch, nothing is returned. If a matching of the size of the patch is found, it is returned immediately, and the algorithm terminates.

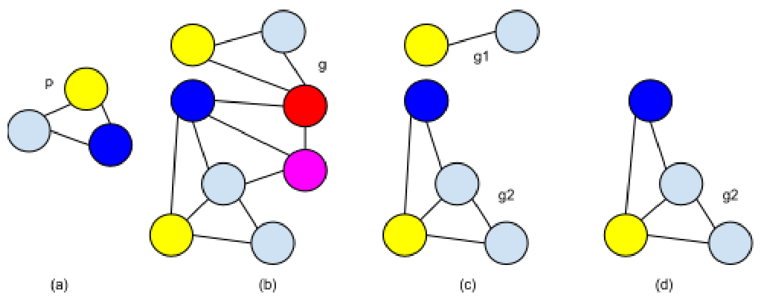


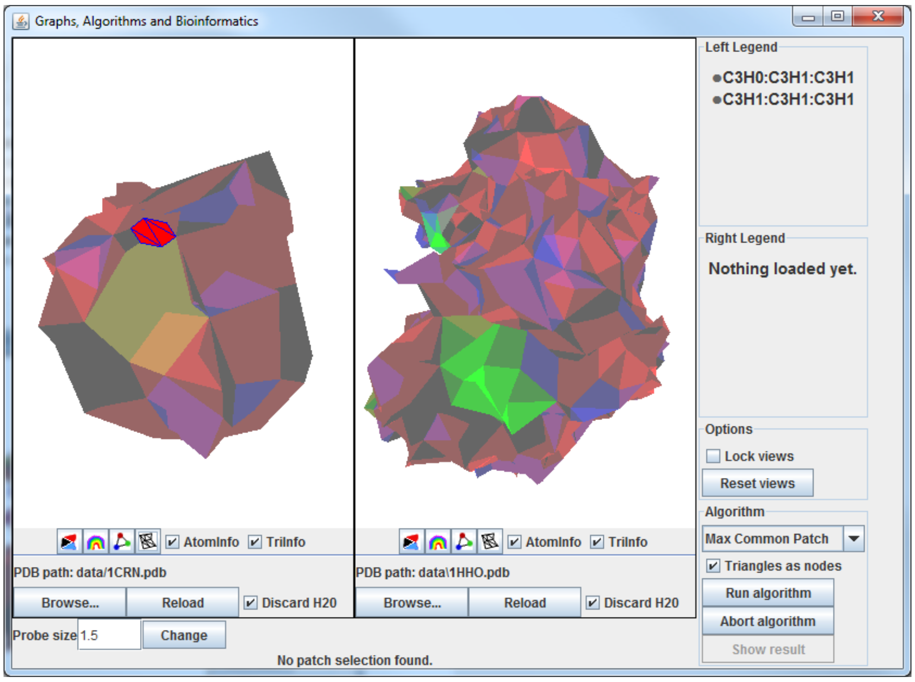
Figure 4.6: (a) A possible input patch to the algorithm, (b) a possible input graph g. (c) shows g divided into connected components g1 and g2, after the first step has been performed. (d) is the result after the preprocessing steps. The remaining graph g2 and the patch p are now given as input to the Maximum Common Subgraph algorithm.

We first considered using an algorithm described by Eppstein (1999), for this problem. The reason for this was that the algorithm exploits the fact that the input graphs are planar, making it very fast even in the worst case. Implementing the algorithm was however very hard, since it uses a considerable amount of preprocessing, building up complicated data structures to reduce the running time. Also, the purpose of the paper seemed to be more to show that it is possible to write such an algorithm, rather than describing how it could actually be implemented; this made the algorithm even more difficult for us to realize in practice. It did however inspire some of the preprocessing steps described above, in particular, dividing the graph into several smaller components.

4.5 Graphical user interface

We started working on the graphical user interface (GUI) very early, to have an environment that allowed us to run our algorithms and see the result easily.

The GUI can be seen in Figure 4.6. It has two panels showing the currently loaded molecule models. Below each of the molecules there are a number of buttons for different display modes. These can make it easier to see the shapes of the molecules and where triangles and atoms are located. For example, one of the buttons toggle wireframe drawing, which means that the contours of the triangles are drawn. The molecule panels also contain controls for loading PDB files. Below the molecule panels there is an input to change the Triominoes probe size. To the upper right are two panels listing the labels of the nodes in the current selection. In the lower right there are controls for selecting and running one of the searches on the graphs. There is also a checkbox for specifying if the algorithms should be run on the atom- or triangle graph representation.


Graphs, Algorithms and Bioinformatics

PDB path: data/1CRN.pdb

PDB path: data\1HHO.pdb

Probe size 1.5

No patch selection found.

Left Legend

●C3H0:C3H1:C3H1
●C3H1:C3H1:C3H1

Right Legend

Nothing loaded yet.

Options

Lock views [ ]

Algorithm: Max Common Patch

Triangles as nodes [✓]

Figure 4.7: The user interface.

All of our algorithms can be run from within the GUI, and the resulting matches are displayed directly in the 3D models, or can be inspected in text format by pressing the “Show result” button.

The Subgraph Isomorphism algorithm takes as input a selection of triangles in one of the molecules, and thus we had to add a way to highlight triangles and atoms easily. For that purpose we implemented mouse picking, so that you can click directly in the model to make a selection of triangles.

4.6 Program workflow

A PDB-file is loaded into the program using a file browsing dialog, after which Triominoes is run on the selected file. The program then interprets the Triominoes output and the PDB data in order to build a graph and a 3D model of the molecule. The 3D model shows up in one of the panels.

When two molecules have been loaded, the user can select one of the available searches. As the algorithm is run, the current status is displayed in the models. In the case of the maximum common subgraph search, the current best matches are highlighted.

When the algorithm is finished, the user can rotate and zoom in on the model to inspect it closer. The resulting patches can be inspected in more detail by hovering them with the mouse cursor. This will show a text overlay with detailed information about the triangle or atom; see Fig. 4.7.

It is also possible to get the resulting matches in text format by clicking the Show result-button.

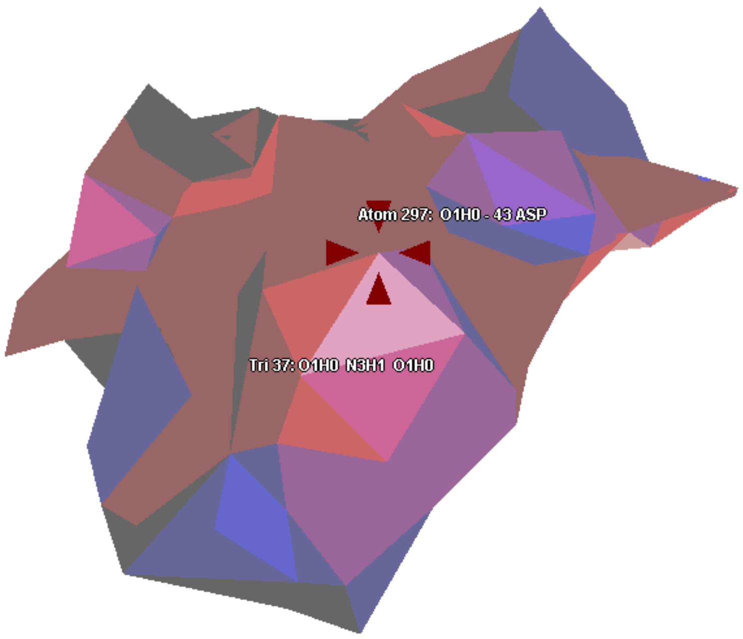


Figure 4.8: The text overlay for triangle and atom mouse-over. For the atom it shows atom index, name, residual number and name. For the triangle it shows the triangle index and the types of its three atoms.

4.7 Program architecture

The main architecture of the program, pictured in Fig. 4.8, was separated according to the Model-View-Controller pattern as much as possible:

1. Model: the classes containing information about the problem domain. In the case of this project, this includes the classes for maintaining information about proteins and surface triplets, as well as all the graph- and algorithm-related classes.
2. View: the classes purely concerned with presentation of data. The pure View classes are those closest interacting with the OpenGL graphics library through JOGL.
3. Controller: the classes used to read input from the user interface. Because the program uses Swing for creating a user interface, these classes contain both Controller and View functionality; however, the main function of these classes is to read user input and delegate events to the corresponding parts of the program.

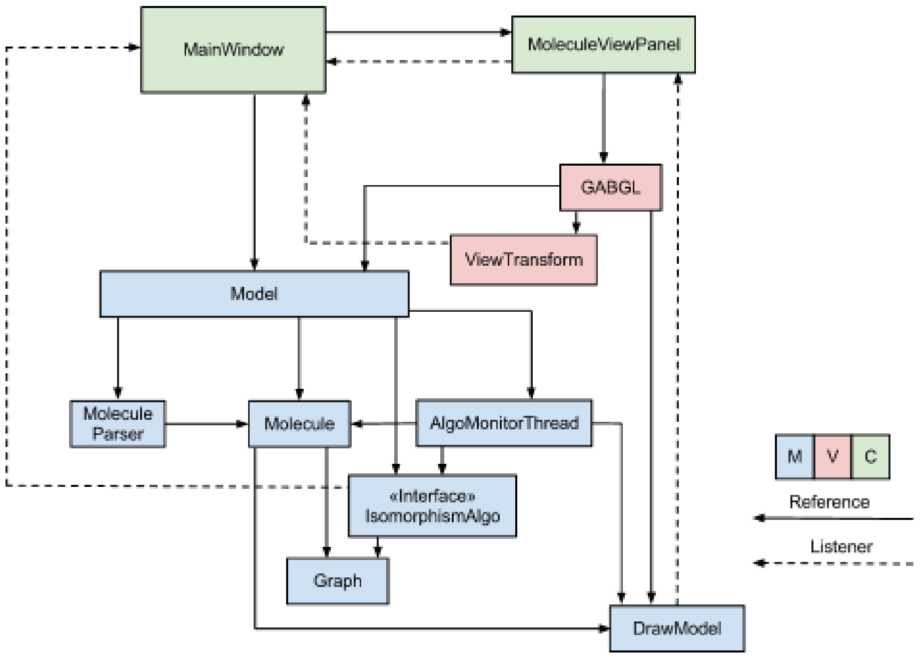


Figure 4.9: A diagram showing the underlying classes of the program, separated according to the Model-View-Controller pattern. Model classes are blue, view classes are red, and controller classes are green.

The Model-View-Controller architecture provides better modularity. The algorithms used have a common interface called IsomorphismAlgo, which makes it easy to add new implementations.

5 Result

In this chapter we describe the program that was the result of the development process.

5.1 Final program functionality and limitations

The final program that is presented here contains many of the features we hoped to have in the program at the onset of the project. Some limitations were necessary however because of a combination of time constraint and lack of expertise in certain areas. The major features of the program are detailed below.

The main purpose of the program is to provide functionality to conduct surface-patch matching of proteins. This functionality comes in two variations: finding the largest common surface region between two proteins and selecting a contiguous surface patches on one protein to be matched for on the other. We will begin by describing the Max Common Patch functionality, which will be abbreviated MCP.

The MCP functionality works by invoking the maximum common subgraph algorithm that is implemented in the program. The matching is limited to two proteins at once, chosen by the user in the form of PDB files. The two protein limitation is partly due to time constraints but also the fact that the underlying algorithm is relatively slow for certain classes of proteins, e.g. two proteins that have many similarities but are still different enough to require considerable searching by the algorithm. Matching proteins with a substantial amount of symmetry in the atom structure can also pose a problem for the algorithm. Figure 5.1 below shows the result of a matching between two proteins.

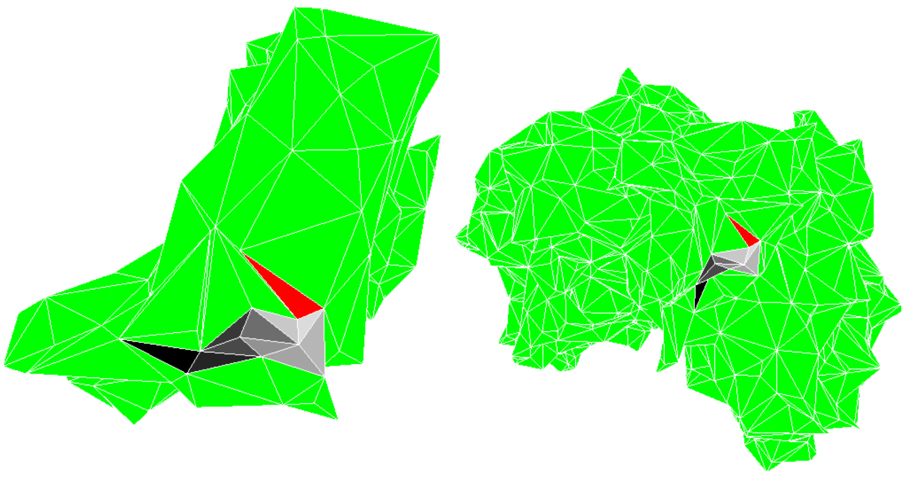


Figure 5.1: The maximum common surface region of crambin and hemoglobin as found by the protein matching algorithm in the program. Green patches are not part of the match.

As is evident by the figure, the result is displayed in 3D. The user also has the ability to rotate the proteins, toggle atom and wireframe visibility as well as change coloring scheme and lightning.

The other variation of the surface-patch matching functionality is what we call the subpatch search feature. As was described above, the subpatch search functionality allows the user of the program to select one or several contiguous surface patches on one protein, to be matched for in the other. The subpatch search function is based on the same algorithm as the MCP, with a small number of modifications. The protein comparison is again limited to two proteins at once, and the reasons for this are the same as for the MCP functionality. The subpatch search algorithm is also limited to triangle graphs, graphs consisting of 3 atoms, single atom matching is not possible. The surface region the user selects in one of the proteins can have zero, one or several corresponding regions in the other protein. The result of a subpatch search search can be seen in Fig. 5.2.

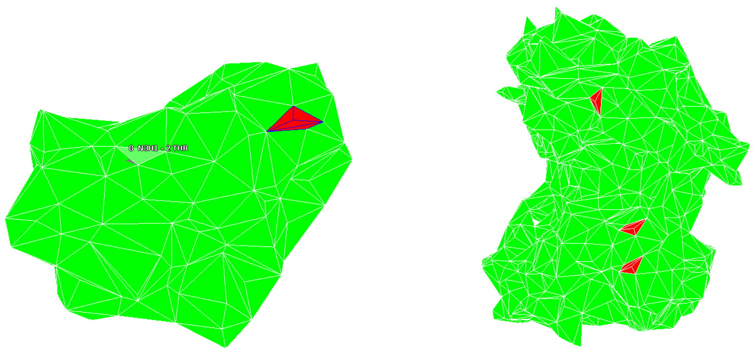


Figure 5.2: The patch highlighted in the left protein is matched to three surface patches on the right protein.

In conclusion it is correct to say that the final program works well, but on a small scale. The limitation that was made to only allow protein comparison of two proteins at once limits the usefulness of the program to some degree, and at the same time it opens up possibilities for future work.

5.2 Performance testing

In this section we describe the performance of the main algorithms present in the final program.

5.2.1 Maximum common subgraph algorithm performance

The results for finding the maximum common subgraph between to graphs was varied, depending on the input graph. For two isomorphic graphs, the program successfully found the isomorphism very quickly. The program also worked satisfactorily when applied to two graphs which were very different. For instance, when comparing PDB proteins crambin (1CRN) with hemoglobin (1HHO) the algorithm terminated within 12 seconds using our dedicated test computer (see Appendix D for specifications of the computer and the complete tables of the test runs).

Protein	1CRN	1HHO	1TRN	2PTN
Time in seconds (average of 10 runs)	1.3	29.8	85.7	15.9
Nr of nodes in solution	444	2464	3502	1628

Table 5.1. Measured time of running the maximum common subgraph algorithm comparing a protein to itself.

Protein	1CRN vs 1HHO	1CRN vs 1TRN	1CRN vs 2PTN	1HHO vs 2PTN
Time in seconds (average of 10 runs)	20.2	37.7	8.6	79.7
Nr of nodes in solution	11	11	9	13

Table 5.2. Measured time of running the maximum common subgraph algorithm on different proteins

Protein	1TRN vs. 2PTN (only A-chains)	1CRN with probesize 1.5 vs probesize 1.8
Time to terminate	Did not terminate	Did not terminate
Time in seconds to find best solution	2	3
Maximum common subgraph size (found within time constraints)	70	338
Graph sizes (nr of nodes)	1760 / 1642	444 / 408

Table 5.3. Test results of protein compared with other proteins that are similar.

5.2.2 Subgraph Isomorphism algorithm performance

We have not performed any measured tests of the subgraph isomorphism algorithm. One reason for this is that this algorithm is just a special case of the algorithm above, so these cases could also apply to the subgraph isomorphism algorithm. Another reason is that for every pair of proteins to test, the number of different test cases are exponential in the number of nodes of the graphs.

6 Discussion and conclusion

In this chapter we discuss obstacles, limitations and possible future work on our project as well as provide final concluding remarks.

6.1 Problems encountered

The main obstacle in the implementation of the program was to construct the maximum common subgraph algorithm that handles the matching of proteins. Initially we started out researching algorithms relating to the problem of subgraph isomorphism, but we were unable to find any algorithm that was plausible for us to implement because of the inherent complexity of the algorithm or the fact that the problem the algorithm in question solved deviated too much from the core problem in the project.

The initial difficulty of finding a suitable algorithm for the maximum common subgraph problem forced us to try to write our own algorithm. Solving the problem from scratch turned out to be very difficult, and the algorithms that were initially constructed suffered from performance issues, and more importantly: they didn't always produce a perfect result. This lead us to further investigate existing algorithms related to the problem. Subsequently we found the algorithm that forms the basis for the maximum common subgraph algorithm in the program; McGregors's backtrack algorithm for the maximum common subgraph problem (McGregor, 1982). This algorithm was implemented with some modification to suit our specific problem, and it worked well enough to form the basis of the protein comparison in the program.

6.2 Limitations of the atom triplet surface format

The output from Triominoes for a certain molecule is not always deterministic, in the sense that the same molecule can have different triangulations depending on rounding errors and the frame of reference of the atom positions. A common example of such a degenerate case is aromatic rings, where atoms make up a ring shape where they all lie in the same plane.

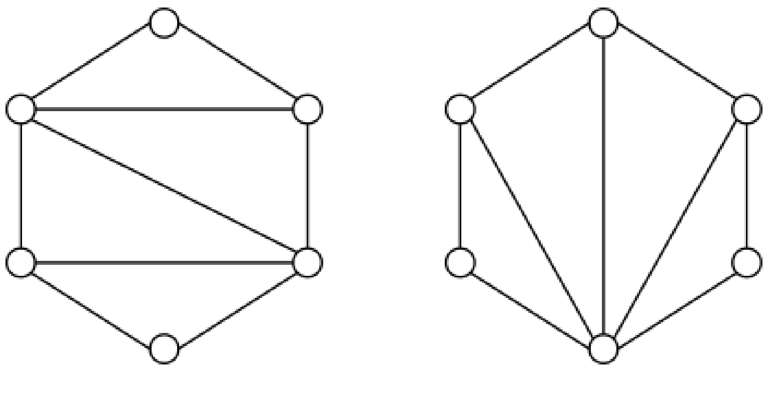


Figure 6.1: Two different triangulations of the same aromatic ring structure.

The two triangulations in Fig 6.1 will lead to different graphs, and thus a match between them would not be found by our algorithm. The same input file will always generate the same output, however, so in that sense the output is deterministic. This limitation is due to the atom triplet surface format, and not our algorithms. One approach which has been proposed in an ongoing masters project at Chalmers University of Technology[1], is to solve this problem by putting a pseudo-atom in the middle of the ring, which will ensure a consistent triangulation, see Figure 6.2.

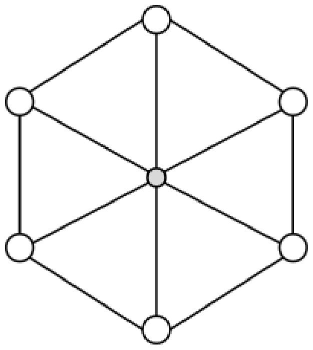


Figure 6.2: A pseudo atom in the middle of the aromatic ring ensures a consistent triangulation.

6.3 Future work

Naturally, there are many ways in which this atom triplet surface approach could be taken further in the future. Here we discuss some of them.

6.3.1 Database searches

One extension to the program could be to let the user make database queries to find all occurrences of a patch. The database could for instance be an online database with PDB files, and the result could be a list of the proteins that contains a matching patch. In general, it would be useful to be able to compare more than two proteins at once which is the current limitation in our program.

6.3.2 Command line interface

Also, the ability to run the program from a command line would be a useful feature, especially in combination with the ability to make database queries, as the GUI slows the program down a bit. This would also make it easier to use the program in combination with other software. For instance, our program could be called as a subroutine from within another program, that might have its own user interface.

6.3.3 Using a more efficient algorithm

The final program makes use of a relatively simple backtrack algorithm for the maximum common subgraph problem that, although sufficient for most comparisons in our program, would struggle in comparing dozens or more proteins at once. For future work a more specialized algorithm like David Eppstein's algorithm (Eppstein, 1999), could be considered. This algorithm is specialized to work with planar graphs, which is sufficient for most cases when making protein surface comparisons. While no known algorithm solves the problem in sub-exponential time, Eppstein's algorithm solves the subgraph isomorphism problem in linear time (when one of the graphs has a fixed size), which would open up the possibility to make comparisons of a considerable amount of proteins at once.

[1]Graham Kemp, email correspondence (2012-05-13)

6.3.4 Shape comparison of matches

The matches obtained from our program do not consider the actual physical shapes of the two areas found, only the similarity of the subgraphs. Therefore it might be useful to be able to evaluate how closely the two patches line up in space. One way to do this would be to use a point cloud matching algorithm to get a value of the error, and take that into account when calculating the quality of the match. However, the three dimensional model visualisation of the matches in our program can give a pretty good feeling for how similar the patches are, and the fact that we show more than one match in the case of the maximum common subgraph can help in finding the one that seems the most promising, but this requires manual work.

6.4 Concluding remarks

This project consists of several parts, all which have been challenging in different ways. To build a GUI designed for chemists when the authors/programmers have little, if any, insight in the subject of chemistry was a challenge in its own right, especially since we have not been able to let the potential users of the software test it during the development process. Furthermore, trying to write efficient algorithms to solve the problems we faced in the project proved to be a difficult and time consuming task. With this in mind, the outcome of the project was satisfactory. The algorithms that are implemented in the program do not always perform perfectly with all input; however, for the comparison of two proteins the performance is acceptable.

The user interface looks appealing, even for users that might not know what the program does, and seeing the results in a three dimensional visual context is a huge step forward from being forced to compare text files with each other. Even though the program might not be a full-fledged product, ready for commercial or academic use at this point, there is definitely room for future improvements and extensions. Since the software is built in a modular, object-oriented environment, there are possibilities to improve the different parts of the program independently; from the parsing of PDB files and extraction of atomic triplets, to the implementation of algorithms and the user interface. The project can also hopefully inspire others to explore the possibilities of using a graph representation of proteins, combined with a human friendly user interface.

References

- Bondy, J.A; Murty, U.S.R. (2008) *Graph theory.* London: Springer. (Graduate text in mathematics, Vol 244).
- Carroll, S. B. (2003). Genetics and the making of homo sapiens. *Nature*, 422(6934), 849-57.
- Diestel, R. (2010) Planar Graphs. In *Graph Theory*, red. pp. 87-114. Heidelberg: Springer.
- Eppstein, D. (1999) Subgraph Isomorphism in Planar Graphs and Related Problems. *Journal of Graph Algorithms and Applications*, Vol 3, Nr 3, pp. 1-27.
- Leskie, C; Eskin, E; Noble, W.S. (2002) The spectrum kernel: A string kernel for SVM protein classification. In the 7th Pacific Symposium on Biocomputing, pp. 566-575, Lihue, Hawaii, USA
- Marini, S; Spagnuolo, M; Falcidieno, B. (2005). From Exact to Approximate Maximum Common Subgraph. Graph-Based Representations in Pattern Recognition, Editors: Brun, L; Vento, M. pp. 133-135. Berlin: Springer.
- McGregor, J. (1982) Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, Vol 12, Nr 1, pp 23-34.
- Mehio, W; Kemp, G.J.L; Taylor, P; Walkinshaw, M.D. (2010). Identification of protein binding surfaces using surface triplet propensities. *Bioinformatics*, Vol 26, Nr 20, pp. 2549-2555.
- Raymond, J.W; Willet, P. (2002) Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design.* Vol 16, Nr 7, pp. 521-533.
- Vijayalakshmi, R; Nadarajan, R; Nirmala, P; Thilaga, M; (2011) Performance Monitoring of Large Communication Networks using Maximum Common Subgraphs. *International Journal of Artificial Intelligence*, Volume 6, Number S11, pp. 72 - 86.

A Contribution summary

This appendix contains a brief summary of our contributions to the written report, and the program made during the project.

Daniel Eddeland

- Report Sections: The triplet surface and the triominoes program, Purpose, Scope, Theoretic Background chapter, Choice of software platform, Representation of proteins, PDB format, Internal graph format for describing triplet surfaces, Construction of graphs from triplet surfaces, Triangle graphs, Maximum Common subgraph of two graphs, Maximum common subgraph algorithm pseudocode. Formatting of report in LaTeX.
- Programming: structuring of classes, parsing of PDB files, creation of triplet surface models in program, 3d rendering of proteins, graph and algorithm interfaces, McGregor algorithm and data structures.

Daniel Kvarfordt

- Report Sections: The triplet surface format and the Triominoes program, Triangle graphs, Details of triangle graph generation, Planar graphs, Graphical user interface, Limitations of the atom triplet surface format, Shape comparison of matches
- Programming: Triangle graph generation, triangle label generation preserving handedness, graphics programming, triangle mouse selection (mouse picking), hover text overlay and markers for atoms and triangles, 3d model user interaction (camera relative rotation), isomorphism checker, refactoring and documentation, logging, GUI work.

Hjalmar Lindskog

- Report Sections: Scope, Maximum Common Subgraph problem, Subgraph isomorphism problem, Usefulness of program, Future Work, Concluding remarks
- Programming of: Algorithms in particular Subgraph isomorphism algorithm

Joakim Sellin

- Report Sections: Purpose, Previous work, Scope, Future work, Concluding remarks.
- Programming of: Program structure, algorithmic parts.

Christofer Warg

- Report Sections: Introduction, Scope, Purpose, Literature Review, Development process, Choice of algorithm, Acknowledgement, Abstract, Sammandrag, Future work, Glossary, References, Problems encountered, Subgraph Isomorphism algorithm, Choice of software platform, Final program functionality and limitations, Concluding remarks.
- Programming of subpatch search algorithm, interface details, datastructures, miscellaneous.

Mikael Weckstén

- Report Sections Introduction, Previous work, Scope, Glossary, Manual, Discussion, Usefulness of program, Testing, Implementation, Theoretic background
- Programming of Interaction between Triminoes (C) and our program (Java), GUI, Fetching and presenting of color-information, Subpatch search algorithm, wide scale deployment
- Special areas and undertakings: group leader

B Program Manual

This appendix contains instructions on how to run the program created in our project.

How to load proteins

Press the "Browse..." button (see A in Fig. B.1) and choose what protein you want to load in the corresponding window. You can now also chose the probe size that is to be used when creating the protein from the PDB file. Simply change the value (see B in Fig. B.1) and click the "Change" button (see C in Fig. B.1). Once you have chosen a protein and changed the probe size to your liking click the "Reload" button (see D in Fig. B.1). This will load the chosen protein into the corresponding window.

How to move the protein

Put the cursor in the window which protein you want to move (see E in Fig. B.1). Hold the left mouse button and drag the cursor to move the protein. If you want to zoom in or out, hover over the protein with the cursor and use the scroll wheel. You can manipulate both windows at the same time by checking the "Lock views" option (see F in Fig. B.1). To reset the views to their original state you can press the "Reset views" button (see G in Fig. B.1). This will only reset the position of the proteins and not reset, for example, the results if they are showing.

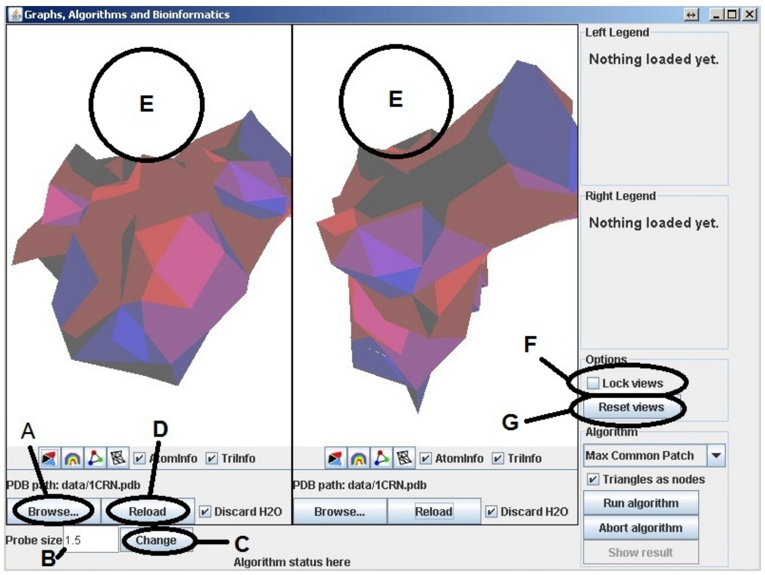


Figure B.1: Main window

How to select triangles

Move the cursor over the triangle (see A in Fig. B.2) you want to select. Press the left mouse button to select it. Press the left mouse button on a selected triangle (see B in Fig. B.2) to deselect it. You will notice that the selected triangles appears in the corresponding Legend (see C in Fig. B.2) with the name of the atoms that corresponds to those triangles and the color they have. If several identical triangles have been selected it will still only show one entry.

How to run an algorithm

Choose an algorithm in the drop down menu in the Algorithm window (see D in Fig. B.2) and toggle if you want to match atoms or triangles by clicking the "Triangles as nodes" check box (see E in Fig. B.2). Triangles are used by default and is recommended.

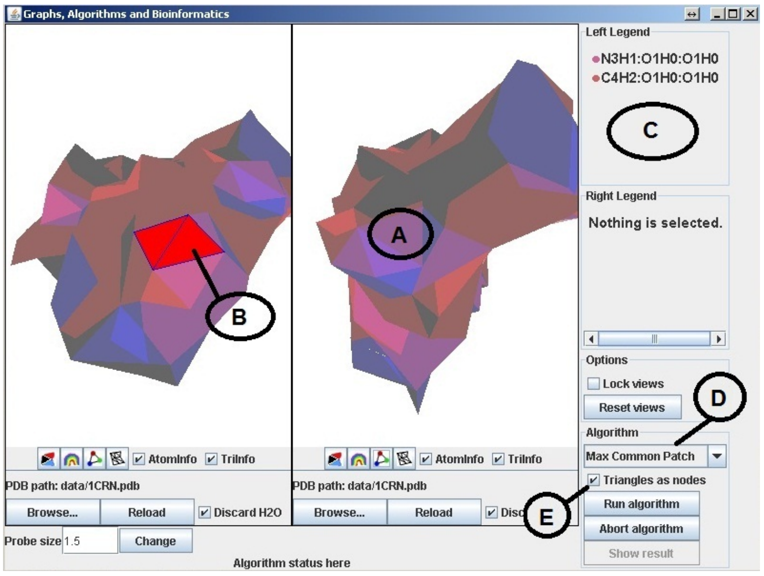


Figure B.2: Main window with selected triangles

In the drop down menu you will have two choices:

- Max Common Patch - Finds the largest common subgraph between the two proteins. It does not need any triangles to be selected and will promptly ignore if there is any selection made prior to the running of this algorithm.
- Subpatch Search - Find the selected subgraph in the other protein. If both proteins have selections in them, the left one will be used. Please do notice that if you try to run this algorithm without any selection at all there will be a message at the bottom telling you that you have made no selection yet.

Make sure you have loaded two proteins in before trying to run any of the algorithms. To run the algorithm press the "Run Algorithm" button (see A in Fig. B.3). You can see if the algorithm is running in the bottom of the screen (see B in Fig. B.3), this is the small information box that are always displayed. There are two choices while the algorithm is running:

Abort the algorithm: Press the "Abort algorithm" button (see C in Fig. B.3). This will abort the current running algorithm and present the current best result graphically if any. Let the algorithm finish: The best result will be graphically presented.

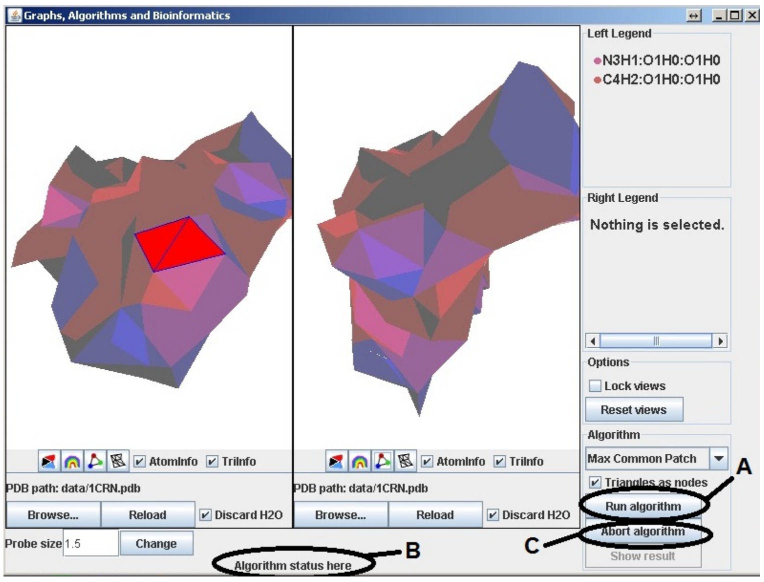


Figure B.3: Main window with selected triangles

Once the algorithm have finished you will see that the proteins have changed color. The triangles change to one of the following colors:

- Green (see A in Fig. B.4): No match were found for this triangle
- Shade of gray/black (see B in Fig. B.4): Match found
- Red (see C in Fig. B.4): Match found and the starting point of the algorithm (if you run the Subpatch search with only one triangle selected the result will only be one red triangle since it will not expand beyond the starting point)

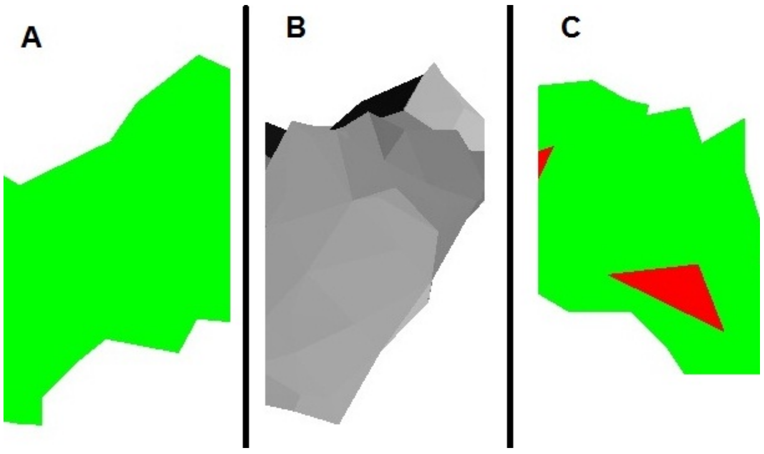
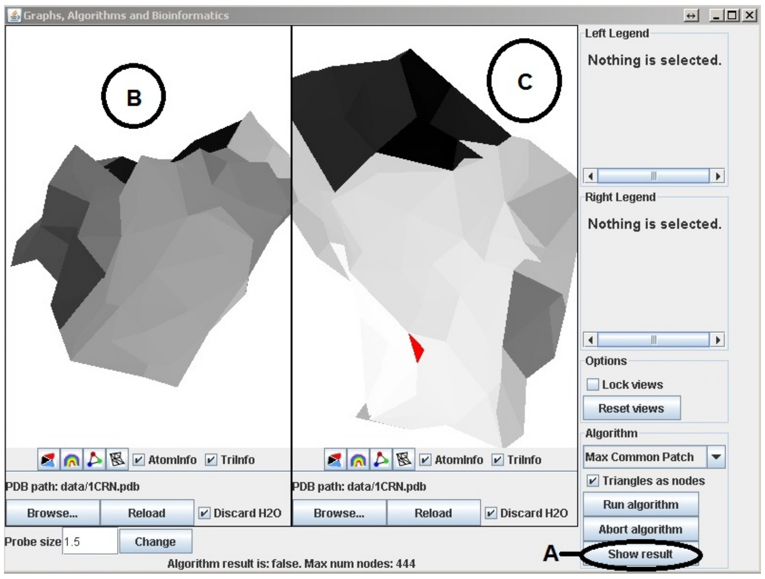


Figure B.4: The three different result colors

You can now press the “Show result” button (see A in Fig. B.5) and you will be presented with the log-file (see Fig. B.6). The left column will contain the found matches references for the PDB-file that was loaded into the left window (see B in Fig. B.5) while the right column will contain the found matches references for the PDB-file that was loaded into the right window (see C in Fig. B.5).



Nothing is selected.

Nothing is selected.

PDB path: data/1CRN.pdb

PDB path: data/1CRN.pdb

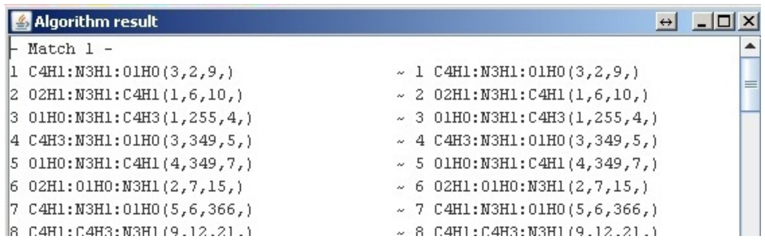
Probe size 1.5

Max Common Patch

✓ Triangles as nodes

Algorithm result is: false. Max num nodes: 444

Figure B.5: Main window with 100 % match between the proteins



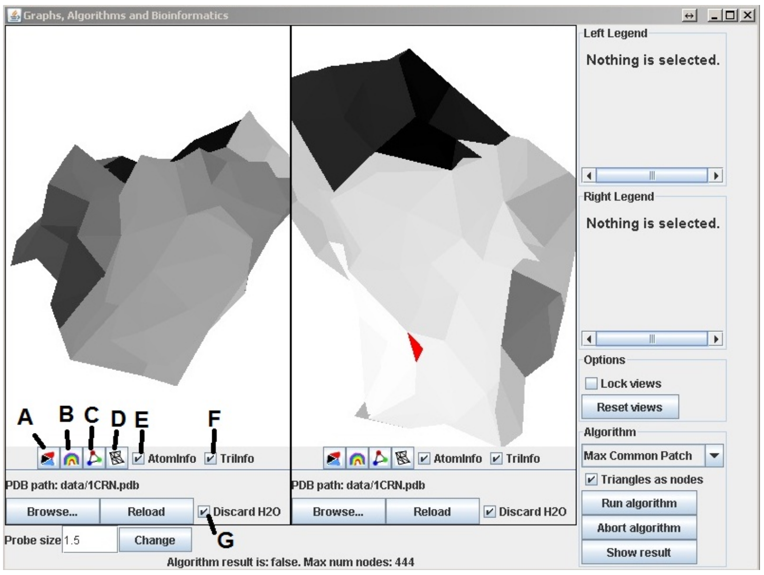
```
├ Match 1 -
1 C4H1:N3H1:O1H0(3,2,9,)                 ~ 1 C4H1:N3H1:O1H0(3,2,9,)
2 O2H1:N3H1:C4H1(1,6,10,)                ~ 2 O2H1:N3H1:C4H1(1,6,10,)
3 O1H0:N3H1:C4H3(1,255,4,)               ~ 3 O1H0:N3H1:C4H3(1,255,4,)
4 C4H3:N3H1:O1H0(3,349,5,)               ~ 4 C4H3:N3H1:O1H0(3,349,5,)
5 O1H0:N3H1:C4H1(4,349,7,)               ~ 5 O1H0:N3H1:C4H1(4,349,7,)
6 O2H1:O1H0:N3H1(2,7,15,)                ~ 6 O2H1:O1H0:N3H1(2,7,15,)
7 C4H1:N3H1:O1H0(5,6,366,)               ~ 7 C4H1:N3H1:O1H0(5,6,366,)
8 C4H1:C4H3:N3H1(9,12,21,)               ~ 8 C4H1:C4H3:N3H1(9,12,21,)
```

Figure B.6: Results window

Extra features

- Few colors (see A in Fig. B.7): The protein will be colored with few colors to differentiate between the triangles. This feature is on by default, thus if you have not changed anything yet pressing this button will appear to do nothing.
- Many colors (see B in Fig. B.7): The protein will be colored with many colors to differentiate between the triangles.
- Show atoms (see C in Fig. B.7): The atoms in the protein will be represented with spheres. Pressing this button again will toggle this feature on/off. It is by default turned off.
- Show wirefarm (see D in Fig. B.7): Shows the wireframe for the protein, these are also the edges between the nodes. Pressing this button again will toggle this feature on/off. It is by default turned off.
- AtomInfo (see E in Fig. B.7): When hovering over a atom of the protein their information is shown in the form off Number in PDB file, Atomname and Residue name. Pressing this checkbox again will toggle this feature on/off. It is by default turned on.
- TriInfo (see F in Fig. B.7): When hovering over a triangle in the protein its information is shown in the form off Triangle number, The three atoms making the triangle. Pressing this checkbox again will toggle this feature on/off. It is by default turned off.

- Discard H2O (see G in Fig. B.7): Discards all water atoms (H2O) when loading the protein. Pressing this checkbox again will toggle this feature on/off. It is by default turned on since water are often not of interest.



PDB path: data/1CRN.pdb

Probe size 1.5

Algorithm result is: false. Max num nodes: 444

Figure B.7: Main window

C Maximum Common Graph Algorithm Pseudocode

In this appendix you find pseudocode describing the main parts of our modified version of the McGregor Algorithm.

Variables used in the description

- G, H: graphs for which the maximum common subgraph is searched.
- mappedNodes: a map containing the nodes in the current solution
- bestMap: a map storing the best solution
- visitedList: a list of nodes in G which are already mapped or marked as absent from the solution.

pickUCandidate() function

returns a node u from G, such that:

- u is not in mappedNodes, and not in visitedList
- u is adjacent to a node in mappedNodes, or mappedNodes is empty
- if no such node u exists, return null.

pickVCandidate(u) function

returns a node v from H, such that

- u and v has the same label
- v is not in mappedNodes
- mappedNodes is empty, or mappedNodes contains a pair (u',v'), such that:
  - u' is a neighbour of u
  - v' is a neighbour of v
  - the labels between (u,u') and between (v,v') have the same edge labels
- if no such node v exists, return null

seek(i) function

recursive function which does the actual search. i is the recursion level (algorithm starts at 0)

Pseudocode for seek:

```
u = pickUCandidate()
If (u == null) then:
    If mappedNodes.size > bestMap.Size
        bestMap = mappedNodes
    return
Else:
    visitedList.add(u)
    v = pickVCandidate(u)
    while (v != null):
        mappedNodes.put(u,v)
        store the state for this i
        seek(i+1)
        restore the state for this i
        v = pickVCandidate(u)
    store the state for this i
    seek(i+1)
    restore the state for this i
    visitedList.remove(u)
```

D Test computer specification

Specifications for the computer used for performance tests:

- Type: Notebook (Acer AO722)
- OS: Windows 7 Home Premium Service Pack 1
- Processor: AMD C-60 APU 1.00 GHz (64-bit processor)
- Installed memory: 4,00 GB (3.73 GB usable)
- Note: Plugged in, not running on batteries