Computer-aided Construction of Padlock Probes

by

Christoffer Hamilton
Greger Hälltorp
Ellen Kindlund
Otasowie Osifo

Internrapport nr. 2001:2

UPPSALA UNIVERSITET
Inst. för informationsteknologi
Avd. för teknisk databehandling

UPPSALA UNIVERSITY
Information Technology
Dept. of Scientific Computing

Abstract

With the event of the mapping of the human genome, genetic information has been made available on a scale unprecedented in the history of molecular biology. To make use of this information new tools for genetic analysis have to be invented, such as new types of probes for investigating sequence variations in single copy genes or copy numbers of specific sequences. One such probe is the padlock probe, which possesses several excellent characteristics. In this report we describe the planning and implementation of a program to be used for automatic design of padlock probes.

The program takes as its input a file containing information on the target sequence and the conditions that will be used during the experiments,its output is a proper padlock probe.

As the requirements of the probes are likely to change with the development of e.g. new detection methods, we have made an effort to ease future changes to the program. The general design of the program is modular,making it easy to add new functions or remove obsolete ones.

Contents

1 Background

When performing genetic analysis one of the major problems is distinguishing the signal from the background, one example is the analysis of single nucleotide variations: In the human genome one nucleotide per 200-1000 differ between individuals.The sites of these differences are called *Single Nucleotide Polymorphisms* (SNPs or snips).

In most cases these variations have no effect on the phenotype but many genetic diseases in humans are believed to be caused by such variations at single sites. The investigation of these diseases requires tools that are capable of pinpointing single nucleotide variations in a genome of 3 billion nucleotides. For these purposes different types of probes are used.

Besides the requirements for specificity and sensitivity, there is often a need to perform multiple analyses at one time, which requires the usage of several probes at once. When performing such experiments there is always a risk of cross-reactions that ligate different probes to each other.

The current methods of choice for genetic analysis are PCR and regular hybridisation reactions. The PCR method is highly specific and sensitive but cross-reactions that occur are not easily discerned.Hybridisation reactions on the other hand are usually limited in both specificity and sensitivity.[1, 2, 3]

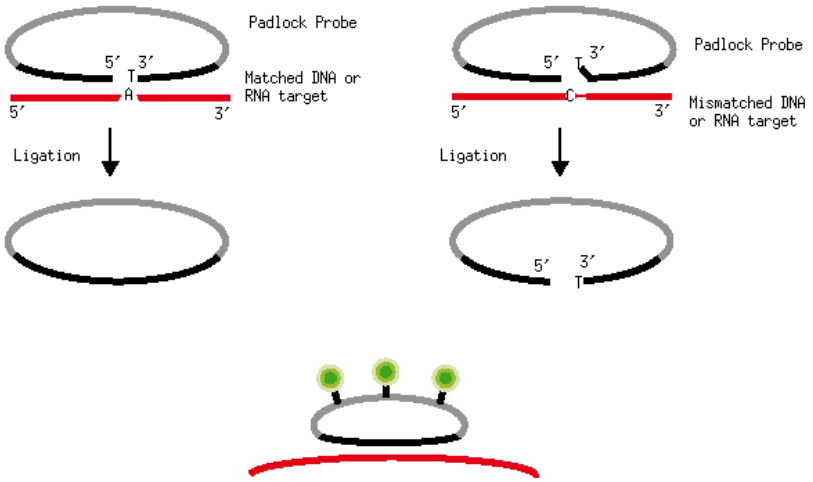

Figure 1: A Padlock probe "at work".

The "Padlock" probe combines the advantages of PCR amplification and DNA hybridisation. A padlock probe is an oligonucleotide consisting of end segments,complementary to a target sequence,connected by a linker sequence. The end segments are designed so as to hybridise with a DNA or RNA target sequence in such a way that when DNA ligation is performed

(joining the ends), the probe will become circularised around the target. See Figure 1.

Hybridisation occurs between the probe and the target in such a way that the 3'- and 5'-ends of the probe meet at the target nucleotide of interest.In the probe,the nucleotide complementary to the target nucleotide is situated at the extreme 3'-end.The probe is locked in place by the use of a ligase that links the sugar moieties of the ends of the probe, creating a single circular DNA molecule that is wrapped around the target.

If the target nucleotide at the 3'-end is not complementary to the probe nucleotide, the resulting topology-distortion is enough to prevent the ligase from creating the phospho-diester bond,this lends the probe a very high specificity for variations in the nucleotide investigated.

The high suitability of padlock probes for analysis of multiple targets is achieved since cross-reactions in the ligation step will not affect signal detection.Such events give rise to linear molecules that are easily distinguished from the circularised probes arising from "correct" reactions.

After finishing the reactions a washing step is needed to remove e.g. unreacted probes. Circularisation of the probe around the target creates a situation where a correctly localised probe will not be washed away from the target even if the washing is performed at temperatures above the melting temperature of the probe-target hybrids, a fact that might be very important since it allows more rigorous washing.

A further advantage of the padlock probe is the possibility to enable amplification and detection of the signal by equipping the linker sequence with useful sequences, e.g. primer sequences for rolling circle replication (RCR) mechanisms and zip codes for use in micro-array assays.[2]

2 Planning the work

In the initial, planning stage of the process we had several meetings with the group to analyse the problem and to decide on how to design the program. We also had to do some research on some of the subproblems, such as alignment algorithms and melting temperature calculations. Once the general outline and structure of the program had been decided upon we divided the task of implementing the program between us. During the implementation stage we often met to discuss problems that had arisen and to discuss our progress.

During the entire project we also had meetings with our supervisors, approximately once a week. These meetings provided us with feedback and gave the supervisors a possibility to monitor our progress. The supervisors also invited Ph.D students and researchers that had been working with padlock probes previously. These meetings provided us with more specific information about the project and their ideas on how the program should

work.

The coordinator of the course, Anders Sjöberg, also had regular meetings with us to enable him to monitor the progress of our work. In the beginning of December, there was a mid-course presentation. This enabled the two groups participating in the course to present their project to the other group and to give them an idea of how the work was proceeding.

3 Materials and Methods

We wrote our program with the assumption that it will be compiled with a C++ compiler that has access to the standard template library (STL). The STL is available with recent C++ compilers. Should the STL not be present, the code would have to be modified to use a built in array instead of the vector class in the template library. The part of the code where the STL string class has been used, would have to be rewritten with char-arrays.

We have not tried to optimise our code during compilation. We compiled the program with the GNU C++ compiler (version 2.95.2) on Sun UltraSPARC-IIi running the Solaris operating system.

4 Manual Probe Design

The process of manually designing a padlock probe is a time-consuming process consisting of, at least, three steps:

1. The end segments must be determined. They must be complementary to the target sequence with the nucleotide of interest at the 3'-end and their lengths must be adjusted to assure that their melting temperatures do not differ too much.

2. The primers and zip code(s) must be inserted in the linker sequence.If multiple targets are to be analysed simultaneously, the probes must be adjusted so that they do not interfere with one another. The probes must have different zip codes.

3. The probe must be checked for intra-molecular complementarity to assure that it will not form any stable secondary or tertiary structures through internal base-pairing. This is an important step since any intra-molecular complementarities that are strong enough will prevent the probe from functioning properly. Intra-molecular hybridisations have a much higher probability of occurring than do inter-molecular hybridisations. If intra-molecular complementarity is discovered and it is judged to be of relevance, the designer will have to go back to step 2.

Though the process is time-consuming if it is performed manually, these steps are all readily performed by a computer. Having a computer perform the design process would also make it easier to keep track of which zip codes that have been used with the use of a simple database. This would in turn make it easier to produce several different probes for use in the same experiment. Thus the whole problem of designing a padlock probe is well suited for a computer-based solution.

5 Program Design

The following list mentions some of the main issues that we found when considering the general design of the computer program:

- How to input data to the program.
- What alignment-algorithms to use when searching for intra-molecular complementarity and how these should be adjusted to account for the fact that one does not search for matching nucleotides but for complementary.
- What estimates to use when computing the melting temperature of hybridised regions.
- How to keep track of zip codes that have already been used.
- How to handle errors that may arise.
- When and how to change segments of the probe that gave rise to encountered errors.

When complete the probe output by the program should have the following components:

- The end segments, complementary to the target sequence, where the last nucleotide (the 3'-end) is the nucleotide of interest.
- Two primer sequences
- One zip code

Our conclusion was that the easiest path would be to design a program that performs essentially the same process as when designing probes manually but with certain optimisations to reduce time-usage.

The first problem to be tackled was which programming language to use. Different alternatives were considered, such as Java, Perl and C++ and we finally decided on C++ since it would produce the fastest program and since our supervisors felt that they would be able to support us the most if we wrote the program in a language they mastered.

The next step was to decide how to divide the problem into smaller parts and, on a more technical level, what objects we would need. This process led us to a general structure that we could use as a starting point in the implementation of the program. After having finished the structure we began the implementation process.

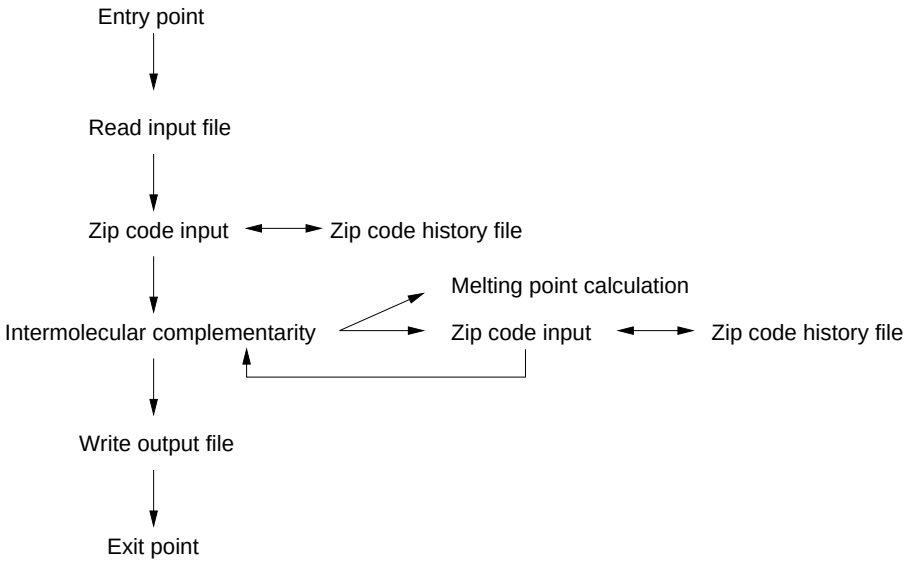

Figure 2: Program structure of MAKEPROBE

5.1 Modularity

The program follows a one-way path (see Figure 2). At first a parameter file is read to retrieve the data from which the probe shall be constructed. This data includes details such as the sodium concentration as well as strings of DNA or RNA that form the basis for the probe.

After the initial parameters have been read, they are checked and a preliminary probe is constructed. This probe is then input to the modules in order. Each module's functions use the probe as input and modify the probe, e.g. by adding a new sequence. The modules might also discard the probe entirely or add comments to it.

6 Program Implementation

The program needs numerous input parameters to complete the construction of a probe, a few of these are:

- The target nucleotide sequence
- The position within that sequence of the nucleotide of interest

- The sodium concentration in the mixture (affects the melting temperature of hybridised nucleic acid)
- The temperature during the ligation reaction
- Whether the target sequence is DNA or RNA
- The two primers to be used
- Name of file containing the zip code database

Since there are so many important input parameters we came to the conclusion that the most convenient way of making these available to the program would be to place them in a file in a directory and to let the user type the name of this directory as a parameter when starting the program.

The zip codes pose an entire problem of their own; there are many zip codes available and the user might want to use a particular one. One of the main points of the program is however that it checks for intra-molecular complementarities and if it finds any that involve the zip code it must be changed. Also it should be possible to produce several probes to be used at the same time, requiring a different zip code for each of the probes. Thus the last of the input parameters mentioned above is a file containing zip codes.

6.1 Error handling

We realized early on that probe design was no clear cut path. A probe cannot be defined as strictly either valid or invalid. Some not entirely functional probes should despite their faults be constructed and accepted by the program but with a comment. We therefore needed a structured way of adding comments to the probe.

The solution we chose was to keep a list of comments to each probe. All warnings or error messages that are connected to the probe are added to this list. As all the modules use this error treatment, a consistent way of handling errors has been supplied. This will not only work with our modules, but also with other additional modules yet to be developed.

6.2 Future steps in the development process

Although we have tested the program with several probes, both valid and invalid, one should probably continue with a larger test set to ensure the correct functionality of the program. Thereafter the blast search part could be integrated into the program. This module is entirely independent of the others and involves checking the probe sequence for similar sequences in the human genome.

The melting point calculation does not use nearest-neighbour calculations due to lack of parameters. These parameters need to be found.

Currently the program only constructs one probe, and a relatively simple extension would be to rewrite the program to enable construction of multiple probes.

7 Basic Classes

One of our main efforts was to construct a suitable framework for the construction of a probe. In the beginning we identified the parts of a probe. We appreciated the view that a probe consists of a collection of ordered sequences, where each sequence usually is 15-20 nucleotides long. We needed a simple way to manipulate these sequence pieces.

7.1 Class Sequence

The sequence class is used to describe a DNA or RNA primary structure. The main data member of the class is an array of nucleotides (a string). The length of each array is also a member of the class.

```
Class Sequence
----------------
string nucleotide_sequence;
int sequence_length;
string sequence_name;
----------------
Sequence()
Sequence(string nucleotide_seq, string sequence_name);
Sequence(const Sequence &seq)
~Sequence();
void seq_init(string nucleotide_seq, string seq_name)
char *toString() // Turns the sequence into a string
void add(char nucl); // Adds a nucleotide to the end of the sequence
string get_nucleotide(); // Returns the entire sequence
string get_nucleotide(int pos); // Returns 1 nucleotide on
position pos
string get_name();
void modify(int pos, char nucleotide); // Changes the nucleotide
on position pos
bool are_equal(int pos1, int pos2);
void concatenate(Sequence seq2); // Appends seq2 at the end
int get_length();
void set_name(string name);
Sequence &operator=(const Sequence &seq)
Sequence make_compl(int n); //make position complimentary in a sequence
Sequence make_reverse(); //reverse a sequence
Sequence seq_from_interval(int i, int j); //extract an interval
of a sequence
int are_complementary(int pos1, int pos2);
```

Functions in this class enable the programmer to check individual nucleotide for complementarity, to print the sequence and some other basic operations.

7.2 Class Probe

This class models the probes. A probe is built of sequences. The probe arms, the zip codes and other components of the probe are sequence objects. See Figure 3.

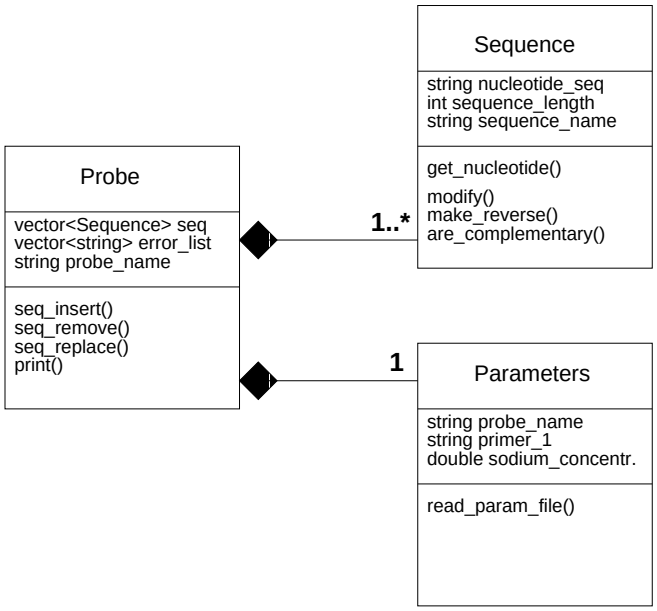

Figure 3: Unified Modelling Language (UML) Diagram of class Probe. A probe is built up by one or several Sequences objects and a Parameter object.

```
Class Probe
---------------
vector<Sequence> sequences; // A vector with 5' - seq ... seq -
3'
vector<string> error_list; // Error messages and warnings
relevant to the probe
string probe_name;
bool valid; // False if there's no point in proceeding with
            // calculations on this probe
Parameters probe_parameters;
---------------
Probe();
Probe(char* parameter_filename);
Probe(Sequence seq);
~Probe(); // Destruktor

void seq_insert(int pos, Sequence s);
void seq_remove(int pos);
void seq_replace(int pos, Sequence seq);
bool is_empty()
Sequence get(int pos);
Sequence make_one_sequence(); // Turn the probe into one long sequence
void print(char *filename = NULL); // Print the probe to file or stdout

void set_valid()
```

```
void set_invalid()
bool is_valid()
```

A probe is built by any number of Sequence objects.

In the Probe class there are functions to handle the order of the individual sequences. There is also the possibility to turn the entire probe into a single, long sequence. This last feature means that the programmer can, if he so wishes, use the Sequence functions on the entire probe.

7.3 Class Parameters

This class is used to read data from a file. Each probe contains one Parameters class, which it uses to get input parameters. See Figure 3

The variables are all public.

```
Class Parameters
----------------
string probe_name;
string target_sequence;
string primer_1;
string primer_2;
int snp; // index into target sequence (also length of 3'-part)
double ligation_temperature;
double sodium_concentration;
int min_arm_length;
int min_bind_nucleotides;
char *organism_filename; // fil för blastsearch
char *zipcode_filename;
char *zipcode_additional_info_filename;
int dna_or_rna; // 0 = DNA, 1 = RNA
char *output_file;
----------------
Parameters(); // konstruktor
~Parameters(); // destruktor
int read_param_file; // Must be called to read data
```

7.4 Class Matrix

In the alignment search a matrix of integers is used. The matrix class simplifies dealing with the matrix structure. The parameters of the class are an array of arrays of integers; a matrix of integers and the size of the matrix.

```
Class Matrix
----------------
int **the_matrix;
int size;
----------------
Matrix();
Matrix(int s);
~Matrix();
void print();
int& operator() (int column, int row);
```

A matrix is created while given a size and then the values are set and fetched using the operator.

7.5 Class Binterval

The Binterval structure is an array that holds the indices of the intervals of alignments in the alignment search. There are also slots in the array that keep track of length of diagonals, if you are in a diagonal and the number of mismatches in that diagonal.

The parameters in the Binterval class are integers in an array.

```
Class Binterval
----------------
int interval[14];
----------------
Binterval()
Binterval(int column, int row);
~Binterval()
const Binterval& Binterval::operator= (const Binterval& B);
void print();
```

The Binterval is handled via the operator, which allows you to get values, change them and to copy an interval.

8 File I/O module

In this description, we use *history file* to refer to the file containing the zip codes that have been used at least once for constructing a probe, while we use *zip code file* to refer to the file from which we can search and choose a zip code. In our implementation, the history file and the zip code file are assumed to be in either of two formats which are referred to as the fixed format and the table format.

8.1 The Interface

The interface to the implementation of the methods is a header file called *FileIO.h*. The file contains functions for reading from and writing to the zip code and history files. The file also contains definitions of some necessary constants:

```
const int ERROR = -1
const int OK = 0
const int MAX_LINE_LENGTH = 256
------------------------------
bool file_in_fixed_format(const char *filename)
bool file_contains_data(const char *filename)
int choose_sequence(Sequence &, const char *, const char *history_file = NULL)
int fixed_format(Sequence &, const char *data_file, const char *history_file)
int table_format(Sequence&, const char *data_file, const char *history_file)
bool is_in_file(string seq, const char *history_file)
int write_to_file(const string , const string sequence, const char *filename)
```

8.2 The Constants

The constant ERROR is an integer that is returned if something should go wrong with the processing of either the zip code file or the history file

while OK is an integer constant that is returned if there are no errors. MAX_LENGTH is used to define the constant size of a buffer which in this case is a character array.

8.3 The Functions

The main function is *choose_sequence()*. This function takes the name of the history file, the name of the zip code file and a reference to a sequence object as input arguments. The flow of execution of *choose_sequence()* is shown in Figure 4. On entry into the function, it opens the zip code file for reading and then checks this file to make sure that it contains data for reading. If the file does not contain data then an error constant is returned. If the zip code file contains data, its format is checked. If the zip code file is in the fixed format then the function *fixed_format()* is called to read the file and to create a zip code sequence from the data read. Otherwise the function *another_format()* is called to perform these services.

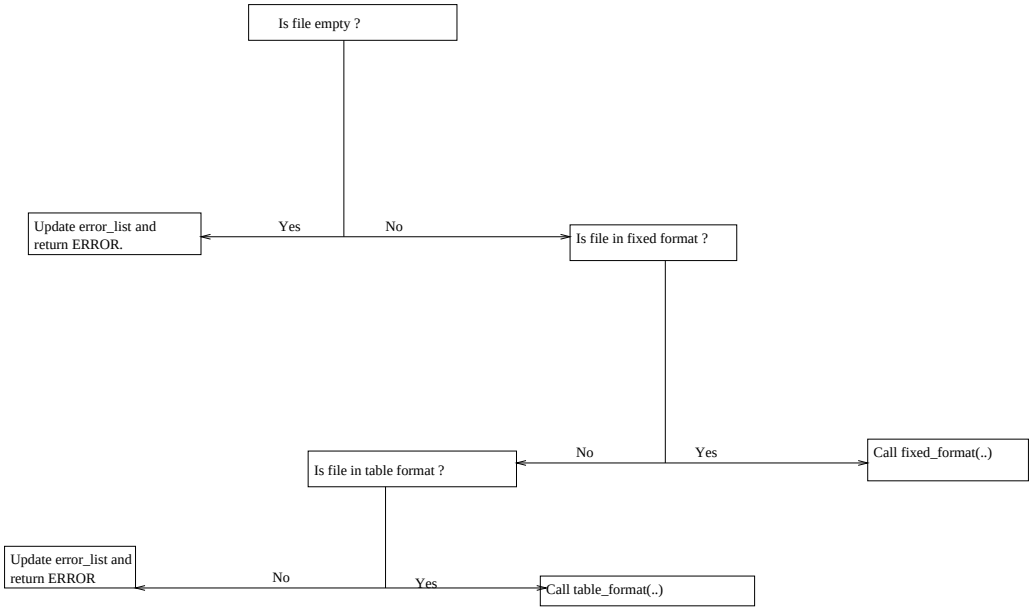

Figure 4: Flow of execution of *choose_sequence()*

file_in_fixed_format() is used to determine if the input zip code file or history file is in the fixed format. The input argument to this function is the name of the file to be checked. *file_contains_data()* is used to determine if a file is empty or not. It returns a boolean variable. The input argument to this function is the name of the file to be checked.

fixed_format() format begins its execution by checking whether the file was given as an input and by checking that the file is not empty. It then opens the zip code file in read mode. A sequence is chosen from the file. If the

history file was given as input, it is checked to see if the chosen sequence has already been used. The flow of execution in this function is as shown below. If the sequence is not in the history file, it is returned for use in constructing a probe. If the sequence is found to be in the history file, another sequence is chosen from the zip code file. This process is repeated until a previously unused sequence is obtained. See Figure 5.

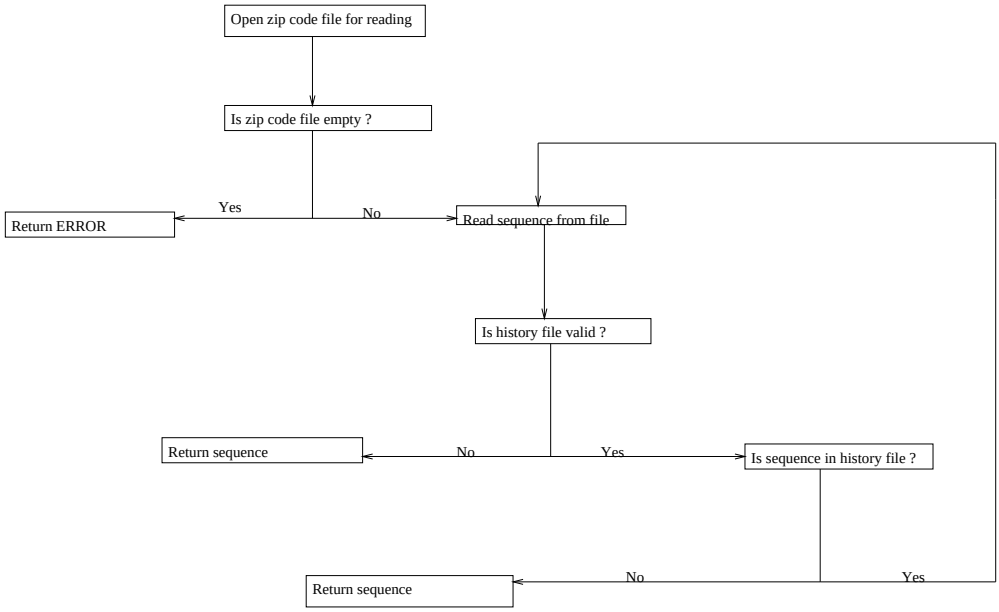

Figure 5: Flow of execution of *fixed_format()*

is_in_file() is the function used to check if a given sequence is in the history file. The function compares the given sequence against all the sequence objects in the history file for a match.

table_format() is used to read a sequence from a file that is not in the fixed format. It's flow of execution is the same as that of *fixed_format()*.

Once a sequence is chosen, it is checked for validity with respect to intramolecular complementarity. If the chosen sequence is found to be valid, it is used to construct a probe. Otherwise the reading process is repeated until a valid sequence is found from the zip code file or it is found that all the sequence objects in the zip code file are in the history file. Once a sequence has been chosen and used to construct a probe, the sequence is appended to the history file. The function that does this is *write_to_file()*. The inputs to this function include information about the sequence to be written, the sequence itself and the name of the history file.

9 The intramolecular complementarity module

The program uses the standard alignment algorithm by Smith-Waterman[4] adjusted to the search for complementary regions instead of regions of similarity. The Smith-Waterman algorithm is a modified version of the Needleman-Wunch algorithm. The modifications from the Needleman-Wunch algorithm switches the focus from searching for the best global alignment to finding the best local fit.

When working with intra-molecular complementarities this is more interesting since even shorter regions of complementarity could compromise the function of the probe by preventing the probe from hybridising with the target.

To find out if the probe has parts complementary to itself as seen in Figure 6, an intramolecular complementarity search is performed. It investigates the new sequence put into the probe, here the zip code and the 3' end of the probe in particular.

The new sequence is exchangeable and therefore the alignments it causes might be unnecessary and avoidable. The function returns an integer that prompts the program to try a new zip code, if necessary. This way a zip code that works fine with the probe is chosen.

The 3' end of the probe is the most important part when it comes to the functionality of the probe. The binding, if ever so small, of the 3' end of the probe drastically decreases its ability to bind to the target sequence. The function issues a warning if the 3' end of the probe is complementary to another part in the probe, but no action is taken unless the 3' end is complementary to an exchangeable sequence in the probe. In that case the function proceeds as above.

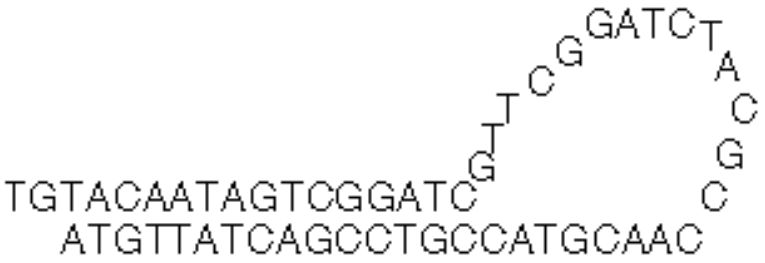

Figure 6: The probe can hybridize to itself

There is also a possibility of one base to be involved in the alignment twice. This happens when two probes bind to each other. When this happens

a warning is attached to the probe, that the alignment found isn't a proper intramolecular alignment.

The Smith-Waterman algorithm is a dynamic programming algorithm, where you create a matrix of score values, with the sequence position of each searched sequence on each axis. Every slot in the matrix is calculated using the results from the previous ones going from left to right, top to bottom in the matrix. The scores are set as 1 for a match, -1 for a mismatch and a gap-penalty of -1 for each base in the gap. This high gap-penalty leads to long diagonal sequences with mismatches instead of more matching bases with gaps in between. This is something we take advantage of later.

As we sum up our dot matrix with the one and minus ones, we also create a trace matrix where we represent the step we have taken to get to a specific slot. A diagonal step is described by a zero, a vertical step with a positive value and a horizontal step with a negative value in the trace matrix. These two matrices are then used in the finding of the best local alignment.

The alignment starts at the highest score in the score matrix. From there, via the trace matrix, the alignment continues until a zero in the score matrix is encountered. The indices of the alignment are stored in a structure called Binterval. There is other information stored in the Binterval, such as when there is a diagonal alignment within the alignment we search, that is, a stretch of aligned bases with no gaps. Also, the indices of the longest diagonal found is stored there.

When the search is completed a melting temperature check is performed on the longest diagonal of the alignment. The temperature check is done only on the diagonal instead of on the entire alignment due to the nature of the function that performs the melting temperature calculation. It does not consider gaps in the sequences. This is the reason of the bulky trace for the alignment where great effort is put in to finding the best-aligned diagonal and the large gap-penalties.

Finally, that temperature is compared to the temperature in which the assay is carried out, the ligation temperature, and if larger, a warning is attached to the probe. If the alignment lies within the zip code, a new zip code is chosen, and the whole process is repeated, otherwise no other action is taken.

The ligation temperature, sodium concentration of the assay,which is needed in the melting temperature calculation, whether the sequence is a stretch of DNA or RNA, and the smallest number of bases needed for an alignment to be an alignment are all given from the user via the input file.

9.1 The Functions

```
int check_intramol_compl(Probe &the_probe, int index);
int sum_matrix(Matrix scoreMatrix, Matrix traceMatrix, Matrix visitedMatrix,
       int column, int row);
Binterval trace_back(Matrix scoreMatrix, Matrix traceMatrix,
```

```
     Binterval the_trace, int last_column,
     int last_row);
int try_new_seq(Probe probe, Binterval in, int index);
double melt_point(char *sequence, char *compl_sequence, int DNA_or_RNA,
  double sodium_concentration, vector<string> &error_list);
int check_if_overlap(Probe probe, Binterval trace);
```

The function *check_intramol_compl()* is the main function in the intramolecular complementarity search. It calls the other functions and keeps track of the relevance of the returned values.

First matrices and Bintervals are created. Then dot matrices of matches and mismatches are formed. These matrices are then converted into sum matrices and trace matrices using the *sum_matrix()* function. The *trace_back()* function uses these matrices to create the Bintervals that hold the intervals of alignments.

The *check_if_overlap()* function checks if indices in the intervals appear twice, that is if the same base is involved in the alignment twice. That would mean that the alignment is actually made up of two probes binding to each other.

The *check_intramol_compl()* function calls on the function *melt_point()* from the Melting module to calculate the melting temperature of the alignment. If this temperature compared to the ligation temperature of the probe,the *try_new_seq()* functions is called. This function checks whether the alignment is in the new sequence (read zip code). If it is, the *check_intramol_compl()* returns one, otherwise zero.

10 Melting Point Calculation Module

The melting temperature of hybridised regions is calculated using an approximative method. This melting temperature is compared with the temperature used during the ligation reaction. If the melting temperature is too high the probe will not function properly, if this problem involves the zip code,the zip code simply exchanged for another, but if the problem involves other parts of the probe,the probe may have to be discarded. The program we are using to compute the melting point comes under the GNU/GPL licensing domain.[5] The program, called MELTING, can calculate the melting temperature in two different ways, either through nearest-neighbor computation or through a more simple approximation. The nearest-neighbor computation yields the best results, but unfortunately not all nearest-neighbor parameters were supplied with the program, so we were forced to always use the simpler approximation. The MELTING program was originally written in C, and we rewrote it to incorporate our error handling and to adapt it to the structure of our own program.

11 Directory structure

Each set of probes that shall be used together is stored in one single directory. This set of probes is called a project. In the project directory is the file that specifies the probe and it has the same name as the directory. Any other project specific information is also stored in the directory. For instance a history file for used zip codes. The history file stores the zip codes used by a probe to ensure that they do not reappear in another probe in the same project. Normally, a history file is project specific, although it may also be defined as common for several projects.

Four directories contain common files supplied in the makeprobe distribution. The directories are the **settings** directory for shared configuration options, the **doc** directory with documentation, the **src** directory with source files and the **bin** directory with binary executables. See Figure 7.

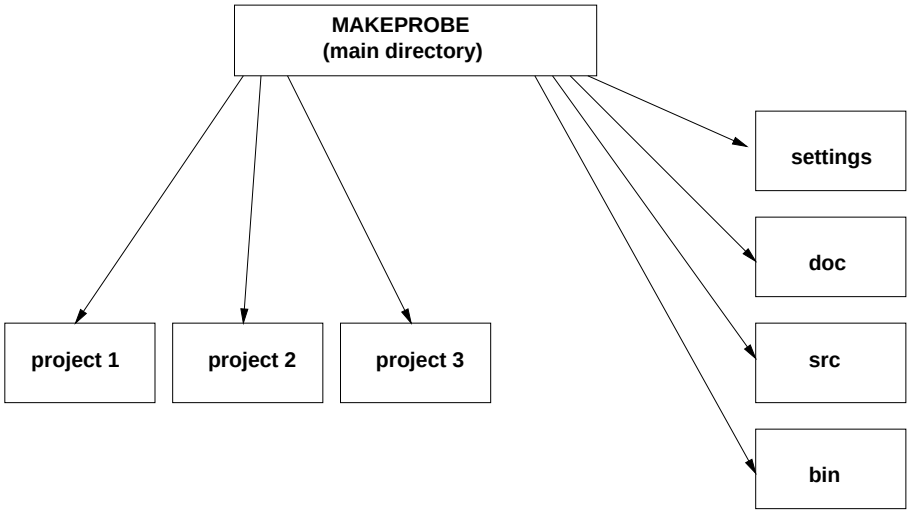

Figure 7: Directory structure of MAKEPROBE

12 Results

The finished program looks a lot different from the first draft of it. Many features were taken out of the program at the first meeting with the future users of the program and labelled future adjustments or simply not useful to them. Other parts have been removed or changed for other reasons. One part was taken out of our project and given to another student. Yet other parts have been considered too difficult or too time-consuming.

These changes from the drafts to the finished program have not effected (hopefully) the usability of the program nor the functionality. In fact, most of them should be considered improvements. The program as it looks today

is designed for the specific needs of the users. As these needs change, the program will have to change with them. There is a lot of space for these changes in the program, one can say it has been designed to be altered later. The modular structure of the program and the easily changeable main function are two examples of this.

The program has been tested on various target sequences, primers and zip codes. Especially the hand-designed probes already used in experiments have been tested with care. These probes should of course come out the same using our program.

Probe	*Melt_1(*C)*	*Comment*
A	11.3922	None
B	19.7789	None
C	15.894	None
D	5.9789	None
E	-1.18777 None	
F	-10.2878	None
G	29.9622	None
H	33.4456	None
3'	-1.08777	3' end binds. Abort probe!
zip	**42.1738**	Intramolecular complementarity
		The alignment is from two different probes.
		Change the new sequence
	36.3922	New zip code inserted
format2	-1.08777	None

Table 1: Results from test runs with sample probes. The first eight probes went through the makeprobe program without any comments. The '3" probe's 3' end hybridizes with another part of the probe. The 'zip' probe's zip code hybridizes with a melting temperature above some criteria (here 37 degrees) and was therefore exchanged. The 'format2' probe reads it's zip code using a different file format than the others.

The results of these test runs are given in Table 1. The probes given to us by the researchers are named A to H and have all been designed similarly in the program as when designing manually. This was the result we hoped for and were aiming at. The next probe, named 3', was a probe with input designed to give a binding including the 3' end of the probe. This probe came out with the comment "3' end of probe binds. Abort probe", which was exactly as expected. The second last probe tested was designed so that the zip code inserted should bind and therefore be replaced. This is done as the comment to the probe indicates. The very last probe was tested with a different file format for the zip codes.

As a conclusion one can say the results from every case were the desired ones. That doesn't necessary imply the program is free of faults. One can

reason about the small size of the testing set, and whether the program has been designed for these test probes, while all other probes would result in a failure of the program. One way to find out is to design a number of new probes manually and via the program and see if the results disagree in any case.

The faults this program might have are probably few. Lots of work remains to be done though. The intramolecular complementarity calculation is not entirely finished, mainly because of the melting temperature calculation not working the way we thought it would. The effects of this are difficult to predict. Of course, if the alignments found were longer, one could assume the melting temperature to be higher. But it's difficult to say to what extent that would effect the resulting probes or the tests made.

The makeprobe program is not ready for use in its current state. But it's a good basis and a good design for further development. This was the goal at the beginning of the project and has well been reached. Therefore we are satisfied with the final result of this project: the makeprobe program in its current state.

Acknowledgements

We would like to thank our tutors Erik Arner and Daniel Nilsson at the Department of Genetics and Pathology as well as the project coordinator Anders Sjöberg at the Department of Scientific Computing.

References

[1] D.-O. Antson, A. Isaksson, U. Landegren, and M. Nilsson. PCR-generated padlock probes detect single nucleotide variation in genomic DNA. *Nucleic Acids Research*, 28, 2000.

[2] Johan Banér, Mats Nilsson, Anders Isaksson, Maritha Mendel-Hartvig, Dan-Oscar Antson, and Ulf Landegren. More keys to padlock probes: Mechanisms for high throughput nucleic acid analysis. *Curr. Op. Biotech.*, Not yet published.

[3] Johan Banér, Mats Nilsson, Maritha Mendel-Hartvig, and Ulf Landegren. Signal amplification of padlock probes by rolling circle replication. *Nucleic Acids Research*, 26:5073–5078, 1998.

[4] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[5] Nicolas Le Novère,
www.pasteur.fr/recherche/unites/neubiomol/meltinghome.html
Last visited: Feb, 2, 2001

13 Popular description

The publication of the entire sequence of the human genome is imminent. The sequencing project has been on its way for several years but with the event of Celera entering the field, the speed has picked up enormously. Though a huge feat, the sequencing of the human genome is not the finishing project of human genomic disease. Once the sequence is known comes the project of trying to interpret the results and to find the genes that lie hidden in the sequence. This mapping project is perhaps an even more difficult one since it requires subtle interpretations and much knowledge about the construction of genes in eukaryotes in general and humans in particular.

Mapping the genome will require new tools as well as brilliance. One interesting feature of the human genome is its variation between individuals, approximately one base in every 200-1000 differs between two human individuals. These variations are called Single Nucleotide Polymorphisms (SNPs) and they do not, generally cause any deleterious effects. Some genetic diseases, however, are believed to be caused by variations at single sites in the genome. One known such case is the disease Sickle-cell anaemia, where a mutation in a single nucleotide changes one amino acid in the protein haemoglobin, the protein responsible for carrying oxygen from the lungs to other parts of the body in the red blood cells. This change of one amino acid totally distorts the structure of the protein rendering it much less capable of transporting oxygen. It is believed that other diseases may have similar causes. To investigate this matter a scientist would need a tool that is sensitive enough to find a variation at one nucleotide in the human genome, which consists of, in total, approximately 3 billion nucleotides. The tools that are used are called probes and today two types of probes are regularly used for this kind of analysis: PCR-probes and hybridisation probes. The hybridisation probes are generally not sensitive enough to allow for finding single nucleotide variations since one mismatch in the hybridisation is not enough to give selectivity. The PCR-probes, though sensitive and selective enough, have another drawback. When investigating i.e. the cause of a genetic disease it might be necessary to investigate several sites. The most convenient way of doing this would be to be able to apply more than one probe at once. This, however, gives rise to new complications since the reactions used in the experiment might lead to the putting together of two different probes. If PCR-probes are used these "cross-reaction" products are not easily distinguished from the correct products of the detection reactions.

A padlock probe is a new kind of probe that eliminates the drawbacks of the two probe types mentioned above. It is a molecule designed to hybridise to the target sequence in such a way that the 3'- and 5'-ends of the probe meet at the nucleotide that is being investigated. This nucleotide in the target sequence hybridises with the nucleotide at the extreme 3'-end of the probe. A ligase is then added, which connects the two ends of the probe

making it circular and wrapped around the target sequence. If there is a variation in the nucleotide of interest, there will be a mismatch and the two ends of the probe will not be perfectly aligned. This one mismatch is enough to prevent the ligase from being able to connect the probe ends. Once the ligation reaction is finished any remaining probes are washed away. The fact that the probe is wrapped around the target means that it will not come loose even if the washing temperature is above the melting-temperature of the target-probe hybrid, which allows for more rigorous washing.

If more than one probe is used at the same time there is still a risk of cross-reactions occurring but the products of these reactions will not be circular molecules and will thus be easy to distinguish from the "correct" results. The probe consists of three distinct parts, the two end segments and a linker segment connecting the end segments. To enhance detection and to amplify the signal, the linker segment can be equipped with sequences that gives it certain such features such as primer sequences for replication reactions to amplify the signal or a zip code, enabling easy detection of probes on a micro-array.

We have written a program that, when given the target sequence, the position of the nucleotide of interest, what zip code to insert and certain experimental conditions, designs a padlock probe that can then be produced chemically and put to work in the investigative field.

The program begins with designing the end-segments, giving them sequences that enable them to hybridise with the target sequence in the correct fashion. It then inserts a zip code and checks whether the probe can hybridise to itself. If the probe can,and this hybridisation includes the zip code,the zip code is replaced by another one until a probe is formed that can function properly.

The program is run in a directory, defining a project, in which there is a file containing possible zip codes. Since it should be possible to apply (and design) several probes at once, the used zip codes are stored in a "history file" which is checked at the beginning of every insertion of a zip code. A zip code that has been used previously is not used again in the same project.

Checking for internal hybridisations is necessary since any such events would severely damage the performance of the probe. If the 3'-end of the probe could hybridise with another region, this would lead to a probe that would probably be totally useless.

The program is written in C++ using object-orientation. Its design is modular, making it easy to add or replace functions of the program, something that will probably be necessary since new detection methods or other features are likely to be developed in the future.