

gRNAid:
A Tool for the
Comparative Analysis
of RNA Secondary
Structures

Shannon Whitmore

A research paper Submitted to
Oregon State University
in partial fulfillment of the requirements for the degree of
Master of Science

Presented: March 9, 1993

Table of Contents

Abstract

Chapter 1 - Introduction

gRNAid In A Nutshell

gRNAid And Secondary Structure Creation

Related Work

Road Map For The Remainder Of This Paper

Chapter 2 - Design

The Alignment File

Nucleotide Numbering

The Template File

Secondary Structure Creation

Editing Capabilities

Template File Management

Structure Creation As An Iterative Process

Chapter 3 - Implementation

Programming Environment

Class Hierarchy

Nucleotide Classes

Bond Classes

List Classes

Document Classes

Application Classes

Dialog Classes

Supporting Classes

Secondary Structure Internal Representation

Why A Linked List?

Dependency Graph

Bond Drawing Algorithm

Noncanonical

Watson-Crick

Conclusions

Algorithm Analysis

Chapter 4 - gRNAid User's Manual

Generating A Secondary Structure

Editing Operations

Selection & Deselection of Nucleotides

Dragging

Selection & Dragging of Bonds

The Hidden Nucleotide Bar

Mapping Hidden Nucleotides

Map Mode

Mapping

Exiting Map Mode

Forming And Breaking Bonds

Menu Commands

Apple Menu

File Menu

Edit Menu

Structure Menu

Nucleotide Menu

Font Menu

Size Menu

Chapter 5 - Conclusions

Results

Analysis of Results

Insertions in Insertions

Further Areas of Study

Final Analysis

Appendix A - Tutorial

Secondary Structure Generation

Selecting and Dragging

Mapping Nucleotides

Forming and Breaking Bonds

Potpourri

Bibliography

Abstract

The secondary structure of a 16S rRNA molecule is a graphical, two dimensional representation used by molecular biologists in determining evolutionary relationships between different organisms. By comparing two secondary structures, scientists can obtain knowledge of how 'related' one species of bacteria may be to another species [OLSE 1986]. To date, there is no widely used computer program for secondary structure creation because these programs generally do not work well with large sets of sequence data. In other words, these programs work adequately on small RNA fragments, but tend to create incorrect or unwieldy secondary structures on larger data sets. Thus, the molecular biologist is forced to create the secondary structure by hand, which is a monotonous and time consuming task. The gRNAid program is a tool implemented for the Macintosh that will aid in the creation of secondary structures. Rather than trying to calculate the molecular bonding in order to form the structure, gRNAid looks at a known secondary structure and uses it as a template to create a new secondary structure for another organism.

Chapter 1

Introduction

The sequence or primary structure of a 16S rRNA molecule consists of approximately 1600 bases, or nucleotides, where each base is either an A (adenine), U (uracil), G (guanine), or C (cytosine). Within the cell, this molecule is folded in three dimensions. This is referred to as the tertiary structure of the molecule. Both secondary and tertiary structure are held together primarily by hydrogen bonds. A Watson-Crick bond is formed when A-U or G-C nucleotides pair together. A weaker bond, referred to as the noncanonical base pair, is formed when G-U or G-A nucleotides bond together, where G-U is stronger than G-A. Although the tertiary structure is useful for studying RNA function, this three dimensional structure is complex to obtain and difficult to analyze [ZUK2 1989]. Thus, a flattened, two-dimensional model of the molecule, referred to as the secondary structure, is used most often in analysis. A secondary structure for the bacteria *Escherichia coli*, from now on referred to as *E. coli*, is shown in Figure 1-1. Notice that the bonding information from the tertiary structure is preserved in the secondary structure.

Research in the area of secondary structure generation can be divided into two main areas. The first attempts to generate these structures were by the calculation of the minimum free energy chemical bonding that occurs in the molecule, thus simulating the way that the molecule folds in its natural environment. This approach will be referred to throughout this paper as the thermodynamic method. The second method relies on the fact that much of the structure of the 16S rRNA molecule is conserved from molecule to molecule. That is, if you compare *any* two 16S rRNA molecules, approximately 60 percent of the secondary structure for these molecules will be same [GUTE 1985]. As you compare molecules that are more evolutionarily related to each other, this percentage will increase and more of the structure of the molecule will be conserved. This approach will be referred to as the phylogenetic method of secondary structure creation [LE 1989].

gRNAid In A Nutshell

The gRNAid program takes the phylogenetic approach to the determination of secondary structure models. That is, it uses a known 16S rRNA secondary structure as a template when generating an unknown structure. To create the new structure, gRNAid requires two files as input: a template file that represents a known secondary structure, and an alignment file that characterizes the aligned sequence of some organism for which we wish to generate the new structure. The sequence data in the alignment file is aligned with the sequence data of the organism represented by the template file. The gRNAid program associates screen coordinate and bonding information from the template file with nucleotides in the alignment file (formally referred to as the mapping process.) When gRNAid cannot map screen positioning information to a nucleotide in the alignment file, the nucleotide is considered to be 'hidden', and it is the responsibility of the user to manually map any hidden nucleotides to screen positions.

The best secondary structures are produced when the organisms representing the alignment and template files are closely related to each other. For example, assume we are trying to find the structure of a 16S rRNA molecule that contains 1550 nucleotides. Different results will be obtained depending on the template file we select. If the two bacteria are conserved in 97% of their structure, then only 3%, or approximately 45 nucleotides, will not be mapped to screen locations.

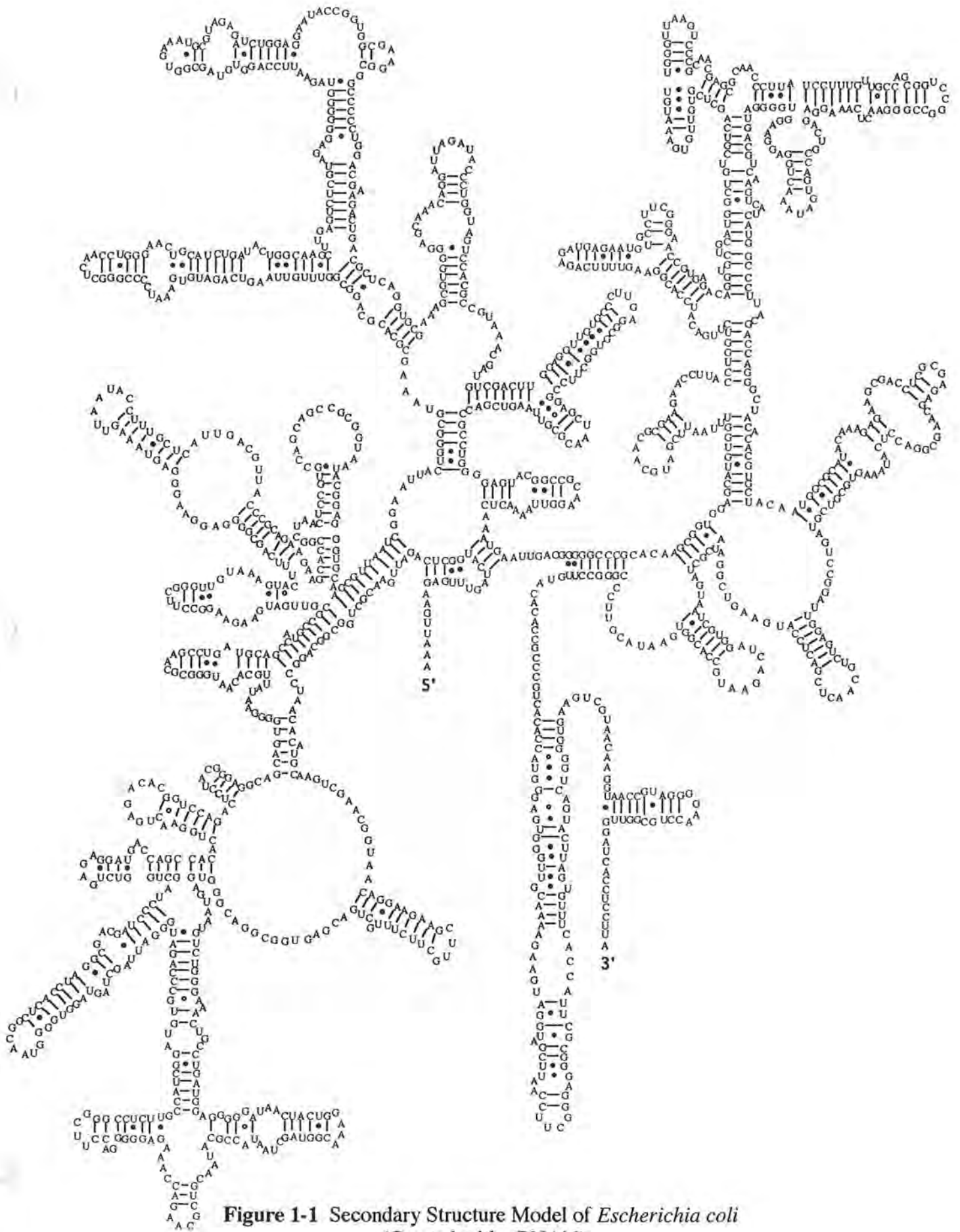


Figure 1-1 Secondary Structure Model of *Escherichia coli*
(Created with gRNAid)

Put another way, 1505 nucleotides will be mapped to screen positions. Therefore, we would only have to edit the 45 nucleotides before completing the structure. However, if we select a template where the two bacteria are only 90% conserved, 155 nucleotides will have to be manually mapped to screen positions.

gRNAid And Secondary Structure Creation

To put the process of secondary structure inference in perspective, let's go through an example of how gRNAid would be used in the molecular biologist's lab. We will walk through an example of the steps that a molecular biologist would go through to create the secondary structure for the bacteria called *Anacystis nidulans*, from now on referred to as *Anacystis*.

First, the organism would be obtained by, for example, visiting Waldo Lake and taking a sample at 50 feet below the surface. The sample would be brought back to the lab where we would determine the sequence of the *Anacystis* 16S rRNA molecule. The first 300 bases of *Anacystis* are presented in Figure 1-2.

Anacystis	AAAUGGGAGAGUUUGAUCCUGGCUCAGGAUGAACGCGUGGCGGCGUGCUUA
Nidulans	ACACAUGCAAGUCGAACGGGCUCUUCGGAGCUAGUGGCGGACGGGUGAGU
	AACGCGUGAGAAUCUGCCUACAGGACGGGGACACAGUUGGAAACGACUG
	CUAAUACCCGAUGUGCCGAGAGGGUGAACAUUUAUGGCCUGUAGAUGAGC
	UCGCGUCUGAUUAGCUAGUUGGUGGGGUAAGGGCCUACCAAGGCGACGAU
	CAGUAGCUGGUCUGAGAGGAUGAUCAGCCACACUGGGACUGAGACACGGC

Figure 1-2 First 300 bases of *Anacystis nidulans* sequence

Once the primary structure of *Anacystis* has been determined, we would select another bacterium for which we already know the secondary structure. We would use this known secondary structure as a template to create the secondary structure for *Anacystis*. Recall that at least 60 percent of the structure of any two 16S rRNA is conserved. It has been hypothesized that at least 90 percent of the secondary structure is conserved when comparing the secondary structures of any two bacteria. This percentage increases for bacteria that are closely related. The field that studies evolutionary relationships between organisms is called phylogeny [OLSE 1986]. The results of phylogenetic studies produce graphs that show evolutionary relationships between organisms, and these results can be used to come up with good estimates of the optimal bacterial secondary structure to select as a template. In this example, we will select *E. coli* because it is closely related to *Anacystis*, and its secondary structure is known.

Next, we would go through the process of aligning the *Anacystis* sequence with the *E. coli* sequence, which means to textually line up the structurally equivalent domains of the two structures nucleotide by nucleotide. An example segment of *E. coli* aligned with *Anacystis* is shown in Figure 1-3. The reason for alignment is twofold: 1) there are nucleotides in the *Anacystis* segment that are not found (genetic deletions) in the homologous *E. coli* segment, and 2) there are nucleotides in the *Anacystis* that have been inserted between nucleotides in the *E. coli* segment. These insertions and deletions denote the areas where the *Anacystis* secondary structure that will be structurally different than the *E. coli* secondary structure. For example, an insertion of 100 bases in the *Anacystis* sequence might denote an area where a new hairpin loop is to be appended in the *Anacystis* secondary structure. A detailed explanation of the process of obtaining an alignment is beyond the scope of this paper. See <reference> for more detailed information on alignment. [Anyone HELP! What is a good reference to the process of alignment?]

There are several additional points to make about the alignment in Figure 1-3. First, notice the thousand, hundreds, tens and units row. Every nucleotide in the *E. coli* sequence is numbered from 1 to 1562. Nucleotides in *Anacystis* that align with nucleotides in *E. coli* are considered to be at the same nucleotide number. When no nucleotides in the *Anacystis* sequence map to nucleotides in the *E. coli* sequence, a deletion is denoted by placing the dash character in the sequence of *Anacystis*, as shown in Figure 1-3. When there are insertions in *Anacystis*, dashes are placed in the *E. coli* sequence in order to make room for the inserted nucleotides in the *Anacystis* sequence.

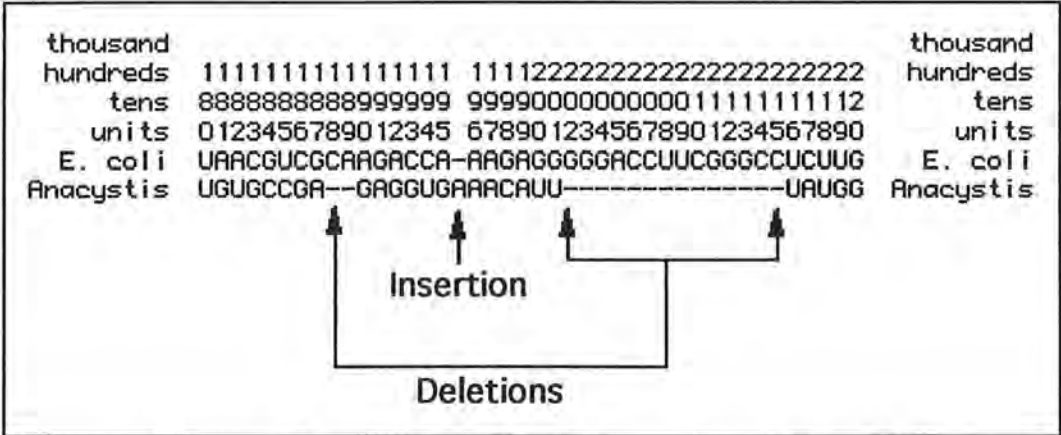


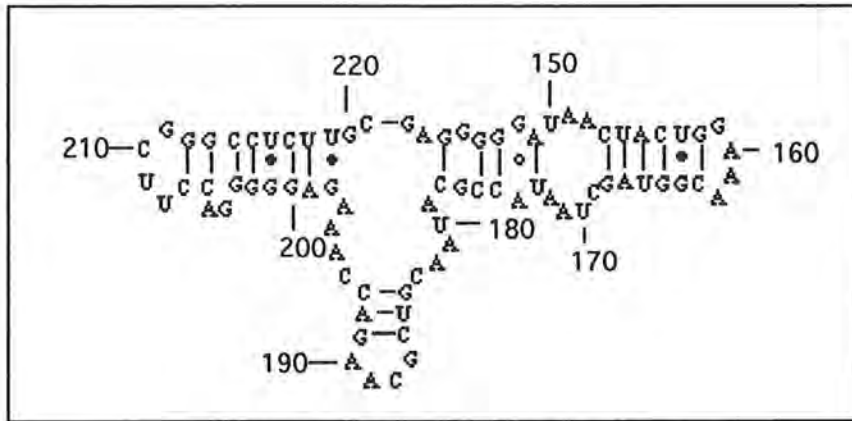
Figure 1-3 Alignment of *Anacystis* with *E. coli*

Once an alignment is complete, gRNAid is used to create the *Anacystis* secondary structure. The gRNAid program takes data from the known *E. coli* secondary structure and maps it onto the nucleotides provided by the *Anacystis* alignment to produce a partial secondary structure for *Anacystis*. Figure 1-4 shows a diagram of the mapping process for a segment of the *Anacystis* aligned sequence. The topmost box in the figure consists of an *Anacystis* sequence segment that has been aligned with *E. coli*, the middle box shows the known *E. coli* secondary structure or template, and the bottom box contains the result of mapping the aligned *Anacystis* sequence onto the *E. coli* secondary structure. The bold G nucleotide in the bottom box indicates an area where there are hidden nucleotides. That is, if you look at position 195 in the aligned *Anacystis* segment, you will notice that there is an additional A inserted after the G. There is not an *E. coli* nucleotide that corresponds with this inserted A. Thus, it is up to the user to associate screen positioning and bonding information with this hidden A. When deletions are encountered in the *Anacystis* sequence, no nucleotides are mapped to physical locations, and the secondary structure tends to look choppy. The two areas of deletions in the bottom box can be arranged to form hairpin loops. Figure 1-5 shows the correct *Anacystis* secondary structure once insertions and deletions have been accounted for in this nucleotide segment. The process of mapping nucleotides to create secondary structure will be explained in Chapter 2.

thousand		thousand
hundreds	11111111111111 1111222222	hundreds
tens	88888889999999 99990011112	tens
units	01234567012345 67890167890	units
Anacystis	UGUGCCGAGAGGGUGAACAUUUUAGG	Anacystis

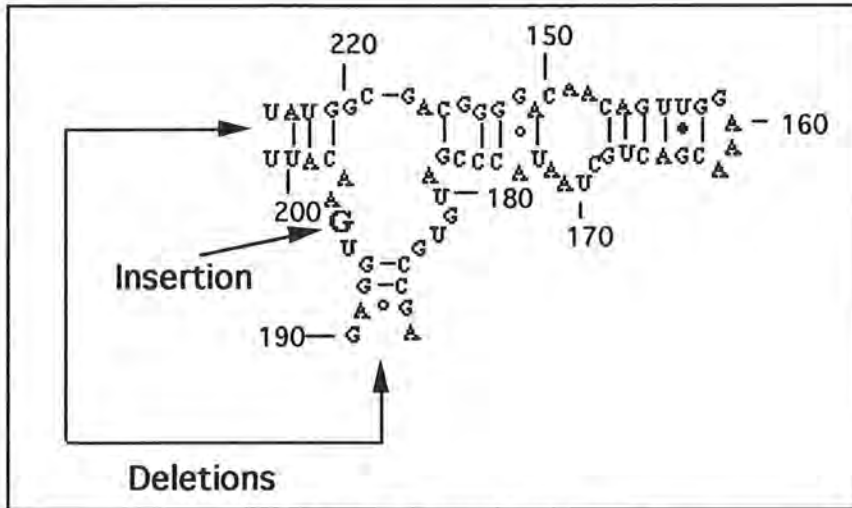
Aligned segment of *Anacystis*

+



Known *E. coli* secondary structure

=



Anacystis partial secondary structure

Figure 1-4 The gRNAid process of secondary structure creation

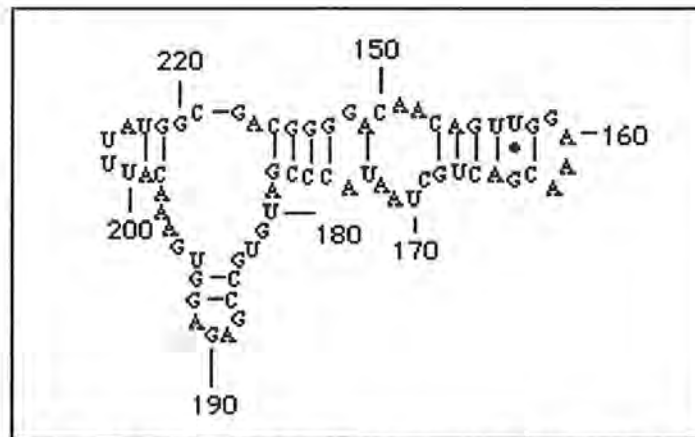


Figure 1-5 Complete *Anacystis* secondary structure segment

Related Work

In Gouy et. al. [GOUY 1987], three types of secondary structure prediction algorithms were outlined:

1. Combinatorial algorithms calculate all helices that can be formed by a particular sequence and then compute secondary structure by selecting the helices with the optimal minimum free energy fold.
2. Recursive algorithms determine structure by breaking the alignment into fragments and then finding the structure of each of the smaller fragments. Minimum free energy calculations are performed on small sequence fragments, and these smaller segments are combined to obtain the structure for the larger segments.
3. Heuristic algorithms make assumptions about the free energy of folding, and as a result, they do not always find the correct secondary structure.

Notice that the phylogenetic method used by gRNAid is not included in the above list, although it was briefly mentioned in the Gouy's paper. In Le et. al [LE 1989], secondary structure algorithms were generalized into two groups, the first of which is a superset of the algorithms mentioned above:

1. The thermodynamic programming method, where the prediction of the secondary structure is based on thermodynamic data.
2. Phylogenetic comparison, where secondary structures are created using conserved regions of the structure as a template.

There are two programs that are currently available on the Macintosh that work with secondary structure data. The first is called Mulfold [ZUK1 1989, JAE1 1989, JAE2 1989], and it is based on the original version called MFOLD that was originally implemented on VMS/VAX. Mulfold is limited to working with 300 bases, so its use on the Macintosh is limited (note that the VAX version works with up to 2000 bases.) Mulfold predicts secondary structures via free energy minimization, and can output the structure in either a graphical or textual format. The graphical format is limited in its usefulness in that it displays the structure in typewriter-style. Figure 1-6 shows an example of a typewriter format secondary structure that was created for 5S RNA *wheat embryo*.

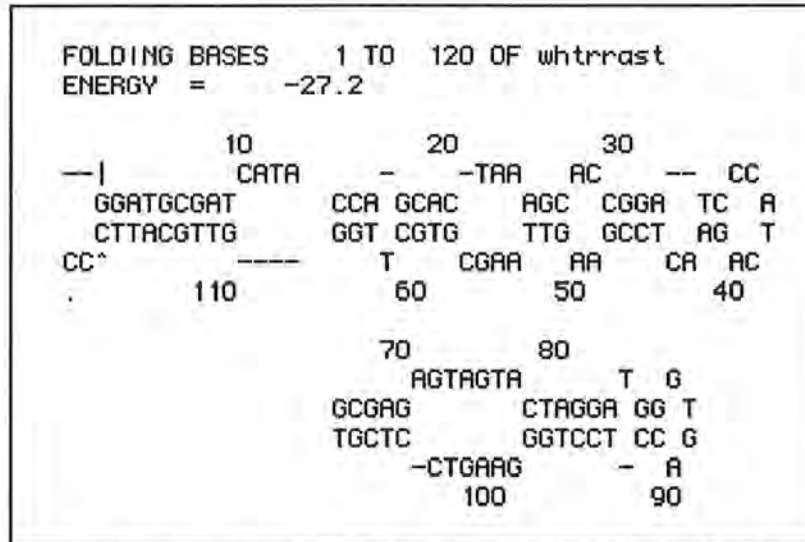


Figure 1-6 *Wheat embryo* secondary structure produced in typewriter format

Mulfold can also save data in a textual form, referred to as connect format, that can be imported into another Macintosh based program called LoopViewer [GILB 1990]. For example, the result of saving the *wheat embryo* 5S RNA into a connect file and displaying the structure with LoopViewer is shown in Figure 1-7. LoopViewer can be used to display structures and save them in a PICT file format that can be used by other drawing applications such as MacDraw®. However, LoopViewer cannot at this time be used to generate or edit secondary structures. LoopViewer was to be a subset of another application called LoopDLoop, which was to allow secondary structure editing features, but as of this writing, the status of LoopDLoop is unknown.

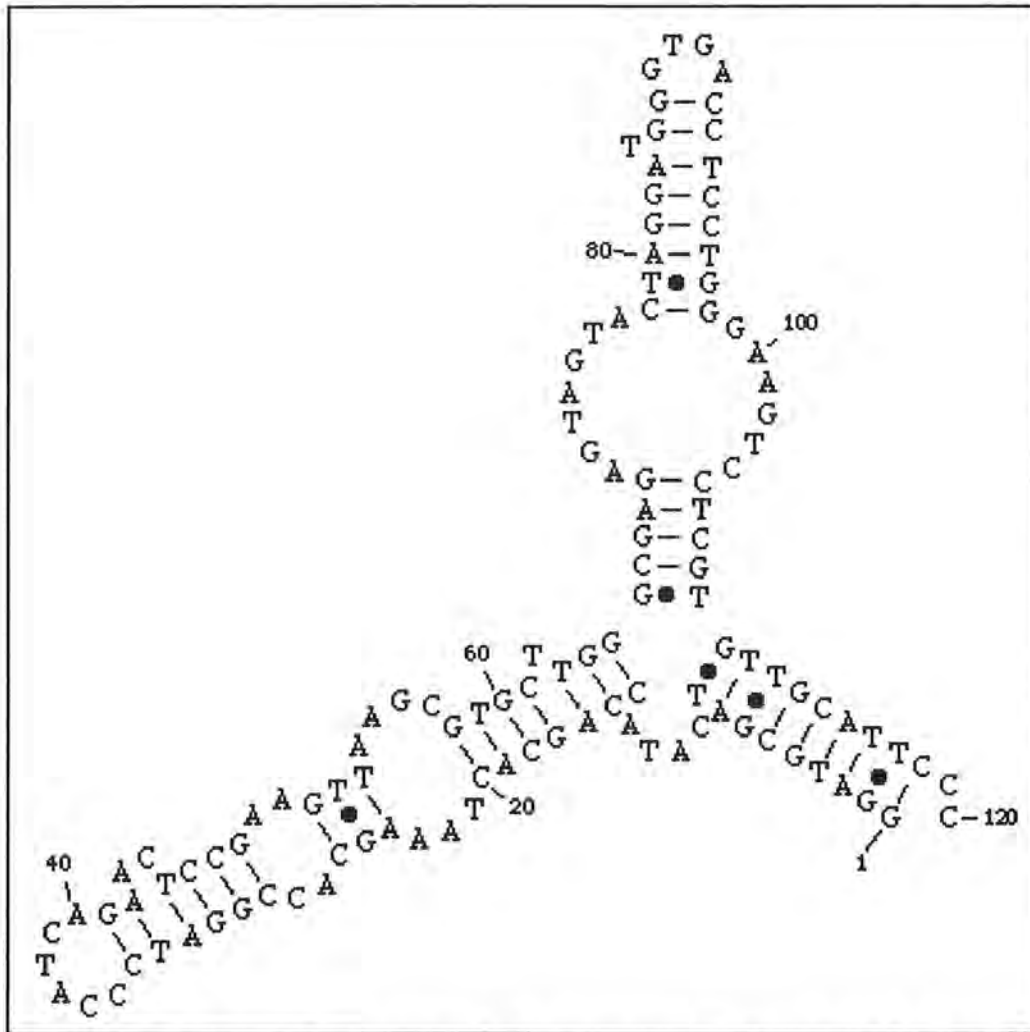


Figure 1-7 Wheat embryo secondary structure displayed with LoopViewer

The Mulfold tool required approximately 32 minutes to predict the wheat embryo sequence of 120 bases. The authors claim that the time that Mulfold takes to predict a secondary structure is on the order of N^3 , where N is the number of nucleotides in the sequence. Thus, it comes as no surprise that it took Mulfold 3 hours and 53 minutes to compute the structure for a 297 base segment of *E. coli*. Actually, Mulfold computed 6 different secondary structure fragments during this time. Of these 6, the best structure is shown in Figure 1-8, where the fragment on the left represents the structure predicted by Mulfold and the structure on the right is the correct structure for the given segment of *E. coli*. Although there is a similarity between the two structures in the figure, there are some obvious differences between the two. The fact that Mulfold did not produce a completely correct secondary structure is probably not attributable to the fact that only a fragment of the *E. coli* data was provided.

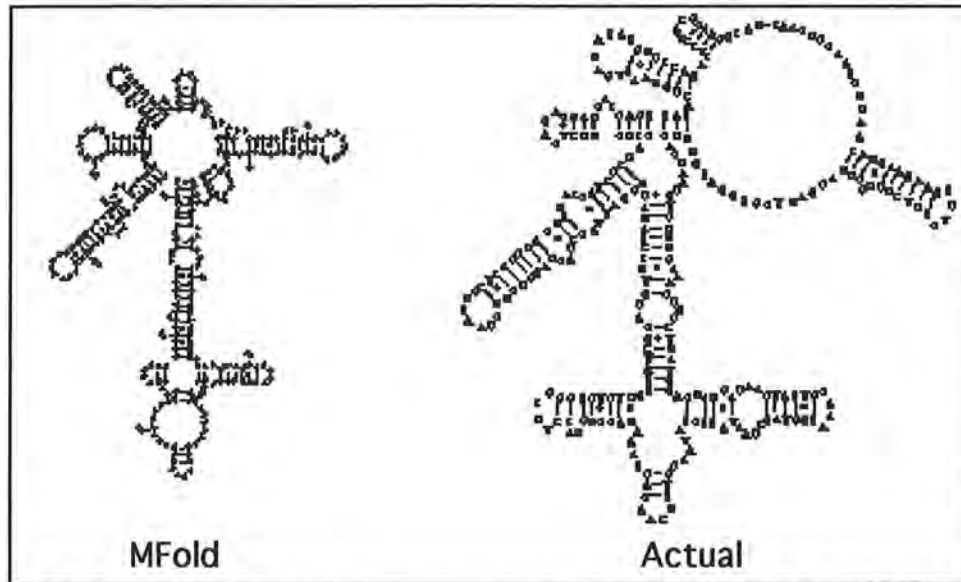


Figure 1-8 Left: *E. coli* fragment predicted by Mulfold. Right: Actual structure of the same *E. coli* fragment.

Most of the secondary structure programs available today are based only on the thermodynamic method of computing secondary structure. The problem that these programs must solve is twofold: 1) predict the secondary structure, and 2) correctly display the secondary structure. Thus, even though a program might be able to generate correct secondary structure data (such as the connect file generated by Mulfold), the display of this data may consist of secondary structure segments that overlap. For example, another structure produced by Mulfold for the *E. coli* fragment is shown in Figure 1-9. Notice that a portion of the structure overlaps, making it difficult to read. As more complex secondary structures are generated, the display of these structures becomes more complicated, and sometimes the result of displaying the structure is so snarled that it is easier to create the structure by hand than try to untangle the mess. Because gRNAid uses a template file to associate nucleotides with screen positions, this problem is effectively avoided.

Road Map For The Remainder Of This Paper

The next chapter delves into the details of how gRNAid generates a secondary structure, and covers topics such as nucleotide numbering, the mapping process, and the importance of the alignment and template files. Chapter 3 looks under the hood and provides implementation level details, such as algorithms and data structures used in the gRNAid program. Chapter 4 is a User's Manual and provides a reference to the functionality provided by gRNAid. Chapter 5 presents the conclusions of this research effort and also a section detailing further areas of study. Finally, Appendix A contains a tutorial that provides step-by-step instruction on how to use gRNAid.

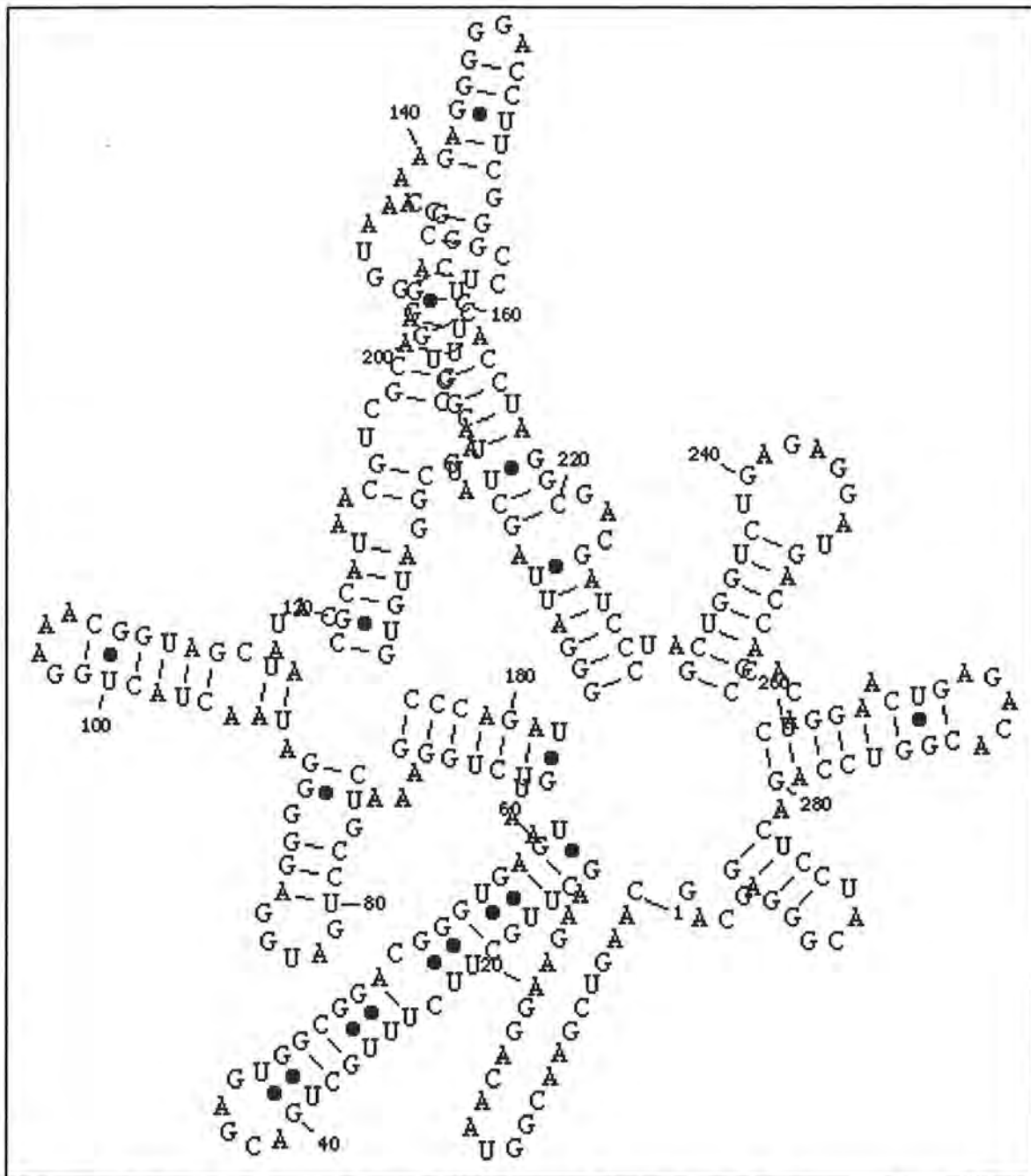


Figure 1-9 Programs that use the thermodynamic method sometimes have problems with displaying secondary structure once it is predicted.

thousand	
hundreds	111111111111111111
tens	66666777777777788888888889999999999000000000111111111
units	567890123456789012345678901234567890123456789012345678
An.nidul	AACGGGCUCUU-----CGGAGCUAGUGGCGGACGGGUGAGU

Figure 2-3 Deletions in the *Anacystis* sequence

thousand	
hundreds	111111111111111111
tens	666667777779999999000000000111111111
units	567890123454567890123456789012345678
An.nidul	AACGGGCUCUUCGGAGCUAGUGGCGGACGGGUGAGU

Figure 2-4 *Anacystis* alignment with the deletion characters removed

Nucleotide Numbering

The nucleotide numbering scheme is used to give conserved regions unique identifiers. That is, nucleotides that form the same structure or are at the same physical location should be given the same identifier. The nucleotide numbering scheme used for *E. coli* is considered to be the standard for all bacteria. Nucleotides in the *E. coli* sequence are numbered consecutively from 1 to 1542, with the 1 at the 5' end and the 1542 at the 3' end (Figure 1-1). Since we already know the structure of *E. coli*, any other sequence that contains nucleotides numbered in the 1 to 1542 range 'inherits' the same structure as *E. coli*. The alignment process can also be defined as trying to assign nucleotide numbers to a sequence with the goal of finding the conserved regions between two bacteria.

The nucleotide numbering scheme provided in both the alignment and template files is the key ingredient gRNAid uses for secondary structure creation. In the alignment file, many of the rows contain a nucleotide number with a maximum of 4 digits (thousand, hundreds, tens, units). For example, A is the first nucleotide in the sequence shown in Figure 2-4 and is at nucleotide number 65. Notice that nucleotide numbers are not provided in the alignment file when there is an insertion, such as the A shown in Figure 2-2. When there are deletions in the alignment, the nucleotide number should be completely removed from the alignment file, as shown in Figure 2-4.

The nucleotide numbering scheme used by the template file is slightly different than that used by the alignment file. Each nucleotide number is considered to be in decimal. For example, nucleotide 10 in the alignment file would be nucleotide 10.0 in the template file. The use of decimals within the nucleotide number gives gRNAid more information when creating secondary structures. Nucleotides at areas of insertion can now be assigned their own unique nucleotide numbers. For example, in Figure 2-5, areas of insertion relative to *E. coli* are shown by nucleotide numbers 14.1 and 14.2. This extra nucleotide numbering information can be used by gRNAid to create the secondary structure in these areas of insertion.

The Template File

The template file contains information about the known secondary structure for a particular bacterium. Once a complete template file has been created, it can be used to predict secondary structures for other bacteria. A segment of a template file for some bacteria from nucleotide number 10.0 through nucleotide number 25.0 is shown in Figure 2-5.

The diagram shows a rectangular box containing a list of nucleotide data. Above the box, the labels 'Nucleotide Number' and 'Base' have arrows pointing to the first and second columns of the data, respectively. To the right of the box, the label 'Bonding Information' has an arrow pointing to the fourth column. Below the box, the label 'Screen (X,Y) Coordinates' has two arrows pointing to the third and fourth columns. The data is as follows:

Nucleotide Number	Screen X	Screen Y	Base	Bonding Information
10.0	514	492	A	24.0
11.0	522	492	G	23.0
12.0	528	493	U	22.0
13.0	535	493	U	21.0
14.0	542	498	U	
14.1	545	498	U	
14.2	547	498	G	
15.0	549	499	G	
16.0	555	493	A	
17.0	557	486	U	918.0
18.0	552	480	C	917.0
19.0	547	474	A	916.0
20.0	541	469	U	915.0
21.0	535	474	G	13.0
22.0	528	475	G	12.0
23.0	522	475	C	11.0
24.0	514	475	U	10.0
25.0	507	475	C	9.0

Figure 2-5 Template file segment for some bacteria

There are four important pieces of information included in the template file:

1. Nucleotide number (in decimal)
2. X,Y screen coordinates
3. Base
4. The nucleotide ID of the base bonded to, if needed

Each nucleotide in the template file has a unique nucleotide number associated with it, including nucleotides at areas of insertion. The screen coordinates give the two-dimensional location of the nucleotide in some coordinate system. The base can be any alphanumeric character, although the base will usually be an A, U, G, or C. The fourth piece of information in the template file contains bonding information, which only exists for nucleotides that form a base-pair. For example, in the first line of the template file in Figure 2-5, nucleotide 10.0 bonds with nucleotide 24.0. Similarly, it is indicated that nucleotide 24.0 bonds with 10.0, giving the template file a 'symmetrical' characteristic in its bonding information.

Secondary Structure Creation

The key to the creation of the secondary structure is in the nucleotide number. Because the alignment file does not assign nucleotide ID's to all bases, it is up to gRNAid to do so. If a nucleotide ID has not been assigned to a base, then the new ID is the sum of the previous ID plus 0.1, as shown in Figure 2-6.

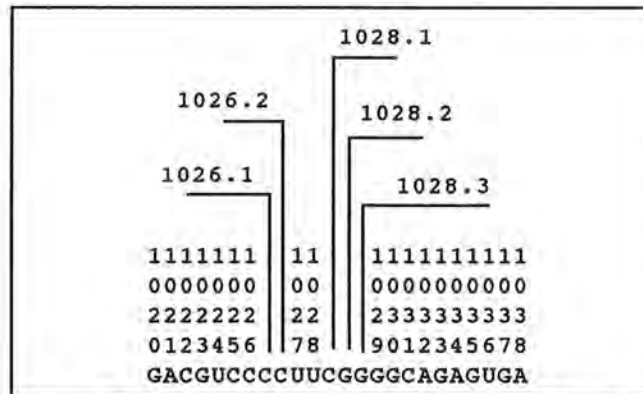


Figure 2-6 Assigning ID's to inserted nucleotides

When creating a new secondary structure, gRNAid requires information from both the alignment and template files. The sequence in the alignment file should be aligned with the sequence representing the template file, or, in the very least, aligned with the E. coli sequence. The goal is to create a secondary structure for the organism represented by the alignment file. When a nucleotide number is in both the alignment and template files, the screen positioning and bonding information from the template file can be associated with the base in the alignment file, as shown in Figure 2-7. In the figure, nucleotide 10 in the alignment file maps to nucleotide 10.0 in the template file. Thus, the G in the alignment file can be associated with a screen position (514,492) and can be bonded with nucleotide 24.0.

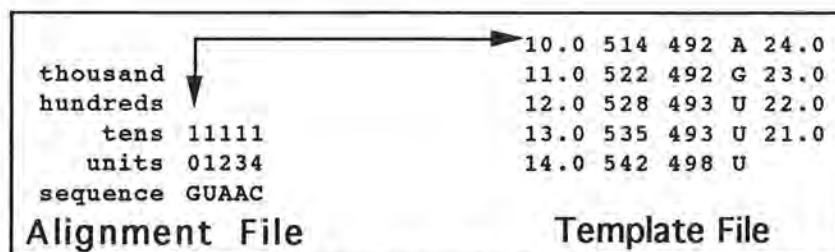


Figure 2-7 Mapping nucleotide 10 with data in the template file

The algorithm for mapping the aligned sequence onto positioning and bonding information can be described by the following three items:

1. If a nucleotide ID is found in both the template and alignment files, then the screen positioning and bonding information from the template file can be mapped onto the nucleotide in the alignment file (see 1 in Figure 2-8.) A nucleotide ID of 16.1 is associated with the A in the alignment file, and there exists a 16.1 in the template file. Thus, information from the template file can be mapped to the A in the alignment file.
2. If a nucleotide ID in the alignment file cannot be found in the template file, then gRNAid *cannot* associate any screen positioning or bonding information with it (see 2 in Figure 2-8.) There is no 17.1 or 17.2 in the template file, so coordinate and bond information cannot be associated with either of these nucleotides, and as a result, these nucleotides are considered to be invisible, or hidden. The process of dealing with hidden nucleotides is discussed later in this section.
3. If a nucleotide ID in the template file cannot be found in the alignment file, ignore it (see 3 in Figure 2-8). There is a nucleotide 15.0 in the template file, but no nucleotide 15 in the alignment file. This extraneous information from the template file can be disregarded.

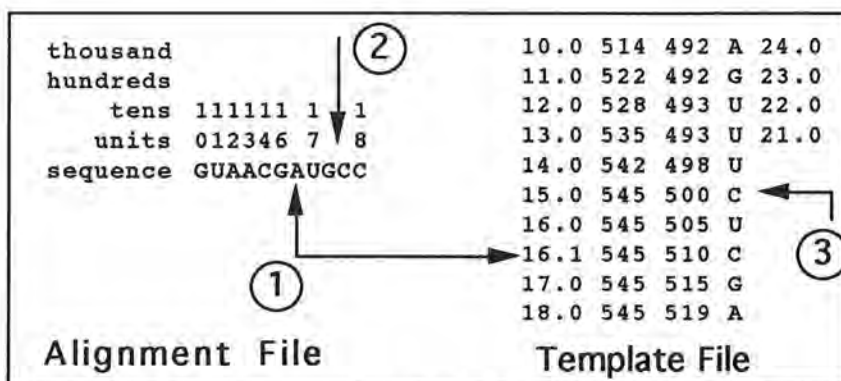


Figure 2-8 Demonstration of the mapping algorithm

In the above algorithm, the bonding information is only mapped if both of the bonds in the base pair are available. For example, the template file in Figure 2-8 has nucleotide 10.0 bonded with nucleotide 24.0. Although nucleotide 10 exists in the alignment file, it is possible that nucleotide 24 does not exist. Thus, the bonding information from the template file cannot be used. Also, only legal Watson-Crick (A-U,G-C) and noncanonical (G•A,G•U) bonds will be formed. For example, assume that nucleotides 10 and 24 exist in the both the alignment and template files. If the bases in the pair do not form a legal bond, then the bonding information will be discarded.

Recall from item 2 in the algorithm that some nucleotides may not be associated with screen positions and are considered hidden. It is up to the user to map these hidden nucleotides to screen positions. It is gRNAid's responsibility to provide access to these hidden nucleotides. If there is a hidden nucleotide in the structure, the last visible nucleotide in the alignment sequence will be used to graphically denote an area of insertion. For example, in number 2 of Figure 2-8, the last visible nucleotide is 17, and will be used to show that there are hidden nucleotides that follow.

In gRNAid, bold nucleotides represent areas where hidden nucleotides exist, as shown in Figure 2-9. In the figure, the bold A and U represent areas where hidden nucleotides can be found. It is the responsibility of the user to eventually associate screen positions with these hidden nucleotides. This mapping process is performed by clicking the mouse button at the desired location in the window for each hidden nucleotide. More details on this mapping process can be found in the *Mapping Hidden Nucleotides* section of Chapter 4.

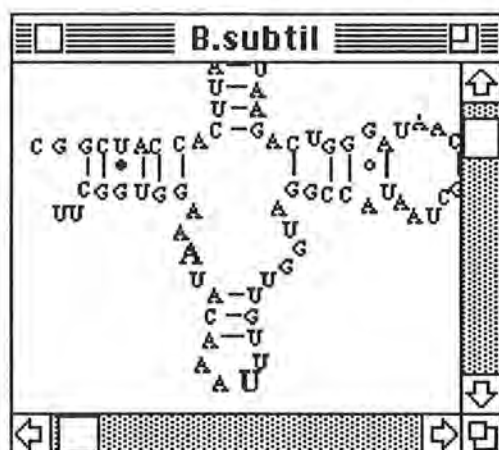


Figure 2-9 Hidden nucleotides appear after bold nucleotides

Editing Capabilities

It is not the intent of gRNAid to provide powerful editing capabilities such as those found in MacDraw® and in other drawing programs. Rather, the goal is to quickly produce a secondary structure for analysis. Since gRNAid produces a secondary structure with hidden nucleotides, it is important to provide a minimal set of editing features to create a correct template file. The set of editing features currently provided by gRNAid are selection, drag, and base-pair operations. Nucleotides within the structure can be selected and dragged. Bonds cannot be selected or dragged by themselves and are considered to be owned by the base-pair that forms the bond. Consequently, the only way to directly select and drag a bond is through the owning base-pair. The forming and breaking of bonds are the base-pair operations available within gRNAid.

Template File Management

The beauty of gRNAid is that newly created secondary structures are saved as template files, and thus can be used in the generation of other secondary structures. Within gRNAid, the template file can be used in two ways: 1) as a secondary structure, and 2) as a file used in the creation of other secondary structures. For the first usage, a template file can be saved to represent the correct secondary structure of a particular organism. Within gRNAid, the structure can be opened, viewed, edited, and saved again. For the second usage, the template is an important ingredient in the creation of a new secondary structure.

Template files can be saved at any time, even when there are hidden nucleotides in the secondary structure. At save time, any nucleotides that do not have screen coordinate information are given the special coordinate @,@. This allows gRNAid to easily recognize the hidden nucleotides when reading the file from disk. This also means that template files containing hidden nucleotides could be used in the secondary structure generation process, which is not recommended.

Structure Creation As An Iterative Process

The gRNAid program assumes that the alignment file is correct, especially in primary structure. The process of alignment and secondary structure generation should be an iterative one:

1. A partially aligned sequence is saved into an alignment file.
2. A secondary structure is created with gRNAid.
3. The user notes areas (if any) where the alignment is incorrect in the secondary structure.
4. If there are errors in the alignment file, correct them and go back to step 1.

During the early stages of creating a model for a new structure, emphasis is on creating a correct alignment file. The user should not spend time editing the secondary structure, since changes will be lost on the next iteration. Once a sequence has been fully aligned, the secondary structure can be edited. It is important that the alignment file be correct in the final iteration because of gRNAid's dependence on the file during the creation of a new secondary structure. Usage of incorrect alignment files may result in an incorrect template file, which in turn can lead to incorrect results during the creation of a new secondary structure.

To produce a correct template file, the secondary structure must never become corrupted by exchanges between bases or other alterations which are physically and chemically impossible. The need for this can be seen by an example. While editing the secondary structure in gRNAid, what if the user swapped an A with nucleotide ID 32 with an A that has a nucleotide ID of 37? Even though this might seem like a harmless mistake, each and every nucleotide within gRNAid is considered to be unique through its nucleotide ID. Therefore, when this swap occurs, the template file will be corrupted because both of the A's will be associated with the wrong screen and bonding information. Thus, if the template is ever used to generate new secondary structures, nucleotides that have ID's 32 and 37 will be mapped incorrectly. In all practicality, the process of creating a correct template file will also be iterative in nature. However, once the problems have been ironed out of a template, it becomes a powerful tool to assist in the creation of new secondary structures.

Chapter 3 Implementation Details

This chapter covers details pertaining to the implementation of gRNAid. It is not the intent of this paper to explain C++, Object Oriented Programming (OOP), and Macintosh Toolbox concepts, and as such this chapter assumes that the reader has some knowledge in these areas.

Programming Environment

The gRNAid program was implemented with the Macintosh Programmers Workshop (MPW) C++ programming language, which is based on AT&T CFront 2.0. The reasons behind selecting this language were threefold. First, the C++ language facilitates the use of the Object Oriented Programming (OOP) paradigm, which would (hopefully) make program development and maintenance easier. Second, a simple Macintosh-based class hierarchy that implemented much of the standard Macintosh user interface behavior was available [WEST 1990] and could be used as a building block for implementing the gRNAid application. Third, it seemed a benefit to learn more about the OOP paradigm and the C++ language, since much work in academia and industry involves these skills.

Class Hierarchy

The gRNAid program uses 29 classes, 7 of which were provided by *Elements of C++ Macintosh Programming* by Dan Weston [WEST 1990]. These 7 classes are TApplication, TDocument, TList, TScrollDoc, TDocList, TLink, and TIterator. This section provides a short paragraph on each of the 29 classes and details how each class is used within gRNAid. Note that this section does not provide an extensive description of each class, but rather touches on its highlights. Most of the class types start with the letter T because this is the standard Apple® method for naming classes.

Nucleotide Classes

Each nucleotide in the secondary structure is represented by a TNuc object. A nucleotide knows how to draw itself and contains data pertaining to its location on the screen, its visibility, a reference to the nucleotide it is bonded to, and a nucleotide ID. Nucleotide numbers are represented by the class TNucID, which consists of overloaded arithmetic and comparison operators. Only TNuc and a select few other classes have access to the TNucID class.



Figure 3-1 Nucleotide and ID classes

Bond Classes

All bonds are also represented by a class, and because there is more than one type of bond, the class hierarchy depicted in Figure 3-2 was created. A bond can either be Watson-Crick or noncanonical, and noncanonical bonds can be further divided into strong or weak types. The `TBond` abstract class is at the top of the hierarchy and handles all generic bond data and functions. For example, all bonds reference the two nucleotides that form the base pair, and this information is stored in the `TBond` class. Most other methods within `TBond` are pure virtual. That is, a stub is provided so that the subclasses can fill in the details, such as for the `Draw()` method, since each type of bond is drawn differently on the screen.

The `TWatsonCrick` class provides specific details pertaining to Watson-Crick bonds. Since Watson-Crick bonds are graphically represented as straight lines, this class is responsible for keeping track of the two screen coordinates that define that line. `TWatsonCrick` objects also have methods to calculate these two points based on the location of the two owning `TNuc` objects that form the base pair. Details of this calculation are covered in more detail in the *Algorithms* section of this chapter.

The `TNonCanonical` class is another abstract class that implements the basic behavior of noncanonical bonds. All noncanonical bonds are represented graphically as small circles. Each circle is represented by a bounding rectangle because rectangles are parameters to the Macintosh Toolbox circle drawing routines. Keeping track of this rectangle is the responsibility of the `TNonCanonical` class. This class also provides methods that calculate this rectangle based on the coordinates of the two owning `TNuc` objects that form the bond. More information on calculating this noncanonical rectangle is provided in the *Algorithms* section of this chapter.

The `TStrongNonCanonical` and `TWeakNonCanonical` classes only differ in their drawing methods: one draws a hollow circle (o) and the other draws a solid black circle (*). All other data and behavior for these two classes are inherited from their ancestors.

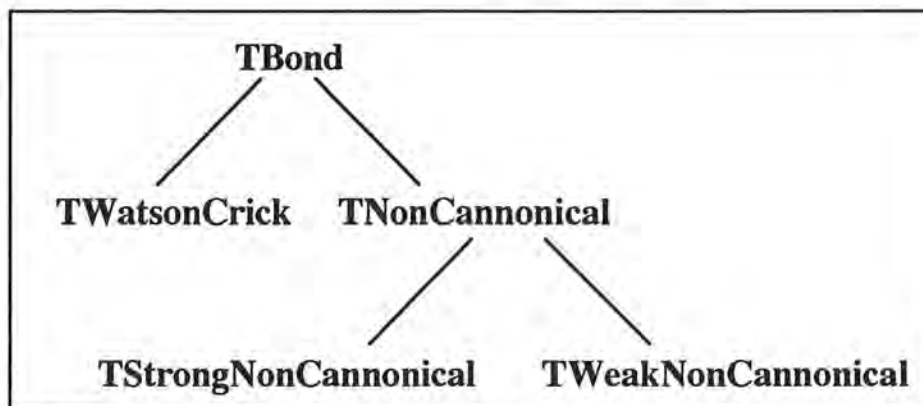


Figure 3-2 Class hierarchy for the bond

List classes

List classes are the main data structures used to store and access information used in `gRNAid`. The class hierarchy representing all lists within `gRNAid` is shown in Figure 3-3. The `TList` class is a generic class that implements a simple linked list. This class will store any object and the only operations available are for the insertion and retrieval of objects in the list. It is up to child classes to enhance the behavior of the list when necessary. There are two other classes, `TLink` and `TIterator`, that are available as utilities to the `TList` class. The `TLink` class, which is

used internally by the TList and its subclasses but is not visible to any other class, implements the links that build the linked list. The TIterator class, which is available to all classes that operate on a list, is used as an aid for iterating through the linked list.

The TDocList class is used internally by the TApplication class (discussed later) to keep track of all open files and windows in the application. It inherits most of its functionality from the parent class. The TBondList class is responsible for keeping track of all bonds in the secondary structure and as well as managing all bonding routines. The TSelectedList class keeps track of all nucleotides that have been selected in the secondary structure window. This class provides routines to drag and select nucleotides as well as drawing all nucleotides in their highlighted state.

The TNucList class is responsible for the data that makes up the secondary structure and is considered to be a major data structure within the grNAid implementation. TNucList stores all nucleotides in the secondary structure and also contains a reference to TBondList and TSelectedList objects. Most higher level classes only know about a TNucList object, and if an event occurs pertaining to bonds or selected objects, the TNucList object delegates the task to the appropriate class object. Thus, the TNucList class is considered to be more of a management class. Details behind the TNucList data structure will be provided later in this chapter.

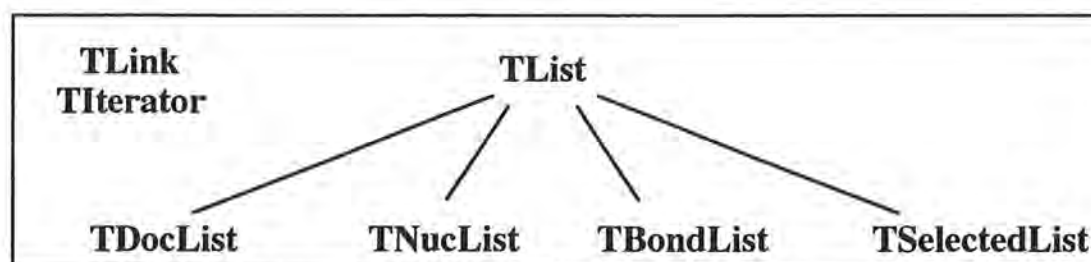


Figure 3-3 List and list utility classes

Document Classes

Data from the application is handled by a generic class called TDocument. Because data is usually displayed in a window in some type of format, the TDocument class handles generic windowing operations such as displaying, dragging, or moving a window. The class hierarchy showing all subclasses of TDocument is shown in Figure 3-4. The TDocument class is more concerned with the user interface aspect of data management than anything else, and it is up to subclasses to provide more detailed operations, such as reading and writing files.

Another generic document class is TScrollDoc, which provides functionality for a scrollable window. TScrollDoc uses much of the inherited windowing behavior from TDocument and overrides other TDocument methods to obtain its scrolling behavior.

TStructureDoc is a high level class that is mainly responsible for the data in the secondary structure window. It inherits much of the classic Macintosh user interface behavior from TScrollDoc and TDocument, and thus concentrates on features unique to the grNAid application. TStructureDoc does not know about TNuc, TBond, TBondList, and TSelectedList classes, and so events related to any secondary structure element are delegated to the TNucList object, which is a member of the TStructureDoc class. When a TStructureDoc does handle an event, it is usually related to the display of the secondary structure within the window.

The TPICTDoc class is used within gRNAid to save and print PICT files. TPICTDoc class was meant to be a generic class, and therefore contains functionality to read PICT files and display the graphics in a scrollable window. However, gRNAid only uses the printing and saving related routines from the class and does not use any of the display related routines.

The TAlignDoc class implements low level alignment file reading code and places the alignment data in a TNucList object. Although it is derived from TDocument, the TAlignDoc class never directly displays data in a window. Because TDocument assumes that data is to be placed in a window once it is read, the TAlignDoc class must override several file-reading related methods.

The TTemplateDoc class is a utility class that implements the reading and writing of the template file. TTemplateDoc is considered to be a helper class to TStructureDoc. That is, the algorithms from TTemplateDoc could be placed in the TStructureDoc class. The reason for separating these classes is as follows. When reading the template file from disk, the TNucList data structure needed to be created, which involved the knowledge of the TNuc, TBond, and TBondList classes. Thus, this template file related code was separated into its own class to hide low level implementation details from TStructureDoc and to keep it from being exposed to lower level classes which it need not know anything about.

The actual mapping algorithm (which maps template screen coordinate and bonding information onto data from the alignment file) is implemented in file reading code of the TTemplateDoc class. Similar to the TAlignDoc class, TTemplateDoc is not responsible for displaying the secondary structure data in a window. Therefore, TTemplateDoc is derived from TAlignDoc, from which it can inherit the same type of user interface behavior as used in its parent.

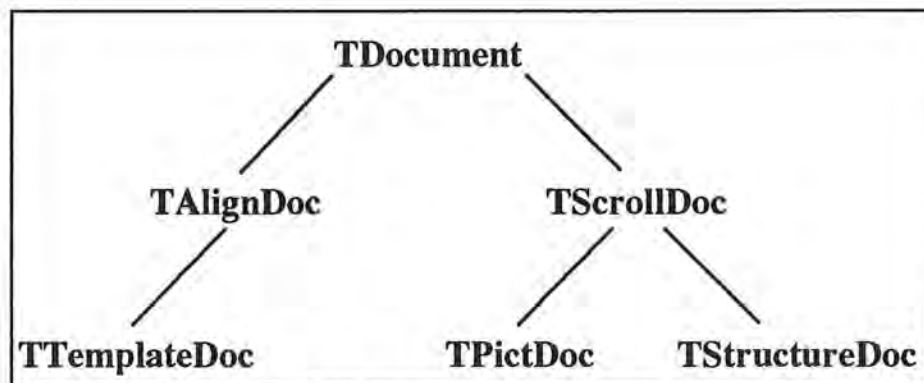


Figure 3-4 The document class hierarchy

Application Classes

The TApplication class provides methods for standard Macintosh behavior, such as the display and control of menus, MultiFinder compatibility, access to desk accessories, and a framework for handling events. The TApplication class was implemented to be a generic base class and does not provide enough functionality to be instantiated on its own. Therefore, the TgRNAidApp class is derived from TApplication (Figure 3-5) and provides application specific information that is not available in the parent. However, because the TApplication class provides much of the typical Macintosh behavior, the number of methods provided in the TgRNAidApp class is minimal.

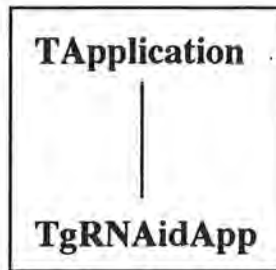


Figure 3-5 Application framework

Dialog Classes

A generic class that handles the display of dialogs is provided by the TDialog class. The functionality from the TDialog class can be accessed by subclassing off of TDialog and providing a few key details, such as the resource ID of the Dialog. The only dialog used within gRNAid was for the **About...** box, thus the presence of the TAbout class (Figure 3-6).

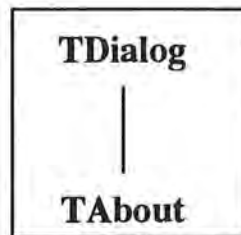


Figure 3-6 Dialog classes

Supporting Classes

Several other classes (Figure 3-7) were implemented to support the gRNAid application. The TFontMgr class is used to keep track of font information, such as current font, font size, and the maximum width of any character within the current font/font size. Font information is very important to several clients, including TStructureDoc, TNuc, and TBond. Because the font and size can be changed via menu selection, TStructureDoc must be able to access the font manager to change these values. Nucleotides and bonds are dependent on font information for their screen coordinate calculations. A TFontMgr object is owned by the TStructureDoc class. However, since TNuc and TBond objects do not access TStructureDoc objects, static (global) methods are implemented so these classes can easily access font information.

The Vector class implements a basic two-dimensional vector data type, and is used by the TBond class when calculating screen positioning information for the bond. The Util class provides utility methods such generic string and file processing functionality. All methods within Util are static and thus can be accessed by any class. Finally, Util_message is used to display messages on the screen while gRNAid is in the middle of processing. The constructor for this class takes a string representing the message to be displayed as an argument. The message window is displayed when the Util_message object is constructed, and removed from the screen when the object's destructor is called.



Figure 3-7 Utility classes

Secondary Structure Internal Representation

The data structure shown in Figure 3-8 shows how several classes described earlier work together to form the internal representation of secondary structure data. As explained earlier, lists are implemented with TLink objects that contain two pointers, one pointer to the object and another pointer to the next TLink in the list. As shown in the figure, both the TNucList and TBondList classes use the TLink class. The remainder of this section implicitly refers to Figure 3-8.

For most other classes such as TStructureDoc, the secondary structure is completely represented by a TNucList object. That is, other classes only know about the TNucList object and they will query this object when they need to access any secondary structure data. Thus, although the TNuc, TNucID, TBondList, and TBond classes are all used to internally define secondary structure data, most other classes only have knowledge of the TNucList class. When the TNucList object receives a query, it will either handle the query itself or dole out the responsibility to the appropriate class. Note, however, that a few classes such as TAlignDoc and TTemplateDoc have access to low level classes such as TNucID and TNuc for performance reasons only. Because approximately 1600 nucleotides and at least 400 bonds need to be created, it was decided to allow TAlignDoc and TTemplateDoc objects direct access to low level classes rather than slow things down by forcing them to go through the TNucList object.

TNucList is responsible for keeping track of all TNuc objects and also implements some TNuc related routines. Each TNuc object references a TNucID object, which stores the unique nucleotide ID and provides overloaded operator methods. If a TNuc is bonded with another nucleotide, the fBond member within the TNuc will reference a TBond object. The TNucList class also contains a member, fBondList, that references a TBondList object that is responsible for storing and handling all bond related information. In fact, the TNucList is not even aware of the existence of the TBond object. Thus, it is the responsibility of TBondList to delegate tasks to the TBond object when necessary. The TBondList relationship with the TBond object parallels the TNucList association with TNuc objects.

There is one TBond object formed for each base-pair, and each TBond object must be referenced by two TNuc objects. Figure 3-8 does not show any direct references to the TBond object because it is an abstract class. That is, it cannot be instantiated by itself because it does not provide enough information to be useful. Instantiated bonds only come from TWatsonCrick, TStrongNonCanonical, or TWeakNonCanonical classes, which are subclasses of the TBond class. The fPartner1 and fPartner2 members, which are references back to the owning nucleotides, are provided by the TBond class and are inherited by any subclasses of TBond. All TWatsonCrick classes contain fFirstPt and fSecondPt members, which are unique to the TWatsonCrick class and

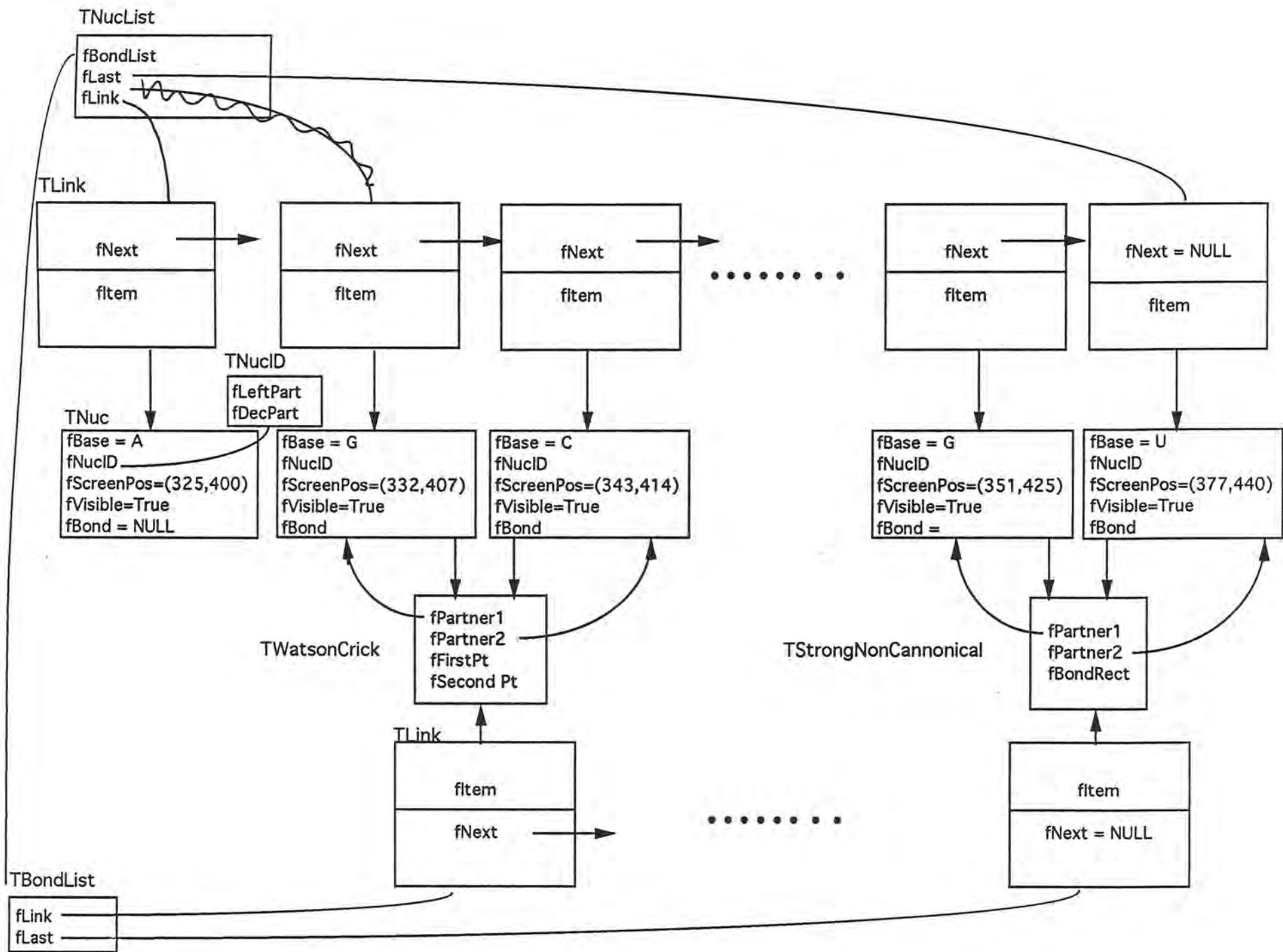


Figure 3-8 Secondary Structure internal representation

are not included in any other bond class. These members are used to represent the screen coordinates that form a straight line, which is the graphical representation of a Watson-Crick bond. The TStrongNonCanonical object contains an fBondRect field, which is inherited from the TNonCanonical class. The graphical representation of a TNonCanonical bond is a circle, and the Macintosh Toolbox routines take a rectangle as an argument when drawing a circle. The only difference between the TStrongNonCanonical and TWeakNonCanonical is in their drawing routines. The weak noncanonical bond (not shown in the figure) is drawn as a hollow circle (o), while the strong counterpart is drawn as a black dot (•).

Why A Linked List?

A linked list representation of the nucleotide and bond data was chosen because it was simple and fast enough to use with 1600 nucleotides. Prior to the decision on the type of data structure to use, a simulation was carried out to analyze the performance of a linked list with 10000 objects on the Macintosh. Each object abstractly represented a TNuc object, and the goal was to determine how long it would take to select a single nucleotide (i.e. - find a nucleotide in the list and select it) and how fast the nucleotides draw on the screen. It was determined that the linked list representation was good enough for a list of 10,000 objects since the time it took to find and invert an object after a selection was minimal. Thus, the linked list representation was the data structure of choice.

Note that other data structures could be substituted for the linked list because of the object oriented nature of the implementation. Because the internal representation of the linked list is hidden from all subclasses, faster data structures such as a hash table or tree could replace the linked list representation.

Dependency Graph

Figure 3-9 on the following page shows a dependency graph of all of the classes used within gRNAid. An arrow from TStructureDoc to TTemplateDoc means that TStructureDoc depends on or has knowledge of TTemplateDoc. The dotted arrows imply that the dependency was made for performance reasons only. That is, the dependency could have been removed, resulting in a more object oriented program with the price of slowing down some algorithm. The dependency graph is depicted in a top-down style, where more management oriented classes are near the top of the graph and classes that implement more of the grunge-level details are near the bottom. Note that the base classes (TApplication, TDocument, TScrollDoc, TList, etc.) are not included in Figure 3-9.

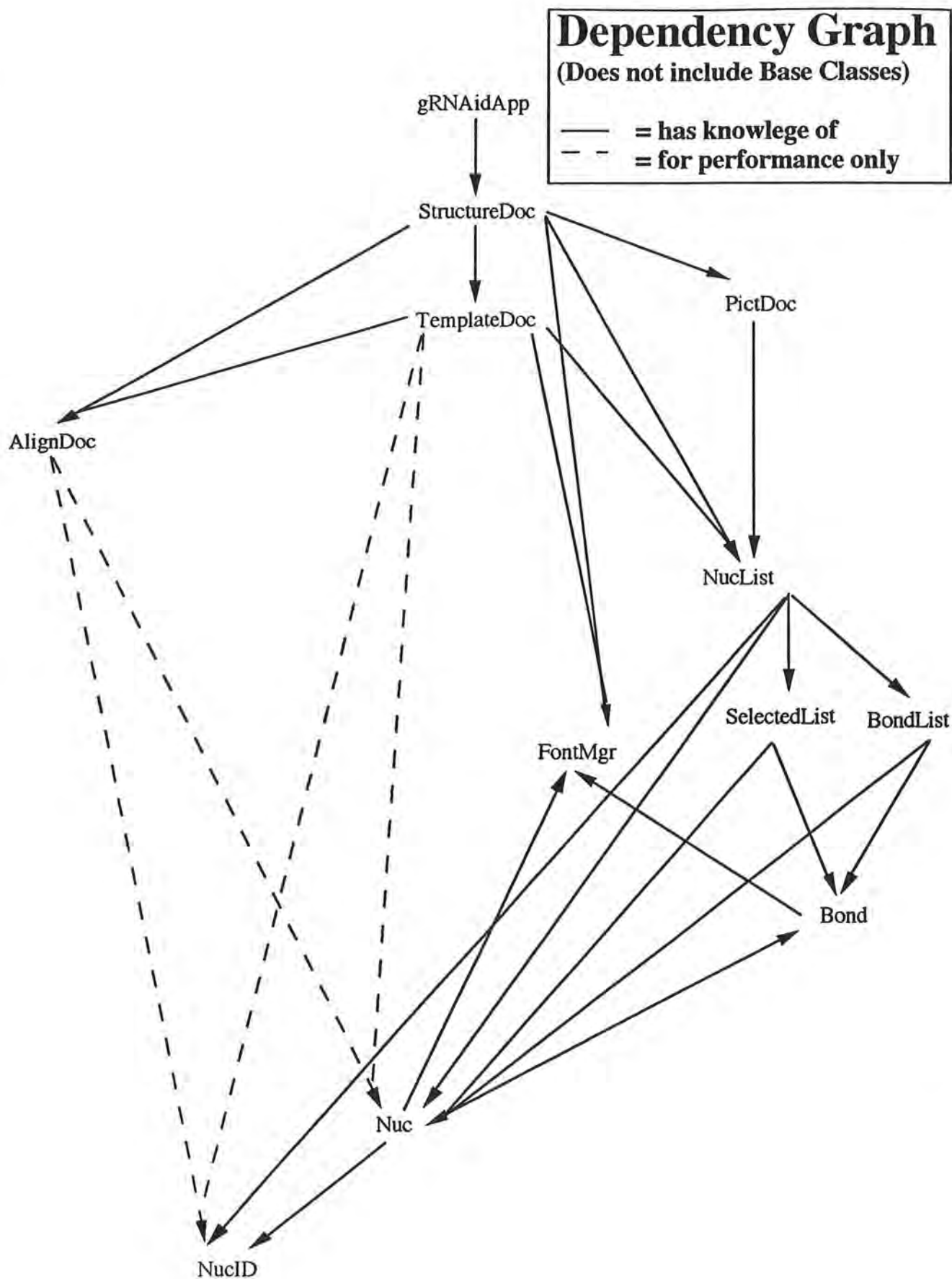


Figure 3-9 gRNAid class dependency graph

Bond Drawing Algorithm

When gRNAid generates a new secondary structure or reads a template file from disk, it must calculate the screen positions associated with the bonds. The template file contains information specifying that two nucleotides are bonded together, but does not give any information on where the bond will reside on the screen. There are three types of bonds that can be formed: Watson-Crick, strong noncannonical, and weak noncannonical. Watson-Crick bonds are represented by straight lines on the screen, while noncannonical bonds are represented by circles. Although strong (*) and weak (o) bonds are drawn differently on the screen, the algorithm used to calculate the screen positions is the same. Thus, there are two algorithms, one for Watson-Crick and one for noncannonical. Because it is easier, we will cover the noncannonical bond forming algorithm first.

Noncannonical

The most complex part of forming a bond is determining where the bond should be placed. Noncannonical bonds, represented by a circle, are drawn differently than Watson-Crick bonds. The first question is, how is a noncannonical bond represented? Noncannonical bonds are either a solid circle (*) between an A and a G, or a hollow circle (o) between a G and a U. The Macintosh toolbox routines PaintOval (which draws the *) and FrameOval (which draws the o) both take a rectangle that surrounds the circle as an argument. Thus, it would be good to save this rectangle as a data member in the parent TNonCannonical class.

The next question is, how is this rectangle calculated? Consider the illustration in Figure 3-10. If a line is drawn from the upper left hand corner of the left hand nucleotide to the lower right hand corner of the right hand nucleotide, the centerpoint of the bond will be at the centerpoint of this line. Thus, to calculate the rectangle, find the centerpoint of the line and then calculate the surrounding rectangle by adding or subtracting some radius.

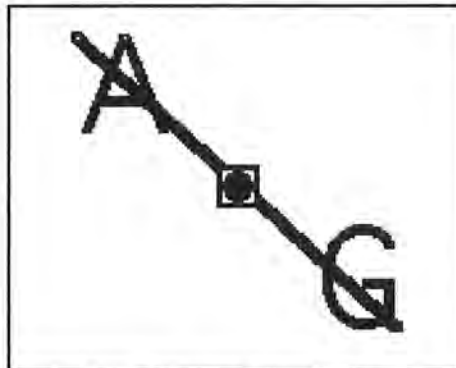


Figure 3-10 Calculating the rectangle for a noncannonical bond

Watson-Crick

The problem of computing Watson-Crick bond screen information for visually attractive bonds was initially complex, but with the use of the object oriented paradigm and a Vector class, the solution became more straightforward. Consider Figure 3-11. We want to draw a Watson-Crick bond from one nucleotide to the other. This means we want to find the two points, A and B, and store them in the TWatsonCrick bond object. Once we find these two points, it is trivial to draw the straight line that goes from A to B.

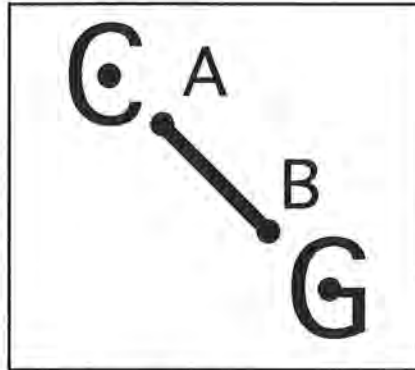


Figure 3-11 Calculation of the line for a Watson-Crick bond

Let the dots in the middle of the C and G represent the centers of the nucleotides. Let vector P reference the dot in the middle of nucleotide G above, and let vector Q represent the dot in the middle of nucleotide C. Figure 3-12 shows these two vectors with some of the details removed for clarity. In this figure, notice the dotted line that goes from Q to P, through points A and B. This line goes in *exactly* the direction that we want the Watson-Crick bond to go. If we could find the vector representing this line, we would be taking a step in the right direction (pun intended). To follow the remainder of this discussion, you may need to get out your vector algebra book.

In Figure 3-12, vector R is vector P minus vector Q ($R = P - Q$). Now we have a vector that goes in the right direction, but it isn't exactly where what we wanted it. Recall that our goal is to find the two points, A and B.

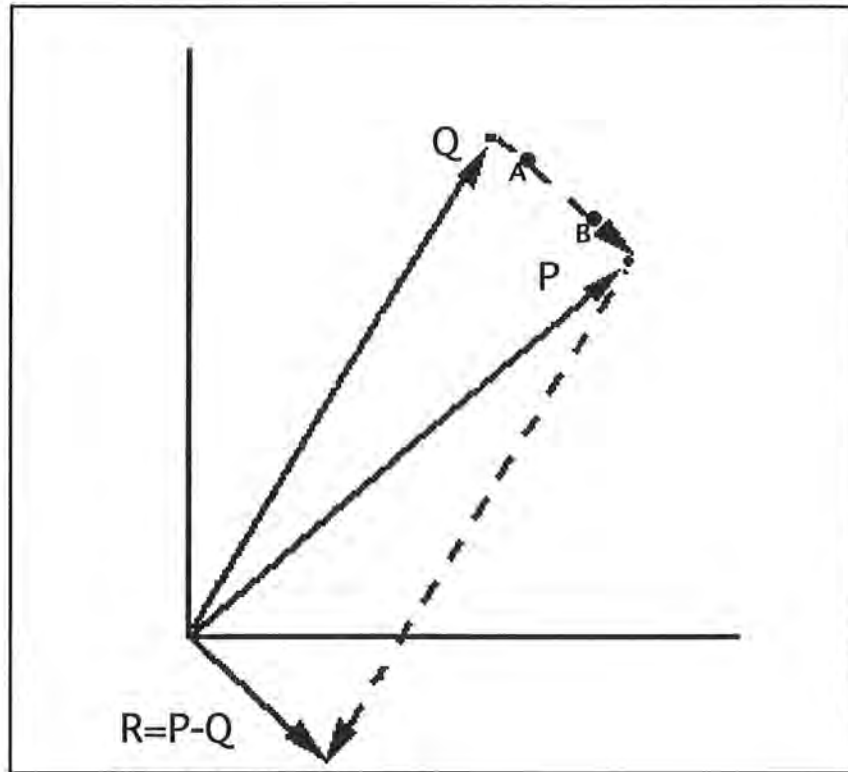


Figure 3-12 Using vectors to calculate the midpoints

Let's first look at trying to find A. Let the length of vector R be equal to L. This also means that the length of the vector from Q to P is L. Now imagine that we have drawn an imaginary circle around the Q vector, as shown in Figure 3-13. If we knew the value of the radius r in Figure 3-13, then we would have it made, since $A = Q + (r/L) * R$. So what is r? Simple -- r is an arbitrary value that can be calculated based on the size of the current font.

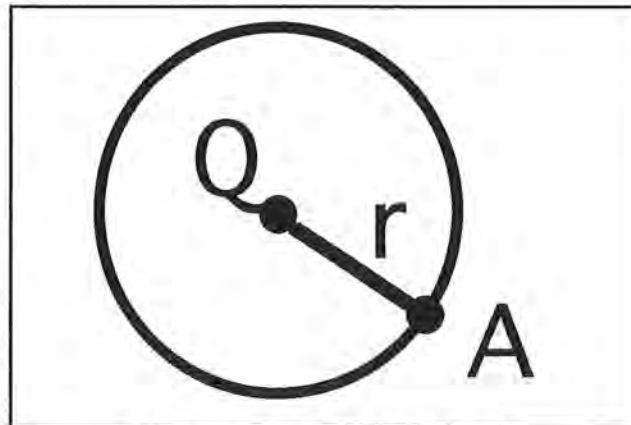


Figure 3-13 Finding A

So there you have it. $A = Q + (r/L) * R$; $R = P - Q$; and L is the length of R. Similarly, to find B, we would use $B = P - (r/L) * R$. Implementation-wise, a vector class was created that had common vector operators such as vector addition and subtraction, scalar multiplication, and an operator to compute the length of a vector. The addition made the code rather elegant and much easier to understand.

Conclusions

One might think that these algorithms could be avoided by placing bond screen positioning information in the template file, and much like nucleotides, bonds could assume screen positions from the template file. However, gRNAid currently does not allow for the manipulation of bonds. That is, the bond objects cannot be moved by themselves and can only be moved with their owning nucleotides. Therefore, it makes sense for gRNAid to calculate the bond coordinates based on the owning base-pair. Thus, when both nucleotides that form the base-pair are moved, the bond is offset to move with the owning bases. When only one of the nucleotides in the base pair is moved, the graphical representation of the bond has a rubberbanding effect, and the coordinates for the bond must be recalculated. Future versions of gRNAid might provide bond editing capabilities, and as such the bond objects could then obtain screen positioning information from the template file. However, because the rubberbanding effect mentioned above would still be a desired behavior, the bond coordinate calculation routines would still be a required part of gRNAid.

Algorithm Analysis

Algorithms within gRNAid are $O(N^2)$ or less (where N represents the number of nucleotides in the sequence), with the majority of the algorithms being $O(N)$. The bonding algorithm described above is $O(1)$ since the operation does not depend on the number of nucleotides or bonds in the secondary structure. Drawing the secondary structure is $O(N + M)$, where N is the number of nucleotides and M is the number of bonds. A nucleotide selection via the mouse is also $O(N)$, since a selection really represents a search of the nucleotide list for the nucleotide that was selected.

The mapping algorithm is more complex to analyze (see Chapter 2 for design level details of this algorithm). The portion of the algorithm that associates screen positions with nucleotides is $O(N)$, because each nucleotide in the nucleotide list must be scanned in order to associate it with a screen position. Most of the work in the mapping algorithm comes from forming the bond. Recall that all `TBond` objects have members `fPartner1` and `fPartner2` (Figure 3-8). These members, which from now on will be collectively referred to as the partner pointers, reference the owning nucleotide objects that form the bond.

Hooking up the partner pointers involves 'finding' both `TNuc` objects in the base pair and then updating the partner pointers to reference the appropriate nucleotide. In the mapping algorithm, the first `TNuc` partner is already available since it has just been constructed. The time intensive task comes in finding the second nucleotide object. Since the nucleotides are stored in a linked list, the only way to find the partner nucleotide is to search the whole list, which is an $O(N)$ task, worst case. Because this search must be done for every base pair, the entire algorithm is worst case on the order of N^2 , although several performance improvements can be made to this algorithm to speed it up. Therefore, the entire mapping algorithm is on the order of N (coordinate mapping) + N^2 (base pairing). On a Macintosh SE30, this mapping algorithm generally takes 20 to 30 seconds when generating a secondary structure for a 16S rRNA with approximately 1600 bases. This is still an improvement over thermodynamic based algorithms that are generally $O(N^3)$. See Chapter 1 for more information on the performance characteristics of other secondary structure creation programs.

Chapter 4

gRNAid User's Manual

This chapter provides a reference to all available gRNAid functionality. First, secondary structure generation is described. Next, the editing operations within gRNAid are outlined, and then the hidden nucleotide bar and the process of mapping hidden nucleotides to screen locations is explained. Finally, each menu item in gRNAid is listed with a description of its function. A tutorial is provided in Appendix A showing how all of these commands work in context with each other.

Generating A Secondary Structure

To create a new secondary structure from scratch within gRNAid, an alignment file and a template file must be available. For the sake of this discussion, assume the alignment file represents the sequence of *Archaeoglobis fulgidus* (from now on referred to as *Archae*) and the template file represents the secondary structure of *E. coli*.

The alignment file represents an alignment of the sequence of *Archae* with the sequence of *E. coli*. Specifically, the file contains the aligned *Archae* sequence and nucleotide numbering information. At this time, gRNAid requires that the alignment file conform to a strict format, and any deviation from this format will result in error. The template file is a representation of a known secondary structure, and the screen coordinate and bonding information from the template file is used in the creation of new secondary structures. Information explaining the ideas behind the alignment files, nucleotide numbering, and template files is in Chapter 2.

The **Generate** menu item from the **Structure** menu is used to commence secondary structure creation. Once this menu item is selected, a file selection dialog is displayed prompting you to select the alignment file of the organism for which you want the secondary structure generated. In Figure 4-1, the alignment file for *Archae* is selected. The gRNAid program displays an informational dialog while processing the alignment file, which is removed from the screen upon the completion of processing this file.

Next, you are prompted to select the template file. In Figure 4-2, the template file for the bacterium *E. coli* is selected. Again, an informational dialog is displayed as the template file is being processed. Once the process is complete, the secondary structure is displayed in a scrollable window. This window contains many of the common artifacts found in other Macintosh applications, including scroll bars, close box, zoom box, title bar, and size box. If you are unfamiliar with these objects, please refer to your Macintosh documentation. A segment of the secondary structure generated from the *Archae* alignment and the *E. coli* template is shown in Figure 4-3.

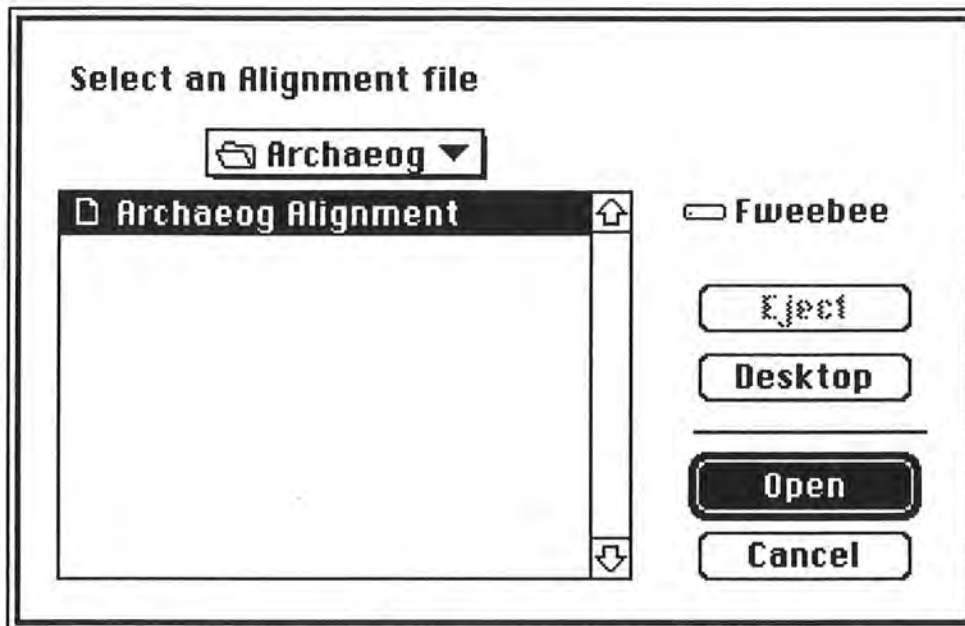


Figure 4-1 File selection dialog used to select an alignment file

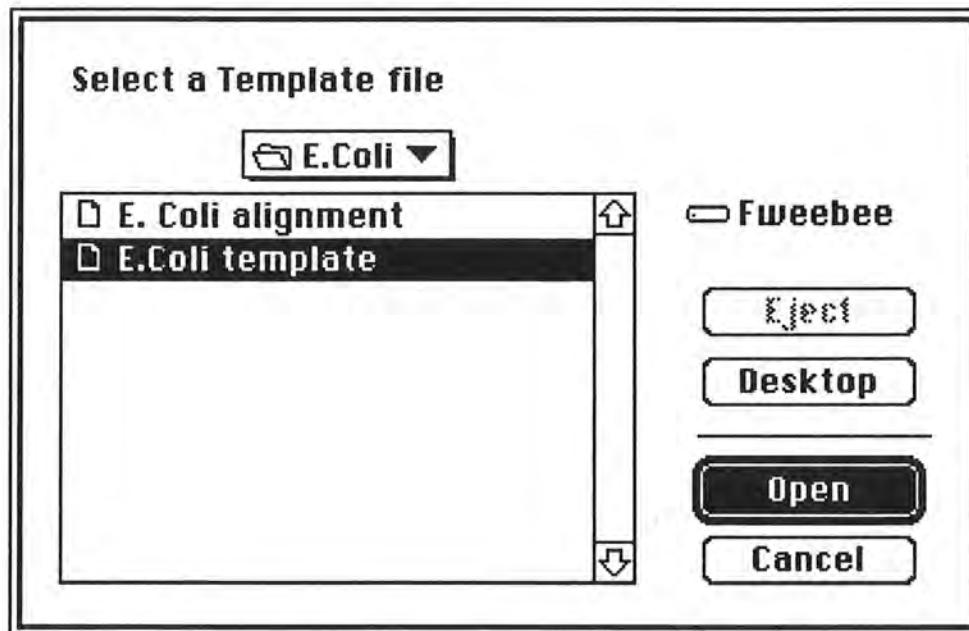


Figure 4-2 File selection dialog used to select a template file

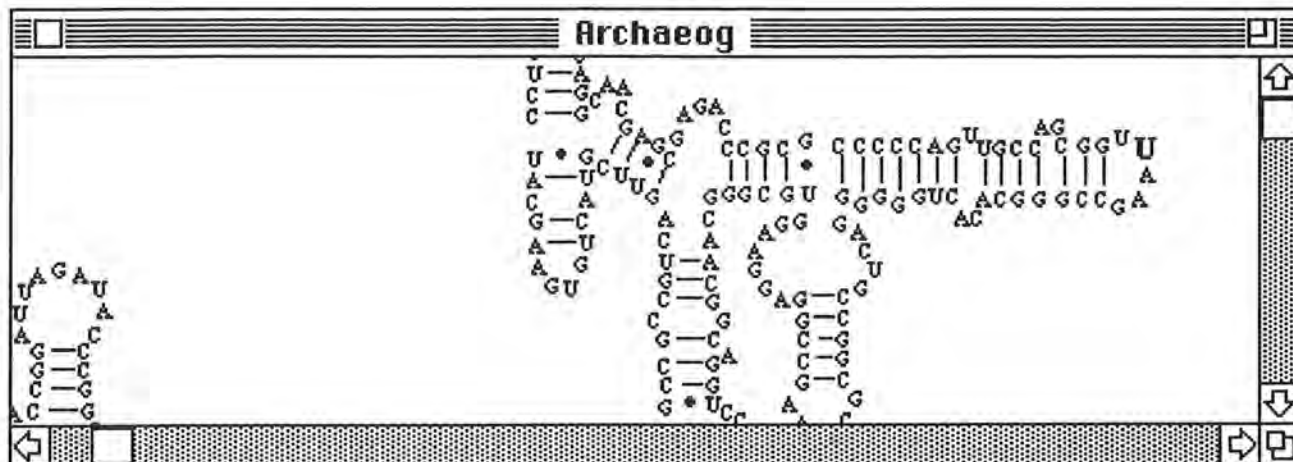


Figure 4-3 A fragment of the *Archae* structure generated with gRNAid

Editing Operations

Selection & Deselection of Nucleotides

A single nucleotide can be selected by clicking the mouse on it, and when the nucleotide is selected, it becomes highlighted. If you click the mouse on another nucleotide, any previously highlighted nucleotide is deselected and the nucleotide just clicked on is selected. To deselect all nucleotides, click the mouse in a blank area of the window.

To select a group of nucleotides, keep the mouse button depressed and drag (Figure 4-4); a selection rectangle is displayed as you drag and when you release the mouse button, all nucleotides that overlap with this rectangle will become highlighted. Again, a mouse click on any non-nucleotide will result in a deselection of all previously selected nucleotides.

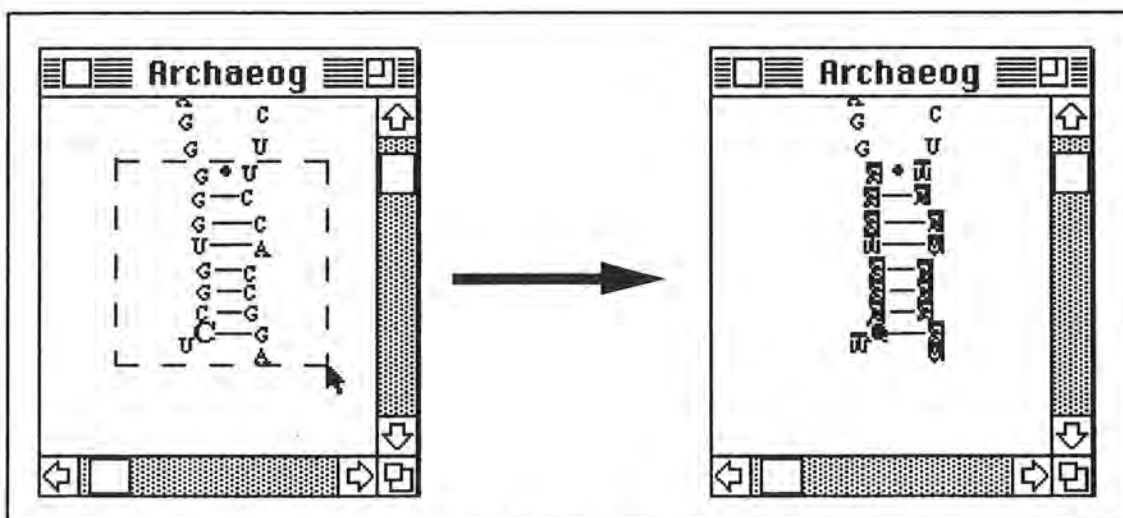


Figure 4-4 Selecting a group of nucleotides

Sometimes you may want to select or deselect another nucleotide without affecting anything else that has been selected. For example, in Figure 4-4, you may want to select several of the unselected nucleotides while keeping all of the other nucleotides selected. To do this, press down on the **shift** key while clicking the mouse on a nucleotide. The **shift-click** mechanism acts as a toggle: if you click on a selected nucleotide, then it will deselect. Otherwise, a click on a deselected nucleotide will result in it being selected. The **shift-click** mechanism allows you to easily add and remove nucleotides from the selection group without affecting any of the nucleotides that are already selected.

Dragging

Once one or more nucleotides have been selected, they can be dragged. The selection and drag of a single nucleotide can be done with a single mouse click. Depress the mouse button when on the nucleotide, move the mouse to the new location for the nucleotide, and release the mouse button. The nucleotide will be offset so that it now resides at the new location. One or more nucleotides can be dragged by clicking the mouse on any selected nucleotide and then dragging. While dragging, an outlined region of what you are dragging will appear as an aid, as shown in Figure 4-5.

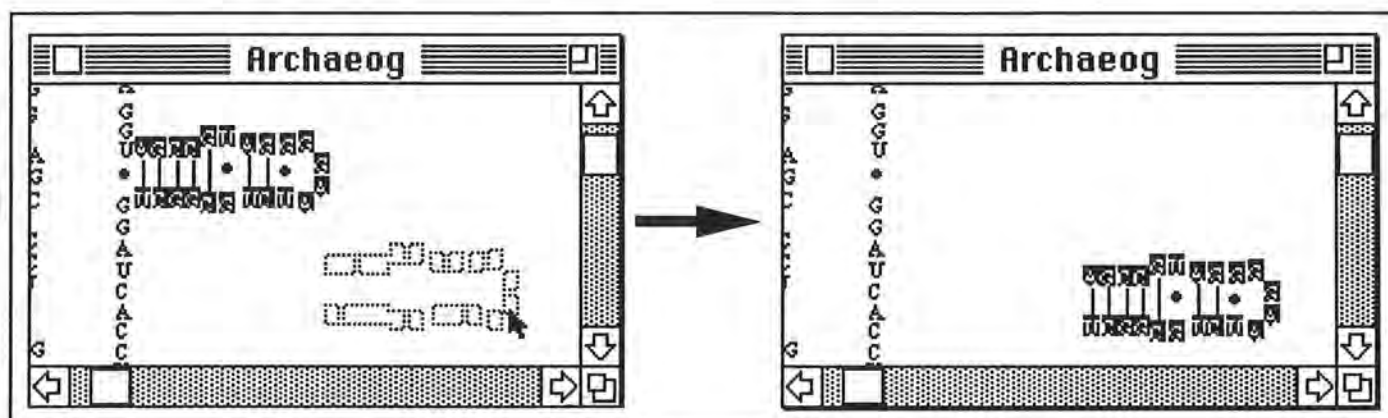


Figure 4-5 Dragging a group of selected nucleotides

At this time, a drag is not allowed if any of the outline region is moved outside of the editing area of the window. As shown in Figure 4-6, a portion of the outline region is moved outside of the window, which gRNAid considers to be an illegal drag. To get around this, make the window bigger or incrementally drag the segment and scroll the window to move the segment to its final position.

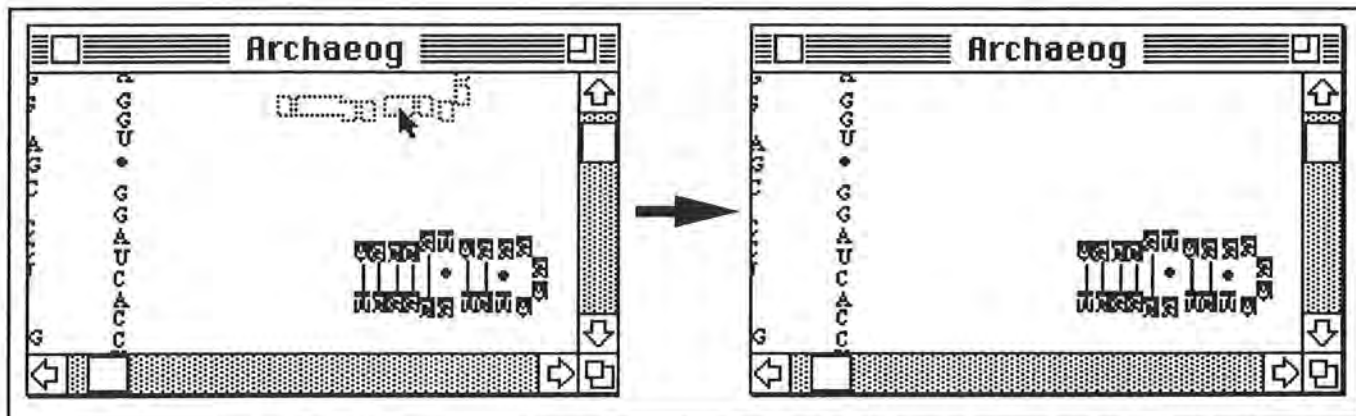


Figure 4-6 It is illegal to try to drag outside of the legal bounds of the window.

Selection & Dragging of Bonds

A bond cannot be edited directly but is considered to be owned by the base-pair that forms the bond. If both of the nucleotides that make up the base pair are selected, then any drag on this base pair will result in the bond being dragged also. Notice in Figure 4-5 that the bonds all moved with their parent base-pair. The topic of dragging bonds brings up an interesting question: what happens when only one of the nucleotides in the bond is selected and dragged to a new position? As shown in Figure 4-7, a rubberbanding of the bond occurs. The rubberbanding of the bond is more obvious with Watson-Crick bonds than with noncanonical bonds. When the noncanonical bond is rubberbanded, the dot is placed central to the two owners, and it is not always immediately obvious which nucleotide pair owns the bond.

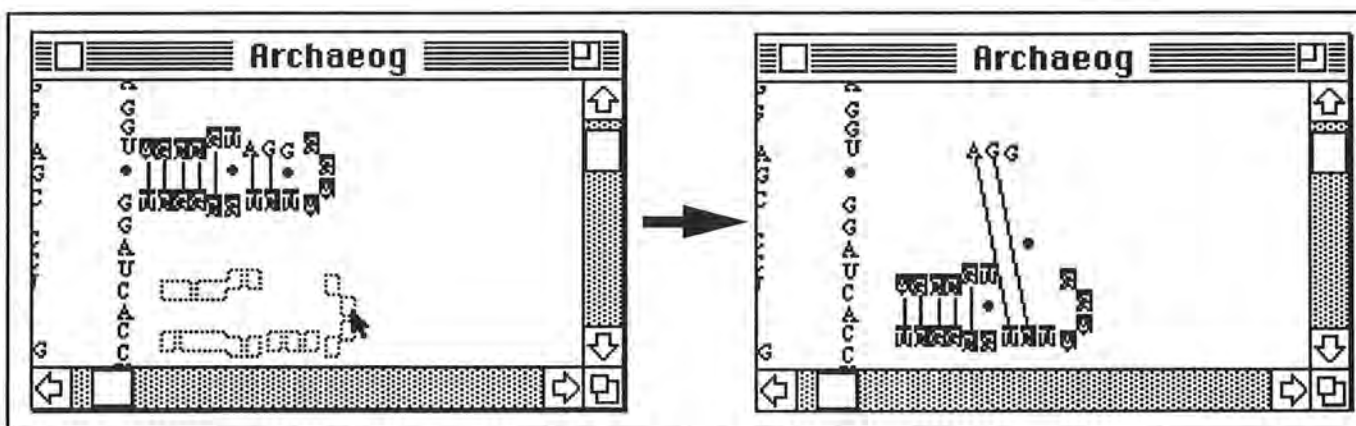


Figure 4-7 A rubberbanding effect occurs when only one nucleotide in the base pair is dragged

The Hidden Nucleotide Bar

Notice that the secondary structure displayed in Figure 4-8 contains a selected bold nucleotide, U. This bold nucleotide represents an area in the secondary structure where gRNAid did not map one or more nucleotides to screen positions. Thus, these nucleotides are hidden and must be mapped to screen positions by the user. When a bold nucleotide is selected, an informational bar will appear at the top of the window. This bar is referred to as the hidden nucleotide bar and gives three pieces of information: 1) the number of nucleotides that are hidden, 2) the sequence of nucleotides that are hidden, and 3) some help text (**ÆM to map**) indicating the command that is needed to begin mapping hidden nucleotides. In Figure 4-8, the bold U is selected and tells you that 10 nucleotides, CCCUUCGGGG, have not been mapped to screen positions. The hidden nucleotide bar is only displayed when a single, bold nucleotide has been selected. Thus, if more than one nucleotide is selected, the hidden nucleotide bar will not be displayed.

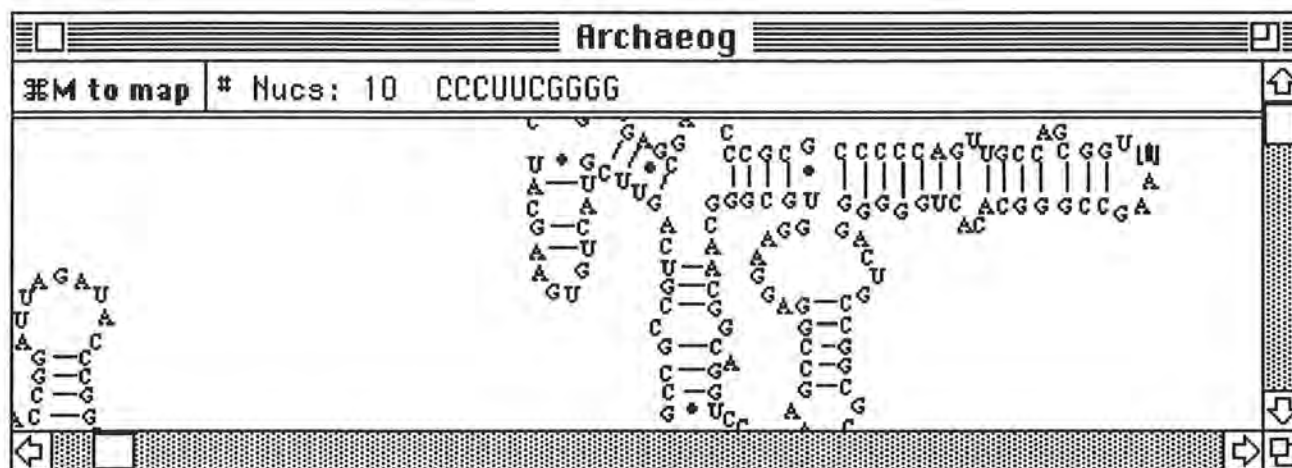


Figure 4-8 The hidden nucleotide bar is displayed when a single, bold nucleotide is selected

Mapping Hidden Nucleotides

The process of mapping hidden nucleotides means to associate screen positions with hidden nucleotides that failed to obtain a screen position. That is, an X,Y screen position is associated with a hidden nucleotide by the click of the mouse at the desired location in the secondary structure window. After the click, the previously hidden nucleotide becomes visible at the X,Y coordinates obtained by the mouse click.

In gRNAid, it is only legal to map hidden nucleotides when a single, bold nucleotide has been selected. When this condition is not met, the hidden nucleotide bar will not be displayed and the mapping commands will be disabled. For the remainder of this section, it is assumed that a single, bold nucleotide has been selected, thus enabling the mapping commands.

Map Mode

Before examining the mapping process closely, it is useful to differentiate between map mode and normal editing mode. In normal editing mode, a single mouse click in a window denotes the selection, deselection, or dragging of one or more nucleotides. There has to be some way to determine when we want to change what a single mouse click means. While mapping nucleotides to screen positions, a mouse click should be used to associate screen positions with hidden nucleotides. Because the behavior of a mouse click is now ambiguous, a new mode called map mode is introduced. When you enter map mode, the mouse click will be interpreted as associating screen positions with hidden nucleotides. When in normal editing mode, a mouse click is used for selection, deselection, and dragging operations.

In gRNAid, the **Map Hidden** and **Exit Map** menu commands from the **Nucleotide** menu (Figure 4-9) can be used as a toggle from one mode to the other. Note that these menu commands appear at the same location in the **Nucleotide** menu and as such are never available for selection at the same time. You can also toggle between modes by using the **⌘M** keyboard equivalent. Thus, if you are currently in normal editing mode, you can select the **Map Hidden** command to enter map mode, which will in effect change the text of the menu to **Exit Map**. If for some reason you decide to exit map mode before mapping all hidden nucleotides to screen position, you can select the **Exit Map** menu item, which will toggle you back to normal editing mode and change the text of the menu item to **Map Hidden**.

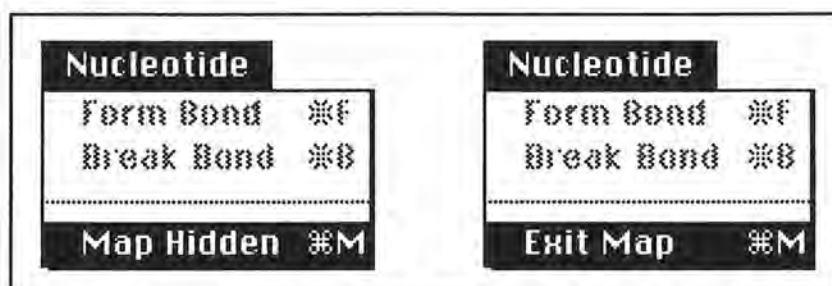


Figure 4-9 Using menu command to toggle between normal and map mode

The current mode you are in can be determined in several different ways. When map mode is entered, the cursor will change to a crosshair when the mouse is in the window. (Note: the mouse changes to an arrow when it is moved outside of the editing region of the window; it changes back to the crosshair when moved back into the window.) When in normal editing mode, the mouse will be an arrow. A second indicator of the current mode is in the hidden nucleotide bar (Figure 4-4). The help text that appears at the upper left-hand corner of the bar is context sensitive: if you are in map mode, the help text will be **⌘M to Quit**, and when in normal editing mode the help text will be **⌘M to Map**. A third sign indicating mode is in the **Nucleotide** menu commands, where the text of the mapping menu item differs depending on the current mode.

Mapping

In Figure 4-8, a sequence of ten hidden nucleotides needs to be mapped to screen positions on the 3' side of the bold U. (Note: if you are not familiar with the 3' and 5' concepts, see Chapter 2.) In Figure 4-4, the 3' side is on the side next to the A, so this is the side where we would want to insert the ten hidden nucleotides.

It is useful to analyze the hidden nucleotide sequence to determine its structure before mapping the nucleotides to their screen positions. For example, the hidden nucleotide bar in Figure 4-4 shows that there is an insertion of 10 nucleotides (CCCUUCGGGG). With a little analysis and some scratch paper, it can be conjectured that the structure of the segment looks something like that shown in Figure 4-10, where the hidden nucleotides are bold.

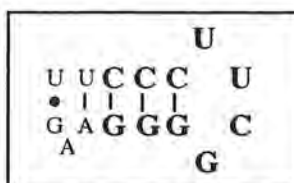


Figure 4-10 The nucleotides shown here in bold represent the 10 hidden nucleotides that need to be mapped to screen positions.

Generally, the structure must be edited to make room for the hidden nucleotides, as shown in Figure 4-11.

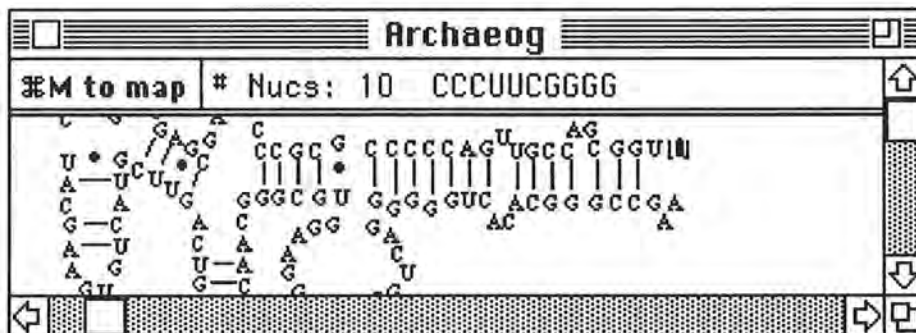


Figure 4-11 Editing the structure to provide room for the hidden nucleotides (compare to Figure 4-8)

The sequence of steps that one must go through to map nucleotides to screen positions is as follows. First, map mode must be entered by selecting the **Map Hidden** menu option or using the ⌘M keyboard equivalent. The help text in the hidden nucleotide bar will change to **⌘M to Quit**. The cursor will then change to a crosshair (+). This cursor will change to an arrow when the mouse is moved outside of the editing area of the window, and will resume its shape as a crosshair when moved back within the window. To associate an X,Y coordinate with the first hidden nucleotide, click the cross-hair at the general location where the first hidden nucleotide is to be placed. In Figure 4-12, the first hidden nucleotide, C, has been mapped to a screen position.



Figure 4-12 Mapping the first hidden nucleotide to a screen position

Once the nucleotide has been mapped to a coordinate, the hidden nucleotide bar is updated - the number of nucleotides that remain hidden is decreased by one and the mapped nucleotide is removed from the list of invisible nucleotides. Also, the newly mapped nucleotide becomes the current bold nucleotide, and the previously bold nucleotide is returned to a normal state. Repeat this process of clicking the cross-hair in the window to map the remaining nucleotides to screen positions. When the last hidden nucleotide has been mapped, the hidden nucleotide bar is removed and no bold nucleotides remain in this segment of the sequence. Figure 4-13 shows the result of mapping the 10 hidden nucleotides to their screen positions.

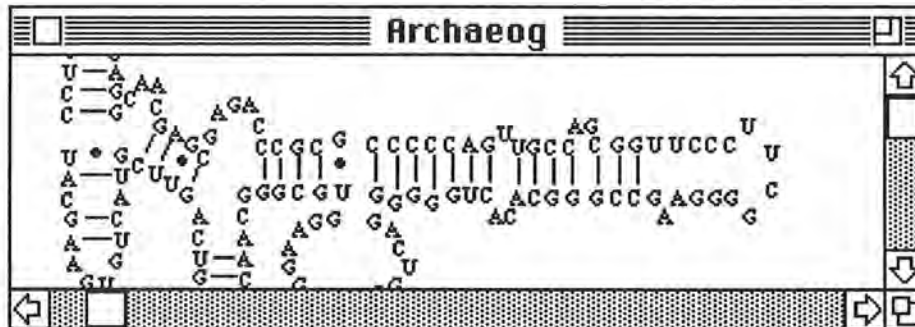


Figure 4-13 The result of mapping all ten hidden nucleotides

Exiting Map Mode


You may not want to map all hidden nucleotides to coordinates at one time. For example, there may be 100 hidden nucleotides that need to be mapped and you may want to map these in 20 nucleotide increments. To exit map mode, at any time select the **Exit Map** command from the **Nucleotide** menu or use the $\%M$ keyboard equivalent. This will toggle you out of map mode and into normal editing mode. A bold nucleotide will represent the place where you stopped mapping the nucleotides, so to resume mapping all you need to do is select this nucleotide and re-enter map mode.

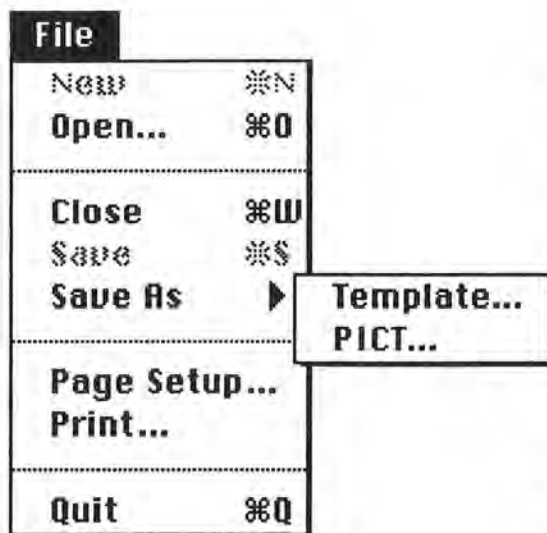
Forming And Breaking Bonds

Many times a bond may need to be removed or created from a newly created portion of the secondary structure. For example, in Figure 4-13, there are several bonds to be formed. To create the bond, select the two nucleotides that are to form the base pair, then select the **Form Bond** command from the **Nucleotide** menu. Only Watson-Crick and noncannonical bonds may be formed. To remove an existing bond, select both nucleotides that form the base pair and then select the **Break Bond** item from the **Nucleotide** menu.

Menu Commands

The remainder of the chapter is devoted to explaining each of the menu commands available within gRNAid. Many of the menu items have a command key equivalent (⌘) that can be used to invoke the command.

The Apple menu contains all desk accessories that have been installed under the  menu as well as the **About gRNAid...** menu item, which provides general information on the gRNAid program.



New

This command is not used and should be removed from gRNAid in the near future.

Open...

When this menu item is selected, you are prompted to select any legal template file that has been previously saved to disk. Recall that the template file is the textual representation of the secondary structure. Only legal template files should be opened with this command. Once the template file has been opened, the secondary structure represented by the file is displayed in a window.

Close

This command closes the currently active window. If any editing operations have been made to the structure since the last save, you will be asked if you would like to save the new changes. This command is only available when windows are open within gRNAid.

Save

Use this menu item to save the current contents of the window. This menu item is only enabled if any changes or editing operations have been performed in the window since the structure was last saved. Note that a simple mouse click in the window will enable the **Save** menu item. If the contents of the window have not previously been saved, you will be prompted to enter a name for the template file, similar to the **Save As...** command. If the template has been saved before, then the template file will be updated to include any new secondary structure information since the last save.

Save As

If the **Save As Template** command is selected, then the currently active secondary structure window is saved to a new file and the name of the active window is changed to the name of the new file. All subsequent file related operations will occur on the new file. If the **Save As PICT** command is selected, then the file is saved to a PICT format, which is readable by other PICT reading programs such as MacDraw®. The **Save As PICT** command does not have any affect on the currently active structure. These commands are deactivated when there are no open windows.

Page Setup...

When this command is selected, a printer-specific dialog will be displayed allowing you to set printer-specific characteristics such as page layout and orientation. This dialog will vary for different models of printers. Use this command before printing to set up printer-related characteristics. This command is available only when there is a structure available for printing.

Print...

This command will print the secondary structure in the currently active window. A dialog is displayed allowing you to set page-layout specific information before printing. This dialog will vary depending on the printer you are currently set up to use. Note that only one page will be printed, no matter how large the secondary structure may be. If the secondary structure in the currently active window is larger than one page size, gRNAid will shrink the image so that it will fit into one page. This command is disabled when there are no open windows within gRNAid.

Quit

When this command is selected, you will be queried to save any windows that need saving and then the application will exit.

The **Edit** menu is provided for desk accessory compatibility and nothing more. Many desk accessories rely on applications to provide a **File** and **Edit** menu that they can use while running. The gRNAid program itself does not use any of the **Edit** menu functions and as such they will always be disabled while gRNAid is being used

Edit	
Undo	⌘Z

Cut	⌘K
Copy	⌘C
Paste	⌘V
Clear	

Select All	

Structure	
Generate...	⌘G

Show Backbone	⌘K
Redraw	⌘R

Generate...

This command will create a secondary structure based on an alignment file and a template file. First, you are asked to provide the alignment file, which represents the aligned sequence of the organism for which a secondary structure will be created. You are next prompted for a template file, which contains the known secondary structure of some other organism. It is assumed that the alignment file represents an

alignment of the new bacterium with the bacterium represented by the template. Once these two files have been entered, gRNAid performs a mapping algorithm to generate a partial secondary structure. Hidden nucleotides, which occur at areas where there are bold nucleotides in the secondary structure, should eventually be mapped to screen positions (see the *Mapping Hidden Nucleotides* section of this chapter for more information).

Show Backbone

This menu item provides a graphical means of looking at primary structure. While editing a secondary structure, errors may occur when nucleotides get moved to incorrect locations. When the backbone is activated, a line is drawn from each nucleotide to its neighbors. Areas of chaos in the window denote areas where the primary structure is incorrect. The **Show Backbone** command is a toggle: if the backbone is currently displayed and this menu item is selected, the backbone will be removed. Conversely, if the backbone is not displayed, the selection of this menu item will result in the display of the backbone. Figure A-21 in Appendix A shows an example of the backbone. This command is not active when there are no open windows.

Redraw

Occasionally, some debris may be left in the window after an editing operation, and this command can be used to clean up the window. This command is not active if a window is not open.

Form Bond

This menu item is only enabled if two nucleotides have been selected and they are bondable. That is, only Watson-Crick and noncannonical bonds are allowed within gRNAid and as such, if two A's have been selected, this menu item will be disabled. Once the two legally bondable nucleotides have been selected, this option can be used to draw the correct bond type between the base-pair. Note that once the bond is drawn, it cannot be selected or edited.

Nucleotide	
Form Bond	⌘F
Break Bond	⌘B

Map Hidden	⌘M

Break Bond

This command is disabled if two bonded nucleotides have not been selected. The result of this command is to break the bond and remove it from the window.

Map Hidden/Exit Map

This menu is enabled only when a bold nucleotide has been selected and is used to map hidden nucleotides to screen positions. Bold nucleotides in the window represent areas where there are one or more nucleotides that have not been mapped to screen positions. This menu option acts as a toggle; to map nucleotides, select the **Map Hidden** menu item. The text of the menu item then changes to **Exit Map**, which you could select at any time to exit map mode. The process of mapping nucleotides to screen positions was explained in the *Mapping Hidden Nucleotides* section of this chapter.

Selecting any item from this menu will change the font used for all nucleotides in the secondary structure of the currently active window. The default font is **Times**. All fonts that have been installed into your system will be displayed in the **Font** menu. This menu item is disabled if there are no windows open within the gRNAid application.

Size

- 6 pt
- 7 pt
- 8 pt
- ✓ 9 pt
- 10 pt
- 11 pt
- 12 pt
- 14 pt
- 18 pt
- 24 pt

The menu options provided in the **Font Size** menu can be used to change the size of font in the currently active window. All font related operations take several moments to complete because all bond positions must be recalculated. This menu item is available only when at least one secondary structure window is open within gRNAid.

Font

- Chicago
- Courier
- Geneva
- Helvetica
- London
- Los Angeles
- Monaco
- New York
- San Francisco
- ✓ Times

Chapter 5

Conclusions

Results

As gRNAid was being developed, small template and alignment files were created to test various parts of gRNAid functionality. While gRNAid did in fact work well with small template files (representing small portions of a secondary structure), there was uncertainty concerning how gRNAid would work with large, 1600 base sequences. The first real template created was for the 16S rRNA of *Anacystis nidulans*, and this bacteria was selected rather than *E. coli* because its secondary structure was available in computer form, which was used as an aid in template file generation. The creation of the first template file was monotonous and time consuming since screen positions needed to be associated with every single nucleotide in the *Anacystis* sequence.

Once the *Anacystis* template file was complete, it was immediately used to create the secondary structure for *E. coli* (Figure 5-1). The result of the generation turned out amazingly well given that it was the first time the mapping algorithm was attempted on a large, 16S rRNA sequence. The secondary structure shown in Figure 5-1 is considered to be incomplete because some editing needs to be performed to create a complete secondary structure. For example, refer to the **1** in Figure 5-1. In the *Anacystis* secondary structure, this fragment represented a hairpin loop. The obvious hint that it is not correct for *E. coli* is the lack of all the bonds. Furthermore, the presence of the bold **G** nucleotide in **1** indicates an area where there are 18 hidden nucleotides that have not been mapped to screen positions. Number **2** shows another area of insertion, where 22 nucleotides need to be mapped to screen positions. The **3** represents an area of deletion, where nucleotides in *E. coli* did not map to nucleotides in the *Anacystis* template. This part of the structure must be edited so that the **4** nucleotides form the end of the hairpin loop. This means that the G•U strong noncanonical bond would have to be removed. Finally, the **4** shows several nucleotides that were not bonded together in the *Anacystis* structure but need to be bonded in the *E. coli* structure. Refer to Figure 1-1 in Chapter 1 to see the *E. coli* secondary structure after the editing operations have been applied.

Once the *E. coli* secondary structure was complete, it was saved to a template file and used in the generation of 9 other secondary structures that have different levels of 'relatedness' to *E. coli* rRNA. For example, the *mouse* 18s small subunit rRNA is not as close evolutionarily to *E. coli* as would be another bacterial small subunit rRNA, and thus the secondary structure created for *mouse* (Figure 5-2) is not as complete as the ones created for bacteria. For example, areas **1**, **2**, and **3** in Figure 5-2 are areas where there are 117, 30, and 172 hidden nucleotides respectively. Segments **4** and **5** show where there are deletions in the mouse sequence. Obviously, it would take more editing effort to transform Figure 5-2 to the complete *mouse* secondary structure. However, once this secondary structure was complete, it could be used as a template in generating secondary structures for other organisms that are evolutionarily close to *mouse*.

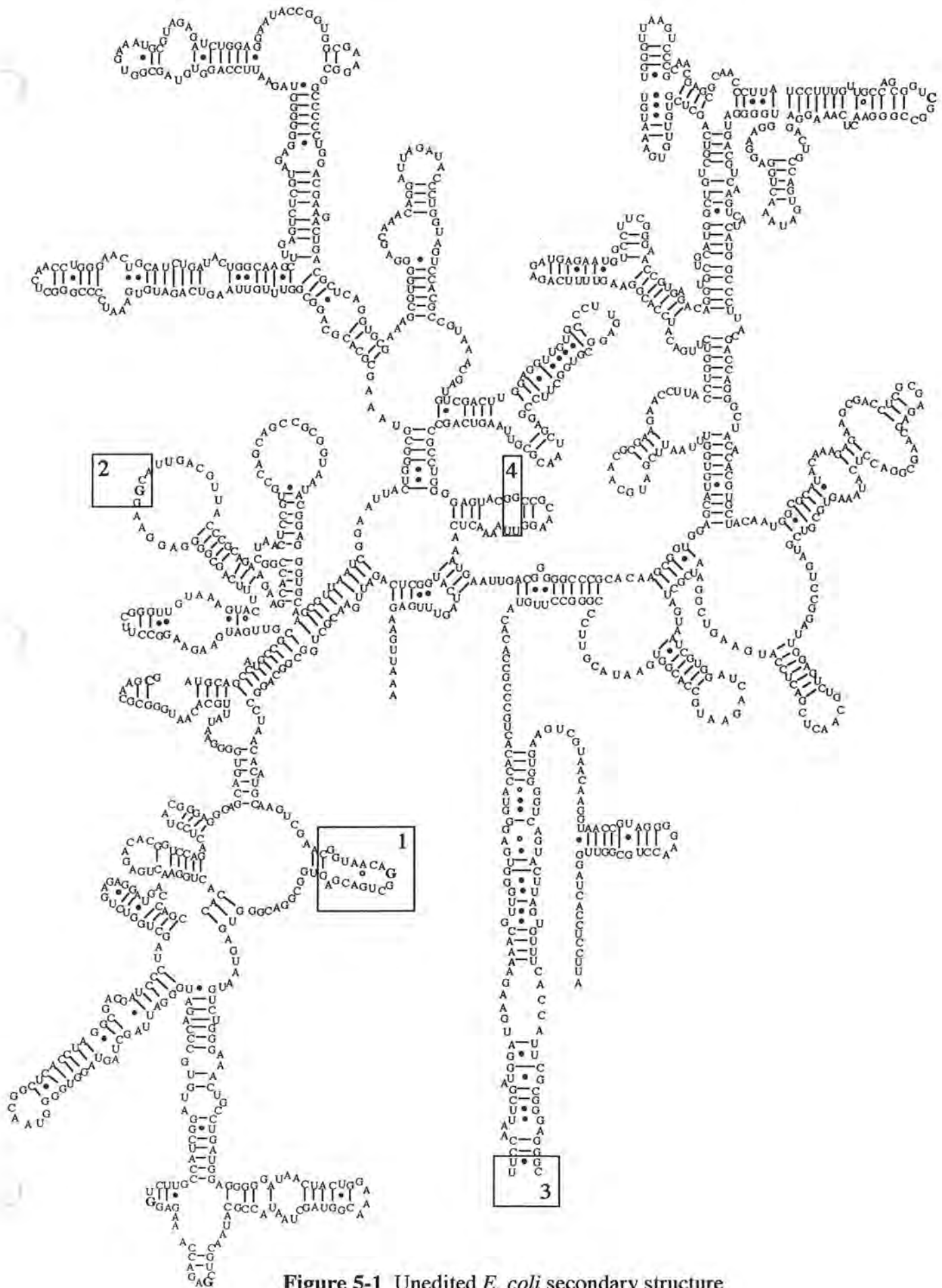


Figure 5-1 Unedited *E. coli* secondary structure created from *Anacystis nidulans* template

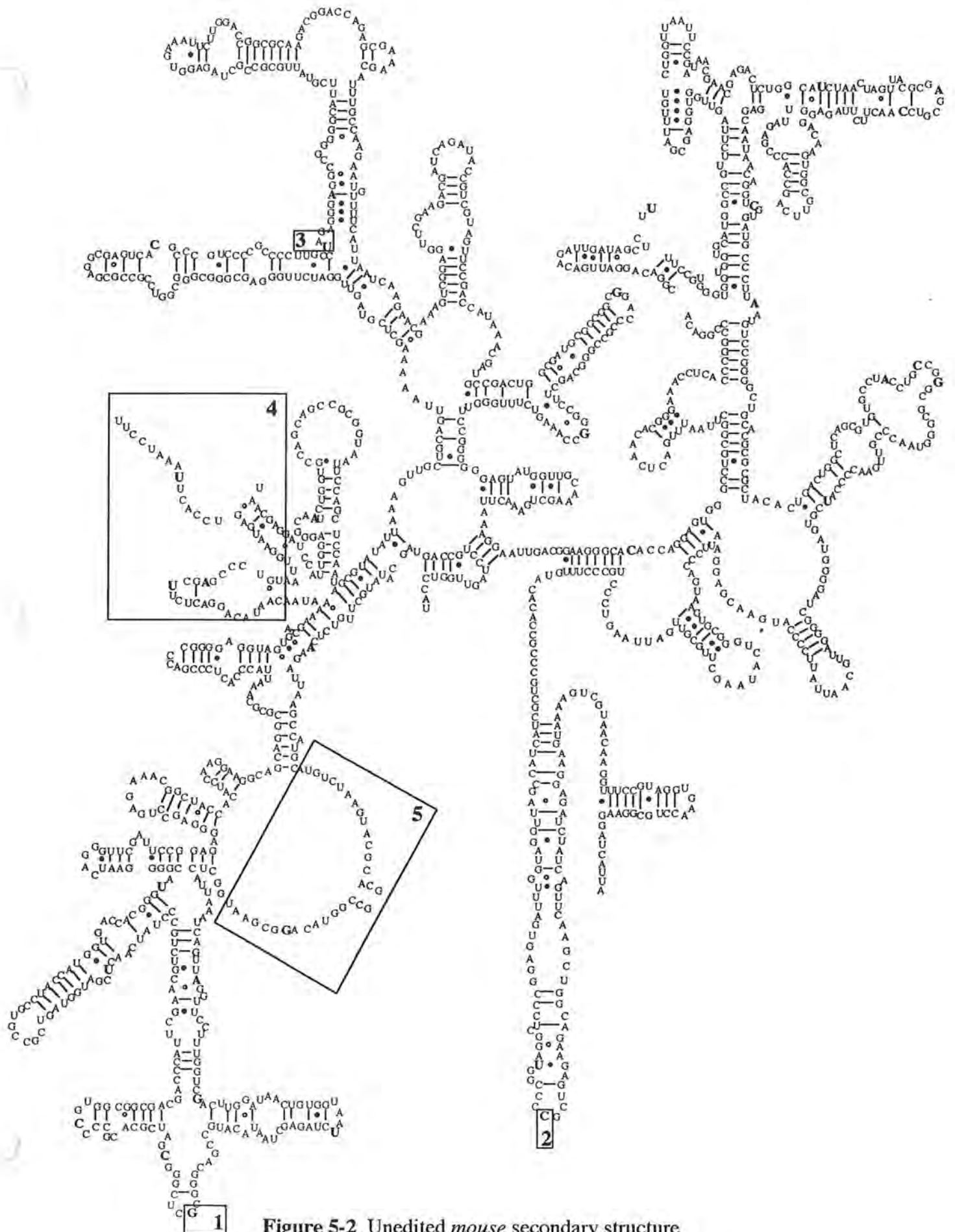


Figure 5-2 Unedited mouse secondary structure created from *E. Coli* template

Analysis of Results

A secondary structure created with gRNAid is only as good as the template and alignment file used in the generation process. Errors in the template file can be passed on from structure to structure, and thus it is a priority to create a correct template file before using it to generate other secondary structures. For example, it is easy to unintentionally swap two adjacent nucleotides, as shown in Figure 5-3. In part A of Figure 5-3, the G and the A at the top of the hairpin loop have been swapped. The Show Backbone functionality does not make it obvious that an error has occurred until the two offending nucleotides are moved further apart (part B). Another template file error occurs when a bond is left out of the template file. These template file errors need to be caught early or the errors will be transmitted to other secondary structures.

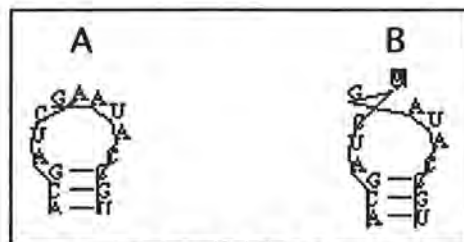


Figure 5-3 A - Two nucleotides (G & A) have been swapped, but are not easily detected via Show Backbone. B - The error is more obvious when the nucleotides have been moved further apart.

Alignment files can contain errors in sequence and alignment. When the sequence is incorrect (for example, the sequence should be AUUGC but is in the alignment file as AAUGC), the secondary structure will obviously be incorrect. When alignment is faulty, incorrect nucleotide numbers will be assigned and the mapping process will be erroneous. For example, in Figure 5-4, the correct alignment shows two C nucleotides at IDs 70 and 71 respectively. Notice that each of these nucleotides is bonded with another nucleotide. The incorrect alignment shows the same two C nucleotides mapped to IDs 69.1 and 69.2. Thus, in the faulty alignment, the C nucleotides are mapped to erroneous screen positions and are not bonded with any other nucleotides.

Correct alignment		Incorrect alignment	
Template file			
	68.0	535	493 U 21.0
	69.0	542	498 U
	69.1	545	502 C
	69.2	550	510 G
	70.0	555	516 C 13.0
	71.0	560	523 G 12.0
	72.0	566	533 G
	72.1	572	539 A
	72.2	581	543 U
	73.0	590	550 C
thousand		thousand	
hundreds		hundreds	
tens	66666777	777	66666 777777
units	56789012	345	56789 012345
bacteria	AACGGCCGCUCUU		bacteria AACGGCCGCUCUU

Figure 5-4 How incorrect alignment affects the mapping process

Insertions in Insertions

The template file produced by gRNAid is dependent on the alignment file from which it was created, mostly because of the nucleotide ID information that it gets from the alignment file. This dependency causes a problem within the mapping process. The following example will illustrate the problem.

Assume that we have already created a complete template for bacterium "A". A fragment of the alignment file and the corresponding template file is shown in Figure 5-5. The template file relies on the correctness of the alignment file to determine nucleotide numbering information, and the secondary structure creation process that gRNAid uses revolves around the correctness of these nucleotide ID's.

Alignment		Template	
thousand		71.0	A 300 300
hundreds		72.0	A 305 295
tens	77 77	72.1	U 310 290
units	12 34	72.2	U 315 285
bact A:	AAUUCG	73.0	C 320 280
		74.0	G 325 275

Figure 5-5 Alignment and template file fragment for bacterium A

Now assume that we have another bacterium, "B", for which we wish to generate an rRNA secondary structure. We want to use the template file for bacterium A to create the secondary structure for bacterium B. The first step is to align the nucleotides from bacterium B with bacterium A. A fragment of the alignment for bacterium B is shown in Figure 5-6.

Alignment	
thousand	
hundreds	
tens	77 77
units	12 34
bact B:	CGACCCGCC

Figure 5-6 Alignment of bacterium B

There is a subtle problem that is depicted in Figure 5-7. The first U in bacterium A is aligned with the first A in bacterium B. However, the next U in bacterium A aligns with the second G in bacterium B. This is referred to as an insertion within an insertion, and gRNAid cannot handle this situation correctly. The underlying problem is that the alignment for bacterium A has implicitly changed (Figure 5-8) to accommodate the new insertions into bacterium B. However, the template file for bacterium A is still based on the old alignment file shown in Figure 5-5. Thus, when gRNAid reads bacterium B's alignment file, it will correctly map the first A in the alignment to 72.1 in the template, and incorrectly map the next nucleotide (C) to 72.2 in the template. The correct mapping process would be to *not* map the CCC sequence fragment in bacterium B to any ID's in the template (thus making them invisible), and to map the last G in the sequence fragment to 72.2 in the template.

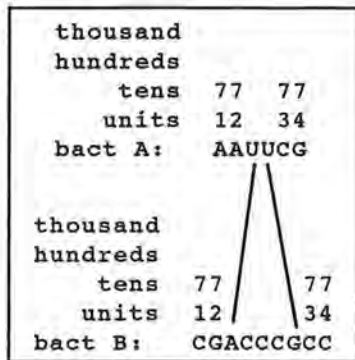


Figure 5-7 Insertions in insertions

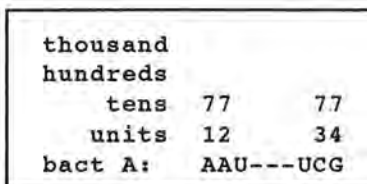


Figure 5-8 Implicit alignment for bacterium A

A work around for this problem might be to recreate the bacteria A template using the "new" alignment shown in Figure 5-8. This is probably more effort than it is worth. Note that this problem only happens in insertions within insertions, and will *never* happen when a bacteria is aligned correctly with *E. coli*, since there are no insertions in *E. coli*.

Further Areas of Study

The insertions within insertions problem is not a hard one to solve and could be implemented in gRNAid in a future version. However, the solution to the problem would make the user of gRNAid do a little bit of extra work. Here's how. A menu option, called **Renumber...** for example, could provide functionality to renumber nucleotide IDs. This menu option would be used when the alignment file associated with the template file has changed because of its alignment with another bacteria. When this menu item is selected, the user would input to gRNAid the new alignment file and the old template file. The gRNAid program would then renumber the nucleotide IDs based on the new alignment information. A simple solution, but it forces the user to make an extra effort in secondary structure generation.

The gRNAid program could also be enhanced by giving it functionality to try to guess structure at areas of insertion. For example, a thermodynamic or palindrome based algorithm could be used to try to predict structure where there are hidden nucleotides. A possible problem with this approach could be with trying to associate screen positions with the newly predicted structure. For example, refer to **3** in Figure 5-2, which is an area where there is an insertion of 172 nucleotides. Notice that this 172 base insertion is not at the end of a hairpin loop, but right in the middle of two other pieces of structure. Do these existing pieces need to be moved in order to make room for the hidden nucleotides? Probably, and determining where these nucleotides should be moved and what screen positions to associate with the hidden nucleotides is not an easy problem to solve. One possible way around this problem would be to guess the structure of the hidden nucleotides and put the result in some kind of "scratchpad" that could be edited by the user and then copied into the existing secondary structure.

There are a large number of analysis functions that could be added to gRNAid to make it more powerful. For example, a function to search for a particular string of nucleotides in the structure would be nice. The ability to compare two structures and highlight conserved regions would also be useful. Also, color could be used to emphasize a particular feature, such as making all base-pairs blue or making Watson-Crick pairs green and noncanonical pairs red. Such coloring functionality would allow the user of gRNAid to be able to find patterns or even find errors in their secondary structures.

Finally, additional editing operations would be very useful to add to gRNAid. The user needs to be able to add, delete, or change a nucleotide. Here is why. What if the user was 75% complete with arranging the secondary structure when they noticed that they had an error in the alignment file? At this point, it would not be a good idea to edit the alignment file and recreate the structure, because 75% of their work would be lost. This is precisely the reason why there is functionality available to save as a PICT - so the user can take their work to a drawing program without losing all of their work.

Final Analysis

All in all, gRNAid seems to work better than expected. Early in the game, it was unsure whether the mapping algorithm would be effective in secondary structure creation. However, once gRNAid was up and running with a few bugs were ironed out, gRNAid began to work well as a secondary structure creation tool. As is always the case, unexpected problems, such as the insertions within insertions, crop up throughout the development process. Despite these problems, gRNAid can still be a tool that is useful within the molecular evolutionist's lab.

Appendix A Tutorial

The goal of this chapter is to familiarize you with the gRNAid application by walking you through the creation of a secondary structure. In this tutorial, the secondary structure of *Anacystis nidulans* will be generated from the *E. Coli* template file. You will edit the secondary structure by selecting and dragging nucleotides on the screen. You will learn how to map hidden nucleotide insertions to screen positions. Finally, you will examine other useful gRNAid features that are helpful in secondary structure creation. This tutorial assumes that you already understand the basic concepts behind gRNAid as explained in Chapter 1 and Chapter of this paper. Also, the tutorial assumes that you have experience using Macintosh computers. For example, it is assumed that you know what it means to scroll a window. Finally, it assumes that you have the gRNAid folder installed on a hard disk or on a floppy that is not locked.

Secondary Structure Generation

The gRNAid application and some example data files are provided on the gRNAid disk. The contents of the gRNAid folder should look something like that shown in Figure A-1. Do not worry if your version does not look exactly like the figure. All you need to run this tutorial is the gRNAid application and the Tutorial folder.

Double click on the gRNAid icon. The gRNAid main menu will appear, as shown in Figure A-2.

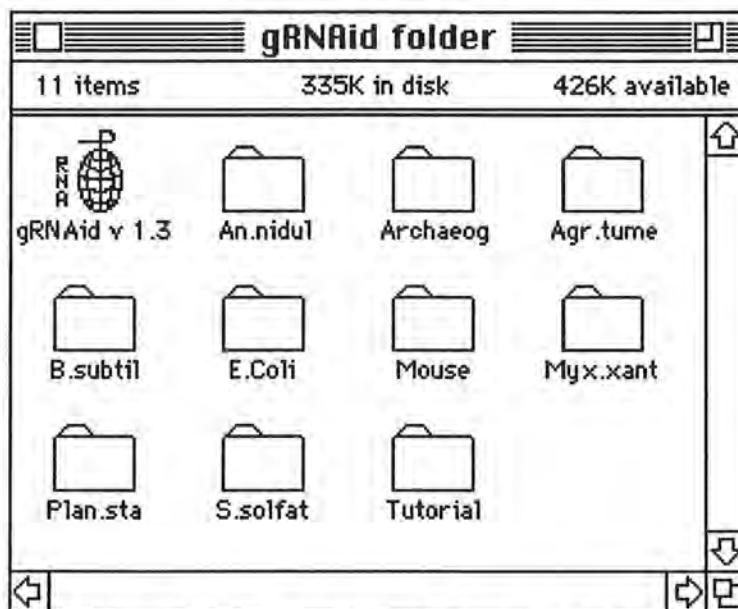


Figure A-1 Contents of the gRNAid folder

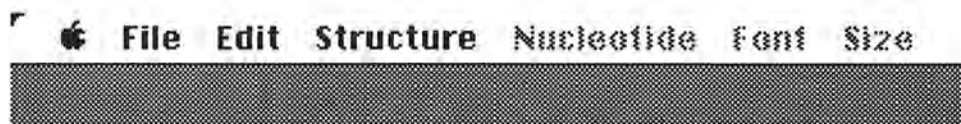


Figure A-2 The gRNAid main menu bar

From the **Structure** menu, select the **Generate...** menu item or use the **⌘ G** keyboard equivalent (Figure A-3).



Figure A-3 Select **Generate** from the **Structure** menu

A file selection dialog will appear prompting you for an alignment file. Descend into the Tutorial Folder and then into the *An.nidul* folder and open the Alignment file as shown in Figure A-4. The gRNAid program normally takes about 15-45 seconds to read an alignment file with approximately 1500 nucleotides, depending on the machine you are using.

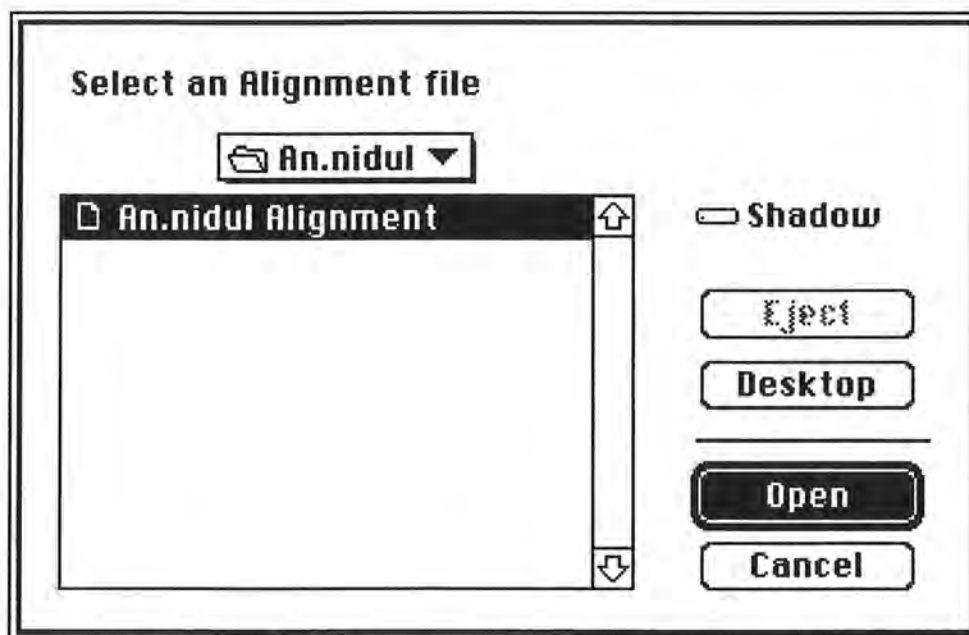


Figure A-4 Open the *An.nidul* Alignment File

Next you will be queried for the template file. Move back up to the gRNAid folder by pressing the mouse arrow on the popup menu and selecting the Tutorial folder, as shown in Figure A-5. (Note: you can also ascend to the next highest level by clicking the mouse button on the hard drive icon that appears above the **Eject** button.)

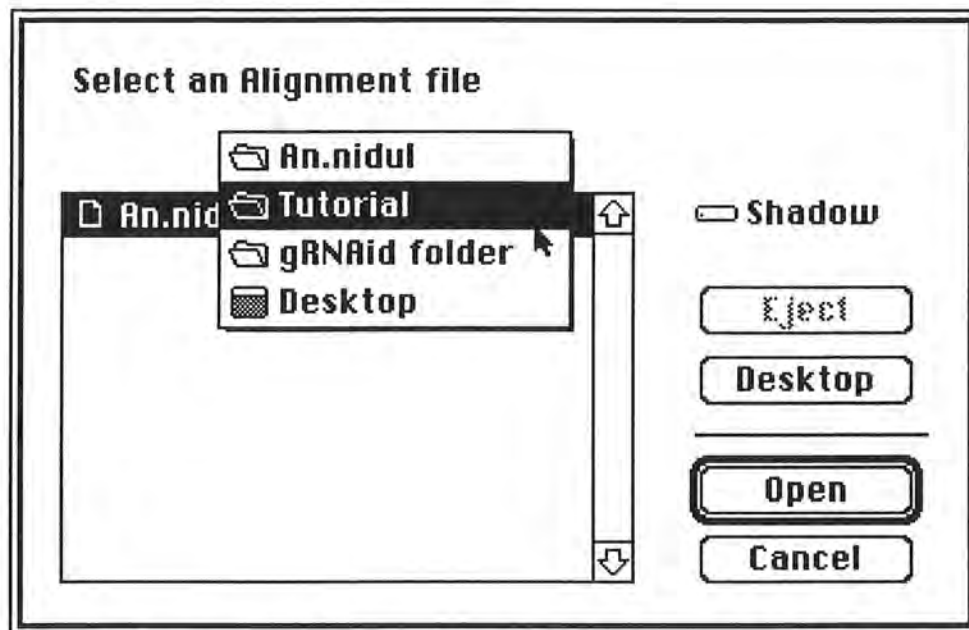


Figure A-5 How to ascend to the *Tutorial* folder

Descend into the *E.Coli* menu and open the *E.Coli* Template file, as shown in Figure A-6. Be sure to select the *E.Coli* Template and not the *E.Coli* Alignment. The gRNAid program will take about 30-60 seconds to read in the template file data and create the secondary structure, depending on the machine you are using.

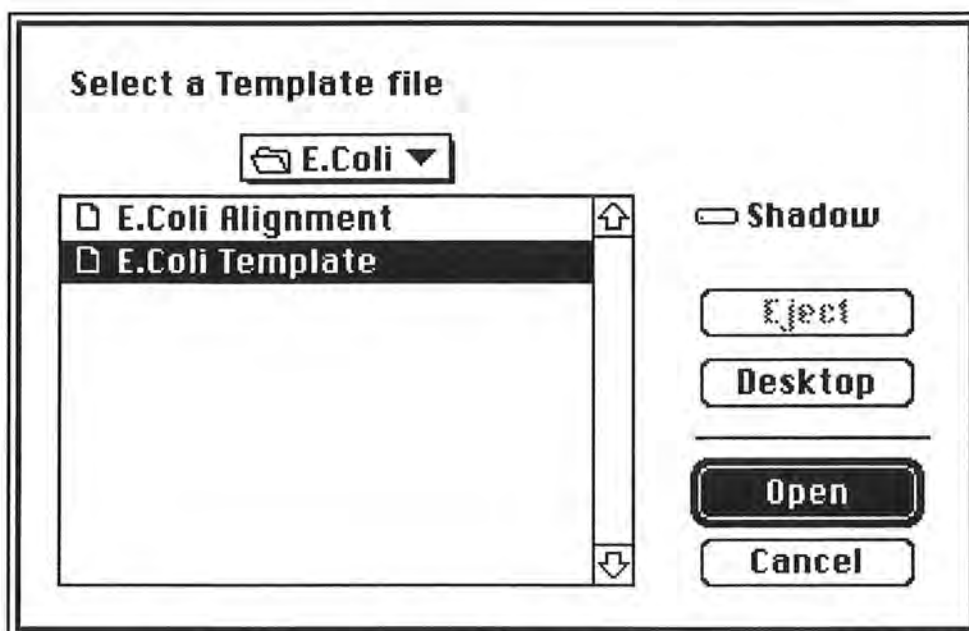


Figure A-6 Select the *E.Coli* Template file

The secondary structure of *Anacystis nidulans* is displayed in a scrollable window. A portion of this secondary structure is shown in Figure A-7. Note that this is not a complete secondary structure. Bold nucleotides represent areas where gRNAid was unable to map nucleotides from the *Anacystis nidulans* sequence onto screen positions.

Scroll the window until you find the section of the structure represented in Figure A-7. See Figure A-8 for a roadmap of where you need to go.

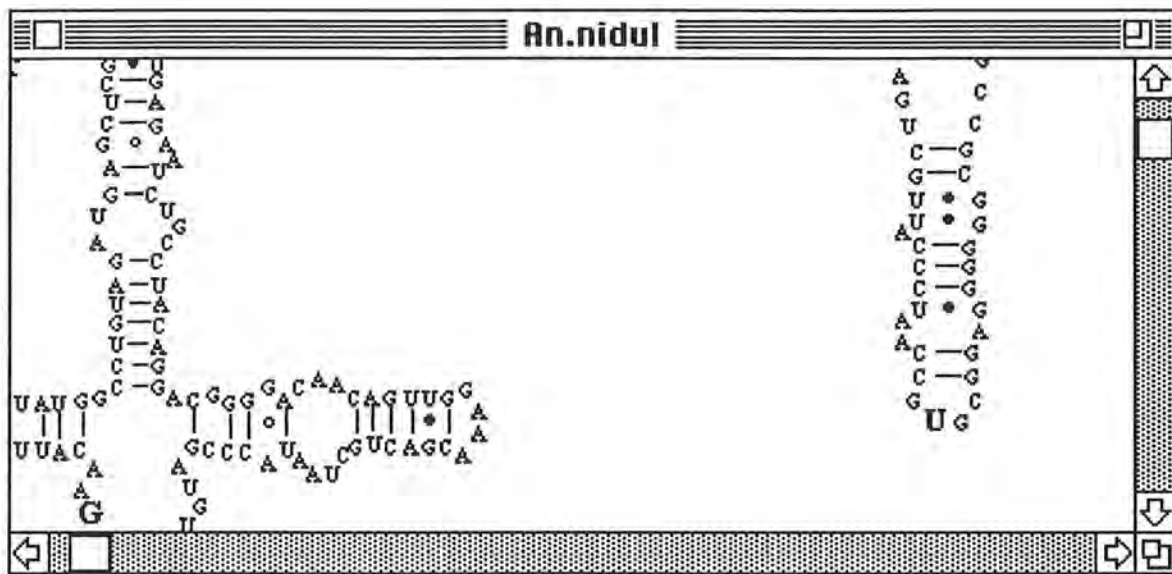


Figure A-7 A portion of the *Anacystis nidulans* secondary structure

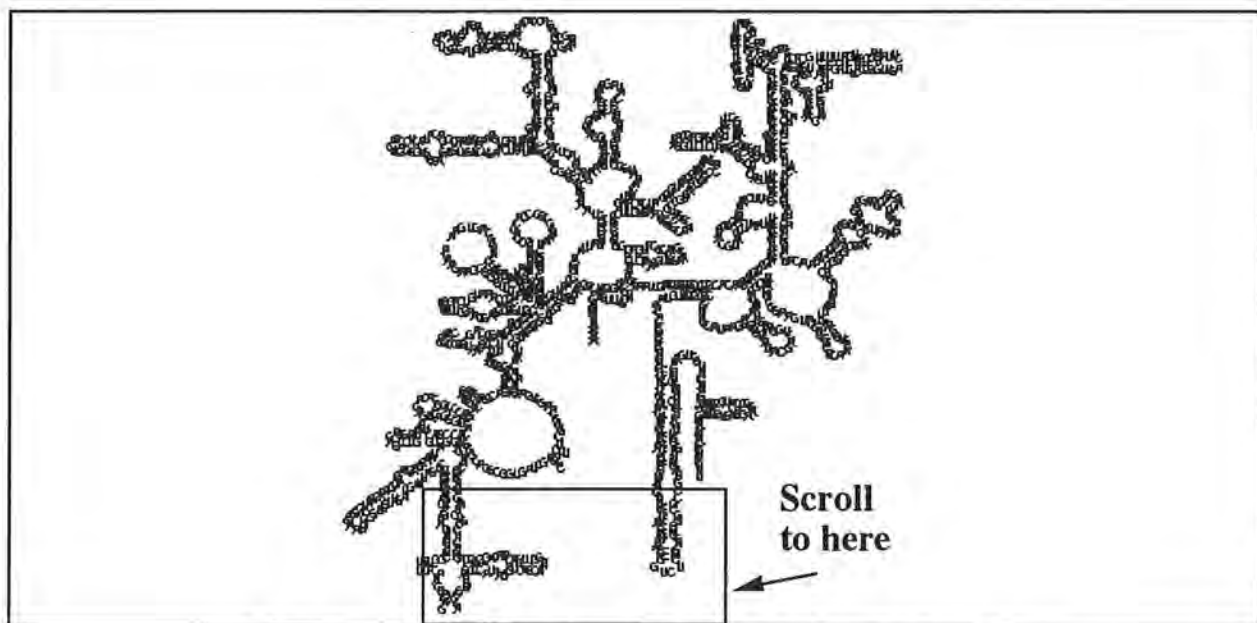


Figure A-8 A roadmap showing where to scroll

Now save the secondary structure (template) just created (as we will edit the file in sections that follow). From the **File** menu, select the **Save** menu item. Because the template has never been saved before, a dialog appears querying you to name the new file. Make sure that the current folder is the *An.nidul* folder (which is in the Tutorial folder) and save the file as *An.nidul* Template, as shown in Figure A-9.

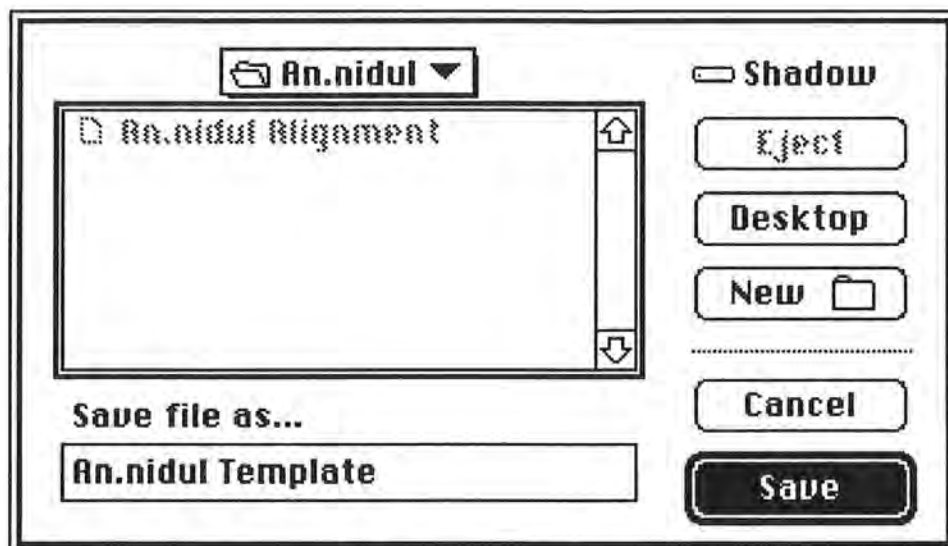


Figure A-9 Save the template as *An.nidul* Template

Selecting and Dragging

In the following section, feel free to practice selection and dragging skills on the *Anacystis nidulans* structure you just created. Do not worry about making a mess. The *An.nidul* Template file you just saved will be used in subsequent sections of this chapter.

Nucleotides can be selected in several different ways. Bonds cannot be selected at all. To select a single nucleotide, click the mouse on it. Figure A-10 shows what the window would look like when a bold nucleotide is selected. Notice that the selected nucleotide is inverted. Also notice that new data appears at the top of the window when the bold nucleotide was selected. This is referred to as the 'hidden nucleotide bar'. It shows the nucleotides that have not been mapped to a screen position. In this example two nucleotides (UC) have yet to be mapped to screen positions. The process of associating screen positions with these 'invisible' nucleotides, referred to as mapping, will be discussed later in this tutorial.

To deselect a nucleotide, click anywhere within the window. If you click on another nucleotide, it will be selected and any other selected nucleotide will become deselected. If you click on an empty portion of the window, all selected nucleotides will deselect.

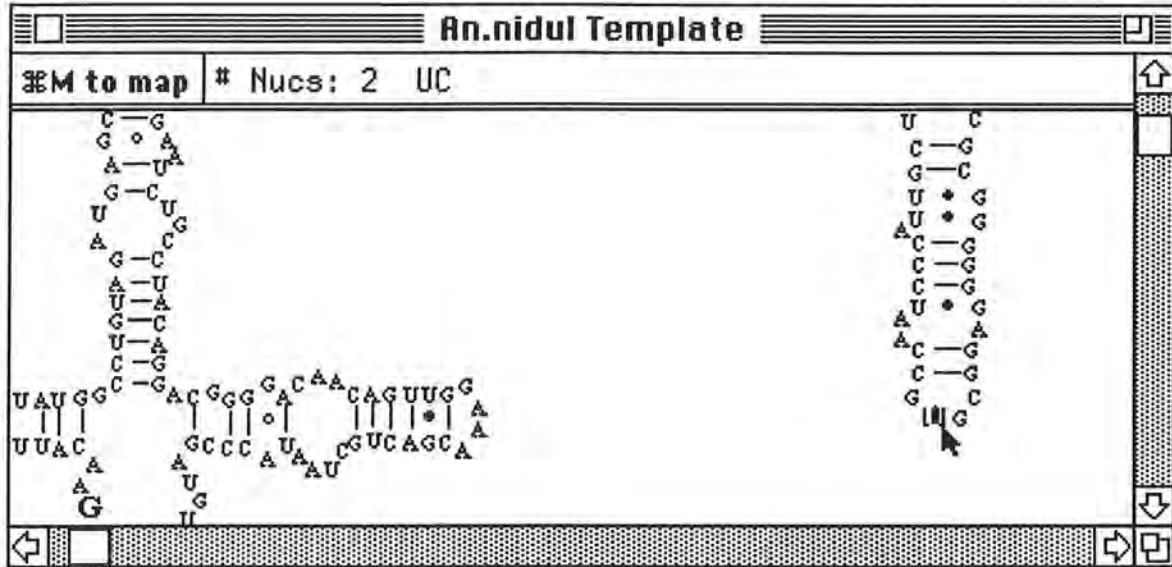


Figure A-10 Select the bold U nucleotide

A group selection can be performed by keeping the mouse button depressed and dragging, as shown in Figure A-11. While dragging, a selection rectangle will appear; once you let up on the mouse button, anything within this rectangle will be selected. This is an easy method of selecting a group of contiguous nucleotides. Figure A-12 shows the results of a group selection. Notice that the hidden nucleotide bar did not appear in the window even though a bold nucleotide is in the group of selected nucleotides. The rule is that the hidden nucleotide bar only appears when a single, bold nucleotide has been selected. Again, to deselect all selected nucleotides, click the mouse on an empty portion of the window.

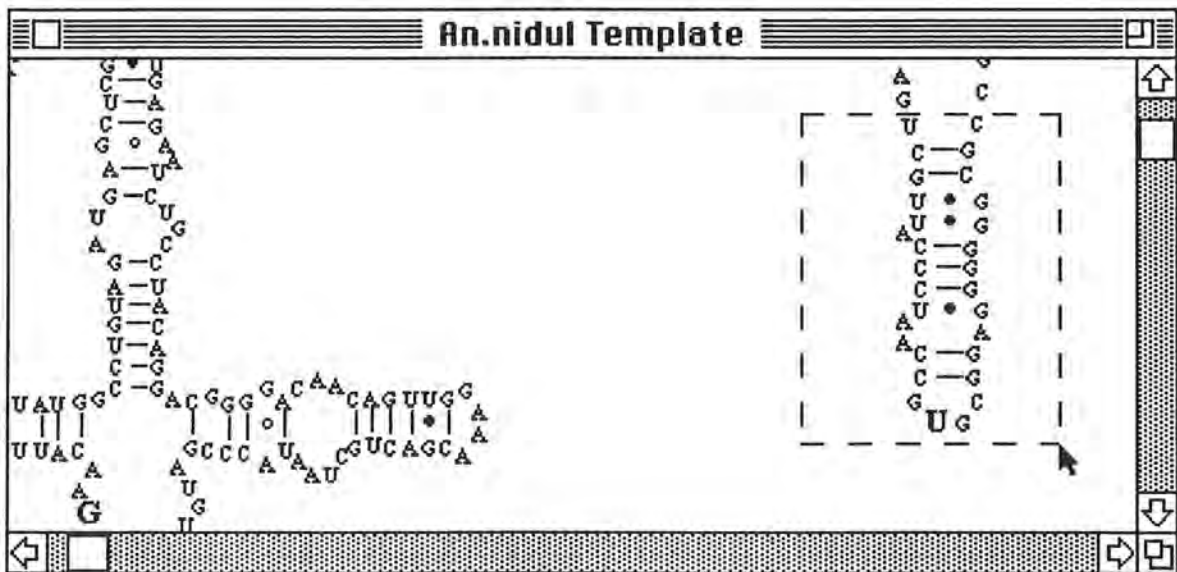


Figure A-11 Selection of a group of nucleotides

The last selection mechanism allows you to select those pesky nucleotides that don't easily fit into the selection rectangle. For example, assume that you have selected nucleotides as shown in Figure A-11 and that you want to select the G and C as shown by number 1 at the top of Figure A-12. To select the G nucleotide without deselecting everything else, keep the **shift** key pressed and click on the G. Do the same thing to select the C.

Assume that U and G as shown by number 2 in Figure A-12 are to be deselected. Again, press the **shift** key and click the U. Keep the **shift** key down, and click the G. The U and G have become deselected without altering the selections of any other of the nucleotides in the group. The **shift-click** mechanism acts as a toggle. If there is a **shift-click** on a selected nucleotide, then it is toggled to its deselected state. Conversely, a **shift-click** on a deselected nucleotide selects it. This mechanism allows you to add and remove nucleotides from the selection area without affecting those nucleotides already selected.

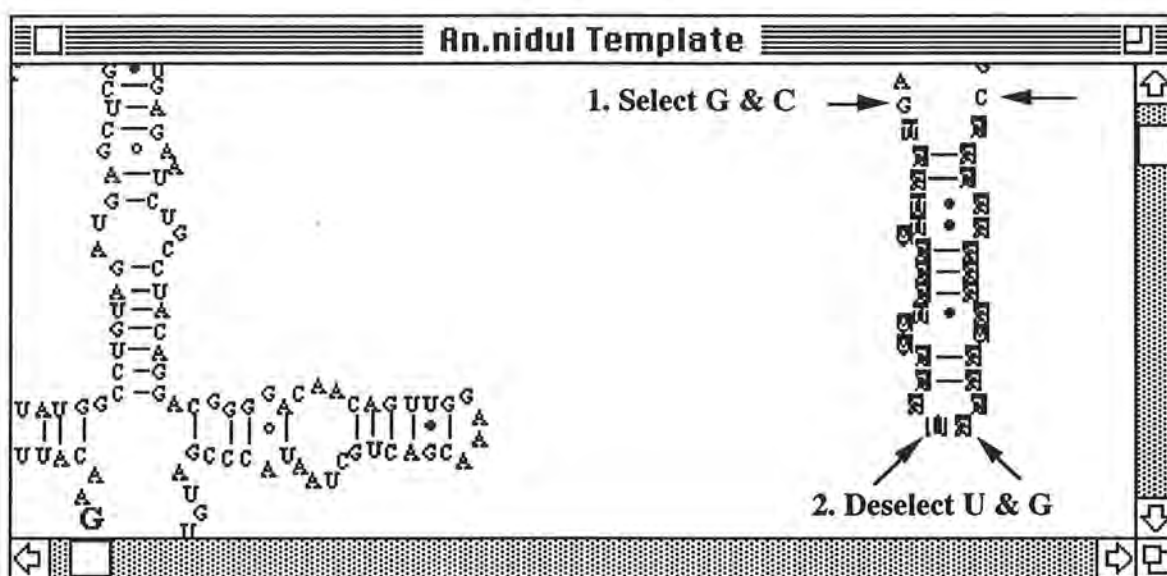


Figure A-12 Examples of selection and deselection

To drag the selected group of nucleotides, click the mouse on any of the selected nucleotides and keep the mouse button pressed down while dragging. An outline of the drag region appears during the drag to aid you in the placement of the dragged nucleotides, as demonstrated in Figure A-13. When the drag is complete, all nucleotides within the selection list are offset to their new position. Note that the bonds move along with their respective nucleotides. Thus, if a Watson-Crick C-G pair is selected and dragged, the bond goes with the nucleotides to their new position.

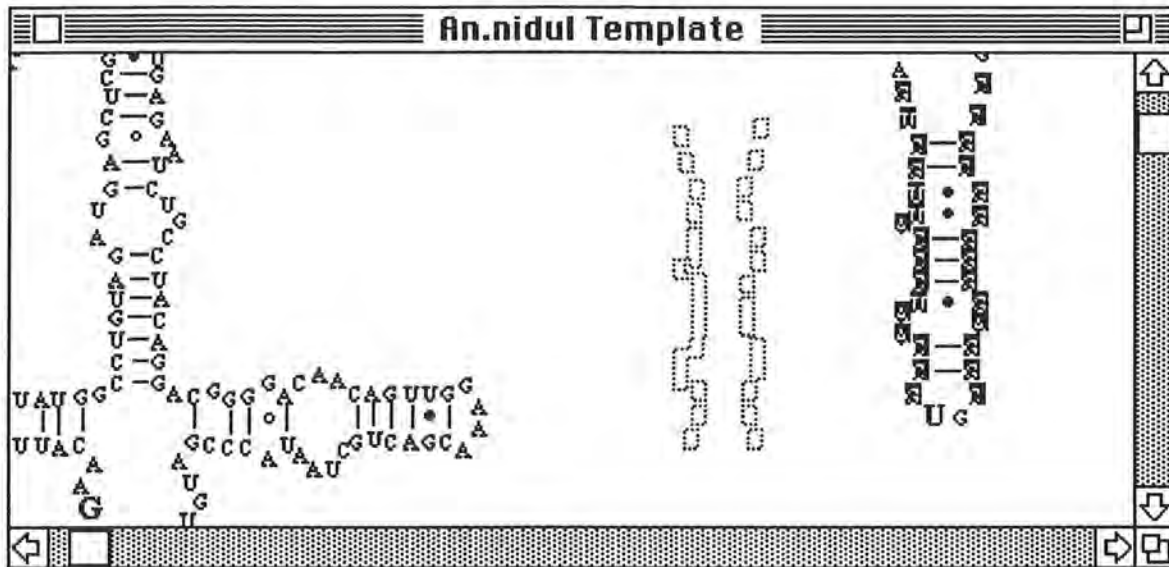


Figure A-13 Group dragging

Mapping Nucleotides

The goal of this section is to demonstrate how to map nucleotides that have not been associated with screen positions. See the User's Manual for more detailed information on nucleotide mapping.

Before getting started, revert to the saved version of the template file. If the window from the last session is still up, close by by selecting the **Close** item from the **File** menu. If you are asked if you want to save the file, click on the **No** button. Now read in the saved file: select **Open** from the **File** menu, and in the *An.nidul* folder, open the Tutorial Template file. Scroll to the same area that you were at in the last section. See Figure A-8 for the roadmap.

Select the bold U nucleotide as shown in Figure A-15. The hidden nucleotide bar appears, and indicates that there is an insertion of 2 nucleotides, U and C, on the 3' side of the bold U. It is important to always place the inserted nucleotide on the 3' side of the bold nucleotide, as shown in Figure A-14. Thus, after the two hidden nucleotides (UC) are inserted, the primary structure of the loop will be GUUCGC, where the underlined portion of the sequence segment is the part that appears in the hidden nucleotide bar.

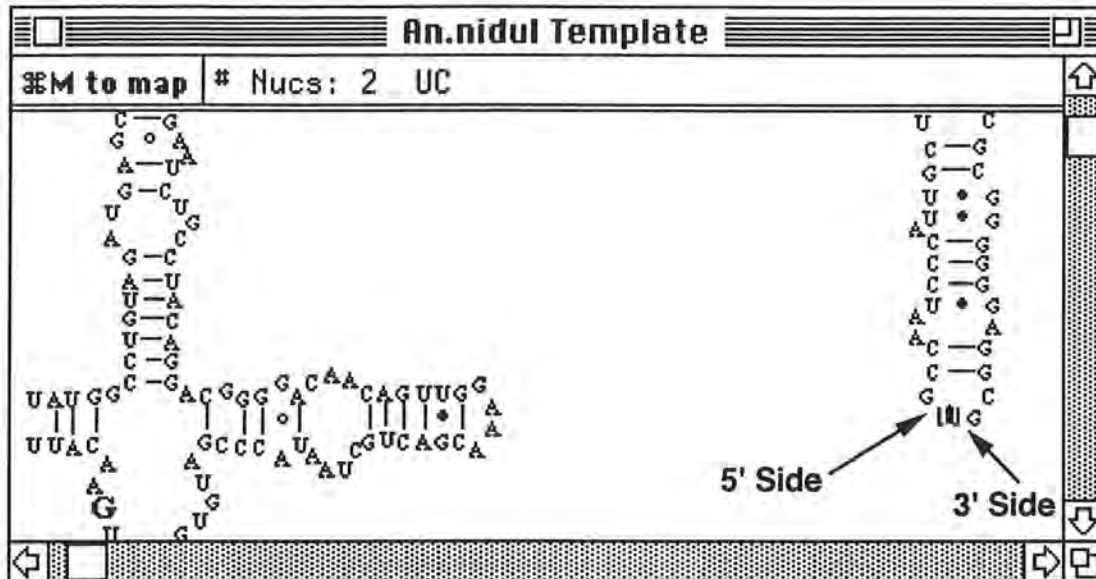


Figure A-14 Map hidden nucleotides on the 5' side of the bold nucleotide

Before the two new nucleotides are mapped to screen positions, make room for them by arranging the hairpin loop as shown in Figure A-15.

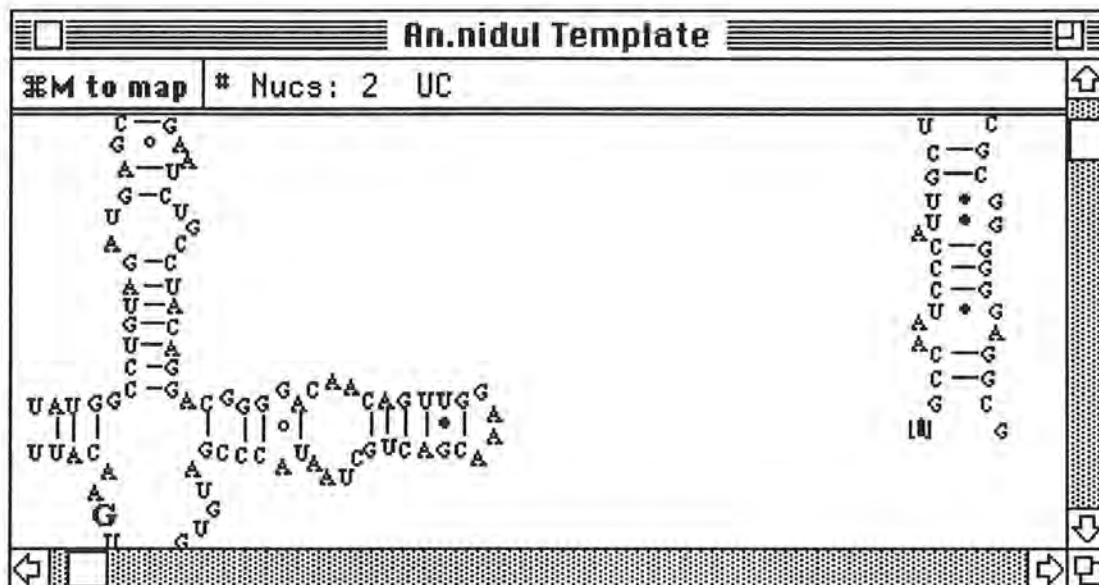


Figure A-15 Make room for the hidden nucleotides

Reselect the bold U nucleotide if it is not already selected. Now enter nucleotide map mode by selecting the **Map Hidden** item from the **Nucleotide** menu or typing **⌘M**, the keyboard equivalent (Figure A-16). Look at the **Nucleotide** menu again. Notice that the text of the last menu item has changed to **Exit Map** (Figure A-17), but that it still has the **⌘M** keyboard equivalent associated with it. You can toggle back and forth between map mode and normal mode by using **⌘M** or the **Map** menu item in the **Nucleotide** menu. If you haven't already done so, exit map mode by typing **⌘M** or selecting the **Exit Map** item from the **Nucleotide** menu. It is

convenient to toggle between modes, when, for example, there are 100 hidden nucleotides. You can enter map mode, map 10 nucleotides, and then exit map mode so that you can arrange the 10 nucleotides into their appropriate positions.

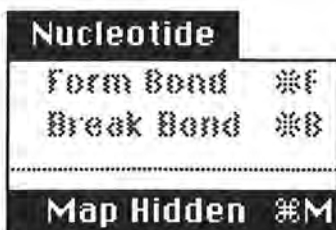


Figure A-16 Nucleotide menu with Map Hidden option



Figure A-17 Nucleotide menu with Exit Map option

Re-enter map mode by typing **⌘M** or selecting Map Hidden from the **Nucleotide** menu. Once you enter map mode, the cursor changes to a cross-hair (+). Note that the cursor changes back to an arrow when the mouse is moved outside of the work area of the window. When the cursor is moved back into the window, it changes back to the cross-hair. You can now click the cross-hair at the position in the window where the first hidden nucleotide (**U**) is to be placed. Figure A-18 shows the general position where you should stamp the first hidden nucleotide.

After the first nucleotide has been stamped, notice that the hidden nucleotide bar has changed to reflect the fact that only one nucleotide remains to be mapped. Also, the nucleotide that was just mapped to a screen position is now bold and the previously bold nucleotide is now normal sized.

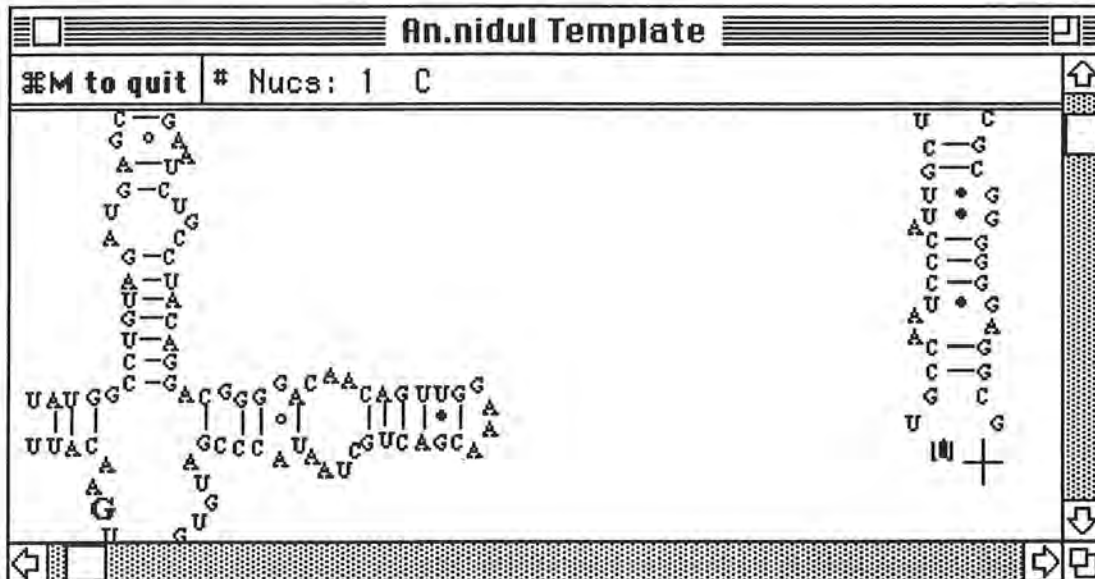


Figure A-18 Map the first hidden nucleotide to a screen position

Map the final nucleotide (C) to its screen position. Notice that the **hidden nucleotide bar** disappears after you map the last nucleotide. Thus, once you are finished mapping a given sequence of invisible nucleotides, you are automatically transferred out of map mode. Figure A-19 shows the final result after mapping the 2 hidden nucleotides to screen positions.

Forming and Breaking Bonds

Bonds can only be manipulated at well-defined times. For example, if you look at the **Nucleotide** menu as shown in Figure A-17, you will notice that the bonding menu items are completely disabled. To form a bond, you must select two nucleotides that are bondable. For example, if you select a nucleotide A and select another nucleotide C, you cannot form a bond. The only legal bonds that can be formed in gRNAid are Watson-Crick (A-U, G-C), and noncanonical (A-G, G-U) bonds. Select the G and C bonds as shown in Figure A-19 by either using the group drag or shift-click selection mechanism. Now select the **Form Bond** item from the **Nucleotide** menu. A Watson-Crick bond is formed.

[Note: At times when one tries to form a new horizontal/vertical Watson-Crick bond, the bond is not drawn on the screen. Use the **Redraw** menu option to correctly draw the bond.]

To break a bond, you must select two nucleotides that are currently bonded together. Select any two nucleotides that are bonded together. Break the bond by selecting the **Break Bond** item from the **Nucleotide** menu. Don't forget to bond it back together again before moving onto the next section.

[Note: At times when one tries to break a horizontal/vertical Watson-Crick bond, the bond is not removed from the screen. Again, use the **Redraw** menu option.]

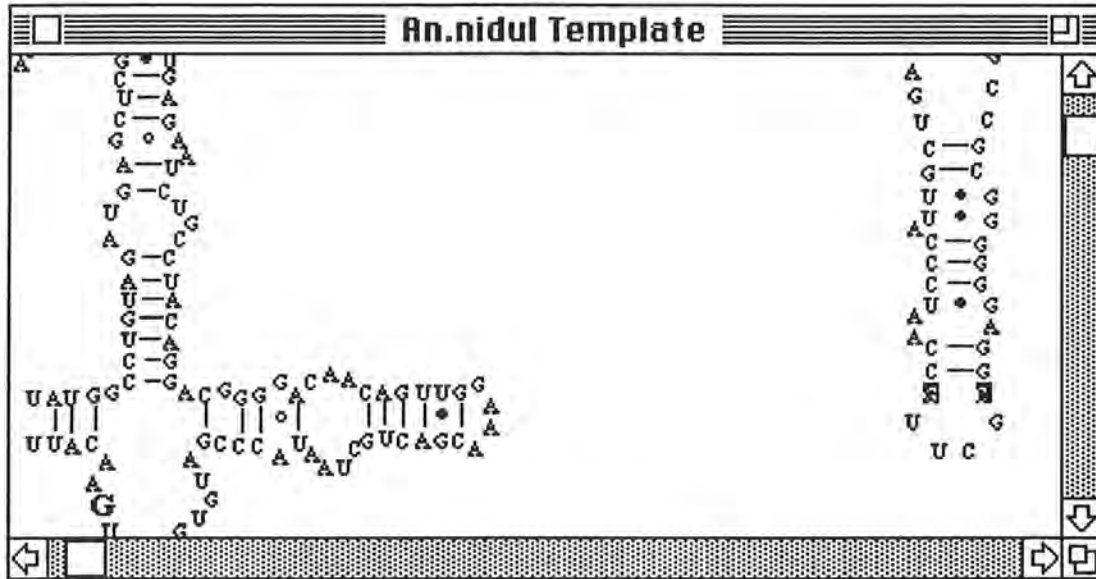


Figure A-19 Select the G and C to form a bond

Potpourri

The rest of this tutorial examines other handy features of gRNAid. First let's examine the **Show Backbone** feature. For the sake of this exercise, swap the U and G nucleotides as shown in Figure A-20.

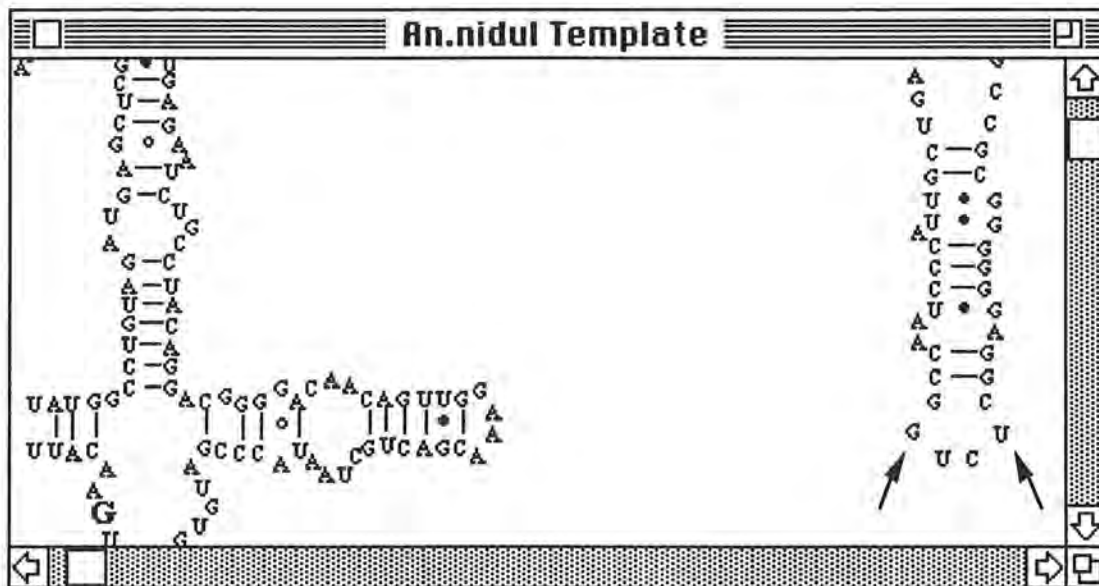


Figure A-20 The G and U have been swapped (see Figure A-19)

Select the **Show Backbone** item from the **Structure** menu. Notice that each nucleotide is now connected to its neighbors by a line. This dot-to-dot illustration is a powerful way of looking at the primary structure in a graphical way. Areas of chaos in the window denote areas where the

apparent primary structure is incorrect, as shown in Figure A-21. Turn the backbone off by selecting the **Show Backbone** menu item, which acts as a toggle. Swap the G and U back to their former screen positions.

[Note: The screen is not updated properly when **Show Backbone** is on and nucleotides are dragged around. To get an updated backbone, use the **Redraw** menu item from the **Structure** menu.]

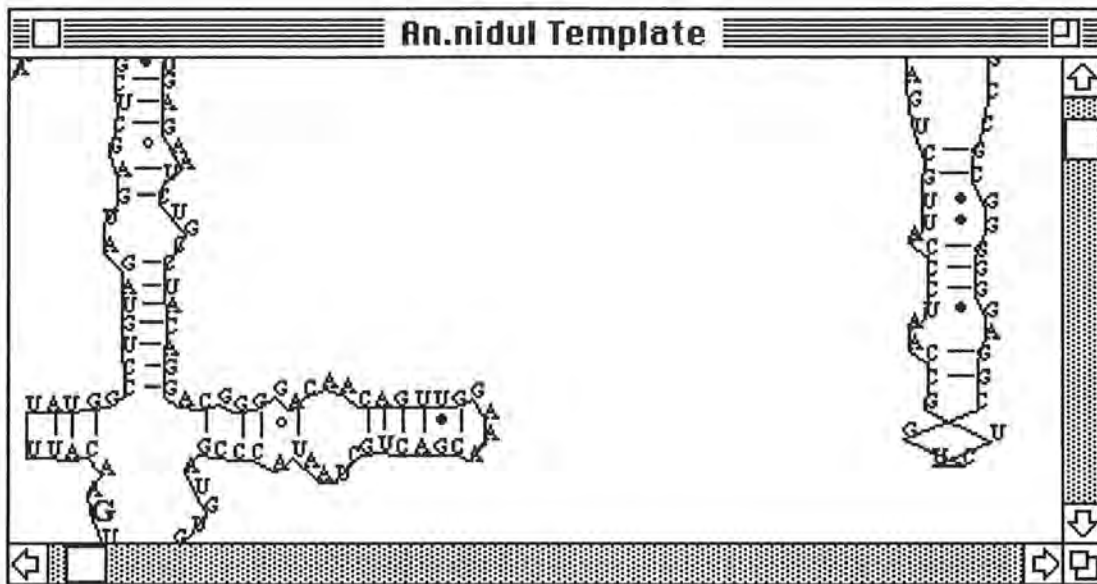


Figure A-21 Show backbone

Now change the font to **Helvetica** and the font size to **12 pt** as demonstrated in Figure A-22. Be patient; when font or font size is changed, gRNAid must recalculate bonding screen position information for each bond in the secondary structure.

Font	Size
ArchiMedium	6 pt
Athens	7 pt
Boston	8 pt
Cairo	✓ 9 pt
Chicago	10 pt
ChicagoSymbols	11 pt
Courier	12 pt
Geneva	14 pt
Helvetica	18 pt
London	24 pt
Los Angeles	
Mobile	
Monaco	
New York	
Palatino	
San Francisco	
Symbol	
✓ Times	
Venice	

Figure A-22 Changing the font and font size

Next, save the Tutorial Template as a PICT file so other, more powerful drawing programs (such as MacDraw®) can be used to edit the secondary structure. From the **File** menu, select the **Save As** hierarchical menu and then select the **PICT...** menu option as shown in Figure A-23. Save the PICT file in the An.nidul folder. Once the file is saved, any drawing program that reads PICT files can read in this newly created PICT file. To determine if a particular drawing program can read PICT files, refer to its documentation.



Figure A-23

To print the secondary structure, use the **Print...** menu options under the **File** menu. The gRNAid program only prints one page. If the secondary structure is larger than one physical page, gRNAid will shrink the secondary structure to fit the page before printing.

You have now completed the **gRNAid** tutorial and are considered dangerous. For general gRNAid information, refer to the gRNAid User's Manual in Chapter 4.

Bibliography

- [JAE1 1989] Jaeger, J.A., Turner, H., and Zuker, M., 1989. Improved Predictions of Secondary Structures for RNA. *Proceedings of the National Academy of Science. USA, BIOCHEMISTRY*, 86:7706-7710.
- [JAE2 1989] Jaeger, J.A., Turner, H., and Zuker, M., 1989. Predicting Optimal and Suboptimal Secondary Structure for RNA. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*. Edited by Doolittle, R.F. *Methods in Enzymology*, 183:281-306.
- [GILB 1990] Gilbert, D.G., 1990. LoopViewer, a Macintosh program for visualizing RNA secondary structure. Published electronically on the Internet, available via anonymous ftp to iubio.bio.indiana.edu.
- [GOUY 1987] Gouy, Manolo, 1987. Secondary Structure Prediction of RNA. *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*. Edited by Bishop, M.J., and Rawlings, C.J., IRL Press, Oxford. pp259-284.
- [GUTE 1985] Gutell, Robin R., Weiser Bryn, Woese, Carl R., and Noller, Harry F., 1985. Comparative Anatomy of 16-S-like Ribosomal RNA. *Progress in Nucleic Acid Research and Molecular Biology*, 32:155-216.
- [LE 1989] Le, Shu-Yun, Nussinov, Ruth, and Maizel, Jacob V., 1989. Tree Graphs of RNA Secondary Structures and Their Comparisons. *Computers and Biomedical Research*, 22:461-473.
- [OLSE 1986] Olsen, Gary J., Lane, David J., Giovannoni, Stephen J., and Pace, Norman R., 1986. *Microbiol Ecology and Evolution: A Ribosomal RNA Approach*. *Annual Review of Microbiology*, 40:337-365.
- [WEST 1990] Weston, Dan, 1990. *Elements of C++ Macintosh Programming*. Addison-Wesley Publishing Company, Inc.
- [ZUK1 1989] Zuker, M., 1989. On Finding All Suboptimal Foldings of an RNA Molecule. *Science*, 244:48-52.
- [ZUK2 1989] Zuker, M., 1989. The Use of Dynamic Programming Algorithms in RNA Secondary Structure Prediction. *Mathematical Methods for DNA Sequences*, Edited by Waterman, M.S., CRC Press, Inc., pp 159-184.