SolPicker: Designing a Software Suite to Select Oligonucleotide Probes for Next-Generation Sequencing

1. Purpose of Research:

SolPicker is computer software that was developed for scientists looking to create oligonucleotide probes for microarray capture prior to next-generation DNA sequencing. In recent years, genomics have made exciting and extensive advances because of breakthroughs in next-generation sequencing technology. Sequencing machines work by reading a segment of DNA and determining what the nucleotide sequence is. Since the Human Genome Project, the costs of sequencing have gone down while the capacity of sequencers has greatly increased. Representing great advances from shotgun Sanger sequencing, some of the new methods includes microelectrophoretic methods, sequencing by hybridization, real-time observation, and cyclic-array sequencing (Shendure and Ji 2008). However, the problem with next-generation sequencing is the overwhelming volume of data that is produced (Rice et al 2000). Even with the decreasing costs of sequencing, it would be very cost inefficient to use sequencing to sequence entire genomes when often biologists only want to study specific genes. One solution to this problem is DNA capture using custom microarrays (Turner et al 2009).

Often, when looking to study the correlation between a genomic segment (called the query or target sequence) with a certain phenotype (disease, physical trait or defect, etc.), researchers will investigate DNA from an organism demonstrating the phenotype in question using a DNA microarray. Microarrays consist of a slide or "chip" that has a series of oligonucleotide probes, with specific DNA sequences that are designed to be complementary to the query sequence. When a sample of DNA is washed over the array of oligonucleotides, the

DNA on the probes will bind (hybridize) to any DNA fragments that contain complementary sequence The researcher can then wash away any unbound DNA and separately elute the bound (captured) DNA. In this way, the scientist can use microarray capture to detect the presence of a certain DNA sequences within a sample from an organism (Hodges et. al 2009). Figure 1 gives a visual representation of the process.

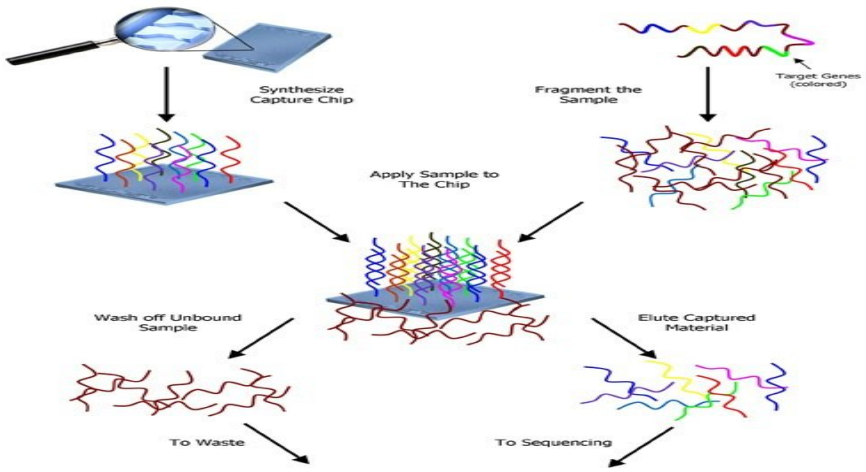

Figure 1-The process of micro-array capture (Lipshutz, Morris, Chee, Hubbell, Koza, Sha, Shen, Yan, and Fodor 1995)

Once the target sequence has been hybridized, the researcher may choose to use DNA sequencing machines to determine the exact sequence of the captured segments, which are usually pieces of DNA hundreds or thousands of bases long that contain sequence complementary to the probes which are usually only 20-60 bases long. The probes will also hybridize target segments that are not exactly identical but very similar to the probe sequence (Hodges et. al 2007).

The goal of the SolPicker software is to create a computationally efficient, dynamic, adaptable, user-friendly program that will allow scientists to use smaller probes by selecting appropriate smaller DNA subsequences (called oligos) from a specified gene or genomic segment. The program will allow users to input the segment of the genome that they want to

study (including the chromosome number, and start and end base), and the length of the oligos that are to be created; the program will then retrieve the appropriate DNA sequence from the internet, create all possible subsequences of the specified length, use a variety of filters to determine which oligos would serve as effective probes, and produce four output files following the standard industry format. A corollary of this research was to develop the software using a robust reusable framework incorporating some of the advances in software development practices as this would encourage collaboration and aid further research and development in the area of extraction tools

2. Rationale for Research:

Given the time, resource and spatial (spatial as the space on the microarray is limited) constraints that researchers operate under, it is not reasonable to create oligonucleotide probes that include an entire gene. SolPicker will allow scientists to create smaller, more cost effective probes that still allow them to capture the gene in question. SolPicker greatly eases the burden on scientists by retrieving, organizing, processing, and outputting the tremendous volume of sequence data so that the scientist can easily determine his or her experimental details. Many of the necessary computations and algorithms would be inefficient, if not impossible, to be performed manually. Although there are similar programs, these programs do not put the same emphasis on producing unique oligos as SolPicker. The importance of picking unique oligos, which is crucial to a scientist, will be described later. The existing programs also have not been designed to be as user-friendly and scalable as SolPicker.

3. Prior Work and Contributions of Others on this Project:

Many other institutions, including the one for whom I helped to write this program, have worked on similar projects, either collaboratively with or separately from my own institution. Other institutions and bioinformaticians, such as my own mentor, have created similar software in the past that may be written in different languages and/or may use different algorithms to select oligos. Most of these programs are home grown and developed to meet a specific goal or researchers need and developed with limited capabilities. Often they are considered proprietary and not shared with others. The goal of SolPicker, is to improve the functionality considerably and enhance and the capability of the software so that it can be easily used by the scientist or researcher in a wide variety of settings and be continually improved through the use of a robust development framework.

This project is a subset of a larger effort involving others. This research report details parts of the software that I was instrumental in designing and developing.

4. Methodology:

The previous programs that we are aware of were written in scripting languages. It was decided to write our new application in an object oriented programming language in order to make it scalable, easy to use, and able to provide features that were new and innovative. We began by actually planning the system architecture before coding and using some of the best practices in software development as opposed to writing "seat of the pants code" that characterizes many programs in this field. Given below are the design decisions that we made in order to efficiently create a robust program:

I. **Java:** For SolPicker, the programming language of choice was Java. Java was an ideal choice for this project for several reasons: Java is robust in that it meticulously

checks for errors before compiling, which is crucial when dealing with such large amounts of data; Java is platform independent and can run the same on any computer, which is extremely useful when distributing the program to users with different operating systems; Java is multithreaded and can perform many tasks simultaneously, a feature that helps when dealing with large data sets; Java is preferred as it automatically handles tasks like garbage collection and memory allocation; finally, and most importantly, Java is object oriented which makes it easier to modify and adapt code, a feature that is crucial when working on such a large, ever changing project with multiple programmers.

II. **Eclipse IDE:** Eclipse is an open source software that provides an Integrated Development Environment (IDE) in which the programmer can write programs. While Eclipse makes writing code easier by relieving the programmer of the responsibility of manually keeping track of all files and projects, it also has many built in, features that were extremely useful in this context:

a. **Documentation:** Most existing software does not come with instructions or documentation and hence is difficult to adapt. We were able to use the documentation feature in Eclipse to easily create Javadocs, which are HTML pages that display information about a class, organizing information about the different methods, fields, and

Figure 2 - Javadoc HTML Page

constructors. A subsidiary objective of this project is contribute to a next-generation sequencing software framework which other developers can integrate with or enhance, and having Javadocs helps achieve this goal. Figure 2 shows an example of a Javadoc.

b. **Debugging view:** Eclipse allowed us to develop code faster, due its advanced debugging features. Debugging lets the programmer examine code line by line in order to easily spot errors. The objective was to develop robust reusable software and this feature was very useful. Figure 3 gives a visual representation of this view.

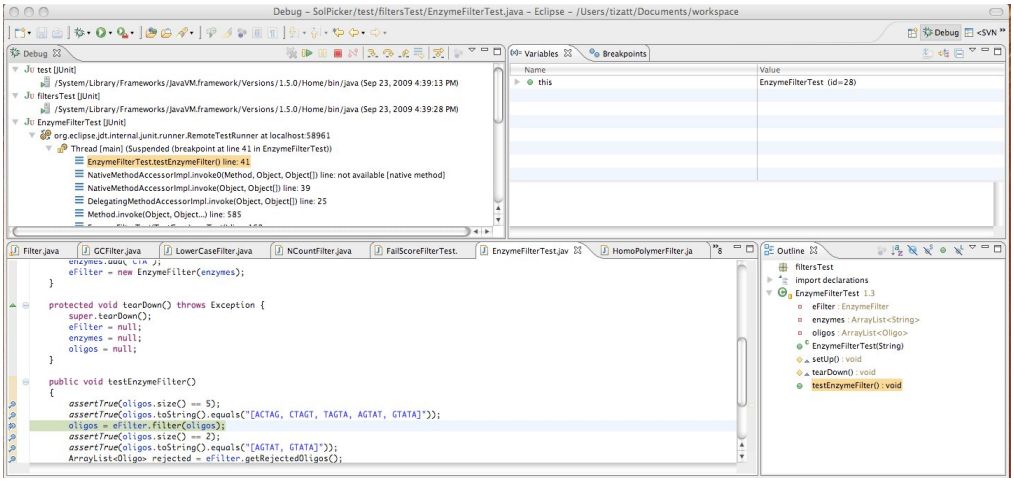

Figure 2-The Debugging Perspective in Eclipse

III. **Source Control systems:** Many of the previous programs were written by a single person. As the idea was encourage collaboration, we used Concurrent Versions System (CVS) as our source control system to allow easy resolution of coding differences and provide a repository for the team. This is another example of the use of the best practices in software development. The CVS Repository for SolPicker is displayed in Figure 4.

Figure 3-CVS Repository for SolPicker

IV. **JUnit Tests:** Again as a departure from previous coding practices, we used JUnit testing software to validate code often and early. This reduced the development time and helped us to deliver a higher quality product.

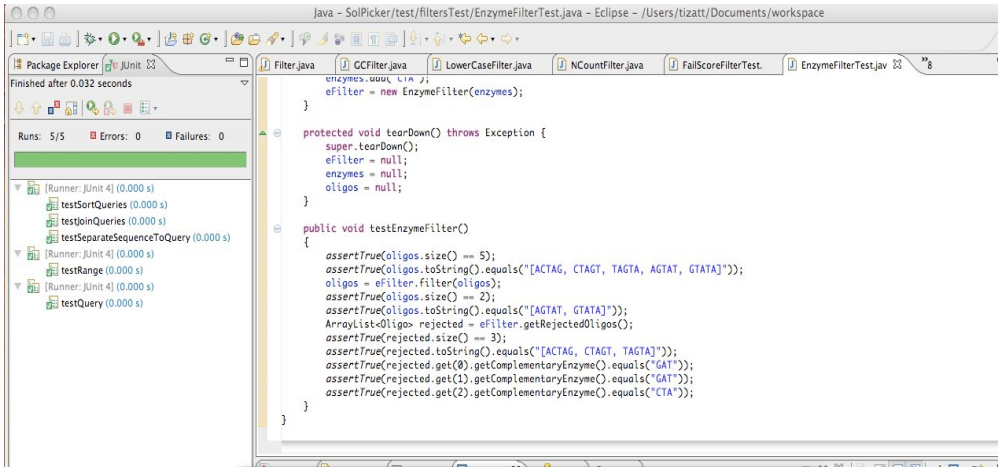

Figure 4-A Successful JUnit Test for the Enzyme Filter

V. **Inheritance:** A feature of object oriented programming, inheritance allows for easy code reuse and modification. SolPicker took advantage of the inheritance capabilities of Java in the creation of multiple filters and this again distinguishes this software from others as this gave enhanced flexibility to the end user.

VI. **GUI:** Java allows for the creation of a Graphical User Interface (GUI). SolPicker uses a GUI that prompts the user for input on all the query, configuration, and job information, processes the input, and carries out all the necessary procedures. The previous programs in this area had text file inputs as the only input source. The flexibility and the usability provided by the use of the GUI cannot be over emphasized for this was the first time that a program of this type provided the functionality we could. The SolPicker's GUI possesses extremely extensive error

checking capabilities; Information cannot be entered if it is not an appropriate input. For example, for a query start and stop position, the program checks to make sure that the user has entered a valid set of numbers as opposed to letters or words. If this is not the case, then the user is notified of the incorrect input, and the program does not proceed until the incorrect input is fixed. Similarly, for input fields that require words or letters, the GUI checks to make sure that the user has not entered numbers or invalid words or letters. It not only checks if the input is in the correct format but also that it makes logical sense in the context; as an illustration, when entering parameters for the G-C Percent filter, the user is not allowed to enter negative numbers or numbers above 100, and the GUI also generates an error message if the minimum percent is higher than the maximum percent. Due to flexibility of the code, we have the ability to add other checks and prompts as needed. The figure below shows the input screen and the multiple filter criteria that can be used to develop unique sequences.

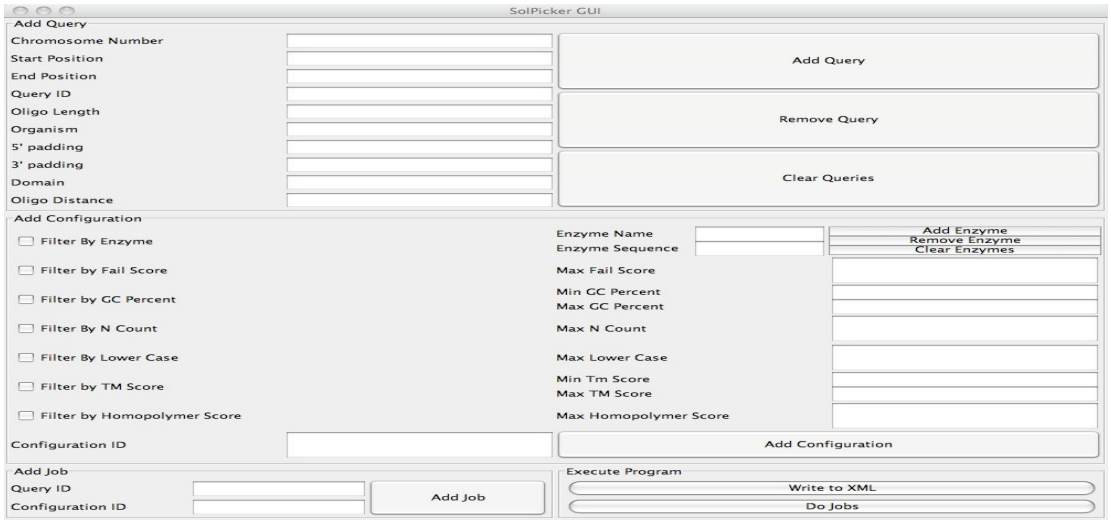

Figure 5-The SolPicker GUI

5. **Design and Results:** In SolPicker users can specify the genome location to be examined and allow the program to complete the rest. Input can be done through the Graphical User Interface (GUI). There are three parts to the input, the first two of which can be completed in any order. First the user must specify all of the different queries for which the program will have to retrieve sequence data. For each query that is inputted, the user is mandated to provide the chromosome number of the query, the start and end position on the chromosome, and the length of the oligos that will be created from the DNA sequence of this region. The user also has the option of specifying 3 and 5 prime padding (the default values are 0), the organism from which the DNA sequence should come (the default organism is Homo sapiens), and the database from which the sequence data is retrieved (the default database is the Genome Browser at a leading university); if any of these values are unspecified then the default values are used. The user may enter several different queries and must provide a name for each different query.

The second step involves entering several configurations, which are what the program uses to select oligos. A configuration must be given a name by the user, and it consists of any combination of several different filters, which will reject oligos that do not meet certain criteria. Except for the Fail Score Filter, all these filters were developed by the team working on SolPicker. The user may elect to include any of the following filters in the configurations that are created.

- **Lowercase Filter:** There are many DNA sequences that are present in hundreds or thousands of different locations within a given genome. These sequences are called repeat regions. Oligos that contain long repeat segments do not serve as useful probes because if they capture a part of the target sequence, even though the sequence that is

captured may be correct, the researcher has no way of knowing if it really represents the correct query or if it is simply an identical segment from a part of the genome that the researcher is not interested in. Additionally, when the BLAST filter (see below) is called upon later, the process can be extremely time-consuming, if not impossible, if it is done on an oligo with a long repeated segment. Most genome browsers/databases will denote repeated segments by making these segments lowercase letters as opposed to uppercase. Researchers that want to filter out oligos with repeated sections can select this filter whenever they create a new configuration, and they can also specify the maximum acceptable number of lowercase letters (the default value of 0 is used if no value is specified).

- **N-Count Filter:** There are many regions of the genome, especially ones near centromere and telomere regions, which researchers still have trouble sequencing because of the way these segments are shielded by the histones. Genome browsers will usually denote such regions by using the letter 'N' instead of 'A', 'C', 'T', or 'G'. Researchers that want to filter out oligos with unknown segments can select this filter when they create a new configuration, and they can also specify the maximum acceptable number of occurrences of the letter 'N' (the default value of 0 is used if no value is specified).
- **Enzyme Filter:** Sometimes researchers use restriction enzymes when conducting DNA experiments. In such cases an oligo becomes useless if it contains the DNA sequence that the enzyme cuts because there will be nothing for it to hybridize – all of the sequences will have been cut by the enzyme. Researchers using enzymes can specify the names of the enzymes being used and the sequences of the binding sites for these

enzymes. This allows the program to filter out any oligos that complement any of the enzymes in either the forward or reverse direction.

- **G-C Percent Filter:** The percentage of guanine and cytosine bases affects certain practical concerns related to microarray capture. The bond between guanine and cytosine bases is much stronger than that between adenine and thymine. Therefore, the temperature necessary for removing the DNA probes from the microarray is directly proportional to the percentage of G-C base pairs. The researcher can choose to filter out oligos that would require a temperature that is either too high or too low, and they can also specify the minimum and maximum acceptable G-C percent (the default values of 45% and 60% are used if no values are specified).
- **Fail Score Filter:** Certain other companies have created a metric for rejecting oligos. The metric penalizes oligos for having unequal proportions of the four different base pairs, for having a removal temperature that is outside a specified range, and for possessing homopolymers (long sequences of identical bases). Users can choose to apply this filter.

To the best of our knowledge, no other software provides this many combinations of filter and capabilities and hence allows the researcher to narrow down their criteria to an extent that was not possible previously. While the lowercase filter has been utilized by other programs, the other filters, especially the N-Count and Enzyme filters have not been used the way we have. The fail score filter is a proprietary filter developed by a biotechnology company as a commercial product. This filter is heavily influenced by an oligo's GC content. While both SolPicker and any programs designed by this other company do use the fail score filter, SolPicker is unique in

that it allows the user to supplement or replace this filter with other filters that are publically available, as users may not want to use or may not be able to use the proprietary fail-score filter.

Additionally, there is a BLAST (Altschul SF 1990) filter. Basic Local Alignment Search Tool (BLAST) is an algorithm that has been developed over the years by many bioinformatics researchers. The algorithm takes a query sequence as input and compares it to a library or database of sequences. BLAST then produces output containing all the different genomic sequences (hits) within the library which closely resemble to query sequence. SolPicker runs every oligo through BLAST and checks that the oligo meets two criteria: First, the oligo must produce a unique hit (that is, it must closely resemble only one segment of the genome being analyzed). Much like the lowercase filter, this ensures that the probe will only capture the genomic segment that the researcher is looking for and not an identical segment from a different region. The reason that this must be used in conjunction with the lowercase filter is that when genome browsers mask repeated regions to lowercase, they only mask the regions that appear several hundred or even thousands of times in the genome. However, there are some sequences that are present in only a few locations, but these oligos would still be poor probes because it is impossible to tell if they hybridize the correct segment of the genome or an incorrect segment with an identical sequence. Checking for unique BLAST hits ensures that oligo sequences are present in only one part of the genome and will not hybridize the wrong part of the target sequence. The second criterion is that the oligo must have an align length of at least 40. Align length is the number of consecutive nucleotides in the target that match the query sequence. Oligos with short align lengths are poor probes because they aren't as effective at binding to sequences that they should hybridize. The program automatically runs every oligo through this filter as a standard feature. The incorporation of the BLAST filter to results narrowed down by

previous filters provides very powerful capabilities and differentiates the SolPicker from other similar programs now available. The BLAST filter ensures unique oligos that will bind will, which are essentially the most important elements of an oligonucleotide probe. This allows SolPicker to provide the researcher with a very powerful capability that they have not had before. Figure 7 gives a visual view of BLAST.

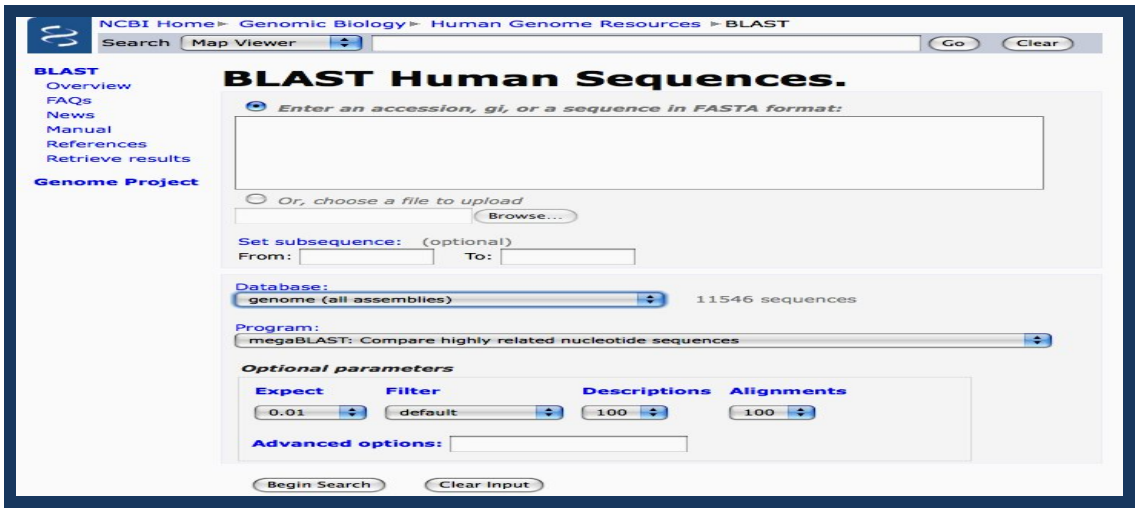

Figure 6-Basic Local Alignment Search Tool (BLAST)

After adding several configurations that contain many different combinations of filters, the user must specify the jobs that the program should do. A job consists of a query and a configuration of filters that will be applied to that query. Our design allows the researcher the flexibility to apply one configuration to several different queries, or apply several different filters to the same query, without having to specify a given query or configuration multiple times.

In addition to manually assigning tasks, the user may input a text file, formatted in a certain way, that would contain all the queries, configurations, and jobs. Due the availability of the variety of input methods, the approach in earlier programs was to develop separate execution modules for each type of input, which is both inefficient and difficult to change. We got around this problem by parsing an Extensible Markup Language (XML) file that is created when the

program starts and is updated as the program progresses. Whenever the user manually inputs a configuration, query, or job, or when the user inputs a text file or some other type of file, the XML file that is created at the start of the program is updated to include the new information. Once the user is done inputting information, the program parses this XML file and performs the appropriate tasks. This makes it much easier and more efficient to add new types of input, because the execution mechanism doesn't have to change since it simply parses an XML file regardless of what input method the user chose to use. Another distinctive and unique characteristic of SolPicker is the fact that before it parses the XML file it validates it against a schema. A schema defines the grammar for a piece of XML code and expresses constraints on the structure and content of the code. It represents a very high level of abstraction in programming. Some programmers in this field eschew the use of a schema because it is so abstract and tedious to produce, but it is still worthwhile to implement because it allows for a high degree of error checking and is generally considered to be a good practice in software development.

The functional output of the SolPicker is in a form that can be readily used by researchers. Once the input stage is finished, the program will retrieve the appropriate sequence data, split each sequence into all possible oligos of the length specified by the user, run each oligo to the appropriate filter, and separate the valid oligos from the invalid ones. For output, the program prompts the user for the file system directory where all the output should be put. The program creates four FASTA files (FASTA is the standard format in bioinformatics to represent sequence data). The first file contains all the sequences of the queries that were entered by the user; the second file contains all the oligos that were created from all the query sequences; the third file contains all the oligos that were rejected at some point in the process; the fourth file

contains all the valid oligos that were not rejected by any of the filters that they went through. SolPicker connects to the backend database through HTTP requests as this is the preferred mode of connectivity to the current database. The design will allows us to easily change the connection protocol or database accessed very easily. Figure 8 illustrates the process.

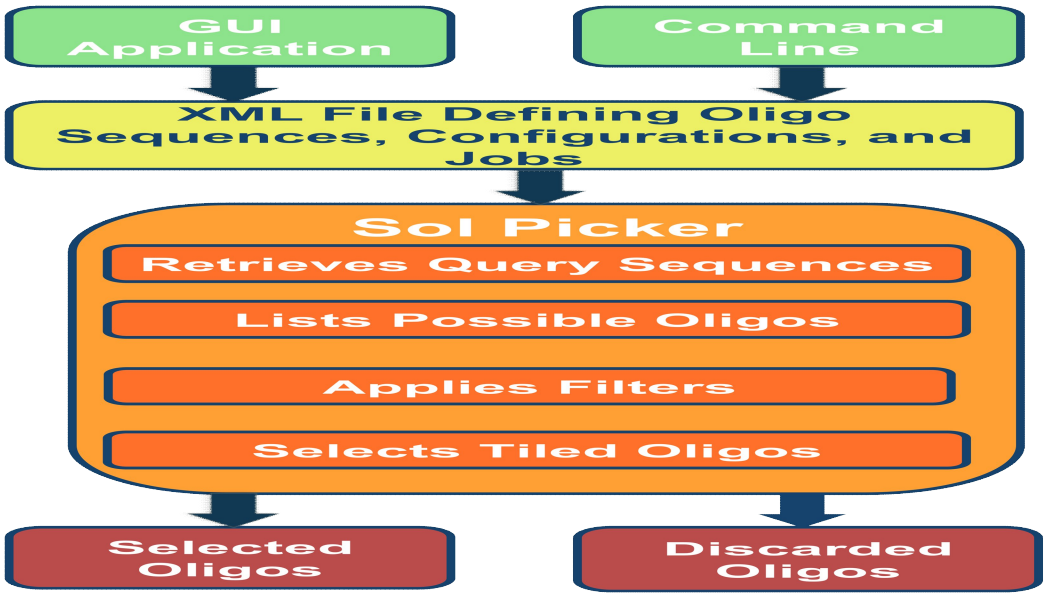

Figure 7-Diagram illustrating the process by which SolPicker runs.

6. Discussion and Implementation:

SolPicker is a software suite that is capable of analyzing a gene or genomic segment and picking short subsequences that can be just as effective probes as the entire sequence. SolPicker is able to successfully retrieve sequence data from the genome browsers and select oligonucleotides. The file with valid oligos contains only oligos that are unique, have a good proportion of G-C base pairs, and contain minimal unknown regions. Because the output gives information on each oligo, the user can check the statistics for each oligo and see that the program has in fact retrieved the correct sequence and filtered oligos properly.

The creative use of filters reduces the amount of data that the researcher will later have to deal with, along with decreasing the costs necessary for creating the probes. As has been emphasized earlier, SolPicker, to the best of our knowledge is the only product of its type to use BLAST to develop unique sequences. This ensures that output is unique and meets multiple selection criteria rather than just one. It also has several other capabilities that make it very easy to use. It has also been designed with scalability and adaptability in mind and hence can be easily modified and enhanced though the use of standard and structured programming practices.

We validated and tested our results returned by the SolPicker by generating sequences based a set of test data. For a test trial we examined a segment of sequence data from chromosome X. We then manually validated the results by actually checking that the sequence was correct, that the program produced correct values when running the oligos through each of the filters, and making sure that the program discarded oligos correctly according to the parameters entered into the GUI.

Our final validation step now currently underway is to test the output by actually creating probes containing the sequences corresponding to the sequences of the oligos chosen by the program. By testing these probes in the context of a microarray experiment we can confirm the efficacy of the methods used by SolPicker to select oligos and completely validate our design. However given the thought that went into the design, development and validation we are confident that our software suite will perform equally well in actual user conditions as it did in test conditions. Additionally, the Perl prototype has been used successfully used to design custom microarrays which were successfully used in research projects.

As a part of our implementation and roll out strategy, we first intend to make it generally available to researchers in this institution and to other researchers and institutions shortly. The

current practice has been to provide the program to the researchers who run it from their desktop. However our design and architecture will allow use in a client-server framework, in which researchers run the tool from a server or the internet. This would allow researchers to be able to use the latest version of the software and concentrate on their research rather than having to write software to meet their needs. Also this will make these advanced tools easily available to a wide range of users, many of whom do not have access to such tools.

7. Conclusions/Future Research:

The effort to develop tools that will help extract and isolate DNA sequences from large databases has been fragmented and limited. Most of these tools were developed by individual researchers to meet their specific needs or as proprietary tools. Until today there has not been an attempt to create a standard framework under which these tools can be created, enhanced and disseminated easily. SolPicker can be characterized as an innovative software program designed to help scientists quickly retrieve unique oligos that they need for the microarray experiment, giving them capabilities they had not had before. It greatly extends current technology as it scalable, extensible and provides many user-friendly features and focuses on retrieval of unique sequences. It also for the first time allows for the creating of framework that others can add to

Based on the success of the suite so far, the team has identified many enhancements and features that can be added to the tool. Currently SolPicker is being expanded to accept more types of inputs, including XML and text files as well as different forms of outputs. In the future, it could be expanded to receive messages using a messaging systems, or take inputs directly from equipment such as sequencers or from programmable computer chips or RFID signals. SolPicker is unique when compared to other similar software as it can be made to support varied

inputs due to the fact that the program executes through an XML. Other future research approaches can also focus on adding additional error checking capabilities and to allow for the tool to recognize patterns and provide suggestions to the user through the addition of some analytical and intelligence capabilities.. For example, the program could be modified to record what sequences are commonly asked for and prompt users if they ask for sequence that is uncommon or not available. We could also change the program so that it can be accessed from the internet rather than from the user's computer or run a client server framework. Finally, more research could be done regarding the order in which the filters are applied. One goal is to make sure that the filters discard as many oligos as possible as early as possible. This cuts down on run-time because while the first filter will always have to examine every single oligo, if that first filter discards many oligos then that reduces the total number of oligos that must be examined by subsequent filters. We are currently researching the best order of filters.

References:

1. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990). "Basic local alignment search tool ". *J Mol Biol* **215** (3): 403–410. doi:10.1006/jmbi.1990.9999

2. Emily H. Turner, Sarah B. Ng, Deborah A. Nickerson, and Jay Shendure. Methods for Genomic Partitioning. Annual review of genomics and human genetics (2009) URL. http://dx.doi.org/10.1146/annurev-genom-082908-150112

3. Hodges et al. Hybrid selection of discrete genomic intervals on custom-designed microarrays for massively parallel sequencing. Nat Protocol. (2009) vol. 4 (6) pp. 960-74. http://dx.doi.org/10.1038/nprot.2009.68

4. Hodges et al. Genome-wide in situ exon capture for selective resequencing. Nat Genet (2007) vol. 39 (12) pp. 1522-7 http://www.nature.com/ng/journal/v39/n12/abs/ng.2007.42.html;jsessionid=18f0rmoglnwgm

5. Lipshutz, RJ, Morris, D, Chee, M, Hubbell, E, Koza,l MJ, Sha,h N, Shen, N, Yan,g R, Fodor, SP. "Using Oligonucleotide Probes to Access Genetic Biodiversity." Biotechniques: 19(3): 442-7. <http://www.ncbi.nlm.nih.gov/pubmed/7495558>. 1995

6. P. Rice, I. Longden, and A. Bleasby, "EMBOSS: the European molecular biology open software suite (2000)," Trends in Genetics, vol. 16, no. 6, pp. 276–277, 2000.

7. Shendure and Ji, Next-generation DNA sequencing. Nat Biotechnology (2008) vol. 26 (10) pp. 1135-45. http://dx.doi.org/10.1038/nbt1486