TileSoft: Sequence Optimization Software For Designing DNA Secondary Structures

Peng Yin* Bo Guo Christina Belmore Will Palmeri Erik Winfree*
Thomas H. LaBean* John H. Reif*

January 28, 2004

Abstract

DNA is a crucial construction material for molecular scale objects with nano-scale features. Diverse synthetic DNA objects hold great potential for applications such as nano-fabrication, nano-robotics, nano-computing, and nano-electronics. The construction of DNA objects is generally carried out via self-assembly. During self-assembly, DNA strands are guided by their sequence information into secondary structures to maximize Watson-Crick pairing of their bases and thus minimize the free energy of the resultant structures. A crucial computational problem in constructing DNA objects is the design of DNA sequences that can correctly assemble into desired DNA secondary structures. However, existing software packages only provide unintuitive text-line interfaces and generally require the user to step through the entire sequence selection process, which could be time-consuming and tedious. *TileSoft* described in this manuscript improves on previous software by delivering the following features: 1) Its graphical user interface renders the molecular architect the ability to define DNA secondary structure and accompanying designing constraints directly on the interface as well as the ability to view the optimized sequence information pictorially. 2) Its fully automatic optimization module relieves the user of the drudgery of manually dictating the sequence selection process, and its evolutionary algorithm produces satisfactory results efficiently. 3) Its graphical user interface and its optimization module are smoothly integrated from user's perspective, while they are at the same time well separated in terms of software architecture, making each amenable to future improvements without negatively affecting the other.

*** Contact**: P. Yin: py@cs.duke.edu, E. Winfree: winfree@caltech.edu, T. H. LaBean: thl@cs.duke.edu, J. H. Reif: reif@cs.duke.edu

1 Introduction

DNA is a common building material for the construction of molecular scale objects with nano-scale features. As such a material, DNA possesses the following appealing properties: 1) its well-defined Watson-Crick complementarity and its immense information encoding capacity, 2) its minuscule size (diameter of $\sim 2\ nm$ for its double stranded B form), 3) the stiffness of its double stranded form (persistence length of $\sim 50\ nm$ or 15 full turns) as well as the flexibility of its single stranded form (persistence length of $\sim$1 nm, or 3 bases), 4) the ease to manipulate it with a rich set of enzymes, and 5) the topological diversity of DNA constructions achievable by a combinatorial arrangement of branched DNA molecules. The above features make it an ideal material for constructing nano-objects such as closed nano-objects, nano-lattices, and nano-mechanical devices. These nano-scale DNA constructions in turn hold great potential for applications in nano-fabrication, nano-robotics, nano-computing, and nano-electronics, among others. For a review of DNA as nano-scale construction material, see [22, 19, 13].

In contrast to the conventional "top-down" approach commonly used to construct macro- and micro-scale objects, DNA constructions are conducted by a "bottom-up" approach known as *self-assembly*, utilizing the Watson-Crick self-complementarity. During self-assembly, synthetic DNA strands are mixed together in solutions containing positive metal ions, such as magnesiums, and are run through the annealing process of cooling slowly the system from high temperature around $90\ ^\circ$C to low temperature as low as $4\ ^\circ$C. As a result, secondary structures are formed as guided by the sequence information of these DNA strands.

To make large structures such as DNA lattices, a two step annealing procedure is sometimes invoked. In the first step, different sets of synthetic DNA strands are mixed to form small secondary structures referred to as *tiles*, by bringing the system from a high temperature (well above the melting temperature of individual tiles) to a medium temperature (below the temperature of individual tiles). Tiles formed during the first step possess *sticky ends*, which are single stranded overhangs extending from the ends of double helical nucleotide fragments. In the second step, tiles are mixed together and the system is further cooled from medium temperature to a low temperature (below the melting temperature of the sticky ends of individual tiles). Consequently, the complementary sticky ends of individual tiles hybridize with each other and the tiles are "glued" together into large structures. It is also possible to combine these two steps into one. Experimentally demonstrated two dimensional lattices include those made from double-crossover (DX) DNA tiles [27], rhombus tiles [16], triple-crossover (TX) tiles [14], and 4x4 tiles [28]. Figure 1 and Figure 2 give examples of DX and TX tiles respectively.

A key computational challenge in DNA self-assembly is to design DNA sequences that form the prescribed secondary structures. Abundant efforts have been made in the DNA computing community to develop algorithms and software that produce a pool of DNA sequences such that each sequence is of maximal difference from others [2, 6, 7, 5, 9, 10, 24, 12, 17, 11, 3, 15, 18, 20, 25, 26, 23, 8]. These sequences can either be used as DNA libraries with unique sequences or associate into linear DNA molecules. For a good survey on DNA sequence design, see [4]. In contrast, research efforts attacking the problem of designing DNA sequences that uniquely assemble into desired *secondary* structures are comparatively scarce. The current software tool widely used is a semi-automatic software package called *Sequin*, which was developed by Seeman in 1990 [21]. Though being a very useful tool, Sequin requires a significant amount of manual operation by the molecular architect in the sequence optimization process. In addition, Sequin only provides a text interface that is not as user-friendly and intuitive for inputting and displaying the graphical nature topological information of the desired secondary structures. As such, desirable improvements of Sequin include relieving the molecular architect of the laborious process of manually carrying out the designing tasks and providing a graphical user interface that can take in the input as well as display the final designed sequences. *TileSoft* described in this manuscript is such an attempt.

TileSoft has two components, a graphical user interface (GUI) developed in $Java$ and an optimization module written in C. The user friendly GUI allows the molecular architect to specify the tile templates directly on the interface as well as to display the optimized sequence information pictorially. The optimization module is fully automated and hence frees the user completely from potentially time-consuming and tedious manual selection of the DNA sequences. It implements an evolutionary algorithm that produces satisfactory results efficiently. We also note that though the GUI and the optimization module are smoothly integrated from user's perspective, they are actually well separated in terms of software architecture. Hence the two components can be improved independently. Currently, TileSoft runs on a FreeBSD Unix platform.

The rest of the paper is organized as follows. We first describe the functions of the GUI from the user's perspective in Section 2. Then we present the design, implementation, and evaluation of the optimization module in Section 3. We close with a discussion of the software and future work in Section 4.

2 Graphical User Interface

The GUI is the I/O interface between the molecular architect and the optimization module of the software. Before describing the graphical interface, we first describe the procedure of designing a tile set using TileSoft.

2.1 Tile Set Design Procedure

To design a tile set, the molecular architect first decides and inputs to the GUI the topological structure of each single tile as well as constraints on the DNA sequences constituting the tiles – *e.g.* the complementarity of paring sticky ends, the specific enzyme recognition site(s), the specification of the base composition of the junctions, *etc*. For a concrete example, see Figure 3. These information is passed from GUI to the optimization module, which subsequently performs the optimization task. The optimized sequences can in turn be displayed on the graphical user interface.

2.2 GUI Overview

Developed in $Java$, the graphical user interface (GUI) provides a graphical representation of the DNA strands. The user can set the length of each strand, define junctions between nucleotide helical fragments, assign bases, specify various constraints directly on the graphical representation of helical fragments. These information is automatically encoded into text vectors, which in turn serve as input to the native optimization module written in C. The optimized DNA strands can be loaded to the window so that the structure is shown graphically and can be modified for further optimization. The user can also monitor the optimization process by inspecting sequence information directly displayed on the graphical user interface during the optimization process.

The default view of the graphical interface is shown in Figure 4. It has the following three major components: main window, menus bar, and buttons bar.

- The **Main window** gives the graphical representation of DNA strands. Two full turns of the DNA helical fragments are represented by 21 bases on each single strand comprising the helices. All bases are set to N as default.
- The **Menus** bar at the top of the window contains three menus. The *File* menu contains items that can save or load a single tile or an entire tile set; the *Edit* menu allows the user to undo or redo the previous manipulation on the DNA strands; the *Tile* menu provides control of the following actions: returning to default window, adding current tile to the tile set, optimizing the tile set by calling the native C program, stopping optimization, and displaying optimized sequences.
- The **Buttons** bar at the bottom of the window contains buttons useful for defining individual tiles. Once a button is selected (activated), corresponding manipulations on the DNA strands can be performed.

2.3 Functions of GUI

The following functions are provided through the graphical user interface.

- **Defining junctions.** When the *Define Crossover* button is activated, the user can define crossovers between helices, by clicking sequentially the two bases to be connected in 5' to 3' order (Figure 5). The color of the resulted new strand will be identical to the color of its 5' end base.
- **Setting endpoints.** Figure 6 and Figure 7 illustrate how the user can set 5' end and 3' end of a strand when the *Set 5' End* and *Set 3' End* buttons are activated respectively. By setting the 5' end and 3' end of a DNA strand, the user specifies the length of the strand, and the unused segment of the strand is deleted automatically. Note that the deleted bases are turned in to gray color.
- **Editing bases.** The user can directly input the base values for a strand by typing when the *Edit Bases* button is selected (Figure 8). Typing more than one character edits consecutive bases in the 5' to 3' direction along the strand. Besides A, T, C, and G, degenerate bases are also allowed so that user can restrict a position to a certain subset of base values. (see Table 1) .
- **Setting WC constraints.** At the default status, all helical DNA segments are required to be Watson-Crick base paired. However, under some circumstances, it is desirable to relieve certain subsegments of a DNA helix the base pairing requirement. The button *Toggle Pairing State* is for this purpose. Once this button is activated, the user can define the subsequences that are not required to be Watson-Crick base paired by simply clicking on the starting and ending bases of the subsequences (Figure 9). The bases contained in such subsegments will assume a plain-text font instead of the default bold font.
- **Setting and showing EQ constraints.** Figure 10 shows the procedure for defining two equal sub-sequences. When the *Set EQ Constraints* button is selected, clicking on two bases in one strand defines the starting and ending points of the first sub-sequence, and a click on another base on either this strand or another strand automatically delineates the second sub-sequence with the same length and direction as the first one. The bases in both sub-sequences then become italicized.

  Since many sets of equal sub-sequences may exist, it is necessary to distinguish between them. The button *Show EQ Constraints* is for this purpose. When it is clicked, a small window is brought up (Figure 11) that contains multiple buttons, with each representing a set of equal sub-sequences. When one of these buttons is clicked, the corresponding sub-sequences will be highlighted in purple.
- **Defining a tile set.** The software permits the user to work on one tile at a time. When the user finishes the specification for a tile, he or she can add the tile to the tile set by selecting the menu item *Add to Tile Set* under the *Tiles* menu. After the current tile is added to the tile set, a new tile with default configuration shows up in the main window.
- **Performing optimization.** When the item *Optimize Tile Set* under *Tile* menu is selected, all the specifications of the tile set are passed to the native *C* program which subsequently performs the optimization. During the optimization process, an evaluation of the sequences being optimized is printed out on the screen so that the user can monitor the progress. The optimization process will go on until the user decides to stop it by hitting the *Stop Optimization* button. The resulting DNA sequences are saved and can be displayed graphically as explained below.

- **Displaying optimized tiles.** A separate window titled *Tiles* contains buttons each representing an individual tile in the tile set, as shown in Figure 12. Clicking on one of these buttons shows the corresponding tile in the main window. Each tile can be displayed either during the optimization process or after the optimization is done.

3 Optimization Module

We first describe the communication between the GUI and the optimization module, two well separated yet smoothly integrated components of TileSoft.

3.1 Communication Between GUI and the Optimization Module

The GUI processes the tile set template and constraints specified by the molecular architect, and encodes the information in plain text form({ **Peng**: Erik, should we describe the encoding scheme? }) and stores it in some template and constraint files. When the user clicks the start-optimization button of the GUI, the GUI fires up the optimization module. The optimization module then reads the encoded information from the template and constraint files, and performs the optimization tasks on the template while observing the constraints. The optimized sequence information is stored in a data file. When the user hits the update-display button on GUI, GUI reads information from the data file and displays it.

The optimization module employs an evolutionary algorithm to find the best solution to the optimization objective function. We first describe the objective function to be optimized.

3.2 Objective Function and Its Evaluation

The objective function in the current version of TileSoft consists of two weighted factors, the count of unwanted complementary sequences (*spurious matches*) and the count of to-be-avoided sub-sequences. However, we note that other factors such as the melting temperature difference among sequences and the free energy of the tiles, could also be factored into the objective function, though not yet implemented.

The first factor is the count of unwanted matches of the DNA sequences measured by two weighted matrices. The first matrix calculates the weighted number of unwanted complementary regions of length 3 to 8 at intra-molecular (same strand), inter-molecular intra-tile (different strands, same tile) and inter-tile level (different strands, different tiles). The second matrix gives the weighted number of unwanted matching regions containing a single mismatch of length 5 to 10 at (again) intra-molecular, inter-molecular intra-tile and inter-tile level. The intra-molecular level matches receive the largest weight while the inter-tile level receives the least weight.

The second factor is the count of subsequences to be avoided. The definition of these bad subsequences is based on rules of thumb, such as “do not have 3 Gs in a row”, “avoid many purines in a row”, and “do not have long AT runs”, *etc*.

{ **Peng**: Erik, feel free to elaborate on other aspects of the optimization module. }

3.3 Evolutionary Algorithm

During the past 30 years there has been a growing interest in evolutionary algorithms. Evolutionary algorithm is often a good choice for NP hard optimization problems whose fitness landscape possesses many local optima. DNA tile set design problem is one such instance.

An evolutionary algorithm maintains a population of DNA sequences, which are generated randomly during *initialization*. During *selection*, the fittest DNA sequences are chosen for reproduction, based on their score according to the fitness function – *i.e.* the objective function described above. These individuals are used to generate new individuals via *mutations* and *crossovers*, and the newly produced individuals are *reinserted* into the population. The process is repeated until meeting some *termination* condition. We describe below in detail the implementation of the evolutionary algorithm in TileSoft.

Initialization. In TileSoft, one tested initialization strategy (simple-initialization) is to replace the degenerate bases in the templates with A, T, C, or G randomly. This process is repeated until enough individuals are produced. However, since we want to avoid the appearance of spurious complementary sequences in the DNA structure, it might be better to consider this factor in the initialization process. Thus, instead of randomly assigning a base to each degenerate position, we check the number of WC complements with the sub-sequences that contain this base in the DNA template. The size of sub-sequences ranges from 3 to 8 bases. From the candidate bases, the one with the smallest spurious match value is selected. Empirical study shows that the latter approach (match-initialization) is slightly better and hence is used in TileSoft. (see Subsection 3.4).

Selection. The general principle of selection is that sequences with better fitness scores have a higher probability of being selected. *Selective pressure*, which is defined as the probability of the best individual being selected compared to the average probability of selection of all individuals, is a crucial term in selection. Stagnation happens when the selective pressure is too small, while premature convergence appears where selection has caused the search to narrow too quickly. We chose to use a modified version of the *Fitness Proportional Selection* for TileSoft [1]. Fitness Proportional Selection allocates the reproduction chances proportional to the ratio between the fitness of the individual to the average fitness of the population. The selection scheme used here is modified to overcome the scaling problems of the proportional fitness assignment and can be expressed as follows:

$$SP_i\% = (2B - V_i)/((2B - V_1) + (2B - V_2) + ... + (2B - V_n)) \times 100\%$$

where SP is the probability of selecting individual i, n is the number of individuals in the population, B is the worst fitness value in the population, and V_i is the fitness value of the individual i. The new value $2B - V_i$ is used in the fitness proportional selection. It is easily shown that the above formula gives a selection pressure between 0.5 and 2.

Mutations and Crossovers. *Genetic operators* are the exploration tools of evolutionary algorithms. The genetic operators combine the building blocks of fit parents, in the hope that the newly created offspring will inherit the information that made the ancestor fit, and thus become better solutions themselves. The most common and important operators are *crossover*, which exchanges information between parents, and *mutation*, which further perturbs the offsprings. A crossover rate of 60% and a single base mutation scheme is used in TileSoft.

Reinsertion. After selection, crossover, and mutation, offsprings produced from parent population are reinserted into the original population. Here we choose the *Replace worst* replacement policy, which replaces the worst parents.

Termination Condition. Since it is hard to predict how small the best value could be for a specific optimization, we avoid assigning such a value. Instead, the optimization process will continue until the user hits the stop-optimization button.

3.4 Optimization Results and Analysis

To justify the use of evolutionary algorithm in TileSoft as well as to determine some of the critical parameters, we performed some simple experiments.

First, we show that evolutionary algorithm gives better performance than the random hill-climbing algorithm, which was used in an early version of TileSoft. In the same experiment, match-initialization is also demonstrated to be superior to the simple-initialization. Three algorithms are implemented and compared: *EA_Simple_Init*, an evolutionary algorithm with random initialization; *EA_Match_Init*, an evolutionary Algorithm with initialized population of minimized exact complementary matches; *Random*, a random hill-climbing algorithm. The template tiles used to test the above algorithms are those in Figure 3.

Table 4 shows 10 final scores corresponding to 10 runs in each algorithm: the minimum score, the maximum score, the mean, and the standard deviation. From the data, we can tell that *EA_Match_Init* is the best among the 3 algorithms in our test. Henceforth, *EA_Match_Init* is the algorithm used for additional tests.

The effect of various mutation rates was then tested (Figure 13). The performance of the algorithm varies significantly as the mutation rates change. Smaller mutation rates (1/3, 1/10, 1) yield steep curves while larger mutation rates (3, 10) generate smooth curves. This indicates that optimization converges faster with smaller mutation rates. The curves suggest that larger mutation rates might generate good final scores if the optimization is performed for a long time. Considering efficiency and final score, single base mutation is shown to be the best among all mutation rates.

4 Conclusions and Discussions

In this draft, we present a user friendly optimization software that automatically assigns sequences to generate desirable DNA secondary structures.

TileSoft is designed to solve optimization problem for a set of multiple tiles. However, designing a single tile is just a specific case that can also be handled by the software. Thus TileSoft is not limited to be used for the tile set design problem but can be used in more general settings. For example, one can use TileSoft to perform sequence optimization task for molecular mechanical devices.

The current optimization module implements an evolutionary algorithm. Another useful heuristic is as follows: build a default library of optimized *DNA duplex sequences*, using one of the various word design softwares surveyed in Section 1; after the user defines the junctions, start optimization on the DNA segments immediately surrounding the junctions; progressively increase the size of optimized region until reaching a satisfactory optimization score. This heuristic is particularly applicable in designing tasks where long stretches of DNA helical fragments are connected by relatively few junctions, for example, a long rhombus molecule [16]. The advantage of this heuristic is that it can possibly avoid redoing the work of optimizing the long stretches of DNA sequences between the junctions, and hence achieve a potentially more desirable optimum in a shorter period of time. A later version of TileSoft might incorporate this feature.

TileSoft was initially motivated by designing tile sets such as TX and DX tiles that have horizontal layouts of their helices. However, it does not provide as intuitive graphical representation for tiles whose helical fragments are parallel to each other, for example, the 4x4 tile. Hence, it would be desirable to develop graphical user interfaces that can handle more geometrically flexible structures.

References

References

[1] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *the Second International Conference on Genetic Algorithms and their Applications*, pages 14–21,

1987.

[2] E. B. Baum. DNA sequences useful for computation. *unpublished, available under http:// www.neci.nj.nec.com/homepages/eric/seq.ps*, 1996.

[3] A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini. Universal DNA tag systems: A combinatorial design scheme. *Journal of Computational Biology*, 7:503–519, 2000.

[4] A. Brenneman and A. E. Condon. Strand design for bio-molecular computation. *to appear, available at http://www.cs.ubc.ca/ condon/papers/wordsurvey.ps*, 2001.

[5] S. Brenner. Methods for sorting polynucleotides using oligonucleotide tags. *US Patent 5,604,097*, 1997.

[6] R. Deaton, R. C. Murphy, M. Garzon, D. T. Franceschetti, and E. Stevens Jr. Good encodings for DNA-based solutions to combinatorial problems. In *Preliminary Proceedings of the 2nd International Meeting on DNA Based Computers*, pages 159–171, 1996.

[7] R. Deaton, R. C. Murphy, J. A. Rose, M. Garzon, D. T. Franceschetti, and S. E. Stevens Jr. Genetic search for reliable encodings for DNA-based computation. In *First Conference on Genetic Programming*, 1996.

[8] U. Feldkamp, H. Rauhe, and W. Banzhaf. Software tools of DNA sequence design. *Genetic Programming and Evolvable Machines*, 4:153–171, 2003.

[9] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25:4748–4757, 1997.

[10] M. Garzon, R. Deaton, P. Neather, D. R. Franceschetti, and R. C. Murphy. A new metric for DNA computing. In *Conference on Genetic Programming, GP-97, Stanford University: Stanford, California, J. R. Koza et al. (eds.)*, 1997.

[11] N. P. Gerry, N. E. Witowski, J. Day, R. P. Hammer, G. Barany, and F. Barany. Universal DNA microarray method for multiplex detection of low abundance point mutations. *Journal of Molecular Biology*, 292:251–262, 1999.

[12] A. J. Hartemink, D. K. Gifford, and J. Kohdor. Automated constraint-based nucleotide sequence selection for DNA computation. *Biosystems*, 52:227–235, 1999.

[13] T. H. LaBean. Introduction to self-assembling DNA nanostructures for computation and nanofabrication. In *Computational Biology and Genome Informatics eds. J.T.L. Wang and C.H. Wu and and P. P. Wang ISBN 981-238-257-7 World Scientific Publishing Singapore*, 2003.

[14] T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N.C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *Journal of American Chemestry Society*, 122:1848–1860, 2000.

[15] F. Li and G. D. Stormo. Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics*, 17:1067–1076, 2000.

[16] C. Mao, W. Sun, and N.C. Seeman. Designed two-dimensional dna holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121:5437–5443, 1999.

[17] A. Marathe, A. E. Condon, and R. M. Corn. On combinatorial DNA word design. In *The 5th International Meeting on DNA Based Computers*, 1999.

[18] G. Raddatz, M. Dehio, T. F. Meyer, and C. Dehio. Primearray: genome-scale primer design for DNA-microarry construction. *Bioinformatics*, 17:98–99, 2001.

[19] J. H. Reif. DNA lattices: A programmable method for molecular scale patterning and computation. *special issue on Bio-Computation, Computer and Scientific Engineering Magazine, IEEE Computer Society*, pages 32–41, 2002.

[20] A. J. Ruben, S. J. Freeland, and L. F. Landweber. Punch: An evolutionary algorithm for optimizing bit set selection. In *7th International Workshop on DNA-Based Computers*, pages 250–260, 2001.

[21] N. C. Seeman. *De novo* design of sequences for nucleic acid structural engineering. *Journal of Biomolecular Structure and Dynamics*, 8:573–581, 1990.

[22] N. C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.

[23] S. Y. Shin, D. M. Kim, I. H. Lee, and B. T. Zhang. Evolutionary sequence generation for reliable DNA computing. In *the 2002 Congress on Evolutionary Computing*, pages 79–84, 2002.

[24] D. D. Shoemaker, R. W. Davis, M. P. Mittmann, and M. S. Morris. Methods and compositions for selecting tag nucleic acids and probe arrays. In *European patent EP0799897*, 1997.

[25] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi. Developing support system for sequence design in DNA computing. In *7th International Workshop on DNA-Based Computers*, pages 129–137, 2001.

[26] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi. Towards a general-purpose sequence design system in DNA computing. In *The 2002 Congress on Evolutionary Computing CEC'02*, pages 73–78, 2002.

[27] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystalsnature. *Nature*, 394:539–544, 1998.

[28] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.

Wildcard	Bases	Wildcard	Bases	Wildcard	Bases	Wildcard	Bases
A	A	C	C	G	G	T	T
R	AG	Y	CT	W	AT	S	CG
M	AC	K	GT	B	CGT	D	AGT
H	ACT	V	ACG	N	ACGT		

Table 1: Degenerate DNA base wildcards used in TileSoft

Algorithm	Min	Max	Mean	Stddev
EA_Simple_Init	2.06752	2.93312	2.524336	0.256857
EA_Match_Init	2.13888	2.69312	2.351392	0.20191
Random	3.24384	4.21216	3.830928	0.34660

Table 2: Comparison of the algorithms

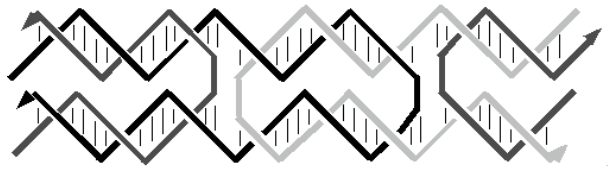

Figure 1: An illustration of a DX tile. A DX tile consists of two double helices connected by holiday junctions. Arrows designate the 3' ends of strands. Vertical bars indicate hydrogen bonds between two strands.

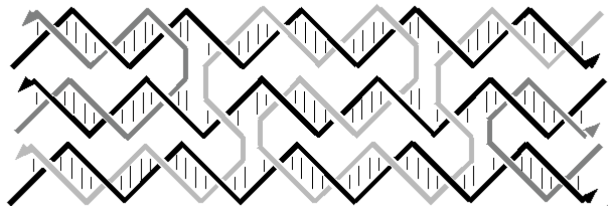

Figure 2: An illustration of a TX tile. A TX tile consists of three double helices connected by holiday junctions.Arrows designate the 3' ends of strands. Vertical bars indicate hydrogen bonds between two strands.

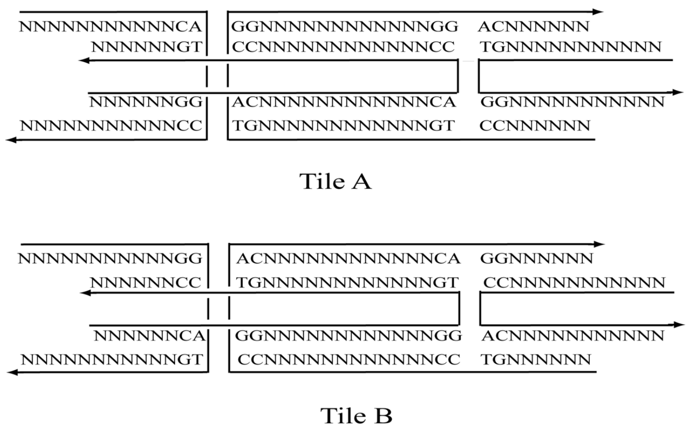

Figure 3: A template for a tile set of two DX tiles before optimization

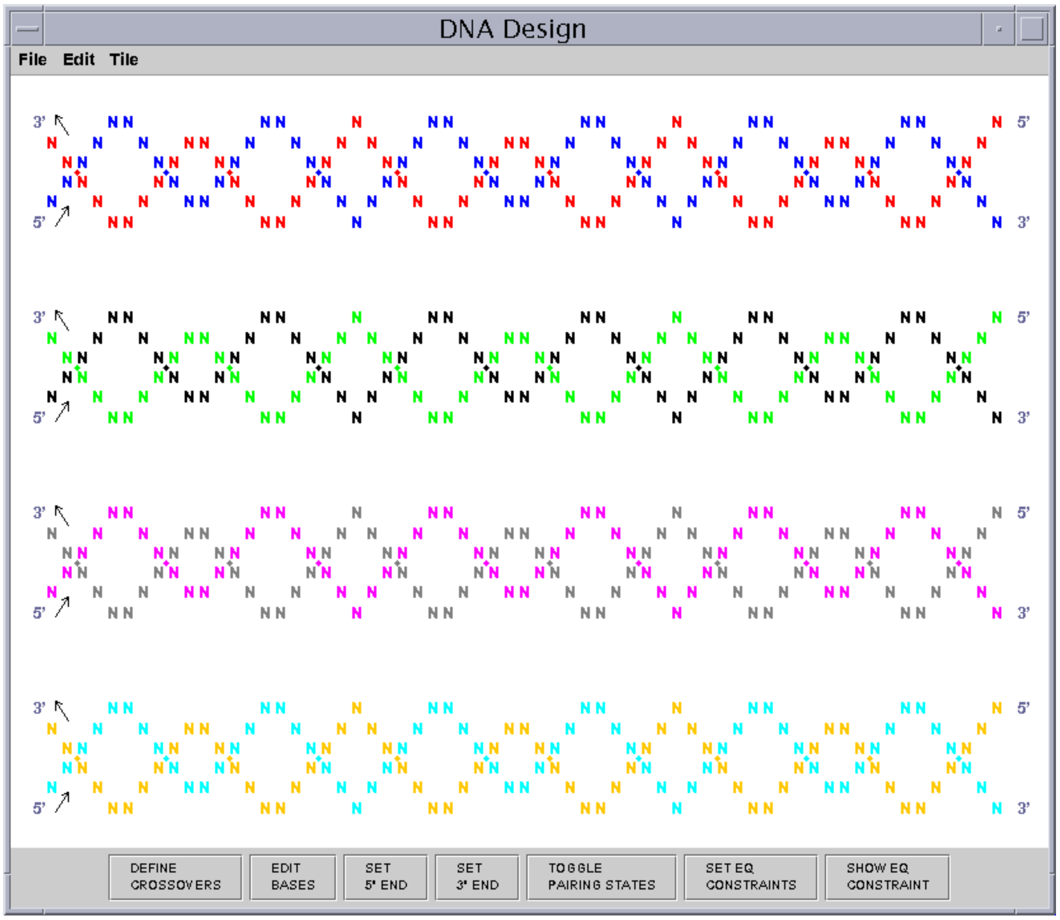

Figure 4: Default window of the Graphical User Interface

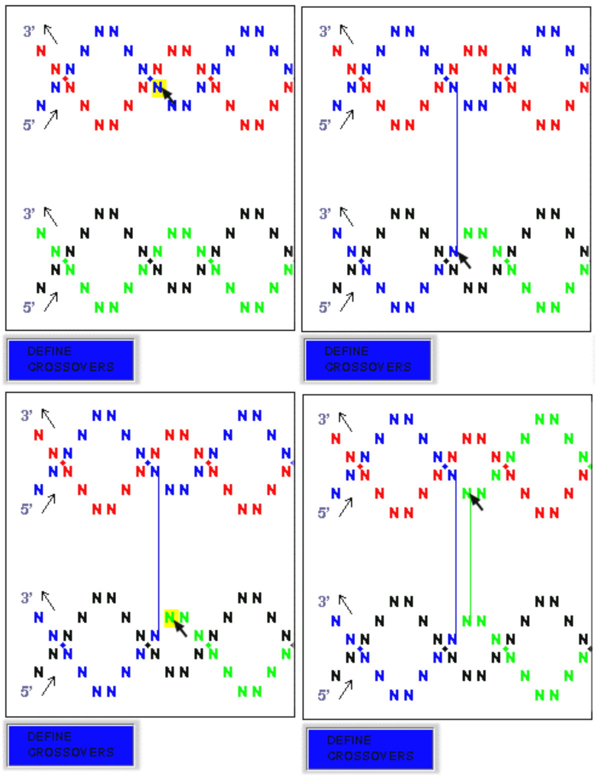

Figure 5: Defining junctions

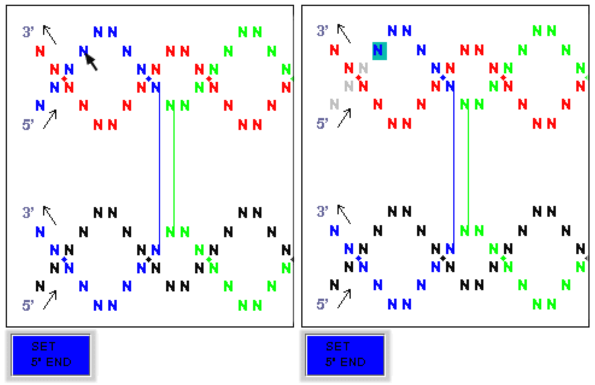

Figure 6: Setting 5' end

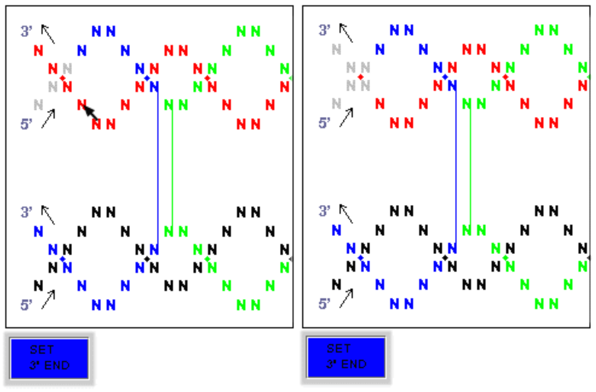

Figure 7: Setting 3’ end

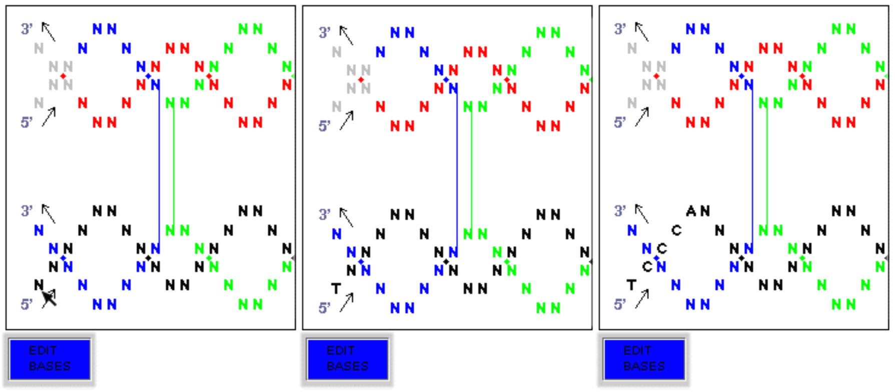

Figure 8: Editing bases

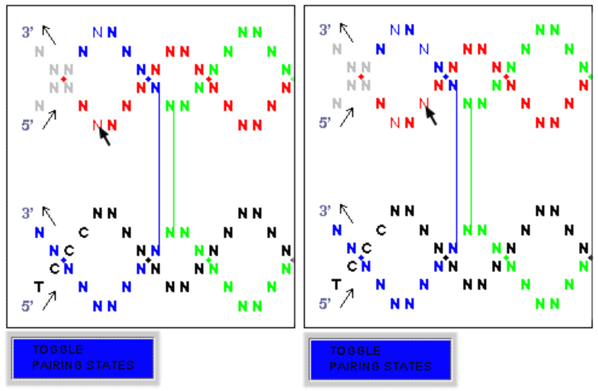

Figure 9: Setting non-Watson-Crick base pairing

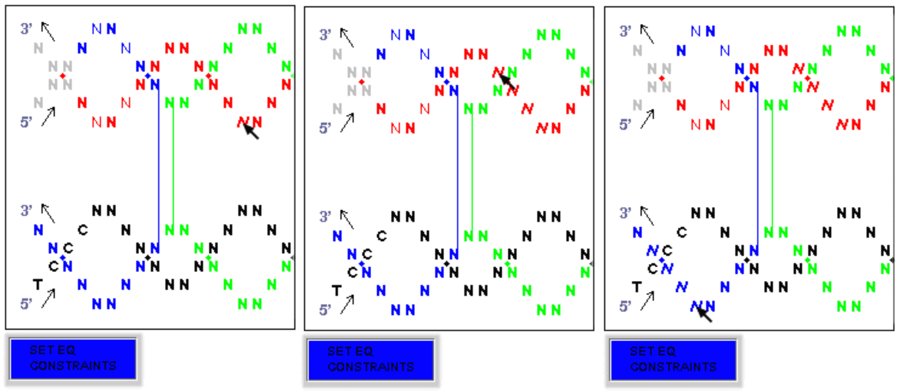

Figure 10: Setting EQ constraints

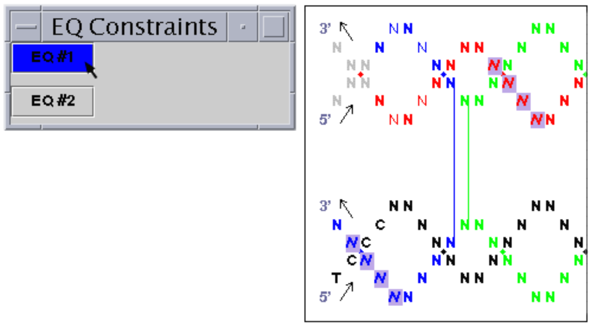

Figure 11: Showing EQ constraints

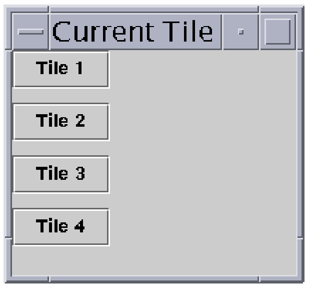

Figure 12: Tile set window

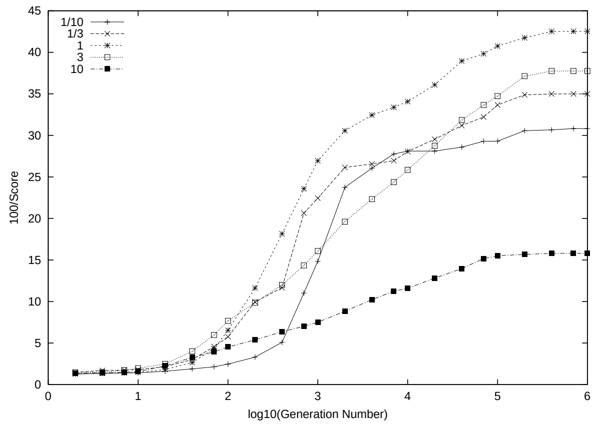

Figure 13: Effects of mutation rate. The curves denote the optimization process of the algorithms with various mutation rates. Score is the average score of 10 runs. The mutation rate is represented by the number of bases to be mutated in each generation. *EA_Simple_Init* is used. The population size is 10, and crossover rate is 60%.