# Software Tools for DNA Sequence Design

UDO FELDKAMP
*Chair of Systems Analysis, University of Dortmund, Germany*
feldkamp@LS11.cs.uni-dortmund.de

HILMAR RAUHE
*Informium AG, Cologne, Germany*
hrauhe@informium.com

WOLFGANG BANZHAF
*Chair of Systems Analysis, University of Dortmund, Germany*
banzhaf@LS11.cs.uni-dortmund.de

**Abstract.** The design of DNA sequences is a key problem for implementing molecular self-assembly with nucleic acid molecules. These molecules must meet several physical, chemical and logical requirements, mainly to avoid mishybridization. Since manual selection of proper sequences is too time-consuming for more than a handful of molecules, the aid of computer programs is advisable. In this paper two software tools for designing DNA sequences are presented, the DNASequenceGenerator and the DNASequence-Compiler. Both employ an approach of sequence dissimilarity based on the uniqueness of overlapping subsequences and a graph based algorithm for sequence generation. Other sequence properties like melting temperature or forbidden subsequences are also regarded, but not secondary structure errors or equilibrium chemistry. Fields of application are DNA computing and DNA-based nanotechnology. In the second part of this paper, sequences generated with the DNASequenceGenerator are compared to those from several publications of other groups, an example application for the DNASequenceCompiler is presented, and the advantages and disadvantages of the presented approach are discussed.

## 1 Introduction

The DNA sequence design problem arises in molecular self-assembly applications like DNA computing and DNA-based nanotechnology [2, 7, 9-11, 14, 18, 20-23, 28, 31-34, 38-43], but also in general purpose lab applications such as probe selection for DNA microarrays or primer design for PCR [3, 5, 17, 19, 24, 35]. Although applications and protocols used are quite different, most requirements posed for the molecules are rather similar. Since specific hybridization is the key process in these applications, DNA molecules are expected to bind to their particular counterpart with good yield, but not to any other molecules. Further, they should not form any undesired secondary structures, hybridization should take place in a given temperature range, and sequences must contain or must not contain certain subsequences.

Meeting just the requirement of specificity of hybridization is already a hard problem, because the DNA sequences must be as dissimilar as possible to each other and to their reverse complement sequences, and there is a multitude of possible sequences to select from. Checking all possible alignments of just two sequences is a time consuming task. Further, it is not clear which formalization of "similarity" is close enough to the hybridization behavior of DNA so as to allow correct modeling of *in vitro* processes, yet simple enough to be computed efficiently. Adding the other requirements does not make sequence design easier, especially when planning to use more than just a handful of sequences.

Thus, the aid of computer programs when searching for sequences that meet all the desired criteria is indispensable. In this paper, we introduce two software tools: the

DNASequenceGenerator for finding a pool of dissimilar sequences, and the DNASequenceCompiler for finding molecules that self-assemble into linear molecules in a specific way determined by a regular grammar.

## 2 Previous Work

Several practical approaches to DNA sequence design have been examined in the past. Seeman and Kallenbach designed sequences using overlapping subsequences to enforce uniqueness [31, 32]. They developed a program to interactively design molecules for 4-way junctions. Deaton et al. used genetic algorithms to generate a set of unique DNA sequences with uniqueness quantified by Hamming distance [9, 10]. Marathe, Condon and Corn chose a dynamic programming approach for DNA sequence design, also employing the Hamming distance [21]. They further described a dynamic programming based algorithm for selection of sequences with given free energy. Frutos et al. developed a so-called template-map strategy to get a large number of dissimilar sequences while having to design only a significantly smaller number of templates and maps [14]. Again, a Hamming-like dissimilarity was used. Hartemink, Gifford and Khodor designed sequences for programmed mutagenesis, which demands similar sequences with only a few mismatches [18]. They use thermodynamic stability estimations instead of string-based distances. Selection of appropriate sequences is done by exhaustive search, which is feasible for short oligomers. Faulhammer et al. described an algorithm for RNA sequence design in the context of solving a chess problem [11]. They did not construct the sequences in one step but repaired them as necessary using a Hamming distance approach to uniqueness. Baum suggested a method to design unique sequences avoiding repetition of subsequences by restricting the choice of nucleotides at the ends of the sequences [2].

Ruben et al. used an evolutionary algorithm (EA) to find sets of dissimilar sequences [28]. Similarity is measured *via* the Hamming distance between subsequences. Additionally, secondary structure folding energy information from the Vienna RNA Package can be integrated into their fitness evaluation. Shin et al. use a multi-objective EA to generate sets of dissimilar sequences (measured by H-measure [15]) that also meet restrictions of melting temperature, GC-ratio etc. [34]. H-measure as well as other objective functions were also studied by Tanaka et al., who applied simulated annealing to optimize the set of sequences [38, 39]. Smith proposed and examined modified De-Bruijn sequences for DNA code words [36]. Ben-Dor et al. made use of circular De-Bruijn sequences from which they cut out unique subsequences [3]. Sequences did not contain A, C, G and T, but only placeholders for strong and weak base pairs, which were instantiated by the bases in a second step. Brenner applied non-overlapping subsequences to generate unique “oligonucleotide tags” for sorting polynucleotides on solid phase supports [5]. Sets of these subsequences were selected so that any subsequence had at least a given number of mismatches with the reverse complement of any other subsequence of the same set. A similar method is proposed by Gerry et al. to design their “zip-code” oligonucleotides [17]. Li and Stormo applied suffix arrays, sequence landscapes and approximate string searching to find probes for gene expression arrays [19]. Raddatz et al. simply shifted candidate sequences along an input sequence to find primer pairs for PCR [24]. Shoemaker et al. developed a pruned tree search algorithm to select sequences by several criteria, e.g. maximal length of common subsequences, melting temperature etc. [35].

Thermodynamics and equilibrium chemistry consideration instead of string-based distance measures is a promising approach to analyze the probability of mishybridizations and for sequence generation [8, 16, 25-27].

For a good survey of DNA sequence design, see [4]. More topics are discussed in the literature, from which we here can only name a few examples. Molecular self-assembly [20, 40] and DNA-based nanotechnology [7, 22, 23, 33, 41-43] are but a small subset.

## 3 Uniqueness and the basic algorithm

The programs described here use a concept of uniqueness that, within a pool of sequences, allows any subsequence of a certain (definable) length to occur at most once in that pool. Also, its reverse complementary sequence must not occur in the same pool. This concept of uniqueness is related to those described by Seeman et al. [31, 32] and Smith [36], but a different approach was chosen. In particular the software described here uses a fully automatic, graph-based algorithm to generate sequences [12, 13].

According to this concept a pool of sequences is said to be *$n_b$-unique* if any subsequence in the pool of length $n_b$ is unique, i.e. a sequence may have a substring of maximum length $n_b - 1$ in common with any sequence in the pool or with any reverse complementary sequence. For example, 20-mers that are 10-unique have common subsequences of at most 9 subsequent nucleotides. Since shorter common subsequences mean less stable mishybridizations, $n_b$ should be chosen as small as possible. Unfortunately, a smaller $n_b$ also means less base strands (the unique strands of minimal length $n_b$) to build the sequences with, so that a minimum for $n_b$ is given by the number and length of sequences to build (see equation (4) below).

The algorithm for generating sequences is implemented with a directed graph, where nodes are base strands and successors of a node are those four strands that may appear as overlapping successors in a longer sequence (see Fig. 1, 2). Thus, a set of $n_b$-unique sequences of length $n_s$ corresponds to a set of paths of length $(n_s - n_b + 1)$ through this graph having no node in common.

Nodes with self-complementary base strands as well as nodes with undesired subsequences are marked as forbidden. Moreover, the user may choose to mark base strands containing start codons, base strands with more than two consecutive guanine or cytosine bases, respectively, or base strands with a GC ratio outside a user-defined range as forbidden nodes.

The basic sequence generation algorithm now draws a start node randomly and tries to find a path of the desired length by randomly choosing successor nodes. The choice is limited to those nodes not marked as forbidden or already used in other sequences. If fraying of the double stranded molecules should be avoided, the choice for the first and last base of a sequence is further restricted to guanine and cytosine. Chosen nodes as well as their reverse complements are marked as used. If none of the four successors of a node is available, backtracking is used to find another path. When a complete path is found, the corresponding sequence is checked for meeting melting temperature and GC-ratio restrictions, and, if necessary, discarded by initiating backtracking. In order to soften a drawback of our approach (see the discussion), the user can also choose a maximum homology between the sequences, which is an upper bound for the difference between the H-measure of any two sequences in the pool and the sequence length, divided by the sequence length. I.e., choosing a maximum homology of 1 means no restriction, while for a value of less than 1 any newly found sequence is checked against all others in the pool and eventually discarded by backtracking. Also the homology between each sequence and the reverse complements of all other sequences is regarded. Once a satisfying sequence has been found it is added to the output pool, another start node is drawn and the path finding routine is restarted, until there are enough sequences in the pool. This basic algorithm can be expanded to meet further requirements of more complex applications (see below).

Note that this concept of uniqueness restricts strictly the number of usable sequences. The number of base strands of length $n_b$ is

$$N_{bs}(n_b) = 4^{n_b} \, . \tag{1}$$

Since complements of already used base strands cannot be used, self-complementary base strands are not allowed at all. The number of base strands that can be used in the generation process is

$$N_{useful}(n_b) = \frac{N_{bs}(n_b) - 4^{n_b/2}}{2} \tag{2}$$

if $n_b$ is even and

$$N_{useful}(n_b) = \frac{N_{bs}(n_b)}{2} \tag{3}$$

if $n_b$ is odd, because there are no self-complementary base strands of odd length.

A sequence of length $n_s$ consists of $n_s$ - $n_b$ + 1 base strands. Thus, the maximum number of sequences that can be built with the algorithm is

$$N_{seqs}(n_s, n_b) = \left\lfloor \frac{N_{useful}(n_b)}{n_s - n_b + 1} \right\rfloor . \tag{4}$$

This estimate is an upper bound because it does not include the constraint of the base strands having to overlap. Additional requirements such as GC-ratio, melting temperature, the exclusion of certain subsequences etc. decrease $N_{seqs}$ further. Experiments have shown that the algorithm is capable of generating over 80 % of this maximum number of sequences for typical sequence and base strand lengths [13].

## 4 DNASequenceGenerator

The program DNASequenceGenerator [13] employs this basic algorithm to iteratively fill a pool with sequences that meet given logical and physical requirements. The main window of the GUI shows the content of this pool. The user can add, import or export sequences to and from it. E.g., one might import sequences already available in the lab and add new sequences that are compatible in terms of uniqueness, melting temperature and GC ratio.

The process of constructing sequences is controlled by using the "Sequence Wizard". This short series of dialog windows enables the user to:

a) import or manually add sequences, e.g. when using the program to expand an existing set of sequences

b) import or manually add sequence templates that are completed to full sequences if possible. Sequence templates use a simple notation: The preset nucleotides (e.g. of a restriction site or other functional subsequence) are specified normally, while an `n` stands for a position to fill. E.g., if given the sequence template `nnnnaacgttnnnn`, the generator replaces the leading and rear four `n`s with nucleotides.
c) generate sequences completely *de novo*.

A pool of sequences can be built iteratively invoking the sequence wizard repeatedly using different parameter sets. The major expansion of the basic algorithm is the consideration of preset nucleotides that predefine the choice of the successor nodes, thus guiding the path finding procedure.

The user can also use the DNASequenceGenerator as a melting temperature ($T_m$) calculator for whole sequence pools. After importing the sequences to the pool, $T_m$ is calculated automatically for each sequence. After changing the pool conditions, $T_m$ will be re-calculated for all sequences in the pool.

Pool conditions that can be parameterized by the user are sample concentration, monoionic salt concentration and formamide concentration. The methods to estimate the melting temperature that can be chosen from are the following.

- Wallace rule: 2 °C for each AT-base pair, 4 °C for each GC-base pair.
- GC-% formula: 81.5 + 0.41 * GC ratio – 500 / length.
- nearest-neighbor method [6, 29] with parameter sets from Breslauer [6], Santa-Lucia [30] and Sugimoto [37].

All temperatures are corrected due to monoionic salt and formamide concentrations by adding + 16.6*[Salt] – 0.62*[Formamide].

## 5 DNASequenceCompiler

### 5.1 The principle

The correspondence of Chomsky-classes of grammars to certain implementations of self-assembling DNA molecules was noted by Winfree et al. [40]. In particular it was shown that the self-assembly of linear, branched and double-crossover molecules is capable of simulating the word generation process for regular, context-free and universal languages, respectively. The input molecules in such a system represent the rules of a grammar. Like the application of grammar rules to a given start symbol assembles terminal symbols to a word of the language defined by the grammar, hybridization assembles the rule molecules (which inherently also represent the right hand terminal symbols) to bigger molecules. Thus, defining rules of a grammar can be considered as a programming step, predetermining the self-assembly process of DNA molecules. The major problem with the design of rule molecules is to ensure that their hybridization behavior really resembles the grammar's word construction.

The program DNASequenceCompiler has been developed for this purpose [12]. It translates rules of a regular grammar to molecules capable of self-assembling to linear structures. Since the rule of a regular grammar has the form $A \rightarrow bC$, where A and C are variables and b is a terminal symbol, the rule molecules consist of a double stranded core sequence for the terminal symbol and sticky ends for the left and right hand variables. For terminating rules of the form $A \rightarrow b$ or $S \rightarrow bC$, where S is the start symbol, one end of the rule molecule can be blunt (no sequence will be generated for the start symbol S) or contain an additional sticky end for further assembly.

Here again, uniqueness must be secured when concatenating variable and terminal sequences. By concatenation, base strands that overlap both sequences appear in these junctions not regarded in the single sequence generation process. In fact, it may be necessary to tolerate multiple uses of base strands in these junctions (see below).

## 5.2 The program

This tool consists of a graphic user interface, which is primarily a simple text editor for writing the source code files, and the command line based core program, which can be called by the GUI. Such a source code file is the input for the compiler and contains not only the rules, but also parameters like sequence length, melting temperature and so forth. Default values can be defined, so that not all properties must be set for each and every sequence.  Parameters for the estimation of melting temperature are also set in this file. Additionally, sequences that do not appear in the rules but should be dissimilar to the new sequences can be preset. Restriction sites or other sticky ends can be declared that are attached to the start and end terminal sequences, e.g. for cloning of the assembled word molecules.

When being started, the compiler parses this source code file and reports errors it finds in the rules or parameters. It then tries to find sequences that meet the given restrictions. If successful, it writes these sequences into several output files: text files containing the sequences for variables and terminal symbols, a text file showing the double stranded rule molecules, a pool file that can be loaded into the DNASequenceGenerator, and a text file containing messages the compiler wrote to the command shell.

By setting an according parameter in the source code file the user can subsequently start an analysis of the rule molecules. The program then writes base strands that occurred more than once into a text file, along with information where to find these base strands in the sequences.

## 5.3 Differences to basic algorithm

Here, the sequences are not merely a collection of single sequences, but terminal and variable sequences become connected, first when building a rule molecule and then again in the self-assembly of these molecules. At these junctions, base strands overlapping both connected sequences come into play. For obtaining uniqueness, these emerging base strands have to be regarded in the sequence generation process, too. This is achieved by generating all terminal sequences first and then finding variable sequences by letting the according paths start at the last base strand of the 5'-adjacent terminal sequences and ending the path search at the first base strand of the 3'-adjacent terminal sequence.

Furthermore, a variable can be adjacent to several different terminals in different rules, and *vice versa*. Therefore, the paths starting from different terminal sequences are prolonged in parallel and converge to one variable path, and after the variable sequence is complete this path diverges into several terminal paths (*Figures 7* and *8*).

Finally, when such a path should diverge into more than four paths, nodes must be used in several braches because each node has at most four successors available. A parameter controls for how many steps the algorithm tolerates this inevitable violation of uniqueness.

## 6 Quality Comparisons with Other Generators

In order to compare our sequence generation algorithm with other approaches, we analyzed sequences from several publications and generated sequences with comparable restrictions with DNASequenceGenerator. We computed $n_b$ for the published sequences, and minimum, maximum, average and standard deviation for melting temperature and homology. Sequences were taken from the publications of Tanaka et al. [39], Shin et al. [34], Arita et al. [1], Faulhammer et al. [11], and Deaton et al. [10]. Melting temperature was estimated with the parameter set of SantaLucia et al. [30], sample concentration was $2*10^{-7}$ M, salt concentration was 0.05 M, and formamide concentration was 0.0 M. Complete generator parameter sets can be downloaded from our website.

Tanaka et al. applied simulated annealing to optimize a set of 20 14-mers [39]. The evaluation function was a weighted sum of different terms evaluating dissimilarity, GC content, melting temperature, self-complementary sequences, complete hybridization at the 3'-end, and continuous appearance of the same base. Dissimilarity was estimated with H-measure.

*Table 1:* Comparison of the sequences of Tanaka et al. and our competing sequences. Shown are the sequences, the minimum length of the base strands for which the sequences are $n_b$-unique, homology and melting temperature. In the $T_m$ and Homology fields the first line contains the minimum and maximum value, the second line contains the average and the standard deviation.

| | Tananka et al. | DNASequenceGenerator |
|---|---|---|
| Sequences | cgagacatcgtgcatatcgt<br>tatagcacgagtgcgcgtat<br>gatctacgatcatgagagcg<br>tctgtactgctgactcgagt<br>cgagtagtcacacgatgaga<br>agatgatcagcagcgacact<br>tgtgctcgtctctgcatact<br>agacgagtcgtacagtacag<br>atgtacgtgagatgcagcag<br>atcactactcgctcgtcact<br>tcagagatactcacgtcacg<br>gacagagctatcagctactg<br>gctgacatagagtgcgatac<br>acatcgacactactacgcac | aaagccgtcgtttaaggagc<br>tagtcgcgtgatttggaagg<br>tacgtctcgaactgatagcc<br>gctgtctttcgtcaataccg<br>tgatcttgtaaaggccaggc<br>tgcagaaaaactatgccgcc<br>ctgaacggaatctagtagcg<br>tacgatacttggcgagccat<br>gcgcggacaattcattggtt<br>aatcgcagtacagatggtgg<br>gtctacggttctcttacgct<br>cttaggcaggtgccacatat<br>ggatgaccagagcacttcaa<br>ccgcaatccggtgaaattag |
| $n_b$ | 9 | 5 |
| Homology | 0.25 to 0.45<br>0.38 (± 0.05) | 0.25 to 0.40<br>0.36 (± 0.04) |
| $T_m$ | 58.82 to 62.81<br>60.62 (± 1.18) | 59.34 to 61.69<br>60.33 (± 0.73) |

The width of the range of melting temperatures is only slightly better for our sequences, as is homology. The major advantage of our sequences lies in the length of the base strands. On the other hand, the sequences of Tanaka et al. have no continuous run of the same base strand, a criterion we neglected when generating our sequences. We only prohibited runs of more than two consecutive guanine bases.

Shin et al. generated 7 20-mers with an evolutionary algorithm [34], computing fitness with similar evaluation terms as those from Tanaka et al.

*Table 2:* Comparison of the sequences of Shin et al. and our competing sequences. The layout is the same as in *Table 1.*

| | Shin et al. | DNASequenceGenerator |
|---|---|---|
| Sequences | aggcgagtatggggtatatc<br>cctgtcaacattgacgctca<br>ttatgattccactggcgctc<br>atcgtactcatggtccctac | aaagccgtcgtttaaggagc<br>tagtcgcgtgatttggaagg<br>tacgtctcgaactgatagcg<br>cttcgtgtcggccatcatat |

| | | |
|---|---|---|
| | cgctccatccttgatcgttt<br>cttcgctgctgataacctca<br>gagttagatgtcacgtcacg | tcatgttggcaccgtatgca<br>ggttcttacgctctactgca<br>ttacacttgaagctggctcg |
| $n_b$ | 7 | 5 |
| Homology | 0.25 to 0.55<br>0.37 (± 0.06) | 0.30 to 0.40<br>0.37 (± 0.04) |
| $T_m$ | 56.95 to 62.08<br>59.76 (± 2.00) | 60.30 to 61.08<br>60.79 (± 0.24) |

The melting temperature range of our sequences is definitely narrower. Also the uniqueness measures $n_b$ and homology are improved. Compared to the sequences from Tanaka et al. those from Shin et al. have more continuous runs of the same base, in particular there is one 4 nt long guanine subsequence.

Arita et al. generated a set of 9 15-mers and 3 20-mers with a genetic algorithm [1]. Fitness was evaluated considering Hamming distance, base repetition, GC content distribution, false position of restriction site and complete hybridization at the 3'-end.

*Table 3:* Comparison of the sequences of Arita et al. and our competing sequences. The layout is nearly the same as in *Table 1*, melting temperature is regarded for the subsets of 15-mers and 20-mers, respectively.

| | Arita et al. | DNASequenceGenerator |
|---|---|---|
| Sequences | ccgtcttcttctgct<br>ttccctccctctctt<br>cgtcctcctcttgtt<br>ccccttcttgtcctt<br>tgcccctcttgttct<br>ctcctcttccttgct<br>cttctcccttcctct<br>ccttccttccctctt<br>tccccttgtgtgtgt<br>gagagagaggccccctatcc<br>gaagagaagggcacccctcc<br>gtggtgttgcgtcccttccc | aaagccgtcgtttcc<br>ttgtggtactctgcg<br>tattagatggccgcc<br>ctagctcctttgtcg<br>gcattgtagtggctg<br>ggcatatagcgtgac<br>gttattgcgacctcg<br>agtcatggaccaacg<br>gaacggttaccgatc<br>aaagacgtgtgaagtgcgct<br>gacgaaagttcagcagcgaa<br>tgttaaaatcaggctcgcgc |
| $n_b$ | 10 | 5 |
| Homology | 0.00 to 0.73<br>0.38 (± 0.19) | 0.27 to 0.47<br>0.39 (± 0.06) |
| $T_m$ (15-mers) | 49.29 to 52.79<br>50.61 (± 1.12) | 49.29 to 51.76<br>50.63 (± 0.85) |
| $T_m$ (20-mers) | 62.56 to 62.72<br>62.61 (± 0.13) | 62.08 to 62.56<br>62.34 (± 0.24) |

*Table 4:* Comparison of the sequences of Faulhammer et al. and our competing sequences. The layout is the same as in *Table 1*.

| | Faulhammer et al. | DNASequenceGenerator |
|---|---|---|
| Sequences | ctcttactcaattct<br>catatcaacatctta<br>atcctccacttcaca<br>ttaaaatcttccctc<br>ctatttctccacacc<br>gcttcaaacaattcc<br>aactctcaaattcaa<br>ctaacctttacttca<br>cattccttatcccac<br>caccctttctcctct<br>tcctcacattactta<br>acttcctttatatcc<br>ttataacaaacatcc<br>acataaccctcttca<br>accttactttccata<br>gtacattctccctac<br>cataatcttatattc<br>ataatcacatacttc<br>tccaccaactaccta<br>ttttaaatttcacaa | aaagccgtcaaatac<br>tacctttttgtctcg<br>taagtatatcgtgcc<br>agtgacactagcatt<br>aagctattgattggc<br>cttctctcacctata<br>ttacagcgttttacc<br>ggcaagaggaataat<br>tggtaggccatttaa<br>cacttgagtacaaca<br>ggatgtccttgttta<br>gcgaaaattaactcc<br>gtctgagctgataaa<br>acaggcgtatctaat<br>gatccggttactaaa<br>atgaggcagtcttta<br>tgcgactatgttatg<br>acctgactcgtaata<br>accaaaccatgatga<br>gtaccgttgaattgt |
| $n_b$ | 8 | 5 |

| | | |
|---|---|---|
| Homology | 0.13 to 0.67<br>0.39 (± 0.10) | 0.20 to 0.47<br>0.39 (± 0.06) |
| $T_m$ | 33.44 to 40.00<br>42.00 (± 4.12) | 44.01 to 51.76<br>45.25 (± 0.59) |

While the melting temperature ranges are comparable (which is remarkable since the GA did not consider melting temperature but only GC content), our sequences are significantly more dissimilar. This probably lies in the difficulty of finding appropriate weights for the different fitness evaluation terms, which Arita et al. mention in their publication.

Faulhammer et al. generated a library of 20 15-mers with their program PERMUTE that repaired sequences until they met the criteria [11]. Similarity was measured with Hamming distance, additionally hybridization of any two sequences by more than seven consecutive bases was prohibited. Further, melting temperature should have an average of 45 °C.

Our sequences are more dissimilar with respect to both measures. The range of melting temperature is significantly narrower for our sequences. The deviation of the $T_m$ average from 45 °C for the sequences of Faulhammer et al. is probably based on the use of a different estimation method or parameter set.

Deaton et al. used a genetic algorithm to generate 7 20-mers, prohibiting several types of mishybridization through Hamming distance restrictions [10].

*Table 5:* Comparison of the sequences of Deaton et al. and our competing sequences. The layout is the same as in *Table 1*.

| | Deaton et al. | DNASequenceGenerator |
|---|---|---|
| Sequences | cttgtgaccgcttctgggga<br>cattggcggcgcgtaggctt<br>atagagtggatagttctggg<br>gatggtgcttagagaagtgg<br>tgtatctcgttttaacatcc<br>gaaaaaggaccaaaagagag<br>ttgtaagcctactgcgtgac | aaagccgtcgtttaaggacc<br>accattttggaggtggaacg<br>tatatcgtagagccacacgc<br>tccgcgtactgataatcctc<br>atatgcttaggcacggttgg<br>tctcgtgaattggtctggac<br>ttactcatctctgtgacgcc |
| $n_b$ | 9 | 5 |
| Homology | 0.25 to 0.50<br>0.35 (± 0.07) | 0.25 to 0.35<br>0.34 (± 0.03) |
| $T_m$ | 52.67 to 68.91<br>59.21 (± 5.57) | 58.34 to 59.86<br>59.18 (± 0.61) |

Both dissimilarity measures deliver better values for our sequences, in particular the minimum base strand length. The melting temperatures of our sequences are distributed conspicuously closer.

## 7 Compiler example: 32-bit molecules

Using the DNASequenceCompiler, we designed a molecular 32-bit data structure, using four byte-molecules. The byte-molecules consist of eight rule molecules, whose terminal sequences represent a 0 or 1 each, while the variable sequences determine the position of this bit in the byte. Further, a byte-molecule is delimited by an end rule molecule and one out of four different start rule molecules, which serves as the “address” of the byte in the four-byte data structure. Terminal sequences are longer than variable sequences and, consequently, have a higher melting temperature so that the rule molecules are stable while self-assembling. The melting temperatures lie in a 3 °C wide range for each group of sequences, avoiding bias in the self-assembly process. Subsequences with three or more consecutive guanines are prohibited because of the danger of quadruplex formation. Because each terminal sequence is flanked by eight different variable sequences, some uniqueness violation at the junctions has to be allowed. The start and end rules are terminated by restriction sites.

In one of 25 trials with different random number generator seeds the program managed to generate sequences with the given restrictions. On a computer with an AMD Athlon CPU with 900 MHz and 256 MB RAM the compiler needed 7 seconds to find the sequences. The output files can be seen in *Figures 9* and *10*.

## 8 Discussion

### 8.1 n-uniqueness

The concept of uniqueness presented in this paper is designed to avoid mishybridizations, not only between different DNA molecules, but also between different regions of one and the same molecule. By allowing subsequences to appear only once and prohibiting the appearance of their reverse complements we achieved this goal. But some secondary structures must be regarded carefully. E.g., $\texttt{5'-accg}\underline{\texttt{t}}\texttt{aagc-3'}$ and $\texttt{5'-gctt}\underline{\texttt{g}}\texttt{cggt-3'}$ have a single-base mismatch at the underlined base and thus are 5-unique, but they will probably form a duplex that would be only slightly less stable than the perfectly matching ones. This problem can be handled by setting the maximum homology parameter (the H-measure based measure) to a value less than 1, but only at the cost of additional running time, because the pairwise comparison of the sequences is computationally expensive. Single-base bulges in one strand only (e.g. for $\texttt{5'-accg}\underline{\texttt{t}}\texttt{aagc-3'}$ and $\texttt{5'-gcttcggt-3'}$) pose a similar problem which cannot be handled by restricting the H-distance. Furthermore, $\underline{\texttt{gcgc}}\texttt{aaa}\underline{\texttt{gcgc}}$ is 5-unique, but may form a hairpin loop with the underlined parts building the stem.

Obviously, it is important to choose the base strand length as small as possible. In particular, it should be significantly smaller than the sequence length, so that the intended hybridizations occur at a significantly higher temperature than these mishybridizations.

### 8.2 The algorithm

According to equation (1), the size of the graph grows exponentially in the base strand length. Thus, the worst case running time also grows exponentially. The actual running time strongly depends on the quality of the case. Obviously, restrictions leading to the prohibition of certain base strands (e.g. avoidance of start codons) affect the running time less negative than those triggering backtracking, especially at the end of sequences (e.g. melting temperature restrictions). For most of the examples we have examined, the generation of the sequences (or the notification of failing to generate the sequences) only took minutes or even seconds.

With an EA, for example, the maximum running time would be easier to control. However, it is uncertain how the evolved sequences would meet the different criteria after a predetermined time, whereas the algorithm presented in this paper enforces strict compliance with the given restrictions. A particularly hard problem when using an EA would be to optimize multiple criteria at the same time, e.g. dissimilarity and melting temperature.

Other graph-based approaches besides greedy algorithms may be promising and deserve examination.

### 8.3 Comparison to similar approaches

The definition of base strands is similar to that of “critons” in the DNA design program of Seeman et al. [31, 32], named SEQUIN. But our algorithmic approach is different, especially it is fully automatic unlike the semi-automatic, interactive SEQUIN. Furthermore, particularly the DNASequenceCompiler aims at a different field of application.

The concept of overlapping subsequences and the prohibition of them or their reverse complements is also related to modified DeBruijn-sequences [3, 36]. With that approach, one gets a long sequence from which a set of shorter n-unique sequences can be cut out. If one wants the sequences to meet additional restrictions, e.g. have a certain melting temperature, and appropriate subsequences to cut out are not found, a completely new DeBruijn-sequence has to be generated, discarding the good subsequences found so far. Our graph-based algorithm searches one desired sequence at a time, so when searching for one sequence one does not have to discard the ones already found. Backtracking gives more flexibility in the order of the base strands, because it can search new paths as opposed to being restricted to one given long path.

As the comparison with several published sets of sequences have shown, the DNASequenceGenerator is capable of generating comparable sequence libraries. In most cases the sequences generated with our program were better in respect to dissimilarity measures and to melting temperature range. No more than three runs of the generator were executed for each comparison case, often even the first run generated the sequences presented. Most runs took three seconds or less on a computer with an AMD Athlon CPU with 900 MHz and 256 MB RAM, only one run needed 1 minute and 10 seconds. Hence, the computational cost and the effort for the user was quite low.

The compiler example demanded more efforts since 25 runs were needed to generate the desired sequences. But here, the restrictions were very strong. Also the problem the compiler has to resolve is a lot harder than the one posed for the generator because of the additional requirement for concatenations.

## 9 Future work

Considering the problems mentioned in the discussion section, the measure of dissimilarity (the n-uniqueness) should be improved, e.g. including thermodynamic information. More comparisons with published algorithms and sequences will give further insight on the strengths and weaknesses of our approach.

Currently, we are expanding the compiler to generate sequences for molecules capable to build more complex structures, including e.g. junctions, hairpin loops and crossover molecules. Therefore, an input language more powerful than regular grammar rule sets is needed. Simply using rules of more general grammars does not seem appropriate, because one had to choose one fixed form of molecules representing certain rules. In order to be more flexible with respect to the molecule structure, a molecule description language is currently being developed.

## 10 Acknowledgements

We would like to thank Jonathan C. Howard and the members of his group from the Institute for Genetics, University of Cologne, for their friendly support. The work was supported in parts by the Stifterverband der Deutschen Wissenschaft/Stiftung Winterling Marktleuthen.

## 11 Availability

Both programs and the parameter sets for generating the presented sequences can be downloaded from
http://ls11-www.informatik.uni-dortmund.de/molcomp/Downloads/downloads.html

## References

[1] M. Arita, A. Nishikawa, M. Hagiya, K. Komiya, H. Gouzu, K. Sakamoto, "Improving Sequence Design for DNA Computing," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), D. Whitley et al. (eds.), Morgan Kaufmann, pp. 875-882, 2000

[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, Z. Yakhini, "Universal DNA Tag Systems: A Combinatorial Design Scheme," Journal of Computational Biology, vol. 7, pp. 503-519, 2000

[4] A. Brenneman, 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] K. J. Breslauer, R. Frank, H. Blöcker, "Predicting DNA duplex stability from the base sequence", Proceedings of the National Acadamy of Sciences, vol. 83, pp. 3746-3750, 1986

[7] J. Chen, N. C. Seeman, "Synthesis from DNA of a molecule with the connectivity of a cube," Nature, vol. 350, pp. 631-633, 1991

[8] R. Deaton, J. Chen, H. Bi, and J. A. Rose, "A Software Tool for Generating Non-crosshybridizing Libraries of DNA Oligonucleotides", in Preliminary Proceedings of the 8th International Meeting on DNA Based Computers, June 10-13 2002,Hokkaido University, pp. 211-220; to appear in a Springer-Verlag LNCS issue, 2003, in press

[9] R. Deaton, R. C. Murphy, M. Garzon, D. T. Franceschetti, S. E. Stevens Jr., "Good Encodings for DNA-based Solutions to Combinatorial Problems," in Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, pp. 159-171, 1996

[10] R. Deaton, R. C. Murphy, J. A. Rose, M. Garzon, D. T. Franceschetti, S. E. Stevens Jr., "Genetic Search for Reliable Encodings for DNA-based Computation," in First Conference on Genetic Programming, 1996

[11] D. Faulhammer, A. R. Cukras, R. J. Lipton, L. F. Landweber, "Molecular Computation: RNA solutions to chess problems," Proceedings of the National Academy of Sciences, vol. 97, pp. 1385-1389, 2000

[12] U. Feldkamp, W. Banzhaf, H. Rauhe, "A DNA Sequence Compiler," Preliminary Proceedings of the 6th DIMACS Workshop on DNA Based Computers, held at the University of Leiden, The Netherlands: p. 253, 2000. Manuscript available at: http://LS11-www.cs.uni-dortmund.de/molcomp/Publications/publications.html

[13] U. Feldkamp, S. Saghafi, W. Banzhaf, H.Rauhe, "DNASequenceGenerator: A Program for the construction of DNA sequences," DNA Computing, 7th International Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10-13 June 2001, pp. 23-32, 2001

[14] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, R. M. Corn, "Demonstration of a word design strategy for DNA computing on surfaces.," Nucleic Acids Research vol. 25, pp. 4748-4757, 1997

[15] M. Garzon, R. Deaton, P. Neathery, D. R. Franceschetti, R.C. Murphy, “A new metric for DNA computing,” in Conference on Genetic Programming, GP-97, Stanford University, Stanford, California, July 13-16, 1997, J. R. Koza et al. (eds.), 1997

[16] M. Garzon, R. J. Deaton, J. A. Rose, and D. R. Franceschetti, “Soft molecular computing,” in Proceedings 5th DIMACS Workshop on DNA Based Computers, held at the Massachusetts Institute of Technology, Cambridge, MA, USA June 14 - June 15, 1999, American Mathematical Society, pp. 91-100, 1999

[17] N. P. Gerry, N. E. Witowski, J. Day, R. P. Hammer, G. Barany, F. Barany, “Universal DNA Microarray Method for Multiplex Detection of Low Abundance Point Mutations,” Journal of Molecular Biology vol. 292, pp. 251-262, 1999

[18] A. J. Hartemink, D. K. Gifford, J. Khodor, “Automated Constraint-Based Nucleotide Sequence Selection for DNA Computation,” in Proceedings of the 4th DIMACS Workshop on DNA Based Computers, held at the University of Pennsylvania, Philadelphia, pp. 227-235, 1998

[19] F. Li, G. D. Stormo, “Selection of optimal DNA oligos for gene expression arrays,” Bioinformatics vol. 17, pp. 1067-1076, 2001

[20] C. Mao, T. H. LaBean, J. H. Reif, N. C. Seeman, “Logical computation using algorithmic self-assembly of DNA triple-crossover molecules,” Nature vol. **407**, pp. 493-496, 2000

[21] A. Marathe, A. E. Condon, R. M. Corn, “On Combinatorial DNA Word Design,” in Proceedings of the 5th International Meeting on DNA Based Computers, 1999

[22] C. A. Mirkin, R. L. Letsinger, R. C. Mucic, J. J. Storhoff, “A DNA-based method for rationally assembling nanoparticles into macroscopic materials,” Nature vol. 382, pp. 607-609, 1996

[23] C. M. Niemeyer, “Self-assembled nanostructures based on DNA: towards the development of nanobiotechnology,” Current Opinion in Chemical Biology vol. 4, pp. 609-618, 2000

[24] G. Raddatz, M. Dehio, T. F. Meyer, C. Dehio, “PrimeArray: genome-scale primer design for DNA-microarry construction,” Bioinformatics vol. 17, pp. 98-99, 2001

[25] J. A. Rose, R. J. Deaton, “The Fidelity of Annealing-Ligation: A Theoretical Analysis”, in DNA Computing, 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 2000, Springer, pp. 231-246, 2001

[26] J. A. Rose, R. J. Deaton, D. R. Franceschetti, M. Garzon, S. E. Stevens Jr., “A Statistical Mechanical Treatment of Error in the Annealing Biostep of DNA Computation,” Proceedings of the Genetic and Evolutionary Computation Conference 1999, Morgan Kaufmann, pp. 1829-1834, 1999

[27] J. A. Rose, R. J. Deaton, M. Hagiya, A. Suyama, “The Fidelity of the Tag-Antitag System,” in DNA Computing, 7th International Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10-13 June 2001, Springer, pp. 138-149, 2002

[28] A. J. Ruben, S. J. Freeland, L. F. Landweber, “PUNCH: An Evolutionary Algorithm for Optimizing Bit Set Selection,” in DNA Computing, 7th International Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10-13 June 2001, pp. 150-160, 2001

[29] W. Rychlik, R. E. Rhoads, “A computer program for choosing optimal oligonucleotides for filter hybridization, sequencing and in vitro amplification of DNA,” Nucleic Acids Research vol. 17, pp. 201-209, 1989

[30] J SantaLucia, Jr., H. T. Allawi, P. Ananda Seneviratne, “Improved Nearest-Neighbor Parameters for Predicting DNA Duplex Stability,” Biochemistry vol. 35, pp. 3555-3562, 1996

[31] N. C. Seeman, N. R. Kallenbach, “Design of immobile Nucleic Acid Junctions”, Biophysical Journal vol. 44, pp. 201-209, 1983

[32] N. C. Seeman, “De Novo Design of Sequences for Nucleic Acid Structural Engineering,” Journal of Biomolecular Structure & Dynamics vol. 8, pp. 573-581, 1990

[33] N. C. Seeman, “DNA engineering and its application to nanotechnology,” Trends in Biotechnology vol. 17, pp. 437-443, 1999

[34] S.-Y. Shin, D.-M. Kim, I.-H. Lee, B.-T. Zhang, “Evolutionary Sequence Generation for Reliable DNA Computing,” in Proceedings of the 2002 Congress on Evolutionary Computing CEC'02, pp. 79-84, 2002

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

[36] W. D. Smith, “DNA computers in vitro and in vivo,” in Proceedings of a DIMACS Workshop, held at Princeton University, 4 April 1995, Amer. Math. Soc., pp. 121-186, 1996

[37] N. Sugimoto, S. Nakano, M. Yoneyama, K. Honda, “Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes,” Nucleic Acids Research vol. 24, pp. 4501-4505, 1996

[38] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, A. Ohuchi, “Developing Support System for Sequence Design in DNA Computing,” DNA Computing, 7th International Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10-13 June 2001, pp. 129-137, 2001

[39] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, A. Ohuchi, “Towards a General-Purpose Sequence Design System in DNA Computing,” in Proceedings of the 2002 Congress on Evolutionary Computing CEC'02, pp. 73-78, 2002

[40] E. Winfree, X. Yang, N. C. Seeman, “Universal Computation via Self-assembly of DNA: Some Theory and Experiments,” Proceedings of the 2nd DIMACS Meeting on DNA Based Computers, Princeton University, June 10-10, (1996)

[41] E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman, “Design and self-assembly of two-dimensional DNA crystals,” Nature vol. 394, pp. 539-544, 1998

[42] H. Yan, X. Zhang, Z. Shen, N. C. Seeman, “A robust DNA mechanical device controlled by hybridization topology”, Nature vol. 415, pp. 62-65, 2002

[43] B. Yurke, A. J. Turberfield, A. P. Mills, Jr., F. C. Simmel, J. L. Neumann, J. L., “A DNA-fuelled molecular machine made of DNA,” Nature vol. 406, pp. 605-608, 2000

```
acgcgctca     complete sequence
acgcgc    ⎫
 cgcgct   ⎬   base strands
  gcgctc  ⎪
   cgctca ⎭
```

*Figure 1:* A sequence of length $n_s = 9$ consisting of $(n_s – n_b + 1) = 4$ overlapping base strands of length $n_b = 6$.

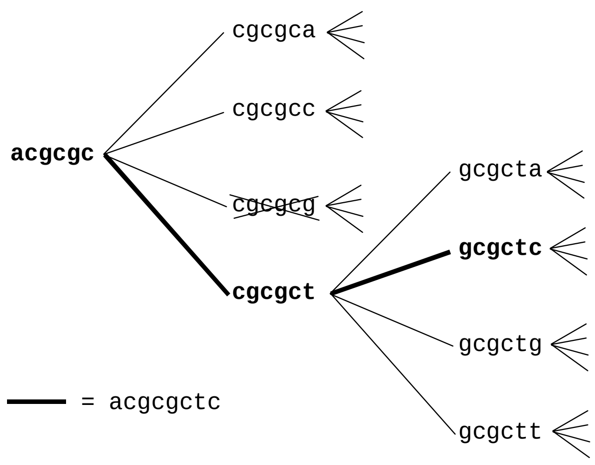


*Figure 2:* Graph of base strands. A sequence of length $n_s$ is represented by a path of $n_s - n_b + 1$ nodes. The node `cgcgcg` is self-complementary and therefore not used.

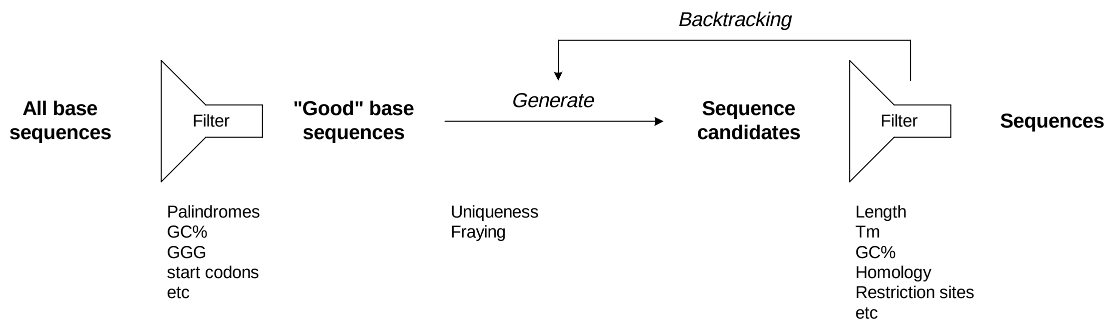
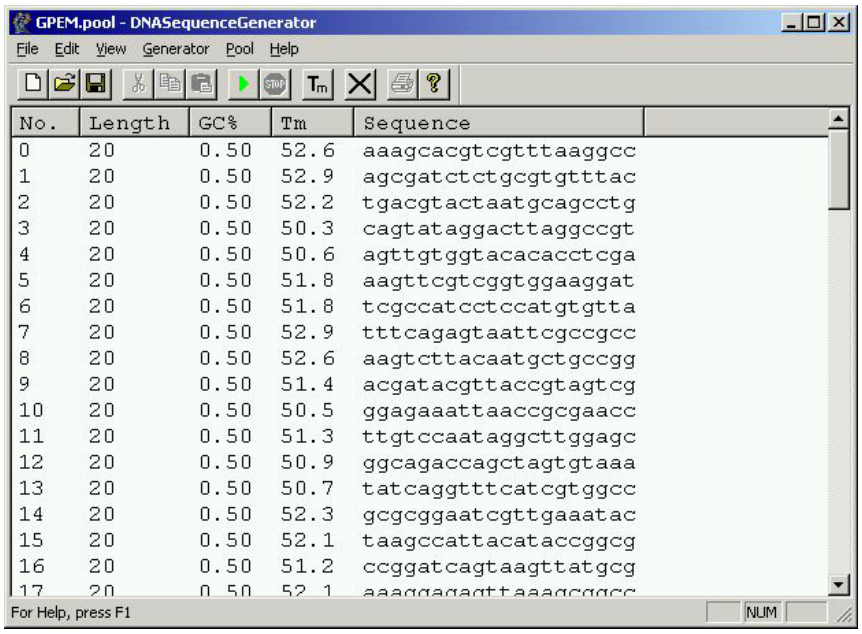
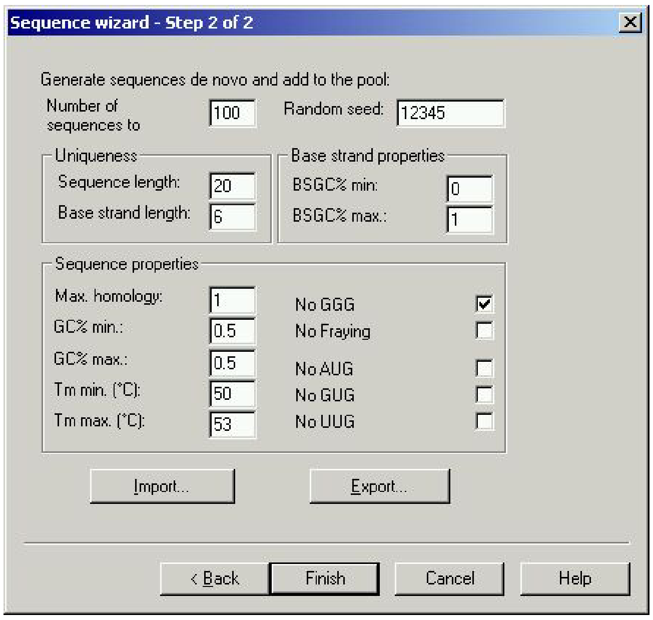


*Figure 3:* The basic sequence generation algorithm. While uniqueness (and the avoidance of fraying) is enforced inherently by the path searching process, the other requirements are met by filtering base strands and complete sequences.



| No. | Length | GC% | Tm | Sequence |
|---|---|---|---|---|
| 0 | 20 | 0.50 | 52.6 | aaagcacgtcgtttaaggcc |
| 1 | 20 | 0.50 | 52.9 | agcgatctctgcgtgtttac |
| 2 | 20 | 0.50 | 52.2 | tgacgtactaatgcagcctg |
| 3 | 20 | 0.50 | 50.3 | cagtataggacttaggccgt |
| 4 | 20 | 0.50 | 50.6 | agttgtggtacacacctcga |
| 5 | 20 | 0.50 | 51.8 | aagttcgtcggtggaaggat |
| 6 | 20 | 0.50 | 51.8 | tcgccatcctccatgtgtta |
| 7 | 20 | 0.50 | 52.9 | tttcagagtaattcgccgcc |
| 8 | 20 | 0.50 | 52.6 | aagtcttacaatgctgccgg |
| 9 | 20 | 0.50 | 51.4 | acgatacgttaccgtagtcg |
| 10 | 20 | 0.50 | 50.5 | ggagaaattaaccgcgaacc |
| 11 | 20 | 0.50 | 51.3 | ttgtccaataggcttggagc |
| 12 | 20 | 0.50 | 50.9 | ggcagaccagctagtgtaaa |
| 13 | 20 | 0.50 | 50.7 | tatcaggtttcatcgtggcc |
| 14 | 20 | 0.50 | 52.3 | gcgcggaatcgttgaaatac |
| 15 | 20 | 0.50 | 52.1 | taagccattacataccggcg |
| 16 | 20 | 0.50 | 51.2 | ccggatcagtaagttatgcg |
| 17 | 20 | 0.50 | 52.1 | aaaggagagttaaagcggcc |

*Figure 4:* Screenshot of the generator's main window, containing a pool of sequences. The melting temperatures were estimated with the nearest-neighbor method, using the parameter set of Sugimoto [37], with a sample concentration of $2*10^{-7}$ M, a salt concentration of 0.05 M and a formamide concentration of 0.0 M.


Sequence wizard - Step 2 of 2

Generate sequences de novo and add to the pool:

Number of sequences to: 100 Random seed: 12345

Uniqueness

Sequence length: 20

Base strand length: 6

Base strand properties

BSGC% min: 0

BSGC% max.: 1

Sequence properties

Max. homology: 1

GC% min.: 0.5

GC% max.: 0.5

Tm min. (°C): 50

Tm max. (°C): 53

No GGG ✓

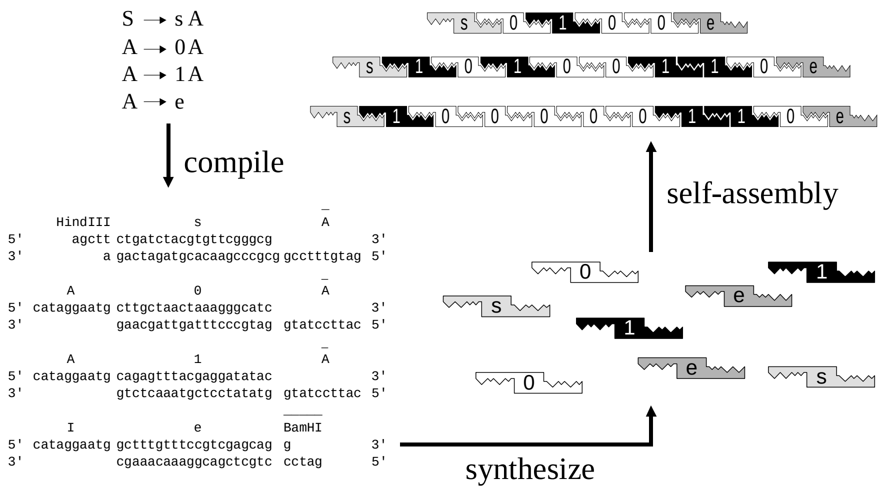
No Fraying ☐

No AUG ☐

No GUG ☐

No UUG ☐

*Figure 5:* Screenshot of the sequence wizard. In this window the user can set the parameters for the sequences to be generated. Selection of NoGGG prohibits the use of more than two consecutive guanine or cytosine bases. Activation of NoFraying forces the first and the last base of each sequence to be guanine or cytosine. The options NoAUG, NoGUG and NoUUG prohibit the use of base strands with the according start codons as subsequences. The U stands for uracil, a base found in RNA. The program substitutes the uracil bases with thymine bases. The shown parameters were used to generate the sequences in Figure 4.



```
S → s A
A → 0 A
A → 1 A
A → e
```

```
                                        _
      HindIII         s                 A
5'      agctt ctgatctacgtgttcgggcg            3'
3'          a gactagatgcacaagcccgcg gcctttgtag  5'

                                        _
        A             0                 A
5'  cataggaatg cttgctaactaaagggcatc            3'
3'             gaacgattgatttcccgtag  gtatccttac  5'

                                        _
        A             1                 A
5'  cataggaatg cagagtttacgaggatatac            3'
3'             gtctcaaatgctcctatatg  gtatccttac  5'

                                   _____
        I             e            BamHI
5'  cataggaatg gctttgtttccgtcgagcag  g          3'
3'             cgaaacaaaggcagctcgtc  cctag      5'
```

*Figure 6:* Programmed self-assembly. The rules of a grammar are translated into molecules that self-assemble in vitro as predetermined by the rules. The grammar shown here defines a simple random bitstring generator.

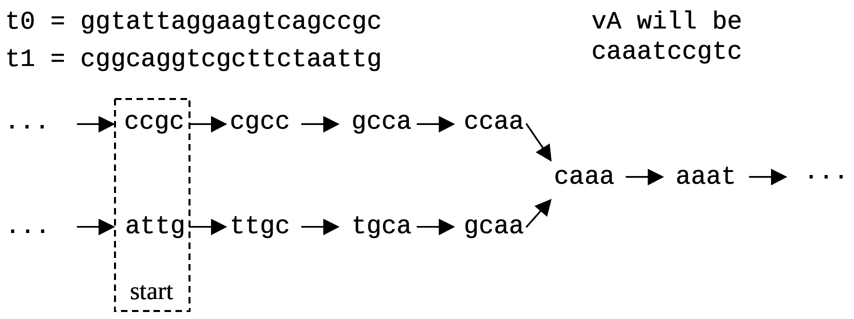


*Figure 7:* Two terminal paths joining to one variable path, for $n_b = 4$. The last base strands of the terminal sequences serve as start nodes. Then both paths are generated in parallel until they join at the first node of the variable sequence (`caaa`). Then the variable path is searched normally.

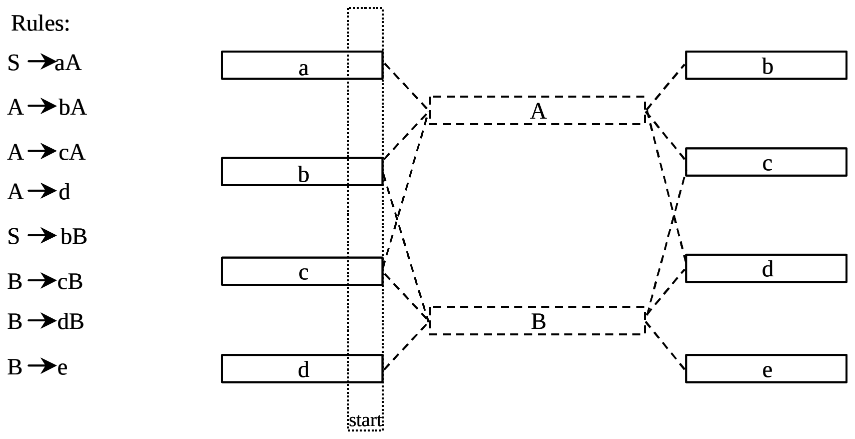


*Figure 8:* Terminal sequences can be adjacent to several variable sequences, and *vice versa*. Therefore, all variable sequences are generated in parallel.

Terminals.txt

```
Start terminator sequences
ts0            tactgcacacagttcggaga          20            0.500000        60.189817
ts1            ggaggtaacaagccgtcata          20            0.500000        59.107435
ts2            acgaacgtatcaagtgtccg          20            0.500000        59.611946
ts3            cgattcggcgctctataaac          20            0.500000        61.607196
Elongator sequences
t0             ggtattaggaaatcagccgc          20            0.500000        59.241532
t1             cggcaggtcgcttttaattc          20            0.500000        61.203583
End terminator sequences
te0            agatgttgtaggctcatcgc          20            0.500000        60.220574
```

Variables.txt

```
vA             caaatccgtc     10             0.500000        29.602949
vB             tcctccttgt     10             0.500000        29.042898
vC             atggactcca     10             0.500000        28.501029
vD             gctgctagta     10             0.500000        29.370492
vE             attgaggcac     10             0.500000        28.268384
vF             tggcactact     10             0.500000        29.507577
vG             gcagacctaa     10             0.500000        29.300213
vH             cagtgatcct     10             0.500000        28.131647
vI             tgaatggtcg     10             0.500000        29.953786
```

*Figure 9:* Output files with terminal and variable sequences. Each line contains the terminal/variable symbol, the sequence, its length, GC ratio and melting temperature in °C.

```
vS -> ts0 vA
agctttactgcacacagttcggaga
    aatgacgtgtgtcaagcctctgtttaggcag

vS -> ts1 vA
agcttggaggtaacaagccgtcata
    acctccattgttcggcagtatgtttaggcag

vS -> ts2 vA
agcttacgaacgtatcaagtgtccg
    atgcttgcatagttcacaggcgtttaggcag

vS -> ts3 vA
agcttcgattcggcgctctataaac
    agctaagccgcgagatatttggtttaggcag

vA -> t0 vB
caaatccgtcggtattaggaaatcagccgc
          ccataatcctttagtcggcgaggaggaaca

vA -> t1 vB
caaatccgtccggcaggtcgcttttaattc
          gccgtccagcgaaaattaagaggaggaaca

vB -> t0 vC
tcctccttgtggtattaggaaatcagccgc
          ccataatcctttagtcggcgtacctgaggt

vB -> t1 vC
tcctccttgtcggcaggtcgcttttaattc
          gccgtccagcgaaaattaagtacctgaggt

vC -> t0 vD
atggactccaggtattaggaaatcagccgc
          ccataatcctttagtcggcgcgacgatcat

vC -> t1 vD
atggactccacggcaggtcgcttttaattc
          gccgtccagcgaaaattaagcgacgatcat
```

```
vD -> t0 vE
gctgctagtaggtattaggaaatcagccgc
          ccataatcctttagtcggcgtaactccgtg

vD -> t1 vE
gctgctagtacggcaggtcgcttttaattc
          gccgtccagcgaaaattaagtaactccgtg

vE -> t0 vF
attgaggcacggtattaggaaatcagccgc
          ccataatcctttagtcggcgaccgtgatga

vE -> t1 vF
attgaggcaccggcaggtcgcttttaattc
          gccgtccagcgaaaattaagaccgtgatga

vF -> t0 vG
tggcactactggtattaggaaatcagccgc
          ccataatcctttagtcggcgcgtctggatt

vF -> t1 vG
tggcactactcggcaggtcgcttttaattc
          gccgtccagcgaaaattaagcgtctggatt

vG -> t0 vH
gcagacctaaggtattaggaaatcagccgc
          ccataatcctttagtcggcggtcactagga

vG -> t1 vH
gcagacctaacggcaggtcgcttttaattc
          gccgtccagcgaaaattaaggtcactagga

vH -> t0 vI
cagtgatcctggtattaggaaatcagccgc
          ccataatcctttagtcggcgacttaccagc

vH -> t1 vI
cagtgatcctcggcaggtcgcttttaattc
          gccgtccagcgaaaattaagacttaccagc

vI -> te0
tgaatggtcgagatgttgtaggctcatcgcg
          tctacaacatccgagtagcgcctag
```

*Figure 10:* Generated rule molecules, consisting of the sequences from *Figure 9*. Note that there is no sequence for the start variable vS. Attached to the start and end rule molecules are restriction sites for further processing steps, e.g. cloning.