DNA Word Design: A New Constraint Model and New Results

Michael Codish[1], Michael Frank[1], and Vitaly Lagoon[2]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Israel
[2] Cadence Design Systems, USA

Abstract. We describe a new constraint model for problem `CSPLib 033: Word Design for DNA Computing on Surfaces`. Using this model, we can compute a solution consisting of 119 words in 6 hours of CPU time. The best previously known result for this problem consists in 112 words (found in 2003). We can compute a solution with 112 words using our model in 25 seconds of CPU time.

1 Introduction

The design of DNA code words has its motivation in DNA computing on surfaces [1] and tasks related to storing information in DNA strands [2]. The problem is related to that of the more general design of error-correcting codes [3]

For the types of combinatorial constraints typically desired, there are no known efficient algorithms. Techniques from coding theory have been applied [2, 1], stocastic local search is applied [4], genetic algorithms are considered [5]. Constraint and SAT programming [6, 7] have also been applied.

`CSPLib 033: Word Design for DNA Computing on Surfaces` is about finding the largest set of eight letter words over the alphabet $\Sigma = \{A, C, G, T\}$ which satisfies a small set of constraints specified in Section 2. We mention the following results reported in the literature:

1. In [8] (2008) the authors present a solution consisting of 87 DNA words, reported to be found in 554 seconds of CPU time using an OPL [9] model.
2. In [1] the authors report a solution with 108 DNA words. They search for solutions that have a specific form which they call template—map.
3. In [4] the authors present an algorithm based on stochastic local search. Initializing their algorithm with the 108 word set found by Frutos *et al.*they construct a set with 112 words in "less than one day of CPU time".
4. Marc van Dongen also reports a solution with 112 words [6].
5. In [7], the authors apply the BEE constraint solver and report finding a solution of 112 words "in a fraction of a second". They also show that this is the optimal code size for the DNA word design problem when using the template—map strategy of Frutos *et al.* [1].

So, the state-of-the-art for this problem is, prior to this paper, a code with 112 words. In this paper we describe how we modeled the search for a solution of a particular form and found a solution with 119 words.

2 CSPLib 033: the Specification

Problem 033 of CSPLib seeks the largest parameter n, such that there exists a set S of n eight-letter words over the alphabet $\Sigma = \{A, C, G, T\}$ which satisfies the following constraints:

The percentage constraint (P): Each word in S has exactly 4 symbols (50%) from $\{C, G\}$.

The Hamming distance constraint (HD): Each pair of distinct words in S differ in at least 4 positions.

The reverse complement Hamming distance constraint (RC): For every $x, y \in S$ (not necessarily distinct): x^R (the reverse of x) and y^C (the word obtained by replacing each A by T, each C by G, and vice versa) differ in at least 4 positions.

We sometimes express this constraint in terms of two separate cases. ($\mathtt{RC_1}$): For every $x \in S$, x^R and x^C differ in at least 4 positions; and ($\mathtt{RC_2}$): Each pair of distinct words $xy \in S$, x^R and y^C differ in at least 4 positions.

3 Our Model: Base Words and their Generated Images

Let $\mathcal{W}$ be the set of all 8 letter words over the alphabet $\Sigma = \{\mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{A}\}$. A word transformation is a mapping $f : \mathcal{W} \to \mathcal{W}$. We are interested in (small) sets of word transformations which are self-inversive and which commute with each other with respect to composition. A word transformation, f, is self-inversive if $f \circ f = id$. A pair of word transformations, f, g is commutative with respect to composition if $f \circ g = g \circ f$. The composition of word transformations is always associative — a property inherited from the composition of relations. So for word transformations, f, g, h we always have $(f \circ g) \circ h = f \circ (g \circ h)$.

We say that a set of word transformations, $\mathcal{T}$, is SCA if for all $f, g, h \in \mathcal{T}$, f is self-inversive, f, g are commutative, and f, g, h are associative.

Let $B \subseteq \mathcal{W}$ and $\mathcal{T}$ be a set of word transformations. We denote by $\mathcal{T}(B) \subseteq \mathcal{W}$ the additive closure of B under composition of transformations from $\mathcal{T}$. These are all of the words that can be generated from B by repetative application of transformations from $\mathcal{T}$. Note that if $\mathcal{T}$ is SCA then $\mathcal{T}(B)$ consists of (at most) $|B| \times 2^{|\mathcal{T}|}$ words. For simplicity, for $w \in \mathcal{W}$, we write $\mathcal{T}(w)$ instead of $\mathcal{T}(\{w\})$.

Example 1. Consider the set $\mathcal{T}_3 = \{wc, fc, hs\}$ consisting of three word transformations defined by:

1. wc: (Watson-Crick complement) applies to a word swapping letters $A \leftrightarrow T$, $C \leftrightarrow G$ (implemented by negating the least significant bits).
2. fc: (full complement) applies to a word swapping letters $A \leftrightarrow G$, $C \leftrightarrow T$ (implemented by negating all bits).

3. *hs*: (half swap) applies to a word by spliting it into two equal size halves and swapping their order.

One can check that $\mathcal{T}_3$ is SCA.

Example 2. Consider the word $w =$ C T A C G A A C and the set of word transformations $\mathcal{T}_3$ introduced in Example 4. The set of eight words generated from w under $\mathcal{T}_3$ is detailed below (to the right of each word the transformation under which is obtained from the base word):

C T A C G A A C	w	A G G T T C G T	$hs(fc(W))$
G A A C C T A C	$hs(w)$	C T T G G A T G	$wc(hs(fc(W)))$
T C G T A G G T	$fc(w)$	A G C A T C C A	$wc(fc(W))$
G A T G C T T G	$wc(w)$	T C C A A G C A	$hs(wc(fc(W)))$

In this paper we represent elements of $\mathcal{W}$ as length 16 bit vectors. We adopt the convention that each letter of $\Sigma = \{A, C, G, T\}$ is represented by two bits:

$C = 00 \quad G = 01 \quad$ (both have msb=0)
$A = 10 \quad T = 11 \quad$ (both have msb=1)

Example 3. The eight words detailed in Example 2 are represented respectively by the following eight bit vectors:

00 11 10 00 01 10 10 00	10 01 01 11 11 00 01 11
01 10 10 00 00 11 10 00	00 11 11 01 01 10 11 01
11 00 01 11 10 01 01 11	10 01 00 10 11 00 00 10
01 10 11 01 00 11 11 01	11 00 00 10 10 01 00 10

A nice property of the set $\mathcal{T}_3$ of word transformations specified as Example 4 is that for a given word $w \in \mathcal{W}$ in bit vector representation, the set $\mathcal{T}_3(w)$ can be specified by a corresponding set of literals.

Example 4. Consider the bit vector

$$w = \langle x_1x_2 \;\; x_3x_4 \;\; x_5x_6 \;\; x_7x_8 \;\; x_9x_{10} \;\; x_{11}x_{12} \;\; x_{13}x_{14} \;\; x_{15}x_{16} \rangle$$

representing an eight letter word of $\mathcal{W}$, the following are the eight words of $\mathcal{T}_3(w)$ it generates, represented as bit vectors.

x_1x_2	x_3x_4	x_5x_6	x_7x_8	x_9x_{10}	$x_{11}x_{12}$	$x_{13}x_{14}$	$x_{15}x_{16}$
x_9x_{10}	$x_{11}x_{12}$	$x_{13}x_{14}$	$x_{15}x_{16}$	x_1x_2	x_3x_4	x_5x_6	x_7x_8
$\neg x_1\neg x_2$	$\neg x_3\neg x_4$	$\neg x_5\neg x_6$	$\neg x_7\neg x_8$	$\neg x_9\neg x_{10}$	$\neg x_{11}\neg x_{12}$	$\neg x_{13}\neg x_{14}$	$\neg x_{15}\neg x_{16}$
$\neg x_9\neg x_{10}$	$\neg x_{11}\neg x_{12}$	$\neg x_{13}\neg x_{14}$	$\neg x_{15}\neg x_{16}$	$\neg x_1\neg x_2$	$\neg x_3\neg x_4$	$\neg x_5\neg x_6$	$\neg x_7\neg x_8$
$x_1\neg x_2$	$x_3\neg x_4$	$x_5\neg x_6$	$x_7\neg x_8$	$x_9\neg x_{10}$	$x_{11}\neg x_{12}$	$x_{13}\neg x_{14}$	$x_{15}\neg x_{16}$
$x_9\neg x_{10}$	$x_{11}\neg x_{12}$	$x_{13}\neg x_{14}$	$x_{15}\neg x_{16}$	$x_1\neg x_2$	$x_3\neg x_4$	$x_5\neg x_6$	$x_7\neg x_8$
$\neg x_1x_2$	$\neg x_3x_4$	$\neg x_5x_6$	$\neg x_7x_8$	$\neg x_9x_{10}$	$\neg x_{11}x_{12}$	$\neg x_{13}x_{14}$	$\neg x_{15}x_{16}$
$\neg x_9x_{10}$	$\neg x_{11}x_{12}$	$\neg x_{13}x_{14}$	$\neg x_{15}x_{16}$	$\neg x_1x_2$	$\neg x_3x_4$	$\neg x_5x_6$	$\neg x_7x_8$

The interested reader can double check that for $w =$ C T A C G A A C, represented by 00 11 10 00 01 10 10 00 these eight bit vectors correspond precisely to those given as Examples 2 and 3.

We say that a word $w \in \mathcal{W}$ is a *base word* for the DNA word design problem with respect to $\mathcal{T}_3$ if the set, $\mathcal{T}_3(w)$, consists of eight words that satisfy the three constraints: (P), (HD), and (RC), and w is the minimal element of $\mathcal{T}_3(w)$ (under the lexicographic order on the bit vector representation of the words).

Example 5. The word $w =$ C T A C G A A C is a base word with respect to $\mathcal{T}_3$. The eight words of $\mathcal{T}_3(w)$ detailed in Example 2 satisfy the constraints (P), (HD), and (RC), and the bit vector representation of w is minimal (in the lexicographic order) on the bit vector representations detailed in Example 3.

Our strategy to solve the DNA word design problem is to seek a solution S constructed in terms of a (smaller) set B of base words such that $S = \mathcal{T}_3(B)$. In fact, B will be eight times smaller than S. However, it is not just about the size of the set B. Expressing the constraints of the DNA word design problem in terms of the set B turns out to be much more compact as well.

Let $x_1 \dots x_{16}$ be a bit vector representing a word in $w \in \mathcal{W}$ and let B be the set of eight bit vectors detailed in Example 3 corresponding to the elements of $\mathcal{T}_3(w)$.

A model that constrains w to be a base word must constrain w to be the lexicographic minimum in $\mathcal{T}_3(w)$. That means imposing 7 constraints.

A model that constrains w to be a base word must impose constraints (P) and ($\mathtt{RC}_1$) on each word of $\mathcal{T}_3(w)$. The following lemma implies that it suffices to impose them only on w. That means checking one constraint instead of eight.

Lemma 1. *Let $w \in \mathcal{W}$. If w satisfies Constraint (P), then so does $\mathcal{T}_3(w)$. If w satisfies Constraint ($\mathtt{RC}_1$), then so does $\mathcal{T}_3(w)$.*

A model that constrains w to be a base word must impose constraints (HD) and ($\mathtt{RC}_2$) on each pair of distinct words from $\mathcal{T}_3(w)$ i.e., two constraints for each pair of distinct words from $\mathcal{T}_3(w)$ or 56 constraints in total. The following lemma implies that it suffices to consider only the pairs including the base word w i.e., use 14 constraints only.

Lemma 2. *Let $w \in \mathcal{W}$. If all pairs (u, w), where $u \in \mathcal{T}_3(w)$ and $u \neq w$ satisfy Constraints (HD) and ($\mathtt{RC}_2$), then $\mathcal{T}_3(w)$ satisfies Constraints (HD) and ($\mathtt{RC}_2$).*

Now consider a set B of base words. A model that constrains $S = \mathcal{T}_3(B)$ to be a solution of the DNA word design problem must impose constraints (HD) and ($\mathtt{RC}_2$) on pairs of words: for each $w_1, w_2 \in B$ (viewing B as a sequence we can assume that w_1 occurs before w_2) all pairs u, v where $u \in \mathcal{T}_3(w_1)$ and $v \in \mathcal{T}_3(w_2)$. The following lemma implies that it suffices, for the pair (w_1, w_2) to impose the constraints for pairs of the form (w_1, v) with $v \in \mathcal{T}_3(w_2)$ and ignoring the symmetric ones of the form (u, w_2). That means checking 16 constraints, two for each pair (u, w_2), instead of $8 \times 8 = 64$ constraints.

Lemma 3. *Let $w_1, w_2 \in \mathcal{W}$. If all pairs (w_1, v) with $v \in \mathcal{T}_3(w_2)$ satisfy Constraints (HD) and ($\mathtt{RC}_2$), then all pairs u, v in $\mathcal{T}_3(w_1) \times \mathcal{T}_3(w_2)$ satisfy Constraints (HD) and ($\mathtt{RC}_2$).*

4 The Constraint Model: DNAbase(n)

We specify the constraint model DNAbase(n) which states that B is a set of n base words such that $S = \mathcal{T}_3(B)$ is a solution of the DNA word design problem.

First some notation. Let M be an $n \times 16$ matrix of Boolean variables to represent the set B. The i^{th} row of M is denoted M_i. We denote by $\prec$ the lexicographic order on Boolean vectors (for example on the rows of M). For each bit vector w in M let $\mathcal{T}_3(w)$ denote the corresponding 8×16 vector of literals detailed as Example 4. Note that the first element of $\mathcal{T}_3(w)$ is w itself. We denote the other seven elements as $\mathcal{T}_3'(w)$ Let w be a bit vector representing a word in $\mathcal{W}$. We write $\mathtt{P(w)}$ and $\mathtt{RC_1(w)}$ to specify that w satsifies the constraints (P) and ($\mathtt{RC_1}$). Similarly, for a pair u, v of bit vectors representing words in $\mathcal{W}$, we write HD(u,v) and $\mathtt{RC_2(u,v)}$.

The constraints of DNAbase(n) are:

1. Constraints to ensure that base words are minimal in the sets they generate under $\mathcal{T}_3$, and we can assume without loss of generality that the base words are sorted:

$$\bigwedge_{i=1}^{n} \bigwedge_{u \in \mathcal{T}_3'(\mathtt{M_i})} \mathtt{M_i} \prec u \qquad \wedge \qquad \bigwedge_{i=1}^{n-1} \mathtt{M_i} \prec \mathtt{M_{i+1}}$$

2. Constraints to ensure that words in M are base-words

$$\bigwedge_{i=1}^{n} \left(\mathtt{P(M_i)} \wedge \mathtt{RC_1(M_i)} \qquad \wedge \qquad \bigwedge_{u \in \mathcal{T}_3'(\mathtt{M_i})} (\mathtt{HD(M_i,u)} \wedge \mathtt{RC_2(M_i,u)}) \right)$$

3. Constraints on words generated from different base-words.

$$\bigwedge_{1 \le i < j \le n} \quad \bigwedge_{u \in \mathcal{T}_3(\mathtt{M_j})} (\mathtt{HD(M_i,u)} \wedge \mathtt{RC_2(M_i,u)})$$

5 Experimenting with the Model DNAbase(n)

The computations reported in this paper are performed using the finite-domain constraint compiler BEE [10] which compiles constraints to CNF, and solves it applying an underlying SAT solver. In the configuration for this paper we use CryptoMiniSAT v2.5.1 as the underlying SAT solver. All computations were performed on a cluster of Intel E8400 cores, each clocked at 2 GHz. Each of the cores in the cluster has computational power comparable to a core on a standard desktop computer. Each SAT instance is run on a single thread.

Table 1 details the results of an experiment to seek solutions to the DNA word design problem which correspond to solutions of DNAbase(n) and are of the form $S = \mathcal{T}_3(B)$ where $|B| = n$, so $|S| = 8 \times n$. The columns in the table detail: n, the size of the set B (the solution size is 8 times this number); BEE,

n	BEE	clauses	variables	sat
1	0.02	1920	447	0.01
2	0.06	6056	1399	0.03
3	0.13	12338	2842	0.07
4	0.20	20766	4776	0.16
5	0.32	31340	7201	0.23
6	0.47	44060	10117	0.36
7	0.65	58926	13524	0.57
8	0.83	75938	17422	0.78
9	0.68	95096	21811	0.99
10	1.19	116400	26691	1.05
11	1.62	139850	32062	14.48
12	1.65	165446	37924	3.66
13	2.12	193188	44277	139.55
14	2.08	223076	51121	21.67
15	1.85	255110	58456	t.o.

Table 1. Computing solutions of model `DNAbase(n)` (time in seconds).

the compilation time (in seconds) to obtain a CNF representation; the number of clauses and variables in the CNF, and sat, the time (in seconds) to obtain a solution using a SAT solver. A solution of 112 words, expressed in terms of a set of 14 base words is found in about 25 seconds. This is not news as a solution of this size was found already in 2003. A computational attempt to apply BEE to determine the existence of a solution when $n = 15$ did not succeed (and we gave up on that computation after 1 week of CPU time). In the next section we prove that no such solution exists.

6 $n = 14$ is Optimal for Model `DNAbase(n)`

To prove the optimality of $n = 14$ for model `DNAbase(n)` with respect to the choice of transformations $\mathcal{T}_3$ we reduce the problem of finding a set of consistent base words to that of finding a maximum clique in a simple undirected graph. In this graph, the size of the maximal clique is equal to the maximal size of consistent base words with respect to $\mathcal{T}_3$.

A simple computation reveals that there are 1424 possible base words for $\mathcal{T}_3$. We define a graph $G_{\mathcal{T}_3} = (V, E)$ where $V = \{w_1, w_2, \ldots, w_{1424}\}$ are these base words. For each pair of vertices $w_i, w_j \in V$, $\{w_i, w_j\} \in E$ if and only if $\mathcal{T}_3(w_i)$ is consistent with $\mathcal{T}_3(w_j)$. In this context consistent means that the set $S_{i,j} = \mathcal{T}_3(w_i) \cup \mathcal{T}_3(w_j)$ is a solution of the DNA word design problem (for which the constraints `(P)`, `(HD)` and `(RC)` hold). Given that each set $S_{i,j}$ consists of only 16 elements, consistency is computationally straightforward to check.

Now, by construction, if B is a set of nodes that is a clique in $G_{\mathcal{T}_3}$, then $\mathcal{T}_3(B)$ is a solution to the DNA word design problem. Specifically, the maximum clique determines the largest number of consistent basewords defined in terms of $\mathcal{T}_3$.

To determine a maximal clique of $G_{\mathcal{T}_3}$ we apply a tool called `cliquer` [11, 12]. Running `cliquer` takes about 15 hours of computation in order to conclude that the maximum clique of G is of size 14. This establishes that $n = 14$ is optimal when solving model $\texttt{DNAbase}(\texttt{n})$.

Given that there is no solution to model $\texttt{DNAbase}(\texttt{n})$ when $n = 15$, we have tried to take a solution (of size 112 DNA words) defined in terms of 14 base words and extend it with some additional words. To this end, a brute force approach does succeed to add a few more words to a given solution. However, this depends on the particular base words obtained from model $\texttt{DNAbase}(\texttt{14})$. Some solutions can be extended and others not. We have found solutions of size 112 defined in terms of 14 base words which extend with up to 6 additional words. In the next section we take a more formal approach where we specifically seek 14 base words that can be extended by k additional words.

7 The Constraint Model: $\texttt{DNAbase}(\texttt{n}, \texttt{k})$

We now consider a model $\texttt{DNAbase}(\texttt{n}, \texttt{k})$ which states that S is a solution to the DNA word design problem and that S is of the form $S = S' \cup S''$ where $S' = \mathcal{T}_3(B)$ is a set of n basewords and S'' is an *extension* consisting of k words. Following are the constraints that we add to the model $\texttt{DNAbase}(\texttt{n})$. In addition to the notation introduced in Section 4, let M' denote a $k \times 16$ matrix of Boolean variables representing the k eight letter words in S'' over the alphabet $\Sigma = \{A, C, G, T\}$.

1. Constraints to ensure that words in the extension S'' are sorted (we can assume so without loss of generality):

$$\bigwedge_{i=1}^{k-1} \texttt{M}'_{\texttt{i}} \prec \texttt{M}'_{\texttt{i}+1}$$

2. Constraints to ensure that words of the extension S'' satisfy the constraints of the problem:

$$\bigwedge_{i=1}^{k} \texttt{P}(\texttt{M}'_{\texttt{i}}) \wedge \texttt{RC}_1(\texttt{M}'_{\texttt{i}}) \quad \wedge \quad \bigwedge_{1 \le i < j \le k} (\texttt{HD}(\texttt{M}'_{\texttt{i}}, \texttt{M}'_{\texttt{j}}) \wedge \texttt{RC}_2(\texttt{M}'_{\texttt{i}}, \texttt{M}'_{\texttt{j}}))$$

3. Constraints to ensure that words of the extension S'' are compatable with words in S':

$$\bigwedge_{\substack{1 \le i \le n \\ 1 \le j \le k}} \bigwedge_{u \in \mathcal{T}_3(\texttt{M}_\texttt{i})} (\texttt{HD}(\texttt{u}, \texttt{M}'_{\texttt{j}}) \wedge \texttt{RC}_2(\texttt{u}, \texttt{M}'_{\texttt{j}}))$$

8 Experimenting with the Model $\texttt{DNAbase}(\texttt{14}, \texttt{k})$

Table 2 details the results of an experiment to seek solutions to the DNA word design problem which correspond to solutions of $\texttt{DNAbase}(\texttt{n}, \texttt{k})$ where $n = 14$

k	BEE	clauses	variables	sat
1	3.141	255917	58600	323.97
2	3.553	287833	65908	13492.31
3	3.766	320029	73280	14709.89
4	4.240	352505	80716	23360.22
5	4.358	385261	88216	26553.53
6	4.933	418297	95780	5054.64
7	5.17	450328	103157	`t.o`

Table 2. Computing solutions of model `DNAbase(14,k)` (time in seconds).

k	BEE	clauses	variables	sat
7	3.45	472570	105312	22436.66
8	3.54	506166	113004	`t.o`

Table 3. Computing solutions of model `DNAbase(14,k)` with the addition of the DoubleLex constraint (time in seconds).

The columns in the table detail: k, the size of the set S'' (the solution size is 112 plus this number); BEE, the compilation time (in seconds) to obtain a CNF representation; the number of clauses and variables in the CNF, and sat, the time (in seconds) to obtain a solution using a SAT solver. A solution of 118 words, expressed in terms of a set of 14 base words with an additional 6 words is found in about 184 minutes. Our attempts to find a solution for model `DNAbase(14,k)` with $k = 7$ has failed (and we gave up on that computation after 1 week of CPU time).

9 Bottom Line: A Solution with 119 Words

The classic DoubleLex constraint was introduced in [13] to lexicographically order (linearly) both rows and columns for matrix search problems where permuting rows and columns preserves solutions. For the DNA word design problem, rows can be permuted. However, due to the `(RC)` constraint, not all column permutations preserve solutions. So, the DoubleLex constraint is not a correct symmetry break — it might prune correct solutions. Nonetheless, applying the DoubleLex constraint to model `DNAbase(14,k)` provides a solution for $k = 7$ in about 6 hours of CPU time. Table 3 details the computation. The attempt to find a solution with $k = 8$ was aborted after 1 week of CPU time. The solution with 119 words is depicted as Figure 1.

10 Discussion and Future Work

We have extended the best know result for `CSPLib 033: Word Design for DNA Computing on Surfaces`. The previous best result obtained in 2003 was a solution with 112 DNA words. We exhibit a solution with 119 words. We have tried

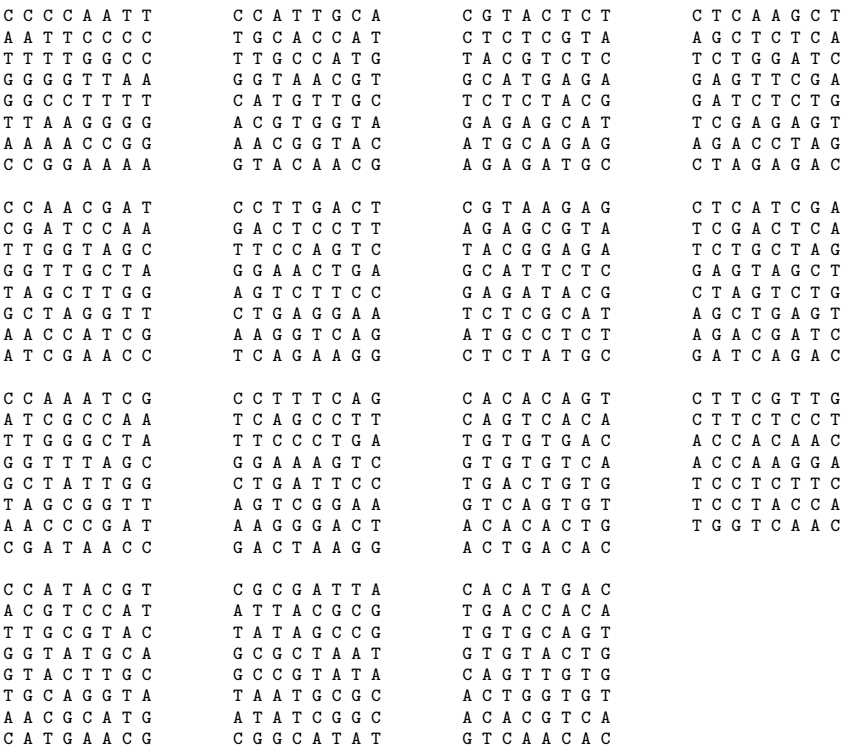

Fig. 1. A solution of the DNA word design problem with 119 words

to apply the same ideas with other sets of transformations: with two, three and four transformations. We have obtained the best results with $\mathcal{T}_3$ as described in the paper. We still have no clue concerning the existence of solutions with 120 DNA words or more. We plan to study the application of our approach to other problems in the area of codes. We hope to succeed to apply `cliquer` [11, 12] to find larger solutions to `CSPLib 033`, or to prove the optimality of the one we have detailed here. So far we have not succeeded to do so.

References

1. Frutos, A.G., Liu, Q., Thiel, A.J., Sanner, A.M.W., Condon, A.E., Smith, L.M., Corn, R.M.: Demonstration of a word design strategy for DNA computing on surfaces. Journal of Nucleic Acids Research **25**(23) (1997) 4748–4757
2. Brenner, S., R.A, L.: Encoded combinatorial chemistry. Proceedings of the National Academy of Sciences of the United States of America **89**(12) (1992) 5381–5383
3. MacWilliams, F.J., Sloane, N.J.A.N.J.A.: The theory of error correcting codes. North-Holland mathematical library. North-Holland Pub. Co. New York, Amsterdam, New York (1977) Includes index.

4. Tulpan, D.C., Hoos, H.H.: Hybrid randomised neighbourhoods improve stochastic local search for DNA code design. In Xiang, Y., Chaib-draa, B., eds.: Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2003, Halifax, Canada, June 11-13, 2003, Proceedings. Volume 2671 of Lecture Notes in Computer Science., Springer (2003) 418–433
5. Garzon, M.H., Deaton, R.J., Rose, J.A.: Soft molecular computing. In Winfree, E., Gifford, D.K., eds.: DNA Based Computers, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, June 14-15, 1999. Volume 54 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science., DIMACS/AMS (1999) 91–100
6. van Dongen, M.: CSPLib problem 033: Word design for dna computing on surfaces. `http://www.csplib.org/Problems/prob033`
7. Metodi, A., Codish, M., Lagoon, V., Stuckey, P.J.: Boolean equi-propagation for optimized SAT encoding. In Lee, J.H., ed.: Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings. Volume 6876 of Lecture Notes in Computer Science., Springer (2011) 621–636
8. Mancini, T., Micaletto, D., Patrizi, F., Cadoli, M.: Evaluating ASP and commercial solvers on the CSPLib. Constraints **13**(4) (2008) 407–436
9. Hentenryck, P.V., Michel, L., Perron, L., Régin, J.: Constraint programming in OPL. In Nadathur, G., ed.: Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings. Volume 1702 of Lecture Notes in Computer Science., Springer (1999) 98–116
10. Metodi, A., Codish, M., Stuckey, P.J.: Boolean equi-propagation for concise and efficient SAT encodings of combinatorial problems. J. Artif. Intell. Res. (JAIR) **46** (2013) 303–341
11. Niskanen, S., Östergård, P.R.J.: Cliquer user's guide. Technical Report version 1.0, Communications Laboratory, Helsinki University of Technology, Espoo, Technical Report T48 (2003)
12. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Appl. Math. **120**(1-3) (August 2002) 197–207
13. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In Van Hentenryck, P., ed.: Principles and Practice of Constraint Programming - CP 2002. Volume 2470 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 462–477