Lecture Notes in Management Science (2014) Vol. **6**: 167–175
6th International Conference on Applied Operational Research, Proceedings

ISSN 2008-0050 (Print), ISSN 1927-0097 (Online)

Three metaheuristics for the construction of constant GC-content DNA codes

R. Montemanni [1], D.H. Smith [2] and N. Koul [1]

[1] IDSIA - Università della Svizzera Italiana/SUPSI,
Via G. Buffi 13, 6904 Lugano, Canton Ticino, Switzerland
roberto@idsia.ch; nikhil.koul@usi.ch

[2] Division of Mathematics and Statistics,
University of South Wales, Pontypridd, CF37 1DL, UK
derek.smith@southwales.ac.uk

Abstract. DNA codes are sets of words of fixed length *n* over the alphabet {A, C, G, T} which satisfy a number of combinatorial conditions. The combinatorial conditions considered are (i) minimum Hamming distance *d*, (ii) fixed GC-content and, in some cases (iii) minimum distance *d* between any codeword and the reverse Watson-Crick complement of any codeword. The problem is to find DNA codes with the maximum number of codewords. In this paper three different metaheuristic approaches for the problem are discussed, and the outcome of an extensive experimental campaign, leading to many new best-known codes, is presented.

Keywords: coding theory; DNA codes; metaheuristics

Introduction

There has been considerable interest recently in the application of metaheuristic algorithms to the construction of DNA codes. These codes have applications to information storage and retrieval in synthetic DNA strands (Marathe *et al* 2001). The DNA codes are sets of words of fixed length over the alphabet {A, C, G, T}, which represents the four bases *adenine*, *guanine*, *cytosine* and *thymine*. The words of a code have to fulfill some properties, and it is desirable that the number of words in each code be as large as possible. Constructive lower bounds using algebraic coding theory, stochastic search, a template-map strategy, genetic algorithms and lexicographic codes have been proposed (Gaborit and King 2005; Chee and Ling 2008; King 2003; Li *et al* 2002; Smith *et al* 2011; Tulpan and Hoos 2003; Tulpan *et al* 2002). In Montemanni and Smith (2008) four new local search algorithms were developed and combined into a variable neighbourhood search framework.

The contributions of the present paper are threefold: (i) new codes, which arise from new runs of the variable neighbourhood search method discussed in Montemanni and Smith (2008), are made public; (ii) a new simulated annealing algorithm, able to handle larger instances than the variable neighbourhood search, is presented; (iii) a new evolutionary algorithm, able to further enlarge the spectrum of tractable instances, is presented. New best-known codes obtained by the methods are made public.

The Problem Description

A DNA word of length n is a string of length n over the alphabet {A,C,G,T}. It will be referred to as a codeword. A set of codewords is referred to as a code. The DNA code design problem considered here is to find the largest possible set of DNA words, each of length n, satisfying certain combinatorial constraints. The constraints considered here are as follows.

Hamming Distance constraint (HD): for all pairs of distinct words σ_1 and σ_2 in the code, $H(\sigma_1, \sigma_2) \geq d$, where $H(\sigma_1, \sigma_2)$ is the Hamming distance between words σ_1 and σ_2, namely the number of positions i at which the ith letter in σ_1 differs from the ith letter in σ_2;

GC-content constraint (GC): a fixed number w (taken here to be $\lfloor n/2 \rfloor$) of letters of each word are either G or C. For a word σ_i the number of letters which are G or C is denoted $GC(\sigma_i)$;

Reverse complement Hamming distance constraint (RC): for a word $\sigma_i = x_1 x_2 \ldots x_n$ the Watson-Crick complement of σ_i is $wcc(\sigma_i) = x_n^C x_{n-1}^C \ldots x_1^C$, where A^C=T, C^C=G, G^C=C, T^C=A. For all (not necessarily distinct) pairs of DNA words σ_1 and σ_2 in the code, the RC constraint specifies that $H(\sigma_1, wcc(\sigma_2)) \geq d$.

$A_4^{GC}(n,d,w)$ and $A_4^{GC,RC}(n,d,w)$ denote the maximum number of codewords in a DNA code satisfying the first two constraints (with the RC constraint not considered) and satisfying all three constraints, respectively.

A Variable Neighbourhood Search approach

In Montemanni and Smith (2008) a metaheuristic algorithm is presented. It combines four local search procedures (described later) in a Variable Neighbourhood Search (VNS) framework. Such algorithms work by applying different local search algorithms one after the other, aiming at differentiating the characteristics of the search-spaces visited (i.e. changing the neighbourhood). The local search methods are applied in turn, starting each time from the best solution retrieved since the beginning, or from an empty solution. The reader interested in more details is referred to Montemanni and Smith (2008). The parameter $Time_{VNS}$ denotes the number of seconds for which the VNS algorithm runs.

Seed Building (SB). One class of algorithm examines all possible codewords in a given order, and accepts codewords incrementally if they are feasible with respect to already accepted ones (Gaborit and King 2005, King 2003). This idea can be hybridized with the concept of *seed codewords*, which are an initial set of codewords with the required GC-content, to which codewords are added in the given ordering if they satisfy the necessary criteria. The algorithm evolves the set of codewords over time, modifying it every *ItrSeed* iterations (an iteration is the examination of all possible codewords). A computation time of $Time_{SB}$ seconds is allowed for each run of the algorithm. **Clique Search (CS).** A partial code can be computed by solving a maximum clique problem (Chee and Ling 2008). In this method a random subset of the codewords of a given code is removed, leaving a partial code. A parameter *CSRem* defines the percentage of codewords removed. All codewords compatible with those left in the code can be identified, a graph can then be built, and a maximum clique problem solved to complete the partial code. A maximum computation time of $Time_{CS}$ seconds is allowed, while a maximum time of $Time_{MC}$ is set for each maximum clique problem. **Hybrid Search (HS).** Hybrid search merges the concepts used in seed building and clique search, attempting to combine the best characteristics of the two methods. For this purpose, a concept of weak (Hamming) distance, regulated by parameter *HSRel*, is introduced. A computation time of $Time_{HS}$ seconds is allowed for each run of the local search. **Iterated Greedy Search (IGS).** The method is inspired by that discussed in Tulpan *et al* (2002), and, in contrast to the local searches previously reviewed, works on an *infeasible* set of codewords, trying to make if feasible. There are three parameters: *IGSChg* regulating the percentage of codewords of the input solution scrambled when the local search starts, *ItrSrc* implementing a restarting mechanism in case of starvation, and $Time_{IGS}$, representing the maximum time allowed for each run of the method.

A Simulated Annealing approach

Longer codes are desirable for applications. The main drawback of the VNS approach previously described is that it cannot handle problems with large values of n. This is a result of the explosion in the computation time required by the local search procedures working on feasible sets of codewords. For this reason a new Simulated Annealing (SA) approach is presented. Simulated Annealing is a metaheuristic algorithm derived from thermodynamic principles, originally applied to combinatorial optimization in Kirkpatrick *et al* (1983). The search algorithm proceeds with the cost function reducing most of the time, but it is allowed to increase sometimes to permit escape from local minima which are not global minima. In particular, such an algorithm is based on the connection between the physical concept of temperature in annealing processes and the mathematical concept of the probability of accepting a cost-increasing solution. The probability will be high initially and will decrease slowly, like the temperature in the annealing process that produces the regular crystal. In the remainder of this section an adaptation of these principles to the problem under investigation will be described. In a similar way to the IGS local search discussed previously, the implementation of the SA idea for DNA code design works on infeasible sets of codewords. This means that not all of the codewords are

compatible with each other according to the constraints. The algorithm tries to make them feasible. The method proceeds by modifying codewords with the target of reducing (a measure of) the constraint violations. If no violations remain, then a feasible solution has been retrieved. The measure of infeasibility adopted for a set of codewords W is given by the following equation:

$$Inf(\mathrm{W}) = \sum_{w_1 \in \mathrm{W}} |GC(\sigma_1) - \lfloor n/2 \rfloor| + \sum_{w_1, w_2 \in \mathrm{W},\ w_1 \neq w_2} \max\{0, d - H(w_1, w_2)\} + \sum_{w_1, w_2 \in \mathrm{W}} \max\{0, d - H(w_1, wcc(w_2))\} \quad (1)$$

$$Meas(\mathrm{W}) = Inf(\mathrm{W}) + \epsilon \sum_{j \in \{1,2,\cdots,n\}} \left(\max_{i \in \{1,2,\cdots,|W|\}} \{v_{ij}\} - \min_{i \in \{1,2,\cdots,|W|\}} \{v_{ij}\} \right) + \epsilon \sum_{i \in \{1,2,\cdots,|W|\}} \left(\max_{j \in \{1,2,\cdots,n\}} \{v_{ij}\} - \min_{j \in \{1,2,\cdots,n\}} \{v_{ij}\} \right) \quad (2)$$

where the last term in the sum is considered only when RC constraints are active. *Inf(W)* gives a measure of by how much constraints are violated in the code W. The target is to repeatedly modify the code W towards a feasible solution (i.e. *Inf(W)=0*). The fitness function used is, however, slightly more complex than that defined in (1). Given a code W, let v_{ij} represent the number of violations between codeword i and the other codewords (and reverse complements of codewords when the RC constraint is considered) in which the letters in position j are the same in the two words. The modified measure of infeasibility adopted for the SA is then (2), where ε is an arbitrary small constant inserted to make the *Inf(W)* term dominate the other two.

The aim of the second and third terms of (2) is to spread the number of violations as evenly as possible over positions (columns of the code) and words (rows of the code). The rationale behind (2) is that usually codes with balanced violations are easier to turn into a feasible code. Preliminary tests showed that this strategy speeds up the algorithm substantially. During each iteration, a codeword w of solution W is selected at random, and the change of one of its letters that guarantees the maximum decrease in the infeasibility measure is selected (ties among possible modifications of W are broken randomly). Non-improving new codes are probabilistically accepted with probability defined by $prob = min\{1, e^{-(\Delta Meas(W)/t)}\}$, where $\Delta Meas(W)$ is the change in the infeasibility measure following the optimal change of one letter of a randomly selected codeword, and t is a temperature parameter initialized to *Initt* and decreased every *Nitert* iterations according to the formula $t := Decrt \cdot t$. The procedure stops when *Inf (W)*=0 or $t < Finalt$. If the loop is exited because *Inf(W)*=0, then a new feasible code W has been found. A random codeword w is then added to W and the procedure is restarted. If *Inf(W)*>0 and $t < Finalt$, then the temperature t is reinitialized to *Initt*, and the annealing process is restarted. The execution of the algorithm stops when a maximum computation time of $Time_{SA}$ seconds has elapsed.

An Evolutionary Algorithm approach

It has been observed that an approach which works by incrementally inserting new feasible random codewords into a partial code, produces promising codes. Here this idea is exploited further by hybridizing it with the infeasibility measure (1) within an Evolutionary Algorithm (EA), targeting therefore longer codes (with higher values of n) and larger

codes (with more codewords) (Koul 2010). An Evolutionary Algorithm is a method incorporating ideas from natural selection, based on survival of the fittest. An Evolutionary Algorithm maintains a population of structures, that evolves according to evolutionary rules simulating selection, recombination, mutation and survival as seen in Nature. These operators are often referred to as genetic operators. A fitness or performance indicator is associated with each individual in the population.

In Nature the fittest individuals are more likely to be selected for reproduction, and recombination and mutation modify those individuals, yielding potentially superior ones in the next generation. In Evolutionary Algorithms, the fitness value associated with each individual will therefore drive these processes and improve the quality of the population in terms of the average fitness (which corresponds to a set of more optimized solutions of the optimization problem under investigation). The interested reader is referred to Bäck (1996) for a more detailed description of this optimization paradigm. The main idea of the approach proposed is to maintain a set of pools of codewords, which are feasible according to the GC-content constraint. These pools will be used to enlarge a partial code W under construction (corresponding to the empty set initially).

The evolutionary phase of the approach resides in the way the pools are evolved once all the codewords feasible with W have been added to W (this phase corresponds to the end of an iteration of the algorithm). The evolutionary operator applied to each pool is a swap mutation on each codeword of the pool. This avoids starvation phenomena of the algorithm, and guarantees the search to be evenly spread over the space, since each pool covers a well-defined portion of the whole search space. A recombination (crossover) operator is not used in this implementation, due to the strategy of statically dividing the search space over the different individuals of the population (pools of codewords).

In more detail, the algorithm starts by assigning every possible combination of positions for the bases G or C evenly to the N_p pools of the population. More combinations may be assigned to a pool than the eventual size of the pool, depending on the ratio n/N_p. Pools are then filled with N_W random codewords that have base G or C in the positions defined by one of the combinations assigned to each pool. This property is respected initially, and during the running of the algorithm: in the initialization phase the pools give the code a better spread over the possible regions of the search space and during the running of the algorithm they reduce the risk that only some regions of the search space are considered. The working code W is initialized to the empty set, and is incremented by selecting feasible codewords from the pools. In particular, a loop over the pools is entered and, for each pool, codewords which are feasible with respect to W are added to W with probability p_A. The loop is exited when no more feasible codewords are contained in the pools. This concludes the proper initialization phase. A double loop, corresponding to the main section of the algorithm, is then entered. In the inner loop, an evolutionary operator is applied to the pools.

The evolutionary operator adopted is to choose a random position for every codeword and to swap the chosen base with the single alternative (i.e. swap C with G or swap A with T). This ensures that the GC-content constraint is still fulfilled. Then evolved pools are ranked by increasing values of the infeasibility measure (1) calculated on the set $W \cup Pool_i$ for each $Pool_i$ of the population (i.e. $Inf(W \cup Pool_i)$). The evolved pools are examined in the order defined by the ranking, trying to enlarge the code W. The inner loop is repeated until N_{IT} consecutive iterations are executed without improvements to

W. In the second part of the outer loop, code *W* is partially destroyed (after having saved *W* in case it was the best code retrieved) in order to differentiate the search over the solution space. In particular, the words of *W* are removed with probability p_D. The outer loop (and the algorithm) terminates when a given maximum time $Time_{EA}$ has elapsed.

Tables

In the following tables lower bounds are given for $A_4^{GC}(n,d,w)$ and $A_4^{GC,RC}(n,d,w)$ for $4 \le n \le 20$, $3 \le d \le n$ and $21 \le n \le 30$, $13 \le d \le n$. For each combination (n,d), the best result obtained by the algorithms proposed here is reported, together with the best known result available from the literature.

The entry *B* summarizes the results presented in Gaborit and King (2005), Chee and Ling (2008), King (2003), Li *et al* (2002), Smith *et al* (2011), Tulpan and Hoos (2003) and Tulpan *et al* (2002). The *V* entries reported without a superscript have been obtained by the VNS algorithm for $4 \le n \le 20$, while *S* entries reported without a superscript have been obtained by the SA algorithm for $21 \le n \le 30$. When the superscript *E* is present, the best result has been obtained by the EA algorithm. When the superscript *N* is present, it means that the best result has been obtained by new runs of the VNS method (with respect to Montemanni and Smith (2008)).

The tests reported in Tables 1 and 2 were carried out on Dual AMD Opteron 250 2.4GHz / 4GB RAM machines with the following parameter settings for the algorithms: $Time_{VNS}$=100000; $Time_{SB}$=500 for problems with $n \le 15$, and 0 for problems with $n>15$; $Time_{CS}$=2000; $Time_{HS}$=200; $Time_{IGS}$=1000; $Time_{MC}$=30; *ItrSeed*=20; *CSRem*=30 for problems with $n \le 15$, and 10 for problems with $n>15$; *HSRel*=2 for problems with $n \le 15$, and 1 for problems with $n>15$; *IGSChg*=20; *ItrSrc*=10000; $Time_{SA}$=300000; *Initt*=30000; *Nitert*=10000; *Decrt*=0.9; *Finalt*=300; $Time_{EA}$ =300000; N_P=10; N_W=10000; N_{IT}=100; p_A=0.1; p_D=0.02. If $Time_{SB}$=0 then SB is run for one iteration in order to obtain a starting feasible solution for the VNS approach. Multiple runs were used for the VNS algorithm; typically 5 runs for each case and never more than 10 runs.

The values reported for the parameters were found experimentally during some calibration experiments, and provided a satisfactory balance between computation times and the effectiveness of the algorithms. The computation times for the best-known results taken from Gaborit and King (2005) were stated there to be "up to a few days" on a slightly slower computer. This is comparable with the computation times used here. Entries of the tables followed by periods are optimal by the Johnson-type upper bounds in King (2003). Entries followed by colons are optimal by max-clique computations in Chee and Ling (2008). Entries in bold are new best lower bounds found by the algorithms discussed in this paper.

Tables 1 and 2 suggest that the VNS algorithm is competitive with state-of-the-art methods on most of the instances with $n \le 20$ considered. In particular, it is able to match 116 previous lower bounds (59 for $A_4^{GC}(n,d,w)$ instances and 57 for $A_4^{GC,RC}(n,d,w)$ instances). It also provides 73 new best lower bounds (30 for $A_4^{GC}(n,d,w)$ instances and 43 for $A_4^{GC,RC}(n,d,w)$ instances). It is interesting to observe that VNS was able to match all previously known optimal codes, and to find a previously unknown optimal code $A_4^{GC}(11,9,w)=11$. The effects of running VNS multiple times can be appreciated by

comparing these results with those reported in Montemanni and Smith (2008), where only one run was considered for each instance. The number of new lower bounds in that case was only 52. The contribution of the SA algorithm can be appreciated for problems with $n \geq 20$, for which results are reported here for the first time. The method proves effective when d is close to n. For the other entries of the second part of Tables 1 and 2, the best known results are retrieved (together with some new codes for $n \leq 20$) by the EA method. It is more general than the other two approaches in terms of instances handled (it is the only method able to deal with codes with more than a few thousand codewords). The EA approach was also able to retrieve 3 further new lower bounds for $n \leq 20$. On the other hand, EA seems to be dominated by the other approaches when d is close to n. The codes corresponding to these new lower bounds, together with the new codes for $21 \leq n \leq 30$, are available at http://www.idsia.ch/~roberto/DNA09.zip or at http://data.research.southwales.ac.uk/projects/.

Table 1. Lower bounds for $A_4^{GC}(n,d,w)$.

n\d		3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	*V*	12	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	12.	4.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	*V*	30	10	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	30:	10.	3.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	*V*	112	40	8	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	112	40.	8:	4.	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	*V*	**288**^{N}	**80**^{E}	22	7	3	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	280	78	22	7.	3.	-	-	-	-	-	-	-	-	-	-	-	-	-
8	*V*	842	256^N	**63**^{N}	**28**	5	4	-	-	-	-	-	-	-	-	-	-	-	-
	B	1056	256	56	24	5.	4.	-	-	-	-	-	-	-	-	-	-	-	-
9	*V*	2480^N	545^N	134^N	**48**^{N}	**18**^{N}	5	3	-	-	-	-	-	-	-	-	-	-	-
	B	3012	555	134	40	16	5.	3.	-	-	-	-	-	-	-	-	-	-	-
10	*V*	-	**2031**^{E}	366^N	101^N	**34**	16	5	4	-	-	-	-	-	-	-	-	-	-
	B	-	2016	504	144	32	16.	5.	4.	-	-	-	-	-	-	-	-	-	-
11	*V*	-	-	977	245^N	**75**^{N}	30^N	***11.***	4	3	-	-	-	-	-	-	-	-	-
	B	-	-	1848	472	72	32	10	4.	3.	-	-	-	-	-	-	-	-	-
12	*V*	-	-	2776^E	653	**183**	**118**^{N}	**24**	9	4	4	-	-	-	-	-	-	-	-
	B	-	-	3696	1888	179	68	23	9.	4.	4.	-	-	-	-	-	-	-	-
13	*V*	-	-	-	1717	430^N	133^N	**46**	20	8	4	3	-	-	-	-	-	-	-
	B	-	-	-	3432	464	134	44	20	8.	4.	3.	-	-	-	-	-	-	-
14	*V*	-	-	-	-	1129	315^N	103^N	40	**17**^{N}	8	4	4	-	-	-	-	-	-
	B	-	-	-	-	1856	512	112	49	16	8.	4.	4.	-	-	-	-	-	-
15	*V*	-	-	-	-	6960	738	**227**^{N}	79^N	**34**^{N}	**15**^{N}	6	4	3	-	-	-	-	-
	B	-	-	-	-	6960	1920	225	124	30	13	6.	4.	3.	-	-	-	-	-
16	*V*	-	-	-	-	4528^E	1898	526^N	174^N	65^N	29^N	12	5	4	4	-	-	-	-
	B	-	-	-	-	27840	6680	532	177	117	60	12.	5.	4.	4.	-	-	-	-
17	*V*	-	-	-	-	10872^E	2558^E	1244	372^N	132^N	54^N	**24**^{N}	9	5	4	3	-	-	-
	B	-	-	-	-	48620	24310	1530	380	132	123	22	9.	5.	4.	3.	-	-	-
18	*V*	-	-	-	-	97034^E	4797^E	3136	871	**282**^{N}	105^N	**46**^{N}	**20**^{N}	9	5	4	4	-	-
	B	-	-	-	-	97520	87516	5508	920	216	180	38	18	9.	5.	4.	4.	-	-
19	*V*	-	-	-	-	-	14336^E	3147^E	**2047**	**615**	**213**^{N}	**83**^{N}	**38**^{N}	**17**^{N}	8	5	4	3	-
	B	-	-	-	-	-	92378	7038	1326	431	163	71	33	15	8.	5.	4.	3.	-
20	*V*	-	-	-	-	-	-	8235^E	3076	1184	457	**167**^{N}	**69**^{N}	**33**^{N}	**16**^{N}	8	5	4	4
	B	-	-	-	-	-	-	24552	5882	2112	520	130	58	31	13	8.	5.	4.	4.
n\d		**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
21	*S*	176^E	86	44	22	12	7	4	4	3	-	-	-	-	-	-	-	-	-
22	*S*	348^E	148	72	40	20	12	6	4	4	4	-	-	-	-	-	-	-	-
23	*S*	672^E	252	114	62	32	17	9	5	4	4	3	-	-	-	-	-	-	-
24	*S*	1380^E	488^E	204	102	54	28	15	9	5	4	4	4	-	-	-	-	-	-
25	*S*	2804^E	960^E	380^E	166^E	130^E	46	24	13	8	5	4	4	3	-	-	-	-	-
26	*S*	5948^E	1954^E	702^E	296^E	134	76	40	22	12	8	5	4	4	4	-	-	-	-
27	*S*	12616^E	3854^E	1310^E	524^E	228	112	64	36	19	11	8	5	4	4	3	-	-	-
28	*S*	27376^E	7974^E	2620^E	918^E	388	186	100	56	30	17	10	8	5	4	4	4	-	-
29	*S*	54752^E	16490^E	5198^E	1796^E	706^E	310	154	84	48	26	15	9	6	5	4	4	3	-
30	*S*	122540^E	35354^E	10852^E	3534^E	1092	532	254	130	72	41	23	14	9	6	5	4	4	4

Table 2. Lower bounds for $A_4^{GC,RC}(n,d,w)$.

n\d		3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	*V*	6	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	6.	2.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	*V*	15	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	15:	3:	1.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	*V*	42^N	16	4	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	44	16:	4.	2.	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	*V*	126^N	35	11	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-
	B	135	36	11:	2:	1.	-	-	-	-	-	-	-	-	-	-	-	-	-
8	*V*	417	101^N	27^N	12	2	2	-	-	-	-	-	-	-	-	-	-	-	-
	B	528	128	28	12	2.	2.	-	-	-	-	-	-	-	-	-	-	-	-
9	*V*	1199	269	67^N	**21^N**	8	2	1	-	-	-	-	-	-	-	-	-	-	-
	B	1354	275	67	20	8	2.	1.	-	-	-	-	-	-	-	-	-	-	-
10	*V*	3927	821	**176^N**	49^N	**17^N**	8	2	2	-	-	-	-	-	-	-	-	-	-
	B	4542	855	175	54	16	8.	2.	2.	-	-	-	-	-	-	-	-	-	-
11	*V*	10993^E	2404	471	117	**37**	**14**	5	2	1	-	-	-	-	-	-	-	-	-
	B	14405	2457	477	117	36	13	5.	2.	1.	-	-	-	-	-	-	-	-	-
12	*V*	-	6186^E	**1381**	311	**87**	**29**	**12^N**	4	2	2	-	-	-	-	-	-	-	-
	B	-	14624	1369	924	83	28	11	4.	2.	2.	-	-	-	-	-	-	-	-
13	*V*	-	-	**3974**	819	**206**	**62**	**23^N**	**10^N**	4	2	1	-	-	-	-	-	-	-
	B	-	-	3954	924	205	61	22	9	4.	2.	1.	-	-	-	-	-	-	-
14	*V*	-	-	-	2354	532	147	**49^N**	**20^N**	**8**	4	2	2	-	-	-	-	-	-
	B	-	-	-	2963	749	180	46	16	7	4.	2.	2.	-	-	-	-	-	-
15	*V*	-	-	-	**6634**	1379	**347**	**109^N**	**37**	16	6	3	2	1	-	-	-	-	-
	B	-	-	-	6430	1600	343	102	35	18	6	3.	2.	1.	-	-	-	-	-
16	*V*	-	-	-	-	-	895	**243^N**	**83^N**	31^N	14	5	2	2	2	-	-	-	-
	B	-	-	-	-	-	3264	230	74	52	24	5	2.	2.	2.	-	-	-	-
17	*V*	-	-	-	-	-	2281	**579**	**175^N**	**62^N**	25	**12^N**	4	2	2	1	-	-	-
	B	-	-	-	-	-	6060	549	164	56	30	11	4.	2.	2.	1.	-	-	-
18	*V*	-	-	-	-	-	-	**1459**	**407**	**133^N**	**49^N**	**21^N**	**10^N**	4	2	2	2	-	-
	B	-	-	-	-	-	-	1403	387	104	43	19	9	4.	2.	2.	2.	-	-
19	*V*	-	-	-	-	-	-	**3678^N**	**960**	**285**	**99^N**	**39^N**	**18^N**	**8^N**	4	2	2	1	-
	B	-	-	-	-	-	-	3519	909	215	80	35	16	7	4.	2.	2.	1.	-
20	*V*	-	-	-	-	-	-	-	1538	230	112	**77^N**	**33^N**	**15^N**	**7^N**	4	2	2	2
	B	-	-	-	-	-	-	-	2868	766	179	64	29	14	6	4.	2.	2.	2.

n\d		13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
21	*S*	88^E	43	22	11	6	3	2	2	1	-	-	-	-	-	-	-	-	-
22	*S*	174^E	74	36	20	10	6	2	2	2	2	-	-	-	-	-	-	-	-
23	*S*	336^E	126	57	31	16	8	4	2	2	2	1	-	-	-	-	-	-	-
24	*S*	690^E	244^E	102	51	27	14	7	4	2	2	2	2	-	-	-	-	-	-
25	*S*	1402^E	480^E	190^E	83^E	65^E	23	12	6	4	2	2	2	1	-	-	-	-	-
26	*S*	2974^E	977^E	351^E	148^E	67	38	20	11	6	4	2	2	2	2	-	-	-	-
27	*S*	6308^E	1927^E	655^E	262^E	114	56	32	18	9	5	4	2	2	2	1	-	-	-
28	*S*	13688^E	3987^E	1310^E	459^E	194	93	50	28	15	8	4	4	2	2	2	2	-	-
29	*S*	29292^E	8245^E	2599^E	898^E	353^E	155	77	42	24	12	7	4	3	2	2	2	1	-
30	*S*	61270^E	17677^E	5426^E	1767^E	546	266	127	65	36	20	11	7	4	3	2	2	2	2

Conclusions

The generation of DNA codes has been considered from an algorithmic perspective. On instances with $4 \le n \le 20$ and $3 \le d \le n$, the methods presented were able to improve the previously known code for 80 instances and matched the best-known code for another 114 instance (over 267 instances considered). The algorithms were also able to provide the first lower bounds for codes with $21 \le n \le 30$ and $13 \le d \le n$.

References

Bäck T (1996). Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press

Chee YM and Ling S (2008). Improved lower bounds for constant GC-content DNA codes. IEEE Transactions on Information Theory 54(1):391-394

Gaborit P and King OD (2005). Linear construction for DNA codes. Theoretical Computer Science 334:99-113

King OD (2003) Bounds for DNA codes with constant GC-content. Electr. Journal of Combinatorics 10:#R33

Kirkpatrick S, Gelatt C and Vecchi M (1983). Optimization by simulated annealing. Science 220(4598):671-680

Koul K (2010). Heuristic Algorithms for Construction of Constant GC-content DNA codes. Master thesis, Università della Svizzera Italiana

Li M, Lee HJ, Condon AE and Corn RM (2002). DNA word design strategy for creating sets of non-interacting oligonucleotides for DNA microarrays. Langmuir 18:805-812

Marathe A, Condon AE and Corn RM (2001). On combinatorial DNA word design. Journal of Computational Biology 8:201-219

Montemanni R and Smith DH (2008). Construction of constant GC-content DNA codes via a Variable Neighbourhood Search algorithm. Journal of Mathematical Modelling and Algorithms 7:311-326

Smith DH, Aboluion N, Montemanni R and Perkins S (2011). Linear and nonlinear constructions of DNA codes with constant GC-content. Discrete Mathematics 311(14):1207-1219

Tulpan DC and Hoos HH (2003). Hybrid randomised neighbourhoods improve stochastic local search for DNA code design. Lecture Notes in Computer Science 2671:418-433

Tulpan DC, Hoos HH and Condon AE (2002). Stochastic local search algorithms for DNA word design. Lecture Notes in Computer Science 2568:229-241