Stochastic Local Search Algorithms for DNA Word Design

Dan C. Tulpan, Holger Hoos, and Anne Condon
University of British Columbia
Department of Computer Science
dctulpan@cs.ubc.ca, hoos@cs.ubc.ca, condon@cs.ubc.ca

10th March 2002

Abstract

We present results on the performance of a stochastic local search algorithm for the design of DNA codes, namely sets of equal-length words over the nucleotides alphabet $\{A, C, G, T\}$ that satisfy certain combinatorial constraints. Using empirical analysis of the algorithm, we gain insight on good design principles. We report several cases in which our algorithm finds word sets that match or exceed the best previously known constructions.

1 Introduction

The design of DNA code words, or sets of short DNA strands that satisfy combinatorial constraints, is motivated by the tasks of storing information in DNA strands used for computation or as molecular bar-codes in chemical libraries [2, 3, 7, 24]. Good word design is important in order to minimize errors due to non-specific hybridization between distinct words and their complements, to obtain a higher information density, and to obtain large sets of words for large-scale applications.

For the types of combinatorial constraints typically desired, there are no known efficient algorithms for design of DNA word sets. Techniques from coding theory have been applied to design of DNA word sets [3, 9]. While valuable, this approach is hampered by the complexity of the combinatorial constraints on the word sets, which are often hard to reason about theoretically. For these reasons, heuristic approaches such as stochastic local search offer much promise in design of word sets.

Stochastic local search algorithms strongly use randomised decisions while searching for solutions to a given problem. They play an increasingly important role for solving hard combinatorial problems from various domains of Artificial Intelligence and Operations Research, such as satisfiability, constraint satisfaction, planning, scheduling, and other application areas. Over the past few years there has been considerable success in developing stochastic local search algorithms as well as randomised systematic search methods for solving these problems, and to date, stochastic search algorithms are amongst the best known techniques for solving problems from many domains. Detailed empirical studies are crucial for the analysis and development of such high-performance stochastic search techniques.

Stochastic search methods have already been applied to the design of DNA word sets. Deaton et al. [4, 5] and Zhang and Shin [25, 26] used genetic algorithms for design of DNA code words, and provide some small sets of code words that satisfy well-motivated combinatorial constraints. However, some details of algorithms are not specified in these papers. Faulhammer et al. [7] also use a stochastic search approach and provide the code for their algorithm. In all cases, while small sets of code words produced by the algorithms have been presented (and the papers make other contributions independent of the word design algorithms), little or no analysis of algorithm performance is provided. As a result it is not possible to extract general insights on design of stochastic algorithms for code design or to do detailed comparisons of their approaches with other algorithms. Our goal is to understand what algorithmic principles are most effective in the application of stochastic local search methods to the design of DNA or RNA word sets (and more generally, codes over other alphabets, particularly the binary alphabet).

Towards this end, we describe a simple stochastic local search algorithm for design of DNA codes, and analyze its performance using an empirical methodology based on run-time distributions [17]. In this study, we have chosen to design word sets that satisfy one or more of the following constraints: Hamming distance (**HD**), GC content (**GC**), and reverse complement Hamming distance (**RC**). We define these constraints precisely in Section 2. Our reason for considering these constraints is that there are already some constructions for word sets satisfying these constraints, obtained using both theoretical and experimental methods, with which we can compare our results. (In follow-up work, we will apply our methods to other constraints too.) Our algorithm takes as input the desired word length and set size, along with a specification of which of these constraints the set should satisfy, and attempts to find a set that meets these requirements.

Our algorithm, described in detail in Section 4, performs local search in a space of DNA word sets of fixed size that may violate the given constraints. The underlying search strategy is based on a combination of randomized iterative improvement and conflict-directed random walk. The basic algorithm is initialized with a randomly selected set of DNA words. Then, repeatedly a conflict, that is, a pair of words that violates a constraint, is selected and resolved by modifying one of the respective words. The algorithm terminates if a set of DNA words that satisfies all given constraints is found, or if a specified number of iterations have been completed.

The performance of this algorithm is primarily controlled by a so-called noise parameter that determines the probability of greedy vs. random conflict resolution. Interestingly, optimal settings of this parameter appear to be consistent for different problem instances (word set sizes) and constraints.

Our empirical results, reported in section 5, show that the run-time distributions that characterize our algorithm's performance on hard word design problems often have a heavy right tail. This indicates a stagnation of the underlying search process that severely compromises performance. As a first approach to overcome this stagnation effect, we extended our algorithm with a mechanism for diversifying the search by occasional random replacement of a small fraction of the current set of DNA words. Empirically, this modification eliminates the heavy-tailed behavior and leads to substantial improvements in the performance of our algorithm.

We compared the sizes of the word sets obtainable by our algorithm with previously known word sets, starting with the already extensively studied case of word sets that satisfy the Hamming distance constraint only. Out of 43 tests, our algorithm was able to find a word set whose size matches the best known theoretical construction in 12 cases. We also did several comparisons with word sets that satisfy at least two of our three constraints, for which again previous results were available. Out of a total of 71 comparisons with previous results, we found word sets that improved on previous constructions in all but three cases. In one of these three cases, while our algorithm was not able to meet the previous best construction when starting from a random initial set of words, we were able to improve on the best previous construction by initializing our algorithm with the best previously known word set plus an additional random word. We later provide several tables of set sizes obtained by our algorithm.

2 Problem Description

The DNA code design problem that we consider is: given a target k and word length n, find a set of k DNA words, each of length n, satisfying certain combinatorial constraints. A DNA word of length n is simply a string of length n over the alphabet $\{A, C, G, T\}$, and naturally corresponds to a DNA strand with left end of the string corresponding to the 5' end of the DNA strand. The constraints that we consider are:

- **H**amming **D**istance Constraint (**HD**): for all pairs of distinct words w_1, w_2 in the set, $H(w_1,w_2) \geq d$. Here, $H(w_1,w_2)$ represents the Hamming distance between words w_1 and w_2, namely the number of positions i at which the ith letter in w_1 differs from the ith letter in w_2.
- **GC** Content Constraint (**GC**): a fixed percentage of the nucleotides within each word is either G or C. Throughout, we assume that this percentage is 50%.
- **R**everse **C**omplement Hamming Distance Constraint (**RC**): for all pairs of DNA words w_1 and w_2 in the set, where w_1 may equal w_2, $H(w_1,wcc(w_2)) \geq d$. Here, $wcc(w)$ denotes the Watson-Crick complement of DNA word w, obtained by reversing w and then by replacing each A in w by T and vice versa, and replacing each C in w by G and vice versa.

Motivation for considering these constraints can be found in many sources; see for example Frutos et al. [9].

The total number of code words of length n defined over any quaternary alphabet is 4^n. The number of possible word sets of size k that can be formed with 4^n code words is:

$$\binom{4^n}{k} = \frac{(4^n)!}{k! \times (4^n - k)!}$$

For the particular example of code words with $n = 8$ and $k = 100$, the number of all possible word sets can be approximated by: 1.75×10^{267}.

The huge number of possible sets that must be explored in order to find a big set of words suggests the use of non-exhaustive search algorithms for solving this type of problems.

3 Related Work

Stochastic search methods have been used successfully for decades in the construction of good binary codes (see for example [6, 14]). Typically, the focus of this work is in finding codes of size greater than the best previously known bound, and a detailed empirical analysis of the search algorithms is now presented.

Deaton et al. [4, 5] and Zhang and Shin [25, 26] describe genetic algorithms for finding DNA codes that satisfy much stronger constraints than the HD and RC constraints, in which "frame shifts" are taken into account. However, they do not provide a detailed analysis of the performance of their algorithms.

Hartemink et al. [13] used a computer algorithm for designing word sets that satisfy yet other constraints, in which a large pool (several billion) of strands were screened in order to determine whether they meet the constraints.

Several other researchers have used computer algorithms to generate word sets (see for example [2]), but provide no details on the algorithms.

Some DNA word design programs are publicly available. The DNASequenceGenerator program [22, 8] designs DNA sequences that satisfy certain subword distance constraints and, in addition, have melting temperature or GC content within prescribed ranges. The program can generate DNA sequences de novo, or integrate partially specified words or existing words into the set. The PERMUTE program was used to design the sequences of Faulhammer et al. [7, 21] for their RNA-based 10-variable computation.

4 A Stochastic Local Search Algorithm

Our basic stochastic local search algorithm performs randomised iterative improvement search in the space of DNA word sets; this overall approach is similar to WalkSAT, one of the best known algorithms for the propositional satisfiability problem [20, 15] and Casanova, one of the best-performing algorithms for the winner determination problem in combinatorial auctions [16]. All word sets constructed during the search process have exactly the target number of words. Additionally, when using the **GC** constraint, we ensure that all words in the given candidate set always have the prescribed GC content.

The other two combinatorial constraints considered in this paper, the **HD** and **RC** constraints, are binary predicates over DNA words that have to be satisfied between any pair of words from a given DNA set. The function our algorithm is trying to minimise is the number of pairs that violate the given constraint(s). Figure 1 gives a general outline of our basic algorithm. In the following, we will describe this algorithm in more detail.

The initial set of words is determined by a simple randomised process that generates any DNA word of length n with equal probability. If the GC content constraint is used, we check each word that is generated and only accept words with the specified GC content. This works well for 50% GC content and typical designs use GC content close to 50%, but could be easily made more efficient for less balanced GC content if needed.

Our implementation also provides the possibility to initialise the search with a given set of DNA words (such sets could be obtained using arbitrary other methods); if such a set has less than k words, it is expanded with randomly generated words such that a set of k words is always obtained. Note that the initial word set may contain multiple copies of the same word.

In each iteration of the search process, first, a pair of words violating one of the Hamming distance constraints is selected uniformly at random. Then, for each of these words, all possible single-base modifications are considered.

As an example of single-base modifications take the code word $ACTT$ of length 4. A new code word $GCTT$ can be obtained by replacing letter A from the first code word with letter G.

When the GC content constraint is used, this is restricted to modifications that maintain the specified GC content. For a pair of words of length n without the GC content constraint, this yields $6n$ new words. (Some of these might be identical.) With a fixed probability θ, one of these modification is accepted uniformly at random, regardless of the number of constraint violations that will result from it. In the remaining cases, each modification is assigned a score, defined as the net decrease in the number of constraint violations caused by it, and a modification with maximal score is accepted. (If there are multiple such modifications, one of them is chosen uniformly at random.) Note that

```
procedure StochasticLocalSearch for DNA word design
    input: Number of words ($k$), word length ($n$), set of combinatorial constraints ($C$)
    output: Set $S$ of $m$ words that fully or partially satisfies $C$
    for $i := 1$ to maxTries do
        $S$ := initial set of words
        $\hat{S} := S$
        for $j := 1$ to maxSteps do
            if $S$ satisfies all constraints then
                return $S$
            end if
            Randomly select words $w_1, w_2 \in S$ that violate one of the constraints
            $M_1$ := all words obtained from $w_1$ by substituting one base
            $M_2$ := all words obtained from $w_2$ by substituting one base
            with probability $\theta$ do
                select word $w'$ from $M_1 \cup M_2$ at random
            otherwise
                select word $w'$ from $M_1 \cup M_2$ such that number of conflict violations is maximally decreased
            end with probability
            if $w' \in M_1$ then
                replace $w_1$ by $w'$ in $S$
            else
                replace $w_2$ by $w'$ in $S$
            end if
            if $S$ has no more constraint violations than $\hat{S}$ then
                $\hat{S} := S$;
            end if
        end for
    end for
    return $\hat{S}$
end StochasticLocalSearch for DNA word design
```

Figure 1: Outline of a general stochastic local search procedure for DNA word design.

using this scheme, in each iteration of the algorithm, exactly one base in one word is modified.

The parameter θ, also called the noise parameter, controls the greediness of the search process; for high values of θ, constraint violations are not resolved efficiently, while for low values of θ, the search has more difficulties to escape from local optima of the underlying search space.

Throughout the run of the algorithm, the best candidate solution encountered so far, i.e., the DNA word set with the fewest constraint violations, is memorised. Note that even if the algorithm terminates without finding a valid set of size k, a valid subset can always be obtained by iteratively selecting pairs of words that violate a Hamming distance constraint and removing one of the two words involved in that conflict from the set. Hence, a word set of size k with t constraint violations can always be reduced to a valid set of at least size $k - t$.

Hamming distances between words and/or their reverse complements are not recomputed in each iteration of the algorithm; instead, these are computed once after generating the initial set, and updated after each search step. This can be done very efficiently, since any modification of a single word can only affect the Hamming distances between this word and the $k - 1$ remaining words in the set.

The outer loop of our algorithm can perform multiple independent tries of the underlying search process. In conjunction with randomised search initialisation, multiple independent tries can yield better performance; essentially, this is the case if there is a risk for the search process to stagnate, i.e., to get stuck in a region of the underlying search space that it is very unlikely to escape from.

In this context, the parameter *maxSteps*, that controls the number of steps after which the search is aborted and possibly restarted from a new initial word set, can have an important impact on the performance of the algorithm; this will become apparent in our experimental results presented in Section 5.

The simple SLS algorithm presented above can easily be enhanced with a random replacement mechanism, which relies on partial replacements of randomly chosen code words at given time intervals. We fix a fraction fr of code words from the initial set and a number of steps *nsteps* which represents a threshold above which we consider the algorithm to be stagnated if no improvements can be observed during the search. When such a stagnation is observed, we replace $fr\%$ of the code words with new random generated ones and we continue with the simple SLS algorithm until next stagnation occurs, until we get a result or until we reach *maxSteps* iterations of the inner for loop in Figure 1.

Problem instance	**CPU time [sec] / search step**		
	k=100	**k=150**	**k=200**
HD constraint	.0016	.0030	.0036
HD,RC constraints	.0079	.0126	.0163
HD,RC,GC constraints	.0027	.0049	.0056
HD,GC constraints	.0006	.0010	.0013

Table 1: CPU times per search step for different constraint combinations and set sizes, $n = 8$, and $d = 4$, measured on a PC with 2 1GHz Pentium III CPUs, 512 MB cache and 1GB RAM running Red Hat Linux 7.1 (kernel 2.4.9). Note: Using the GC constraint reduces the complexity of search steps because of the smaller number of possible base substitutions.

5 Results

To evaluate the performance of our SLS algorithm (and its variants), we performed two types of computational experiments. Detailed analyses of the run-time distributions of our algorithm on individual problem instances were used to study the behavior of the algorithm and the impact of parameter settings. For these empirical analyses, the methodology of [17] for measuring and analyzing run-time distributions (RTDs) of Las Vegas algorithms was used. Run-time was measured in terms of search steps, and absolute CPU time per search step was measured to obtain a cost model of these search steps (see Table 1). The other type of experiment used the optimised parameter settings obtained from the detailed analyses for obtaining DNA word sets of maximal size for various word lengths and combinatorial constraints.

Introducing noise in our algorithmic approach, i.e. using probabilistic moves when taking decisions, provides robustness to our algorithm and allows it to escape from local minima encountered during search. Thorough experimentation (not reported here due to restricted space) shows that the setting of the noise parameter, θ, has a substantial effect on the performance of our algorithm: the time required for solving a given problem instance can easily vary more than an order of magnitude depending on the noise parameter setting used. Somewhat surprisingly, empirically optimal settings of θ are consistently close to 0.2 for different problem instances and sizes; consequently, this value was used for all experimental results reported here.

5.1 RTD Analysis

To study and characterize the behavior of our algorithm, we measured RTDs from 1000 runs of the algorithm applied to individual problem instances, using extremely high settings of the cutoff parameter, *maxSteps*, to ensure that a solution was found in each run without using random restarts. For each run, we recorded the number of search iterations (steps) required for finding a solution. From this data, the RTD gives the probability of success as a function of the number of search steps performed.

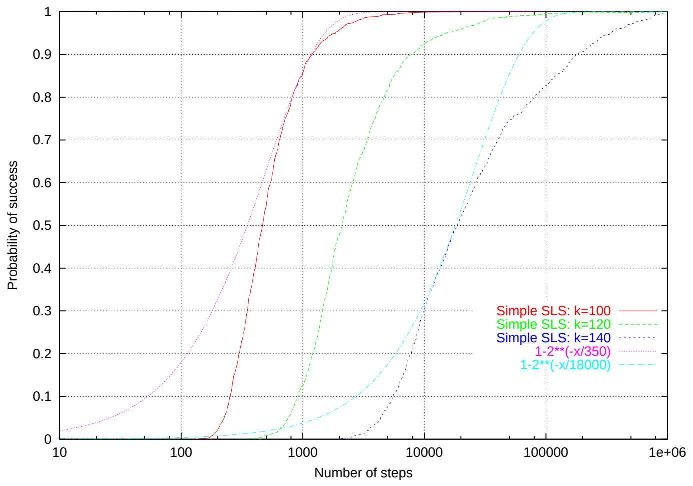

Figure 2: Run time distributions for the simple SLS algorithm applied to problem instances with the HD and GC constraints, word length $n = 8$, Hamming distance $d = 4$, and set sizes $k \in \{100, 120, 140\}$; two exponential distributions are shown to illustrate "fat" right tail of the RTDs indicating search stagnation for long run-times.

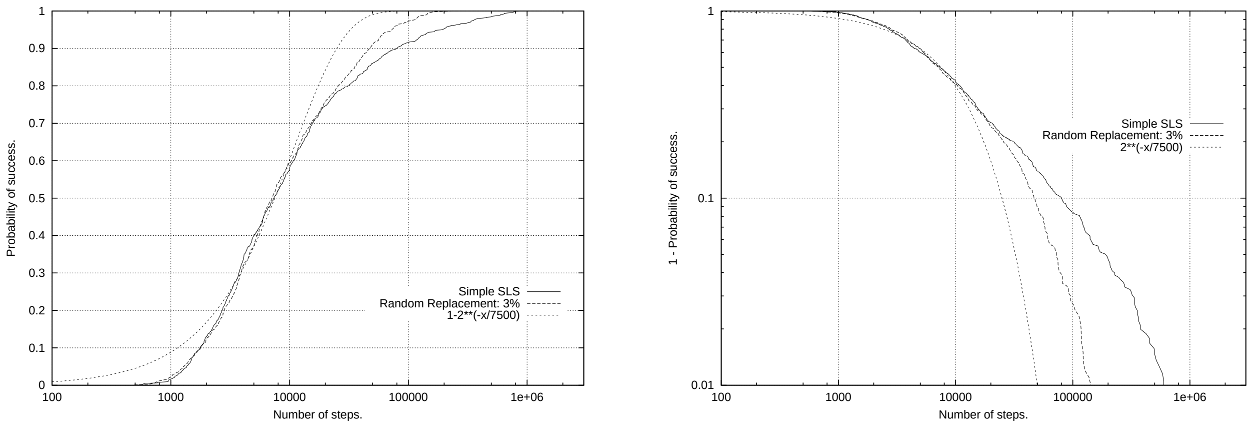

Figure 3: RTDs for SLS algorithm with and without random replacement applied to a problem instance with all three constraints, $n = 8$, $d = 4$, and $k = 70$. Right side (same data): the failure probability (1 - success probability) decays faster than polynomial but slower than exponential.

As can be seen in Figure 2, for given constraints, n, and d, the time required for obtaining a word set of size k with a fixed probability p increases with k for all p. Furthermore, for high p, this increase is much more dramatic than for low p, as can be seen in the "fat" right tails of the RTDs. This indicates that our simple SLS algorithm, even when using optimised noise settings, can suffer from stagnation behavior that compromises its performance for long run-times. The easiest way to overcome this stagnation effect is to restart the algorithm after a fixed number *maxSteps* of search steps. Optimal values of *maxSteps* can be easily determined from the empirical RTD data (see [17]). When solving a problem instance for the first time, however, this method is not applicable (unless the RTD can be estimated *a priori*), and the actual performance depends crucially on the value used for *maxSteps*.

It is therefore desirable to find different mechanisms for ameliorating or eliminating the observed search stagnation. The random replacement variant of our simple SLS algorithm described at the end of the previous section was designed to achieve this goal, and in many cases substantially reduces the stagnation effect (Figure 3 shows a typical example), leading to a significant improvement in the robustness and performance of our algorithm. However, as can be shown by comparing the empirical RTDs with appropriately fitted exponential distributions, even this modification does not completely eliminate the stagnation behavior.

5.2 Quality of Word Sets Obtained by our Algorithm

Using the improvements and parameter settings obtained from the detailed analysis described above, we used our SLS algorithm to compute word sets for a large number of DNA code design problems. The corresponding results can be summarised as follows (detailed results and best known theoretical bounds are tabulated in the Appendix):

HD constraint only. There is a significant body of work on the construction of word sets over an alphabet of size 4 that satisfy the Hamming distance constraint only, with the best bounds summarized by Bogdanova et al. [1]. We used our algorithm to design word sets in which the word lengths ranged from 4 to 12 with the minimum Hamming distance between pairs of words ranging from 3 to 11. Out of 43 tests, our algorithm was able to find a word set whose size matches the best known theoretical construction in 12 cases.

HD+RC constraints. In this case, we considered word lengths ranging from 4 to 12, and the minimum Hamming and reverse complement Hamming distances between pairs of words ranging from 2 to 11. Out of 57 tests, our algorithm was able to find a word set whose size matched or exceeded the theoretical results of Marathe et al. [19] in 56 cases.

HD+GC constraints. We obtained word sets satisfying the Hamming distance and the 50% GC content constraints for words length ranging from 4 up to 20, and the minimum Hamming distance between pair of words ranging from 2 to 10. Compared to the results reported by Corn et al. [18], we obtained bigger sets in almost all cases except two particular combinations, namely word length 8 and Hamming distance 4 and word length 12 and Hamming distance 6.

HD+RC+GC constraints. We obtained word sets for word lengths ranging from 4 to 12, and the minimum Hamming and reverse complement Hamming distances between pairs of words ranging in length from 2 to 11. There was no body of previous results that we could compare with except in one case, namely for words of length 8, GC content of 50%, that satisfy the Hamming and reverse-complement constraints with at least 4 mismatches between pairs of words. For this case, Frutos et al. [9] constructed a set of 108 words of length 8. We were not able to obtain any 108 DNA word sets satisfying these constraints, using our algorithm when initialized with a random initial set. The biggest set found had 92 code words. However, when we initialized our algorithm with the set of 108 words obtained by Frutos et al., along with one randomly chosen word, we were able to obtain a set of size 109. Continuing this incremental improvement procedure, we were able to construct sets containing 112 words in less than one day of CPU time. Interestingly, this set has the same general "template-map" structure as the set of Frutos et al.

6 Conclusions

We have presented a new stochastic local search algorithm for DNA word design, along with empirical results that characterize its performance and indicate its ability to find high-quality sets of DNA words satisfying various combinations of combinatorial constraints.

In future work, we will examine ways to further improve our algorithm. One possibility is to consider a more diverse neighbourhood instead of the rather small neighbourhood that we now explore (which is based on single base modifications). We conjecture that, particularly when the GC constraint is imposed, the current small neighbourhood structure limits the effectiveness of the algorithm. Another possibility is to consider more complex SLS strategies, which we expect to achieve improved performance that will likely lead to larger word sets.

In another direction of future work, we will test our methods on different word design constraints, including those based on thermodynamic models. While modified or additional constraints can be quite easily accommodated by relatively minor modifications of our current SLS algorithm and its variants, different algorithmic strategies might be more efficient for solving these word design problems.

Finally, it would be interesting to see if better theoretical design principles can be extracted from the word sets that we have now obtained empirically. In the construction of classical codes, theory and experiment are closely linked, and we expect that the same can be true for the construction of DNA codes.

References

[1] Galina T. Bogdanova, Andries E. Brouwer, Stoian N. Kapralov & Patric R. J. Ostergard, *Error-Correcting Codes over an Alphabet of Four Elements,* Andries Brouwer (aeb@cwi.nl)

[2] R.S. Braich, C. Johnson, P.W.K. Rothemund, D. Hwang, N. Chelyapov, and L.M. Adleman, "Solution of a satisfiability problem on a gel-based DNA computer," Preliminary Proc. Sixth International Meeting on DNA Based Computers, Leiden, The Netherlands, June, 2000.

[3] S. Brenner and R. A. Lerner, "Encoded combinatorial chemistry," Proc. Natl. Acad. Sci. USA, Vol 89, pages 5381-5383, June 1992.

[4] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr., "Good encodings for DNA-based solutions to combinatorial problems," Proc. DNA Based Computers II, DIMACS Workshop June 10-12, 1996, L. F. Landweber and E. B. Baum, Editors, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, pages 247-258.

[5] R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, and S. E. Stevens, Jr., "Genetic search of reliable encodings for DNA-based computation," Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors), Proceedings of the First Annual Conference on Genetic Programming 1996.

[6] A. A. El Gamal, L. A. Hemachandra, I. Shperling, and V. K. Wei, "Using simulated annealing to design good codes," IEEE Transactions on Information Theory, Vol. IT-33, No. 1, January 1987.

[7] Faulhammer, D., Cukras, A. R., Lipton, R.J., and L. F. Landweber, "Molecular computation: RNA solutions to chess problems," Proc. Natl. Acad. Sci. USA, 97: 1385-1389.

[8] U. Feldkamp, W. Banzhaf, H. Rauhe, "A DNA sequence compiler," Poster presented at the 6th International Meeting on DNA Based Computers, Leiden, June, 2000. See also http://ls11-www.cs.uni-dortmund.de/molcomp/Publications/publications.html (visited November 11, 2000).

[9] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn, "Demonstration of a word design strategy for DNA computing on surfaces," Nucleic Acids Research, Vol. 25, No. 23, December 1997, pages 4748-4757.

[10] M. Garzon, R. Deaton, P. Neathery, D. R. Franceschetti, and S. E. Stevens, Jr., "On the encoding problem for DNA computing," Preliminary Proc. 3rd DIMACS Workshop on DNA Based Computers, June 23-25, 1997, pages 230- 237.

[11] M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens Jr., and M. Wittner, "Encoding genomes for DNA computing," Proc. 3rd Genetic Programming Conference, Madison, WI, 1998.

[12] M. Garzon, R. J. Deaton, J. A. Rose, and D. R. Franceschetti, "Soft molecular computing," Preliminary Proc. Fifth International Meeting on DNA Based Computers, June 14-15, MIT, 1999, pages 89-98.

[13] A.J. Hartemink, D.K. Gifford, and J. Khodor, "Automated constraint-based nucleotide sequence selection for DNA computation," 4th Annual DIMACS Workshop on DNA-Based Computers, Philadelphia, Pennsylvania, June 1998

[14] I.S. Honkala, and P.R.J. Ostergard, "Code design," In Local Search In Combinatorial Optimization (E. Aarts and J.K. Lenstra, eds.), Wiley-Interscience Series in Discrete Mathematics and Optimization, 1997.

[15] Holger H. Hoos, *Stochastic Local Search - Methods, Models, Applications,* infix-Verlag, Sankt Augustin, Germany, ISBN 3-89601-215-0, 1999.

[16] H.H. Hoos and C. Boutilier, "Solving combinatorial auctions using stochastic local search," In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 22–29, Austin, TX, 2000.

[17] H.H. Hoos and T. Stützle, "Evaluating Las Vegas Algorithms — Pitfalls and Remedies," In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 1998, pages 238-245.

[18] M. Li, H-J. Lee, A. E. Condon, and R. M. Corn, "DNA Word Design Strategy for Creating Sets of Non-interacting Oligonucleotides for DNA Microarrays," Langmuir, 18, 2002, pages 805-812.

[19] A. Marathe, A. Condon, and R. Corn, "On combinatorial DNA word design," J. Computational Biology, 8:3, 2001, 201-220.

[20] D. McAllester, H. Kautz, and B. Selman, "Evidence for invariants in local search," *Proc. AAAI-97*, 321–326, Providence, RI, 1997.

[21] The PERMUTE program. Available at the PNAS web site
http://www.pnas.org/cgi/content/full/97/4/1385/DC1. Visited November 22, 2000.

[22] Programmable DNA web site,
http://ls11-www.cs.uni-dortmund.de/molcomp/Downloads/downloads.html. Visited November 11, 2000.

[23] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya, "Molecular computation by DNA hairpin formation," Science, Vol. 288, May 2000, pages 1223- 1226.

[24] H. Yoshida and A. Suyama, "Solution to 3-SAT by breadth first search," Proc. 5th Intl. Meeting on DNA Based Computers, M.I.T. 1999, pages 9-20.

[25] B-T. Zhang and S-Y. Shin, "Molecular algorithms for efficient and reliable DNA computing," Proc. 3rd Annual Genetic Programming Conference, Edited by J. R. Koza, K. Deb, M. Doringo, D.B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Morgan Kaufmann, 1998, pages 735-742.

[26] B-T. Zhang and S-Y. Shin, "Code optimization for DNA computing of maximal cliques," Advances in Soft Computing - Engineering Design and Manufacturing, Springer-Verlag, 1998.

Appendix

In this appendix, we report the sizes of word sets obtained by the SLS algorithm presented in this paper. This data illustrates in detail the performance of our algorithm and will be useful as a baseline for future work. The actual word sets will be made publicly available from our webpage [URL will be added in final version]. For comparison, we also report best previously known bounds on the word set sizes.

The data on our word sets is grouped as follows:

- Table 2 shows the results for the **HD** constraint only;
- Table 4 shows the results for the **HD+RC** constraints;
- Table 6 shows the results for the **HD+GC** constraints;
- Table 8 shows the results for the **HD+RC+GC** constraints.

In these tables, entries in **bold face** match or exceed the best previously known lower bounds. All other values are smaller than known lower bounds for corresponding (n, d) values. Cells in the tables marked with X correspond to specific (n, d) combinations for which the implementation of our algorithm ran out of working memory. Dashes ("–") indicate cases of limited or no interest for DNA word design, for which we did not run our algorithm.

Along with each maximal word set size found for our algorithm, we also report the corresponding median number of steps spent by our algorithm in order to find that set (in square brackets, [...], typically specified in thousands of search steps). Each table entry is based on between 5 and 20 runs of our algorithm; the precise number of runs was varied with problem size and solution quality; median run times are taken over successful runs only.

Tables 3 and 5 contain the best previously known theoretical lower and upper bounds on sizes of word sets. In cases, where lower and upper bounds are identical, only a single value is shown.

$n \setminus d$	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
4	**16** [$.5k$]	**4** [$.01k$]	-	-	-	-	-	-	-
5	**64** [$7k$]	**16** [$1.5k$]	**4** [$.02k$]	-	-	-	-	-	-
6	116 [$30k$]	**64** [$80k$]	**9** [$15k$]	**4** [$.02k$]	-	-	-	-	-
7	366 [$300k$]	78 [$70k$]	23 [$75k$]	**8** [$.5k$]	**4** [$.04k$]	-	-	-	-
8	1164 [$65k$]	219 [$65k$]	55 [$40k$]	19 [$50k$]	**5** [$.03k$]	**4** [$.09k$]	-	-	-
9	3800 [$440k$]	616 [$285k$]	138 [$65k$]	41 [$50k$]	15 [$50k$]	**5** [$.1k$]	**4** [$.12k$]	-	-
10	>10000 [$37k$]	1783 [$471k$]	358 [$312k$]	95 [$130k$]	32 [$75k$]	12 [$5k$]	**5** [$.5k$]	**4** [$.32k$]	-
11	X	5598 [$523k$]	967 [$592k$]	227 [$165k + .1k$]	70 [$36k$]	27 [$545k$]	10 [$15k$]	**4** [$.05k$]	**4** [$.8k$]
12	X	>10000 [$24k$]	2689 [$263k$]	578 [$300k$]	156 [$735k$]	53 [$390k$]	23 [$532k + .5k$]	**9** [$.5k + 5k$]	**4** [$.1k$]

Table 2: Empirical bounds on **HD** quaternary codes.

$n \setminus d$	3	4	5	6	7	8	9	10	11
4	16	4	-	-	-	-	-	-	-
5	64	16	4	-	-	-	-	-	-
6	164 - 179	64	9	4	-	-	-	-	-
7	512 - 614	128 - 179	32	8	4	-	-	-	-
8	2048 - 2340	320 - 614	70 - 128	32	5	4	-	-	-
9	8192 - 9360	1024 - 2340	256 - 512	64 - 128	18 - 20	5	4	-	-
10	17408 - 30427	4096 - 9360	1024 - 2048	256 - 512	40 - 80	16	5	4	-
11	65536 - 109226	16384 - 30427	4096 - 6241	1024 - 2048	92 - 320	32 -64	12	4	4
12	262144 - 419430	65536 - 109226	8192 - 20852	4096 - 6241	256 - 1280	64 - 242	24 - 48	9	4

Table 3: Best previously known bounds on the size of codes that satisfy the **HD** constraint as a function of word length n and Hamming distance d [1].

$n \setminus d$	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
4	**32** [.3k]	**6** [.1k]	**2** [.02k]	-	-	-	-	-	-	-
5	**115** [51k]	**32** [4k]	**4** [.02k]	**2** [.005k]	-	-	-	-	-	-
6	**502** [30k]	**59** [160k]	**28** [28k]	**4** [.1k]	**2** [.02k]	-	-	-	-	-
7	**1968** [45k]	**180** [10k]	**40** [254k]	**11** [3k]	**2** [.005k]	**2** [.05k]	-	-	-	-
8	**8074** [135k]	**589** [247k]	112 [850k]	**27** [7k]	**10** [31k]	**2** [.01k]	**2** [.03k]	-	-	-
9	X	**1898** [794k]	**314** [677k]	**72** [142k]	**20** [37k]	**8** [16k]	**2** [.02k]	**2** [.04k]	-	-
10	X	**6083** [108k]	**938** [702k]	**180** [386k]	**49** [287k]	**16** [495k]	**8** [11.01k]	**2** [.01k]	**2** [.02k]	-
11	X	8000	**2750** [117k]	**488** [257k]	**114** [145k]	**35** [6k]	**12** [1k]	**5** [2k]	**2** [.05k]	**2** [.01k]
12	X	X	>**8000** [72k]	**1340** [400k]	**290** [327k]	**79** [236k]	**27** [712.5k]	**11** [828k]	**4** [.2k]	**2** [.05k]

Table 4: Empirical bounds on **(HD,RC)** quaternary codes.

$n \setminus d$	2	3	4	5	6	7	8
2	2	-	-	-	-	-	-
3	$0 - 2^3$	$0 - 2^3$	-	-	-	-	-
4	2^5	$2 - 8$	2	-	-	-	-
5	$4 - 2^7$	$2 - 2^7$	$0 - 26$	$0 - 8$	-	-	-
6	2^9	$16 - 107$	$8 - 107$	$2 - 5$	2	-	-
7	$33 - 2^{11}$	$16 - 2^{11}$	$2 - 372$	$2 - 372$	$0 - 34$	$0 - 4$	-
8	2^{13}	$160 - 1310$	$128 - 1310$	$8 - 118$	$8 - 118$	$2 - 3$	2
9	$354 - 2^{15}$	$160 - 2^{15}$	$8 - 4681$	$8 - 4681$	$2 - 372$	$2 - 372$	$2 - 14$
10	2^{17}	$1184 - 16912$	$320 - 16912$	$32 - 1202$	$32 - 1202$	$8 - 142$	$8 - 8$
11	$3903 - 2^{19}$	$576 - 2^{19}$	$60 - 61680$	$32 - 61680$	$8 - 3964$	$8 - 3964$	$8 - 420$
12	2^{21}	$13233 - 226719$	$2304 - 226719$	$173 - 13294$	$96 - 13294$	$8 - 1276$	$8 - 1276$
13	$45012 - 2^{23}$	$4096 - 2^{23}$	$487 - 838860$	$128 - 838860$	$18 - 45221$	$8 - 45221$	$8 - 3964$
14	2^{25}	$155496 - 3121342$	$12539 - 3121342$	$1444 - 155705$	$512 - 155705$	$64 - 12555$	$32 - 12555$
15	$541020 - 2^{27}$	$40385 - 2^{27}$	$4280 - 11671106$	$1024 - 11671106$	$128 - 541746$	$64 - 541746$	$64 - 40439$
16	2^{29}	$1901386 - 43826196$	$132111 - 43826196$	$13066 - 1902111$	$4096 - 1902111$	$576 - 132160$	$512 - 132160$

Table 5: Best known bounds on the size of codes that satisfy the **(HD,RC)** constraints as a function of word length n and Hamming distance d [19].

n/d	2	3	4	5	6	7	8	9	10
4	**48** [.3k]	[]	[]	-	-	-	-	-	-
5	**142** [93k]	[]	[]	-	-	-	-	-	-
6	[]	**85** [860k]	[]	-	-	-	-	-	-
7	[]	**230** [650k]	[]	-	-	-	-	-	-
8	[]	[]	209 []	-	-	-	-	-	-
9	[]	[]	**400** [72k]	-	-	-	-	-	-
10	[]	[]	[]	**256** [250k]	-	-	-	-	-
11	[]	[]	[]	**620** [48k]	-	-	-	-	-
12	[]	[]	[]	-	410 [245k]	-	-	-	-
13	[]	[]	[]	-	**990** [36k]	-	-	-	-
14	[]	[]	[]	-	-	**500** [5k]	-	-	-
15	[]	[]	[]	-	-	**1571** [1000k]	-	-	-
16	[]	[]	[]	-	-	-	**966** [242k]	-	-
17	[]	[]	[]	-	-	-	**2355** [745k]	-	-
18	[]	[]	[]	-	-	-	-	**1200** [15k]	-
19	[]	[]	[]	-	-	-	-	**3451** [396k]	-
20	[]	[]	[]	-	-	-	-	-	**2193** [596k]

Table 6: Empirical bounds on **(HD,GC)** quaternary codes.

n/d	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
4	48	-	-	-	-	-	-	-	-
5	120	-	-	-	-	-	-	-	-
6	-	56	-	-	-	-	-	-	-
7	-	224	-	-	-	-	-	-	-
8	-	-	224	-	-	-	-	-	-
9	-	-	360	-	-	-	-	-	-
10	-	-	-	132	-	-	-	-	-
11	-	-	-	528	-	-	-	-	-
12	-	-	-	-	528	-	-	-	-
13	-	-	-	-	728	-	-	-	-
14	-	-	-	-	-	240	-	-	-
15	-	-	-	-	-	960	-	-	-
16	-	-	-	-	-	-	960	-	-
17	-	-	-	-	-	-	1224	-	-
18	-	-	-	-	-	-	-	380	-
19	-	-	-	-	-	-	-	1520	-
20	-	-	-	-	-	-	-	-	1520

Table 7: Best known bounds on the size of codes that satisfy the **(HD,GC)** constraints as a function of word length n and Hamming distance d [18].

n/d	2	3	4	5	6	7	8	9	10	11
4	20 [.03k]	5 [.02k]	2 [.001k]	-	-	-	-	-	-	-
6	282 [1k]	37 [19k]	11 [3k]	2 [.003k]	2 [.006k]	-	-	-	-	-
8	3981 [20k]	350 [119k]	92* [4000k]	19 [1.2k]	7 [10k]	2 [.01k]	2 [.02k]	-	-	-
10	X	3700 [287k]	640 [406k]	127 [170.5k]	37 [38k]	11 [134k]	5 [1.3k]	2 [.02]	1 [.005k]	-
12	X	X	5685 [455k]	933 [531k]	210 [121.5k]	59 [77k]	21 [217k]	9 [341.5k]	3 [1.5k]	2 [.07k]

Table 8: Empirical bounds on **(HD,RC,GC)** quaternary codes. For the $\star$'d entry we found a better bound, namely 112 code words as described in Section 5.2.