Choosing the Right Neighbourhood: a Way to Improve a Stochastic Local Search Algorithm for DNA Word Design

Dan C Tulpan and Holger H Hoos

Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
{dctulpan,hoos}@cs.ubc.ca
WWW home page: http://www.cs.ubc.ca/labs/beta

Abstract. We present results on increasing 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 and in turn improve the performance of the SLS algorithm. We report several cases in which our algorithm finds word sets that match or exceed the best previously known constructions.

Keywords: DNA word design, stochastic local search

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]. 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 [16] 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).

Our previous work [15] presents results on the performance of a new stochastic local search algorithm for the design of DNA codes fulfilling different combinations of combinatorial constraints, the same as the ones described in Section 2. We presented empirical results that characterize the SLS algorithm performance and indicate its ability to find high-quality sets of DNA words

Towards this end, we describe new improvements added to the simple stochastic local search algorithm for design of DNA codes described in [15].We analyze its performance using an empirical methodology based on run-time and run-length distributions [13]. In this study, we have chosen to design word sets that fullfil all 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.

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 modification step is based on different neighbourhood generation mechanisms which will be further described in the paper. 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 was primarily controlled by a so-called noise parameter that determines the probability of greedy vs. random conflict resolution. Optimal settings for this parameter have been reported in [15] and we'll show how this settings will be affected by choosing a different neighbourhood generation mechanism.

Our empirical results, reported in Section 5, show that the run-time distributions that characterize our algorithm's performance on hard word design problems improved dramatically compared to those for Simple SLS algorithm. Empirical results reported in the same section, show that the run-time distributions that characterize the enhanced algorithm's performance get better by eliminating fat right tails for set sizes equal to the one studied in [15].

We compared the sizes of the word sets obtainable by our algorithm with previously known word sets, starting with the previous studied case of word sets that satisfy all three constraints. Out of a total of 31 comparisons with previous results (see Tables 2, 3), we found word sets that equal or improved on previous constructions in all but one case. In this particular case, while our algorithm was not able to meet the previous best construction when starting from a random initial set of words, we were still 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 a table 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. One class of such methods are stochastic local search algorithms and they have been used with success for many years in code design as well as in other combinatorics areas [12].

The combinatorial problem considered in this paper, namely the DNA Word Design is slightly different from the classical and well-studied combinatorial problems like SAT, TSP and CSP. The difference is mainly based on the way the constraints are defined. Each constraint involves code words having the same fixed length and all code words form a huge set. Nevertheless, when trying to solve the problem, we search not only through the set of all code words by picking and analyzing a fairly small subset of them (also called neighbourhood), but we search through the set of all possible sets of codes having the same size (cardinality), each single set containing a fixed number of distinct code words that must fulfill the aforementioned constraints. When we come to the point to evaluate the quality of the set of code words 'hunted' by our SLS algorithm, we count the number of 'broken' constraints along the set and we continue the search in a probabilistically informed way.

3 Related Work

Stochastic search methods have been used successfully for decades in the construction of good binary codes (see for example [6, 11]). 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 [16] 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. [10] 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 [14, 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] for their RNA-based 10-variable computation.

As we know by now, many optimization problems of practical interest are computationally intractable. One practical approach to solve these problems is to use neighbourhood search algorithms where at each iteration an improving solution is found by searching the "neighbourhood" of the current solution. A survey on very large scale neighbourhoods and corresponding search techniques is presented in [1]. They study neighbourhood search algorithms in which the size of the neighbourhood is 'very large' with respect to the input data.

4 The Improved Stochastic Local Search Algorithm

The basic stochastic search algorithm, which is subject to further improvement and development, performs local search in a space of code word sets of fixed size which violate the given constraints. Figure 1 contains the outline of the simple SLS algorithm as described in [15].

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 modification process consists in replacing one of the code words involved in a conflict with a new code word chosen from a pool of new code words representing a neighbourhood. Here, we consider different types of neighbourhood generation mechanisms, which provide improvements of the existing SLS algorithm performance. 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.

Next follows the basic neighbourhoods types considered in this paper, together with explanations and rationale behind them. Before presenting the neighbourhoods we mention that all neighbourhoods contain code words that fulfill the GC-content constraint.

k-mutation neighbourhood A k-mutation neighbourhood is obtained by performing successive 1-point mutations on a certain code word. Let w_0 = CCCCAAAA be one code word of length 8. A one-point mutation of code word w_0 consists in generating a new code word w_1, which differ from the initial word w_0 in one base (Eg. w_1 = CCCCATAA). By applying a one-point mutation on a pair of code words of length n, someone can generate $2 \times n$ new distinct code words fulfilling the **GC** constraint.

```
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
            $M$ := all neighbouring words corresponding to $w_1$ and $w_2$
            with probability $\theta$ do
                select word $w'$ from $M$ at random
            otherwise
                select word $w'$ from $M$ such that
                number of conflict violations is maximally decreased
            end with probability
            if $w' \in M$ 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
```

Fig. 1. Outline of the stochastic local search procedure for DNA word design.

A 2-mutation neighbourhood can be defined in the same way as the 1-mutation neighbourhood. Each of the $2 \times n$ component code words of the 1-mutation neighbourhood can be in turn mutated in one extra position at a time, by taking care of non-duplicating the existing neighbouring code words, i.e. avoiding two 1-point mutations of the same position. For example w_2 = CCCCAATT represents a 2-point mutation of code word w_0.

By extending the 2-mutation neighbourhood with one extra 1-point mutation operation we can get a 3-mutation neighbourhood. Code word w_3 = CCCCATTT represents a 3-point mutation of code word w_0. The size of the 3-mutation neighbourhood is given by: $size = n \times (n \times n + 5)/3$ (Eg. $n = 8 \Rightarrow$ Set $size = 184$). This neighbourhood will also include code words obtained by performing 1,2 and 3-point mutations on the initial ones.

Random neighbourhood Another simple way of chosing neighbourhoods is by generating a fixed number of random code words having the same length n and fixed GC-content (50% in this paper). This rather simplistic neighbourhood mechanism offered a real breakthrough for boosting up the performance of the algorithm, as we can see in the Results section.

***k*-mutation + random neighbourhood** Here we have 3 new types of neighbourhoods obtained by adding extra random code words to the previously defined k-mutation neighbourhoods. By adding randomly generated code words to k-mutation neighbourhoods we add more diversification to the search. The algorithm can explore now regions of the search space that couldn't be reached easily using mutation-based mechanisms and it has a bigger chance to exit from local minima regions and eventually find solutions faster. The size of these neighbourhoods can be computed by adding a constant number of random code words to the original size of the k-mutation neighbourhoods. This extra randomly chosen code words have a big impact on algorithm time-related performance and especially on attaining high level solutions, i.e. bigger sets of code words that fulfill the desired constraints.

The proposed neighbourhood mechanisms can be classified in two main groups: first group is composed by the first 2 neighbourhoods and represents simple neighbourhood approaches; second group is composed from the other neighbourhood mechanism (namely k-mutation + random neighbourhood) and represents combinations of items belonging to the first group. The use of bigger neighbourhoods is mostly motivated by previous observations related with search stagnation avoidance and pushing the algorithm towards bigger set sizes.

In the next section we will discuss the efficiency of these neighbourhoods and their impact on improving algorithm performance.

5 Results and Discusion

To evaluate the performance of our improved SLS algorithm, we performed two types of computational experiments. Detailed analyses of the run-time and run-length 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 [13] for measuring and analyzing run-time distributions (RTDs) and run-length distributions (RLDs) 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.

5.1 Noise parameter

Introducing noise in the simple SLS algorithm, i.e. using probabilistic moves when taking decisions, provides robustness to the algorithm and allows it to

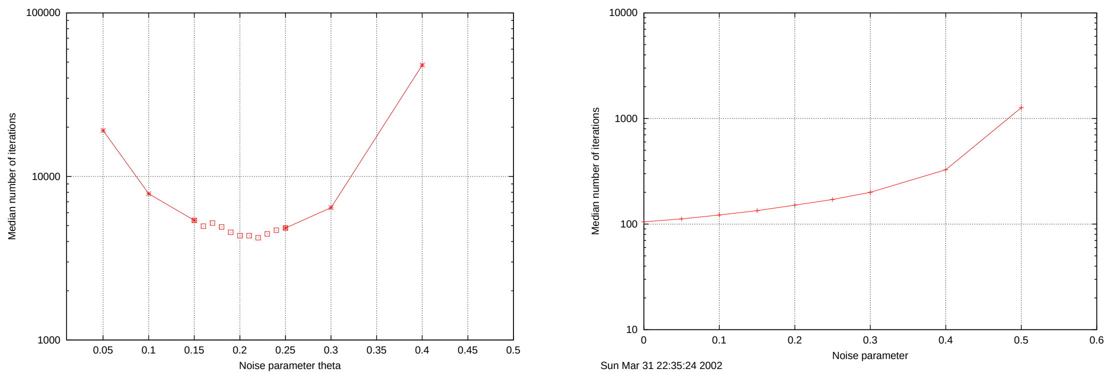

Fig. 2. Number of iterations as a function of noise parameter values: all three constraints, $n = 8$, $d = 4$, and $k = 70$. Left side: 1-mutation neighbourhood; Right side: 1-mutation + random neighbourhood.

escape from local minima. Using the previous 1-mutation neighbourhood, we found an optimal setting for this parameter somewhere close to 0.2 for different problem instances and sizes, as described in [15]. When considering random-based neighbourhoods, the optimal value for the noise parameter seems to be shifted towards 0 as can be seen in Figure 2.

One possible explanation for this phenomenon may reside in the extra need of greediness for the search algorithm when searching bigger neighbourhoods. The extra diversification property obtained by using random-based neighbourhoods can actually compensate and even substitute the existent noise leverage in terms of probabilistically accepting worsening steps for non-zero noise parameters.

5.2 RTD and RLD Analysis

To study and characterize the behavior of the new proposed neighbourhood mechanisms for the simple SLS algorithm, we measured RTDs and RLDs from 1000 successful runs of the algorithm applied to the proposed problem instance, namely DNA Word Design for code words of length $n = 8$, hamming distance $d = 4$ and all 3 constraints (**HD, RC, GC**). Using extremely high settings of the cutoff parameter ensured us that a solution was found in each individual run without using random restarts. For each run corresponding to different neighbourhood settings we collected the number of search iterations required for finding a solution. From this data, the RLD gives the probability of success as a function of the number of search iterations performed. We obtain also RTDs by multiplying the number of iterations with the corresponding CPU cost per step.

Figure 3 shows RLDs for different types of neighbourhoods for the problem instance considered above. The time required for obtaining a set of words of size $k = 70$ with a fixed probability p increases with k and p. For high p, this increase is much more dramatic than for low p values especially for k-mutation neighbourhoods. The right 'fat' tails of the RLDs emphasize this phenomenon. As we

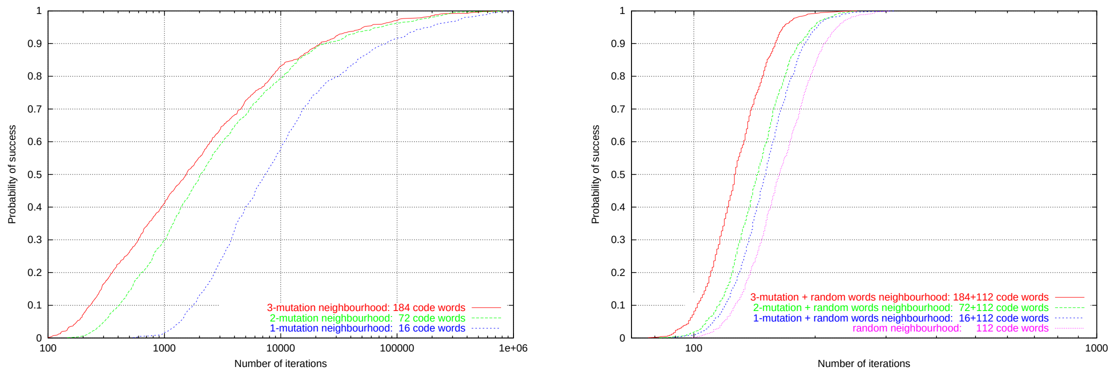

Fig. 3. RLDs for different neighbourhoods (# of iterations), set size $k = 70$, word length $n = 8$, hamming distance $d = 4$, all 3 constraints. Left side: k-mutation neighbourhoods; Right side: k-mutation + random code words neighbourhood.

can easily notice, the RLDs for mixed and more complex neighbourhoods are more shifted to the left and straighter. This means that the algorithm spends less iterations for attaining the same solution quality, than when using k-mutation neighbourhoods. We observe also the tendency of elimination of fat right tails for randomly-based neighbourhoods. The median number of iterations required to find solutions decreases by almost two orders of magnitude for the randomly generated neighbourhoods as compared to the simple k-mutation neighbourhoods.

One way to explain the observed improvement in the run-time of the algorithm is based on the increased diversification of the search, by considering bigger and more diverse neighbourhoods.

There is one issue that we have to consider when using neighbourhoods. The larger the neighbourhood, the better is the quality of the locally optimal solutions, and the greater is the accuracy of the final solution that is obtained. At the same time, the larger the neighbourhood, the longer it takes to search the neighbourhood at each iteration. Figure 4 shows improvement in the algorithm run-time speed with almost three orders of magnitude for the worst situation (higher percentiles), between the small 1-mutation neighbourhood and larger randomized neighbourhoods.

Increasing the number of random words added to k-mutation neighbourhoods produces a decrease in the number of iterations spent by the algorithm to attain a certain solution quality but it increases the CPU time for each iteration. Increasing the same factor, we obtain an increase of the chances for the SLS algorithm to reach word sets of bigger size.

5.3 Scaling

Increasing the size of the code word sets for the same parameter settings, will increase the difficulty of the given problem instance. In other words, it is much harder to find solutions for bigger problems than for smaller ones.

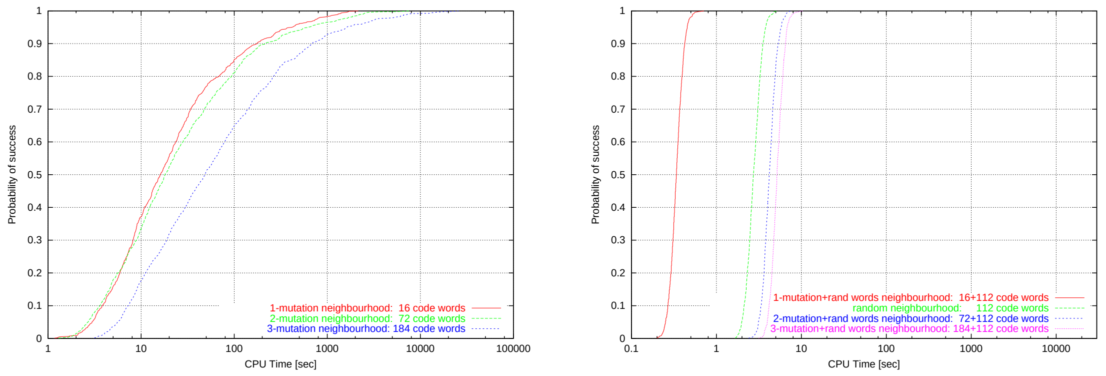

Fig. 4. RTDs for different neighbourhoods (# of CPU seconds), set size $k = 70$, word length $n = 8$, hamming distance $d = 4$, all 3 constraints. Left side: k-mutation neighbourhoods; Right side: k-mutation + random code words neighbourhood.

Neighbourhood	**CPU Time**	**Neighbourhood**	**CPU Time**
1-mutation	*.002184*	**1-mutation+rand-1**	*.002247*
2-mutation	*.008830*	**1-mutation+rand-2**	*.002340*
3-mutation	*.031493*	**1-mutation+rand-4**	*.003898*
random 16	*.002533*	**1-mutation+rand-8**	*.004040*
random 72	*.007544*	**1-mutation+rand-16**	*.007846*
random 114	*.016750*	**1-mutation+rand-32**	*.008864*
random 128	*.015100*	**1-mutation+rand-64**	*.019701*
random 500	*.065556*	**1-mutation+rand-112**	*.022889*
1-ex + rand 128	*.016471*	**2-mutation+rand-112**	*.029167*
		3-mutation+rand-112	*.040833*

Table 1. CPU Times for set size n = 70, noise=0.2, fixed random seed and different neighbourhoods. CPU times have been measured on a PC with 2 1GHz Pentium III CPUs, 512 Mb cache and 1GB RAM running Red Hat Linux 7.2 (kernel 2.4.9-6smp). The unit of measurement is CPU seconds per algorithm iteration.

Figure 5 shows comparisons between 3 neighbourhood types: 1-mutation (16 code words), pure random neighbourhood of size 30 and 1-mutation plus 14 random code words. Using the last two neighbourhoods, we obtained an improvement in the median number of iterations for different target set sizes as compared to the first type.

The median CPU time was also improved. We also observed (no results are shown here) that, by increasing the size of neighbourhoods, the differences in number of iterations and CPU time for small set sizes decrease. We also conjecture that by considering 1-mutation plus random words and pure random neighbourhoods of fairly big and equal sizes, the difference in performance for the proposed SLS algorithm will decrease dramatically.

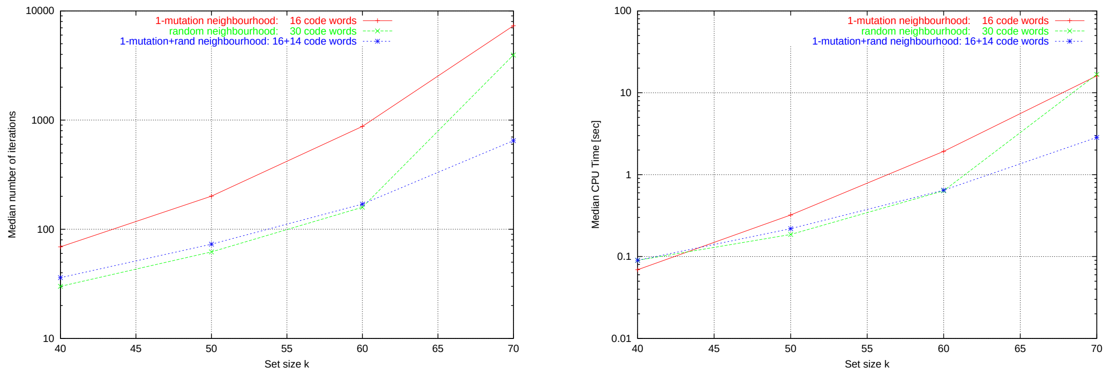

Fig. 5. Median number of iterations and CPU seconds for different set sizes and different neighbourhoods, word length $n = 8$, hamming distance $d = 4$, all 3 constraints. Left side: Number of iterations; Right side: Number of CPU seconds.

5.4 The Importance of having 1-mutation Neighbourhood

We proved that adding random code words to k-mutation neighbourhoods provides improved effectiveness in searching the space of sets. This raises the following question: is there any reason to keep the k-mutation neighbourhood as part of the bigger neighbourhood?

We performed two types of experiments. They confirm that having the simple k-mutation neighbourhood is important. In our first experiment we kept fixed the neighbourhood size to 30 code words and we vary the number of code words corresponding to k-mutation neighbourhood and random neighbourhood. When using one randomly chosen code word from the k-mutation neighbourhood and the rest from the random neighbourhood, the number of iterations spent by the algorithm to find solutions increased dramatically (see left picture from Figure 6).

To answer the question stated above we performed the second experiment and we increased the size of the neighbourhood to 100 code words. We were able to observe the same phenomenon but only for bigger set sizes (80 code words). We conjecture that adding more random code words is benefic for the algorithm but keeping the 1-mutation neighbourhood is a key factor for the problem considered in this paper. One possible explanation for the observed phenomenon could be that 1-mutation neighbourhood code words can be easily 'mutated' back into the original word by flipping the same base again. This mechanism offers the algorithm a easy and cheap way 'to repair the mistakes' that have been done in the previous iterations by accepting the wrong code word into the target set of k code words.

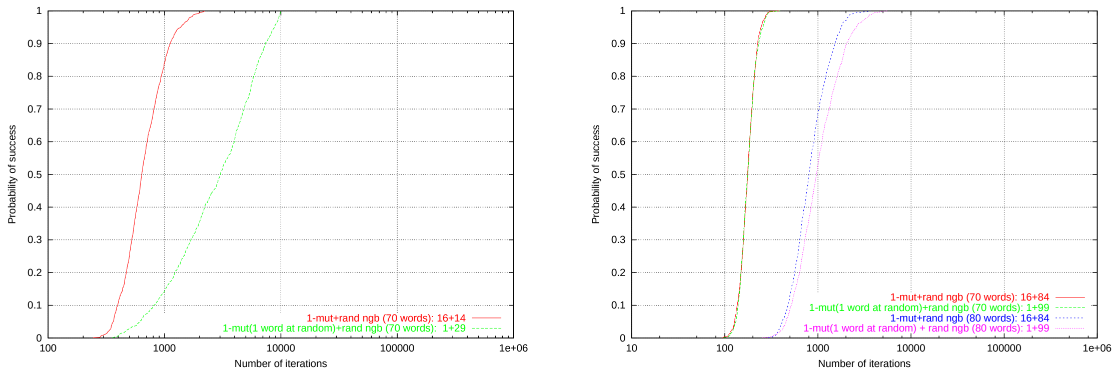

Fig. 6. RLDs for 1-mutation + random code words neighbourhoods (# of iterations), word length $n = 8$, hamming distance $d = 4$, all 3 constraints. Left side: neighbourhood size: 30 code words; Right side: neighbourhood size: 100 code words, different set sizes (70 and 80 code words).

5.5 Quality of Word Sets Obtained by our Algorithm

Using the new improvement strategies based on bigger and more randomized neighbourhoods, we used the enhanced SLS algorithm to generate word sets for the proposed problem instance.

We obtained 107 DNA code words of length 8 satisfying the Hamming and reverse-complement constraints with at least 4 mismatches between pairs of words, GC content 50%, and using a random initial set of code words. For the same case, Frutos et. al. [9] constructed a set of 108 words of length 8 and we reported in [15] sets of 92 code words obtained by initialising the simple SLS algorithm with random sets and sets of 112 code words obtained by initialising the same algorithm with the best known set containing 108 code words plus one extra code word at a time.

For the problem instance considered in this paper, namely $n = 8$, $d = 4$ and all constraints satisfied we compared the sizes of the word sets obtainable by our algorithm with previously known word sets. Out of a total of 31 comparisons with previous results (see Tables 2, 3), we found word sets that equal or improved on previous constructions in all but one case. In this particular case, while our algorithm was not able to meet the previous best construction when starting from a random initial set of words, we were still 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 have also evidence that better results can be obtained for problem instances fulfilling combinations of constraints containing fixed GC content (Eg. **HD+GC**) and further results and development will be presented in future work.

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 2. Empirical bounds on **(HD,RC,GC)** quaternary codes obtained with the simple SLS algorithm (Results are reported in [15]).

n/d	2	3	4	5	6	7	8	9	10	11
4	**24** [.03k]	**6** [.01k]	2 [.001k]	-	-	-	-	-	-	-
6	**310** [.7k]	**41** [1.5k]	**15** [.3k]	**4** [.005k]	2 [.002k]	-	-	-	-	-
8	**4022** [16.7k]	**390** [3.4k]	**107*** [40k]	**26** [64k]	**12** [4.8k]	2 [.002k]	2 [.002k]	-	-	-
10	X	**4007** [25.7k]	**790** [9.4k]	**158** [2k]	**41** [1.2k]	**15** [.6k]	**6** [2.6k]	2 [.002]	**2** [.002k]	-
12	X	X	**6100** [256k]	**988** [23.3k]	**240** [3.1k]	**70** [1.7k]	**25** [1.2k]	9 [3.2k]	**4** [.2k]	2 [.07k]

Table 3. Empirical bounds on **(HD,RC,GC)** quaternary codes obtained with the improved SLS Algorithm. 1-mutation+random code words neighbourhoods have been used. The number of random code words used here are $\{10, 100, 1000, 5000\}$. For $n = 8$, $d = 4$ we found a better bound, namely 112 code words by initializing our algorithm with the best previously known word set (108 code words) plus an additional random word..

6 Conclusions

We presented new improvements that can be done for the simple SLS algorithm proposed in [15], based on new neighbourhood generation mechanisms, along with empirical results that characterize its performance. New insights on neighbourhood mechanisms have been described and we showed evidence that randomizing neighbourhoods provides improved performance that lead the SLS algorithm to larger word sets.

In future work, we plan to examine further ways for improving the algorithm. One possibility is to consider more complex SLS strategies, which are expected to achieve improved performance that hopefully will lead to larger word sets.

In another direction of future work, we plan to expand the neighbourhood mechanisms to work on different constraint combinations.

Finally, it would be interesting to see if better theoretical design principles can be extracted from the word sets that have been empirically obtained. Search space analysis may provide more insight on the hidden mechanisms that make problems difficult to be solved efficiently.

References

1. Ravindra K. Ahuja, Ozlem Ergun, James B. Orlin, “A Survey of Very Large Scale Neighborhood Search Techniques“, Discrete Applied Mathematics, July 22, 1999
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. 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
11. 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.
12. Holger H. Hoos, *Stochastic Local Search - Methods, Models, Applications,* infix-Verlag, Sankt Augustin, Germany, ISBN 3-89601-215-0, 1999.
13. 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.
14. Programmable DNA web site, http://ls11-www.cs.uni-dortmund.de/molcomp/Downloads/downloads.html. Visited November 11, 2000.
15. Dan Tulpan, Holger Hoos, Anne Condon, “Stochastic Local Search Algorithms for DNA Word Design“, DNA 8 Conference, Japan, March 2002

16. 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.