Hybrid Randomised Neighbourhoods Improve Stochastic Local Search for DNA Code 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
http://www.cs.ubc.ca/labs/beta

Abstract. Sets of DNA strands that satisfy combinatorial constraints play an important role in various approaches to biomolecular computation, nanostructure design, and molecular tagging. The problem of designing such sets of DNA strands, also known as the DNA code design problem, appears to be computationally hard. In this paper, we show how a recently proposed stochastic local search algorithm for DNA code design can be improved by using hybrid, randomised neighbourhoods. This new type of neighbourhood structure equally supports small changes to a given candidate set of strands as well as much larger modifications, which correspond to random, long range connections in the search space induced by the standard (1-mutation) neighbourhood. We report several cases in which our algorithm finds word sets that match or exceed the best previously known constructions.

1 Introduction

DNA codes, *i.e.*, sets of DNA strands that satisfy combinatorial constraints, play an important role in various approaches to biomolecular computation [7, 8], nanostructure design [16, 18], and molecular tagging [1, 2, 6]. Good code design is important in order to minimise errors due to non-specific hybridization between distinct strands and their complements, to obtain a higher information density, and to obtain large sets of strands for large-scale applications.

For the types of combinatorial constraints typically desired, there are no known efficient algorithms for DNA code design. Techniques from coding theory have been applied to the design of DNA codes [2, 8]; while valuable, this approach is hampered by the complexity of the combinatorial constraints on the sets of DNA strands ("code words"), which are often hard to reason about theoretically. For these reasons, heuristic approaches such as stochastic local search offer much promise in design of DNA codes.

Stochastic local search (SLS) 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, and scheduling. Over the past few years there has been considerable success in developing

[⋆] To whom correspondence should be addressed.

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 codes. Deaton *et al.* [3, 4] and Zhang and Shin [19] used genetic algorithms for designing DNA codes, and provide some small sets of code words that satisfy well-motivated combinatorial constraints. However, some details of their algorithms are not specified in these papers. Faulhammer *et al.* [6] also use a stochastic search approach and provide an implementation of 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 the design of stochastic algorithms for DNA code design or to do detailed comparisons of their approaches with other algorithms. Our goal is to understand which algorithmic principles are most effective in the application of SLS methods to the design of DNA or RNA word sets (and more generally, codes over other alphabets, particularly the binary alphabet).

Our previous work [17] presents results on the performance of a new SLS algorithm for the design of DNA codes fulfilling different combinations of combinatorial constraints, the same as the ones described in Section 2. In that work, we reported empirical results that characterised performance of the SLS algorithm and indicated its ability to find high-quality sets of DNA codes.

In this paper, we describe an improved version of the simple stochastic local search algorithm for DNA code design presented in [17]. In particular, we describe how by using hybrid randomised neighbourhoods, substantial performance improvements can be achieved. To our best knowledge, this type of neighbourhood has not been previously described and may well be applicable to SLS algorithms for other problems.

In this study, we have chosen to design word sets that fullfil the following three 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 codes of fixed size (= number of strands) that may violate the given constraints. The underlying search strategy is based on a combination of randomised iterative improvement and conflict-directed random walk. The basic algorithm is initialised 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 neighbourhoods which will be further described in Section 4. The algorithm terminates when a set of DNA strands that satisfies all given constraints is found, or after a specified number of iterations has 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. Optimal settings for this parameter have been reported in [17] and we will show how these are affected by different choices of neighbourhoods.

Our empirical results, reported in Section 5, show that compared to our previous, simple SLS algorithm, our new SLS algorithm shows dramatically improved performance on hard DNA code design problems. In particular, by empirically analysing its run-time distributions, we show that for DNA code design problems studied in [17], the new algorithm does not suffer from the previously reported severe stagnation behaviour.

We compared the sizes of the word sets obtainable by our algorithm with previously known word sets, starting with the previously studied case of word sets that satisfy all three constraints. Out of a total of 30 comparisons with previous results (see Tables 1 and 2), 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 additional random words.

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. We consider the following constraints:

- **Hamming Distance 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 letters within each word is either G or C. Throughout, we assume that this percentage is 50%.
- **Reverse Complement 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.* [8].

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 is approximately 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 [11].

3 Related Work

Stochastic search methods have been used successfully for decades in the construction of good binary codes [5, 10]. 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 not presented.

Deaton *et al.* [3, 4] and Zhang and Shin [19] 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.* [9] used an 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 computational methods to generate word sets (see for example [1]), but provide no details on their algorithms. Some DNA code design programs are publicly available. The DNASequenceGenerator program [15, 7] 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.* [6] for their RNA-based 10-variable computation.

4 The Improved Stochastic Local Search Algorithm

Our basic stochastic local 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 shows the outline of the simple SLS algorithm as described in [17].

The underlying search strategy is based on a combination of randomised iterative improvement and conflict-directed random walk. The search is initialised with a randomly selected set of DNA strands. 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 selection process for conflicting code words is done uniformly at random from the pool of candidate code words involved in one or more conflicts. The modification process is based on replacing a code words w that is currently involved in a conflict, with a new code word w' chosen from a pool of related DNA words called the neighbourhood of w. With probability $(1 - \theta)$, we select w' such that the number of conflicts in the set of code words, S, is maximally reduced; otherwise we select a neighbour of w uniformly at random. (θ is a parameter of our algorithm.) Here, we propose different

```
procedure StochasticLocalSearch for DNA Code 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
            $M$ := $\mathcal{N}(w_1) \cup \mathcal{N}(w_2)$, i.e. all words from the neighbourhoods of $w_1$ and $w_2$
            with probability $\theta$ do
                select word $w'$ from $M$ uniformly at random
            otherwise
                select word $w'$ from $M$ such that
                number of constraint violations in $S$ is maximally decreased
            end with probability
            if $w' \in \mathcal{N}(w_1)$ 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 Code Design
```

Fig. 1. Outline of the stochastic local search procedure for DNA code design; $\mathcal{N}(w)$ denotes the neighbourhood of code word w.

types of neighbourhoods (mostly based on randomisation), which lead to performance improvements of the basic SLS algorithm. If after a user specified number of steps no valid code (*i.e.*, set of DNA words satisfying all given constraints) has been found, the search process is re-initialised and new attempt at finding a solution is made (outer loop in Figure 1). The algorithm terminates when a valid code is found, or a given number of unsuccessful tries have been completed.

It should be noted that our algorithm considers only words with the prescribed GC content during the search, and the neighbourhoods are restricted accordingly. In this paper, we consider the following neighbourhoods:

ν-mutation neighbourhood. The ν-mutation neighbourhood of a given code word w_0 consists of all code words that can be obtained from w_0 by modifying up to ν bases, except w_0 itself. Our previous SLS algorithm was based on the 1-mutation neighbourhood; for a given pair of code words of length n (as considered in each step of our algorithm), there are $2 \times n$ 1-mutation neighbours that fulfill the GC constraint. The 3-mutation neigbourhood of a pair of code words, in contrast, consists of $n \times (n \times n + 5)/3$ code words satisfying the GC content constraint.

Pure random neighbourhoods. Another simple way of defining the neighbourhood of a given word w is by choosing a fixed number of random code words with the same length and GC-content as w. Note that this pure random neighbourhood will differ between search steps in which the same w is chosen. This type of neighbourhood increases the mobility of the algorithm within the search space and supports search steps that are equivalent to several search steps in a ν-neighbourhood with small ν. Somewhat surprisingly, using this rather simplistic neighbourhood mechanism leads to substantial improvements in the performance of our algorithm, as we will document in Section 5.

Hybrid randomised neighbourhoods. These are obtained by adding elements of the pure random neighbourhood to a ν-mutation neighbourhood. This effectively enables the algorithm to explore regions of the search space that could not be reached easily using pure ν-mutation neighbourhoods, while still keeping the search focussed on local regions of the space of candidate sets of code words. The additional randomisation of the search achieved by adding random code words to the set of ν-mutation neighbours enhances the ability of the SLS algorithm to escape from local minima regions and eventually find solutions faster. As our empirical results show, this novel type of neighbourhood leads to substantial improvements in the performance of our SLS algorithm.

In the next section we will discuss the impact of these neighbourhoods on the performance of our SLS algorithm for DNA code design. In particular, we will see that the use of randomised neighbourhoods helps to avoid search stagnation and allows the algorithm to find bigger DNA codes.

5 Results and Discussion

To evaluate the performance of our improved SLS algorithm, we performed two types of computational experiments. First, detailed analyses of the run-time and run-length distributions (RTDs and RLDs) of our algorithm on individual problem instances were used to study the behaviour of the algorithm and the impact of parameter settings. For these empirical analyses, the methodology of [12] for measuring and analysing RTDs and RLDs was used. Run-length 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. Then, in a second type of experiment, we used the optimised parameter settings obtained from the detailed analyses for obtaining DNA code sets of maximal size for various word lengths and combinatorial constraints.

5.1 Neighbourhoods

To study and characterise the behaviour of the new proposed neighbourhood mechanisms for the simple SLS algorithm, we measured RTDs and RLDs from 1000 suc-

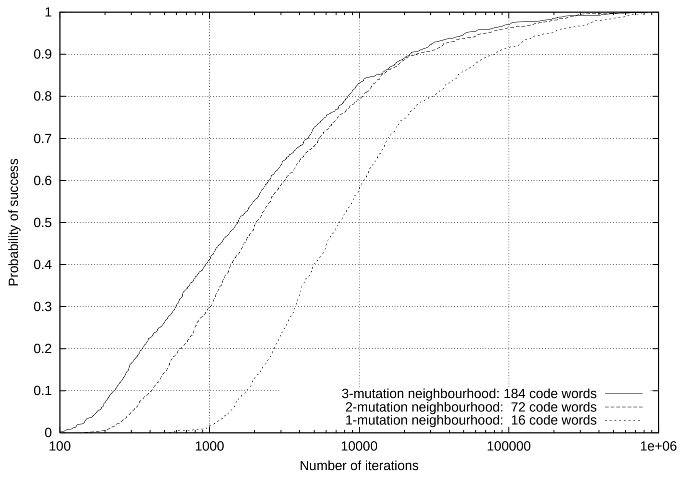

Fig. 2. RLDs for different ν-mutation neighbourhoods, set size $k = 70$, word length $n = 8$, Hamming distance $d = 4$, all 3 constraints.

cessful runs of the algorithm applied to a representative problem instance with set size $k = 70$, words length $n = 8$, Hamming distance $d = 4$ and all 3 constraints (HD, RC, and GC). Experiments with other problem instances gave analogous results (not reported here) to the ones observed for this instance. 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 neighbourhood, we performed a number of independent runs of the algoritm in which we measured the number of search iterations required for finding a solution. The empirical run-length distribution that can be easily obtained from this data gives the probability of finding a solution as a function of the number of search iterations performed. Run-time distributions are obtained by multiplying the number of search steps represented in the respective RLD with the corresponding CPU time per step.

ν-mutation neighbourhoods. The 1-mutation neighbourhood has been successfully used in the simple version of the SLS algorithm described in [17]. While one would expect that using ν-mutation neighbourhoods with $\nu > 1$ decreases the number of search steps required for finding certain word sets, it is not clear whether the increased computational cost of scanning these larger neighbourhoods can be amortised. Figure 2 shows RLDs for different types of ν-mutation neighbourhoods for the problem instance described above. The time required for obtaining a set of words of size $k = 70$ with a fixed probability p increases with ν and p. For high p, this increase is much more dramatic than for low p values. The right ‘fat’ tails of the RLDs emphasise this phenomenon. As can be seen from Figure 3, the higher time complexity of the search steps using ν-mutation neighbourhoods with $\nu > 1$ is not amortised by the reduction in the number of search steps. Similar results were obtained for other problem instances.

Pure random neighbourhoods. Interestingly, using pure random neighbourhoods leads to better performance of our algorithm than any of the ν-mutation neighbourhoods. For our representative problem instance, this can be seen when comparing the medians of the RTDs for ν-mutation neighbourhoods in Figure 3 with the median run-times for pure random neighourhoods for varying sizes shown in Figure 6. At the same time, using pure random neighbourhoods leads to RTDs that do not have the same "fat" right tails that indicated the stagnation behaviour of our SLS algorithm for the ν-mutation neighbourhoods. This leads to even more substantial performance advantages of pure random neighbourhoods over the ν-mutation neighbourhoods when comparing the mean CPU times for solving a given problem instance or high percentiles of the respective RTDs.

When varying the size of the pure random neighbourhoods, *i.e.*, the number of code words considered for replacing a given word in a candidate word set, we found that typically there is an optimal range of neighbourhood sizes. When using smaller neighbourhoods, the performance of the algorithm decreases since intuitively a higher number of "shorter" search steps is required for covering the same distance in the search space (*e.g.*, to the nearest solution). For larger neighbourhoods, the time complexity for each search step increases, and at some point the reduction in the number of search steps required for finding a solution no longer amortises this higher cost. This is illustrated for our representative problem instance in Figure 6.

Hybrid neighbourhoods. We noticed that adding random code words to ν-mutation neighbourhoods leads to improved performance of our SLS algorithm. This raises the following question: is there any reason to keep the ν-mutation neighbourhood as part of a bigger, hybrid randomised neighbourhood?

We investigated this question in an experiment in which we compared three hybrid neighbourhoods of 200 words that include the 1-mutation, 2-mutation, and 3-mutation neighbours of a given word, respectively, as well as a purely random neighbourhood of size 200. From Figures 4 and 5 we can see that for our representative problem instance, including the 1-mutation neighbourhood in a bigger, hybrid randomised neighbourhood results in slight performance improvements in terms of iterations as well as CPU time, while including the 2- or 3-mutation neighbourhoods is disadvantageous.

In a second experiment we tested whether this result also holds for different neighbourhood sizes. As can be seen from Figure 6, hybrid neighbourhoods obtained by adding random neighbours to the 1-mutation neighbourhood generally leads to improved performance compared to using pure random neighbourhoods of the same size. One intuitive explanation for the efficiency of using hybrid neighbourhoods is based on the fact that 1-mutation neighbours can be easily mutated back into the original word. This mechanism allows the algorithm to easily and cheaply reverse problematic search steps that, *e.g.*, lead into a local minimum of the underlying search space.

Further experiments have been performed for different (k, n, d) combinations (*e.g.*, (102,10,5), (10,10,7), (15,12,8), (25,6,3)) as well as for different set sizes and GC content fractions (*e.g.*, $k = 56$ and GC-content = 3, $k = 28$ and GC-content = 2, and $k = 8$ and GC-content = 1). Our algorithm found solutions faster when we used hybrid neighbourhoods than when using pure random neighbourhoods, considering the total size of the neighbourhood as being fixed in both cases. The CPU time per step is roughly the

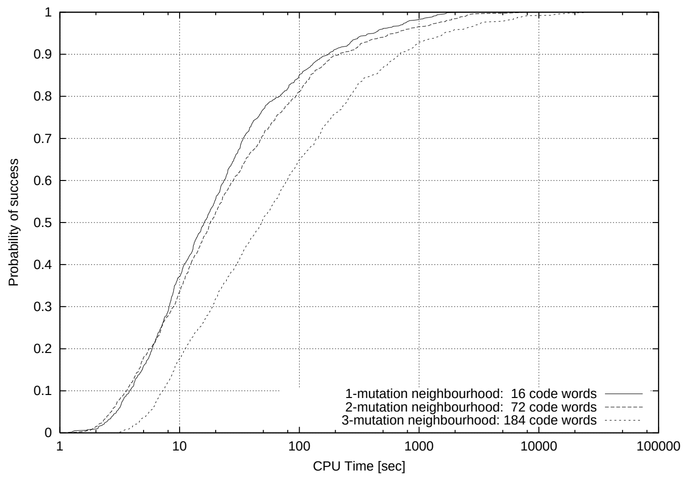

Fig. 3. RTDs for different ν-mutation neighbourhoods, set size $k = 70$, word length $n = 8$, Hamming distance $d = 4$, all 3 constraints.

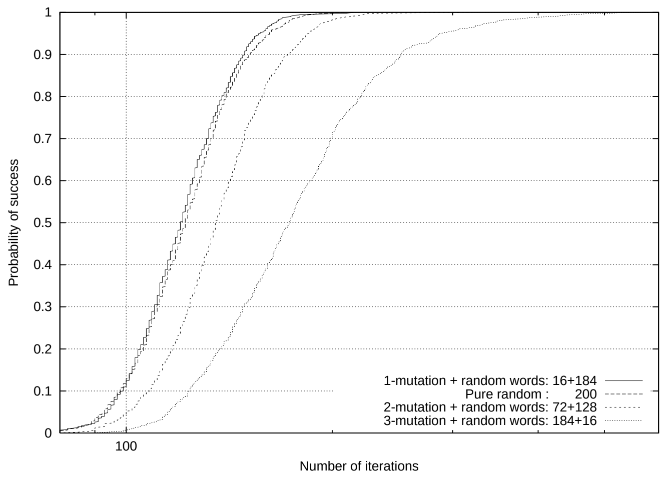

Fig. 4. RLDs for pure random and hybrid neighbourhoods of size 200, set size $k = 70$, word length $n = 8$, Hamming distance $d = 4$, all 3 constraints.

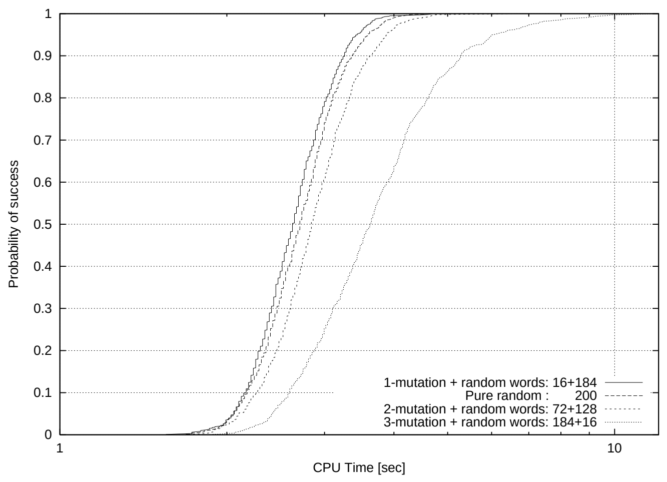

Fig. 5. RTDs for pure random and hybrid neighbourhoods of size 200, set size $k = 70$, word length $n = 8$, Hamming distance $d = 4$, all 3 constraints.

same for hybrid and pure random neighbourhoods of the same size, but when using the hybrid neighbourhood, a smaller number of steps is required for reaching the same solution quality than when using pure random neighbourhoods. Overall, in all the cases we examined, using hybrid neighbourhoods consisting of all 1-mutation neighbours and additional random code words lead to better performance than using pure random or ν-mutation neighbourhoods.

5.2 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 escape from local minima. Using the previous 1-mutation neighbourhood, we found an optimal setting for this noise parameter θ around 0.2 (Figure 7) for different problem instances and sizes, as described in [17]. When considering randomised neighbourhoods, the optimal value for the noise parameter appears to be 0 as can be seen in Figure 8.

One possible explanation for this phenomenon may reside in the added need of greediness in the search algorithm when searching bigger neighbourhoods. Furthermore, the additional diversification provided by random neighbours can apparently compensate and even substitute for the effect of the noise mechanism – both provide mechanisms for escaping from local optima in the underlying search space.

5.3 New DNA Codes and Empirical Bounds

After studying the impact of neighbourhood type and size as well as the noise parameter setting, we used the enhanced SLS algorithm (based on hybrid randomised neighbourhoods) to solve a number of challenging instances of our DNA code design problem.

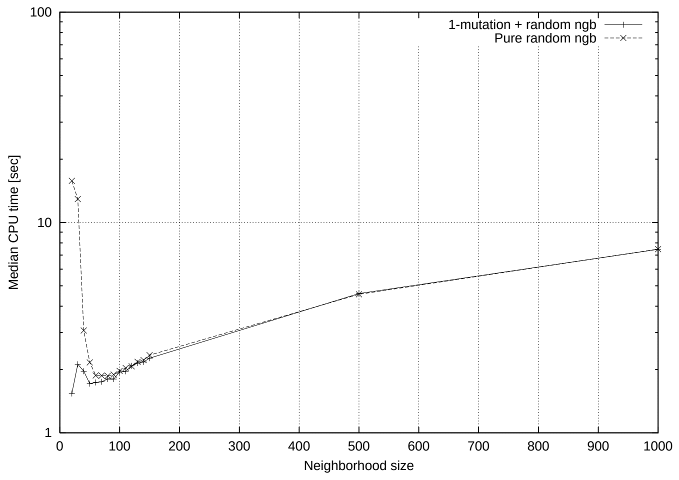

Fig. 6. Median number of CPU seconds for different neighbourhood sizes, set size $k = 70$, word length $n = 8$, Hamming distance $d = 4$, GC-content = 50%, all 3 constraints.

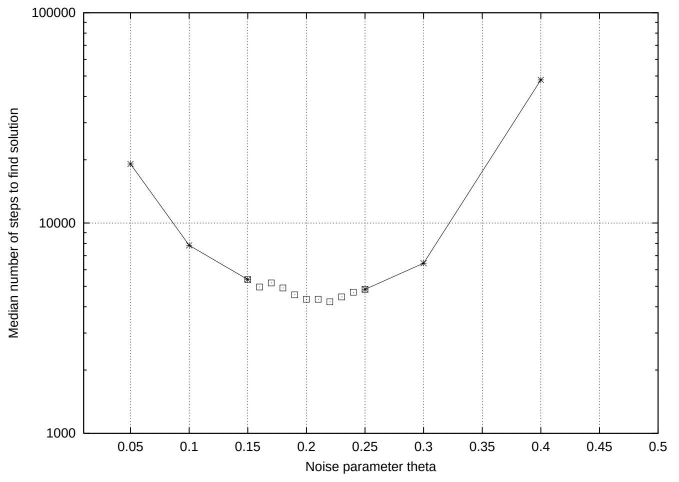

Fig. 7. Median number of iterations as a function of noise parameter values: 1-mutation neighbourhood, all three constraints, $n = 8$, $d = 4$, and $k = 70$.

For the DNA code design problem with all three constraints (HD, GC, RC) and 50% GC-content, we compared the sizes of the word sets obtained with our new SLS algorithm with previously known word sets [17]. Out of a total of 31 comparisons with previous results (see Tables 1 and 2), we found word sets that equal or improved on previous constructions in all but one case. In this particular case ($n = 8$ and $d = 4$),

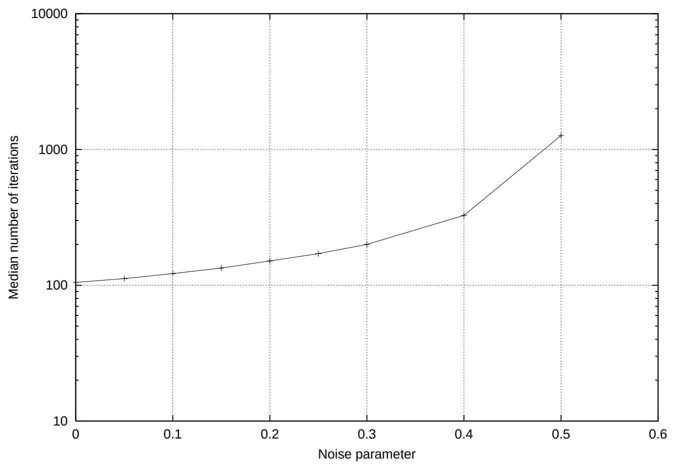

Fig. 8. Number of iterations as a function of noise parameter values: hybrid neighbourhood, all three constraints, $n = 8$, $d = 4$, and $k = 70$.

n/d	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
4	20 $[.03k]$	5 $[.02k]$	2 $[.001k]$	-	-	-	-	-	-
6	282 $[1k]$	37 $[19k]$	11 $[3k]$	2 $[.003k]$	2 $[.006k]$	-	-	-	-
8	3981 $[20k]$	350 $[119k]$	$92^\star$ $[4000k]$	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 $[1.5k]$

Table 1. Set sizes for **(HD,RC,GC)** DNA codes obtained with the simple SLS algorithm presented in [17]. 1-mutation neighbourhood have been used for all the results. The numbers in square brackets represent the average number of iterations spent by the algorithm to obtain the set with the specified size.

we obtained a code of size 107 when initialising the search with a random set of code words. This is a substantial improvement over our simple SLS algorithm, which only found codes sizes of 92. However, for the same case, Frutos *et al.* [8] constructed a set of 108 words. But even with our simple algorithm we have obtained sets of 112 code words by initialising the search with the best known set containing 108 code words and by iteratively expanding this set with one additional code word at a time (initialised at random) [17]. The same code size of 112 is also achieved by our new SLS algorithm with randomised neighbourhoods. It may be noted that Frutos *et al.* used a theoretical approach to design the 108 set. Their map-template construction relies on symmetries and other mathematical properties of this specific code design problem and, different from our SLS algorithm, it cannot be used for iteratively improving or expanding a given code.

It may be noted that, to our best knowledge, there are no theoretical bounds for DNA codes fulfilling the HD, RC, and GC constraints known from the literature. Some

n/d	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
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** $[.2k]$

Table 2. Set sizes for **(HD,RC,GC)** DNA 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. Bold-face numbers represent improved set sizes compared with the previous ones obtained in [17]. The numbers in square brackets represent the average number of iterations spent by the algorithm to obtain the set with the specified size.

theoretical upper and lower bounds have been published by Marathe *et al.* [14] for codes satisfying the HD and RC constraints. We compared our results with their bounds, keeping also in mind that our codes have a fixed GC content. For the $(n, d) = (8, 4)$ case, our best result (code size 112) is quite close to the lower bound of 128 from Marathe *et al.*, but of course it is not clear whether that bound applies for codes that additionally have to satisfy the 50% GC content constraint we used. In other situations, our results improved on the Marthe *et al.* bounds. For example, for $(n, d) = (10, 5)$, their lower bound is 32 code words, while our simple and enhanced SLS algorithm reach code sizes of 127 and 158, respectively. It is also worth noting in most cases, the ranges between the theoretical lower and upper bounds from Marateh *et al.* are very large. For example, for the (10,5) case, the the upper bound is 1202, compared to a lower bound of 32. This provides some indication that there might be room for substantial improvements in the code sizes achievable for these and related code design problems.

Finally, it is worth mentioning that based on a very limited initial investigation, our new SLS algorithm based on randomised neighbourhoods achieves performance improvements similar to the ones reported here for the (HD, GC, RC) constraint combination for other code design problems that include the GC content constraint (*e.g.*, HD and GC constraints). We are currently performing an in-depth analysis of our algorithm's performance on these closely related code design problems, the results of which we plan to present in the near future.

6 Conclusions

We presented an improved version of the simple SLS algorithm for DNA Code Design proposed in [17], based on a new neighbourhood generation mechanism, along with empirical results that characterise its performance. New insights on the role of the neighbourhood type and size have been described and we showed evidence that by using hybrid randomised neighbourhoods, the performance of our original SLS algorithm can be significantly improved. Intuitively, the use of randomised ν-mutation neighbourhoods enhances the ability of the SLS algorithm to escape from local minima regions, and facilitates the exploration of regions in the underlying search space that are very far apart with respect to the traditional 1-mutation neighbourhood.

In future work, we plan to examine further ways for improving the algorithm. The existing theoretical bounds on combinations of constraints (see Section 5.3) similar to the ones considered here, indicate that there should be sustantial room for further improvements. One possibility to improve the SLS algorithm 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 use hybrid randomised neighbourhood mechanisms for DNA code design problems with different constraint combinations as well as for the design of binary codes.

Search space analysis may provide more insight on the hidden mechanisms that make it difficult to computationally solve DNA code design problems and shed more light on the precise reasons for the efficiency of the hybrid neighbourhoods studied here. Finally, it would be interesting to see if better theoretical design principles can be extracted from the codes that are empirically obtained from high-performance SLS algorithms for DNA code design.

Acknowledgments. This work has been supported by NSERC Individual Research Grant #238788; it builds on previous research in collaboration with Anne Condon, who also provided valuable feedback on the ideas presented here.

References

1. 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.
2. S. Brenner and R.A. Lerner, “Encoded combinatorial chemistry”, Proc. Natl. Acad. Sci. USA, Vol 89, pages 5381–5383, June 1992.
3. 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, pages 247–258, 1999.
4. 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.
5. 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.
6. D. Faulhammer, A.R. Cukras, R.J. Lipton, and L.F. Landweber, “Molecular computation: RNA solutions to chess problems,” Proc. Natl. Acad. Sci. USA, 97: 1385-1389, 2000.
7. 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).

8. 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, pages 4748-4757, December 1997.
9. 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.
10. 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.
11. H.H. Hoos, "Stochastic Local Search - Methods, Models, Applications", infix-Verlag, Sankt Augustin, Germany, ISBN 3-89601-215-0, 1999.
12. 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)*, pages 238-245, 1998.
13. 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, pages 805-812, 2002.
14. A. Marathe, A. Condon, and R. Corn, "On combinatorial DNA word design," J. Computational Biology, 8:3, pages 201-220, 2001.
15. Programmable DNA web site, `http://ls11-www.cs.uni-dortmund.de/molcomp/Downloads/downloads.html`. Visited November 11, 2000.
16. J.H. Reif, T.H. LaBean, and N.C. Seeman, "Challenges and Applications for Self-Assembled DNA Nanostructures", Proc. Sixth Inter.l Workshop on DNA-Based Computers, Leiden, The Neth., June, 2000. DIMACS Ed. by A. Condon and G. Rozenberg, Lecture Notes in CS, Springer-Verlag, Berlin Heidelberg, vol. 2054, pages 173-198, 2001.
17. D.C. Tulpan, H.H. Hoos, A. Condon, "Stochastic Local Search Algorithms for DNA Word Design", DNA 8 Conference, Japan, March 2002.
18. B. Yurke, A.J. Tuberfield, A.P.Jr Mills, F.C. Simmel and J.L. Neumann, "A DNA-fuelled molecular machine made of DNA." Nature 406, pages 605-608, 2000.
19. 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, pages 735-742, 1998.