A Hybridization of Constructive Beam Search with Local Search for Far From Most Strings Problem

Sayyed R Mousavi

Abstract—The Far From Most Strings Problem (FFMSP) is to obtain a string which is far from as many as possible of a given set of strings. All the input and the output strings are of the same length, and two strings are said to be far if their hamming distance is greater than or equal to a given positive integer. FFMSP belongs to the class of sequences consensus problems which have applications in molecular biology. The problem is NP-hard; it does not admit a constant-ratio approximation either, unless $P = NP$. Therefore, in addition to exact and approximate algorithms, (meta)heuristic algorithms have been proposed for the problem in recent years. On the other hand, in the recent years, hybrid algorithms have been proposed and successfully used for many hard problems in a variety of domains. In this paper, a new metaheuristic algorithm, called Constructive Beam and Local Search (CBLS), is investigated for the problem, which is a hybridization of constructive beam search and local search algorithms. More specifically, the proposed algorithm consists of two phases, the first phase is to obtain several candidate solutions via the constructive beam search and the second phase is to apply local search to the candidate solutions obtained by the first phase. The best solution found is returned as the final solution to the problem. The proposed algorithm is also similar to memetic algorithms in the sense that both use local search to further improve individual solutions. The CBLS algorithm is compared with the most recent published algorithm for the problem, GRASP, with significantly positive results; the improvement is by order of magnitudes in most cases.

Keywords—Bioinformatics, Far From Most Strings Problem, Hybrid metaheuristics, Matheuristics, Sequences consensus problems.

I. INTRODUCTION

THE Far From Most Strings Problem (FFMSP) is a combinatorial optimization problem which receives, as its inputs, a set S of strings of the same length m over an alphabet and a positive integer d not greater than m and asks for a string of length m over the alphabet which is *far* from as many strings in S as possible [1], [2], [3], [4], [5]. In other words, it is to maximize the number of input strings, i.e. those in S, which are far from the output string. The criterion for two strings to be far (from each other) is to have their hamming distance equal to or greater than the input integer d. FFMSP belongs to a more general class of problems called *sequences consensus*, which includes such problems as finding similar regions in a given set of DNA, RNA, or protein sequences and have applications in Bioinformatics and coding theory [6], [7], [8], [2], [9], [10]. Among other sequences consensus problems are Closest String Problem (CSP) [11], [12], [13], Closest SubString Problem (CSSP) [14], [15], [16], [17], [18], Farthest String Problem (FSP) [19], [4], Farthest SubString Problem (FSSP) [2], [20], Close to Most Strings

S. R. Mousavi is with the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Islamic Republic of IRAN, e-mail: srm@cc.iut.ac.ir.

Problem (CMSP) [3], [20], Distinguishing SubString Selection problem (DSSS) [21], and Distinguishing String Selection Problem (DSSP) [3], [20], [22], which are also known as string selection and comparison problems.

FFMSP has been proved to be NP-hard [1], [3]. Therefore, no PTIME algorithm is (currently) known to solve every instance of FFMSP to optimality [23]. It does not admit a constant-ratio approximation either, unless $P = NP$ [1], [3].

In the recent years, heuristic and metaheuristic algorithms for FFMSP have been proposed. Meneses et al. devised a heuristic algorithm which consists of a greedy constructive phase followed by an iterative improvement, a greedy perturbative, phase [4]. Festa proposed a Greedy Randomized Adaptive Search Procedure (GRASP) and reported improved results over the algorithm suggested by Menses et al. [5]. The GRASP metaheuristic was originally devised by Feo and Resende [24], [25]. It involves the execution of a number of iterations, where each iteration consists of a constructive phase followed by a local search phase which are, respectively, similar to, but not quite the same as, the constructive and iterated improvement phases in the Meneses et al.'s algorithm. The construction phase in (each iteration of) GRASP builds a candidate solution by specifying the values of the underlying variables one at a time. The variable-value selection criterion is based on a heuristic function; but not always the best choice is made. Instead, a *restricted candidate list (RCL)* of options, based on the underlying heuristic function, is developed from which a candidate is randomly selected. The local search phase receives the candidate solution made by the construction phase and tries to improve it, e.g. through an iterated improvement algorithm such as hill climbing. Two main parameters in a typical GRASP are *RCL-size* and *itr-num*. The former determines the number of candidates in RCL and can be adjusted to make an appropriate balance between greediness and randomness in the construction phase. The latter specifies the (maximum) number of iterations GRASP executes its pair of construction and local search phases. The best candidate solution, based on the underlying problem's objective function, found over all the iterations is returned as the final output of GRASP. For detail on GRASP and its variants, the reader may refer to the annotated bibliography by Festa et al. [26], [27].

In this paper, a new metaheuristic algorithm, called Constructive Beam and Local Search (CBLS), is investigated for the problem. As its name stands, the proposed algorithm is a hybridization of constructive beam search and local search algorithms. More specifically, the CBLS algorithm consists of two phases of constructive and local search procedures. In the

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

constructive phase, several candidate solutions are obtained using the constructive beam search algorithm, which are then used as the starting points in the search space for further improvement via local search. The best solution found is returned as the final solution to the problem. The proposed algorithm is also similar to memetic algorithms in the sense that both use local search to further improve individual solutions [28]. The CBLS algorithm uses a novel heuristic function proposed by the author in [29]. However, it extends the use of the heuristic function to the constructive beam search phase as well. In other words, the heuristic function is used not only to evaluate complete solutions for the purpose of local search but also to evaluate and compare partial solutions for the purpose of constructive beam search. The CBLS algorithm is compared with the state-of-the-art algorithm published in the literature which is a GRASP algorithm by Festa [5], over not only random but also real data. The experimental results show that the proposed algorithm outperforms the state-of-the-art, in most of the cases by orders of magnitude.

Although the ideas proposed in this paper are general and not restricted to a particular alphabet, we develop the theoretical and experimental results as in [4], [5], based on the four-letter alphabet of $\Sigma = \{A, T, C, G\}$. The letters A, T, C, and G stand, respectively, for *Adenine*, *Thymine*, *Cytosine*, and *Guanine*, which are four different bases in DNA strands. Similar results can be obtained, by possibly minor changes, based on other alphabets.

The rest of the paper is organized as follows. The next section provides problem definition and basic notations used in the paper. In Section 3, the estimated Gain-per-Cost heuristic is described. A triangle, called Δ-triangle, which is used to determine the estimated Gain-per-Cost heuristic for candidate solutions is explained in Section 4. Section 5 proposes the hybrid heuristic function $h_{f, \widetilde{GpC}}(.)$. These are mainly from the author's previous research in [29]. The CBLS algorithm is presented in Section 6. Experimental results are reported in Section 7, and Section 8 concludes the paper.

II. PRELIMINARIES

Let s be a string of length m. We use s^k, where k is an integer such that $1 \leq k \leq m$, to denote the kth character of s. Let s_1 and s_2 be two strings of the same length m. The hamming distance between s_1 and s_2 is denoted by $d_H(s_1, s_2)$ and defined as $\sum_{k=1}^{m} \delta(s_1^k, s_2^k)$, where:

$$\delta(s_1^k, s_2^k) = \begin{cases} 1 & \text{if } s_1^k \neq s_2^k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The Far From Most Strings Problem is defined as follows:
FFMSP:
Instance: a pair $< S, d >$, where S is a set of strings $s_1, s_2, \ldots, s_n, n > 1$, all of the same length m over an alphabet Σ and d is an integer, called distance threshold, such that $1 \leq d \leq m$.
Output: a string X of length $m > 0$ over the alphabet Σ.
Maximize: the number of strings $s_k \in S$ such that $d_H(X, s_k) \geq d$.

In the rest of the paper, we consistently use the following notations. We use the pair $< S, d >$ to denote the underlying instance of FFMSP. We assume that $S = \{s_1, s_2, \ldots, s_n\}$; that is, the *input strings* are denoted by the small letter s indexed from 1 to n, where $n > 1$ is the number of the input strings. Note that, since S is a set, the input strings are all assumed to be distinct. We use (possibly indexed) X to denote a candidate solution. We use m to denote the length of the strings and assume that $m > 0$. We use Σ as the alphabet of the characters used in any input string and assume $|\Sigma| > 1$. The objective function for FFMSP is denoted by $f(.)$. That is, $f(X)$ is the number of strings in S whose distance from X is at least d, where X is a candidate solution. The value $f(X)$ is called the *objective value* for the solution X. For simplicity, we use $d_j(X)$ to denote $d_H(X, s_j)$, $j = 1, 2, \ldots, n$. That is, $d_j(X)$ is the hamming distance between s_j and X. We say that a string s_j is *far* from X if $d_j(X) \geq d$; it is otherwise *near* X. The set of strings in S which are near X is denoted by $Near(X)$. The cost of a string s_j is defined as $c_j(X) = d - d_j(X)$. We further define

$$f_j(X) = \begin{cases} 1 & \text{, if } d_j(X) \geq d; \\ 0 & \text{, otherwise} \end{cases} \quad (2)$$

Therefore,

$$f(X) = \sum_{j=1}^{n} f_j(X) \quad (3)$$

Example 1: Let $< S, 3 >$ be an instance of FFMSP, where $S = \{s_1, s_2, s_3\}$, $s_1 = $ "GATTG", $s_2 = $ "GATCA", $s_3 = $ "CTCGA", and consider the candidate solution $X = $ "GATTC". Then, $n = 3$, $m = 5$, $d = 3$, $d_1(X) = 1$, $d_2(X) = 2$, $d_3(X) = 5$, $c_1(X) = 2$, $c_2(X) = 1$, and $c_3(X) = -2$. The input strings s_1 and s_2 are near X, whereas s_3 is far from it. Therefore, $f_1(X) = f_2(X) = 0$ but $f_3(X) = 1$. That is, $Near(X) = \{s_1, s_2\}$.

By a *walk* of length L, or for short an L-*walk*, $L \in \mathbb{N}$, $1 \leq L \leq m$, where $\mathbb{N}$ is the set of natural numbers, from a point X_{old} in the search space we mean to alter the values of exactly L distinct characters of X_{old}, resulting in a new point X_{new} in the search space such that $d_H(X_{old}, X_{new}) = L$. We call L the *length* and the points X_{old} and X_{new}, respectively, the *source* and the *destination* of the walk. We extend the definition to the case $L = 0$, for *no-walk*, in which case $X_{old} = X_{new}$. An L-walk is called *random* if each point X_j in the search space such that $d_H(X_{old}, X_j) = L$ is equally-likely to be its destination, where X_{old} is the source of the walk.

Let X_{old} and X_{new} be, respectively, the source and the destination of an L-walk and s_j be a string in S. We define Δ_j for this L-walk as $d_j(X_{new}) - d_j(X_{old})$. If the walk is random, Δ_j will be a random variable. We use $Pr_L(\Delta_j = k)$, $k \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of proper numbers, to denote the probability for the random variable Δ_j to take the value k as the result of a random L-walk. Similarly, we use, for example, $Pr_L(\Delta_j \leq k)$, $k \in \mathbb{Z}$, to denote the probability for the random variable Δ_j to take any value less than or equal to the value k as the result of a random L-walk. We still keep $Pr(.)$- without index- to denote the conventional probability function and also $Ex(.)$ to denote the statistical expectation function.

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

III. THE ESTIMATED GAIN PER COST HEURISTIC

In this section, the *Estimated Gain per Cost $(\widetilde{GpC})$* heuristic evaluation function is proposed. Some preliminaries are first required.

Definition 1: Let X_{old} be a candidate solution and s_j be a string in $Near(X_{old})$. By a *fix* for s_j from X_{old}, we mean an L-walk whose source and destination are, respectively, X_{old} and X_{new} such that $d_j(X_{new}) = d$. We assume that the walk is designed independently from the other strings in S, i.e. it can be treated as a random L-walk with respect to such strings. Informally-speaking, a fix for a string s_j from the current candidate solution is to alter exactly L characters of the current candidate solution, where $L = d - d_j$, in such a way that s_j is far from the resulting candidate solution X_{new} in order to contribute 1 unit to the (new) objective value $f(X_{new})$. However, since the walk is random with respect to the other strings, $f(X_{new})$ is still a random variable, for $n > 1$.

Definition 2: Let X_{old} be a candidate solution, and s_j be a string in $Near(X_{old})$. The *potential gain*, or for short the *gain*, of s_j with respect to X_{old} is denoted by $g_j(X_{old})$ and defined as the expected value of $f(X_{new})$, where X_{new} is the destination of a fix for s_j from X_{old}.

The next theorem shows that the gain of a string s_j as defined above can be calculated if the probability distribution for Δ_j, with respect to the underlying L-walk, is known.

Theorem 1: Let X_{old} be a candidate solution and s_j be a string in $Near(X_{old})$. Then:

$$g_j(X_{old}) = 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Pr_L(\Delta_k \geq c_k(X_{old})) \qquad (4)$$

where $L = c_j(X_{old})$
To prove this we first provide a lemma.

Lemma 1: Let X_{old} be a candidate solution and s_k be a string in S. Then the probability for the string s_k to be far from the destination X_{new} of a random L-walk from X_{old} is $Pr_L(\Delta_k \geq c_k(X_{old}))$.

Proof: The probability for the string s_k to be far from X_{new} is:

$$\begin{aligned} Pr(f_k(X_{new}) = 1) &= Pr(d_k(X_{new}) \geq d) \\ &= Pr_L(\Delta_k \geq d - d_k(X_{old})) \\ &= Pr_L(\Delta_k \geq c_k(X_{old}))r \end{aligned} \qquad (5)$$

∎

We now prove the theorem.
Proof:

$$\begin{aligned} g_j(X_{old}) &= Ex(f(X_{new})) \\ &= Ex(\sum_{k=1}^{n} f_k(X_{new})) \\ &= Ex(f_j(X_{new}) + \sum_{\substack{k=1 \\ k \neq j}}^{n} f_k(X_{new})) \\ &= Ex(1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} f_k(X_{new})) \end{aligned}$$

$$\begin{aligned} &= 1 + Ex(\sum_{\substack{k=1 \\ k \neq j}}^{n} f_k(X_{new})) \\ &= 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Ex(f_k(X_{new})) \\ &= 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Pr(f_k(X_{new}) = 1) \\ &= 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Pr_{c_j(X_{old})}(\Delta_k \geq c_k(X_{old})) \end{aligned}$$

∎

Definition 3: Let X be a candidate solution and s_j be a string in $Near(X)$. The gain-per-cost of s_j, with respect to X, is defined as the ration of its gain $g_j(X)$ to its cost $c_j(X)$. The Gain-per-Cost of X is denoted by $GpC(X)$ and defined as the average of the gain per costs of the strings in $Near(X)$. That is:

$$GpC(X) = \frac{1}{|Near(X)|} \sum_{s_j \in Near(X)} \frac{g_j(X)}{c_j(X)} \qquad (6)$$

where $|Near(X)| \neq 0$; it is defined as to be zero if $|Near(X)| = 0$.

Given a candidate solution X, the gain-per-cost of a string s_j in $Near(X)$, as defined above, is a measure which is directly proportional to its gain but is conversely-proportional to its cost. Informally-speaking, the higher the expected value of the number of strings far from the destination of a fix for s_j is, the higher its gain-per-cost will be. On the other hand, the lower its cost is, the higher its gain-per-cost will be. The Gain-per-Cost measure of X is the average of such gain-per-cost values, over strings in $Near(X)$. Base on this definition, we propose the Gain-per-Cost of a candidate solution X as a *heuristic* to indicate the closeness of X to better candidate solutions, i.e. those with higher objective values.

The problem, however, with the Gain-per-Cost of candidate solutions as defined by Definition 3 is that, for each possible length of the random walks, it requires the probability, or equivalently the cumulative, distribution function for Δ_k, as stated by Theorem 1, which can be different for different strings s_k in S. In this paper, we do not intend to determine these probability distribution functions for every string $s_k \in S$. Instead, we *estimate* them with the probability distribution function for a random variable Δ (with no index) which corresponds to a random string as a representative for the strings in S. The variable Δ, with respect to a random L-walk, is then defined as $d_H(s, X_{new}) - d_H(s, X_{old})$, where X is a random string, as opposed to one in S, and X_{old} and X_{new} are the source and destination of the random L-walk, respectively. Of course, this estimation can loose information. Nevertheless, as the experimental results indicate, the use of the heuristic evaluation function based on this approximation still improves state-of-the-art considerably. Our proposed *estimated Gain-per-Cost $(\widetilde{GpC})$* heuristic evaluation function is then the following (unless $|Near(X)| = 0$ in which

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

$$X_{old} = \texttt{"ACGTC"}$$
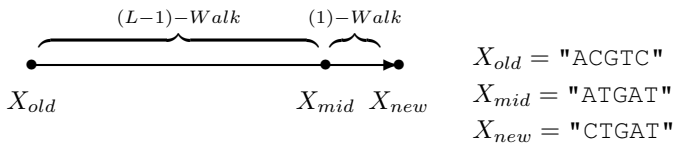$$X_{mid} = \texttt{"ATGAT"}$$
$$X_{new} = \texttt{"CTGAT"}$$

Fig. 1. An L-walk, $L > 0$, may be considered as an $(L-1)$-walk followed by a 1-walk. Sample source and destination points for $L = 4$ are displayed at the right.

case $\widetilde{GpC}(X)$ is defined as 0):

$$\widetilde{GpC}(X) = \frac{1}{|Near(X)|} \sum_{s_j \in Near(X)} \frac{\widetilde{g}_j(X)}{c_j(X)} \qquad (7)$$

where $\widetilde{g}_j(X)$ represents the estimated gain of a string s_j, with respect to X, defined as:

$$\widetilde{g}_j(X) = 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Pr_{c_j(X_{old})}(\Delta \geq c_k(X_{old})) \qquad (8)$$

The probability distribution for the random variable Δ is derived in the next section.

IV. PROBABILITY DISTRIBUTION FUNCTION FOR Δ

In this section, the probability distribution function (PDF) for the random variable Δ with respect to a random L-walk is derived. For simplicity, in this section we use $p_L(.)$ to denote this function; that is, $p_L(k) = Pr_L(\Delta = k)$, $L \in \mathbb{N}_0$, $k \in \mathbb{Z}$, where $\mathbb{N}_0$ is the set of natural numbers including zero. By definition, we now this for the case $L = 0$:

$$p_0(k) = \begin{cases} 1 & , \texttt{if } k = 0; \\ 0 & , \texttt{otherwise} \end{cases} \qquad (9)$$

The following theorem gives a recursive formula to determine the probability distribution for Δ, for $L > 0$.

Theorem 2: Let s be a random string and $\Delta = d_H(s, X_{new}) - d_H(s, X_{old})$, where X_{old} and X_{new} are the source and the destination, respectively, of a random L-walk, $L > 0$. Then:

$$p_L(k) = \frac{1}{2}p_{L-1}(k) + \frac{1}{4}(p_{L-1}(k-1) + p_{L-1}(k+1)) \quad (10)$$

Proof: Since $L > 0$, the random L-walk can be considered as a random $(L-1)$-walk followed by a random 1-walk with the constraint that the latter does not change any character already changed by the former. Let X_{mid} denote the destination of the random $(L-1)$-walk, which is the source of the subsequent random 1-walk (see Figure 1). Let k be the index of the character changed as the result of the 1-walk. That is, X_{mid} and X_{new} are different in their k^{th} characters, where $1 \leq k \leq m$. Note that X_{old} and X_{mid} cannot, therefore, be different in their k^{th} characters. There exist altogether three possible cases:

  i $s^k = X_{mid}^k$ and $s^k \neq X_{new}^k$
  ii $s^k \neq X_{mid}^k$ and $s^k = X_{new}^k$
  iii $s^k \neq X_{mid}^k$ and $s^k \neq X_{new}^k$

In case (i), the kth character of s is the same as the kth character of X_{mid} which has to be different from the kth character of X_{new}, hence $d_H(s, X_{new}) = d_H(s, X_{mid}) + 1$. In case (ii), similarly, the kth character of s is the same as the kth character of X_{new} which has to be different from the kth character of X_{old}, hence $d_H(s, X_{new}) = d_H(s, X_{mid}) - 1$. Finally, in case (iii), the kth character of s is different from both the kth character of X_{mid} and the kth character of X_{new}, hence $d_H(s, X_{new}) = d_H(s, X_{mid})$. These imply:

$$-1 \leq d_H(s, X_{new}) - d_H(s, X_{mid}) \leq 1 \qquad (11)$$

The probabilities for each of these cases to happen are as follows. Note that X_{mid}^k and X_{new}^k has to be different because of the 1-walk and that we have assumed $\Sigma = \{\texttt{A}, \texttt{T}, \texttt{C}, \texttt{G}\}$, hence $|\Sigma| = 4$. The probability for case (i) is $\frac{1}{4}$ (for any possible value for X_{mid}^k, s^k can equally-likely take four values one of which satisfies $s^k = X_{mid}^k$). Similarly, the probability for case(ii) is $\frac{1}{4}$ (for any possible value for X_{new}^k, s^k can equally-likely take four values one of which satisfies $s^k = X_{new}^k$). What remains makes the probability for case(iii) , i.e. $1 - (\frac{1}{4} + \frac{1}{4}) = \frac{1}{2}$.

By definition of Δ, we have:

$$p_L(k) = Pr_L(d_H(s, X_{new}) - d_H(s, X_{old}) = k) \qquad (12)$$

Let A and B denote, respectively, $(d_H(s, X_{new}) - d_H(s, X_{mid}))$ and $(d_H(s, X_{mid}) - d_H(s, X_{old}))$. Then:

$$p_L(k) = Pr(A + B = k) \qquad (13)$$

From Inequation 11, we know $-1 \leq A \leq 1$, hence:

$$\begin{aligned} p_L(k) = Pr(\quad &(\quad A = -1 \; \wedge \; B = k+1) \; \vee \quad (14) \\ &(\quad A = 0 \; \wedge \; B = k) \; \vee \\ &(\quad A = 1 \; \wedge \; B = k-1)) \end{aligned}$$

Since the three disjoint cases in the right-hand side of the equation are mutually exclusive, the probability in the right-hand side can be written as the sum of the probabilities for those three cases. That is:

$$\begin{aligned} p_L(k) = \quad &Pr \quad (A = -1 \; \wedge \; B = k+1) + \quad (15) \\ &Pr \quad (A = 0 \; \wedge \; B = k) + \\ &Pr \quad (A = 1 \; \wedge \; B = k-1) \end{aligned}$$

Since A can take any of the values -1, 0, or 1 independently from the value of B, the conjoint conditions within each of the three probabilities in the right-hand side are independent. Therefore:

$$\begin{aligned} p_L(k) = \;&Pr(A = -1) \quad \times \quad Pr(B = k+1) + \quad (16) \\ &Pr(A = 0) \quad \times \quad Pr(B = k) + \\ &Pr(A = 1) \quad \times \quad Pr(B = k-1) \\ = \;&Pr(A = -1) \quad \times \quad p_{L-1}(k+1) + \\ &Pr(A = 0) \quad \times \quad p_{L-1}(k) + \\ &Pr(A = 1) \quad \times \quad p_{L-1}(k-1) \end{aligned}$$

On the other hand, $Pr(A = -1)$, $Pr(A = 0)$, and $Pr(A = 1)$ are, respectively, equal to the probabilities for the cases (i),

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

(iii), and (ii), which are $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{1}{4}$, respectively. Therefore:

$$p_L(k) = \frac{1}{2}p_{L-1}(k) + \frac{1}{4}(p_{L-1}(k-1) + p_{L-1}(k+1)) \quad (17)$$

■

Note that $p_L(k)$ is zero for $k > L$. This can be derived both as a corollary from the above theorem and directly from the fact that as a result of an L-walk, L characters are only changed.

Corollary 1: Let $T(.,.)$ be a function from $\mathbb{N}_0 \times \mathbb{Z}$ to $\mathbb{N}_0$, recursively defined as:

$$T(0,k) = \begin{cases} 1 & \text{,if } k = 0; \\ 0 & \text{,otherwise} \end{cases}$$

$$T(L,k) = T(L-1,k-1) + 2T(L-1,k) + T(L-1,k+1), L > 0 \quad (18)$$

Then,

$$p_L(k) = \frac{T(L,k)}{4^L}, L \geq 0, k \in \mathbb{Z} \quad (19)$$

Proof: We prove this by mathematical induction on L.
Base Case: When $L = 0$, $4^L = 1$, and the lemma holds using Equation 9.
Induction Hypothesis: We assume that it holds for $L = t$. That is:

$$p_t(k) = \frac{T(t,k)}{4^t}, k \in \mathbb{Z} \quad (20)$$

Induction Step: We prove that it also holds for $L = t + 1$: Using Theorem 2:

$$\begin{aligned} p_{t+1}(k) &= \frac{1}{4}p_t(k-1) + \frac{1}{2}p_t(k) + \frac{1}{4}p_t(k+1) \quad (21) \\ &= \frac{1}{4}\frac{T(t,k-1)}{4^t} + \frac{1}{2}\frac{T(t,k)}{4^t} + \frac{1}{4}\frac{T(t,k+1)}{4^t} \\ &= \frac{1}{4^{t+1}}(T(t,k-1) + 2T(t,k) + T(t,k+1)) \\ &= \frac{T(t+1,k)}{4^{t+1}} \end{aligned}$$

■

We provided the above corollary in order to present the probability distribution for Δ using a triangle of integers, called Δ-triangle, in a fashion similar to that of Khayyam-Pascal binomial coefficients triangle [30], [31], using dynamic programming. The two-variable function $T(.,.)$ as defined in the above Corollary is presented in Figure 2. Starting from $L = 0$, the Lth row presents $T(L,k)$ for various k as specified underneath the triangle. However, $T(L,k)$ is nonzero only for $-L \leq k \leq +L$. Therefore, there are altogether $2L + 1$ nonzero values for $T(L,k)$, which are shown in the Lth row of the triangle, each corresponding to one value for k. The zero values for $T(L,k)$, i.e. for $|k| > L$ are implicit and not shown; only the nonzero ones make the triangle. Only the first seven rows, for $L = 0$ to $L = 6$, of the triangle are displayed, but it can be extended to subsequent rows following the recursive definition of $T(.,.)$ (see Equation 18); each value is derived from three values of the previous row: the value just above and its just right and just left values. Note that one of these three values is zero, hence outside the triangle, when determining the left-most or the right-most value in a row.
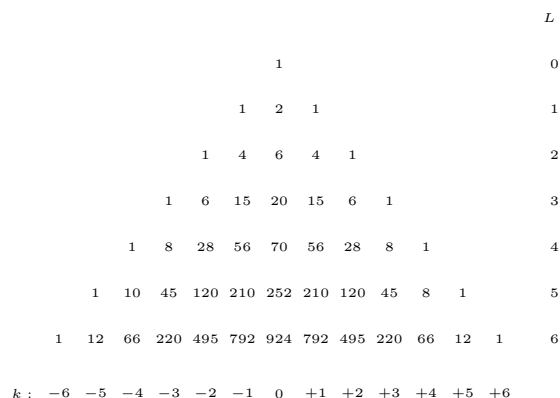
$k:$	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5	+6	$L:$
							1							0
						1	2	1						1
					1	4	6	4	1					2
				1	6	15	20	15	6	1				3
			1	8	28	56	70	56	28	8	1			4
		1	10	45	120	210	252	210	120	45	10	1		5
	1	12	66	220	495	792	924	792	495	220	66	12	1	6

Fig. 2. The first seven rows of the Δ-triangle developed to determine the probability distribution function for the random variable Δ. The value $p_L(k)$, i.e. the probability $Pr_L(\Delta = k)$, is obtained by dividing the integer specified at the row L and the column k by the value 4^L. For example, $p_3(2)$ is $\frac{6}{4^3} \approx 0.09$.

Using the above corollary, the value for $p_L(k)$ can then be obtained by simply retrieving $T(L,k)$ from the Δ-triangle and dividing it by 4^L. Recall that $p_L(k)$ is zero for $|k| > L$. Among interesting properties of the Δ-triangle are (i) the left-most and the right-most numbers in each row are always 1, (ii) the largest number in each row corresponds to $k = 0$, (iii) the triangle is symmetric with respect to the $k = 0$ axis, (iv) the numbers are all integers, and (v) there are $2L + 1$ numbers in the Lth row which sum up exactly to 4^L.

Having derived the probability distribution function for the random variable Δ, the following example, though simple, illustrates how the value of $\widetilde{GpC}(X)$ is calculated for a candidate solution X.

Example 2: Consider the instance $< S, 3 >$ and the candidate solution given in Example 1. That is, $S = \{s_1, s_2, s_3\}$, $s_1 = $ "GATTG", $s_2 = $ "GATCA", $s_3 = $ "CTCGA", and $X = $ "GATTC". Recall that $c_1(X) = 2$, $c_2(X) = 1$, and $c_3(X) = -2$. Therefore, $Near(X) = \{s_1, s_2\}$. Then:

$$\begin{aligned} \widetilde{g_1}(X) &= 1 + Pr_{c_1(X)}(\Delta \geq c_2(X)) + Pr_{c_1(X)}(\Delta \geq c_3(X)) \\ &= 1 + Pr_2(\Delta \geq 1) + Pr_2(\Delta \geq -2) \\ &= 1 + (\frac{5}{16}) + (\frac{16}{16}) \\ &= \frac{37}{16} \end{aligned}$$

Similarly,

$$\widetilde{g_2}(X) = 1 + Pr_{c_2(X)}(\Delta \geq c_1(X)) + Pr_{c_2(X)}(\Delta \geq c_3(X)) = 2$$

Therefore,

$$\widetilde{GpC}(X) = \frac{1}{2}(\frac{\widetilde{g_1}(X)}{c_1(X)} + \frac{\widetilde{g_2}(X)}{c_2(X)}) = \frac{1}{2}(\frac{37}{32} + 2) = \frac{101}{64}$$

V. THE HYBRID HEURISTIC EVALUATION FUNCTION

In this section, a hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}(.)$ is proposed which combines the objective evaluation function $f(.)$ and the estimated Gain-per-Cost heuristic

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

evaluation function $\widetilde{GpC}(.)$. In order to restrict the application of $\widetilde{GpC}(.)$ to the discrimination between candidate solutions with the same objective value, as explained in Section 1, this combination should be designed in such a way that $f(.)$ *dominates* $\widetilde{GpC}(.)$. That is, $\forall X_i \in \Sigma^m$ and $\forall X_j \in \Sigma^m$:

$$f(X_i) > f(X_j) \Rightarrow h_{f,\widetilde{GpC}}(X_i) > h_{f,\widetilde{GpC}}(X_j) \qquad (22)$$

We propose the hybrid heuristic evaluation function as $h_{f,\widetilde{GpC}}(X) = \eta.f(X) + \widetilde{GpC}(X)$, where η is a constant. The following theorem shows that if η is greater than the number of input strings n then the requirement 22 will be met.

Theorem 3: For an arbitrary candidate solution X, $\widetilde{GpC}(X) \le n$.

Proof: Since $|S| = n$, the objective value $f(X)$ is upper bounded by n. This implies that the (estimated) gain of a string, by its definition, with respect to any candidate solution, is also upper bounded by n. On the other hand, the cost of an arbitrary string s_j in $Near(X)$ has to be at least 1. Therefore, the (estimated) gain-per-cost of a string cannot be greater than n either. This means, by Equation 7, that $\widetilde{GpC}(X) \le n$. ∎

Based on this theorem, we use $n+1$ as the value for η, and the heuristic evaluation function will be the following:

$$h_{f,\widetilde{GpC}}(X) = (n+1).f(X) + \widetilde{GpC}(X) \qquad (23)$$

Given a candidate solution X, $h_{f,\widetilde{GpC}}(X)$ can be determined in $O(nm + n^2)$, provided that we pay a memory cost and a one-off time cost of $O(m^2)$ each. We first create a two-dimensional array to keep the Δ-triangle (see Figure 2) which implicitly presents the probability distribution function (PDF) for Δ. Only the first m rows of the triangle are needed, because the length of a walk is not greater than the length of the strings. However, in order to be able to retrieve $Pr_L(\Delta \ge k)$ for each possible k in $O(1)$, as used in Equation 8, we construct another two-dimensional array to store the cumulative distribution function (CDF) for Δ (note that $Pr_L(\Delta \ge k) = 1 - Pr_L(\Delta \le k-1)$). The memory cost to define these arrays and the one-off time cost to populate them with the PDF and CDF values using dynamic programming are $O(m^2)$ each. On the other hand, the costs $c_j(X)$ of the input strings s_j and the set $Near(X)$ can be determined in $O(nm)$. Consequently, retrieving the CDF values in $O(1)$, $\widetilde{GpC}(X)$ can be calculated in $O(nm + n^2)$ by Equations 7 and 8. Therefore, $h_{f,\widetilde{GpC}}(X)$ can be determined using Equation 23 in $O(nm + n^2)$, having paid the memory and the one-off time costs of $O(m^2)$.

VI. THE CBLS ALGORITHM

Prior to presenting the proposed CBLS algorithms, we need to extend our terminology, presented in Section 2, to cater for *partial* solutions used in the beam search. By a partial solution, we mean a solution string of length m whose characters are not all specified. For this purpose, some definitions and notations regarding partial solutions are provided next. For simplicity, and without loss of generality, we assume that the question mark character ? does not belong to the alphabet, and we use it to denote an unspecified character. In other words,

a partial solution is a string of m characters, each either a letter in Σ or unspecified ?, whereas a *complete* solution is a string of m characters, each a letter in Σ. In the rest of the paper, we denote a solution (either complete or partial) by a (possibly indexed) small letter x. By an *unassigned* (respectively *assigned*) *position* of a solution x, we mean an integer k, $1 \le k \le m$, such that $x^k = ?$ (respectively $x^k \neq ?$). The set of unassigned and the set of assigned positions of a solution x are denoted by $U(x)$ and $A(x)$, respectively. Note that $U(x) = \{\}$ in the case x is a complete solution. A solution x is called *empty* if $A(x) = \{\}$. In order to cater for partial solutions, we also need to extend our notations and definition for hamming distance between strings. We extend the definition for the hamming distance between a solution x and an input string s_i as $d_i(x) = \sum_{k \in A(x)} \delta(x^k, s_i^k)$. Note that this is consistent with the general definition for the hamming distance between strings as $d_i(x)$ shrinks to $d_H(x, s_j)$ when x is a complete solution. Let x be a solution, k a position between 1 and m inclusive, and c a character in Σ. The solution obtained by the setting $x^k = c$ is denoted by $x|(k, c)$. Similarly, the Gain-per-Cost heuristic is extended to be used for partial solutions as well, where we use, as for the objective function for partial solutions, the following:

$$f(x) = \sum_{i=1}^{n} min\{d_i(x), d\} \qquad (24)$$

Having defined and extended the required notations and definitions, we now present the CBLS algorithm, shown in Figure 3. As can be seen in the presented pseudo code, the algorithm consists of two main phases of beam search and local search. The beam search algorithm, in its standard form, is a deterministic, yet heuristic, tree search. It is similar to the breath-first search algorithm except that it does not keep all the leaves but only β of them, where $\beta > 0$ is called the beam size. It turns to a pure constructive greedy heuristic in the case $\beta = 1$; it also turns to the exhaustive breath-first search if β is large enough to keep all the leaves. Therefore, the beam size β may be thought of as a control parameter to make an appropriate balance between greediness and exhaustiveness and is usually used to avoid otherwise exponential complexity.

The outcome of the constructive beam search algorithm in the first phase of CBLS is several complete candidate solutions, which are then used as the stating point for local search in the search space. More specifically, the local search algorithm, which is a first-move hill-climbing algorithm, is applied to each of the solutions obtained by the first phase trying to achieve improved solutions, the best of which will be returned as the final solution. The heuristic function used in the beam search algorithm is $h(x) = (n+1)f(x) + \widetilde{GpC}(x)$, where $f(x) = \sum_{i=1}^{n} min\{d_i(x), d\}$ (see equation 24). Similarly, the heuristic function used in the local search algorithm is the hybrid heuristic function of Equation 23, i.e. $h_{f,\widetilde{GpC}}(X) = (n+1).f(X) + \widetilde{GpC}(X)$.

The speed of CBLS depends on the adopted local search algorithm; it can also be controlled by appropriately setting the beam size parameter. Since its first phase is the polynomial-time constructive beam search and its second phase is the hill-

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

Require: $S = \{s_1, s_2, \ldots, s_n\}, n > 1$, each s_i a string of length $m > 1$
Ensure: a string x of the length m
  {initialization:}
  construct the Δ-triangle
  {phase 1: constructive beam search:}
  $B \leftarrow$ {the empty solution}
  for $k = 1$ **to** n **do**
    $C \leftarrow \{\}$
    for all $x_i \in B$ **do**
      for all $ch \in \Sigma$ **do**
        $C \leftarrow C \cup \{x_i | (k, ch)\}$
      end for
    end for
    {calculate heuristics:}
    for all $x_i \in C$ **do**
      calculate heuristic value $h(x_i)$
    end for
    {selection:}
    $B \leftarrow$ a set of β best members of C {with respect to the heuristic values}
  end for
  {phase 2: local search:}
  $bestSoFar \leftarrow$ an arbitrary solution in B
  for all $x_i \in B$ **do**
    $x_i \leftarrow localSearch(x_i)$
    if $f(x_i) > f(bestSoFar)$ **then**
      $bestSoFar \leftarrow x_i$
    end if
  end for
  return $bestSoFar$

Fig. 3. The pseudo code for the algorithm CBLS

climbing local search, it is quite fast in practice, except for too large beam sizes.

VII. EXPERIMENTAL RESULTS

To evaluate the proposed CBLS algorithm, we implemented and compared it with the most recent algorithm published in the literature for the purpose of FFMSP, which is GRASP proposed in [5] as the current state-of-the-art. The algorithms were implemented in Java and run on a Pentium 4 Desktop machine with 3.21 GHz clock speed and 2 GB of RAM. As for the GRASP parameters, we set $RCL-Size$ to 2, which we found the best. However, in order to provide fair comparison, we did not use a fixed value as for the $itr - num$ parameter. Instead, we measured the time taken by CBLS, with $\beta = 10$, and allowed GRASP to run for the same amount of time.

We examined and compared the algorithms on various instances of FFMSP with different sizes. We considered three different values of 100, 200, and 300 as for the number of strings n. For each value v_n for n, we considered three different values of v_n, $2v_n$, and $4v_n$ as for the length of strings m. Finally, for each pair of values $< v_n, v_m >$ for $< n, m >$, we considered three different values of $0.75v_m$, $0.85v_m$, and $0.95v_m$ as for the distance threshold d. These

make $3 \times 3 \times 3 = 27$ different *types* of instances altogether. By the type of an instance in this section, we mean the triple of values for its $< n, m, d >$.

The algorithms were evaluated on both random and real data. For each instance type, we generated 10 random instances, hence $27 \times 10 = 270$ random instances altogether. For real data, we used 3 instances for each type making $27 \times 3 = 81$ real instances altogether. So, the algorithms were examined over the total of $270 + 81 = 351$ instances. The random data was generated by the standard Java pseudo-random number generator. The real data were obtained from the sequence data produced by the US Department of Energy Joint Genome Institute [1] and curated at the Virginia Bioinformatics Institute [2].

The results for both random and real data are presented in Table I. The first column in this table shows the instance types. The second (F_G) and the third (F_C) columns report the objective values of the solutions returned by GRASP and CBLS, respectively, on random instances. The fourth column ($\alpha\%$) calculates the improvement percentage on random instances defined as $\frac{F_C - F_G}{F_C} \times 100$. The next three columns report the same quantities as those of the second to the fourth ones, respectively, but for real instances. In addition, the last row in the table shows the average of improvement percentage for both random and real data. The figures are rounded up to one decimal figure.

As can be seen in Table I, CBLS performs at least as good as GRASP in 100% of the 54 cases. In one random and eight real cases, both of the algorithms achieve the maximum value n, hence zero improvement percentage. In all the other 45 cases, CBLS outperforms GRASP. In 17 out of the 54 cases (i.e. about 30% of cases), where GRASP fails to give a nonzero objective value, the improvement percentage is 100%; CBLS never gives a zero objective value. Even in some of the cases where GRASP gives nonzero objective values, the improvement is by orders of magnitude, e.g. for the (100,100,95) random and real cases. On average, CBLS achieves 62.4% and 53.1% improvemnts over GRASP for random and real instances, respectively. This shows the strong effectiveness of the CBLS algorithm. In general, this efectiveness is more clearly seen in the cases in Table I where GRASP performs poorly giving low objective values.

In order to observe the relative behaviors of the algorithms over longer time, we increased beam size in CBLS. More specifically, we tried the algorithm for beam sizes of 20, 40, 70, and 100 as well. Again, we measured its run time and let GRASP run for the same amount of time for fair comparison. We performed these longer experiments on the random and real instances with $n = 200$, $m = 400$, and $d = 340$, i.e. moderate value for each. As before, we used 10 random and 3 real instances and measured the improvement percentage, α, which are depicted in Figure 4. As can be seen in Figure 4, not much improvement is made as the result of longer execution, and CBLS still keeps its remarkable distance from GRASP in these longer runs.

[1] http://www.jgi.doe.gov
[2] http://www.vbi.vt.edu

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

TABLE I
COMPARISON OF THE (AVERAGE) NUMBER OF FAR STRINGS RETURNED
BY GRASP (F_G) AND CBLS (F_C) OVER BOTH RANDOM AND REAL
INSTANCES

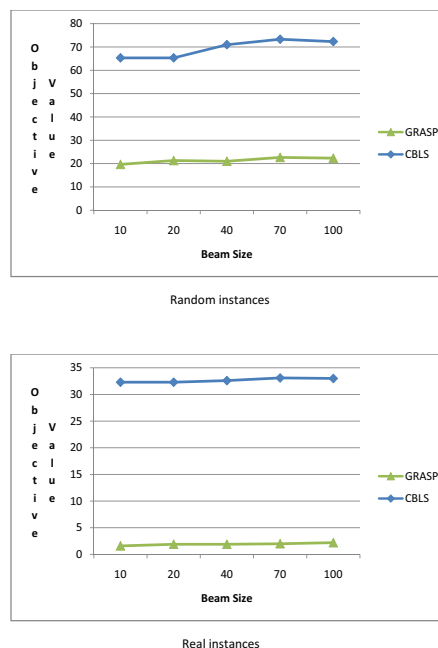| (n,m,d)	Random			Real		
	F_G	F_C	$\alpha\%$	F_G	F_C	$\alpha\%$
(100,100,75)	95.9	100	4.1	100	100	0
(100,100,85)	20.2	30.2	33.2	48	58.7	18.2
(100,100,95)	0.8	7.6	89.5	2	9.4	78.6
(100,200,150)	97.9	100	2.1	100	100	0
(100,200,170)	7.2	28.3	74.6	26.4	49	46.3
(100,200,190)	0	5.7	100	0.4	6.7	95
(100,400,300)	100	100	0	100	100	0
(100,400,340)	2.1	29.5	92.9	17.7	53	66.7
(100,400,380)	0	4.7	100	0	5.7	100
(200,200,150)	186.2	200	7	199.7	200	0.2
(200,200,170)	7	32.1	78.2	48.4	80.4	39.9
(200,200,190)	0	5.4	100	0	7.7	100
(200,400,300)	192.3	200	3.9	200	200	0
(200,400,340)	1.6	32.3	95.1	19.7	65.4	69.9
(200,400,380)	0	3.9	100	0	6	100
(200,800,600)	199.2	200	0.5	200	200	0
(200,800,680)	0	26.5	100	9.4	71.7	87
(200,800,760)	0	3.2	100	0	3.7	100
(300,300,225)	272.1	294.6	7.7	299.7	300	0.2
(300,300,255)	2.9	34.6	91.7	37	93	60.3
(300,300,285)	0	3.8	100	0	4.4	100
(300,600,450)	284.6	300	5.2	300	300	0
(300,600,510)	0.6	27.7	97.9	17	91.7	81.5
(300,600,570)	0	2.3	100	0	1.4	100
(300,1200,900)	296.3	300	1.3	300	300	0
(300,1200,1020)	0	23.9	100	8.7	95.7	91
(300,1200,1140)	0	1.1	100	0	1.4	100
Average			62.4			53.1

Random instances

Real instances

Fig. 4. Comparison of the (average) number of far strings returned by GRASP and CBLS for various beam sizes of 10, 20, 40, 70, 100, over both random (up) and real (down) problem instances of $n = 200$, $m = 400$, and $d = 340$.

even be modified to try on other NP-hard discrete optimization problems in Bioinformatics and other disciplines.

VIII. CONCLUSION

In this paper, a hybrid algorithm called Constructive Beam and Local Search (CBLS) was proposed for the Far From Most Strings Problem, which belongs to the class of sequences consensus problems, and which has applications in Bioinformatics. The proposed algorithms is based on a hybrid of constructive beam search and iterated improvement local search. In fact, the algorithm is obtained by replacing the first phase of the GRASP algorithm with constructive beam search. The algorithm has also been partly inspired from memetic algorithms, where local search is applied to individual solutions obtained, for example, by genetic algorithms. The use of such hybridization of algorithms has recently been increased for a variety of problems in various domains.

The proposed CBLS algorithm was compared with the most recently published algorithm for the problem, GRASP, known as the current state-of-the-art. The comparison was made on both random and real data with significantly positive results; the improvement was by orders of magnitudes on average. Due to its success with respect to FFMSP, a possible avenue for future work is to adapt the proposed algorithm to address other problems of the sequences consensus family. It may

REFERENCES

[1] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, "Distinguishing string selection problems," in *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, pp. 633–642.

[2] J. K. Lanctot, "Some string problems in computational biology," Ph.D. dissertation, Univesity of Waterloo, 2000.

[3] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, "Distinguishing string selection problems," *Inf. Comput.*, vol. 185, no. 1, pp. 41–55, 2003.

[4] C. N. Meneses, C. A. S. Oliveira, and P. M. Pardalos, "Optimization techniques for string selection and comparison problems in genomics," *Engineering in Medicine and Biology Magazine, IEEE*, vol. 24, no. 3, pp. 81–87, May-June 2005.

[5] P. Festa, "On some optimization problems in molecular biology," *Mathematical Biosciences*, vol. 207, no. 2, pp. 219 – 234, 2007, bIOCOMP2005 Special Issue. [Online]. Available: http://www.sciencedirect.com/science/article/B6VHX-4N0X60P-2/2/3a9aa7c9fde1693bef0402f190eb1a36

[6] G. D. Cohen, M. G. Karpovsky, H. F. M. Jr., and J. R. Schatz, "Covering radius - survey and recent results," *IEEE Transactions on Information Theory*, vol. 31, no. 3, pp. 328–343, 1985.

[7] A. J. L. Macario and E. C. de Macario, *Gene probes for bacteria*. Academic Press, 1990.

[8] M. Li, B. Ma, and L. Wang, "Finding similar regions in many strings," in *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1999, pp. 473–482.

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:4, No:8, 2010

[9] J. Gramm, F. Hffner, and R. Niedermeier, "Closest strings, primer design, and motif search," in *Sixth Annual International Conference on Computational Molecular Biology*, April 2002, pp. 74–75.

[10] S. T. Crooke and B. Lebleu, *Antisense Research and Applications*. CRC Press, 1993.

[11] M. Li, B. Ma, and L. Wang, "On the closest string and substring problem," *Journal of the ACM*, vol. 49, no. 2, pp. 157–171, March 2002.

[12] C. N. de Meneses, Z. Lu, C. A. S. Oliveira, and P. M. Pardalos, "Optimal solutions for the closest-string problem via integer programming," *INFORMS Journal on Computing*, vol. 16, no. 4, pp. 419–429, 2004.

[13] J.-C. Chen, "Iterative rounding for the closest string problem," *CoRR*, vol. abs/0705.0561, 2007.

[14] S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusöz, Eds., *Combinatorial Pattern Matching, 15th Annual Symposium, CPM 2004, Istanbul,Turkey, July 5-7, 2004, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3109. Springer, 2004.

[15] B. Ma and X. Sun, "More efficient algorithms for closest string and substring problems," in *Research in Computational Molecular Biology: 12th Annual International Conference RECOMB 2008*, April 2008, pp. 396–409.

[16] J. Wang, J. Chen, and J. M. Huang, "An improved lower bound on approximation algorithms for the closest substring problem," *Information Processing Letters*, vol. 107, no. 1, pp. 24–28, June 2008.

[17] J. Gramm, "Closest substring," in *Encyclopedia of Algorithms*, 2008.

[18] F. C. Gomes, C. N. de Meneses, P. M. Pardalos, and G. V. R. Viana, "A parallel multistart algorithm for the closest string problem," *Computers & OR*, vol. 35, no. 11, pp. 3636–3643, 2008.

[19] C. H. Cheng, C. C. Huang, S. Y. Hu, and K. Chao, "Efficient algorithms for some variants of the farthest string problem," in *21st Workshop on Combinatorial Mathematics and Computation Theory*, may 2004, pp. 266–273.

[20] C. N. Meneses, P. M. Pardalos, M. G. C. Resende, and A. Vazacopoulos, "Modeling and solving string selection problems," in *Second International Symposium on Mathematical and Computational Biology*, December 2005, pp. 54–64.

[21] J. Gramm, J. Guo, and R. Niedermeier, "On exact and approximation algorithms for distinguishing substring selection," in *FCT*, 2003, pp. 195–209.

[22] J. Gramm, R. Niedermeier, and P. Rossmanith, "Fixed-parameter algorithms for closest string and related problems," *Algorithmica*, vol. 37, no. 1, pp. 25–42, 2003.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[24] T. Feo and M. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations Research Letters*, vol. 8, pp. 67–71, 1989.

[25] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[26] P. Festa and M. Resende, "Grasp: An annotated bibliography," in *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2002, pp. 325–367.

[27] P. Festa and M. G. C. Resende, "An annotated bibliography of GRASP; part I: Algorithms," *International Transactions in Operational Research*, vol. 16, no. 1, pp. 1–24, 2009.

[28] M. Babaei and S. R. Mousavi, "A memetic algorithm for closest string problem and farthest string problem," in *ICEE '10: Proceedings of the 18th Iranian Conference on Electrical and Electronics Engineering, to appear*. IEEE, 2010.

[29] S. R. Mousavi, M. Babaei, and M. Montazerian, "An improved heuristic for the far from most strings problem," *submitted for publication*.

[30] H. A. Peelle, "Euclid, fibonacci, and pascal recursed," *International Journal of Mathematical Education in Science and Technology*, vol. 6, no. 4, pp. 395–405, November 1975.

[31] A. W. F. Edwards, *Pascal's Arithmetical Triangle: The Story of a Mathematical Idea*. Johns Hopkins University Press, 2002.