J Heuristics (2012) 18:239–262
DOI 10.1007/s10732-011-9177-z

An improved heuristic for the far from most strings problem

Sayyed Rasoul Mousavi · Maryam Babaie · Manizheh Montazerian

Received: 1 August 2009 / Accepted: 1 June 2011 / Published online: 9 August 2011
© Springer Science+Business Media, LLC 2011

Abstract The Far From Most Strings Problem (FFMSP) asks for a string that 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 far if their Hamming distance is greater than or equal to a given threshold. FFMSP belongs to the class of sequence consensus problems which have applications in molecular biology, amongst others. FFMSP is NP-hard. It does not admit a constant-ratio approximation either, unless $P = NP$. In the last few years, heuristic and metaheuristic algorithms have been proposed for the problem, which use local search and require a heuristic, also called an evaluation function, to evaluate candidate solutions during local search. The heuristic function used, for this purpose, in these algorithms is the problem's objective function. However, since many candidate solutions can be of the same objective value, the resulting search landscape includes many points which correspond to local maxima. In this paper, we devise a new heuristic function to evaluate candidate solutions. We then incorporate the proposed heuristic function within a Greedy Randomized Adaptive Search Procedure (GRASP), a metaheuristic originally proposed for the problem by Festa. The resulting algorithm outperforms state-of-the-art with respect to solution quality, in some cases by orders of magnitude, on both random and real data in our experiments. The results indicate that the number of local optima is considerably reduced using the proposed heuristic.

S.R. Mousavi (✉) · M. Babaie
Isfahan University of Technology, Isfahan 84156-83111, Iran
e-mail: srm@cc.iut.ac.ir

M. Babaie
e-mail: m.babaie@ec.iut.ac.ir

M. Montazerian
Birkbeck College, University of London, London WC1E 7HX, UK
e-mail: gmont05@dcs.bbk.ac.uk

Keywords Bioinformatics · Evaluation functions · Far from most strings problem · Heuristic functions · Local optima · Matheuristic · Metaheuristics · Sequence consensus

1 Introduction

The Far From Most Strings Problem (FFMSP) is a combinatorial optimization problem that receives, as input, a set S of strings of the same length m over an alphabet and a positive integer $d \leq m$ and asks for a string of length m over the alphabet that is *far* from as many strings in S as possible (Lanctot et al. 1999, 2003; Lanctot 2000; Meneses et al. 2005; Festa 2007). In other words, it is to maximize the number of input strings, i.e. those in S, that are far from the output string. Two strings are far (from each other) if their Hamming distance is equal to or greater than the input integer d. FFMSP belongs to a more general class of problems called *sequence consensus*, which includes such problems as finding similar regions in a given set of DNA or protein sequences and have applications in bioinformatics, among others (Cohen and Karpovsky 1985; Macario and de Macario 1990; Li et al. 1999; Lanctot 2000; Gramm et al. 2002; Crooke and Lebleu 1993). Other sequence consensus problems include Closest String Problem (CSP) (Li et al. 2002; de Meneses et al. 2004; Chen 2007; Babaie and Mousavi 2010), Closest Substring Problem (CSSP) (Sahinalp et al. 2004; Ma and Sun 2008; Wang et al. 2008; Gramm 2008; Gomes et al. 2008), Farthest String Problem (FSP) (Cheng et al. 2004; Meneses et al. 2005), Farthest Substring Problem (FSSP) (Lanctot 2000; Meneses et al. 2005), Close to Most Strings Problem (CMSP) (Lanctot et al. 2003; Meneses et al. 2005), Distinguishing Substring Selection Problem (DSSS) (Gramm et al. 2003), and Distinguishing String Selection Problem (DSSP) (Lanctot et al. 2003; Meneses et al. 2005; Gramm et al. 2003), which are also known as string selection and comparison problems.

FFMSP has been proved NP-hard (Lanctot et al. 1999, 2003), and no polynomial-time algorithm is (currently) known to be able to optimally solve every instance of FFMSP (Garey and Johnson 1979). It does not admit a constant-ratio approximation algorithm either, unless $P = NP$ (Lanctot et al. 1999, 2003).

In the recent years, heuristic and metaheuristic algorithms have been proposed for the problem. Meneses et al. devised a heuristic algorithm consisting of a greedy constructive phase followed by an iterative improvement phase (Meneses et al. 2005). Festa proposed a Greedy Randomized Adaptive Search Procedure (GRASP) and reported improved results over the Meneses et al.'s algorithm (2007). The GRASP algorithm was originally devised by Feo and Resende (1989, 1995). It comprises of a number of iterations, each of which consists of a constructive phase followed by a local search phase which are, respectively, similar to the constructive and iterated improvement phases in the Meneses et al.'s algorithm. The construction phase constructs a candidate solution by specifying the values of the variables one at a time. The variable-value selection is based on a heuristic function; but not always the best option is selected. Instead, a *restricted candidate list (RCL)* of options is developed from which a candidate is randomly selected. The local search phase receives, as input, the candidate solution constructed by the construction phase and tries to improve

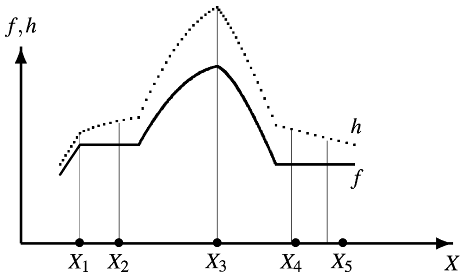

Fig. 1 Parts of landscapes corresponding to an objective function $f(.)$ (*solid line*) and a heuristic function $h(.)$ (*dotted line*) used to evaluate one-dimensional candidate solutions. The points X_1, X_2, X_4, and X_5 correspond to local maxima in the landscape for $f(.)$. As a consequence, a hill-climbing algorithm can get stuck at any of them if $f(.)$ is used, whereas it will move on toward the global maximum if $h(.)$ is used

it via, for example, an iterated improvement algorithm such as hill climbing. The best candidate solution found over all the iterations is returned as the final output of GRASP. For more details on GRASP and its variants, the interested reader is referred to the annotated bibliography by Festa and Resende (2002, 2009).

Every local search algorithm to-date uses a heuristic function to evaluate and compare candidate solutions. Such a heuristic function is also called an evaluation function (Russel and Norvig 2003) and, especially in the context of Genetic Algorithms, a fitness function (Fogel 2006). Both the Meneses et al.'s heuristic and the Festa's metaheuristic algorithms use in their local search phase the problem's objective function $f(.)$ for this purpose. For example, in the local search phase of the Festa's GRASP, which is a hill-climbing algorithm, a move from the current candidate solution X_1 to one of its neighbors X_2 is made only[1] if $f(X_2) > f(X_1)$. The use of the problem's objective function to evaluate candidate solutions is quite common in local search and metaheuristic algorithms for many combinatorial optimization problems. However, depending on the underlying problem, the resulting search landscape may include many points corresponding to local optima. Consequently, a hill climbing algorithm, for example, is likely to stop earlier than expected at a local optimum missing the opportunity to explore enough of the search space. Even other more sophisticated algorithms such as Evolutionary Algorithms (Ashlock 2006) and Tabu Search (Glover and Laguna 1997) which have provisions to escape local optima still suffer from the existence of (too many) local optima.

We try to illustrate this issue using a rather simple landscape depicted in Fig. 1, for a one-dimensional variable. This landscape does not of course apply to FFMSP and is simply for the purpose of illustration. Assume that two candidate solutions are neighbors if and only if they are adjacent on the X-axis. Both the objective function $f(.)$ and a heuristic function $h(.)$ are shown in the figure, the former by solid and the latter by dotted lines. Consider first the landscape corresponding to $f(.)$. The points, i.e. candidate solutions, X_1, X_2, X_4, and X_5 correspond to local maxima, some of which corresponding to plateaux, in this landscape. The point X_3 corresponds to the

[1]To accept a move also depends on whether the first-improvement or the best-improvement scheme, or a hybrid of them, is adopted.

global maximum, hence an optimal solution. Now consider the landscape which corresponds to the heuristic function $h(.)$. The points X_1 and X_2 are no longer evaluated the same; X_2 is more likely than X_1 to be near better solutions and is, therefore, given a higher score than that of X_1. Similarly, the score given to X_4 is higher than the score given to X_5. Moreover, for all pairs of points $\langle X_i, X_j\rangle$, if $f(X_i) > f(X_j)$ then $h(X_i) > h(X_j)$. As the result, the points X_1, X_2, X_4, and X_5 which were local maximum points in the landscape for $f(.)$ are no longer so in the landscape for $h(.)$. Consequently, a hill-climbing algorithm, for example, which could get stuck at any of these points, when $f(.)$ used, can now move on toward the global optimum due to the use of $h(.)$.

We believe that the landscape for FFMSP based on using the objective function $f(.)$ to evaluate candidate solutions contains many local maximum points, because there are altogether $|\Sigma|^m$ points in the search space with only n different objective values, where Σ is the alphabet and n and m are, respectively, the number and the length of the input strings. Therefore, we believe, that the search landscape could be improved by replacing the objective function used to evaluate candidate solutions with a more appropriate heuristic function for this purpose. In this paper, we first devise a novel heuristic function called estimated Gain-per-Cost ($\widetilde{GpC}$) to evaluate candidate solutions based on their *likelihood* to lead to better solutions with as few changes as possible. Of course, we do not intend to measure this likelihood precisely but to provide a very rough estimate of that, hence proposing a *heuristic* for this purpose. We then propose a hybrid heuristic function $h_{f,\widetilde{GpC}}(.)$ which takes into account both the objective function $f(.)$ and the estimated Gain-per-Cost heuristic function $\widetilde{GpC}(.)$ in such a way that, given two candidate solutions X_1 and X_2, X_1 is evaluated as to be better than X_2 if and only if either $f(X_1) > f(X_2)$, or $f(X_1) = f(X_2)$ and X_1 has a higher estimated Gain-per-Cost value than X_2. Experimental results indicate that the number of local maximum points is reduced remarkably, usually by an order of magnitude, due to using $h_{f,\widetilde{GpC}}$, as opposed to $f(.)$, to evaluate candidate solutions.

In order to evaluate the proposed heuristic function $h_{f,\widetilde{GpC}}(.)$, as opposed to the objective function $f(.)$ used as the heuristic function to evaluate candidate solutions, we compare two versions of GRASP, one using $f(.)$ as the heuristic function, i.e. the state-of-the-art (Festa 2007), and the other using $h_{f,\widetilde{GpC}}(.)$. Although the previous works (Meneses et al. 2005; Festa 2007) restrict their experiments to random data, we compare the algorithms on both random and real data. The results indicate that the new GRASP, based on $h_{f,\widetilde{GpC}}$, outperforms the state-of-the-art in *all* cases, with respect to solution quality, in some by orders of magnitude.

The rest of the paper is organized as follows. The next section provides problem definition and basic notations. In Sect. 3, the estimated Gain-per-Cost heuristic is devised. How to determine the estimated Gain-per-Cost heuristic values is addressed in Sect. 4. Section 5 proposes the hybrid heuristic evaluation function and presents the pseudo-code for the algorithm. Experimental results are reported in Sect. 6; Sect. 7 concludes the paper.

2 Problem definition and basic notations

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

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

The Far From Most Strings Problem is formally defined as follows:

FFMSP:

Instance: a triple $\langle \Sigma, S, d \rangle$, where Σ, called alphabet, is a set of characters, S is a set of strings $s_1, s_2, \ldots, s_n$, $n > 1$, of length $m > 1$ over the alphabet Σ, $|\Sigma| > 1$, and d is an integer, called distance threshold, such that $1 \leq d \leq m$.
Output: a string X of length m over the alphabet Σ.
Maximize: the number of strings $s_j \in S$ such that $d_H(X, s_j) \geq d$.

In the rest of the paper, we assume that the underlying instance of FFMSP is denoted by the triple $\langle \Sigma, S, d \rangle$. We further 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 is the number of the input strings. We use m to denote the length of the strings and (possibly indexed) X to denote a candidate solution. 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. For simplicity, we use $d_j(X)$ to denote $d_H(X, s_j)$, $j = 1, 2, \ldots, n$. We say that a string s_j is *far* from a candidate solution X if $d_j(X) \geq d$; it is otherwise *near* X. The set of strings in S that are near X is denoted as *Near*(X). We define the cost of a string s_j as $c_j(X) = d - d_j(X)$. We also define

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

Therefore,

$$f(X) = \sum_{j=1}^{n} f_j(X) \tag{2}$$

Example 1 Let $\langle \Sigma, S, 3 \rangle$ be an instance of FFMSP, where $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$, $S = \{s_1, s_2, s_3\}$, $s_1 =$ "GACTC", $s_2 =$ "GATCA", $s_3 =$ "CTAGA". Let $X =$ "GATTC" be a candidate solution. 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 $c_3(X) = -2$. Therefore, the input strings s_1 and s_2 are near X, whereas s_3 is far from X, which means $f_1(X) = f_2(X) = 0$ but $f_3(X) = 1$, hence *Near*$(X) = \{s_1, s_2\}$.

Each candidate solution X_i can be viewed of as a *point* in the m-dimensional search space Σ^m. We use *Neighbors*(X_i), where X_i is a point, to denote the set of points defined as to be the neighbors of X_i. One convenient way of defining neighborhood, for example, is to consider two points as neighbors if and only if their Hamming distance is one, in which case:

$$Neighbors(X_i) = \{X_j | X_j \in \Sigma^m, d_H(X_i, X_j) = 1\}, \quad X_i \in \Sigma^m$$

In order to disambiguate the heuristic function used to evaluate candidate solutions with, for example, those used in greedy constructive algorithms which are capable of evaluating incomplete variable assignments, in the rest of the paper, as in Russel and Norvig (2003), we use the term *evaluation function* to refer to a function used to evaluate candidate solutions. If the objective function $f(.)$ is used for this purpose, we will call it the *objective evaluation function*. Otherwise, i.e. when a heuristic function other than $f(.)$ is used for this purpose, it will be called a *heuristic evaluation function*.

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_s in the search space is the replacement of X_s with a point X_d such that $d_H(X_s, X_d) = L$. We call L the *length* and the points X_s and X_d, respectively, the *source* and the *destination* of the walk. We refer to the case $L = 0$ as *no-walk*, where $X_s = X_d$. An L-walk is called *random* if its destination is equally-likely to be any point X_j in the search space such that $d_H(X_s, X_j) = L$, where X_s is the source of the walk.

Let X_s and X_d be, respectively, the source and the destination of an L-walk and s_j be an input string. We define Δ_j for this L-walk as $d_j(X_d) - d_j(X_s)$. If the walk is random, Δ_j will be a random variable, in which case we use $Pr_L(\Delta_j = k)$, $k \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of proper numbers, to denote the probability of $\Delta_j = k$. Similarly, we use, for example, $Pr_L(\Delta_j \leq k)$, $k \in \mathbb{Z}$, to denote the probability of $\Delta_j \leq k$ as the result of a random L-walk. We reserve $Pr(.)$—with no subscript—and $Ex(.)$ to denote the conventional probability and expectation functions respectively.

3 The proposed estimated gain per cost heuristic

In this section, the proposed *Estimated Gain per Cost* ($\widetilde{GpC}$) heuristic evaluation function is introduced. Some definitions and theorems are first presented.

Definition 1 Let X_s be a candidate solution and s_j be a string in $Near(X_s)$. By a *fix* for s_j from X_s, we mean an L-walk whose source and destination are, respectively, X_s and X_d such that $d_j(X_d) = d$. We assume that the walk is designed independently from the other input strings so it can be viewed as a random L-walk with respect to them.

Informally-speaking, a fix for a string s_j from the current candidate solution is to alter exactly L characters of X, where $L = d - d_j(X)$, in such a way that s_j becomes far from the resulting candidate solution X_d in order to contribute 1 unit to the (new) objective value $f(X_d)$. However, because the walk is random with respect to the other strings, $f(X_d)$ is still a random variable, for $n > 1$.

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

The next theorem shows that the gain of a string s_j can be calculated using the probability distribution for Δ_j for the underlying L-walk.

Theorem 1 *Let X_s be a candidate solution and s_j a string in $Near(X_s)$. Then*:

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

where $L = c_j(X_s)$.

A lemma is first presented.

Lemma 1 *Let X_s be a candidate solution and s_k be an input string. The probability of the string s_k being far from the destination X_d of a random L-walk from X_s is equal to $Pr_L(\Delta_k \geq c_k(X_s))$.*

Proof The probability of s_k being far from X_d is:

$$\begin{aligned} Pr(f_k(X_d) = 1) &= Pr(d_k(X_d) \geq d) \\ &= Pr(d_k(X_d) - d_k(X_s) \geq d - d_k(X_s)) \\ &= Pr_L(\Delta_k \geq d - d_k(X_s)) \\ &= Pr_L(\Delta_k \geq c_k(X_s)) \qquad (3) \end{aligned}$$

□

Proof of Theorem 1

$$\begin{aligned} g_j(X_s) &= Ex(f(X_d)) \quad \text{(using Definition 2)} \qquad (4) \\ &= Ex\left(\sum_{k=1}^{n} f_k(X_d)\right) \quad \text{(using (2))} \qquad (5) \\ &= Ex\left(f_j(X_d) + \sum_{\substack{k=1 \\ k\neq j}}^{n} f_k(X_d)\right) \\ &= Ex\left(1 + \sum_{\substack{k=1 \\ k\neq j}}^{n} f_k(X_d)\right) \quad \text{(since the walk is a fix for } s_j \text{ from } X_s\text{)} \qquad (6) \end{aligned}$$

$$= 1 + Ex\left(\sum_{\substack{k=1\\k\neq j}}^{n} f_k(X_d)\right)$$

$$= 1 + \sum_{\substack{k=1\\k\neq j}}^{n} Ex(f_k(X_d))$$

$$= 1 + \sum_{\substack{k=1\\k\neq j}}^{n} Pr(f_k(X_d) = 1) \quad (\text{since } f_k(X_d) \text{ is either 0 or 1}) \tag{7}$$

$$= 1 + \sum_{\substack{k=1\\k\neq j}}^{n} Pr_{c_j(X_s)}(\Delta_k \geq c_k(X_s)) \quad (\text{by Lemma 1}) \tag{8}$$

□

Definition 3 Let X be a candidate solution and s_j 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 $GpC(X)$ is defined as the average of the gain per costs of the strings in $Near(X)$:

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

It is defined to be zero if $|Near(X)| = 0$.

The gain-per-cost of a string s_j in $Near(X)$, as defined above, is directly proportional to its gain but 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 too. Because the Gain-per-Cost of X is the average of such gain-per-cost values, over strings in $Near(X)$, 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 with the Gain-per-Cost of candidate solutions as defined by Definition 3 is that its calculation requires the probability distribution function of Δ_k for all possible lenghts of random walks, as stated by Theorem 1, which can be different for different strings s_k in S. Therefore, to reduce the required computational cost, we do not propose to determine all these probability distribution functions, but approximate them with the probability distribution function for a unique random variable Δ (with no index) which corresponds to a random string as a representative of all the strings in S. The variable Δ, with respect to a random L-walk, is then defined as $d_H(s, X_d) - d_H(s, X_s)$, where s is a random, as opposed to an input, string and X_s and X_d are the source and destination of the random L-walk, respectively. As confirmed by the experimental results, the use of the heuristic evaluation function based on this approximation still improves significantly the state-of-the-art. The proposed

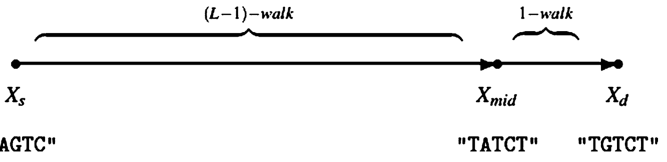

Fig. 2 An L-walk, $L > 0$, may be considered as an $(L-1)$-walk followed by a 1-walk. Sample source and destination points for $L = 4$ are displayed at the *right*

estimated Gain-per-Cost it $\widetilde{GpC}$) heuristic evaluation function is formally defined as follows (unless $|Near(X)| = 0$ in which case it is defined as 0):

$$\widetilde{GpC}(X) = \frac{1}{|Near(X)|} \sum_{s_j \in Near(X)} \frac{\tilde{g}_j(X)}{c_j(X)} \tag{9}$$

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

$$\tilde{g}_j(X) = 1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} Pr_{c_j(X)}(\Delta \geq c_k(X)) \tag{10}$$

The next section proposes a method, using dynamic programming, to efficiently determine the probability distribution function for the random variable Δ.

4 Probability distribution function for Δ

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

$$p_0(k) = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The following theorem provides a recursion to determine the probability distribution for Δ, for $L > 0$.

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

$$p_L(k) = \frac{|\Sigma| - 2}{|\Sigma|} p_{L-1}(k) + \frac{1}{|\Sigma|}(p_{L-1}(k-1) + p_{L-1}(k+1))$$

Proof Because $L > 0$, the random L-walk can be viewed as a random $(L-1)$-walk followed by a random 1-walk with the constraint that the 1-walk does not change any character already changed by the $(L-1)$-walk. Let X_{mid} be the destination of the $(L-1)$-walk, which is the source of the subsequent random 1-walk (see Fig. 2). Further assume that k is the index of the character changed by the 1-walk, hence X_{mid} and X_d are only different in their kth characters, where $1 \le k \le m$. This implies that the kth characters of X_s and X_{mid} will have to be the same. Therefore, exactly one of the following three cases must hold:

(i) $s^k = X^k_{mid}$ and $s^k \neq X^k_d$
(ii) $s^k \neq X^k_{mid}$ and $s^k = X^k_d$
(iii) $s^k \neq X^k_{mid}$ and $s^k \neq X^k_d$

In case (i), the kth character of s is the same as the kth character of X_{mid} but different from the kth character of X_d, hence $d_H(s, X_d) = d_H(s, X_{mid}) + 1$. Similarly, in case (ii), the kth character of s is the same as the kth character of X_d but different from the kth character of X_{mid}, which means $d_H(s, X_d) = d_H(s, X_{mid}) - 1$. Finally, in case (iii), the kth character of s is different from those of X_{mid} and X_d, which implies $d_H(s, X_d) = d_H(s, X_{mid})$. Therefore, in any case,

$$-1 \le d_H(s, X_d) - d_H(s, X_{mid}) \le 1 \tag{12}$$

The probability of case (i) is $\frac{1}{|\Sigma|}$ (because for any possible value of X^k_{mid}, s^k can equally-likely take $|\Sigma|$ values one of which meets the requirement $s^k = X^k_{mid}$). Similarly, the probability of case (ii) is $\frac{1}{|\Sigma|}$ (for any possible value of X^k_d, s^k can equally-likely be any of the $|\Sigma|$ values one of which satisfying $s^k = X^k_d$). Consequently, the probability of case (iii), is $1 - (\frac{1}{|\Sigma|} + \frac{1}{|\Sigma|}) = \frac{|\Sigma|-2}{|\Sigma|}$.

By definition of Δ,

$$\begin{aligned} p_L(k) &= Pr_L(d_H(s, X_d) - d_H(s, X_s) = k) \\ &= Pr_L((d_H(s, X_d) - d_H(s, X_{mid})) + (d_H(s, X_{mid}) - d_H(s, X_s)) = k) \end{aligned}$$

Let $D_1 = (d_H(s, X_d) - d_H(s, X_{mid}))$ and $D_{L-1} = (d_H(s, X_{mid}) - d_H(s, X_s))$, respectively. Then:

$$p_L(k) = Pr(D_1 + D_{L-1} = k)$$

From inequation (12), we know that $-1 \le D_1 \le 1$. Therefore,

$$\begin{aligned} p_L(k) = Pr((D_1 = -1 \ \wedge \ D_{L-1} = k+1) \\ \vee (D_1 = 0 \wedge D_{L-1} = k) \\ \vee (D_1 = 1 \wedge D_{L-1} = k-1)) \end{aligned}$$

Because the three disjoint cases in the right-hand side of the equation are mutually exclusive, we conclude

$$\begin{aligned} p_L(k) = Pr(D_1 = -1 \wedge D_{L-1} = k+1) \\ + Pr(D_1 = 0 \wedge D_{L-1} = k) \\ + Pr(D_1 = 1 \wedge D_{L-1} = k-1) \end{aligned}$$

Similarly, because D_1 can take any of the values -1, 0, or 1 independently from the value of D_{L-1}, the conjoint conditions in each parenthesis in the right-hand side are independent. Therefore:

$$\begin{aligned} p_L(k) &= Pr(D_1 = -1) \times Pr(D_{L-1} = k + 1) \\ &\quad + Pr(D_1 = 0) \times Pr(D_{L-1} = k) \\ &\quad + Pr(D_1 = 1) \times Pr(D_{L-1} = k - 1) \\ &= Pr(D_1 = -1) \times p_{L-1}(k + 1) \\ &\quad + Pr(D_1 = 0) \times p_{L-1}(k) \\ &\quad + Pr(D_1 = 1) \times p_{L-1}(k - 1) \end{aligned}$$

Finally, the probabilities $Pr(D_1 = -1)$, $Pr(D_1 = 0)$, and $Pr(D_1 = 1)$ are equal to, respectively, the probabilities of the cases (i), (iii), and (ii), which are, respectively $\frac{1}{|\Sigma|}$, $\frac{|\Sigma|-2}{|\Sigma|}$, and $\frac{1}{|\Sigma|}$. We conclude

$$p_L(k) = \frac{|\Sigma| - 2}{|\Sigma|} p_{L-1}(k) + \frac{1}{|\Sigma|}(p_{L-1}(k - 1) + p_{L-1}(k + 1)) \qquad \square$$

Note that $p_L(k)$ is zero for $|k| > L$ because, as a result of an L-walk, exactly L characters are only changed.

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

$$\begin{aligned} &T(0, k) = \begin{cases} 1 & \textit{if } k = 0 \\ 0 & \textit{otherwise} \end{cases} \\ &T(L, k) = T(L - 1, k - 1) + (|\Sigma| - 2)T(L - 1, k) + T(L - 1, k + 1), \\ &\quad L > 0 \end{aligned} \tag{13}$$

Then,

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

Proof We use mathematical induction on L.

Base case: For the case $L = 0$, $|\Sigma|^L = 1$, and the lemma trivially holds using (11).
Induction hypothesis: We assume that the lemma holds for $L = t$. That is:

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

Induction step: We now prove that it also holds for $L = t + 1$.

Using Theorem 2:

$$p_{t+1}(k) = \frac{1}{|\Sigma|} p_t(k - 1) + \left(\frac{|\Sigma| - 2}{|\Sigma|}\right) p_t(k) + \frac{1}{|\Sigma|} p_t(k + 1)$$

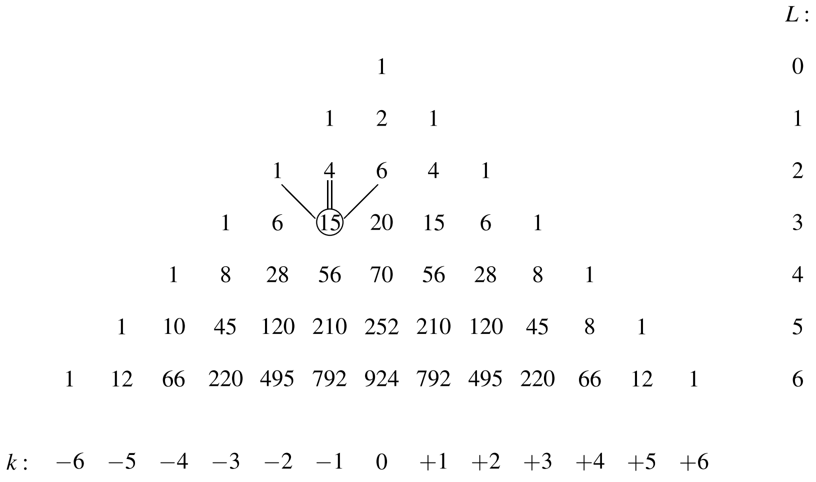

Fig. 3 The *first seven rows* of the Δ-triangle developed to determine the probability distribution function for the random variable Δ, for the case $|\Sigma| = 4$.The value at the top of the triangle, i.e. for $L = 0$, is 1. Every other value is recursively obtained from three values in its *above row*, as illustrated for the value 15 (see (13)). In the case any of these three values is outside the triangle, 0 will instead be used. 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 $|\Sigma|^L$, i.e. by 4^L here. For example, $p_3(2)$ is $\frac{6}{4^3} \approx 0.09$

$$= \frac{1}{|\Sigma|}\frac{T(t,k-1)}{|\Sigma|^t} + \left(\frac{|\Sigma|-2}{|\Sigma|}\right)\frac{T(t,k)}{|\Sigma|^t} + \frac{1}{|\Sigma|}\frac{T(t,k+1)}{|\Sigma|^t}$$

(by the hypothesis)

$$= \frac{1}{|\Sigma|^{t+1}}(T(t,k-1) + (|\Sigma|-2)T(t,k) + T(t,k+1))$$

$$= \frac{T(t+1,k)}{|\Sigma|^{t+1}} \qquad \square$$

The above corollary enables us to use a two-dimensional array to present the probability distribution for Δ. If we only keep nonzero values, the array can be presented as a triangle of integers, called Δ-triangle, which can be efficiently determined using dynamic programming, in a fashion similar to that of Khayyam-Pascal (binomial coefficients) triangle (Peelle 1975; Edwards 2002). The first seven rows of the triangle, for the walk lengths of 0 to 6, is presented in Fig. 3 for the case $|\Sigma| = 4$. Starting from $L = 0$, the Lth row corresponds to $T(L,k)$ for different value of k as specified underneath the triangle. Note that $T(L,k)$ is nonzero only for $-L \le k \le +L$ (and all the zero values are outside the triangle). Therefore, there are a total of $2L + 1$ nonzero values for $T(L,k)$, in the Lth row of the triangle, each of which corresponding to one value of k. Based on the recursive definition of $T(.,.)$ (see (13)), the value of each entry in the triangle is efficiently calculated from three values of the previous row, which are the value just above and the values just above right and above left (note that some of these values may be outside the triangle, in which case they are zero).

Having constructed the triangle, and by the above corollary, the probability value $p_L(k)$ is simply determined by retrieving $T(L,k)$ from the triangle and dividing it by the value $|\Sigma|^L$. The Δ-triangle possesses several interesting properties, including: (i) the values on both sides of the triangle are always 1, (ii) the greatest value in each row is in the middle, corresponding to $k=0$, (iii) the triangle is symmetric with respect to the vertical axis $k=0$, (iv) the values are all integers, and (v) there are exactly $2L+1$ values in the Lth row, whose summation is $|\Sigma|^L$.

The following example, though simple, illustrates how the triangle can be used to calculate a heuristic value $\widetilde{GpC}(X)$ for a candidate solution X.

Example 2 Consider the instance $\langle \Sigma, S, 3\rangle$ and the candidate solution of Example 1. That is, $\Sigma=\{\mathtt{A},\mathtt{T},\mathtt{C},\mathtt{G}\}$, $S=\{s_1,s_2,s_3\}$, $s_1=$ "GACTC", $s_2=$ "GATCA", $s_3=$ "CTAGA", and $X=$ "GATTC". Recall that $c_1(X)=2$, $c_2(X)=1$, and $c_3(X)=-2$, and $Near(X)=\{s_1,s_2\}$. Therefore,

$$\begin{aligned}
\widetilde{g_1}(X) &= 1+Pr_{c_1(X)}(\Delta \geq c_2(X)) + Pr_{c_1(X)}(\Delta \geq c_3(X)) \\
&= 1+Pr_2(\Delta \geq 1)+Pr_2(\Delta \geq -2) \\
&= 1+(p_2(1)+p_2(2))+(p_2(-2)+p_2(-1)+p_2(0)+p_2(1)+p_2(2)) \\
&= 1+\left(\frac{5}{16}\right)+\left(\frac{16}{16}\right) \\
&= \frac{37}{16}
\end{aligned}$$

Similarly we have:

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

We conclude

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

5 The hybrid heuristic evaluation function and the final algorithm

In this section, we first propose a hybrid heuristic evaluation function and then present the pseudo-code for the resulting algorithm. The proposed hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}(.)$ is a combination of the objective evaluation function $f(.)$ and the proposed estimated Gain-per-Cost heuristic evaluation function $\widetilde{GpC}(.)$ such that $f(.)$ *dominates* $\widetilde{GpC}(.)$. More specifically, the following requirement is to be met:

$$\forall X_i \in \Sigma^m \forall X_j \in \Sigma^m, \quad f(X_i) > f(X_j) \quad \Rightarrow \quad h_{f,\widetilde{GpC}}(X_i) > h_{f,\widetilde{GpC}}(X_j) \tag{14}$$

The following theorem indicates that, in order to meet the above requirement, we can use the hybrid heuristic evaluation function as $h_{f,\widetilde{GpC}}(X)=\eta.f(X)+\widetilde{GpC}(X)$, where η is a constant is a combination of greater than the number of input strings.

Theorem 3 *Let X be a candidate solution. Then $\widetilde{GpC}(X) \leq n$.*

Proof Because the estimated gain of a string is the summation of 1 and $n-1$ probability values and that a probability value cannot be greater than 1, the estimated gain of a string is upper bounded by n. On the other hand, the cost of an arbitrary string s_j in $Near(X)$ is at least 1, which means that the (estimated) gain-per-cost of a string is also upper bounded by n. Therefore this means, by (9), $\widetilde{GpC}(X) \leq n$. □

Based on this theorem, we propose the following hybrid heuristic evaluation function:

$$h_{f,\widetilde{GpC}}(X) = (n+1).f(X) + \widetilde{GpC}(X) \tag{15}$$

Corollary 2 *Let N_1 be the number of local maximum points in the search landscape corresponding to $f(.)$, and $N_{\widetilde{GpC}}$ be the number of local maximum points in the search landscape corresponding to $h_{f,\widetilde{GpC}}(.)$ that are not optimal solution to the problem. Then*

$$N_{\widetilde{GpC}} \leq N_1$$

Proof It is enough to show that if an arbitrary point X_i is not a local or a global maximum point in the landscape for $f(.)$, it will not be a local maximum point in the landscape for $h_{f,\widetilde{GpC}}$ either. Suppose that a point $X_i \in \Sigma^m$ is not a local or a global maximum point in the landscape for $f(.)$. Then, $\exists X_j \in Neighbors(X_i)$, $f(X_j) > f(X_i)$. Since the heuristic evaluation function $h_{f,\widetilde{GpC}}$ satisfies the property 14, this implies:

$$\exists X_j \in Neighbors(X_i), \quad h_{f,\widetilde{GpC}}(X_j) > h_{f,\widetilde{GpC}}(X_i)$$

which means that X_i cannot be a local maximum point in the search landscape for $h_{f,\widetilde{GpC}}$. □

Informally speaking, this corollary indicates that replacing the objective evaluation function $f(.)$ with the heuristic evaluation function $h_{f,\widetilde{GpC}}$ does not worsen the landscape with respect to the number of local maximum points; the only possible new local maximum points will be those that are optimal solutions to the problem, getting stuck at which means achieving an optimal solution. In fact, as reported in the next section, the use of $h_{f,\widetilde{GpC}}(.)$ as opposed to $f(.)$ is quite promising in reducing the number of local maximum points.

We now present a high level pseudo-code for a GRASP algorithm `GRASP-FFMSP` for FFMSP, followed by a more detailed pseudo-code for the proposed hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}(.)$. Figure 4 presents the pseudo-code for `GRASP-FFMSP`. As a GRASP, it executes a pair of construction and local search phases a number of times, specified by the parameter *ItrNum*; the best solution *BestSoFarX* found over all the iterations is returned as the final output.

In the construction phase, a candidate solution X is incrementally built, starting from a null string and appending to it one alphabet character at a time. As can be seen in the algorithm (Fig. 4), the outer `for` loop in the construction phase iterates

m times, each to determine one of the characters of X. To decide on which alphabet character to choose, all possible $|\Sigma|$ characters are examined, in the corresponding inner `for` loop, using some heuristic function. Here, as in Festa (2007), the heuristic value given to a character c as a candidate for X^i is the number of input strings whose ith characters differ from c and that are *near* X. Note that we defined, in Sect. 2, an input string s_j as to be near a (complete) candidate solution X if its Hamming distance from X, $d_j(X)$, is less than the threshold d. However, because of our usage of partial solutions in the construction phase of the algorithm, we generalize the definition to the following. By the Hamming distance between an input string s_j and a partial solution X of length t, $1 \leq t \leq m$, we mean the Hamming distance between the strings of the first t characters of s_j and X; we still use $d_j(X)$ to denote this Hamming distance. Note that this definition is consistent with the previous one, considering a complete solution as a special case of a partial one whose length is m. In summary, the heuristic value given to an alphabet character c as a candidate for X^i is:

$$c.score = \sum_{\substack{j=1 \\ s_j^i \neq c \\ d_j(X)<d}}^{n} 1$$

Once the heuristic values for all candidate characters are calculated, some of the characters will be considered as the members of Restricted Candidate List, *RCL*, from which one is to be selected randomly. The number of elements in *RCL* may be viewed as a control parameter to make an appropriate balance between greediness and randomness in the construction phase. Among various mechanisms to do so are so-called *cardinality-based* and *value-based RCL*s. The former refers to the case where the size of *RCL* is predefined and fixed, whereas the latter refers to the case where it is determined dynamically based on the heuristic values of the candidates. A convenient way for the latter is to put in *RCL* every candidate whose heuristic value is better than a given percentage γ of the best heuristic value over all the candidates. The selection randomness in turn may or may not be based on the uniform probability distribution. More details on various strategies for GRASP may be found in Festa and Resende (2002, 2009).

The local search phase of GRASP receives the complete solution obtained by the construction phase and tries to improve it using a local search procedure. As can be seen in Fig. 4, the local search phase of `GRASP-FFMSP` is a simple hill-climbing procedure, which is based on the first-move (also called first-improvement) as opposed to the best-move (also called best-improvement) strategy (Blum and Roli 2003). That is, the current solution X will be replaced with the first solution of a higher value found in its neighborhood. The neighbors of a candidate solution X are defined to be those which differ from X in exactly one character; that is those whose Hamming distance from X is exactly 1. Candidate solutions are evaluated and compared using an evaluation function $h(.)$. The pseudo-code presented so-far (Fig. 4) is almost the same for both our algorithm and the algorithm in Festa (2007); the main distinction lies in the adopted heuristic evaluation function $h(.)$ used in the local search phase. In Festa (2007), the objective function $f(.)$ was used for this purpose, whereas we

Algorithm GRASP-FFMSP
Input: $S = \{s_1, s_2, \dots, s_n\}, n > 1$,each s_i is a string of length $m > 1$,
and an alphabet Σ
and an integer d, $1 \le d \le m$
Output: a string of length m
Parameters: *ItrNum* and γ
Function calls: it invokes $f(.)$ and $h_{f,\widetilde{GpC}}(.)$, which return the objective and the hybrid heuristic evaluation values for a given candidate solution, respectively. The pseudo-code for the latter may be found in Fig. 5.

```
1:  $BestSoFarX \leftarrow$ a random solution
2:  for $itr = 1$ to $ItrNum$ do
3:      $X \leftarrow$ the null string {initialize $X$ to an empty partial solution}

        {construction phase:}
4:      for $i = 1$ to $m$ do
5:          $RCL \leftarrow \phi$ {initialize $RCL$ to an empty set}
6:          for all $c \in \Sigma$ do
7:              $c.score \leftarrow$ number of strings $s_j$ such that $s_j^i \ne c$ and $d_j(X) < d$
8:          end for
            {make $RCL$:}
9:          $MaxScore \leftarrow Max\{c.score | c \in \Sigma\}$
10:         for all $c \in \Sigma$ do
11:             if $c.score > \gamma \times MaxScore$ then
12:                 $RCL \leftarrow RCL \cup \{c\}$
13:             end if
14:         end for
15:         $c \leftarrow$ a member of $RCL$ selected randomly
16:         $X^i \leftarrow c$ {add $c$ at the end of the partial solution $X$}
17:     end for

        {local search phase:}
18:     $improved \leftarrow true$
19:     while $improved$ do
20:         $improved \leftarrow false$
21:         for $k = 1$ to $m$ do
22:             for all $c \in \Sigma$ other than $X^k$ do
23:                 $tmpX \leftarrow$ the string obtained by replacing $X^k$ with $c$
24:                 if $h_{f,\widetilde{GpC}}(tmpX) > h_{f,\widetilde{GpC}}(X)$ then
25:                     $X \leftarrow tmpX$
26:                     $improved \leftarrow true$
27:                 end if
28:             end for
29:         end for
30:     end while

        {update $BestSoFarX$:}
31:     if $f(X) > f(BestSoFarX)$ then
32:         $BestSoFarX \leftarrow X$
33:     end if
34: end for

    {finally, return the best solution found over all the iterations:}
35: return $BestSoFarX$
```

Fig. 4 The pseudo-code for the algorithm GRASP-FFMSP

propose the use of the hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}(.)$. To summarize how the proposed function is determined, a pseudo-code for this function is presented in Fig. 5. It assumes that a two-dimensional array $T[\,][\,]$ has already been populated to represent the two-dimensional function $T(.,.)$ defined in Corollary 1 (see (13)). As can be seen in the pseudo-code, at first the values for $d_j(X)$, $c_j(X)$, $j = 1, \ldots, n$, $|Near(X)|$, and $f(X)$ are determined. Then, the value $\widetilde{GpC}(X)$ is calculated, which in turn requires the calculation of the values $\widetilde{g}_j(X)$, where $d_j(X) < d$. These calculations simply follow from (9), (10), and Corollary 1. The latter is used to calculate $Pr_{c_j}(\Delta = v)$, $c_k(X) \leq v \leq c_j(X)$, as $T[c_j, v]/|\Sigma|^{c_j}$. Finally, having calculated $\widetilde{GpC}(X)$, the algorithm calculates and returns the hybrid heuristic evaluation value $h_{f,\widetilde{GpC}}(X)$.

We now show that, given a candidate solution X, the value $h_{f,\widetilde{GpC}}(X)$ can be determined in $O(nm + n^2)$, having paid a memory cost and a one-off time cost of $O(m^2)$ to define and populate the two-dimensional array $T[\,][\,]$. The array T implicitly presents the probability distribution function (PDF) for Δ (see Fig. 3 for an illustration). As can be seen in Fig. 5, in order to determine $Pr_{c_j(X)}(\Delta \geq c_k(X))$, a `for` loop is used. However, it is possible to create another triangle of the same size to keep the cumulative distribution function (CDF) for the random variable Δ to retrieve $Pr_{c_j(X)}(\Delta \geq c_k(X))$ in $O(1)$ (note that $Pr_{c_j(X)}(\Delta \geq c_k(X)) = 1 - Pr_{c_j(X)}(\Delta \leq c_k(X) - 1)$). The memory cost and the one-off time cost to create and populate (the first m rows of) these arrays with the PDF and CDF values are still $O(m^2)$. The Hamming distances $d_j(X)$ and the costs $c_j(X)$ of the input strings s_j are determined in $O(nm)$, as can be seen in the beginning of the pseudo-code (Fig. 5). Note that the calculation of the Hamming distance between an input string and a candidate solution requires the comparison of m characters. Consequently, and since retrieving the CDF values is performed in $O(1)$, the value $\widetilde{GpC}(X)$ can be calculated in $O(nm + n^2)$. Therefore, $h_{f,\widetilde{GpC}}(X)$ is also computed in $O(nm + n^2)$, in addition to the memory and the one-off time costs of $O(m^2)$, which results in the total time complexity of $O((n + m)^2)$.

6 Experimental results

To evaluate the power of the proposed hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}$ we implemented two versions of GRASP, one as the state-of-the-art (Festa 2007), i.e. with the objective evaluation function $f(.)$ and the other with the heuristic evaluation function $h_{f,\widetilde{GpC}}$. We refer to these two versions as $\texttt{GRASP}_1$ and $\texttt{GRASP}_{\widetilde{GpC}}$, respectively. The algorithms were implemented in Java, using Eclipse, and run on a Pentium 4 Desktop machine with 3.21 GHz clock speed and 2 GB of RAM.

As for the value-based GRASP parameter, we set γ to 0.5. However, we did not use the same value for *ItrNum* in the algorithms, because each iteration of $\texttt{GRASP}_{\widetilde{GpC}}$ usually takes longer than each iteration of $\texttt{GRASP}_1$. We used 10 and 10,000 for this parameter in $\texttt{GRASP}_{\widetilde{GpC}}$ and $\texttt{GRASP}_1$, respectively, which guaranteed that $\texttt{GRASP}_{\widetilde{GpC}}$ would use much less time than $\texttt{GRASP}_1$. The local search phase in both algorithms was the hill-climbing iterated improvement, and two candidates were considered as neighbors if and only if their Hamming distance was 1.

```
Function h_{f,\widetilde{GpC}}
Input: a complete candidate solution $X$
Output: the hybrid heuristic evaluation value for $X$
Parameters: none
Function calls: none
{The variables $d_j, c_j, GpC, g_j, j = 1, \ldots, n, f$, and $NumNear$ are used to represent
$d_j(X), c_j(X), \widetilde{GpC}(X), \widetilde{g}_j(X), f(X)$, and $|Near(X)|$, respectively, for the given candidate solution $X$.}
 1: for $j = 1$ to $n$ do
 2:     $d_j \leftarrow$ the Hamming distance between $s_j$ and $X$
 3:     $c_j \leftarrow m - d_j$
 4:     if $d_j < d$ then
 5:         $NumNear \leftarrow NumNear + 1$
 6:     end if
 7:     $f \leftarrow n - NumNear$
 8: end for

    {determine $\widetilde{GpC}(X)$ using (9):}
 9: if $NumNear = 0$ then
10:     $GpC \leftarrow 0$
11: else
12:     $sumGpC \leftarrow 0$
        {determine $\widetilde{g}_j(X)$ using (10):}
13:     for $j = 1$ to $n$ do
14:         if $d_j < d$ then
15:             $g_j \leftarrow 1$
16:             for $k = 1$ to $n$ do
17:                 if $k \neq j$ then
18:                     $sumP \leftarrow 0$
                        {determine $Pr_{c_j(X)}(\Delta \geq c_k(X))$ (note that this probability will be 0 if $c_k > c_j$):}
19:                     for $v = c_k$ to $c_j$ do
20:                         $sumP \leftarrow sumP + T[c_j, v]/|\Sigma|^{c_j}$
21:                     end for
22:                     $g_j \leftarrow g_j + sumP$
23:                 end if
24:             end for
25:             $sumGpC \leftarrow sumGpC + g_j/c_j$
26:         end if
27:     end for
28:     $GpC \leftarrow sumGpC/NumNear$
29: end if

    {finally, return the hybrid heuristic evaluation value $h_{f,\widetilde{GpC}}(X)$:}
30: return $(n + 1) * f + GpC$
```

Fig. 5 The pseudo-code to calculate $h_{f,\widetilde{GpC}}(X)$

We examined and compared the algorithms on various instances of FFMSP with different numbers n and lengths m of input strings over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$. We used three different values of 100, 200, and 300 as the number of strings n. For each value v_n for n, we used three different values of v_n, $2v_n$, and $4v_n$ as the length of strings m. Finally, for each pair of values $\langle v_n, v_m \rangle$ for $\langle n, m \rangle$, we considered three different values of $0.75v_m$, $0.85v_m$, and $0.95v_m$ as the distance threshold d. These make $3 \times 3 \times 3 = 27$ different *instance types* altogether. By an instance type, we mean the triple of values for its $\langle n, m, d \rangle$.

The algorithms were evaluated on both random and real data. For each instance type, we generated 3 random instances, hence $27 \times 3 = 81$ random instances altogether. We used one real instance per instance type making altogether $81 + 27 = 108$ problem instances. The random data was generated by the standard Java pseudo-random number generator. As for real data, we used the Hyaloperonospora parasitica V 6.0 sequence data (HpV6Transcript.ntseq file).[2] These sequence data were produced by the US Department of Energy Joint Genome Institute[3] and curated at the Virginia Bioinformatics Institute.[4] For each pair of values $\langle v_n, v_m \rangle$ for the number and length of the required sequences, starting from smaller v_n followed by smaller v_m, we used the first n sequences not already-used whose length are at least m and truncated them to m.

The results for random data are presented in Table 1. The first column in this table shows the instance types. The second (F_1) and the third (T_1) columns show the average solution quality (i.e. the number of far strings) and average run-time (in milliseconds), respectively, for $\texttt{GRASP}_1$. The next two columns ($F_{\widetilde{GpC}}$ and $T_{\widetilde{GpC}}$) report these two quantities for $\texttt{GRASP}_{\widetilde{GpC}}$. The reported run-time does not include the time used to read in data files. The last column ($\alpha\%$) measures the improvement percentage defined as $\frac{F_{\widetilde{GpC}} - F_1}{F_{\widetilde{GpC}}} \times 100$. The last row in the table calculates the average of improvement percentage. The figures are rounded up to two decimal places.

As can be seen in Table 1, $\texttt{GRASP}_{\widetilde{GpC}}$ performs better than $\texttt{GRASP}_1$ in all the 27 cases. The minimum improvement percentage is 8.5 (for the first case) and its maximum is 100 (in 11 cases), where $\texttt{GRASP}_1$ gives a zero solution quality. Furthermore, for majority of cases where $\texttt{GRASP}_1$ gives nonzero solution quality, e.g. for the (100, 100, 95) instance type, the improvement is by orders of magnitude. The average improvement to the solution quality is more than 69 percent. This shows the significant effectiveness of the proposed hybrid heuristic evaluation function for the purpose of FFMSP. In general, this effectiveness is more clearly seen in the cases in Table 1 where $\texttt{GRASP}_1$ performs poorly giving low objective values.

The results for real data are similarly presented in Table 2. This table is the same as Table 1 except that it reports the results on real, as opposed to random, instances. Similarly, Table 2 indicates that $\texttt{GRASP}_{\widetilde{GpC}}$ is superior to $\texttt{GRASP}_1$ in all the 27 cases. The minimum improvement percentage is 2 (again for the first case) and its maximum is 100 (again in 11 cases), where $\texttt{GRASP}_1$ gives a zero solution quality. In most cases, the improvement in the solution quality is by orders of magnitude, with an average value of about 66 percent. Again, this suggests that the proposed hybrid heuristic evaluation function is quite effective for the purpose of FFMSP.

Tables 1 and 2 also show that the performance of the algorithms is usually better on real than random instances. Except for the cases where it gives the maximum n as for the objective values for both random and real data, $\texttt{GRASP}_{\widetilde{GpC}}$ performs better on

[2]Available at http://vmd.vbi.vt.edu/download/index.php.

[3]http://www.jgi.doe.gov.

[4]http://www.vbi.vt.edu.

Table 1 Comparison of $\texttt{GRASP}_1$ and $\texttt{GRASP}_{\widetilde{GpC}}$ with respect to average solution quality and average run-time on random instances

(n, m, d)	$\texttt{GRASP}_1$		$\texttt{GRASP}_{\widetilde{GpC}}$		$\alpha\%$
	F_1	T_1	$F_{\widetilde{GpC}}$	$T_{\widetilde{GpC}}$	
(100,100,75)	89.67	23391	98	151	8.5
(100,100,85)	10.33	10292	29.33	375	64.78
(100,100,95)	0.33	6620	7.67	422	95.7
(100,200,150)	85.33	38292	100	417	14.67
(100,200,170)	2.67	14578	28.33	1412	90.58
(100,200,190)	0	14162	5.33	1338	100
(100,400,300)	79.67	63515	100	401	20.33
(100,400,340)	1	28135	29.33	6802	96.59
(100,400,380)	0	28494	4.33	5240	100
(200,200,150)	162.33	128521	180.33	1011	9.98
(200,200,170)	3.67	33063	32	3328	88.53
(200,200,190)	0	31271	5.33	2745	100
(200,400,300)	153.67	203776	195.33	3536	21.33
(200,400,340)	0.67	60494	31.33	17261	97.86
(200,400,380)	0	61089	4	11161	100
(200,800,600)	146.33	326932	200	6005	26.84
(200,800,680)	0	130198	25.67	63104	100
(200,800,760)	0	131182	3.33	51250	100
(300,300,225)	230	338886	253.67	3166	9.33
(300,300,255)	1.67	76417	33	13969	94.94
(300,300,285)	0	76250	4.33	11031	100
(300,600,450)	222	586625	266.33	10250	16.64
(300,600,510)	0	165818	26	60115	100
(300,600,570)	0	165740	2.67	37901	100
(300,1200,900)	207.33	1009984	300	33073	30.89
(300,1200,1020)	0	413484	23	280307	100
(300,1200,1140)	0	412495	0.67	123260	100
Average					69.91

real than random instances (of the same type). This also holds for $\texttt{GRASP}_1$ except for the cases where it gives zero for both random and real instances.

Although, the number of iterations is large enough (10,000) for $\texttt{GRASP}_1$. In order to observe its performance over a longer time, we further increased *ItrNum* to 100,000 (i.e. by 10 times) and ran it on both random and real instances with $n = 200$, $m = 400$, and $d = 340$, i.e. moderate value for each. For both random and real cases, the algorithm returned the average solution quality of 1 (for both cases) within an average time of more than 9 minutes. As can be seen in Tables 1 and 2, $\texttt{GRASP}_{\widetilde{GpC}}$ obtained the average solution quality of 31.33 and 70 for random and real cases in less than 2 seconds, respectively.

Table 2 Comparison of $\texttt{GRASP}_1$ and $\texttt{GRASP}_{\widetilde{GpC}}$ with respect to average solution quality and average run-time on real instances

(n, m, d)	$\texttt{GRASP}_1$		$\texttt{GRASP}_{\widetilde{GpC}}$		$\alpha\%$
	F_1	T_1	$F_{\widetilde{GpC}}$	$T_{\widetilde{GpC}}$	
(100,100,75)	98	23484	100	47	2
(100,100,85)	24	10797	53	328	54.72
(100,100,95)	1	6484	9	344	88.89
(100,200,150)	94	38594	100	94	6
(100,200,170)	4	14516	46	1234	91.3
(100,200,190)	0	13922	8	1407	100
(100,400,300)	90	64766	100	172	10
(100,400,340)	1	27563	51	4609	98.04
(100,400,380)	0	27484	6	5719	100
(200,200,150)	191	158125	200	312	4.5
(200,200,170)	4	32844	82	3062	95.12
(200,200,190)	0	30828	8	3281	100
(200,400,300)	180	225812	200	500	10
(200,400,340)	1	60250	70	11047	98.57
(200,400,380)	0	60047	7	12828	100
(200,800,600)	164	350734	200	828	18
(200,800,680)	0	127437	83	41328	100
(200,800,760)	0	127593	7	54375	100
(300,300,225)	287	494922	300	578	4.33
(300,300,255)	2	75703	83	8218	97.59
(300,300,285)	0	75563	5	9937	100
(300,600,450)	268	712953	300	1312	10.67
(300,600,510)	0	165375	92	37000	100
(300,600,570)	0	164718	3	38813	100
(300,1200,900)	245	1131797	300	2407	18.33
(300,1200,1020)	0	408469	92	140328	100
(300,1200,1140)	0	409875	1	161859	100
Average					66.97

Finally, in order to see the effect of using the hybrid heuristic evaluation function $h_{f,\widetilde{GpC}}$ on the search landscape, particularly on reducing the number of local maximum points, we recorded for each run of the algorithms the average number of uphill moves in their local search (hill climbing) phases. An uphill move from a point in the search space means that the point is not a local, or a global, maximum point. Since the total number of points is fixed, 4^m in our case, the higher the number of such points is, the lower the number of local or global maximum points is. Again, we have averaged these quantities over several (10 random and 3 real) instances of the same type. The results are presented in Table 3, where the number of uphill moves (UP_1 for $\texttt{GRASP}_1$ and $UP_{\widetilde{GpC}}$ for $\texttt{GRASP}_{\widetilde{GpC}}$), as opposed to the objective values in Table 1,

Table 3 Comparison of the (average) number of up-hill moves in the local search phases of GRASP$_1$ (UP_1) and GRASP$_{\widetilde{GpC}}$ ($UP_{\widetilde{GpC}}$) over both random and real instances

(n,m,d)	Random			Real		
	UP_1	$UP_{\widetilde{GpC}}$	$\beta\%$	UP_1	$UP_{\widetilde{GpC}}$	$\beta\%$
(100,100,75)	11.32	62.9	82	13.82	61.3	77.45
(100,100,85)	1.11	92.13	98.79	1.71	101.3	98.31
(100,100,95)	0	110.13	100	0	102.8	100
(100,200,150)	8.59	143.33	94	9.7	97.3	90.03
(100,200,170)	0.04	267.93	99.98	0.06	236.7	99.97
(100,200,190)	0	257.6	100	0	256.5	100
(100,400,300)	5.62	228.73	97.54	6.34	133.3	95.24
(100,400,340)	0	811.33	100	0	601.5	100
(100,400,380)	0	575	100	0	589.5	100
(200,200,150)	17.62	73.4	75.99	29.72	128.9	76.94
(200,200,170)	0.08	257.07	99.96	0.1	232.2	99.95
(200,200,190)	0	264.53	100	0	271	100
(200,400,300)	14.38	197.97	92.73	19.14	190	89.92
(200,400,340)	0	872.8	100	0	529.2	100
(200,400,380)	0	573.17	100	0	570.9	100
(200,800,600)	9.15	542.77	98.31	11.52	251.5	95.41
(200,800,680)	0	1966.07	100	0	1303.6	100
(200,800,760)	0	1159.83	100	0	1103.6	100
(300,300,225)	21.54	83.6	74.23	46.69	152.4	69.36
(300,300,255)	0	555.63	100	0.01	357.2	99.99
(300,300,285)	0	441.83	100	0	359.4	100
(300,600,450)	18.93	166.57	88.63	30.7	271	88.67
(300,600,510)	0	1410.53	100	0	873.1	100
(300,600,570)	0	781.63	100	0	635.3	100
(300,1200,900)	12.27	783.43	98.43	16.86	416.3	95.95
(300,1200,1020)	0	3347.8	100	0	2100.3	100
(300,1200,1140)	0	1268.23	100	0	1100.5	100
Average			96.32			95.45

are reported. The fourth and the seventh columns report the improvement percentage ($\beta\%$), which is defined as $\frac{UP_{\widetilde{GpC}}-UP_1}{UP_{\widetilde{GpC}}} \times 100$ and used as a (rough) measure of the improvement in the search landscape with respect to the reduction in the number of local maximum points due to the use of the proposed hybrid heuristic function $h_{f,\widetilde{GpC}}$.

As can be seen in Table 3, the number of uphill moves is improved by orders of magnitude in *all* of the cases. This indicates remarkable reduction in the number of local maximum points in the search landscape due to using the heuristic evaluation function $h_{f,\widetilde{GpC}}$. On average, the increase in the number of uphill moves are 96.32% and 95.45% for random and real instances, respectively.

7 Conclusion

In general, a metaheuristic algorithm for an optimization problem uses a heuristic function to evaluate candidate solutions. Such heuristic functions are also called evaluation functions and, especially in the context of genetic algorithms, fitness functions. The very first choice for such a heuristic function is naturally the problem's objective function. However, in the case many candidate solutions can be of the same objective value, this can result in a search landscape in which many points correspond to local optima. The existence of local optima is widely-known to be an important factor in degrading the performance of metaheuristics, especially local search, for many optimization problems.

We believe that many combinatorial optimization problems, including FFMSP, are prune to this issue, because the number of candidate solutions are normally exponential whereas the number of different objective values are often polynomial in problem size for many such problems; these are $|\Sigma|^m$ and $1+n$, respectively, for FFMSP. As a starting point to address this issue, this paper concentrated on FFMSP. Firstly, the estimated Gain-per-Cost heuristic was devised to indicate, though roughly, the likelihood for a given candidate solution to be close to better ones. A triangle called Δ-triangle, in a similar fashion to that of Khayyam-Pascal triangle, was then developed to help determine this heuristic. Then, in order to restrict the application of the estimated Gain-per-Cost heuristic to the discrimination between candidate solutions with the same objective value, a hybrid heuristic function $h_{f,\widehat{GpC}}$ combining both the objective function and the estimated Gain-per-Cost heuristic was proposed.

Acknowledgements The authors would like to thank Dr. P. Festa for her initial help and comments regarding FFMSP.

References

Ashlock, D.: Evolutionary Computation for Modeling and Optimization. Springer, Berlin (2006)

Babaie, M., Mousavi, S.R.: A memetic algorithm for closest string problem and farthest string problem. In: Electrical Engineering (ICEE), 2010 18th Iranian Conference on (2010), pp. 570 –575, 11–13

Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)

Chen, J.-C.: Iterative rounding for the closest string problem. In: CoRR, arXiv:0705.0561v2 (2007)

Cheng, C.H., Huang, C.C., Hu, S.Y., Chao, K.: Efficient algorithms for some variants of the farthest string problem. In: 21st Workshop on Combinatorial Mathematics and Computation Theory, pp. 266–273 (2004)

Cohen, G.D., Karpovsky, M.G., Jr, H.F.M., Schatz, J.R. Covering radius—survey and recent results. IEEE Trans. Inf. Theory **31**(3), 328–343 (1985)

Crooke, S.T., Lebleu, B.: Antisense Research and Applications. CRC Press, Boca Raton (1993)

de Meneses, C.N., Lu, Z., Oliveira, C.A.S., Pardalos, P.M.: Optimal solutions for the closest-string problem via integer programming. INFORMS J. Comput. **16**(4), 419–429 (2004)

Edwards, A.W.F.: Pascal's Arithmetical Triangle: The Story of a Mathematical Idea. Johns Hopkins University Press, Baltimore (2002)

Feo, T., Resende, M.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**, 67–71 (1989)

Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995)

Festa, P.: On some optimization problems in molecular biology. Math. Biosci. **207**(2), 219–234 (2007). BIOCOMP2005 Special Issue

Festa, P., Resende, M.: Grasp: An annotated bibliography. In: Essays and Surveys on Metaheuristics, pp. 325–367. Kluwer Academic, Amsterdam (2002)

Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP; Part I: Algorithms. Int. Trans. Oper. Res. **16**(1), 1–24 (2009)

Fogel, D.B.: Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, 3rd edn. IEEE Press, New York (2006)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Amsterdam (1997)

Gomes, F.C., de Meneses, C.N., Pardalos, P.M., Viana, G.V.R.: A parallel multistart algorithm for the closest string problem. Comput. OR **35**(11), 3636–3643 (2008)

Gramm, J.: Closest substring. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, p. 156-8. Springer, Berlin (2008)

Gramm, J., Guo, J., Niedermeier, R.: On exact and approximation algorithms for distinguishing substring selection. In: FCT, pp. 195–209 (2003)

Gramm, J., Hüffner, F., Niedermeier, R.: Closest strings, primer design, and motif search. In: Sixth Annual International Conference on Computational Molecular Biology, pp. 74–75 (2002)

Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica **37**(1), 25–42 (2003)

Lanctot, J.K.: Some string problems in computational biology. PhD thesis, Univesity of Waterloo (2000)

Lanctot, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. In: SODA '99: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 633–642. Society for Industrial and Applied Mathematics, Philadelphia (1999)

Lanctot, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Inf. Comput. **185**(1), 41–55 (2003)

Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. In: STOC '99: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pp. 473–482. ACM, New York (1999)

Li, M., Ma, B., Wang, L.: On the closest string and substring problem. J. ACM **49**(2), 157–171 (2002)

Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. In: Research in Computational Molecular Biology: 12th Annual International Conference RECOMB 2008, pp. 396–409 (2008)

Macario, A.J.L., de Macario, E.C.: Gene probes for bacteria. Academic Press, New York (1990)

Meneses, C.N., Oliveira, C.A.S., Pardalos, P.M.: Optimization techniques for string selection and comparison problems in genomics. IEEE Eng. Med. Biol. Mag. **24**(3), 81–87 (2005)

Meneses, C.N., Pardalos, P.M., Resende, M.G.C., Vazacopoulos, A.: Modeling and solving string selection problems. In: Second International Symposium on Mathematical and Computational Biology, pp. 54–64 (2005)

Peelle, H.A.: Euclid, Fibonacci, and Pascal recursed. Int. J. Math. Educ. Sci. Technol. **6**(4), 395–405 (1975)

Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, New York (2003)

Sahinalp, S.C., Muthukrishnan, S., Dogrusöz, U. (eds.): Combinatorial Pattern Matching. In: Proceedings 15th Annual Symposium, CPM 2004, Istanbul,Turkey, 5–7 July 2004. Lecture Notes in Computer Science, vol. 3109. Springer, Berlin (2004)

Wang, J., Chen, J., Huang, J.M.: An improved lower bound on approximation algorithms for the closest substring problem. Inf. Process. Lett. **107**(1), 24–28 (2008)