Why Large Closest String Instances Are Easy to Solve in Practice

Christina Boucher[1] and Kathleen Wilkie[2]

[1] David R. Cheriton School of Computer Science,
University of Waterloo
cabouche@cs.uwaterloo.ca
[2] Department of Applied Mathematics,
University of Waterloo
kpwilkie@math.uwaterloo.ca

Abstract. We initiate the study of the smoothed complexity of the Closest String problem by proposing a semi-random model of Hamming distance. We restrict interest to the optimization version of the Closest String problem and give a randomized algorithm, we refer to as *CSP-Greedy*, that computes the closest string on smoothed instances up to a constant factor approximation in time $O(\ell^3)$, where ℓ is the string length. Using smoothed analysis, we prove *CSP-Greedy* achieves a $\left((1+\frac{\epsilon e}{2^n})\right)^{\ell}$-approximation guarantee, where $\epsilon > 0$ is any small value and n is the number of strings. These approximation and runtime guarantees demonstrate that Closest String instances with a relatively large number of input strings are efficiently solved in practice. We also give experimental results demonstrating that *CSP-greedy* runs extremely efficiently on instances with a large number of strings. This counter-intuitive fact that "large" Closest String instances are easier and more efficient to solve gives new insight into this well-investigated problem.

1 Introduction

The Closest String is one of the central theoretical problems in bioinformatics and as such, has been studied extensively in bioinformatics and computational biology [7,8,14,15,17,20,21]. It has a wide variety of applications, including universal PCR primer design [10,15,18,26], genetic probe design [15], antisense drug design [9,15], finding transcription factor binding sites in genomic data [21], determining an unbiased consensus of a protein family [4], and motif-recognition [15,24,25]. The Closest String problem is NP-complete, unless P = NP [13], and therefore is unlikely to be solvable in polynomial time.

We initiate the study of the smoothed complexity of the optimization version of the Closest String problem, which can be defined as follows: given a set of ℓ-length strings $S = \{s_1, \ldots, s_n\}$[1] from the alphabet Σ, determine a string s of length ℓ such that $d(s, s_i) \leq d$ for all $s_i \in S$ and d is minimized. We refer

[1] Technically, this is a mulitset since we allow any string to occur multiple times

E. Chavez and S. Lonardi (Eds.): SPIRE 2010, LNCS 6393, pp. 106–117, 2010.

to s as the *closest string* for the set S. Here $d(\cdot, \cdot)$ is the Hamming distance. The concept of smoothed analysis was introduced as an intermediate measure between average case analysis and worst case analysis; whereas average analysis studies the average behaviour of an algorithm over all instances of a problem, smoothed analysis studies the algorithm's average behaviour on each "local region" of the instance space [28]. If the algorithm has good average performance on each local region, then for any reasonable probabilistic distribution on the whole instance space, the algorithm should perform well. If the smoothed complexity is low, worst case instances are not robust under small changes. Most small changes to the instance destroy the property of being worst-case; a small random perturbation to the instance destroys the property of being worst-case.

Several other papers discuss the smoothed complexity of continuous problems [5,12] and of discrete problems [3,19]. The smoothed complexity of other string and sequence problems has been considered by Andoni and Krauthgamer [2], Manthey and Reischuk [22], and Ma [19]. Andoni and Krauthgamer [2] study the smoothed complexity of sequence alignment by the use of a novel model of edit distance; their results demonstrate the efficiency of several tools used for sequence alignment, most notably PatternHunter [21]. Manthey and Reischuk gave several results considering the smoothed analysis of binary search trees [22]. Ma demonstrated that a simple greedy algorithm runs efficiently in practice for Shortest Common Superstring [19], a problem that has application to string compression and DNA sequence assembly.

We give a smoothed analysis of an efficient probabilistic algorithm for Closest String instances where the alphabet is binary. To the best of our knowledge this is the first analysis of a natural random model of the Closest String problem. The main contributions of this paper are as follows:

- We describe our algorithm, *CSP-Greedy*, prove it achieves a 2-approximation guarantee, and give a bound on the probability that *CSP-Greedy* returns an optimal solution to the Closest String instance.
- Next, we empirically study *CSP-Greedy* and demonstrate that Closest String instances with a large number of strings (*i.e.* $n \geq 15$) are solved with extreme efficiency by this algorithm.
- Lastly, we define a natural model of perturbation for Closest String instances and use smoothed analysis to show that *CSP-Greedy* achieves an $1 + f(\epsilon, n, \ell)$-approximation guarantee in time $O(\ell^3)$, where $\epsilon > 0$ is any small value and $f(\epsilon, n, \ell)$ approaches zero in time that is exponential in n. Hence, this result gives analytical explanation for our empirical results.

1.1 Preliminaries

Let s be a string over the alphabet Σ. We restrict interest to Closest String instances where the alphabet is binary and hence, unless otherwise stated we assume Σ is the binary alphabet. Denote the length of s by $|s|$, and the jth letter of s by $s(j)$. Hence, $s = s(1)s(2)\ldots s(|s|)$. Lastly, we denote a function g to be an asymptotic estimation of f as $f \asymp g$.

Given a set of strings $S = \{s_1, \ldots, s_n\}$, each string of length ℓ, then a string s is a *closest string* for S if and only if there is no string s' such that $\max_{i=1,\ldots,n} d(s', s_i) < \max_{i=1,\ldots,n} d(s, s_i)$. Let s be a closest string for S then the *optimal closest distance* d is equal to $\max_{i=1,\ldots,n} d(s, s_i)$. We refer to a *majority vote* for S as the ℓ-length string containing the letter that occurs most often at each position; this string is not necessarily unique.

1.2 Previous Results

Lanctot *et al.* [15] gave a polynomial-time algorithm that achieves a $\frac{4}{3} + o(1)$ approximation guarantee. Li *et al.* [17] proved the existence of a polynomial time approximation scheme (PTAS) for this problem, though the high degree in the polynomial complexity of the PTAS algorithm renders this result only of theoretical interest. In 2005, Brejová *et al* [7] proved the existence of sharper upper and lower bounds for the PTAS on a slight variant of the Closest String problem (which they refer to as the Consensus Pattern problem) and in 2006, Brejová *et al* [8] improved upon the analysis of the PTAS for various random binary motif models. Andoni *et al.* [1] gave a novel PTAS that has improved time complexity, and most recently, Ma and Sun [20] presented a PTAS with time complexity $O(n^{\Theta(n^{-2})})$, which is currently the best known running time.

Another approach to investigate the tractability of this NP-complete problem is to consider the parameterized complexity of the Closest String problem. A problem φ is said to be *fixed-parameter tractable* (FPT) with respect to parameter k if there exists an algorithm that solves φ in $f(k) \cdot n^{O(1)}$ time, where f is a function of k that is independent of n [11]. Gramm *et al.* [14] demonstrated that the Closest String problem is FPT when the number of strings, denoted as $|S|$, remains fixed. This FPT result is based on an integer linear programming formulation with a constant number of variables (assuming n is fixed), and the application of the result of Lenstra [16], which states that ILP is polynomial-time solvable when the number of variables remains fixed. Unfortunately, such an integer programming formulation is only of theoretical interest since the corresponding algorithms lead to very long running times even when the number of strings is small. Other parameterizations of the Closest String problem also exist; for example, when d is fixed, the problem can be solved in $O(nl + nd(d+1)^d)$ time [14]. Ma and Sun gave an $O(n|\Sigma|^{O(d)})$ algorithm, which is a polynomial-time algorithm when $d = O(\log n)$ and Σ has constant size [20].

2 A Randomized Algorithm for Closest String

In [27], Schöning considers the following simple probabilistic algorithm for solving the NP-complete problem of k-SAT: randomly choose a starting assignment and subsequently augment this initial assignment until a satisfying one is obtained. Papadimitriou introduced this random paradigm in the context of 2-SAT and obtained an expected quadratic time bound [23]. This type of algorithms are referred to as Monte Carlo algorithms with one-sided error; a useful property of

such algorithm is that the error probability can be made arbitrarily small with repeated independent random repetitions of the search process. We analyze a similar probabilistic approach for the Closest String problem. Boucher and Brown [6] introduced a similar algorithm to Algorithm 2 for the decision version of the Closest String problem and conjectured about the probability that the algorithm successfully determines a solution to a Closest String instance; the results in this section resolve this conjecture.

Algorithm 2 begins with a string $s_{maj,0}$ randomly selected from all majority strings and iteratively *augments* the string so that it is closer to one of the strings in S a maximum of t times. Let t be the number of times a random majority string is chosen and augmented ℓ times. Later in the section we define t in terms of the parameters ℓ, n, and d. In order to determine a closest string corresponding to the optimal closest distance, this search process is repeated with the degeneracy parameter ranging from 0 to ℓ. We let Δd be the current degeracy in the search process.

Let $s_{maj,i}$ be $s_{maj,0}$ after it has been updated i times. At iteration $i+1$, we obtain the string $s_{maj,i+1}$ by augmenting $s_{maj,i}$ so that it has smaller Hamming distance to at least one string s_k in S where $d(s_k, s_{maj,i}) > \Delta d$. This process is repeated ℓ times at which point the process is restarted if there is at least one string s_k in S where $d(s_k, s_{maj,\ell}) > \Delta d$.

Algorithm 1. Procedure *augment*

Input: A set S of n ℓ-length strings, parameter d.
Output: A ℓ-length string s or "not found"
Let $\mathcal{S}$ be the set of all ℓ-length majority strings
Select $s_{maj,0}$ randomly from $\mathcal{S}$.
For $i = 0, \ldots, \ell$:
  If $d(s_{maj,i}, s_j) \leq d$ for all $s_j \in S$ then return s and terminate.
  Else $P = \{j : s_j \in S \text{ and } d(s_j, s_{maj,i}) > d\}$
  Choose any $p \in P$ and $1 \leq k \leq \ell$ such that $s_p(k) \neq s_{maj,i}(k)$
  Set $s_{maj,i+1}$ to be equal to s_p at position k and equal to $s_{maj,i}$ at all other positions
Return "not found"

Algorithm 2. *CSP-Greedy*

Input: A set S of n ℓ-length binary strings.
Output: A ℓ-length string s
For Δd each from $0 \rightarrow \ell$
  Repeat *augment* t times with parameter Δd

2.1 Approximation Guarantee for *CSP-Greedy*

In this subsection we give worst-case bounds on the approximation guarantee of *CSP-Greedy*. Also, we present examples of inputs for which the algorithm performs poorly and hence, give lower bounds on the approximation guarantee.

Theorem 1. *The approximation ratio of the CSP-Greedy algorithm is at most 2 for any alphabet size* $|\Sigma|$.

Proof. Since *CSP-Greedy* begins with a randomly selected majority string, it is sufficient to show that for any set of strings S the maximum distance from any majority string to any string in S is twice the optimal closest distance. Let S be a set of strings with optimal closest distance equal to d_{opt}. Without loss of generality assume 0^ℓ is a closest string for S and hence the maximum number of non-zero positions in each string $s_i \in S$ is at most d_{opt}. By the pigeonhole principle, the maximum number of positions containing greater than $n/2$ non-zero positions in a given column is at most $2d_{opt}$. Therefore, any majority string can have at most $2d_{opt}$ non-zero positions and is at distance at most $2d_{opt}$ from each string in S. □

The following example demonstrates that the 2-approximation guarantee is tight $S = \{10000001111, 01000001111, 11111111111, 11111111111\}$, where majority string is $c_1^* = 11111111111$ and optimal closest distance is $\ell/4$. On the first iteration the majority string could be altered to $c_2^* = 01111111111$. On the second iteration we could choose to alter this sequence back to c_1^*. It is possible to alternate between c_1^* and c_2^* and end up returning c_1^*, which is equal to twice the optimal closest distance.

2.2 Probabilistic Analysis of *CSP-Greedy*

The process of augmenting a randomly selected majority string ℓ times or until a closest string is found can be viewed as a Markov chain. This abstraction will be useful in achieving an upper bound on the probability that *CSP-Greedy* returns an optimal solution.

Let a set S be *uniquely satisfiable* if there exists exactly one string s^* where $d(s_i, s^*) \leq d^*$ for all $s_i \in S$. If S is an instance that does not have a string s such that $d(s, s_i) \leq d^*$ for all $s_i \in S$, then the *augment* procedure will return "not found". So we assume otherwise, that the set S is uniquely satisfiable and denote the probability of obtaining s^* when $\Delta d = d^*$ as p_s. If $s_{maj,i}$ is not equal to s^* then there is at least one letter of $s_{maj,i}$ that can be changed so that $d(s_{maj,i}, s^*)$ decreases by one; the probability of this occurring is at least $1/\ell$. Denote $X_i \in \{0, 1, \ldots, \ell\}$ $(i = 0, 1, \ldots)$ as the random variable that is equal to the Hamming distance between $s_{maj,i}$ and s^*, where i is the number of iterations of the *augment* procedure. Each time a position is selected and the value of that position is augmented, either the Hamming distance is increased or decreased by one.

The process $X_0, X_1, X_2, \ldots$ is a Markov chain with a barrier at state ℓ and contains varying time and state dependent transfer probabilities. This process is overly complicated and we instead choose to analyze the following process: $Y_0, Y_1, Y_2, \ldots$, where Y_i is the random variable which is equal to the state number after i steps and there exists infinitely many states. Initially, this Markov chain is started like the stochastic process above (*i.e.* $Y_0 = X_0$). As long as the inner loop is iterating, we let $Y_{i+1} = X_i - 1$ if the process decreases the Hamming distance

between $s_{maj,i}$ and s^* by one; and $Y_{i+1} = X_i + 1$ otherwise. After the loop exits, we continue with the same transfer probabilities. By induction on i, it is clear that for each i, $X_i \leq Y_i$, and it follows that p_s is at least $\Pr[\exists t \leq c\ell : Y_t = 0]$.

We made the assumption that the set was uniquely satisfiable, however, this assumption is not needed – the random walk may find another closest string while not in the terminating state but this possibility only increases the probability the algorithm terminates.

The following asymptotic estimation is used in the proof of the next theorem. See the appendix for the justification of this fact.

Fact 1. *For $i > 0$ and $j > 0$ the following asymptotic estimation exists:*

$$\sum_{i=0}^{j} \binom{2i+j}{i} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j} \asymp \left(\frac{1}{\ell-1}\right)^j. \tag{1}$$

Theorem 2. *Let S be a set of strings and d^* be the optimal closest distance. Then the probability of procedure 'augment' obtaining a closest string for S is at least $e \cdot 2^{-\ell}$ for sufficiently large ℓ.*

Proof. Suppose s^* is a closest string for S and let k be the Hamming distance between $s_{maj,0}$ and s^*. Denote $\Pr[Y_{i+1} = j - 1 | Y_i = j]$ by q_i. Given that the Markov chain starts in some state j, it can reach a halting state in at least j steps by making transitions through the states $j-1$, $j-2$, …, $1, 0$. The probability of this happening is at least $\sum_{i=0}^{j} q_i$. For $i = 0, 1, 2, \ldots$ the halting state can be reached after $2i + j$ steps, where i steps are "bad" and $j + i$ steps are "good". The probability of this happening is:

$$\Pr[Y_{2i+j} = 0, \text{ and } Y_k > 0 \,\forall\, k < 2i + j \,|\, Y_0 = j],$$

which is at least $q_i^{i+k}(1-q_i)^i$ times the number of ways of arranging i bad steps and $i+j$ good steps such that the sequence starts in state j, ends in state 0, and does not reach 0 before the last step. Using the Ballot theorem we know there are $\binom{2i+j}{i}\frac{i}{2i+j}$ possible arrangements of these i and $i+j$ steps. Therefore, the above probability is at least:

$$\binom{2i+j}{i}\frac{i}{2i+j}(1-q_i)^i q_i^{i+j}.$$

This expression is not defined in the case $i = j = 0$. In this case, the probability is equal to 1. Thus, we have:

$$\begin{aligned}
\Pr[Y_{2i+j} = 0, \text{ and } Y_k > 0 \,\forall\, k < 2i + j | Y_0 = j] = \\
&\geq \sum_{j=0}^{\ell} 2^{-\ell} \binom{\ell}{j} \sum_{2i+j \leq c\ell} \binom{2i+j}{i} \frac{i}{2i+j} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j} \\
&\geq 2^{-\ell} \sum_{j=0}^{\ell} \binom{\ell}{j} \sum_{i=0}^{j} \binom{2i+j}{i} \left(\frac{\ell-1}{\ell}\right)^i \left(\frac{1}{\ell}\right)^{i+j}
\end{aligned}$$

Using Fact 1, we have:

$$\begin{aligned}\Pr[Y_{2i+j} = 0, \text{ and } Y_k > 0 \,\forall\, k < 2i + j | Y_0 = j] &\asymp 2^{-\ell} \sum_{j=0}^{\ell} \binom{\ell}{j} \left(\frac{1}{\ell - 1}\right)^j \\ &= \left(\frac{\ell}{2(\ell-1)}\right)^{\ell} \text{ [by the Binomial theorem]} \\ &\asymp e \cdot 2^{-\ell} \left(\frac{\ell}{\ell - 1}\right) \text{ [by Taylor's theorem]}\end{aligned}$$

For sufficiently large ℓ we have the probability is at least $e \cdot 2^{-\ell}$. □

The following corollary, an immediate consequence of the previous lemma, bounds the probability of error.

Corollary 1. *If $t = \ell$ and ℓ is sufficiently large, CSP-Greedy has time complexity $O(\ell^3)$ and one-sided error probability of no more than $\left(1 - e \cdot 2^{-\ell}\right)^{\ell}$.*

3 Experimental Results

We empirically investigate how the number of augmentations changes as the number of strings (parameter n) increases. For these experiments, we fix ℓ to be 15, d to be 4, and vary the value of n from 12 to 36. For each value of n, we generate 100 motif sets at random by selecting a string s at random from the set of all 2^{ℓ} possible strings and then n motif strings at random from the set of all strings of distance at most d from s. For each motif set we run Procedure *augment* twice–once with $s_{maj,0}$ initialized to be equal to a majority string, and a second time with $s_{maj,0}$ initialized to be equal to a random string–and in both cases, we count the number of alterations required to obtain a closest string. We determine the mean number of alterations for 100 motif sets. Figure 1 illustrates this data, which shows that the number of augmentations required to obtain a closest string is significantly larger if $s_{maj,0}$ is initialized to be a random string. Further, as the value of n increases the disparity between the number of augmentations of a majority string and the number of augmentations of a random string increases substantially.

Table 1 illustrates the change in the (Hamming) distance as the values n, ℓ, and d increase. We consider three different values of ℓ and d, varied the value of n from 5 to 35, generate 100 random motif instances, calculate the Hamming distance between $s_{maj,0}$ and s^* (a closest string found by Algorithm 2) and determine the mean Hamming distance of the 100 motifs. This data demonstrates a drastic decrease in the distance between the majority string and the closest string as n increases. Note that when the value of n is significantly large the distance between the majority string and the closest string is equal to zero – implying the majority string is a closest string.

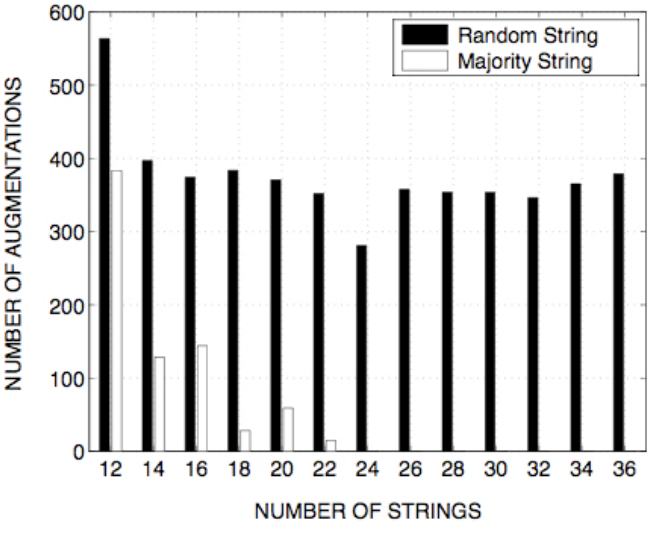

Fig. 1. Illustration of the effect of the initialization of the starting string on efficiency of Procedure *augment* as the value of n increases. A comparison between the mean number of augmentations required to obtain a closest string if starting from a majority string (white) or if starting from a random string (black). One hundred random motif sets are generated for each value of n and the mean of the augmentations is determined. ℓ is fixed to 15 and d is fixed to 4.

Table 1. An Illustration of the variation in the Hamming distance between a majority and closest string as ℓ, d, and n change. An illustration of the Hamming distance between a randomly chosen majority string and a closest string with respect to n. We considered the following (ℓ, d) pairs: $(15, 4)$, $(18, 6)$, and $(25, 5)$ and varied n to be every even value from 5 to 35. For each (ℓ, d) pair and value of n we generated 100 random motif sets, determined the Hamming distance between a randomly selected majority string and the closest string found by the algorithm, and calculated the mean Hamming distance.

n	(ℓ, d)		
	$(15, 4)$	$(18, 6)$	$(25, 5)$
5	3.2	2.9	3
8	2.7	2.3	2.1
10	0.8	1.0	1.8
12	0	0.7	1.2
15	0	0	0.5
25	0	0	0
30	0	0	0
35	0	0	0

In summary, our experimental results illustrate the following trends: as the value of n increases the (Hamming) distance between a randomly selected majority string and a closest string decreases and the number of augmentations required to obtain a closest string from a majority string decreases. Furthermore, regardless of the value of ℓ and d, for significantly large values of n the majority string is also likely to be a closest string. Similar results were also reported by Boucher and Brown [6].

4 Smoothed Analysis of *CSP-Greedy*

Using smoothed analysis, we demonstrate that the approximation ratio of the greedy algorithm on small perturbations of the worst-case instances is equal to $\left(1 + \frac{\epsilon(e-1)}{2^n}\right)^{\ell}$. Hence, our analysis shows that the approximation is equal to $1 + o(1)$ for significantly large n. This result explains why *CSP-Greedy* performs well in practice on large instances (*i.e.* instances containing a significantly large number of strings).

A *perturbed instance* S is defined to be $S' = \{s'_1, s'_2, \ldots, s'_n\}$, where each s'_i has length ℓ and each s'_i is obtained by mutating uniformly at random each letter of s_i with a small probability $p > 0$. In more general terms, an adversary chooses n length-ℓ sequences from the binary alphabet at random, and every symbol is perturbed with a small probability p. Next, let S be a closest string instance, S' be the corresponding perturbed instance, and q be $\Pr[s_i(j) = 1]$. We have

$$\begin{aligned} Pr[s'_i(j) = 0] &= \Pr[s_i(j) = 1] \Pr[s_i(j) \text{ was permuted}] + \\ &\quad \Pr[s_i(j) = 0] \Pr[s_i(j) \text{ was not permuted}] \\ &= qp + (1 - q)(1 - p). \end{aligned}$$

Assuming, $\epsilon > 0$ is a small number, and the perturbation probability $\frac{\epsilon \log(\ell n)}{\ell n} \leq p \leq \frac{1}{2}$, we obtain the following:

$$\Pr[s'_i(j) = 0] = 1 - q - p + 2qp \tag{2}$$

$$\geq 1 - q - p + q \quad \text{since } p \leq 1/2 \tag{3}$$

$$\geq \frac{\epsilon \log(\ell n)}{\ell n} \tag{4}$$

The next theorem, our main result, demonstrates that for significantly large ℓ, as n (and to a lesser extend ℓ) increases the approximation ratio approaches 1. This result explains our experimental results analytically.

Theorem 3. (*CSP-Greedy* under limited randomness) *For any given small $\epsilon > 0$, for perturbation probability $\frac{\epsilon \log(\ell n)}{\ell n} \leq p \leq \frac{1}{2}$ and significantly large ℓ, the expected ratio of 'CSP-Greedy' on the perturbed instances is $\left(1 + \frac{\epsilon e}{2^n}\right)^{\ell}$.*

Proof. Given a Closest String instance S, we define the instance as *good* if each majority string of S is also a closest string for S; otherwise, we define

the instance as *bad*. Since each iteration of the *augment* procedure begins by selecting a random majority string and determining whether it has distance at most d from each string in S, it follows that *CSP-Greedy* will return a closest string when S is a good instance. Let p_{bad} be the probability that the perturbed instance is bad.

In order to bound p_{bad} we first calculate the probability that a majority string does not match the closest string at a particular position. There exists at least one string s such that $d(s, s'_i) \leq d$ for all $s'_i \in S'$. Without loss of generality assume that 0^ℓ is a closest string. We calculate the probability that 0^ℓ is a majority string. Let $X_{i,j}$ be a binary random variable representing the symbol of s_i at the j-th position. For a given j, let the number of ones be $X_j = \sum_i X_{i,j}$.

$$\begin{aligned}\Pr[X_j > n/2] &= \sum_{i=n/2}^{n} \binom{n}{i} (\Pr[s'_i(j) = 1])^i (1 - \Pr[s'_i(j) = 1])^{n-i} \\ &\geq 1 - (1 - \Pr[s'_i(j) = 1])^n\end{aligned}$$

Let $\alpha = 1 - (1 - \Pr[s'_i(j) = 1])^n$. We give a lower bound for the probability that the instance is good.

$$\begin{aligned}1 - p_{bad} &= 1 - \Pr[S' \text{contains at least one bad column}] \\ &= 1 - \sum_{i=1}^{\ell} \binom{\ell}{i} \Pr[X_j > n/2]^i \\ &\geq 1 - \sum_{i=1}^{\ell} \binom{\ell}{i} \alpha^i \\ &\geq 1 - \alpha^\ell \text{ [by Binomial Theorem]}\end{aligned}$$

Therefore, it follows that p_{bad} is at most $(1 - (1 - \Pr[s'_i(j) = 1])^n)^\ell$ and hence, we get:

$$p_{bad} \leq (1 - (1 - \Pr[s'_i(j) = 1])^n)^\ell \leq \left(1 - \frac{\epsilon}{2^n}\right)^\ell$$

In Theorem 1 the greedy algorithm was proved to have a worst-case approximation ratio of 2. Therefore, by Theorem 1 and Corollary 1 we obtain the following:

$$\begin{aligned}E[ratio] &\leq 2 \left(1 - \frac{\epsilon}{2^n}\right)^\ell \left(1 - \frac{e}{2^\ell}\right)^\ell \text{ [by Corollary 1]} \\ &\leq 2 \left(1 + \frac{\epsilon e}{2^{n+\ell-1}} - \frac{e}{2^\ell} - \frac{\epsilon}{2^n}\right)^\ell\end{aligned}$$

For significantly large values of n and ℓ, we have:

$$\left(1 + \frac{\epsilon e}{2^{n+\ell-1}} - \frac{e}{2^\ell} - \frac{\epsilon}{2^n}\right)^\ell \leq \frac{1}{2^{1/\ell}} \left(1 + \frac{\epsilon e}{2^n}\right)^\ell,$$

and therefore,

$$E[ratio] \leq \left(1 + \frac{\epsilon e}{2^n}\right)^\ell. \qquad \square$$

We note that our perturbation is very small compared to the size of the instance. With perturbation probability $p = \frac{\epsilon \log(\ell n)}{\ell n}$, each set of n strings of length ℓ is expected to change by $\log(\ell n)$ letters.

Acknowledgements

The authors would like to thank Professor Bin Ma for his discussions and insights concerning the results presented in this paper and Professor Ming Li for suggesting this area of study.

References

1. Andoni, A., Indyk, P., Patrascu, M.: On the optimality of the dimensionality reduction method. In: Proc. of 47th FOCS, pp. 449–456 (2006)
2. Andoni, A., Krauthgamer, R.: The smoothed complexity of edit distance. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 357–369. Springer, Heidelberg (2008)
3. Banderier, C., Beier, R., Mehlhorn, K.: Smoothed analysis of three combinatorial problems. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 198–207. Springer, Heidelberg (2008)
4. Ben-Dor, A., Lancia, G., Perone, J., Ravi, R.: Banishing bias from consensus strings. In: Hein, J., Apostolico, A. (eds.) CPM 1997. LNCS, vol. 1264, pp. 247–261. Springer, Heidelberg (1997)
5. Blum, A., Dunagan, J.D.: Smoothed analysis of the perceptron algorithm for linear programming. In: Proc. of 13th SODA, pp. 905–914 (2002)
6. Boucher, C., Brown, D.G.: Detecting motifs in a large data set: applying probabilistic insights to motif finding. In: Rajasekaran, S. (ed.) BICoB 2009. LNCS, vol. 5462, pp. 139–150. Springer, Heidelberg (2009)
7. Brejová, B., Brown, D.G., Harrower, I., López-Ortiz, A., Vinař, T.: Sharper upper and lower bounds for an approximation scheme for CONSENSUS-PATTERN. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 1–10. Springer, Heidelberg (2005)
8. Brejová, B., Brown, D.G., Harrower, I., Vinař, T.: New bounds for motif finding in strong instances. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 94–105. Springer, Heidelberg (2006)
9. Deng, X., Li, G., Li, Z., Ma, B., Wang, L.: Genetic design of drugs without side-effects. SIAM J. Comput. 32(4), 1073–1090 (2003)
10. Dopazo, J., Rodríguez, A., Sáiz, J.C., Sobrino, F.: Design of primers for PCR amplification of highly variable genomes. CABIOS (9), 123–125 (1993)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
12. Dunagan, J.D., Spielman, D.A., Teng, S.-H.: Smoothed analysis of the renegar's condition number for linear programming. In: Proc. of SIOPT (2002)
13. Frances, M., Litman, A.: On covering problems of codes. Th. Comp. Sys. 30(2), 113–119 (1997)
14. Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-parameter algorithms for closest string and related problems. Algorithmica 37, 25–42 (2003)

15. Lanctot, J.K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Inf. Comput. 185(1) (2003)
16. Lenstra, W.H.: Integer programming with a fixed number of variables. Math. of OR 8, 538–548 (1983)
17. Li, M., Ma, B., Wang, L.: Finding similar regions in many strings. J. Comput. Syst. Sci. 65(1), 73–96 (2002)
18. Lucas, K., Busch, M., Össinger, S., Thompson, J.A.: An improved microcomputer program for finding gene- and gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. CABIOS 7, 525–529 (1991)
19. Ma, B.: Why greedy works for shortest common superstring problem. In: Ferragina, P., Landau, G.M. (eds.) CPM 2008. LNCS, vol. 5029, pp. 244–254. Springer, Heidelberg (2008)
20. Ma, B., Sun, X.: More efficient algorithms for closest string and substring problems. In: Vingron, M., Wong, L. (eds.) RECOMB 2008. LNCS (LNBI), vol. 4955, pp. 396–409. Springer, Heidelberg (2008)
21. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)
22. Manthey, B., Reischuk, R.: Smoothed analysis of binary search trees. Th. Comp. Sci. 378(3), 292–315 (2007)
23. Papadimitriou, C.H.: On selecting a satisfying truth assignment. In: Proc. of 32nd FOCS, pp. 163–169 (1991)
24. Pavesi, G., Mauri, G., Pesole, G.: An algorithm for finding signals of unknown length in DNA sequences. Bioinformatics 17, S207–S214 (2001)
25. Pevzner, P., Sze, S.: Combinatorial approaches to finding subtle signals in DNA strings. In: Proc. of 8th ISMB , pp. 269–278 (2000)
26. Proutski, V., Holme, E.C.: Primer master: A new program for the design and analyiss of PCR primers. CABIOS 12, 253–255 (1996)
27. Schöning, U.: A probabilistic algorithm for k-sat and constraint satisfaction problems. In: Proc. of 40th FOCS, pp. 410–414 (1999)
28. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm ususally takes polynomial time. In: Proc. of 33rd STOC, pp. 296–305 (2001)