Highly Scalable Algorithms for Robust String Barcoding[*]

Bhaskar DasGupta[1], Kishori M. Konwar[2], Ion I. Măndoiu[2], and Alex A. Shvartsman[2]

[1] Department of Computer Science, University of Illinois at Chicago
Chicago, IL 60607-7053, dasgupta@cs.uic.edu
[2] Computer Science and Engineering Department, University of Connecticut
371 Fairfield Rd., Storrs, CT 06269-2155, kishori,ion,aas@cse.uconn.edu

Abstract. String barcoding is a recently introduced technique for genomic-based identification of microorganisms. In this paper we describe the engineering of highly scalable algorithms for robust string barcoding. Our methods enable distinguisher selection based on *whole genomic sequences* of hundreds of microorganisms of up to bacterial size on a well-equipped workstation, and can be easily parallelized to further extend the applicability range to thousands of bacterial size genomes. Experimental results on both randomly generated and NCBI genomic data show that whole-genome based selection results in a number of distinguishers nearly matching the information theoretic lower bounds for the problem.

1 Introduction

String barcoding is a recently introduced technique for genomic-based identification of microorganisms such as viruses or bacteria. The basic barcoding problem [11] is formulated as follows: given the genomic sequences $g_1, \ldots g_n$ of n microorganisms, find a minimum number of strings $t_1, \ldots, t_k$ *distinguishing* these genomic sequences, i.e., having the property that, for every $g_i \neq g_j$, there exists a string t_l which is a substring of g_i or g_j, but not of both. A closely related formulation was independently proposed in [3], where it is assumed that it is possible to detect not just the presence or absence of a distinguisher t_i, but also the number of repetitions of t_i as a substring, up to a threshold of $R > 0$. The formulation in [11], which we adopt in this paper, corresponds to $R = 1$.

Identification is performed by spotting or synthesizing on a microarray the Watson-Crick complements of the distinguisher strings $t_1, \ldots, t_k$, and then hybridizing to the array the fluorescently labeled DNA extracted from the unknown microorganism. Under the assumption of perfect hybridization stringency, the hybridization pattern can be viewed as a string of k zeros and ones, referred to as the *barcode* of the microorganism. By construction, the barcodes corresponding to the n microorganisms are distinct, and thus the barcode uniquely identifies any one of them. To improve identification robustness, one may also require *redundant distinguishability* (i.e., at least m different distinguishers for every pair of microorganisms, where $m > 1$ is some fixed constant) and impose a lower bound on the edit distance between any pair of selected distinguishers [11].

The algorithms previously proposed for string barcoding are based on integer programming [11], and on Lagrangian relaxation and simulated annealing [3]. Unfortunately, the run-time of these algorithms does not scale well with the number of microorganisms and the length of the genomic sequences, e.g., the largest instance sizes reported in [11] have a total genomic sequence length of around 100,000 bases.

In this paper we describe the engineering of highly scalable algorithms for robust string barcoding. Our methods enable distinguisher selection based on *whole genomic sequences* of hundreds of microorganisms of up to bacterial size on a well-equipped workstation, and can be easily parallelized to further extend the applicability range to thousands of bacterial size genomes. Whole-genome based selection is beneficial in at least two significant ways. First, it simplifies assay design since the DNA of the unknown pathogen can be amplified using inexpensive general-purpose whole-genome amplification methods such as specialized forms of degenerate primer multiplex PCR [5] or multiple displacement amplification [8]. Second, whole-genome based selection results in a reduced number of distinguishers, often very close to the information theoretic lower bound of $\lceil \log_2 n \rceil$.

[*] The work of BD was supported in part by NSF grants CCR-0206795, CCR-0208749 and NSF CAREER grant IIS-0346973. The work of IIM was supported in part by a "Large Grant" from the University of Connecticut's Research Foundation.

Our algorithms are based on a simple greedy selection strategy – in every iteration we pick a substring that distinguishes the largest number of not-yet-distinguished pairs of genomic sequences. This selection strategy is an embodiment of the greedy setcover algorithm (see, e.g., [12]) for a problem instance with $O(n^2)$ elements corresponding to the pairs of sequences. Hence, by a classical result of [6, 9, 10], our algorithm guarantees an approximation factor of $2 \ln n$ for the barcoding problem. Very recently, Berman et al. [1] have shown that no approximation algorithm can guarantee a factor of $(1 - \epsilon) \ln n$ unless $NP = DTIME(n^{\log \log n})$, and also proposed an information content greedy heuristic achieving an approximation factor of $1 + \ln n$. Experimental results given in Section 5 show that our setcover greedy algorithm produces solutions of virtually identical quality to those obtained by the information content heuristic.

The setcover greedy algorithm is extremely versatile, and can be easily extended to handle redundancy and minimum edit distance constraints, as well as other biochemical constraints on individual distinguisher sequences. Furthermore, unlike the information content heuristic of [1], the greedy setcover algorithm can also take into account genomic sequence uncertainties expressed in the form of degenerate bases. Although degenerate bases are ubiquitous in genomic databases, previous works have not recognized the need to properly handle them. For example, experiments in [11] have implicitly treated degenerate bases in the input genomic sequences as distinct nucleotides; under this approach a substring of degenerate nucleotides such as NNNNN, might be erroneously selected as a distinguisher although it encodes for any possible substring of length 5.

To achieve high scalability, our implementation relies on several techniques. First, we use an incremental algorithm for quickly generating a representative set of candidate distinguishers and collecting all their occurrences in the given genomic sequences. To reduce the number of candidates, we avoid generating any substring that appears in all genomic sequences, which typically eliminates very short candidates. For each genomic sequence, we also generate only one of the substrings that appear exclusively in that sequence, this optimization eliminates from consideration most candidate distinguishers above a certain length. Unlike the suffix tree method proposed by Rash and Gusfield [11], our approach may generate multiple candidates that appear in the same set of k genomic sequences (for $1 < k < n$). However, the penalty of having to evaluate redundant candidates in the candidate selection phase is offset in practice by the faster candidate generation time. Finally, the efficient implementation of the greedy selection phase of algorithm combines a partition based method for computing the coverage gain of candidate distinguishers (this method was first proposed in the context of the information content heuristic in [1]) with a "lazy" strategy for updating coverage gains.

The rest of the paper is organized as follows. In Section 2 we give formal problem formulations and review previous work. In Section 3 we describe the efficient implementation of the setcover greedy algorithm for the basic string barcoding problem. In Section 4 we discuss the modifications required in the implementation for handling degenerate bases in input genomic sequences, redundancy and edit distance constraints, as well as biochemical constraints such as constraints on melting temperature and GC-content. In Section 5 we give the results of a comprehensive experimental study comparing, on both randomly generated and genomic data, our setcover greedy algorithms with other scalable methods including the information content heuristic and a recent set multicover randomized rounding approximation algorithm.

2 Preliminaries and Problem Formulation

Let $\Sigma = \{a, c, g, t\}$ be the DNA alphabet, and Σ^* be the set of string over Σ. A *degenerate base* is a non-empty subset of Σ. We identify degenerate bases of cardinality 1 with the respective non-degenerate bases. Given a DNA string $x = x_1 \dots x_k \in \Sigma^*$ and a string of degenerate bases $y = y_1 \dots y_n$, $n \geq k$, we say that

- x has a *perfect match* at position i of y iff $y_{i+j-1} = \{x_j\}$ for every $1 \leq j \leq k$,
- x has a *perfect mismatch* at position i of y iff there exists $1 \leq j \leq k$ such that $\{x_j\} \not\subseteq y_{i+j-1}$, and
- x has an *uncertain match* at position i of y iff $\{x_j\} \subseteq y_{i+j-1}$ for every $1 \leq j \leq k$, but $y_{i+j-1} \neq \{x_j\}$ for at least one j.

String $x = x_1 \dots x_k \in \Sigma^*$ *distinguishes* two sequences of degenerate bases y and z iff (a) x has a perfect match at one or more positions of y, and has perfect mismatches at all positions of z, or, symmetrically, (b) x has a perfect match at one or more positions of z, and has perfect mismatches at all positions of y. The robust string barcoding problem with degenerate bases is formulated as follows: *Given sequences of* *degenerate bases $g_1, \dots g_n$ and redundancy threshold m, find a minimum number of strings $t_1, \dots, t_k \in \Sigma^*$ such that, for every $i \neq j$, there exist m distinct strings t_l distinguishing g_i and g_j.*

It is easy to see that, for $m = 1$, at least $\lceil \log_2 n \rceil$ distinguishers are needed to distinguish any n genomic sequences. However, achieving this lower bound requires distinguishers that have perfect matches in nearly half of the sequences. In practice, additional constraints, such as lower bounds on the length of distinguishers, may result in no string having perfect matches in a large number of sequences, and therefore much more than a logarithmic number of distinguishers. The next theorem, the proof of which we omit due to space constraints, establishes under a simple probabilistic model that there is an abundance of distinguishers perfectly matching at least a constant fraction of the input sequences.

Theorem 1. *Consider a randomly generated instance of the string barcoding problem over a fixed alphabet Σ in which there are n strings, each string $s = s_0 s_1 \dots s_{\ell-1}$ is of length exactly ℓ generated independently randomly with $Pr[s_i = a] = 1/|\Sigma|$ for any i and any $a \in \Sigma$. Also assume that ℓ is sufficiently large compared to n. Then, for a random string $x \in \Sigma^*$ of length $O(\log \ell)$, the expected number of the input strings which contain x as a substring is pn for some constant $0 < p < 1$.*

Previous work. The robust string barcoding problem was introduced (for the case when genomic sequences contain no degenerate bases) by Rash and Gusfield [11]; they provided some experimental results based on integer programming methods, and left open the exact complexity and approximability of this problem. The problem without redundancy constraints was independently considered by Borneman *et al.* [3], who also considered non-binary distinguishability (based on detecting the multiplicity of a distinguisher as a substring) and a slightly more general problem in which the objective is to pick a given number of distinguishers maximizing the number of distinguished pairs. The main motivation for the formulations in [3] comes from minimizing the number of oligonucleotide probes needed for analyzing populations of ribosomal RNA gene (rDNA) clones by hybridization experiments on DNA microarrays. Borneman *et al.* provided computational results using Lagrangian relaxation and simulated annealing techniques, and noted that the problem is NP-hard assuming that the lengths of the sequences in the prespecified set were unrestricted. Very recently, Berman, DasGupta and Kao [1] considered a general framework for test set problems that captured the string barcoding problem and its variations; their main contribution is to establish theoretically matching lower and upper bounds on the worst-case approximation ratio. Cazalis et al. [4] have independently investigated similar greedy distinguisher selection strategies for string barcoding. Unlike our work, the algorithms in [4] consider only a small random subset of the possible distinguishers and also prescribe their length in order to achieve practical running time.

3 Efficient Implementation of the Greedy Setcover Algorithm

In this section we present the implementation of the setcover greedy algorithm in the context of the basic string barcoding problem, i.e., we disregard redundancy constraints and the presence of degenerate bases in the input sequences. Implementation modifications needed to handle the robust barcoding problem in its full generality are discussed in Section 4.

Our implementation of the setcover greedy algorithm has two main phases: a *candidate generation phase* and a *candidate selection phase*. In the candidate generation phase a representative set of candidate distinguishers is generated from the given genomic sequences. For each generated candidate, we also compute the list of sequences with which the candidate has perfect matches; this information is needed in the candidate selection phase. To reduce the number of candidates, we avoid generating any substring that appears in all genomic sequences, which typically eliminates very short candidates. For each genomic sequence, we also make sure to generate only one of the substrings that appear exclusively in that sequence, this optimization eliminates from consideration most candidate distinguishers above a certain length. Unlike the suffix tree method proposed by Rash and Gusfield [11], our approach may generate multiple candidates that appear in the same set of k genomic sequences (for $1 < k < n$). However, the penalty of having to evaluate redundant candidates in the candidate selection phase is offset in practice by the faster candidate generation time.

Efficient implementation of the above candidate elimination rules is achieved by generating candidates in increasing order of length and using exact match positions for candidates of length $l - 1$ when generating candidates of length l. For each position p in the input genomic sequences, we also maintain a flag to

```
Input: Set $C$ of candidate distinguishers
Output: Set $D$ of selected distinguishers
--------------------------------------------------------------------------------
$D \leftarrow \emptyset$; For every $c \in C$, $\Delta_{old}(c) \leftarrow \infty$
Repeat
    $\Delta^* \leftarrow 0$
    For every $c \in C$ with $\Delta_{old}(c) > \Delta^*$ do        // Since $\Delta(c, D) \leq \Delta_{old}(c)$, $c$ can be ignored if $\Delta_{old}(c) \leq \Delta^*$
        $\Delta_{old}(c) \leftarrow \Delta(c, D)$
        If $\Delta(c, D) > \Delta^*$ then $\Delta^* \leftarrow \Delta(c, D)$; $c^* \leftarrow c$
    If $\Delta^* > 0$ then $D \leftarrow D \cup \{c^*\}$
While $\Delta^* > 0$
```

Fig. 1. The greedy candidate selection algorithm

indicate whether or not the algorithm should evaluate candidate substrings starting at p. The possible values for the flag are TRUE (the substring of current length starting at p is a possible candidate), FALSE (we have already saved the substring of current length starting at p as a candidate), or DONE (all candidates containing as prefix the substring of current length starting at p are redundant, i.e., the position can be skipped for all remaining candidate lengths). Initially all flags are set to TRUE. The FALSE flags are reset to TRUE whenever we increment the candidate length, however, we never reset DONE flags. For every candidate length l, candidate evaluation proceeds sequentially over all positions of the genomic sequences. Whenever we reach a position p whose flag is set to TRUE, we use the list of matches for the substring of length $l - 1$ starting at p (or a linear time string matching algorithm if l is the minimum candidate length) to determine the list of matches for the substring of length l starting at p, and set the flag to FALSE for all positions where these matches occur. If the substring of length l starting at p has matches only within the source sequence, and we have already generated a "unique" candidate for this sequence, we discard the candidate and set the flag of p to DONE.

A further speed-up technique is to generate candidate distinguishers from a strict subset of the input sequences. Although this speed-up can potentially affect solution quality, the results in Section 5 show that the solution quality loss for whole-genome barcoding is minimal, even when we generate candidates based on a single input sequence, which corresponds to pre-assigning a barcode of all 1's to this sequence.

After the set of candidates is generated we select the final set of distinguishers in the greedy phase of the algorithm (Figure 1). We start with an empty set of distinguishers D. While there are pairs of sequences that are not yet distinguished by D, we loop over all candidates and compute for each candidate c the number $\Delta(c, D)$ of pairs of sequences that are distinguished by c but not by D, then add the candidate c with largest Δ value to D. Two sequences s and s' are distinguished by a candidate c iff exactly one of s and s' appears in the list P_c of perfect matches of c, which is available from the candidate generation phase. A simple method for computing Δ values is to maintain an $n \times n$ symmetric matrix indicating which of the pairs of sequences are already distinguished, and then to probe the $|P_c| \cdot (n - |P_c|)$ entries in this matrix corresponding to pairs (s, s') with $s \in P_c$ and $s' \notin P_c$ when computing $\Delta(c, D)$. A more efficient method is based on maintaining the partition defined on the set of sequences by D. If the partition defined by D consists of sets $S_1, \ldots, S_k$, then we can compute $\Delta(c, D)$ in $O(k + |P_c|) = O(n)$ time using the observation that

$$\Delta(c, D) = \sum_{i=1}^{k} |S_i \cap P_c| \cdot |S_i \setminus P_c| \tag{1}$$

In addition to the fast partition based computation, our implementation of the greedy selection phase uses a lazy strategy for updating the Δ values, based on the observation that they are monotonically non-increasing during the algorithm (see Figure 1).

4 Extended Barcoding Requirements

In this section we describe the modifications needed to the basic implementation given in previous section when handling practical extensions of the barcoding problem.

Degenerate bases. In the presence of degenerate bases in the input genomic sequences, the hybridization of a particular distinguisher may depend on which bases are actually present positions with degeneracy greater

Table 1. Average statistics for instances with 1,000 random sequences of length 10,000, redundancy 1, and number of source sequences varying from 1,000 down to 1.

#Source Seq.	1000	50	10	5	4	3	2	1
#Candidates	7213568.8	1438645.4	402700.5	225842.7	186871.3	146054.3	102800.6	55738.9
#Matches	55696250.9	35246584.9	23162063.6	18384468.1	16898179.9	15037610.7	12532936.0	8741587.0
Gen. time	132.3	44.7	35.5	31.4	31.3	30.6	28.1	24.9
Selection time	31.7	10.7	5.3	3.6	3.4	3.1	2.3	1.6
#Distinguishers	14.1	14.1	14.1	14.1	14.0	14.1	14.2	14.5

than 1. The greedy setcover algorithm takes into account this possibility for uncertain hybridization by only counting a pair (g, g') as distinguished by a candidate c if and only if c has a perfect match with one and only perfect mismatches with the other. For each generated candidate, in addition to the list of sequences that have only perfect matches we also save a list containing all sequences with at least one uncertain match. This allows fast computation of the (typically much longer) list of sequences having only perfect mismatches. To avoid generating candidate distinguishers containing degenerate bases, we set the DONE flag as soon as the corresponding substring extends past a degenerate base. Finally, since the partition of genomic sequences is no longer defined in the presence of uncertain hybridization, formula (1) is no longer applicable and we have to use the $n \times n$ "distinguished so far" matrix for computing Δ values.

Biochemical constraints on individual distinguishers. Since selected distinguishers must hybridize under the same experimental conditions, in practice it is natural to impose a variety of constraints on individual distinguishers, such as minimum and maximum length, GC content, melting temperature, etc. Furthermore, we may want to avoid using as distinguishers strings which appear in other organisms that may contaminate the sample. All individual constraints are easily incorporated as a simple filter in the candidate generation phase.

Redundancy constraints and minimum edit distance constraints. In practice, robust identification requires redundant distinguishability, i.e., more than one distinguisher distinguishing any given pair of genomic sequences. One may also impose a lower bound on the edit distance between any pair of selected distinguishers [11]. Taking into account redundancy requirements is done by maintaining the number of times each pair of genomic sequences has been distinguished. In order to incorporate the minimum edit distance constraint, after selecting a distinguisher we eliminate from consideration all candidates that are within an edit distance smaller than the given threshold.

5 Experimental Results

We performed experiments on both randomly generated instances and whole microbial genomes extracted from the NCBI databases [7]. Random testcases were generated from the uniform distribution induced by assigning equal probabilities to each of the four nucleotide; these testcases do not contain any nucleotides with degeneracy greater than 1. The NCBI testcase represents a selection of 29 complete microbial sequences, varying in length between 490,000 and 4,750,000 bases (over 76 million bases in total). All experiments were run on a PowerEdge 2600 Linux server with 4 Gb of RAM and dual 2.8 GHz Intel Xeon CPUs – only one of which is used by our sequential algorithms.

5.1 Algorithm Scalability

As described in Section 3, there are two main phases in the algorithm: candidate distinguisher generation, and greedy candidate selection. Figure 2 gives the average candidate selection CPU time for n random sequences of length 10,000 and redundancy 1, averaged over 10 instances of each size. Combining the two speed-up techniques for this phase (partition based coverage gain computation and lazy update of candidate gains) results in over two orders of magnitude reductions in runtime.

As mentioned in Section 3, a further speed-up technique is to generate candidate distinguishers only from a small number of "source" input sequences. Table 1 gives the average number of candidates, number of matches, runtimes for candidate generation and greedy selection, and number of selected distinguishers for instances with 1,000 random sequences of length 10,000 and redundancy 1, when the number of source

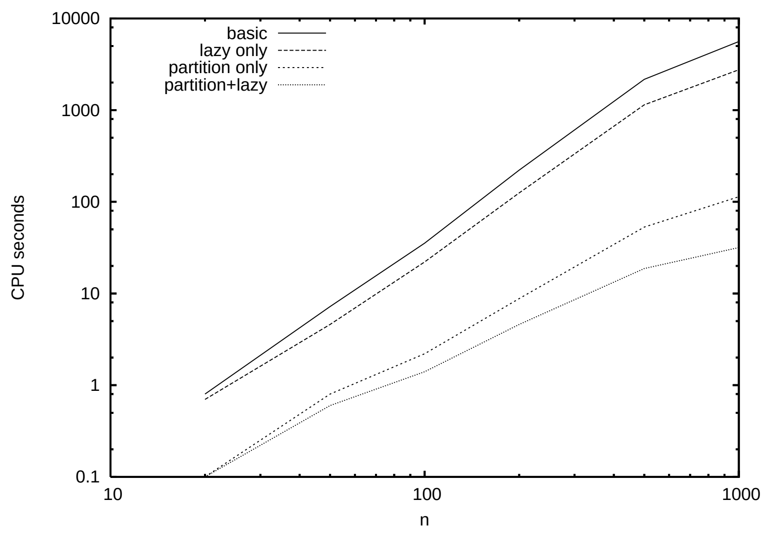

Fig. 2. Candidate selection CPU time (in seconds) for n random sequences of length 10,000 and redundancy 1, averaged over 10 instances of each size.

sequences is varied from 1,000 down to 1 (the source sequences were chosen at random). Although this speed-up can potentially affect solution quality, we found that on large instances the solution quality loss is minimal even when we generate candidates based on a single input sequence; this case corresponds to pre-assigning a barcode of all 1's to the source sequence. The technique reduces significantly both the memory requirement (which is proportional to the number of candidates and the number of times they match input sequences) and the runtime required for candidate generation and greedy selection.

5.2 Solution Quality on Random Data

Table 2 gives the number of distinguishers returned by the setcover greedy algorithm for redundancy varying between 1 and 20 on between 10 and 1,000 random sequences of length 10,000. For comparison we include in the table the results obtained by the information content heuristic results of [1], as well as the information theoretic lower bound of $\lceil \log_2 n \rceil$ for the case when the redundancy requirement is 1. We note that the number of distinguishers returned by the setcover greedy algorithm is virtually identical to that returned by the information content heuristic, despite the latter one having a better approximation guarantee [1]. Furthermore, the results for redundancy one are within 50% of the information theoretic lower bound for the range of instance sizes considered in this experiment. The gap between the solutions returned by the algorithms and the lower bound does increase with the number of sequences; however it is not clear how

Table 2. Number of distinguishers returned by the setcover greedy algorithm (SGA) for varying redundancy and number of sequences. For each value of n we report the average over 10 testcases, each consisting of n random sequences of length 10,000. For comparison we include information content heuristic results (ICH) and the information theoretic lower bound of $\lceil \log_2 n \rceil$ for redundancy 1 (LB).

Algorithm	r	$n = 10$	$n = 20$	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
LB	1	4	5	6	7	8	9	10
ICH	1	4.0	5.0	7.0	8.0	10.0	12.2	14.1
SGA	1	4.0	5.0	7.0	8.0	10.0	12.3	14.1
SGA	2	6.7	8.3	10.6	12.5	14.1	16.7	18.9
SGA	3	8.8	11.6	13.6	15.5	17.3	20.1	22.4
SGA	4	10.8	14.0	16.5	18.7	20.7	23.5	26.1
SGA	5	13.6	16.6	19.5	21.5	23.7	26.8	29.5
SGA	10	22.5	26.8	32.0	34.6	37.5	41.7	44.9
SGA	20	43.0	47.6	55.6	59.5	63.4	68.0	72.6

Table 3. Number of distinguishers returned by the setcover greedy algorithm (SGA) and the multi-step rounding algorithm in [2] (RND) for varying redundancy and number of sequences. For each value of n we report the average over 10 testcases, each consisting of n random sequences of length 1,000. Boldface entries correspond to instances for which the multi-step rounding algorithm has better solution quality than setcover greedy.

Algorithm	r	$n = 10$	$n = 20$	$n = 50$	$n = 100$	$n = 200$
SGA	1	4.0	5.0	7.0	9.0	11.0
RND	1	5.0	6.8	10.5	13.0	16.0
SGA	2	6.3	8.2	11.2	12.9	15.0
RND	2	7.3	10.7	14.8	17.0	20.4
SGA	5	13.2	16.1	19.5	22.4	24.6
RND	5	13.2	18.2	23.5	27.3	31.2
SGA	10	22.8	27.0	32.1	36.1	39.4
RND	10	**20.2**	30.9	37.4	41.9	48.3
SGA	20	43.4	48.8	57.0	61.0	65.8
RND	20	**38.9**	50.7	62.6	69.4	76.2
SGA	50	100.9	112.0	125.6	133.8	142.0
RND	50	**92.6**	**107.8**	**125.2**	141.6	159.5
SGA	100	195.0	217.2	239.0	255.5	264.0
RND	100	**184.9**	**205.2**	**236.0**	270.0	289.0

much of this increase is caused by degrading algorithm solution quality, and how much by degrading lower bound quality.

We also compared our setcover greedy algorithm with a recently proposed multi-step rounding algorithm for set multicover [2]. The rounding algorithm has the following steps:

1. Solve the fractional relaxation of the natural integer program formulation of problem [11] (we used the commercial solver CPLEX 9.0 for implementing this step)
2. Scale the fractional solution by an appropriate constant factor (see [2] for details)
3. Deterministically select all distinguishers with a scaled fractional value exceeding 1
4. Randomly select a subset of the remaining candidates, each candidate being chosen with a probability equal to the scaled fractional value
5. Finally, if the selected set of distinguishers is not yet feasible, add further distinguishers using the setcover greedy algorithm

The approximation guarantee established in [2] for the general set multicover problem translates into an approximation factor of $2 \ln n - \ln r$ for robust string barcoding with redundancy r, which suggests that the multi-step rounding algorithm is likely to improve upon the setcover greedy for high redundancy constraints. Table 3 gives the results of experiments comparing the setcover greedy and multi-step rounding algorithms on testcases consisting of up to 200 random sequences, each of length 1,000 for redundancy requirement ranging from 1 to 100. The results confirm that the multi-step rounding algorithm has better solution quality than setcover greedy when redundancy requirement is very large relative to the number of sequences (entries typeset in boldface), yet the setcover greedy still has the best performance for most combinations of parameters.

5.3 Experiments on Genomic Data

We ran our algorithm on a set of 29 complete microbial genomic sequences extracted from NCBI databases [7]. Sequence lengths in the set vary between 490 Kbases and 4.75 Mbases, with an average length of 2.6 Mbases (over 76 Mbases total). Unlike random testcases, the sequences in the NCBI data set contain a small number of degenerate bases, 861 bases in total. Therefore, we cannot use the partition method for computing the number of sequence pairs distinguished by a candidate in the greedy selection phase, and we have to use the slower matrix datastructure. In these experiments we varied the redundancy requirement from 1 to 20. To see the effect of length and edit distance requirements on the number of distinguishers, for each redundancy requirement we computed both an unconstrained solution, and a solution in which distinguishers must have length between 15 and 40, and there should be a minimum edit distance of 6 between every two selected

Table 4. Results on a set of 29 NCBI complete microbial genomes. Candidate generation time is approximately 335 seconds for all combinations of parameters.

Redundancy	l_{min}	l_{max}	MinEdit	Select time	#Distinguishers
1	0	∞	0	14.2	6.0
1	15	40	6	2.6	8.0
5	0	∞	0	20.3	21.0
5	15	40	6	8.7	31.0
10	0	∞	0	22.9	41.0
10	15	40	6	16.4	60.0
20	0	∞	0	26.8	76.0
20	15	40	6	33.4	123.0

distinguishers (these values are similar to those used in [11]. In all experiments, we generated candidates based only on the shortest sequence of 490 Kbases.

The results on the NCBI dataset are given in Table 4. Naturally, meeting higher redundancy constraints requires more distinguishers to be selected. Additional length and edit distance constraints further increase the number of distinguishers, but the latter is still within reasonable limits. The length constraints reduce the number of candidates (from 1,775,471 to 122,478), which, for low redundancy values has the effect of reducing greedy selection time. However, for high redundancy requirements the reduction in number of candidates is offset by the increase in solution size, and greedy selection becomes more time consuming with length and edit distance than without (selection time grows roughly linearly with solution size).

Acknowledgments

The authors would like to thank Claudia Prajescu for her help with the implementation of the multi-step rounding algorithm in [2].

References

1. P. Berman, B. DasGupta, and M.-Y. Kao. Tight approximability results for test set problems in bioinformatics. In *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *Lecture Notes in Computer Science*, pages 39–50, Berlin, 2004. Springer-Verlag.
2. P. Berman, B. DasGupta, and E. Sontag. Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks. In *Proc. 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 2748 of *Lecture Notes in Computer Science*, pages 39–50, Berlin, 2004. Springer-Verlag. (Also to appear in a special issue on computational biology of *Discrete Applied Mathematics.*).
3. J. Borneman, M. Chrobak, G.D. Vedova, A. Figueora, and T. Jiang. Probe selection algorithms with applications in the analysis of microbial communities. *Bioinformatics*, 1:1–9, 2001.
4. D. Cazalis, T.Milledge, and G. Narasimhan. Probe selection problem: Structure and algorithms. In *Proc. 8th Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004)*, pages 124–129, 2004.
5. V.G. Cheung and S.F. Nelson. Whole genome amplification using a degenerate oligonucleotide primer allows hundreds of genotypes to be performed on less than one nanogram of genomic dna. *Proc. Natl. Acad. Sci. USA.*, 93:14676–14679, 1996.
6. V. Chvátal. A greedy heuristic for the set covering problem. *Math. of Op. Res.*, 4:233–235, 1979.
7. NCBI Completed Microbial Genomes. http://www.ncbi.nlm.nih.gov/genomes/microbes/complete.html, 2004.
8. F.B. Dean, S. Hosono, L. Fang, X. Wu, A. Fawad Faruqi, P. Bray-Ward, Z. Sun, Q. Zong, Y. Du, J. Du, M. Driscoll, W. Song, S.F. Kingsmore, M. Egholm, and R.S. Lasken. Comprehensive human genome amplification using multiple displacement amplification. *Proc. Natl. Acad. Sci. USA.*, 99:5261–5266, 2002.
9. D.S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Sys. Sci.*, 9:256–278, 1974.
10. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.
11. S. Rash and D. Gusfield. String barcoding: Uncovering optimal virus signatures. In *Proc. 6th Annual International Conference on Computational Biology*, pages 254–261, 2002.
12. V.V. Vazirani. *Approximation Algorithms.* Springer-Verlag, Berlin, 2001.