A Data-Based Coding of Candidate Strings in the Closest String Problem

Bryant A. Julstrom
Department of Computer Science
St. Cloud State University
St. Cloud, MN 56301 USA
julstrom@stcloudstate.edu

ABSTRACT
Given a set of strings $S$ of equal lengths over an alphabet $\Sigma$, the closest string problem seeks a string over $\Sigma$ whose maximum Hamming distance to any of the given strings is as small as possible. A data-based coding of strings for evolutionary search represents candidate closest strings as sequences of indexes of the given strings. The string such a chromosome represents consists of the symbols in the corresponding positions of the indexed strings. A genetic algorithm using this coding was compared with two GAs that encoded candidate strings directly as strings over $\Sigma$. In trials on twenty-five instances of the closest string problem with alphabets ranging in size from 2 to 30, the algorithm that used the data-based representation of candidate strings consistently returned the best results, and its advantage increased with the sizes of the test instances’ alphabets.

Categories and Subject Descriptors
G.2.1 [Mathematics of Computing]: Discrete Mathematics—Combinatorics; I.2.8 [Problem Solving, Control Methods, and Search]: Heuristic Methods

General Terms
Algorithms

Keywords
Closest string problem, data-based coding, genetic algorithm

1. INTRODUCTION
Given a collection of numbers, a representative value will be identified by their mean, their median, or some other measure of central tendency. Given a collection of strings, a representative string might be one that is as close as possible to the given strings; in particular, one whose largest Hamming distance to any given string is as small as possible. The closest string problem seeks such a string.

The obvious coding of candidate strings for evolutionary search in this problem is directly as strings over the alphabet of the given strings, and this coding can be restricted at each position to just the symbols that actually appear there in the given strings. Alternatively, a data-based coding represents candidate strings as indexes of the given strings. The symbol in each position of the string such a chromosome represents is the symbol in that position of the indexed given string. By restricting the symbols in a candidate string to the symbols that appear in the given strings and by biasing choices of these symbols in proportion to their appearances in the given strings, this coding enables better performance in an evolutionary algorithm for the closest string problem than do direct representations.

Three genetic algorithms use the codings just suggested, and tests on twenty-five sets of ten strings each confirm our expectations. At the cost of a small increase in execution time, the GA that uses the data-based representation outperforms the other two.

Section 2 below describes the closest string problem in detail. Sections 3, 4, and 5 present a simple direct coding of candidate closest strings, a restricted string coding, and the data-based coding, respectively. Section 6 presents genetic algorithms for the closest string problem that use the codings, and Section 7 describes the twenty-five test problem instances. Section 8 presents and discusses the results of the trials of the three GAs on the test instances, which indicate the relative effectiveness of the data-based coding in evolutionary algorithms for the closest string problem.

2. THE PROBLEM
The Hamming distance $d_H(s, t)$ between two strings $s$ and $t$ of equal length is the number of positions in which they differ. For example, if $s = “ACCAG”$ and $t = “TCCCG,”$ the Hamming distance between them is $d_H(s, t) = 2$.

Given a set of $n$ strings $S = \{s_1, s_2, \ldots, s_n\}$ of uniform length $l$ over an alphabet $\Sigma$, the closest string problem (CSP) seeks a string $t$ of length $l$ over $\Sigma$ whose maximum Hamming distance from any string in $S$ is as small as possible; that is, that minimizes

$$\max_i(d_H(t, s_i)).$$

Such a string is called a center of the set of strings $S$; for this reason the CSP is sometimes called the center string problem. Note that a center is not in general unique.


An obvious candidate for a center of a set $S$ of strings is the majority string or consensus string $m$, each of whose positions holds a symbol that is most common in that position in the strings of $S$. In general, however, a consensus
string—like a center, it need not be unique—is not a center of \( S \). For example, if \( s_1 = \text{"ACCGT"} \), \( s_2 = \text{"AACTT"} \), and \( s_3 = \text{"CAGGC"} \), their consensus string is \( m = \text{"AACGT"} \), whose Hamming distances to \( s_1 \), \( s_2 \), and \( s_3 \) are 1, 1, and 3, respectively. However, the string \( \text{"CACGT"} \) has Hamming distance 2 to each of the \( s_i \).

We have stated the CSP as an optimization problem: Minimize the maximum Hamming distance between a candidate center string and the strings in \( S \). In the corresponding decision problem, we are given a positive integer \( d \) and asked if there exists a string over \( \Sigma \) whose Hamming distances to the strings in \( S \) do not exceed \( d \). Gramm, Niedermeier, and Rossmanith [4] have showed that when \( d \) is constant, the decision problem can be solved in time that is linear in the maximum Hamming distance between any two strings in \( S \).

Applications of the closest string problem often generate problem instances whose strings are identical at many positions. The common symbol at each such position is immediately the appropriate symbol in a closest string, so these positions, called monomorphic alleles, can be removed from the problem. We assume in the following discussion that this has been done.

Li, Ma, and Wang [6] described polynomial-time approximation schemes for the closest string and closest substring problems. Meneses et al. [10] presented three integer programming formulations of the CSP and a heuristic for it as well as a branch-and-bound algorithm based on one of the IP formulations. Gomes et al. [3] described a parallel implementation of a local-search heuristic for the problem.

Gasieniec, Jansson, and Lingas [2] presented several approximation algorithms for the Hamming radius-\( p \) clustering problem, which seeks \( p \) strings that minimize the maximum Hamming distance from each of the given strings to the nearest of the \( p \) strings. Note that this problem reduces to the CSP when \( p = 1 \).

Researchers have described several genetic algorithms for the CSP. In a GA by Mauch, Melzer, and Hu [9], chromosomes were strings over the alphabet \( \{A, C, G, T\} \). Their coding is reprinted and extended in Section 3 below. Liu, He, and Sykora [7] described parallel implementations of a genetic algorithm and simulated annealing for the problem. Liu et al. [8] described a GA-SA hybrid that included a heuristic mutation operator.

3. A STRING CODING

Given a set \( S \) of strings over an alphabet \( \Sigma \), a candidate closest string is also a string over \( \Sigma \), so the obvious coding of candidate strings for evolutionary search in the CSP is directly as such strings, as used by Mauch, Melzer, and Hu [9] with \( \Sigma = \{A, C, G, T\} \) and by others. We extend this coding trivially to whatever alphabet \( \Sigma \) underlies the given strings \( S \). The fitness of a chromosome/string is the largest of the Hamming distances from it to each of the given strings.

It is straightforward to generate random chromosomes: Choose at random from \( \Sigma \) for each position. These chromosomes support the usual positional variation operators: \( k \)-point or uniform [12] crossover and position-by-position mutation. The latter, with small probability, randomly replaces each symbol in a parent chromosome.

4. A RESTRICTED STRING CODING

In the set \( S \) of given strings, particular symbols may not appear in particular locations. For example, in a set \( S \) of strings over \( \Sigma = \{A, C, G, T\} \), the symbol \( \text{\char'\^} \) may never appear in position 27 of any string. The straightforward coding described in the previous section, however, may place \( \text{\char'\^} \) there, clearly a poor choice since any other would reduce the candidate string’s Hamming distance to some of the given strings.

A simple extension of the string coding, then, restricts the symbols at each position to those that appear there in the given strings. This guarantees that every position in every chromosome matches at least one given string.

The selection of symbols for “random” chromosomes is now restricted to the symbols that appear at each position, as is the selection of replacement symbols in mutation. Crossover is unaffected.

To support this coding, it is necessary to keep track of the symbols that appear at each position in the given strings. Building the necessary data structure requires a little time initially, but once built, an appropriate symbol can be randomly selected from it for each position in constant time.

5. A DATA-BASED CODING

The restricted string coding of Section 4 clearly represents an improvement over the naive coding of Section 3, but it has two flaws. First, it requires an additional data structure to keep track of the symbols allowed at each position. Second, it does not reflect the relative frequencies of symbols at each position in the given strings. If \( S \) contains 50 strings, of which 48 have the symbol \( \text{\char'\^} \) at position 27 while the remaining two have \( \text{\char'\^} \) there, random generation of chromosomes and their mutation will place \( \text{\char'\^} \) and \( \text{\char'\^} \) at position 27 with equal probabilities, thus biasing the search away from likely close strings.

To overcome both these difficulties, we propose a coding based on the particular strings in the problem instance. In particular, encode candidate closest strings not directly as strings over the alphabet \( \Sigma \) of the given strings but rather as strings over the indexes of the given strings. The string such a chromosome of indexes represents consists of the symbols in the corresponding positions of the indexed given strings.

That is, given \( n \) strings \( S = \{s_1, s_2, \ldots, s_n\} \) of length \( \ell \), a chromosome \( c[I] \) is a string of length \( \ell \) over \( \{1, 2, \ldots, n\} \).

The \( i \)-th symbol in the string \( t \) over \( \Sigma \) that \( c[I] \) represents is the \( i \)-th symbol in \( s_{c[i]} \): \( t[i] = s_{c[i]}[i] \).

Figure 1 shows an example of such a chromosome. In it, \( c[I] = 2 \), so the first symbol in the string the chromosome represents is \( \text{\char'\^} \), the first symbol in \( s_2 \), and so on.

The fitness of such a chromosome is again the maximum distance of the string it represents to any given string. It is straightforward to generate random chromosomes, and as with the two direct string encodings, positional variation operators can generate offspring from parent chromosomes.

This coding improves on the simple and restricted string
The chromosome consists of symbols from the given strings at the corresponding positions: \( t[i] = s_{c[i]} [i] \).

Figure 1: A data-based chromosome. Chromosomes are strings over \( \{1, 2, 3, 4, 5\} \). The string \( t[i] \) that \( c[i] \) represents consists of symbols from the given strings \( S \) at the corresponding positions: \( t[i] = s_{c[i]} [i] \).

Three generational genetic algorithms implement the simple string coding, the restricted string coding, and the data-based coding; call them GA1, GA2, and GA3, respectively. Each GA initializes its population with appropriate random chromosomes. They select chromosomes to be parents in \( k \)-tournaments without replacement, and they apply either crossover or mutation, never both, to generate each offspring chromosome. The crossover operator is uniform crossover, and mutation is position-by-position: with a small probability, each symbol is replaced with a random one. The GAs are 1-elitist; a single best chromosome from the current generation is copied unaltered into the next, and they run through fixed numbers of generations.

The algorithms were implemented in C++ and executed on a Pentium 4 processor with 1GByte of RAM running at 2.53 GHz under Red Hat Linux 9.0. In the comparisons that Section 8 reports, on CSP instances with \( n \) given strings of length \( m \), the GAs’ populations contained 125 chromosomes. The probability with which they applied crossover was 0.60, the probability of mutation therefore 0.40. In mutation, the probability that any one symbol was changed was 0.01. Each GA initializes its population with appropriate random chromosomes.

8. COMPARISONS

The three genetic algorithms were compared on twenty-five randomly-generated instances of the closest string problem; that is, on twenty-five sets of given strings. Each instance consists of ten strings of length \( \ell = 500 \). The sizes of the instances’ alphabets are 2, 4, 10, 20, and 30; five instances were generated with each.

Closest strings and the maximum distances of them to the instances’ given strings are not known, but as noted in Section 2, half the Hamming diameter for a set of strings is a lower bound on the maximum distance of a closest string to any of the given strings. The algorithm sizes do not exceed the number of strings, so we expect that few symbols will be missing from any of the given strings’ positions. The restriction in GA2 to the symbols that appear in each position of the given strings confers no discernible advantage.

Because this coding is based on the strings in the current target instance of the CSP, we call it data-based. We expect it to provide better performance in evolutionary search in the CSP than those with the direct string codings.

The performances of the three algorithms are similar, but there are consistent trends in the data. First, the results of the simple string-coded GA1 and the restricted string-coded GA2 are essentially the same on the instances with alphabet sizes \( |\Sigma| = 2, 4, \) and 10. On these instances, the alphabet size does not exceed the number of strings, so we expect that few symbols will be missing from any of the given strings’ positions. The restriction in GA2 to the symbols that appear in each position of the given strings confers no discernible advantage.

When significant numbers of symbols are certain to be missing from the given strings’ positions, the restriction of candidate closest strings to the symbols that do appear generates strings that in fact are closer in Hamming distance to the given strings.

The performances of the three algorithms are similar, but there are consistent trends in the data. First, the results of the simple string-coded GA1 and the restricted string-coded GA2 are essentially the same on the instances with alphabet sizes \( |\Sigma| = 2, 4, \) and 10. On these instances, the alphabet size does not exceed the number of strings, so we expect that few symbols will be missing from any of the given strings’ positions. The restriction in GA2 to the symbols that appear in each position of the given strings confers no discernible advantage.

The three genetic algorithms were compared on twenty-five randomly-generated instances of the closest string problem; that is, on twenty-five sets of given strings. Each instance consists of ten strings of length \( \ell = 500 \). The sizes of the instances’ alphabets are 2, 4, 10, 20, and 30; five instances were generated with each.

Closest strings and the maximum distances of them to the instances’ given strings are not known, but as noted in Section 2, half the Hamming diameter for a set of strings is a lower bound on the maximum distance of a closest string to any of the given strings. The algorithm sizes do not exceed the number of strings, so we expect that few symbols will be missing from any of the given strings’ positions. The restriction in GA2 to the symbols that appear in each position of the given strings confers no discernible advantage.

When significant numbers of symbols are certain to be missing from the given strings’ positions, the restriction of candidate closest strings to the symbols that do appear generates strings that in fact are closer in Hamming distance to the given strings.

The performances of the three algorithms are similar, but there are consistent trends in the data. First, the results of the simple string-coded GA1 and the restricted string-coded GA2 are essentially the same on the instances with alphabet sizes \( |\Sigma| = 2, 4, \) and 10. On these instances, the alphabet size does not exceed the number of strings, so we expect that few symbols will be missing from any of the given strings’ positions. The restriction in GA2 to the symbols that appear in each position of the given strings confers no discernible advantage.

When significant numbers of symbols are certain to be missing from the given strings’ positions, the restriction of candidate closest strings to the symbols that do appear generates strings that in fact are closer in Hamming distance to the given strings.

Fourth, the times of the three GAs differ little on the instances with \( |\Sigma| = 2 \), to the slight disadvantage of GA3, but on the instances with larger alphabets, GA2 usually takes slightly longer than GA1, and GA3 slightly longer than GA2. These results are reasonable. To the computations in GA1, GA2 adds some function calls associated with choosing from a restricted set of symbols at each position, and GA3 adds additional steps in the evaluation of each chromosome to build the candidate string it represents from the given strings indexed by the chromosome’s values.

Finally, all three algorithms’ times diminish as the sizes of the instances’ alphabets increase. This effect is most pronounced for GA1, less so for GA2. It is smallest for GA3, but still apparent. It is not clear why the alphabet size should affect the algorithms’ times in this way, though its lesser impact on GA3 is likely due to the fact that nothing in GA3 responds directly to the size of the alphabet. The alphabet size is imposed implicitly by the symbols in the given strings but, in contrast to its necessary inclusion in GA1 and GA2, it appears nowhere in GA3’s code.

This points out an additional advantage of the data-based coding. Because the symbols in candidate closest strings are copied from the given strings, the code of GA3 depends not...
Table 1: The results of the sets of 40 trials of the three genetic algorithms on the CSP test instances. For each instance, the table lists its size, number, and Hamming diameter. For each set of trials on each instance, the table lists the best (smallest) maximum Hamming distance of a string to the given strings found in the entire set of trials, the mean of the trials’ best values, the standard deviation $s$ of these values, and the mean time $\bar{t}$ of the trials.

| Instance $n = |\Sigma|$ | Num $D_H(S)$ | best | GA1 | GA2 | GA3 |
|-------------------------|-------------|------|-----|-----|-----|
|                         |             | $\bar{x}$ | mean | $s$ | $\bar{t}$ | best | mean | $s$ | $\bar{t}$ | best | mean | $s$ | $\bar{t}$ |
| 2                       | 1           | 189 | 189.8 | 0.53 | 23.5 | 189 | 189.8 | 0.52 | 21.9 | 188 | 189.3 | 0.69 | 22.5 |
| 2                       | 2           | 188 | 189.0 | 0.51 | 22.9 | 188 | 188.9 | 0.56 | 21.8 | 187 | 188.5 | 0.88 | 22.4 |
| 3                       | 281         | 192 | 193.9 | 0.58 | 21.7 | 193 | 193.8 | 0.54 | 21.9 | 191 | 192.9 | 0.91 | 21.4 |
| 4                       | 273         | 197 | 197.3 | 0.45 | 21.8 | 196 | 197.2 | 0.53 | 22.0 | 194 | 195.9 | 0.94 | 24.5 |
| 5                       | 275         | 190 | 191.6 | 0.59 | 23.4 | 190 | 191.5 | 0.60 | 21.9 | 189 | 191.0 | 0.93 | 22.4 |
| 4                       | 398         | 294 | 295.5 | 0.93 | 21.9 | 295 | 296.0 | 0.75 | 22.0 | 292 | 293.8 | 0.93 | 23.0 |
| 4                       | 394         | 295 | 296.0 | 0.81 | 21.8 | 296 | 296.7 | 0.62 | 22.1 | 293 | 294.0 | 0.68 | 23.0 |
| 3                       | 398         | 294 | 295.6 | 0.71 | 21.8 | 295 | 296.2 | 0.70 | 22.0 | 292 | 293.3 | 0.89 | 23.1 |
| 4                       | 392         | 296 | 298.3 | 0.86 | 22.0 | 297 | 299.0 | 0.80 | 22.1 | 294 | 295.9 | 0.90 | 23.0 |
| 5                       | 397         | 297 | 298.6 | 0.84 | 21.7 | 297 | 299.0 | 0.86 | 22.0 | 294 | 296.3 | 0.75 | 22.9 |
| 10                      | 461         | 370 | 371.8 | 0.95 | 19.9 | 370 | 371.7 | 0.92 | 20.3 | 364 | 365.0 | 0.36 | 21.8 |
| 2                       | 466         | 369 | 371.0 | 0.97 | 20.0 | 369 | 370.4 | 0.86 | 22.5 | 363 | 364.2 | 0.65 | 21.9 |
| 3                       | 463         | 372 | 373.8 | 0.99 | 19.8 | 372 | 374.1 | 0.88 | 22.3 | 366 | 367.1 | 0.47 | 21.8 |
| 4                       | 466         | 374 | 375.1 | 0.83 | 19.8 | 373 | 375.3 | 0.93 | 20.2 | 368 | 368.4 | 0.50 | 21.6 |
| 5                       | 465         | 371 | 373.0 | 1.00 | 21.0 | 371 | 373.5 | 0.93 | 20.3 | 365 | 366.3 | 0.68 | 21.7 |
| 20                      | 489         | 406 | 408.1 | 1.11 | 18.2 | 404 | 405.4 | 0.74 | 21.1 | 395 | 397.3 | 0.76 | 20.5 |
| 2                       | 486         | 405 | 406.7 | 1.04 | 18.2 | 402 | 403.6 | 0.81 | 19.0 | 394 | 395.5 | 0.64 | 20.6 |
| 3                       | 486         | 406 | 407.6 | 0.93 | 18.2 | 403 | 404.8 | 0.97 | 19.0 | 396 | 396.9 | 0.72 | 20.6 |
| 4                       | 485         | 406 | 407.3 | 0.95 | 18.3 | 403 | 404.7 | 0.86 | 19.0 | 395 | 396.5 | 0.72 | 20.6 |
| 5                       | 483         | 406 | 407.6 | 0.93 | 18.3 | 403 | 404.8 | 1.14 | 19.0 | 395 | 396.5 | 0.60 | 20.6 |
| 30                      | 491         | 420 | 422.2 | 1.42 | 17.5 | 415 | 416.2 | 0.70 | 18.4 | 408 | 408.7 | 0.68 | 20.0 |
| 2                       | 489         | 420 | 422.4 | 0.95 | 17.6 | 414 | 416.5 | 0.90 | 18.5 | 407 | 408.3 | 0.62 | 20.2 |
| 3                       | 492         | 418 | 420.8 | 1.26 | 17.7 | 412 | 413.6 | 0.90 | 18.5 | 403 | 405.3 | 0.92 | 20.1 |
| 4                       | 492         | 419 | 423.6 | 1.38 | 17.5 | 416 | 417.9 | 0.81 | 18.3 | 409 | 410.5 | 0.81 | 19.9 |
| 5                       | 491         | 421 | 422.5 | 1.08 | 18.8 | 415 | 416.4 | 0.90 | 18.4 | 407 | 408.2 | 0.69 | 20.0 |

at all on the particular alphabet underlying those strings. Thus the algorithm can be applied to CSP instances over any alphabet with no particular accommodation of the instances’ alphabet sizes.

9. CONCLUSION

Given a collection $S$ of strings of length $\ell$ over an alphabet $\Sigma$, a center is a string of length $\ell$ over $\Sigma$ whose maximum Hamming distance to any of the strings of $S$ is as small as possible. The NP-hard closest string problem seeks a center.

The obvious coding of candidate centers for evolutionary search is directly as strings over $\Sigma$, but a data-based coding represents candidate centers as strings of indexes of the strings in $S$. The string that such a chromosome represents consists of the symbols in the corresponding positions of the indexed given strings. Two genetic algorithms for the CSP represent candidate centers directly, while a third uses the data-based representation.

In comparisons on twenty-five instances of the CSP, each consisting of ten strings of length $\ell = 500$ over alphabets of from 2 to 30 symbols, the GA that used the data-based coding consistently returned the best results. Its advantage was small on the instances over a binary alphabet but increased with alphabet size. These results indicate that the data-based coding of candidate strings is relatively more effective than direct codings for evolutionary search in the closest string problem, and an algorithms that uses this coding enjoys the additional advantage of requiring in its code no acknowledgment or accommodation of the alphabet sizes of the CSP instances it addresses.

10. REFERENCES


