This paper appeared in the 6th International Conference on Genetic Algorithms (ICGA'95), Pittsburgh, Pennsylvania, July 15–19, 1995.

A Preliminary Study of Genetic Data Compression

Wee K. Ng* Chinya V. Ravishankar

Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122

Abstract

Data compression is the process of creating binary representations of data which require less storage space than the original data. Compression is achieved by exploiting redundancy in the source data. In text compression, this redundancy manifests in the form of frequently occurring strings of arbitrary lengths. It is known that optimal identification of these strings is NP-complete for a given text (Storer and Szymanski 1982). In this paper, we present a method for approximating optimal identification using genetic algorithms. Our preliminary study examined the strengths and weaknesses of this approach.

1 INTRODUCTION

Data compression is the process of creating compressed binary representations of data from sources such as speech, images or texts (Gray 1990, Veldhuis 1993, Williams 1991). Lossless compression techniques are used when perfect reproductions are required, while lossy compression techniques are used when perfect reproductions are not possible or require too many bits. In this paper, we focus our attention on lossless compression, in particular, the compression of texts (Bell, Cleary and Witten 1990).

Due to its lossless requirement, the primary paradigm for text compression is string substitution. Frequently occurring strings in the text are detected and substituted by shorter codes to achieve compression. These codes permit perfect reproduction of the original text. The strings and their corresponding codes are stored in a *dictionary*. Hence, these techniques are commonly referred to as dictionary techniques.

A dictionary technique usually performs two passes through the given text. During the first pass, it segments the text looking for strings to construct the dictionary. It then performs the actual coding using the dictionary during the second pass. The dictionary contents affect the compression efficiency because they determine not only which strings will be substituted but their lengths as well. Since the construction of an optimal dictionary for a given text is NP-complete (Storer and Szymanski 1982), optimal text compression is hard.

Genetic algorithms (Goldberg 1989, Holland 1992) are becoming a widely-used and accepted method for very difficult combinatorial problems (DeJong and Spears 1989). In this paper, we present a genetic algorithm to construct a dictionary for a given text. To the best of our knowledge, this is the first application of its kind to the domain of data compression.

The organization of this paper is as follows. In Section 2, we formulate a precise statement of the dictionary construction problem. Section 3 briefly describes the conventional algorithm for computing dictionaries. We next describe chromosome representations and genetic operators for the genetic algorithm in Section 4. Section 5 presents some performance measurements of the proposed genetic algorithm. Finally, the last section concludes with a summary of our work and directions for future research.

2 PROBLEM FORMULATION

Let A be a finite set of symbols or *alphabet*. A^+ is the set of all non-empty, finite strings $w = a_1 a_2 \cdots a_n$, $n \geq 0$, $a_i \in A$, composed of symbols from A. A text file (or simply text) T is a string from A^+; i.e., $T \in A^+$. Let $|T|$ denote the number of characters or bytes in the text. Note that $|T| \times 8$ gives the number of binary bits required to store T, assuming that each symbol is a byte of 8 bits.

A dictionary $D = \langle W, C \rangle$ consists of a finite, ordered set W of $m > 0$ strings and its corresponding ordered set C of codes, i.e., for each string w_i in W, there

*The author may also be reached at "wkn@eecs.umich.edu" and at URL: "http://www.eecs.umich.edu/~wkn".

Word (W)	Frequency	Probability	Huffman code (C)
a	4	0.0513	1111
nh	5	0.0641	1110
nf	5	0.0641	1101
bo	6	0.0769	1100
ee	9	0.1154	101
il	9	0.1154	100
hin	10	0.1282	011
nd	10	0.1282	010
oia	10	0.1282	001
ede	10	0.1282	000

Table 1: Dictionary $D = \langle W, C \rangle$.

Word (W')	Frequency	Probability	Huffman code (C')
f	5	0.0275	11111
b	6	0.0330	11110
l	9	0.0495	1110
a	14	0.0769	1101
h	15	0.0824	1100
o	16	0.0879	011
d	20	0.1100	010
i	29	0.1593	101
n	30	0.1648	100
e	38	0.2088	00

Table 2: Dictionary $D' = \langle W', C' \rangle$.

is a corresponding code c_i in C. The size of D is the number of strings in W; i.e., $\|D\| = |W| = |C| = m$. During coding, every occurrence of w_i in T is mapped into a code c_i for each $w_i \in W$, $c_i \in C$, $1 \leqslant i \leqslant m$.

A dictionary D must satisfy the *completeness* and *prefix* properties. D is *complete* with respect to text T if and only if T can be *parsed* entirely into strings of D. In formal terms, let $g : \{1, 2, \ldots, m\} \rightarrow W$ be a function that returns a string $w_i \in W$ given its ordinal position i in W. Then D is complete with respect to T if and only if there exist a parse $h_1 h_2 \cdots h_p$, $h_j \in \{1, 2, \ldots, m\}$, $p > 0$, with respect to dictionary D such that $T = g(h_1)g(h_2)\cdots g(h_p)$. If D is complete with respect to T, we say that D *encodes* T. D satisfies the prefix property if no string in W is a prefix of another string in W.

Example 1: Let $A = \{a, b, d, e, f, h, i, l, n, o\}$ be an alphabet. The following text is a string of A^+:

```
hinnfbondnfoiaedeoiaoianhoianhndileei
liloianhilileeoiaedendeeaedeedendaboa
edendedehinilhinhineehinhinnfboeeiloi
abonfedendoiandhinoiaedenhhinhineeboe
deilndeeeendaedenfoiailnheendbohin
```

The dictionary for the text is shown in Table 1. It can be verified that the dictionary satisfies the completeness and prefix properties. The text can be unambiguously parsed using W into words that can be replaced by codes in C. ∎

The text in the above example is specially constructed to also illustrate the complexity of dictionary construction. Unlike English text where the space character is often a delimiter of words, there may be no clear demarcations in the text. That is why choosing the right words (of the right lengths) for an arbitrary text in order to obtain an optimal dictionary is hard.

Given text T, there may exist more than one dictionaries capable of encoding T. Let S_T be the set of all dictionaries that encode T and $\hat{T}_D \in C^+$ be the result of coding T with $D \in S_T$. An *optimal* dictionary $D^{\mathrm{opt}} \in S_T$ for T yields the most compact overall coding; i.e., $|\hat{T}_{D^{\mathrm{opt}}}| = \min\{|\hat{T}_D| : D \in S_T\}$.

Example 2: An alternative dictionary D' for the text in Example 1 is shown in Table 2. Let us compare the *compression efficiencies* of D and D'. A common measure for efficiency is the *rate of coding*, which is the average number of bits produced by each code per input symbol. In Example 1, the total number of bits in the encoded text is equal to the size (in bits) of each code (column C) multiplied by its frequency; i.e., $4 \times 4 + 5 \times 4 + \cdots + 10 \times 3 = 254$. The total number of text symbols is computed to be 182 characters. This gives a coding rate of $254/182 = 1.40$ bits per symbol for dictionary D. Considering that each symbol is an ASCII character of one byte (8 bits), this is an improvement of 82.5%. The rate for D' is $538/182 = 2.96$ bits per symbol. Since it is higher than the rate of D, D is more desirable as the dictionary for encoding T. ∎

We formally define the Optimal Dictionary Construction problem (ODC) as follows:

Definition 1 (ODC): *Given $T \in A^+$, find $D^{opt} = \langle W, C \rangle$ such that $|\hat{T}_{D^{opt}}| = \min\{|\hat{T}_D| : D \in S_T\}$.*

The definition of ODC places no restriction on the size of the dictionary and the length of strings in the dictionary. In order to facilitate a genetic algorithmic implementation, we choose to fix the size of the dictionary as m and define an upper limit n on the length of strings in the dictionary. The restricted version of ODC is defined as follows:

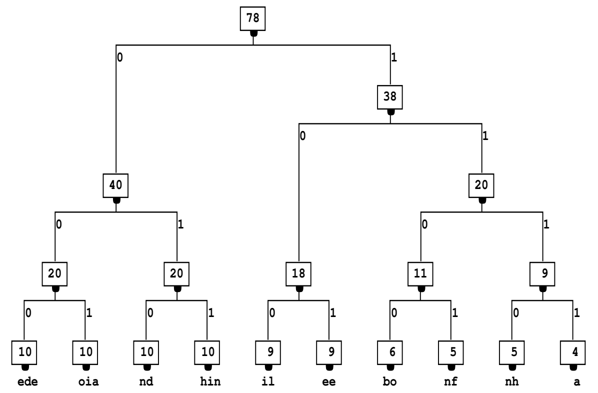

Figure 1: Huffman tree for the dictionary in Example 1.

Definition 2 (ODC(m,n)): *Given $T \in A^+$, $m > 0$ and $n > 0$, find $D^{opt} = \langle W, C\rangle$ such that $|\hat{T}_{D^{opt}}| = \min\{|\hat{T}_D| : D \in S_T\}$ and $1 \leqslant |w_i| \leqslant n$ for all $w_i \in W$, $1 \leqslant i \leqslant m$.*

Fixing the dictionary size m and the length of each string n in the dictionary restricts our ability to obtain the best dictionary for encoding a text, but is often performed in practice due to limitations in computational and memory capacity. Henceforth, we focus our attention on ODC(m,n) only.

3 CONVENTIONAL ALGORITHM

The conventional algorithm for computing the dictionary for a given text works as follows: The text is parsed at the character level and the frequency of occurrence for each distinct character in the text is computed. Suppose character a occurs n_a times in the text, then its frequency of occurrence is $p_a = n_a/N$, where N is the total number of characters in the text.

Although the text may also be parsed at the string level, it is usually not, unless justified by *a priori* information about the text. For instance, if it is known that the text consists mostly of words of length 5, then parsing may be performed at the string (of length 5) level. Unfortunately, there are no generally accepted word lengths for texts in English or in other languages.

With this set of frequencies for all characters, the text is coded in such a way that the size of a code assigned to a character is inversely proportional to its frequency of occurrence; i.e., more frequently occurring characters are assigned shorter codes. Examples of such coding techniques include Huffman coding (Huffman 1952) and *arithmetic coding* (Witten, Neal and Cleary 1987). The Huffman tree for dictionary D in Example 1 is shown in Figure 1. We shall refer to this type of conventional algorithm by the abbreviation CO.

4 GENETIC ALGORITHM DESIGN

There are many design choices in the construction of a genetic algorithm for a particular problem. They include the chromosome coding of candidate solutions to the problem; reproduction operators that generate offspring chromosomes; strategies that select parent chromosomes for reproduction; population size, etc. This section describes our implementation of a genetic algorithm that approximates a solution to ODC(m,n). We shall refer to this implementation by the abbreviation GA.

4.1 CHROMOSOME REPRESENTATION

Each chromosome in our genetic algorithm represents a dictionary. For example, $\langle w_1 w_2 \cdots w_m\rangle$ represents a chromosome where each gene $w_i \in A^+$ is a string whose length is bounded; i.e., $1 \leqslant |w_i| \leqslant n$. The alphabet A is the set of *printable* ASCII characters consisting of the 26 letters of the English alphabet in both lower and upper cases, the numerals and the special symbols. The population consists of a set of chromosomes whose genes are randomly initialized with characters from A; i.e., each gene has a length of one initially.

Our implementation incorporates the notion of gene *ranking*; an idiosyncratic reflection of the ODC(m,n) problem. As each gene is a string whose occurrence in the text is to be substituted, we define its rank as its frequency of occurrence in the text. Given a set W of strings comprising the dictionary, the set of codes for encoding the text is usually constructed such that the size of each code is inversely proportional to the occurrence frequency of the string it replaces. Therefore, higher ranking genes are more desirable. We refer to high and low ranking genes as *strong* and *weak* genes respectively.

4.2 FITNESS VALUE

The fitness of each chromosome is its code rate R with respect to a text. It is defined as the ratio b/s where

b is the number of bits in the text encoded via the chromosome and s is the number of symbols in the uncoded text (see Example 2). The code rate for any chromosome is always strictly positive. Strong chromosomes are those that yield lower code rates.

4.3 REPRODUCTION

Selection: We have three strategies for choosing parent chromosomes to undergo genetic operations: (1) parent chromosomes are randomly selected from the population; (2) parent chromosomes of relatively the same fitnesses are selected; i.e., strong chromosomes are matched with strong chromosomes and vice versa; and (3) parent chromosomes of widely differing fitnesses are selected; i.e., strong chromosomes are matched with weak chromosomes.

Exchange: The exchange operator exchanges genes between two parent chromosomes. Two strategies are available for selecting genes for exchange: (1) genes from each parent chromosomes are randomly selected, and (2) the weak gene of one parent chromosome is replaced by the strong gene of the other parent chromosome; this is *one-way* exchange.

Fusion: The fusion operator *fuses* two genes into a single gene. For example, strings ab and cd may be fused into $abcd$. This operator provides the means to lengthen the strings of the dictionary. Again two strategies are available for selecting genes for fusion: (1) genes from each parent chromosomes are randomly selected, and (2) the weak gene of one parent chromosome is fused with the weak gene of the other parent chromosome. Each fusion operation between two parent chromosomes results in four possible offspring chromosomes. For example, the following results when the d and n genes of two parent chromosomes are fused:

$$\begin{array}{ccccc} \langle a & b & c & \boxed{d} & e\rangle \\ & & + & & \\ \langle m & \boxed{n} & o & p & q\rangle \end{array} \longrightarrow \begin{array}{ccccc} \langle a & b & c & dn & e\rangle \\ \langle a & b & c & nd & e\rangle \\ \langle m & dn & o & p & q\rangle \\ \langle m & nd & o & p & q\rangle \end{array}$$

Diffusion: The diffusion operator is the reverse of fusion; it splits genes into smaller genes. This operator shortens strings of the dictionary. Two strategies are available for selecting genes for diffusion: (1) genes are randomly selected, and (2) the weak gene of is diffused into two genes. Depending on the word length n, each diffusion results in $n-1$ possible pairs of offspring chromosomes. For example, with a word length of 4, the diffusion of $wxyz$ in chromosome $\langle a\ b\ c\ wxyz\ e\rangle$ results in 3 pairs of offspring chromosomes:

$$\begin{array}{ccc} \langle a\ \ b\ \ c\ \ w\ \ e\rangle & \langle a\ \ b\ \ c\ \ wx\ \ e\rangle & \langle a\ \ b\ \ c\ \ wxy\ \ e\rangle \\ \langle a\ \ b\ \ c\ \ xyz\ \ e\rangle & \langle a\ \ b\ \ c\ \ yz\ \ e\rangle & \langle a\ \ b\ \ c\ \ z\ \ e\rangle \end{array}$$

Operator selection: When two parent chromosomes are selected to undergo reproduction, the choice of applying any of the three operators is made probabilistically. In our implementation, the exchange, fusion and diffusion operators are employed with probabilities p_1, p_2, and p_3 respectively where $p_1+p_2+p_3 = 1$.

Validity checking: Since a chromosome represents a dictionary, each new offspring chromosome must satisfy the prefix property. After the application of a reproduction operator, we accept only those offspring chromosomes that are valid. We waive the completeness property because the restrictions on the size of the dictionary and the word lengths imply that the dictionary may not necessary encodes a text completely. This issue is elaborated in Section 5.1.

5 PERFORMANCE COMPARISON

In this section, we perform simulations to compare the performance of GA dictionaries (dictionaries discovered by the genetic algorithm) and CO dictionaries (dictionaries computed by the conventional algorithm) when encoding texts. Our primary interest is in determining the abilities of GA and CO to derive a set of words to encode a text given a restriction on the size of the set.

5.1 MEASURES OF INTEREST

Several measurements are made in the simulations. The first one is the code rate. Given a dictionary $D = \langle W, C\rangle$, let U be the set of strings in the text that remain uncoded by D. (The existence of U will be explained later on.) Let $f : A^+ \rightarrow \mathbb{Z}$ be a function that returns the frequency of occurrence of a string in a text. Then, the code rate (or fitness) of D is $R = (A + B)/|T|$ where

$$A = 8\sum_{u\in U} f(u)|u|$$

is the size of the uncoded text in bits, assuming that each symbol is a byte (8 bits); and

$$B = \sum_{w \in W} f(w)g(w)$$

is the size of the coded text in bits. Function $g : A^+ \rightarrow \mathbb{Z}$, defined as

$$g(w) = \phi\left(\log_2 \frac{f(w)}{|T|}\right)$$

gives the number of bits in the binary representation of code $c \in C$ corresponding to $w \in W$ and ϕ is the rounding function.

U, the set of uncoded strings exists due to the restriction on dictionary size, which implies that the words in the dictionary may not always completely encode the text. Thus, the dictionary is incomplete. This possibility arises in both GA and CO. This restriction implies that the sizes of both GA and CO dictionaries at the same word length will be approximately equal. Thus, we omit the dictionary size in the computation of the coding rate.

The second measure is the degree of *incompleteness* of a dictionary. It is given by the ratio $C = A/B$ where A and B are as defined previously. A *complete* dictionary has $C = 0$. Dictionary with a higher C value is more incomplete; there are more strings in the text that remain uncoded.

The third measure is the number of iterations it takes for GA to converge or equivalently for the code rate to stabilize. The measure is taken of multiple GAs executed at different word lengths (see below).

The fourth measure is the number of iterations needed for the completeness value of chromosomes to stabilize. Again, the measure is taken of multiple GAs executed at different word lengths.

The final measure is the word length, which is the upper limit on the size of words in the dictionary. In all the simulations, the above measurements are taken of multiple GAs and COs executed at different word lengths.

5.2 SIMULATION PARAMETERS

We performed a total of 9 sets of 5 simulations with both GA and CO. Each set of simulation has the following configuration of parameters: (1) the alphabet size $|A| = 80$, (2) the dictionary size $m = 20$, (3) the text size $|T| = 1000$ characters (or bytes), (4) the population size is 30, (5) the parent selection strategy is random; (6) the exchange strategy is such that weak genes of one parent chromosome are replaced by strong genes of another parent chromosome; and the (7) fusion and (8) diffusion strategies both select weak genes for fusion and diffusion respectively. (9) The probabilities of invoking exchange, fusion and diffusion are $0.75, 0.15, 0.10$ respectively.

Each set of simulation is performed at different word lengths of $n = 1, 2, \ldots, 9$. For each word length, 5 simulations are repeated and the measurements averaged.

At word length n, the CO parses the text n characters at a time while the GA generates dictionaries of words whose lengths are at most n. Once a dictionary is obtained from either algorithms, the frequencies of words in the dictionary is computed with respect to the text. These frequencies are used to compute A and B as described in the previous section.

5.3 SIMULATION RESULTS

Figure 2 shows the results of the simulations. The following observations may be made:

1. From Graphs 1 and 2, the average code rate of CO dictionaries increases with longer word lengths while that of GA dictionaries decreases up to a saturation point. Beyond the point, the average code rate of GA dictionaries also increases with longer word lengths.

2. The average code rate of CO dictionaries is consistently higher than that of GA dictionaries. In fact, some of the code rates are higher than 8 bits per symbol.

3. From Graphs 3 and 4, the average completeness of CO dictionaries increases with longer word lengths while that of GA dictionaries decreases up to a saturation point beyond which, the average completeness also increases with longer word lengths. This behavior is similar to that of code rates.

4. From Graph 3, the lowest average incompleteness value of GA dictionaries is 0.001 at word length 4. In reality, some individual GA dictionaries attain full completeness, i.e., they have 0 incompleteness value. These individual completeness values which are used to compute the average are not shown in the graphs.

5. From Graph 5, the average number of iterations it takes for the genetic algorithm to converge increases with longer word lengths. On the other hand, the average number of iterations for the

completeness of GA dictionaries to stabilize decreases up to a saturation point beyond which it increases with longer word lengths (Graph 6).

6. Graphs 7 and 8 do not exhibit much differences among the two types of convergence schedules for GAs at word lengths of $1, 3, 6, 9$.

5.4 DISCUSSION

As the word length increases, CO dictionaries parse the text with larger segments. In general, when words in a dictionary are short, there is a higher probability of them occurring more frequently in the text than when the words are long. Thus, a larger amount of the text will be encoded by the dictionary, resulting in lower code rates. The opposite is true when the words are long. Hence, we observe the average code rates of CO dictionaries increasing with longer word lengths (Graph 2).

GA dictionaries behave differently. At longer word lengths, they have more flexibility to mix and match symbols to produce words of different lengths that best encode the text. Compared with their counterparts in CO at the same word length n, some of the words in the GA dictionaries will be shorter than n when the GA discover that having all words of length n does not yield good dictionaries. This flexibility allows the GA to yield dictionaries with lower average code rates (Graph 1).

This same flexibility also allows both the average incompleteness values (Graph 3) and the average number of iterations for completeness stability (Graph 6) to decrease with longer word lengths because the GA is able to discover the best mix of words (of different lengths) to *cover* more of the text, resulting in less amount of uncoded text. Thus, the dictionaries are more complete and they also stabilize more quickly.

The reduction in average code rate of GA dictionaries continues up to a saturation point due to the restriction in dictionary size (Graph 1). As the word length continue to increase, GA looses its earlier flexibility of obtaining a good mix of words. As a result, the average lengths of words in the dictionary increase. GA begins to resemble CO now because the increase in the average word lengths results in higher code rates. Hence, we observe average code rates increasing as the word length extends beyond the saturation point.

These effects are also observed in the average incompleteness values (Graph 3) and the average number of iterations for completeness stability (Graph 6). In particular, the increase in incompleteness is now observed by both the GA and the CO, which has increasing incompleteness all along (Graph 4).

The average number of iterations for convergence shows a steady increase with longer word lengths (Graph 5). This increase has to do with the increasingly amount of combinations to mix and match words that best encode the text. Because there are now more possible combinations, there is a higher probability of generating an invalid chromosome, resulting in more rejections. Hence, more iterations are needed.

One observation that was made above is that some of the dictionaries produced by GA are complete. This illustrates an additional bonus for GA because complete dictionaries are clearly more useful and *correct* than incomplete dictionaries.

6 CONCLUSIONS

We have developed a genetic algorithm for a lossless data compression technique for which no genetic algorithm has been previously described. The compression technique has two parts: dictionary construction and coding. Our genetic algorithm approximates a solution to the optimal dictionary construction problem which is known to be NP-complete.

On the genetic algorithm side, many design alternatives remain to be explored. This includes other possible reproduction operators; probabilistic selection of genes; probabilities with which operators are applied; other strategies for parent chromosome selection and removal; combination of alternatives not already investigated, etc. We would like to conduct a thorough investigation of these issues in our future work.

On the data compression side, the technique we have examined in this paper belongs to a class of lossless compression techniques called the *semi-adaptive* compression techniques (Bell, Clearly and Witten 1990). As part of our future work, we plan to extend the genetic algorithmic approach to *completely adaptive* compression techniques; these are techniques that build a dictionary incrementally at the same time as coding progresses in one single pass.

Acknowledgements

The authors gratefully acknowledge the Consortium for International Earth Science Information Networking for funding this work.

References

T. C. Bell, J. G. Cleary, I. H. Witten. *Text Compression*. Prentice Hall Inc., Englewood Cliffs,

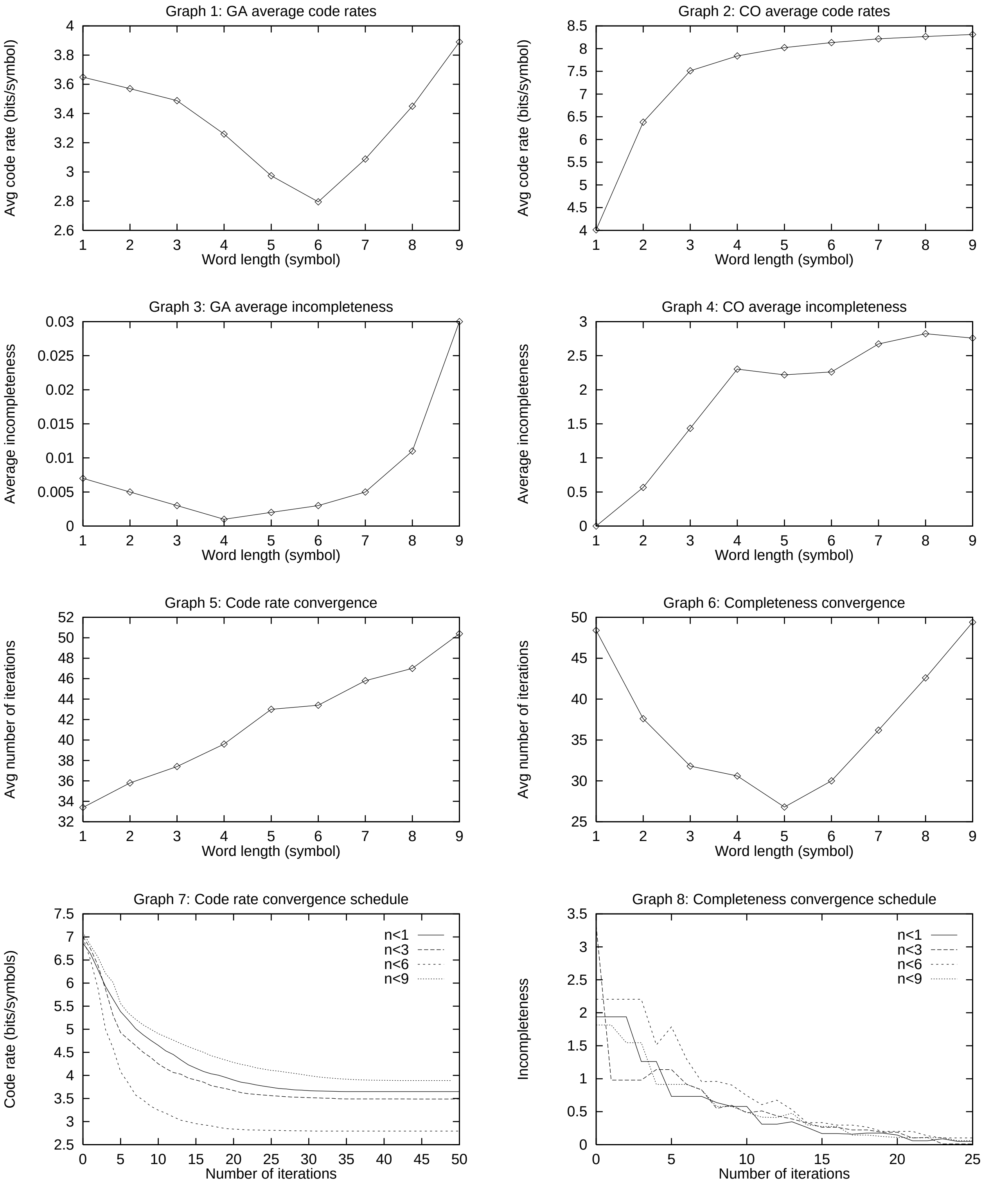

Figure 2: Genetic versus conventional algorithms.

New Jersey, 1990.

K. A. DeJong, W. M. Spears. Using Genetic Algorithms to Solve NP-Complete Problems. *Proceedings of the 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, California, 1989.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

R. M. Gray. *Source Coding Theory.* Kluwer Academic Publishers, Boston, Massachusetts, 1990.

J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1992.

D. A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the I.R.E.*, Vol. 40, No. 9, pp. 1098–1101, 1952.

J. A. Storer, T. G. Szymanski. Data Compression via Textual Substitution. *Journal of the ACM*, Vol. 29, No. 4, pp. 928–951, 1982.

R. Veldhuis. *An Introduction to Source Coding.* Prentice-Hall, New York, New York, 1993.

R. N. Williams. *Adaptive Data Compression.* Kluwer Academic Publishers, Boston, Massachusetts, 1991.

I. H. Witten, R. M. Neal, J. G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, Vol. 30, No. 6, pp. 520–540, June 1987.