Published in final edited form as:
J Mol Biol. 1991 October 20; 221(4): 1367–1378.

An Efficient Algorithm for Identifying Matches with Errors in Multiple Long Molecular Sequences

Ming-Ying Leung,
Division of Mathematics, Computer Science and Statistics, The University of Texas at San Antonio, San Antonio, TX 78249-0664, U.S.A

B. Edwin Blaisdell,
Department of Mathematics, Stanford University, Stanford, CA 94305, U.S.A

Chris Burge, and
Department of Mathematics, Stanford University, Stanford, CA 94305, U.S.A

Samuel Karlin
Department of Mathematics, Stanford University, Stanford, CA 94305, U.S.A

Abstract

An efficient algorithm is described for finding matches, repeats and other word relations, allowing for errors, in large data sets of long molecular sequences. The algorithm entails hashing on fixed-size words in conjunction with the use of a linked list connecting all occurrences of the same word. The average memory and run time requirement both increase almost linearly with the total sequence length. Some results of the program's performance on a database of *Escherichia coli* DNA sequences are presented.

Keywords

algorithm; multiple sequence matching; *E. coli*; REP elements

1. Introduction

With the rapid accumulation of molecular sequence data, including the advent of the human genome initiative, there is an increasing need for efficient and versatile computer algorithms for finding and classifying similarity patterns and relationships in data sets of long sequences. Over the past decade there has been a variety of approaches for finding similar segments among multiple sequences, including those of Queen *et al.* (1982), Waterman (1984), Bacon & Anderson (1986), Sobel & Martinez (1986), Posfai *et al.* (1989), Karlin *et al.* (1988a, b), Pearson & Lipman (1988), Staden (1989), Stormo & Hartzell (1989), Hertz *et al.* (1990), Lawrence & Reilly (1990), Smith *et al.* (1990), and Schuler *et al.* (1991). In many cases the central choice in designing any appropriate algorithm is the particular measure of multiple alignment quality to be used. In this context a widely used sequence comparison protocol is based on alignments derived by dynamic programming as introduced by Needleman & Wunsch (1970; for recent reviews, see Waterman, 1989; Doolittle, 1990; Spouge, 1989; Karlin *et al.*, 1991; Schuler *et al.*, 1991).

Here, we describe in some detail a close to linear-time algorithm (conjoined hash and linked list method) for finding approximate multiple repeats within and between nucleic acid or protein sequences. The advantages of the method are speed and the ability to look for internal matching segments simultaneously with matching segments among multiple sequences. Multiple sequences are concatenated into a single composite sequence, converting the problem of matching to the search for repeats in a single sequence. The algorithm entails hashing on words of a fixed size in conjunction with the use of a linked list connecting all occurrences of the same word. We shall later illustrate the performance of the program for a set of 389 DNA sequences from *Escherichia coli* totalling about 1·431 million bases.

A match is an aggregate of extended identity blocks of length $\geq b$, separated by short error blocks of length $\leq \varepsilon$. A set of program parameters (including b and ε) are used for selecting matches to be reported in the output. These parameters may either be specified by the user, or be set, by default to stringent values that ensure that the matches located are unlikely to occur by chance. After a first round of locating significant matches, the matching segments are extended and refined by the procedures outlined in Refinement of the Program Output. Some caveats on the concepts and implementation of the program are also discussed there.

The method achieves great speed by requiring the segments of all reported alignments to share a “core block” of identical letters exceeding some minimum length. It is recommended that this “core” parameter be chosen with reference to the statistical properties of maximal length common words among random letter sequences (Karlin & Ost. 1988). Such blocks can be found essentially in linear time in the length of the sequences under study, and it requires little additional time to extend them allowing for error blocks.

Useful biological information is sometimes revealed by sequence comparison in several alphabets derived from nucleotide and amino acid classifications. A list of the commonly used DNA and amino acid alphabets is given in a review by Karlin *et al.* (1989). Issues of sensitivity in detecting weak similarities (matches) pose a problem in protein sequence alignments because of the requirement that long matches should contain a satisfactory core block. This difficulty can in part be alleviated by using multiple alphabets. The program includes a translation module so that DNA and protein sequences can be compared in any of these derived alphabets. The basic design of the algorithm is independent of the alphabet, we therefore describe only the algorithm for the standard $\{A, C, G, T\}$ DNA alphabet. The algorithm also applies to the location of internal repeats in individual sequences, and to finding other word relations (e.g. inverted complementary pairings in DNA).

The matching program and these variations of it have been coded in C and organized into a small software package that has been implemented on an IBM PC compatible (running DOS), a VAX main frame computer (running VMS), a Sun Sparc station 470 (running UNIX) and a Cray Y-MP8/864 super-computer (running UNICOS). A profile of the performance on the four machines with the set of *E. coli* DNA sequences is reported in

Performance Analysis. Among the matches the program finds are the repetitive extragenic palindromic elements of the *E. coli* genome and other groups consisting of significant identities and dyad symmetry pairings. These results with interpretations are presented in An Example of All Known *E. coli* Sequences.

Working copies of these programs can he obtained by writing to M.-Y. Leung. Final and further updated versions of them will be distributed through the University of Texas System Center for High Performance Computing at Austin, TX.

2. The Algorithm

Our algorithm treats matches between sequences as repeats in the composite sequence formed by concatenating the individual sequences, with the restriction that the repeating segments do not overlap the junction of two different sequences. In the following discussion, an identity block refers to a group of sequence segments (called identity segments) in exact agreement with one another. None of these crosses sequence boundaries. An identity block is designated by $(a_1, \ldots, a_m; l)$ where $a_1,.., a_m$ represent the starting positions of the identity segments in increasing order with respect to the composite sequence; and l the length of the block. A match is an aggregate of such identity blocks separated by short "error blocks" of mismatched or inserted/deleted letters.

There are four main steps in the matching algorithm. First, the composite sequence of N letters is replaced by a sequence of $N - k + 1$ integers (called keys) representing successive words of k consecutive letters (Karlin *et al.*, 1983: 1988b). Simultaneously, a *linked list* array, which connects each occurrence of a k-word (a string of k consecutive letters) to the next occurrence of the same word in the composite sequence, is constructed. Second, all reasonably long identity blocks, called core blocks, are located. Third, these core blocks are extended by flanking them with other identity blocks in both directions to form longer matches, allowing interruption by small error blocks. Fourth, maximally extended matches that meet the prescribed printing criteria (explained below) will be produced in the output file.

Results in the program output are then refined as follows. (1) Sequence sets containing statistically significantly long common words found in the first search are reduced to smaller segments centered on the long words and are re-examined using less stringent limitations on core block and extension block lengths. (2) Common matches that do not qualify in their own right do qualify if sufficiently well aligned with and if not too distant from qualifying matches. Such aligned matches are also printed out. (3) The original database is searched for words having not more than 20% to 30% mismatch errors with the long words found in (1) and (2). These refinement schemes are described in Refinement of the Program output.

The algorithm will be illustrated by a data set consisting of four DNA sequences, each of length 50 base-pairs (bp[†]), listed in Table 1. Before detailing the individual steps of the algorithm, we first discuss a number of program parameters. These parameters may be

[†]Abbreviations used: bp, base-pairs; REP, repetitive extragenic palindromic.

specified by the user or set to their default, values according to certain statistical criteria built into the program.

(a) Program parameters

(i) Multiplicity parameters—The number of identity segments and the number of different sequences in which the identity segments occur are called, respectively, the block multiplicity and sequence multiplicity of the block. In our sample data set, the five-word CCCCT occurs at positions 7 and 14 of sequences 1 and 3, positions 8 and 16 of sequence 2, but only at position 14 of sequence 4. The collection of these occurrence of CCCCT forms an identity block (7, 14, 58, 66, 107, 114, 164; 5) with block multiplicity 7 and sequence multiplicity 4. In certain cases a set of sequences divide naturally into different groups (e.g. sequences from different organisms such as *E. coli* and *Salmonella typhimurium*, or from different functional classes such as nuclear and membrane proteins). The number of groups containing an identity block is called its group multiplicity.

The user may specify minimum values for all three multiplicities that will be denoted by m_0, s_0 and g_0, respectively, where $m_0 \geq 2$, $s_0 \geq 1$, $g_0 \geq 1$. By selecting appropriate minimum values for these multiplicities, different objectives can be achieved. With $s_0 = 1$ the program ascertains all internal repeats within individual sequences as well as matches between different sequences. Setting $s_0 = 2$, however, will make the program report only matches involving at least two different sequences. Setting $g_0 = 2$ will find only those matches common to at least one sequence from two different groups.

(ii) Block length parameters—To ensure a significant amount of similarity, we set the following requirements on all matches.

1. All constituent identity blocks with block multiplicity m must be of length b_{m} or more. The b_{m} values are called *minimal block lengths*.

2. A match must contain at least one reasonably long identity block called a *core block*. A core block with block multiplicity m must be of length c_m or more. The c_m values are called *core block lengths* and generally c_m is approximately $2b_m$. A match may contain more than one core block. In such cases, the leftmost core block is called the *leading core block*.

3. An error block must not contain more than ε letters, where ε (generally $< b_m/2$) is a small positive integer (usually 3 for DNA but may be higher; in protein sequences we recommend $\varepsilon = 1$ or 2) called the *error block length*. Mismatches as well as insertion/deletion type errors are allowed if the error block length is ≤ 3. For practical reasons, only mismatches are considered for larger ε.

It is reasonable to accept smaller minimal and core block lengths for identity blocks with higher block multiplicities than for those with low block multiplicities. For practical use, we find it sufficient for the program to accept five decreasing (not necessarily strictly) minimal block lengths (likewise for core block lengths) corresponding respectively to block multiplicities m_0, $m_0 + 1$, $m_0 + 2$, $m_0 + 3$, and the fifth value to $m_0 + 4$ and above.

(iii) Printing criteria—The program outputs the location and content of each match whenever its aggregated matching length (not including the error blocks) reaches the printing level. Higher printing criteria should be required for a match with more error blocks. Furthermore, lower printing criteria may be used for matches involving a higher number of sequences because they are less likely to occur by chance. Thus, printing criteria are specified as a two-dimensional array (u_{is}), where i represents the number of error blocks and s the sequence multiplicity of the leading core block of the match. Experience indicates that rarely does a match contain more than 14 error blocks. The program accepts possibly different printing criteria u_{is} for i ranging from 0 to 14, and s from the lowest allowable sequence multiplicity s_0 to $s_0 + 4$. Matches with more than 14 error blocks are treated as if there were only 14; and those with sequence multiplicity $s > s_0 + 4$ are treated as if $s = s_0 + 4$.

(iv) Default parameter values—In an initial run of the program on a new data set, the user may not know *a priori* what parameter values should be used. If the user prefers not to make a choice for some or all of the parameters described above, the program can provide default parameter values. The choice of default block lengths and printing criteria, as described below, tend to be conservative so as to ensure that the program will run very rapidly and produce an initial output representing only the “statistically significant,” matches among the sequences. The user, after reviewing the output, can then adjust these parameter values to suit his or her purposes.

Default multiplicity parameters take their smallest possible values: $m_0 = 2$, $s_0 = g_0 = 1$. The minimal extension block lengths b_m are set to the same value b for all m, where b satisfies $b\alpha^b < N \le (b + 1)\alpha^{b+1}$, $\alpha = 4$ being the alphabet size for DNA and N the composite sequence length. By an idealized consideration of a random ball in urn model for DNA sequences (Karlin & Leung, 1991), most b-words occur at least once in a sequence of length N with this choice of b. Table 2 shows the recommended values of b corresponding to different ranges of N. The maximal error block length ε is set to 3, one codon length, if $b \le 7$. When $b \ge 8$. ε is set to $b/2$, truncated to the largest integer. However, mismatches are the only type of error allowed by the program when the error block length exceeds 3.

Selection of core block lengths and printing criteria is guided by the statistical properties of independently generated random letter sequences of the same composition as the data sequences (Karlin & Ost, 1988). Thus, the core block length c_m is set to be the smallest integer greater than or equal to the theoretical expectation of the length of the maximal m-fold repeat L_m in a random sequence with the same total length N and the same letter frequencies as the composite sequence. The default printing criterion u_{is} for a match containing i errors and involving any s data sequences is set to the 1 % significance level relative to the expected length of the maximal common word C_{is}, allowing for i single letter mismatches, in s out of t random letter sequences (assumed to be independently generated with the same letter composition as the data) each of length n. Here, t is the number of sequences in the data set and n the average length of the sequences. Since this algorithm aims at dealing with long sequences, these values are calculated on the basis of the asymptotic distributions of L_m and C_{is} as $N \to \infty$, and then, to be conservative, rounded up to the nearest integers. Formulas for these distributions were derived by Karlin & Ost (1988) and illustrated in Karlin *et al.* (1989).

(b) The linked list

The repetition of k-words (keys) in the composite sequence is described by a linked list. In a sequence of N letters, there is a k-word beginning at each position $i = 1, \ldots, N - k + 1$. The linked list of keys, called L, is an array of integers whose ith element $L(i)$ is the position (address) in the composite sequence of the next occurrence of the key at position i. If the k-word does not occur after position i, $L(i)$ is zero. Thus, given any k-word, if we know where it first occurs in the composite sequence, the linked list allows us to locate rapidly all its occurrences in sequence. Alternatively, given any position i, following the linked list L will yield all subsequent positions in the composite sequence that have the same key as that at i.

The program first reads in all the sequences to be compared, concatenates them and stores the composite sequence in an array S. This letter sequence S is replaced by an integer sequence K of the keys fog the k-words starting at each of the first $N - k + 1$ positions. In an a-letter alphabet, the key of each distinct k-word is uniquely represented by a number w between 0 and $a^k - 1$. Details of this numerical representation of the keys for k-words can be found in articles by Karlin *et al.* (1983, 1988b). To construct the linked list of keys, we use a "bucket array" B of size a^k, which holds at location w, $0 \leq w \leq a^k - 1$, the position of the most recent occurrence of the key w. For use in the next step of the algorithm, we need one more bucket array F that stores the positions of the first occurrence of each key. All three arrays L, B and F are initialized to zero. Scanning through the composite sequence of keys stored in K, the following operations are then performed at each position j except for those at which the k-word crosses the boundary between two sequences.

1. Look up in K the key w of the k-word at j.
2. Examine the content of $i = B(w)$. If $B(w) = 0$, set $F(w) = j$ because this is the first occurrence of w. Otherwise, set $L(i) = j$ because this is the first occurrence of w after position i.
3. Set $B(w) = j$ to reflect the most recent occurrence of w.

With this linked list, we may analyze the composite sequence by keys instead of by letters. This procedure speeds up the comparison process, but it also means that any repeats of length less than k will not be detected at this stage. These may, however, be detected by the refinement procedures described in Refinement of the Program Output. Considering the minimal block length requirements, the ideal choice of k will be the smallest length l of an identity block that the user cares to detect. However, this will not be practical when l is large (e.g. 15) because the bucket arrays F and R of size a^k will consume an excessive amount of computer memory. As a compromise, the program determines the value of k such that it is the largest integer $\leq l$ such that a^k does not exceed 1/64 of the total number of bytes of core memory of the computer.

(c) Finding core blocks

The algorithm requires that all matches must contain at least one reasonably long core block. A core block is an identity block that satisfies the multiplicity and core block length requirements, and has been extended to its maximal length, i.e. is not part of a longer identity block. Suppose, for instance, that we are matching the four sequences in Table 1 with minimal core block length 4. The identity block CCCTC occurring at position 8 of sequences 1, 3 and 4, and position 9 of sequence 2, is a maximally extended core block as the letters immediately preceding the identity segments (namely C, C, C, T) are not in complete agreement, and the same is true for the letters immediately following the segments (C, G, C, T). Embedded in this core block are two smaller identity blocks (CCTC and CCCT), both of which satisfy the tort block length requirements but will not be core blocks because they are merely part of the larger identity block CCCTC. The block CCTC is said to be left-embedded, since it extends further to the left to a larger identity block. Likewise, CCCT is said to be right-embedded.

For each key w, $w = 0, \ldots, \alpha \ldots 1$, we obtain from the first occurrence bucket array F and the linked list L a chain $A(w) = (a_1, a_2, \ldots, a_p)$ giving all the positions in the composite sequence at which the key w occurs, where $a_1 = F(w)$ and $a_i = L(a_{i-1})$ for $i = 2, \ldots, p$. In other words, there is an identity block of length k or more at these positions. If this identity block is left embedded, $A(w)$ is discarded (i.e. not further examined). Otherwise, the program will attempt to extend this identity block further to form core blocks consisting of all or some of the identity segments as described below.

When the k-word at position a_i agrees with that at a_j, we can tell if this agreement extends to length $k + \delta$ for $1 \leq \delta \leq k$ by examining whether position $a_i + \delta$ is linked to position $a_j + \delta$ in the linked list. This can be accomplished by successively examining the addresses $L(a_i + \delta)$, $L(L(a_i + \delta))$, … and stopping as soon as the address equals or exceeds $a_j + \delta$. There is an extension of the match to a length $k + \delta$ if and only if we stopped with an equality. Using this principle, the program pulls out all possible (necessarily disjoint) subchains $D = (d_1, \ldots, d_q)$ of $A(w)$ (D is a subchain of $A(w)$ if the set of addresses $(d_1, \ldots, d_q)$ is a subset of $(a_1, \ldots, a_p)$) that extend the identity at least to a basal length l where l is either the lowest core block length or $2k$, whichever is smaller (i.e. $l = \min(c_{m0+4}, 2k)$). This basal length l is chosen because any core block, regardless of multiplicity, must be of length no less than c_{m0+4} since $c_m \geq c_{m0+4}$ for $m = m_0, \ldots, m_0 + 3$ and $c_m = c_{m0+4}$ for all $m \geq m_0 + 4$. So c_{m0+4} is a basal requirement on all potential core blocks. However, since the extension process works only with $\delta \leq k$, it can be used only for extending the identity of $A(w)$ to at most length $2k$ and hence if $c_{m0+4} \geq 2k$, we have to take $2k$ as the basal length.

To each of the subchains $D = (d_1, \ldots, d_q)$ of $A(w)$ obtained from the above extension, the following recursive procedure is applied.

1. Discard D if it does not satisfy the multiplicity requirements. Also discard D if it is left-embedded, because any identity block formed with D will be part of a larger block that will be found when considering a different chain $A(w')$ for another key w' or a different subchain D' of the same $A(w)$.

2. Extend the identity segments to the right as far as is possible until they are no longer all in perfect agreement to obtain the maximal length $\hat{l}$ of this identity. This is done by comparing keys at the relevant positions of the sequence-array K.
3. If $\hat{l} \geq c_q$, record $(d_1, \ldots, d_q; \hat{l})$ as a core block. Otherwise discard D. Note that c_q may be greater than the basal length l if either $m < m_0 + 4$ or $c_{m_0+4} > 2k$.
4. Select all subchains of D that extend the identity to a length greater than $\hat{l}$ and then repeat steps (1) to (4) on each of these subchains.

These steps are iterated until all subchains of $A(w)$ are exhausted. Then proceed to a new key, until all keys are exhausted.

As each new core block is found, it is joined to a core list that is maintained in increasing order of the position of the first identity segment of the block. If there is a long run of a single letter or a long stretch of certain periodic patterns, the recursive procedure above will locate a number of core blocks with different multiplicities, but all referring to the same region. To reduce such redundancy, the program checks for covering as the core blocks are being joined to the core list. For example, if we examine AAAAATAGCC and TAAAAACGTT, taking the core block length to be 4, we will find a core block of AAAA starting at positions 1 and 2 of the first sequence and positions 2 and 3 of the second; and another core block of AAAAA at position 1 of the first sequence and position 2 of the second. Although both are qualified core blocks, only the latter will be recorded in the core list because all identity segments in the first block are covered by some identity segment of the second block.

(d) Extension with errors

Once all core blocks have been located, extension allowing for errors is attempted on each of them. First the program extends a core block to the left, then to the right. Since the algorithm is essentially the same in both directions, only left-extension will be described.

For a given core block (the set of all identity segments) the set of all potential extending k-words is examined to pick out "matching words" or words that could extend at least m_0 occurrences of the core block. Rather than extending all such matching words, at this point we apply certain criteria to reduce the set of matching words that must be considered further. The strategy is essentially to consider first those matching words that extend the greatest number of identity segments of a core block and to consider other matching words only if they extend identity segments that have not been previously extended. A more detailed description of this process follows.

(i) Extension to the left—Denote the core block by $(a_1, \ldots, a_q; l)$ where a_i is the address of the ith identity segment of the core block in the composite sequence and l is the core block length. The potential extending k-words are then located at positions $a_i - k$ (zero error), $a_i - k - 1$ (1 error), …, $a_i - k - \varepsilon$ (ε errors) for each a_i. We now extract each of these words (actually the key corresponding to each word) from the key array K and with each word we associate two numbers: i, the core block index (a number in 1, …, q) and j, the "number of errors" (a number in 0, …, ε). To visualize this, we can put the words into a matrix array with q columns and $\varepsilon + 1$ rows. An example of such an "extension array" (for the 4 sequences of Table 1 and the core block GGGAGAGG, with parameters $k = 3$ and $\varepsilon = 3$) is shown in Table 3.

Now the program sorts the set of triplets (key,i,j) according to key number, so that all occurrences of a given word are grouped together. It is now easy to go through this list and, for each word, count how many distinct columns it occurred in. Words occurring in fewer than m_0 columns are ignored from here on. Words occurring in m_0 or more columns (called matching words) are shown for this example (with $m_0 = 2$) in Table 4.

If there are no matching words, then extension to the left is complete. Otherwise, we choose from among the matching words the word that occurs in the greatest number of different columns (in the example, CCT). There are several possibilities at this point.

First, if there is not a tie for most) distinct columns and if the maximally occurring word occurs at most once per column, then the word is maximally extended to the left in all the columns in which it occurs. This is done by examining the keys of the k-words immediately preceding each error block successively until a disagreement is reached. At this point, there are three possibilities.

1. The extension block does not meet the minimal block length requirement and is removed from further consideration.
2. The extension block is long enough to be a core block itself and is discarded. This condition ensures that a match containing more than one core block will be found only through right-extension of its leading core block and thus will not be reported twice.
3. The extension block meets the minimal block length requirement but is not a core block. In this case, the process of extension with errors is repeated recursively until no more extension is possible.

For the example case, we extend CCT to the left two more letters (CC), for a total extension block length of five. Then, if our parameters are set such that a length of five falls into category (3), above, we extend allowing for errors again.

A second possibility is that two or more matching words may be tied for occurrence in the greatest number of distinct columns. In this case, the program maximally extends each of the competing matching words and chooses for further consideration only the one that extends furthest. Ties at this stage are broken by picking the matching word that entails the fewest errors, i.e. the word for which the sum of the row numbers for all the array positions is lowest. If still tied at this point, further extension with errors is carried out for both words.

A third possibility is that the highest matching word or words may occur more than once in some columns. (This situation generally applies only to repetitive words, e.g. CCC.) In this case, there are multiple potential "extension paths" that may be taken through the extension array. This sort of tie is broken in the same way as for ties between different matching words, i.e. the extension path should be chosen that extends furthest and, if tied for length, entails the fewest errors.

At this stage, left-extension with errors has been completed for the matching word present in the most columns. A separate array of q 1s and 0s keeps track of which columns have been covered (i.e. contain a matching word which has been extended). If either all columns have been covered, or all matching words have been exhausted at this point, then we are finished with this core block. Otherwise, the matching words are again searched, this time for the word that covers (i.e. is present in) the greatest number of “new” (not previously covered) columns. Ties arc broken as before. This word is then maximally, and recursively, extended with errors as just described. This process is repeated until either all columns have been covered, or all matching words have been exhausted. When this process has been completed, all of the original set of core blocks which extend sufficiently to meet the printing criteria are printed in a group, in a format indicating which subsets of these blocks were extended by the above procedure and the locations of all extension blocks. In the example above, after the CCT matching word has been recursively extended to its maximum extent, all the columns would have been covered, so the matching word CTC would be ignored.

(ii) Extension to the right—When extension to the left has been completed, the core block is extended to the right. Right-extension is carried out in a manner analogous to left-extension, but operating in the opposite direction. An important difference is that, while we discard any matching blocks that reach the core block length requirement in extension to the left, in extension to the right they are kept. Again, the purpose is to ensure that multiple close core blocks are found only through right-extension of the leading core block so that they will be counted only once.

After the core block has been fully extended both to the left and to the right, we put together all the left- and right-extensions to obtain one or more matches that are aggregates of matching blocks separated by no more than ε letters. The total matching length, equal to the sum of the identity block lengths, is then compared with the printing criteria. If it qualifies for printing, it is added to a list of matches. After all matches associated with the core block have been found, the location of the blocks, their lengths and the matching segments will all be printed to the output file. Then we proceed to process the next core block.

(e) Variations of the matching algorithm

The matching algorithm can be easily adapted to handle internal repeats and searching for other word relationships such as dyad pairings in DNA sequences, as described below.

(i) Internal repeats—The algorithm locates matches by identifying repeats in the composite sequence. In fact, both internal repeats within sequences and matches between sequences can be found simultaneously when the multiplicity parameters are appropriately set. In the cases where only internal repeats in each individual sequences are sought, the sequences can be processed independently of one another. We therefore add an outermost loop to execute the program, loading only one sequence at a time to the core memory, until all the sequences have been analyzed. This way, the memory capacity of the computer poses a limit only on the length of the individual sequence instead of the total sequence length of the entire data set, making the repeat program very easy to use even on small-memory machines. The repeat program is further stream-lined by removing all checks for sequence and group multiplicities, sequence junctions, etc., which now become superfluous.

(ii) Word relationships—The matching program applies equally well to locate word relationships other than direct matches. For example, to locate dyad symmetry among sequences, one can make a copy of inverted complements of all the sequences and join them to the data set. The original sequences are put in one group and their inverted complements in another. To ensure that only the dyad pairings are reported but not the direct matches, the multiplicity parameters are set to $m_0 = s_0 = g_0 = 2$. The main disadvantage of this simple adaptation is that two copies of the sequences need to be stored. We are developing a new program that uses only one copy of the sequences. Instead of hashing through every distinct k-word, a word and its dyad word are processed concurrently to produce core dyad blocks and then the dyad symmetry blocks are extended with errors.

3. Refinement of the Program Output

A first application of this program aims primarily at dealing expeditiously with large databases of long sequences by very efficiently locating only the most significant matches among the sequences. The block length requirements are set with the intention that each match located will consist of matching segments composed of stretches of long exact identities interrupted by a small number of errors. At times, however, it may be desirable to perform a finer resolution analysis with the sequence data to locate other matches consisting of identity blocks not all of which satisfy the block length requirement. This, in principle, can be achieved by lowering the block length requirements. However, with a large data set such as the 1·431 million bp *E. coli* DNA database, the program will produce voluminous output reporting matches most of which occur merely by chance. Below are described three methods that can be used to refine the results of the program.

(a) Low block length refinement

Sequences involved in a significant match are picked out from the data set. Regions of length about 1000 bp centering around the matching segments are extracted from the selected sequences forming a much smaller data set. The program is rerun on the reduced data set with lower minimal and core block length parameters. The significant match itself, of course, will be rediscovered in the rerun. In addition, any further extension of the match with smaller identity blocks that did not meet the previous block length requirements will now be revealed.

(b) Aligned match refinement

Common matches that do not qualify in length nor multiplicity in their own right do qualify if sufficiently well aligned with, and not too distant from, qualifying matches. This algorithm has been described by Karlin *et al.* (1988*a*).

(c) Percentage error refinement

After a match has been extended with high resolution, we search through the entire data set to see whether there are any other sequence segments that are sufficiently similar to the match but have not been found because the block length or multiplicity requirements are not met. To locate these refinement matches, we take the matching segments from a significant match and search through the entire database to locate all segments that match any of the matching segments with no more than 30% of mismatched letters.

Table 1 shows the common 24-word GGGAGAGGGTTAGGGTGAGGGGAA at position 22 in sequences 1, 2 and 3. The alignment extension refinement finds the error-free word GTCCCCTCGCCCC in sequences 1, 2 and 3 of Table 1 at positions 5, 6 and 5, respectively, with corresponding error block gaps of 4, 3 and 4 greater than the program criterion of 3. The percentage error refinement with the 24-word cited above found ecoglnhpq with three mismatches and with the 13-word cited above again found glnhpq with two mismatches.

4. Performance Analysis

The performance of the program was tested with a database of *E. coli* DNA sequences totalling 1,431,059 bp (Rudd *et al.*, 1991). The database contains 389 sequences altogether. We measure the program run time starting with the first two sequences and repeating the measurement with an increasing number of sequences. The program parameters are chosen as follows. All multiplicities, the error block length and the minimal block lengths are set to their default values as given in The Algorithm, section (a). Core block lengths and printing criteria are set respectively to two times and three times the minimal block lengths.

The program was run on four different computers: an IBM XT-compatible microcomputer with 640 kilobyte RAM, a VAX 8650 main-frame computer with four megabyte memory allocation, a Sun Spare station 470 with 32 megabyte RAM, and a Cray Y-MP8/864 supercomputer with 48 megabyte memory allocation. The integer data type on the microcomputer has a two byte representation, thus limiting the allowable total sequence length to about 32,000 bp. The run times are shown in Table 5. With the VAX main-frame computer, the four megabyte memory allocation allowed for a total sequence length close to 500,000 bp. Longer sequences can be handled with virtual memory but the program will work much more slowly. We stopped the timing experiment at the sequence length at which a substantial drop of efficiency was observed. With the Sun Sparc station and the Cray the program completes matching the entire database within minutes. The run times for the VAX, the Sun Sparc and the Cray computers are displayed graphically in Figure 1. Note that the program performs only about three times faster on the supercomputer as compared to the Sun Sparc station. This is because the timing was done strictly on the scalar version of the program, without using any vectorization or parallelization features on the Cray, just for the purpose of indicating the efficiency of the algorithm as it now stands. A vectorized version of the program that makes full use of the vectorized computing power of the Cray Y-MP8/864 will be implemented.

To give an idea of the growth of run time with total sequence length, we employed the SYSTAT Version 5·0 software package to fit regression lines for ln(run time) against ln(sequence length) yielding the relations:

$$T_c = 19 \cdot 7 \times L^{1 \cdot 062},$$
$$T_s = 55 \cdot 7 \times L^{1 \cdot 058},$$

where T_c and T_s, in microseconds, denote the run times on the Cray and the Sun Sparc computers, respectively, and L the sequence length. Both relations indicate that the run times increase almost linearly with the total sequence length.

5. An Example of All Known *E. coli* Sequences

We have applied our general program to the *E. coli* DNA contig collection developed by Rudd *et al.* (1990, 1991). This database consists of all the *E. coli* sequences) in GenBank (plus a few unpublished sequences) from which all duplicates have been removed and all established overlapping sequences have been appropriately joined. There are more than 350 contigs, totalling about $1{\cdot}431 \times 10^6$ bp, distributed almost uniformly around the *E. coli* genome. Some contigs span more than 25×10^3 bp and include more than ten genes. For convenience we name the individual contigs Rudd files. Segments that read clockwise around the circular genome are located on a scale of 0 to 100. Segments that read counterclockwise are entered in the data as their inverted complements (dyads) but are located on the same scale as the clockwise segments.

Our program was applied to this entire database with initial search parameters: core block 16 bp, minimum extension block 8 bp, maximum single error block length 3 bp, minimum printing length 20 bp for matches with no errors and 24 bp for one or more error blocks. Both direct and dyad matching segments formed 585 sets of two or more Rudd files containing long word matches (19,000 lines of output). Some matching groups of as many as 13 files contained the same or dyad long word. Guided by theoretical probabilities of long matching words that were set forth by Karlin & Ost (1987, 1988) and reviewed for practical use by Karlin *et al.* (1989), we chose the following approximate criterion lengths for selecting significant matches: length 25 bp for a pair of matches, 22 bp for three matches, and 19 bp for four or more matches. We ignored the presence of an error block that in fact would require a greater length to qualify as a statistically significant match. We tallied all sets containing a matching word meeting these criteria and noticed that almost all these words lay in noncoding sequences of the Rudd files and none substantially overlapped a coding–noncoding interface. A few lay in genes. We, therefore, selected a set, of 141 distinct sequences that contained any significantly long matching word. These consisted of complete flanking, or intergenic sequences, or a few complete coding gene sequences.

The general program was applied to the set of these 141 sequences (about 182×10^3 bp) using less stringent parameters: core block 12 bp, minimum extension block 6 bp, maximum single error block length 3 bp, minimum printing length 24 bp for matching pairs with no errors and length 18 bp for three or more matches. This process yielded 801 sets containing long word matches within the 141 sequences (55,000 lines of output). Some sets of as many as 24 sequences contained the same or dyad-significant word. This set is a subset of a 59 member set containing an identical 12 bp core block (including a total of 8 cases of repetitions of the core block spread over some of the segments). Specifically, Rudd file ecothr contains a non-coding sequence (co-ordinates 4877 to 5540) involving some long words common to 58 other non-coding sequences. In aggregate, there are 82 sequences (of the 141) that contain a significantly long word shared directly with eeothr or with one of the sequences that matches ecothr and so on. For example, metJecoM, which shares two significantly long words with ecothr, also shares long words with seven other Rudd files not in the set of 59 cited above. All of these sequences are non-coding. The long common words have considerable parts matching the consensus repetitive extragenic palindromic (REP) sequences first noted by Higgins *et al.* (1982) and reviewed recently by Gilson *et al.* (1991). The functions of these REP elements is currently unknown, although it has been observed that many of them strongly bind DNA gyrase (Yang & Ames 1988). The four 14 base consensus sequences of the REP words are here designated

$$\begin{aligned}
a &= \mathrm{GCC}^{\mathrm{C}}_{\mathrm{T}}\mathrm{GATGCG}^{\mathrm{G}}_{\mathrm{A}}\mathrm{CG}^{\mathrm{C}}_{\mathrm{T}},\\
b &= {}^{\mathrm{C}}_{\mathrm{A}}\mathrm{CG}^{\mathrm{C}}_{\mathrm{T}}\mathrm{CTTATC}^{\mathrm{C}}_{\mathrm{A}}\mathrm{GGC},\\
b' &= \mathrm{GCC}^{\mathrm{T}}_{\mathrm{G}}\mathrm{GATAAG}^{\mathrm{A}}_{\mathrm{G}}\mathrm{CG}^{\mathrm{T}}_{\mathrm{C}},\\
a' &= {}^{\mathrm{A}}_{\mathrm{G}}\mathrm{CG}^{\mathrm{T}}_{\mathrm{C}}\mathrm{CGCATC}^{\mathrm{A}}_{\mathrm{C}}\mathrm{GGC}.
\end{aligned}$$

Note that *b* is an approximate dyad of *a* (correct in 12 of 14 sites), *b*′ is the exact dyad of *b*, and *a*′ is the exact dyad of *a*. A REP unit consists of the sequence *a* followed by *b* at a distance generally of up to four bases. A succeeding REP unit has the inverted complementary form *b*′*a*′, where *b*′ follows the preceding REP unit sequence *b* at a distance of generally 5 to 50 bases, and *a*′ follows *b*′ at a distance generally of up to four bases. Thus, a commonly occurring REP element can be symbolized *a*(0 to 4)*b*(5 to 50)*b*′(0 to 4)*a*′ or more simply *abb*′*a*′. The form *b*′*a*′ab also occurs frequently. The same or dyad significant word common to 24 sequences (cited above) has the text $\underline{\mathrm{ATGCG}}$ CTXCG $\underline{\mathrm{CTTATCAGGCCT}}$, where words common to *a* and *b* are underlined. The principal 12 base core block is the second of the underlined segments.

The refinement program was used to search the entire Rudd database for segments differing by two or fewer bases from one of the 32 possible forms of the 14 base consensus sequences. Only *ab* or *b*′*a*′ double occurrences were counted. This search uncovered 13 additional REP units for a total of 95 in the Rudd database. Their locations and information about them are displayed in Table 6.

The general program applied to the set of 141 segments (completely non-coding or completely coding) found, in addition to the 82 segments containing long REP elements, four small independent closed groups of four to twenty segments. The groups were closed in the sense that each member of the group had a long word matching at least one other member of the group and no member of the group matched a long word in any of the 141 sequences not in the group. Table 7 shows an example of one of these groups containing six locations identified by the general program, together with two additional locations found by the percentage error refinement program.

Application of the aligned match refinement to the segments containing the long words of Table 7 found a 13 base word matching exactly across the six segments at hemBeco1516, ecoaptadk6634, entDecoM16577. entDecoM24033. bglBecoM6631, ecophnaq15183 at slants of 22 or 23 (16 or 17 for the inverted complements). Reapplication of the percentage error refinement program recovered segments for entDecoM16288 and glnhpq at slants 23 and 17, respectively. The aligned matching word. GTCCCCTCGCCCC, for four of these segments may be observed in Table 1 (GTTCCCTCTCCCC in sequence 4, where mismatches are underlined).

In the refinement program and in the absence of a recognized target text, what criterion can be used for limiting the degree of relaxation of the subset selection criterion? In the example in Table 7. where the target text is of length 19. a criterion admitting 0, 1, 2, 3, 4, 5, 6 errors yields subset increments of 6, 1, 0, 1, 1, 22, 135 members, respectively. It is apparent that criteria of 5 or 6 errors admit abrupt large increases in numbers of members, largely due to background noise. The file for error criterion 3 is accepted because it introduces a segment that is a close dyad to a segment meeting criterion 0. The file for error criterion 4 is rejected because it introduces a coding sequence, while all others in the subset are non-coding sequences. REP words have been found in some of the files of Table 7 but at different locations: in ecoglnhpg, at 3015, bglBecoM at 6543, 8565, and ecophnaq at 3228, 6180, 11501. Significantly long common words in a second small closed group have been found in some of these files but again at different locations: in ecoglnhpq at 2308, 2640, in bglBecoM at 6337 and in ecophnaq at 4817.

6. General Discussion

While this program performs very efficiently when the block length parameters are appropriately set, these parameters also limit the sensitivity of the program. It is possible that some long matches composed of short approximately aligned identity blocks will not be detected. The purpose of the several refinement schemes in Refinement of the Program Output is to reduce this risk of missing some sequence features that may be of biological interest. However, we have found that most meaningful long matches do contain a satisfactorily long core block.

The matching program works equally well with protein sequences. Because protein sequences have a 20 letter alphabet, long stretches of matching amino acids are much less likely to occur. The block length parameters that we use for proteins are much lower than those for DNA. Typical minimal/core/error block lengths are 2/4/1 for small data sets (≤5000 amino acid residues) and 3/6/2 for larger ones. Use of the matching program on amino acids and other derived alphabets will be illustrated elsewhere.

Finally, as parallel computational equipment will be increasingly available to the scientific community in the near future, we would like to mention a very intuitive way of parallelizing our matching algorithm. Most of the computing time of this program is spent on finding core blocks and then extending them with errors. As the core blocks are found by hashing all distinct *k*-words and the long matches by extending each core block with errors, the average run time will be reduced if a number of *k*-words (likewise a number of core blocks) are processed in parallel, with one word (1 core block) on one processor. Such a parallelization scheme is well suited for implementation on a shared memory multiple instruction parallel computer system.

Acknowledgments

M.-Y.L. is supported in part by the Texas Higher Education Coordinating Board Advanced Research grant 010115-012, NSF grant DMS9005833, and the San Antonio Area Foundation. B.E.B., C.B. and S.K. are supported in part by NIH Grants HG00335-03 and GM10452-28 and NSF grant DMS86-06244.

References

Bachmann BJ. Linkage map of *Escherichia coli* K12. Edition 8. Microbiology Rev. 1990; 54:130–197.

Bacon DJ, Anderson WF. Multiple sequence alignment. J Mol Biol. 1986; 191:153–161. [PubMed: 3806669]

Doolittle RF. Molecular evolution: computer analysis of protein and nucleic acid sequences. Methods Enzymol. 1990; 183:99–110. [PubMed: 2314299]

Gilson E, Saurin W, Perrin D, Bachellier S, Hofnung M. Palindromic units are part of a new bacterial interspersed mosaic element (BIME). Nucl Acids Res. 1991; 19:1375–1383. [PubMed: 2027745]

Hertz GZ, Hartzell GW III, Stormo GD. Identification of consensus patterns in unaligned DNA sequences known to be functionally related. CABIOS. 1990; 6:81–92. [PubMed: 2193692]

Higgins CF, Ames GFL, Barnes WM, Clement JM, Hofnung M. A novel intercistronic regulatory element of procaryotic operons. Nature (London). 1982; 298:760–762. [PubMed: 7110312]

Karlin S, Leung M-Y. Some limit theorems on distributional patterns of balls in urns. Ann Appl Prob. 1991 in the press.

Karlin S, Ost F. Counts of long aligned word matches among random letter sequences. Advan Appl Prob. 1987; 19:293–351.

Karlin S, Ost F. Maximal length of common words among random letter sequences. Ann Prob. 1988; 16:535–563.

Karlin S, Ghandour G, Ost F, Tavare S, Korn LJ. New approaches for computer analysis of nucleic acid sequences. Proc Nat Acad Sci, USA. 1983; 80:5660–5664. [PubMed: 6577449]

Karlin S, Morris M, Ghandour G, Leung MY. Algorithms for identifying local molecular sequence features. CABIOB. 1988a; 4:41–51.

Karlin S, Morris M, Ghandour G, Leung M-Y. Efficient algorithms for molecular sequence analysis. Proc Nat Acad Sci, USA. 1988b; 85:841–845. [PubMed: 3124111]

Karlin, S.; Ost, F.; Blaisdell, BE. Patterns in DNA and amino acid sequences and their statistical significance. In: Waterman, MS., editor. Mathematical Methods for DNA Sequences. CRC Press; Boca Raton, FL: 1989. p. 133-157.

Karlin S, Bucher P, Brendel V, Altschul SA. Statistical methods and insights for protein and DNA sequences. Annu Rev Biophys Biophys Chem. 1991; 20:175–203. [PubMed: 1867715]

Lawrence CE, Reilly AA. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. Proteins. 1990; 7:41–51. [PubMed: 2184437]

Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 1970; 48:443–453. [PubMed: 5420325]

Pearson WR, Lipman DJ. Rapid algorithms for sequence matching. Proc Nat Acad Sci, USA. 1988; 85:2444–2448. [PubMed: 3162770]

Posfai J, Bhagwat AS, Posfai G, Roberts RJ. Predictive motifs derived from cytosine methyltransferases. Nucl Acids Res. 1989; 17:2421–2435. [PubMed: 2717398]

Queen CM, Wegman N, Korn LJ. Improvements to a program for DNA analysis: a procedure to find homologies among many sequences. Nucl Acids Res. 1982; 10:449–456. [PubMed: 6174938]

Rudd KE, Miller W, Ostell J, Benson DA. Alignment of *Escherichia coli* K12 DNA sequences to a genomic restriction map. Nucl Acids Res. 1990; 18:313–321. [PubMed: 2183179]

Rudd KE, Miller W, Werner C, Ostell J, Tolstoshev C, Satterfield SG. Mapping sequenced *E. coli* genes by computer: software, strategies and examples. 19. Nucl Acids Res. 1991; 19:637–646. [PubMed: 2011534]

Schuler GD, Altschul SF, Lipman DJ. A workbench for multiple alignment construction and analysis. Proteins. 1991; 9:180–190. [PubMed: 2006136]

Smith HO, Annau TM, Chandrasegaran S. Finding sequence motifs in groups of functionally related proteins. Proc Nat Acad Sci, USA. 1990; 87:826–830. [PubMed: 1689055]

Sobel E, Martinez H. A multiple sequence alignment program. Nucl Acids Res. 1986; 14:363–374. [PubMed: 3753772]

Spouge JL. Speeding up dynamic programming algorithms for finding optimal lattice paths. SIAM J Appl Math. 1989; 49:1552–1566.

Staden R. Methods for discovering novel motifs in nucleic acid sequences. CABIOS. 1989; 5:293–298. [PubMed: 2684350]

Stormo GD, Hartzell GW III. Identifying protein-binding sites from unaligned DNA fragments. Proc Nat Acad Sci, USA. 1989; 86:1183–1187. [PubMed: 2919167]

Waterman MS. General methods on sequence comparisons. Bull Math Biol. 1984; 46:473–496.

Waterman, MS. Editor of Mathematical Methods for DNA Sequences. CRC Press Inc; Boca Raton, FL: 1989.

Yang Y, Ames GFL. DNA gyrase binds to the family of prokaryotic repetitive extragenic palindromic sequences. Proc Nat Acad Sci, USA. 1988; 85:8850–8854. [PubMed: 2848243]

Yang, Y.; Ames, GF-L. The family of repetitive extragenic palindromic sequences: interaction with DNA gyrase and histonelike protein HU. In: Drlica, K.; Riley, M., editors. The Bacterial Chromosome. Amer. Soc. Microbiol; Washington DC: 1990. p. 211-225.

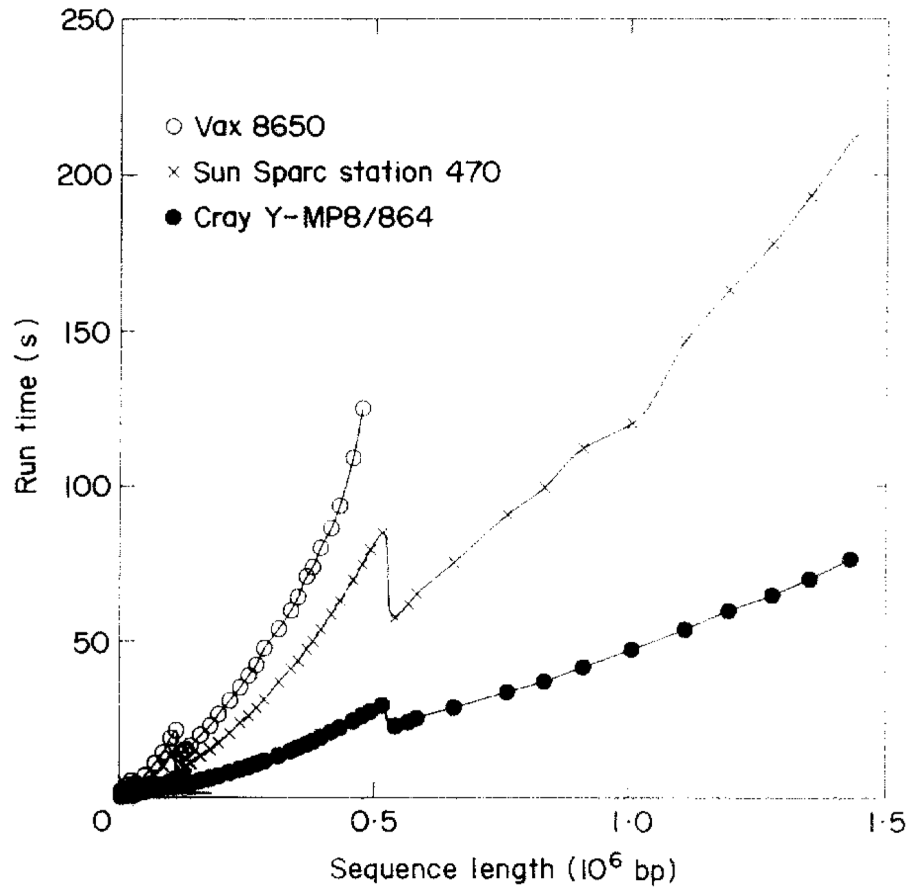

Figure 1.
Run times of the matching program *versus* sequence length on the VAX 8650, Sun Sparc station 470, and Cray Y-MP8/864 computers.

Table 1

Sample data set of four DNA sequences selected from Table 7

Sequence 1 (bglBecoM)
GTCGGTCCCC TCGCCCCTCT GGGGAGAGGG TTAGGGTGAG GGGAAAACCG
Sequence 2 (ecoaptadk)
GGACAGTCCC CTCGCCCCCT CGGGAGAGGG TTAGGGTGAG GGGAACAGGC
Sequence 3 (entDecoM)
ATCCGTCCCC TCGCCCCTTT GGGGAGAGGG TTAGGGTGAG GGGAACAGCC
Sequence 4 (ecoglnhpq)
GGCAGTTCCC TCTCCCCTAT GGGGAGAGGA TTAGGGTGAG GGGCGCAAAC

Table 2

Recommended minimal block lengths for different composite sequence lengths

Total sequence length *N*	**Minimal block length *b***
≤1000	3
1001–5000	4
5001–25,000	5
25,001–100,000	6
100,001–500,000	7
500,001–2,500,000	8
2,500,001–10,000,000	9
10,000,001–50,000,000	10

Table 3

Sample extension array

Errors	**Position**	a_1 **(S_1 : 22)**	a_2 **(S_2 : 22)**	a_3 **(S_3 : 22)**	a_4 **(S_4 : 22)**
0	$a_i - k$	CTG (w[1, 0])	CTC (w[2, 0])	TTG (w[3, 0])	ATG (w[4, 0])
1	$a_i - k - 1$	TCT (w[1, 1])	CCT (w[2, 1])	TTT (w[3, 1])	TAT (w[4, 1])
2	$a_i - k - 2$	CTC (w[1, 2])	CCC (w[2, 2])	CTT (w[3, 2])	CTA (w[4, 2])
3	$a_i - k - 3$	CCT (w[1, 3])	CCC (w[2, 3])	CCT (w[3, 3])	CCT (w[4, 3])

Table 4

Matching words

Word	Array positions (col.,row)	No. columns
CCT	(1,3); (2,1); (3,3); (4,3)	4
CTC	(1,2); (2,0)	2

Table 5

Run times on the IBM XT-compatible (12 MHz, 640 K RAM) microcomputer

No. of sequences	Total length (bp)	Run time (s)
2	3670	8
3	5571	3
4	9833	7
5	14,033	10
6	16,567	14
7	18,768	17
8	22,691	24
9	23,913	26
10	27,297	11
11	27,597	12
12	27,797	12
13	27,997	12
14	28,197	13
15	31,571	15

Table 6

REP words found in the *E. coli* data base of K. Rudd

min[†]	Rudd[‡] file	Location[§]	Rep pattern[‖]
NA	ecogacar	1504	*abb′a′a*
NA	econtrla	1089	*ab*
00.0	ecothr	5422	*ab*
00.5	ant-ecoM.I	18,256	*abb′a′*
00.9	folAecoM.D	1063	*ab*
01.4	polBecoM.D	4161	*abb′a′abb′a′abb′a′*
02.4	leuAecoM.I	27,954	*b′a′abb′a′ab*
03.0	ecglde	2498	*abb′a′*
03.7	mrcBecoM.I	8851	*abb′a′*
07.8	lacTeco	1242	*abb′a′*
07.9	cynTecoM.D	3205	*b′a′*
09.8	m23546	84	*b′a′*
11.8	ecopurek	1983	*abb′a′ab′a′*
13.2	entDecoM.D	2107	*abb′a′*
13.2	—	24,096	*b′a′*
15.4	asnBecoM.D	2328	*abb′a′*
15.4	.I	10,675	*aa′abb′a′*
15.4	ecotgop	958	*bab*
16.0	ecophrorf	1991	*ab*
16.3	gltAecoM.I	6317	*b′a′ab*
16.3	—	10,669	*b′a′abb′a′ab*
16.7	ecocyd	191	*ab*
16.8	ecoarog	1613	*b′a′*
17.6	bioAecoM.I	8253	*abb′a′*
18.3	ecoglnhpq	2948	*abb′a′*
22.5	appAeco	118	*aa′*
22.5	—	1576	*b′a′*
25.2	econdh	12	*b′a′ab*
25.2		1669	*ab*
27.0	ecogdhak	1547	*ab*
27.7	narLecoM.D	11,559	*ab*
37.4	ecosodb	782	*abb′a′*
41.0	ecoruvab	1941	*b′a′abb′a*
42.4	cheZecoM.D	23	*b′a′*
43.2	ecohag	1610	*ab*
43.2	srmBeco	1859	*bb′a′*
46.1	cdd-eco	984	*ab*
50.2	ecorcsbc	3543	*bb′a′b′bb*
50.3	ecgyraam	240	*abb′a′*

min[†]	Rudd[‡] file	Location[§]	Rep pattern[‖]
50.5	econrda	5836	*b′a′abb′a′ab*
52.4	argTecoM.D	2224	*abb′a′*
54.3	alaWecoM.D	4141	*ab*
54.6	cysPeco	4450	*b′a*
54.6	—	5524	*b′a*
54.8	ecoglya	1651	*abb′a′*
55.7	purCecoM.D	847	*ab*
58.5	rrnGecoM.D	276	*b′a′*
60.4	ecoprou	4008	*b′a′ab*
63.1	ecfucose	190	*aa′*
66.6	ecofdapgk	5546	*abb′a′*
66.3	speBecoM.D	1142	*b′a′abb′a′*
67.8	exbDecoM.D	194	*abb′a′ab*
68.1	rpsUecoM.I	8763	*b′a′*
68.9	ecocca	1703	*abb′a′ab*
70.0	rnpBecoM.I	1093	*abb′a′*
71.3	deaDecoM.D	7220	*b′a′ab*
76.4	malQecoM.D	196	*bb′a′*
76.4	.I	8089	*b′a′*
76.7	glpRecoM.?	4301	*abb′a′*
77.5	ugpQecoM.D	3631	*ab*
77.5	.D	5262	*abb′a′*
77.5	.D	11,090	*abb′a′*
79.8	ecofpp	1792	*abb′a′*
80.4	ecoavt	100	*abb′a′*
81.0	ecocysxe	1057	*ab*
81.2	mtlAecoM.I	2333	*abb′a′*
83.4	gyrBecoM.D	1219	*abb′a′*
84.2	bglBecoM.D	6543	*b′a′*
84.2	.D	8565	*ab*
85.1	ilvGecoM.I	7057	*ab*
85.9	uvrDecoM.I	2597	*abb′a′*
86.8	fadAecoM.D	28	*abb′a′abb′a′abb′a′*
87.6	ecogln	1509	*ab*
88.4	ecocpxa	1677	*b′a′ab*
88.5	pfkAecoM.I	1112	*abb′a′*
88.9	metJecoM.D	1	*abb′a′*
88.9	.I	4400	*b′a′*
90.8	purDecoM.I	14,141	*b′a′abb′a′*
91.2	IysCecoM.I	3756	*b′a′ab*
91.4	xylEecoM.D	107	*ab*
91.4	.I	5105	*b′babb′a′*

min[†]	**Rudd**[‡] **file**	**Location**[§]	**Rep pattern**[‖]
91.4	.I	9331	*b′a′abb′a′*
91.6	plsBecoM.D	778	*abb′a′*
91.9	ecotyrba	150	*ab*
92.9	ecophnaq	3228	*b′a′abb′a′abb′a′abb′a′abb′a′ab* (*a at end of phnA-30*)
92.9		6180	*b′a′ab*
92.9		11,501	*abb′a′*
93.4	melAecoM.I	4527	*abb′*
96.5	valSecoM.D	3648	*abb′a′*
98.6	merCecoM	11,090	*ab*
99.5	deoCecoM.I	1920	*abb′a′*
99.6	ecotrpr	719	*b′a′abb′a′*

[†]Location of the REP in the genome scaled 0 to 100 clockwise from the origin of replication.

[‡]Rudd *et al.* (1990, 1991). I indicates the Rudd file is in the same polarity as the coding polarity. D indicates the sequence has been complemented and inverted.

[§]Location of the 1st base of the 1st word of the REP element relative to the 1st base of the Rudd file.

[‖]REP words are designated as follows:

$$\begin{gathered}a{=}\mathrm{GCC}^{\mathrm{G}}_{\mathrm{T}}\mathrm{GATGCG}^{\mathrm{G}}_{\mathrm{A}}\mathrm{CG}^{\mathrm{C}}_{\mathrm{T}},\\ b{=}{}^{\mathrm{G}}_{\mathrm{A}}\mathrm{CG}^{\mathrm{C}}_{\mathrm{T}}\mathrm{CTTAT}^{\mathrm{C}}_{\mathrm{A}}\mathrm{GGC},\\ b'{=}\mathrm{GCC}^{\mathrm{T}}_{\mathrm{G}}\mathrm{GATAAG}^{\mathrm{A}}_{\mathrm{G}}\mathrm{CG}^{\mathrm{t}}_{\mathrm{C}},\ \text{and}\\ a'{=}{}^{\mathrm{A}}_{\mathrm{G}}\mathrm{CG}^{\mathrm{T}}_{\mathrm{C}}\mathrm{CGCATC}^{\mathrm{A}}_{\mathrm{C}}\mathrm{GGC}.\end{gathered}$$

(Yang & Ames, 1990).
A concatenation *ab* or *b′a′* is called a repetitive extragenic, palindromic sequence (REP). A cluster of REPs has been called a REP element. Note that *b* is the approximate dyad of a (correct in 12 out of 14 sites) and that *b′* is the exact dyad of *b*. and *a′* is the exact dyad of *a*.

Table 7

A significantly long common word found in a closed subset of the files in the *E. coli* database of K. Rudd: target word CTCAC CCTAA CCCTC TCCC

Min[†]	Rudd[‡] name	Location[§]	Adjacent genes[||]	Distances to genes[¶]
8·7	hemBeco	1516.I	*hemB*,*	56, 339
10·7	ecoaptadk	6634.D	*adk*,*	34, 170
13·5	entDecoM.?	16577.D	*fepB*, *entC*	83, 273
13·5	.?	16628.I	*fepB*, *entC*	134, 222
13·6	.I	24033.I	ORF, ORF	59, 107
18·3	ecoglnhpq	1466.D	*glnH*, *glnP*	47, 74
84·2	bglBecoM.D	6631.D	*pstA*, *pstB*	143, 21
92·9	ecophnaq	15183.I	*phnP*, *phnQ*	27, −66

ORF. open reading frame.

[†]Location of the word in the genome scaled 0 to 100 clockwise from the origin of replication (Bachmann, 1990).

[‡]Rudd *et al.* (1990, 1991). I designates that the interval containing the long word is in the same polarity as the coding polarity of the neighboring genes; D designates that the interval and the neighboring genes have been inverted and complemented (dyad); ? designates that 1 of the neighboring genes is in the dyad polarity and 1 is not.

[§]Location of the 1st base of the word relative to the 1st base of the Rudd file. I designates that the word is the same polarity as the displayed text; D designates the dyad form.

[||]*Designates that the word occurs in a terminal non-coding segment of the Rudd file.

[¶]The 1st number is the number of bases between the end of the preceding gene and the 1st base of the common word. The 2nd number is the number of bases between the last base of the common word and the 1st base of the succeeding gene.