Abstract: A method is presented for solving the unique common superstring problem, a variant of the shortest common superstring. The method has applications to the sequencing of biological deoxyribonucleic acid (DNA) strings from experimental data. The algorithm takes as input sufficiently long, overlapping substring fragments and assembles them into the unique common superstring. On insufficiently long data, the algorithm terminates early to signal the need for additional data. The algorithm has running time linear in its input size, which is a significant improvement over other approaches to sequence assembly.

A Linear Time Algorithm for DNA Sequencing

David E. Foulser

Research Report YALEU/DCS/RR-812
August 1990

This research supported by the Office of Naval Research under grant N00014-86-J-1906 and by the National Library of Medicine under NIH Grant T15 LM07056.

1 Introduction

An open computational problem in molecular biology has been to efficiently determine the contents of a linear string of DNA (based on the natural four-letter alphabet) from experimentally determined fragments of the desired string. This is equivalent to finding the unique common superstring of multiple string fragments. The overlapping string fragments are *sequenced*, perhaps using the Maxim-Gilbert approach, and then assembled to form the complete string. This problem can be viewed as a special case of the NP-complete shortest common superstring problem, for which other authors have presented complete or heuristic solutions [5, 6, 7, 8, 9]. Recourse to the full shortest common superstring problem is unnecessary, however, and may not be biologically correct.

By imposing an extra condition on the input data, we are able to develop a linear-time algorithm that either (1), performs the unique common superstring assembly or (2), halts prematurely to indicate that insufficient data exist for the method to identify a unique common superstring. We show that premature halting of the method identifies all instances in which insufficient data are input, thus proving that the extra condition is necessary and sufficient to deduce the unique common superstring (which may not be a shortest common superstring) from the input data fragments. The method can thus be used both to assemble known data and to guide molecular biology experimentation to derive additional data where needed.

2 Definitions

Let a sequence S be composed of letters from a finite alphabet. Inputs to the method are created by breaking S into fragments (typically by the use of *restriction enzymes*) whose sequence composition is determined by some method. We assume that fragments are created by cutting the original sequence at all occurrences of given letter patterns (restriction enzymes operate in this fashion, with each enzyme cutting every occurrence of a fixed pattern). For simplicity of presentation, we shall assume that the fragments are determined exactly, although extensions to inexactly determined inputs are not difficult. Multiple sets of fragments are determined by treating separate batches of S with different sets of restriction enzymes; the fragments from different sets overlap to some degree. We also assume that one batch of S is treated with all enzymes so as to provide a decomposition of S into some basic sequence fragments (this assumption can be removed by using the four individual nucleotide bases A, C, G, and T as the basic fragments).

Let $\mathcal{A} = \{a_1, \ldots, a_m\}$ be the alphabet of basic sequence fragments of the desired sequence S. $\mathcal{A}$ can be determined by applying all restriction enzymes to the sequence S at once. Each a_i is

typically a short substring of nucleotides. Let $S = s_1 \cdots s_n$ denote the output (or desired) sequence in terms of the basic building blocks, so that each $s_i \in \mathcal{A}$.

The input to the problem consists of p copies of S, each of which has been cut into fragments by some subset of restriction enzymes. Let $I_{i*} = I_{i1}, \ldots, I_{in_i}$ denote the fragments of copy i of S, for $1 \leq i \leq p$. The desired output is the assembled value of the entire sequence S.

An essential property of this algorithm is that it restricts the set of inputs. The *repeat rule* states that, for any repeat $R = r_1 \cdots r_k$ (with each $r_i \in \mathcal{A}$) that appears more than once in S, and for each occurrence of R in S, at least one input copy of S has a fragment I_{ij} for which $R \subset I_{ij}$ and I_{ij} is both a leftward and rightward extension of R. Thus for each different occurrence of R, there are $a', a'' \in \mathcal{A}$ and I_{ij} such that $a'Ra'' \subset I_{ij} \subset S$. The repeat rule ensures that, as S is being composed from left to right and as any repeat R in encountered, it will be clear how to extend beyond the repeat and into the next unique substring of S. The repeat rule is a necessary and sufficient condition for accurate determination of S. The repeat rule may impose a severe restriction over the shortest common superstring when S is composed of many repeats of a long pattern.

3 Algorithm

3.1 Initialization

Assuming the input satisfies the repeat rule, we begin as follows. Set the output $S = \phi$, the empty string. Form a trie [2] for each set of input fragments I_{i*}. The trie is a type of suffix tree composed of the union of I_{i*} in which all sequence fragments are rooted at the trie root. In other words, the child (level-1) nodes of the root represent the basic building blocks that begin fragments. The nodes at the second level represent the two-block strings beginning fragments, and so on.

Initialize a pointer array $P(1:p)$ so that element $P(i)$ points to the root of the trie of fragments I_{i*}. Initialize a length p array $L(1:p)$ to zero; element $L(i) = 0$ indicates that 0 blocks of S are initially known from contributions in I_{i*}.

3.2 Startup

To begin, search the subnodes found as children of the nodes pointed to by $P(1), \ldots, P(p)$ to find the unique basic building block string that can begin the sequence S. Note that every trie has a level-1 node (a child of the root) corresponding to s_1. By the repeat rule, if s_1 is repeated elsewhere in S, then there is some sequence copy I_{i*} that is missing each occurrence (its copy extends both

to the left and to the right, so it does not begin with s_1). If there are two or more candidates for s_1, extend each to s_2, s_3, and so on, as necessary, until application of the repeat rule excludes all other candidates.

Having found, for each i, an interval I_{ij_i} that starts S, determine the index i' whose fragment $I_{i'j_{i'}}$ is of maximal length. Assign $S = SI_{i'j_{i'}}$. For each sequence copy $i \in [1, p]$, set $L(i) = |I_{ij_i}|$, the number of basic building blocks in the corresponding first fragment. Let $P(i)$ remain pointing to the root.

3.3 Induction step

Now begin the intermediate step of the algorithm. Loop over all sequence copies for which $L(i) < |S|$, that is, those copies whose contribution does not include all of S as it is now determined. Note that if all sequence copies have identical values of $L(i) = |S|$, and all $P(i)$ are at root nodes, then S is fully determined and the algorithm terminates. The method also terminates with all $L(i) = |S|$ and some $P(i)$ at non-root nodes which happen to contain the terminal of a sequence fragment.

For each sequence i, start at node $P(i)$ and proceed through the trie using information from the known part of S. That is, while $P(i)$ points to an internal node of the current fragment I_{ij}, follow the remainder of S through the trie along fragment I_{ij}. At each node to node transition, update $P(i)$ to the correct child node and increment $L(i)$ by one, until either a fragment is exhausted but $L(i) \leq |S|$ (in which case set $P(i) =$ root and restart the intermediate step for this I_{i*}) or $L(i) = |S|$ and the current fragment is not exhausted.

When $L(i) = |S|$ and the current fragment is not exhausted, there are two cases. First, the current branch of the trie (the subtree below $P(i)$) has only the contribution from one fragment I_{ij}, in which case the remainder of I_{ij} determines an extension of S. In this case, append the remainder of the fragment to S, set $L(i) = |S|$, and reset $P(i) =$ root.

Second, there are several intervals in I_{i*} that can extend S. Because the trie for copy i of the input starts only with fragment beginnings, these two (or more) fragments have an identical prefix string. The repeat rule states that some fragment in another I_{i*} exists which extends leftward and rightward beyond the repeated section. Therefore, by the time all $L(i)$ are incremented up to the current value of $|S|$, at least one fragment will contain enough information to break the tie at $P(i)$. So leave $P(i)$ at the current node, leave $L(i) = |S|$, and continue to the next input copy I_{i*} for which $L(i) < |S|$.

3.4 Termination

When the end of S is reached, clearly all $L(i) = |S|$ and all $P(i) =$ root (or an interior node that signals a fragment termination). Should all $L(i) = |S|$ and some $P(i)$ not have this property, then the input sequences have violated the repeat rule and the algorithm terminates with an error condition.

It should be clear that the algorithm terminates for all legal inputs, because it is never possible to get into an infinite loop. Such a loop could only occur when some s_k is repeated in S and the algorithm does not know how to select the appropriate intermediate segments between copies of s_k. But the repeat rule always extends beyond the repeated s_k on both sides, clearly indicating which intermediate segment to follow next. Thus the repeat rule is sufficient to determine the unique superstring. The same argument shows that the method is guaranteed to give the correct S on all valid inputs.

Failure to meet the repeat rule implies that, at some point in S, it may be impossible to tell which path to take after a repeated s_k. If $S = ss\ldots ss$, some number of fragments repeated end to end, it is impossible to determine the length and content of S. Thus the repeat rule is a necessary condition as well.

Note that in some cases of incomplete input, consideration of valid extensions of the several candidate basic building blocks will indicate a shortest common extension of S (multiple tandem repeats $ss\ldots s$ are replaced with ss) and allow the method to continue; the cost of such a continuation may be greater than linear in the input size. The continuation from the violation condition requires the application of the startup step from the current values of P and L. However, such a continuation is not guaranteed to determine the unique common superstring, which violates the biology of the application.

4 Time Complexity

The running time of the algorithm is $O(N)$, where $N = \sum |I_{ij}| = p|S|$ is just the totality of all input data. The cost of forming and traversing the trie is just $O(N)$. There is the additional cost of checking branches of the trie nodes when applying the startup step, which is executed once at a cost of $O(N)$.

As an extension of the basic algorithm, when one overcomes invalid input by applying the initial step to data violating the repeat rule, an additional cost of at most $O(N)$ is imposed for each new startup. Thus if there are c candidate basic building blocks that must be extended, the time for

one reinitialization is $O(cN)$. Applying k such operations recursively (when a new reinitialization is required before the previous one is resolved) appears to have exponential cost $O(c^kN)$, which is prohibitive.

5 Conclusions

We have presented a linear time algorithm for solving the sequence assembly problem of finding the unique common superstring from a collection of overlapping input fragments. The method is distinguished from the NP-complete shortest common superstring solutions to the same biological problem by two main features: it is significantly faster and is guaranteed to produce a biologically meaningful answer. The method is based on the repeat rule, which states necessary and sufficient overlapping requirements on the input fragments.

Extensions of the basic method are straightforward. In particular, inexactly determined sequence fragments would cause difficulty for the exact method, but can be handled by using an inexact matching criterion in forming the initial trie on the input fragments. For instance, one might consider using a heuristic program [3, 4] and some statistical analysis of inexact matching [1] in order to derive quite good estimates of the unique common superstring on inexact inputs.

This presentation of the method assumes that each basic building block is formed in the correct 5'–3' orientation; details concerning the handling of complementary strands of DNA are straightforward and simply involve extra bookkeeping to identify complementary strands. The repeat rule should be extended in this case to prohibit the palindromic sequences called *inverted repeats.*

6 Acknowledgements

I am grateful to Eric Lander, who reminded me that, although they have been assembling sequences for many years, computational molecular biologists have not found a theoretically satisfactory solution to this problem. Thanks are also due to the National Research Council, which kindly covered the expenses of attending the Workshop on Computing and Molecular Biology.

References

[1] S. Karlin and F. Ost. Maximal segmental match length among random sequences from a finite alphabet. In L. M. Le Cam and R. A. Olshen, editors, *Proceedings of the Berkeley Conference in Honor of Jerzy Neyman and Jack Kiefer*, volume 1, pages 225–243. Wadsworth, 1985.

[2] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, Massachusetts, 1973.

[3] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[4] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences USA*, 85:2444–2448, 1988.

[5] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *Information Processing 83 (Proc. IFIP Congress, 1983)*, pages 53–64, 1983.

[6] R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucleic Acids Research*, 10(15):4731–4751, 1982.

[7] J. Storer. *Data compression: Methods and Theory.* Computer Science Press, 1988.

[8] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science*, 57:131–145, 1988.

[9] J. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83:1–20, 1989.