An Algorithm for the Extraction of DNA Fragments using Restriction Enzymes

Kaustav Das Pallab Dasgupta P.P. Chakrabarti
Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur, INDIA
{pallab,ppchak}@cse.iitkgp.ernet.in

***Abstract*— Extraction of DNA fragments from existing base sequences is an important step in genetic manipulation. Existing tools for sequence alignment and creating the restriction map of families of restriction enzymes do not give a solution to the extraction problem. In this paper we explain why the task of choosing the most suitable enzymes for an extraction is computationally non-trivial. We present an automated algorithm for solving the extraction problem. Experimental results show that our algorithm is capable of performing extraction from large DNA sequences efficiently.**

I. Introduction

Gene manipulation is one of the most exciting topics in *genetic engineering* [6]. Recent advances in genetic engineering have enabled new combinations of genetic material to be artificially constructed in the laboratory by the controlled insertion and manipulation of nucleic acid sequences. Plasmids and viruses carry these new sequences into host cells where they can be propagated and amplified. In various cases this facilitates transcription into mRNA and subsequent translation into proteins.

One of the main steps in genetic engineering is to extract the desired DNA fragment from a given base sequence. Once this extraction is done, the spliced DNA fragments are incorporated into vectors (such as plasmids) and used for transmission into host cells. Typically the task of cleaving a DNA sequence to extract the target sequence is done using *restriction enzymes*. These restriction endonucleases recognize specific DNA nucleotide sequences (called *recognition sequences*), and cleave the DNA double helix at or near these specific sites. The task of extracting a *target* DNA sequence from a given *base* sequence consists of two tasks, namely:

1) Finding out the sites in the base sequence that matches the target sequence,
2) Cleaving the base sequence near the endpoints of the match using appropriate restriction enzymes to extract a DNA sequence that is similar to the target sequence.

Typically a match need not be exact, but must have a high score based on a problem-defined matching function. However, an exact match (if it exists) is usually preferred.

Example 1: Suppose the given base sequence is:

CATGACGCGCG

and the target sequence is CGCG. In this case, exact matches of the target sequence can be found starting from the sixth and the eighth positions of the base sequence. These matches are denoted by $[6 \ldots 9]$ and $[8 \ldots 11]$.

On searching the restriction enzyme database [4], we find the following enzymes:

Enzyme	Recognition Sequence
FnuDII	CG.CG
NlaIII	CATG.
HhaI	GCG.C

The 'dot' indicates the position at which the cleavage takes place. For example, *NlaIII* can cleave the base sequence immediately after the fourth position, and *HhaI* can cleave the base sequence immediately after the ninth position. Therefore, by using these two enzymes the DNA fragment $[5 \ldots 9]$ can be extracted, which contains the target sequence.

It is interesting to note that though *FnuDII* has two cleavage sites in the given base sequence, it is not appropriate since it cleaves the target sequence as well. □

There are several tools (such as BLAST [1]) for finding matches for a target sequence in a base sequence. There are also some tools for computing the *restriction map* of a given base sequence [5], [7]. The restriction map of a base sequence with respect to a set of enzymes indicates the cleavage sites of those enzymes on the base sequence. However these tools do not solve the extraction problem automatically and currently the task of finding mutually compatible enzymes [1] for extracting the target sequence is solved manually. Since there are about 3660 restriction enzymes (today), the problem of finding out the best pair for a given problem instance is a non-trivial problem.

In this paper, we present an algorithm for finding out the best pair of enzymes for a given extraction problem. We present the main features of this algorithm that provide insights into the complexity of the problem. We have implemented a tool that implements this algorithm considering about 3660 restriction enzymes. We present experimental results indicating the computational efficiency of the tool.

The paper is organized as follows. Section II presents an outline of the proposed algorithm. Section III presents our method for creating the restriction map of a given base

[1] In some cases a single enzyme can cleave at both ends of the target sequence. In other cases, we require a pair.

sequence. Section IV presents the method of selecting the best pair on enzymes for the extraction. Section V presents experimental results.

II. Outline of the algorithm

The salient features of the proposed algorithm are as follows:

- The algorithm uses the popular sequence alignment tool BLAST [1] for finding the matching sites of the target sequence in the base sequence. Since BLAST is a local alignment tool, it will produce many matches (with low score) that align only a small portion of the target sequence with the base sequence. To eliminate these matches we choose only those matches that have a score greater than a threshold score of $2/3$ times the score of the perfect alignment.
- For computing the restriction map, we present an algorithm which creates a finite automaton out of the entire family of restriction enzymes. This automaton acts as the acceptor for the recognition sequences of the entire family. For a given family of restriction enzymes, this automaton is created once. The extraction algorithm creates the restriction map for a given base sequence by a single co-simulation of this automaton with the base sequence.
- We present several criteria for determining the compatibility between pairs of restriction enzymes. These criteria are used by our algorithm to create a compatibility matrix between restriction enzymes for a given instance of the extraction problem.
- The algorithm uses the compatibility matrix, the restriction map, and the matching sites to compute the best way to perform the desired extraction.

The following sections describe each of the above steps in detail.

III. Construction of the Restriction Map

For constructing the restriction map for a given base sequence, we create an acceptor for the recognition sequences of the restriction enzymes. The automaton for the acceptor is created in two steps:

1) We create the automaton for individual restriction enzymes using the Knuth-Morris-Pratt (KMP) algorithm [2].
2) We create the composite acceptor for the entire family of enzymes by computing a product of the individual automata.

The KMP algorithm involves pre-computing the following prefix function π for a given pattern P:

$$\pi[q] = \max\{k \mid k < q \text{ and } P_k \supset P_q\}$$

where P_k denotes the prefix of P consisting of the first k symbols of P, and $w \supset x$ denotes that w is a suffix[2] of x. In other words, $\pi[q]$ is the length of the longest prefix of P that is a proper suffix of P_q.

One difficulty in computing this function for the recognition sequences of restriction enzymes is due to the presence of ambiguity characters [3] in some recognition sequences. Each ambiguity character represents more than one nucleotide base. We resolve the problem by forming all the possible unambiguous recognition sequences represented by it. We then compute the prefix function for each sequence independently.

Example 2: The enzyme AccB1I has the recognition sequence GGYRCC, where Y and R are ambiguity characters. Y represents C or T, and R represents G or A. The four possible unambiguous sequences are:

GGCGCC
GGCACC
GGTGCC
GGTACC

We consider each sequence seperately and construct the respective automatons. □

The length of the recognition sequence of the majority of known restriction enzymes is between 4 and 10. Also the typical number of ambiguity characters (where they exist) are very few. Therefore the above method of expanding ambiguous sequences into sets of non-ambiguous sequences does not cause any serious combinatorial blow-up, except for a few enzymes that have non-ambiguous characters at both ends of the recognition sequence, but have a long run of N's in the middle. N is an ambiguous character that represents either of the four nucleotide bases.

Example 3: The restriction enzyme BcgI has the recognition sequence GACNNNNNNTGG. □

In these cases, the number of non-ambiguous sequences generated for a sequence having k N's would be 4^k. For example, the number of non-ambiguous recognition sequences for *BcgI* would be $4^6 = 4096$. Such sequences can thereby significantly increase the number of states in the composite automaton. To overcome this problem, we consider only the non-ambiguous front and rear portions of these sequences for creating their acceptors. We also set a special flag for these sequences, so that when one of the end portions matches with a base sequence, we return to the base sequence to examine whether the whole sequence matches at that particular position.

After computing the prefix function for each of the recognition sequences, we proceed to create the composite automaton that accepts occurences of these recognition sequences in the given base sequence. Each state S in the composite

[2]A k-length suffix of a string is the sequence of the last k symbols of the string.

automaton corresponds to a set of positions $\{x_1, \ldots, x_k\}$ in the individual recognition sequences $\{P^1, \ldots, P^k\}$ of the enzymes, such that whenever we reach state S after scanning a pattern T, then for each P^i, the first x_i symbols of P_i is the longest prefix of P_i that is also a suffix of T. The starting state of the composite automaton corresponds to the start of each recognition sequence. The following function recursively constructs the entire automaton starting from the composite start state.

Funtion RecursiveBuild(S)

1) Let the current state be $S = \{S_1, \ldots, S_k\}$, where S_i denotes the local state of the i^{th} automaton.
2) For each i, if S_i is an accepting state of the i^{th} automaton, then store the corresponding enzyme names in the state description of S.
3) For each nucleotide n do the following:
   a) Determine the next state S'_i for each S_i in the i^{th} local automaton for input n. Therefore, the composite next state is $S' = \{S'_1, \ldots, S'_k\}$.
   b) If S' is already present in the composite automaton, then add a transition from S to S' on input n.
   c) Otherwise, add the state S' in the composite automaton, add a transition from S to S' on input n, and recursively call RecursiveBuild(S').

The following example shows the individual automata for a couple of enzymes and their composite product.

Example 4: The Enzyme HindI has the recognition sequence CAC. Fig 1 shows the automaton that accepts this sequence. From each state, the transitions corresponding to the remaining inputs lead to state-1 (start state) of the automaton.

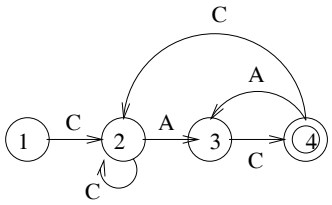

Fig. 1. Acceptor for the sequence CAC

The enzyme MboI has the recognition sequence GATC. The acceptor for this sequence is shown in Fig 2.

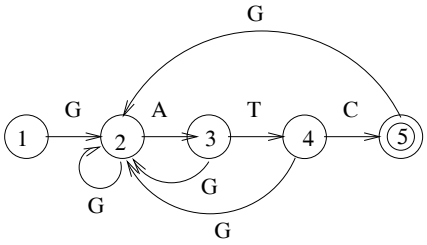

Fig. 2. Acceptor for the sequence GATC

The composite automaton computed by the function RecursiveBuild() is shown in Fig 3.

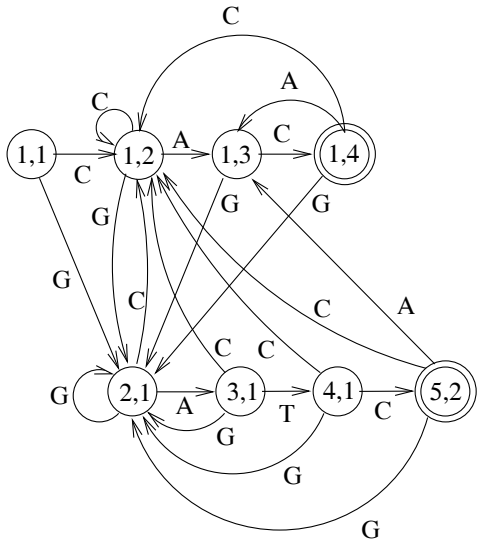

Fig. 3. The composite automaton

The states of the composite automaton are shown as doublets containing the local states of the two individual automata. From each state, the transitions corresponding to the remaining inputs lead to the start state of the composite automaton. □

It may be noted that the construction of the composite automaton is a one-time effort for a given family of enzymes. The automaton may be saved and re-used many times for creating the restriction maps of different base sequences. Once the automaton is constructed, the restriction map for a given base sequence can be constructed by simulating the automaton with the base sequence as the input.

IV. Finding the enzymes for extraction

In general, we look for two enzymes for the extraction, one each for the two ends of the match. The enzyme that cleaves near the start of the match will be called the *left-cut enzyme* and the one that cleaves near the end of the match will be called the *right-cut enzyme*.

Example 5: Consider the base sequence:

GCAGCTTTAAATAATGCAGGTACCTT

Suppose we wish to extract the sequence AATAATGCA. This sequence occurs in the base sequence at $[10 \ldots 18]$. The enzyme AhaIII has the recognition sequence TTT.AAA, and is therefore a candidate for the *left-cut enzyme*. The enzyme AhaB8I has the recognition sequence G.GTACC, and is a candidate for the *right-cut enzyme*. □

There are several constraints that influence the choice of the left-cut and right-cut enzymes for a given base sequence and a target sequence.

Admissibility:

The chosen enzyme should not have a cleavage site between the start and the end of a match. In Example 1 the enzyme FnuDII was rejected for this reason.

Mutual compatibility:

An enzyme X will be called *dependent* on another enzyme Y if enzyme Y cleaves the recognition site of X in a given base sequence. X and Y are incompatible (for becoming the chosen pair of enzymes) if each is dependent on the other [3].

Compatibility issues are highlighted through the following example.

Example 6: Consider the base sequence:

CGCGCATCGCGA

Suppose the target sequence is CATCG. The target occurs in the base sequence at $[5 \ldots 9]$.

Let us again consider the enzymes FnuDII (having recognition sequence CG.CG) and HhaI (having recognition sequence GCG.C). If we use the enzyme FnuDII, we can extract the sequence $[3 \ldots 9]$. Once we use FnuDII, it destroys the recognition site of HhaI, rendering the enzyme ineffective. However, if we use HhaI first and then FnuDII, then we are able to extract the target sequence exactly. For this base sequence, HhaI is *dependent* on FnuDII. □

Thus along with each occurrence of the recognition sequence of a particular enzyme, we also need to determine the set of restriction enzymes (called dependency set) that can cleave that occurrence of the recognition sequence and make it ineffective at that position.

We now outline the method for choosing the best pair of mutually compatible enzymes for an extraction. Let S^b denote the base sequence and S^q denote the target sequence. We use the notation $S[i \ldots j]$ to denote the subsequence of S between the positions i and j.

Let $S^b[Start_b \ldots End_b]$ and $S^q[Start_q \ldots End_q]$ be the respective fragments of the base and target sequence that have been aligned against each other in the particular BLAST alignment under consideration.

Let $E_{admissible}$ denote the set of *admissible* enzymes for the given alignment, that is, an enzyme E belongs to $E_{admissible}$ iff E does not have a cleavage site between $Start_b$ and End_b.

Each enzyme $E \in E_{admissible}$ may cleave the base sequence at one or more positions, producing one or more fragments. Consider the enzyme E and the fragment that contains the matched portion of the sequence. Cut_{left} is the position of the left end (the end near $Start_b$) of the fragment in S^b. Similarly, Cut_{right} is the position of the right end (the end near End_b) of the fragment in S^b. Formally,

Cut^E_{left} = max$\{Cut_i \mid Cut_i$ is a cleavage site of E in S^b and $Cut_i \leq Start_b\}$

Cut^E_{right} = min$\{Cut_i \mid Cut_i$ is a cleavage site of E in S^b and $Cut_i \geq End_b\}$

We further define two quantities *Left-overhang* and *Right-overhang* for each $E \in E_{admissible}$ for the given alignment:

Left-overhang(E) =
$$c\left(S^b[Cut^E_{left} \ldots Start_b], S^q[0 \ldots Start_q]\right)$$
Right-overhang(E) =
$$c\left(S^b[End_b \ldots Cut^E_{right}], S^q[End_q \ldots L_q]\right)$$

where $c(S_i, S_j)$ represents the cost of optimally aligning the two sequences S_i and S_j, and L_q is the length of the target sequence. A low value of overhang signifies that the enzyme has a cut near the corresponding end of the matching, and it is to be preferred for cleavage at that end.

The method for choosing the enzyme pairs works as follows:

1) Two ordered lists of the admissible enzymes are formed, namely *LeftCutList* in ascending order of *Left-overhangs* and *RightCutList* in ascending order of *Right-overhangs*.
2) We choose the first K (for some constant K) enzymes from both the lists as possible candidates for the *left-cut enzyme* and the *right-cut enzyme* respectively. We then form all the K^2 possible enzyme pairs (E_L, E_R) and score the pairs as:
$$Score(E_L, E_R) = c\left(S^b[Cut^{E_L}_{left} \ldots Cut^{E_R}_{right}], S^q[0 \ldots L_q]\right)$$
3) The pairs are then sorted in ascending order of this score. For each pair we determine if one enzyme is present in the dependency list of the other. In case only one is dependent on the other, we order the enzymes in the pair such that the dependent enzyme is applied prior to the other enzyme. In case both of them depend on each other, we simply discard the pair[4].

Finally, we output the enzyme pairs in this sorted list as the best candidates for the extraction of the target sequence from the base sequence.

V. Experimental Results

We have implemented a prototype tool for the extraction problem. The tool considers a set of 3660 restriction enzymes whose recognition sequences were taken from the Rebase [4] site. The composite automaton that can recognize these sequences has 1588 states.

The test cases for the base sequences were taken as fragments of the human chromosome-1 (as obtained from the

[3]If the depedency is one-way, then we can apply the dependent enzyme first to perform the extraction.

[4]In such cases, it might be possible to use a third enzyme to resolve the dependencies, but that possibility is not considered here.

NCBI ftp site). The target sequences were taken as random fragments of the base sequences. We have run the program varying the size of the base sequence and the target sequence.

Table I shows the results of running our program on these test cases. L_b denotes the length of the base sequence, L_q denotes the length of the target sequence. The third column, M_t, denotes the number of initial local alignments found by BLAST, while the fourth column, M_s, shows the number of significant global alignments among the local alignments found by BLAST. T_1 denotes the total runtime (in seconds), T_2 denotes the time taken by the BLAST procedure calls, and T_3 denotes the time taken to identify the set of enzymes including the creation of the restriction map.

L_b	L_q	M_t	M_s	T_1 (sec)	T_2 (sec)	T_3 (sec)
1×10^5	20	8	5	0.40	0.04	0.31
	50	7	1	0.15	0.04	0.05
	100	17	1	0.18	0.05	0.07
	200	27	1	0.16	0.04	0.06
2×10^5	20	2	1	0.19	0.04	0.06
	50	9	1	0.18	0.05	0.05
	100	37	1	0.17	0.05	0.05
5×10^5	20	6	1	0.22	0.06	0.06
	50	602	1	0.49	0.21	0.16
	100	38	1	0.23	0.06	0.10
1×10^6	20	8	1	0.33	0.11	0.05
	50	34	1	0.38	0.11	0.08
	100	16	1	1.22	0.80	0.05
2×10^6	20	10	3	0.70	0.17	0.23
	50	26	1	0.53	0.18	0.06
1×10^7	20	12	3	2.21	0.73	0.20
	50	60	1	2.19	0.79	0.06
	100	17	1	1.96	0.69	0.07
	200	415	2	2.49	0.92	0.27
	500	33	1	2.08	0.72	0.08
2×10^7	20	62	11	4.48	1.29	0.86
	50	44	1	3.70	1.29	0.07
	100	41	1	3.60	1.26	0.07
	1000	114	1	3.81	1.35	0.18
5×10^7	20	17	1	10.25	3.10	0.12
	50	30	1	9.02	3.15	0.06
	200	40	1	9.04	3.19	0.06
2×10^8	20	18	1	43.71	15.64	0.08
	100	26	1	44.52	15.56	0.06

TABLE I
RESULTS OF THE EXTRACTION TOOL

For each base sequence we have randomly selected target subsequences of length $20, 50, 100, 200$ and 500 respectively. We have omitted the results for which no enzyme set could be found that can produce the target sequence. As the length of the target sequence increases, it becomes more probable that the enzymes that cut near the end also cut somewhere in between, and therefore no enzyme may be found which cuts only near the ends of the match. We see that in general target sequences of length upto 200-500 can be extracted by using restriction enzymes (by this method).

The difference between T_1 and T_2 + T_3 is due to the time required to read the base and the target sequence from the disk.

VI. ACKNOWLEDGEMENTS

P.P. Chakrabarti acknowledges the support of CSIR (NMITLI Bioinformatics Project) for this work.

VII. CONCLUSION

The current tool only provides the basic mechanism for finding enzymes for a given extraction problem. In practice however several other constraints dictate the choice of the enzymes to be used in a wet lab. We feel that such constraints may be suitably integrated into our tool to solve the problem more effectively with little or no computational overhead.

VIII. REFERENCES

[1] Altschul, S.F., *et al*, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25, 3389-3402, 1997.
[2] Knuth, D.E., Morris, J.H., and Pratt, V.R., Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323-350, 1977.
[3] Nomenclature Committee, 1985, *European Journal of Biochemistry*, 150, 1-5, 1985.
[4] Rebase, http://rebase.neb.com
[5] Restriction Enzyme Site Mapper version 3 http://www.restrictionmapper.org
[6] Strickberger, M.W., *Genetics*, Prentice Hall, 1985.
[7] Webcutter, http://www.firstmarket.com/cutter/cut2.html