BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 02/10 pp11–20
Volume 6, Number 1, March 2023
DOI: 10.26599/BDMA.2022.9020030

# A Method for Bio-Sequence Analysis Algorithm Development Based on the PAR Platform

Haipeng Shi, Huan Chen, Qinghong Yang, Jun Wang, and Haihe Shi*

**Abstract:** The problems of biological sequence analysis have great theoretical and practical value in modern bioinformatics. Numerous solving algorithms are used for these problems, and complex similarities and differences exist among these algorithms for the same problem, causing difficulty for researchers to select the appropriate one. To address this situation, combined with the formal partition-and-recur method, component technology, domain engineering, and generic programming, the paper presents a method for the development of a family of biological sequence analysis algorithms. It designs highly trustworthy reusable domain algorithm components and further assembles them to generate specifific biological sequence analysis algorithms. The experiment of the development of a dynamic programming based LCS algorithm family shows the proposed method enables the improvement of the reliability, understandability, and development efficiency of particular algorithms.

**Key words:** partition-and-recur (PAR); domain engineering; biological sequences; feature model; component assembly

## 1 Introduction

With the advent of the post-genomic era, bioinformatics has entered a golden period of development and has become one of the most studied areas in the 21st century. Bioinformatics aims to reveal the complexity of genome information structure and the fundamental law of genetic language. This field is an organic combination of the three major scientific issues, namely, genome, information structure, and complexity in science and technology in the 21st century[1].

The field of biological sequence analysis is accompanied by many popular problems, such as the longest common subsequence (LCS)[2], sequence alignment[3], genome rearrangement[4], and sequence assembly[5]. During decades of research, scientists have proposed many algorithms to solve these problems. For example, in the LCS problem, dozens of algorithms such as Wagner-Fischer[6], Hischberg[7], TOP_MLCS[8], Fast_LCS[9], Level_DAG[10], etc., can be used. Problem-solving processes may be very similar, but certain differences still exist among these algorithms, resulting in variations in their application scenarios, algorithm efficiency, and accuracy.

The LCS problem includes two sequences. It has been an important topic, and the dynamic programming based LCS algorithm is the most widely used. In the 1970s, Wagner and Fischer[6] proposed the earliest dynamic programming solving algorithm. This algorithm needs to construct a matrix of $(m+1)\times(n+1)$ as the computing space, where $m$ and $n$ are the numbers of characters of the given sequences. The algorithm applies the standard dynamic programming strategy, and its time complexity and space complexity are both $O\ (m\times n)$.

Given the high cost of time and space, using the basic dynamic programming based LCS algorithm to solve problems involving long sequences is difficult. Thus, researchers have conducted follow-up studies. In

• Haipeng Shi is with the School of Software, Jiangxi Normal University, Nanchang 330022, China. E-mail: option2001@jxnu.edu.cn.
• Huan Chen, Qinghong Yang, Jun Wang, and Haihe Shi are with the School of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330022, China. E-mail: 202041600064@jxnu.edu.cn; yangqh120@163.com; wangjun2018@jxnu.edu.cn; haiheshi@jxnu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2022-06-28; revised: 2022-08-12; accepted: 2022-08-17

1977, Hirschberg[7] designed Hirschberg's algorithm, which has a linear space complexity. In 1982, Nakatsu et al.[11] improved the basic dynamic programming LCS algorithm by calculating the matrix in line or diagonal order[11]. In 1988, Myers and Miller[12] used the divide-and-conquer method to optimize the basic dynamic programming algorithm, and reduced the space complexity to $O\ (m+n)$.

Many variants of the LCS problem have emerged in recent years; these variants include the constrained LCS (CLCS) problem, merged LCS problem, and repetition-free LCS problem. In 2003, Tsai[13] first proposed the CLCS problem and its solution algorithm. Based on the LCS problem, the CLCS problem also involves a constraint substring, which is a sequence fragment containing some important features or information and may be omitted in the calculation of LCS; all the LCSs must contain the constraint substring. In the solution, algorithm breakpoints are introduced, and the CLCS is solved based on the breakpoints generated by each character of the constraint substring in the input sequence. Peng[14] introduced the penalty factor to solve the CLCS problem based on the LCS dynamic programming algorithm, that is, adding the penalty factor to the entry corresponding to each path in the score matrix and finally selecting the path with the highest score and containing the constraint sequence as the computing result. In 2005, Arslan and Eğecioğlu[15] presented an algorithm based on Tsai's algorithm. This algorithm calculates the prefix sequence of optional CLCS in the score matrix of two target sequences and deletes most intermediate calculation results that are unnecessary for result calculation, therefore reducing the complexity of Tsai's algorithm. In 2011, Chen and Chao[16] improved Arslan and Eğecioğlu's algorithm on top of the optimal sub-structure theory.

The solving algorithms for the same problem are complex, their application scenarios are different, and the algorithm development cycle is long. In addition, users experience difficulty in distinguishing and selecting a solution from the algorithm family. To address these issues, this paper combines the ideas of the formal partition-and-recur (PAR) method, component technology, generic programming, and domain engineering, and presents a new method for the development of an algorithm family, i.e., biological sequence analysis algorithm development. The method generates specific domain algorithms through the design of highly trustworthy reusable algorithm components, thus enabling the optimization of the algorithm development process through the reuse-based design. Moreover, an example of the use of this method for the development of a dynamic programming based LCS algorithm family is illustrated.

## 2 Preliminary

The following subsections briefly introduce the PAR method, domain engineering, and dynamic programming based LCS algorithms used in this study.

### 2.1 PAR method

The PAR method is a unified software development method[17, 18]. It is mainly based on the PAR theory, and it covers the brute-force method, divide-and-conquer, greedy method, and other typical algorithm design methods. The PAR method consists of recurrence-based algorithm design language (Radl), abstract programming language (Apla), unified-algorithm design-and-proof method, and its supporting platform, including Apla→C++/C#/Java program generation systems. PAR supports the formal design process from Radl specifications to Radl algorithms and the automatic generation of Apla programs from Radl algorithms as well as advanced programming language programs from Apla programs.

Radl language is an algorithm design language in the PAR method. Its key idea is to develop algorithms based on recursive relations. It can also be used to describe the formal problem specification and specification transformation rules. With mathematical referential transparency, Radl can be easily used for the formal derivation of an algorithm. It is mainly user-oriented as the front end of the Apla.

Apla is an intermediate conversion language in the PAR platform. The Radl $\rightarrow$ Apla program generator considers it as the target language, and Apla→C++/C#/Java program generators deem it as the source language. Apla fully embodies popular programming ideas, such as functional and data abstractions, and it can be easily used for program development. Moreover, the Apla program has high readability, understandability, and verifiability and is easy to convert into various executable programs. Apla supports generic programming mechanisms, such as type parameterization, subroutine parameterization, and user-defined generic abstract data types (ADTs).

The formal algorithm development theory based on the PAR method and platform can obtain highly reliable algorithmic programs. The development flow of algorithmic programs using the PAR method is as follows.

**Step 1:** Obtain the natural language requirements of the problem to be solved, and then use the Radl language to describe the problem and construct its formal function specification.

**Step 2:** In accordance with the established formal functional specification, the problem is divided into several independent sub-problems, and all of them can continually be decomposed until the obtained subproblems can be directly solved.

**Step 3:** Construct the problem-solving recursive relationship, and combine it with the initialization conditions of functions and variables to obtain the Radl algorithm.

**Step 4:** Based on the recursive relationship, determine all the variables needed in the program and describe the variation rules and functions of each variable, and furthermore develop the program loop invariant based on the new loop invariant development strategy in PAR.

**Step 5:** Based on the Radl algorithm and the obtained loop invariant, the Apla program for solving the problem is obtained using the Radl$\rightarrow$Apla program generation system.

**Step 6:** Supported by PAR platform, the Apla program is transformed into an equivalent advanced programming language program.

## 2.2 Domain engineering

In the 1960s, the idea of software reuse was proposed to deal with the software crisis. It is an effective way to carry out the industrial production mode of the software industry[19]. Software reuse can be divided into the production stage of reusable software assets and the development stage of application systems based on reusable software assets[20]. Domain engineering is located in the production stage of reusable software assets, and is the main means for the production of reusable software assets. It mainly aims to identify software systems and develop and organize domain reusable software assets. These reusable software assets are an indispensable base for the subsequent software development.

Domain engineering can be divided into domain analysis, design, and implementation based on the development process.

Domain analysis is the foundation of domain engineering and includes three aspects as follows. First, the requirements of some typical systems in the domain are analyzed considering the expected demand change, technology development, and objective constraints. Second, the boundary range of the domain is determined, and the common features and variable features of the domain are identified. Third, the functional requirements that are worthy of reuse in the domain are obtained and abstracted to construct the domain model.

The domain design identifies and organizes reusable assets such as architecture and components, in the domain based on the domain model. Based on the first two stages, domain implementation is applied to develop new software systems using domain reusable assets. Domain engineering can also be used for the development of new algorithms in the same field as follows. First, the domain model is established based on existing typical algorithms of the domain, and the specification of a new algorithm is described based on the model and a problem requirement. Second, in accordance with specifications, the appropriate functional components are selected from the domain to assemble a new algorithm. Through the application of the idea of domain engineering to algorithm development, algorithm development can reuse a large number of reusable component assets of this algorithm domain. It does not need to start from scratch, and only needs to implement the unique functional components of the new algorithm and select appropriate reusable components to assemble the required new algorithm.

In the 1990s, feature-oriented domain analysis (FODA)[21] was introduced into domain engineering by the Institute of Software Engineering of Carnegie Mellon University. The main idea of FODA is the extraction of necessary functional features from the domain and the use of their interaction among them to construct the domain feature model as an important part of the domain model. The features in domain engineering can be divided into two categories: common features and differences. The former exist in all components of the domain and show commonness; the latter only exist in some domain components and exhibit variability. Compared with commonness, variability is more worthy of study. Variable features can be divided into the following types.

(1) **Optional features.** They play an extra optimization role in the domain and exist in some algorithms;

(2) **Mutual exclusion feature (XOR).** When the same function in the domain has a variety of implementations, these implementations cannot coexist with each other, that is, mutually exclusive features;

(3) **Multiple features (OR).** The given function

should include at least one or more features of the feature set, namely, multiple features.

Figure 1 shows the representation of mandatory features and three variable features in the domain model.

### 2.3 Dynamic programming based LCS algorithms

The LCS and CLCS algorithms are based on dynamic programming, and their problem-solving ideas are similar. The former's score matrix is two-dimensional, and that of the latter, whose scoring template is different from that of the former, is three-dimensional. Given the space limitation, only the basic dynamic programming solution of the LCS algorithm is introduced here.

First, the following definitions are given:

**Definition 1 (Sequence).** A string consists of elements in a finite set of characters $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ arranged in a certain order.

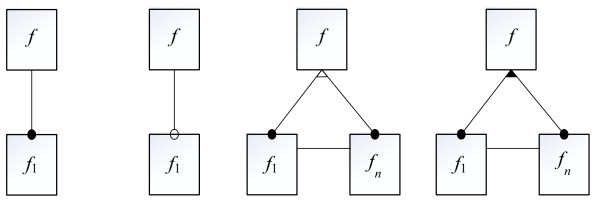
**Definition 2 (Subsequence).** For a given sequence $a$, its subsequence can be obtained by deleting zero or more characters from $a$. For example, given a sequence $\{A, G, T, T, G\}$, $\{G, T, G\}$ and $\{A, T, T, G\}$ are its subsequences.

**Definition 3 (Common subsequence).** Given a sequence set $S = \{a_1, a_2, \ldots, a_n\}$, if sequence $b$ is the subsequence of all sequences in $S$, $b$ is called the common subsequence of $a_1, a_2, \ldots, a_n$.

**Definition 4 (LCS).** If $b$ is the common subsequence of $a_1, a_2, \ldots, a_n$, and its length is greater than other common subsequences of $a_1, a_2, \ldots, a_n$, then $b$ is the LCS of $a_1, a_2, \ldots, a_n$. Evidently, the LCS is not unique.

The dynamic programming based LCS algorithm was first proposed in 1977. It mainly adopts dynamic programming to construct a matrix (score matrix) to store the calculation results between input sequences. Finally the LCS is obtained by backtracking the recorded results. The values in the matrix are calculated by the following recursive equation:

$$L[i,j] = \begin{cases} 0, & i=0 \text{ or } j=0; \\ L[i-1,j-1]+1, & A[i]=B[j]; \\ \max(L[i-1,j], L[i,j-1]), & A[i] \neq B[j] \end{cases} \quad (1)$$



(a) Mandatory (b) Optional (c) XOR (d) OR

**Fig. 1 Feature relationship diagram.**

where $L[i,j]$ denotes the value in the matrix, $A$ and $B$ are the input sequences, and $i, j$ are the index of the characters in $A$ or $B$, respectively.

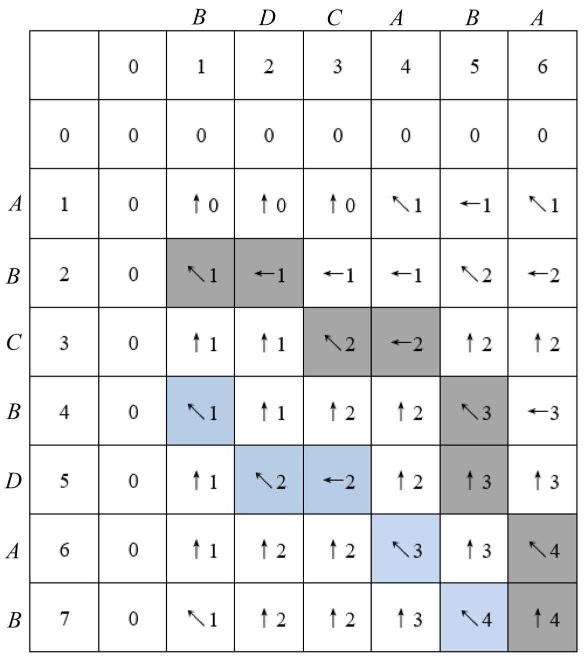
For example, given two sequences $\{B, D, C, A, B, A\}$ and $\{A, B, C, B, D, A, B\}$, Fig. 2 shows the solving score matrix and the backtracking path.

The value of each $[i, j]$ is shown in Fig. 2, and the arrow denotes the backtracking direction. The gray and blue paths are backtracking paths from the highest score to the initial point, i.e., the path of the LCS. From the backtracking path in Fig. 2, we can obtain the LCSs of the input sequence, which are $\{B, C, B, A\}$ and $\{B, D, A, B\}$, respectively.

## 3 Method for Biological Sequence Analysis Algorithm Development

In biological sequence analysis, various algorithms are used to solve the same problem. Complex similarities and differences exist among these algorithms, which lead to differences existing in their application scenarios, solving efficiency, or accuracy. In the paper, we present a method of biological sequence analysis algorithm development by combining the ideas of the PAR method, component technology, generic programming, and domain engineering. It can be applied to the rapid development of algorithms for solving a class of similar problems. The process is as follows:

(1) To analyze the solving algorithm family for a biological sequence analysis problem and abstract



| | | | *B* | *D* | *C* | *A* | *B* | *A* |
|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *A* | 1 | 0 | ↑0 | ↑0 | ↑0 | ↖1 | ←1 | ↖1 |
| *B* | 2 | 0 | ↖1 | ←1 | ←1 | ←1 | ↖2 | ←2 |
| *C* | 3 | 0 | ↑1 | ↑1 | ↖2 | ←2 | ↑2 | ↑2 |
| *B* | 4 | 0 | ↖1 | ↑1 | ↑2 | ↑2 | ↖3 | ←3 |
| *D* | 5 | 0 | ↑1 | ↖2 | ←2 | ↑2 | ↑3 | ↑3 |
| *A* | 6 | 0 | ↑1 | ↑2 | ↑2 | ↖3 | ↑3 | ↖4 |
| *B* | 7 | 0 | ↖1 | ↑2 | ↑2 | ↑3 | ↖4 | ↑4 |

**Fig. 2 Score matrix of dynamic programming.**

similarities and differences from them, this study further establishes the feature model of the algorithm domain from three aspects, namely, service (S), function (F), and behavior (B), by means of the FODA method.

(2) Based on domain feature model, the main function is designed as the algorithm component, and the dependencies between the components can be determined based on the interaction between the algorithm components.

(3) According to the PAR method, for the precise description of the formal specification of each algorithm component by Radl language and definition of ADTs based on the data dependency relationship between them, all ADTs must be further implemented by the Apla language and the domain algorithm component library be obtained.

(4) In accordance with the algorithm functional specification, compatible algorithm components from the component library are selected to assemble the required Apla program, and an executable high-level programming language program is generated by means of the PAR platform.

The required algorithm program can be assembled on top of the reusable components. Through this manner, the algorithm development efficiency can be improved. In addition, the verifiability of the Apla language and the program generation systems of PAR platform enables the improvement of the accuracy and reliability of the algorithm.

## 4 Implementation of Dynamic Programming Based LCS Algorithm Family

Here, we design and implement the dynamic programming based LCS algorithm family to demonstrate the proposed method.

### 4.1 Domain analysis

The solution method of the LCS problem has a strong commonality with that of the CLCS problem in the variant problem. We take the LCS and CLCS solution algorithms based on dynamic programming as a unified research field and apply the implementation method of the LCS algorithm family based on the PAR platform proposed in this paper to study it. First, the domain feature modeling method, that is, FODA, is used to analyze the S, F, and B of the DP-MLCS algorithm domain. Then, based on the established domain feature model, the LCS domain algorithm component library based on dynamic programming is designed and implemented.

In Section 2.3, the original LCS algorithm based on dynamic programming is introduced. Based on the idea of dynamic programming, a two-dimensional score matrix is established, and Eq. (1) is used as a scoring template to recursively solve the value of the score matrix. Finally, the LCS of the target sequence is obtained by backtracking from the position of the highest score. Afterward, the research in this aspect has achieved many improvements based on the original dynamic programming LCS algorithm and proposed many new solving algorithms, which mainly focus on the following two aspects: (1) Use the divide-and-conquer method to split the two target sequences to reduce the scale of the problem to solve the problem more quickly; (2) Optimize the score matrix solution formula (which we call the scoring template here) and delete the corresponding values of positions that do not need to be used in the LCS solution process to improve the computational time and space efficiency. The solution and improvement methods of the CLCS algorithm are the same as those of LCS, and the main difference is that the score matrix and template established by the CLCS algorithm are relatively complex.

The dynamic programming based LCS and CLCS algorithms are extremely similar, and the main difference is that the CLCS algorithm needs to establish a three-dimensional score matrix. Figure 3 shows the operation flow of the LCS and CLCS algorithms.

As this paper considers biological sequences as the experimental object, the LCS search algorithm first needs to check whether the input sequence has a biological significance (for example, DNA sequences contain only $A$, $C$, $G$, and $T$ characters). Then, based

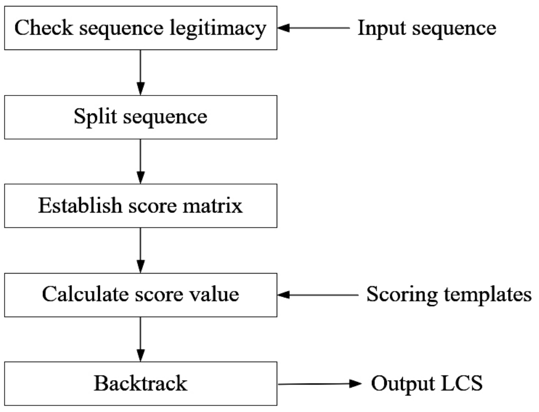


**Fig. 3 Flow chart of LCS algorithm.**

on the needs of the algorithm to establish the score matrix, the scoring template is selected depending on the demand and used to calculate the score of each position in the score matrix. Finally, all positions with the highest score in the score matrix are traced back to output all the LCSs that meet the requirements.

Based on the accurate analysis of the LCS and CLCS algorithm domains, we become familiar with the operation process of the algorithm in this field, the main functions of each process, and the differences between the algorithms in this field. The FODA modeling method can be used to extract the common and difference features to construct the LCS domain feature model.

(1) Sequence legitimacy check (`seq_check`) is a preprocessing operation that must be performed before each algorithm runs as a common function.

(2) Parts of the algorithms in this field use the idea of the divide-and-conquer method to split the input sequence. Thus, the search problem of LCS is decomposed into a search problem of LCS with two short sequences, and the split operation (split) is considered an optional function.

(3) All algorithms in this field need to establish a score matrix to store the score. The LCS algorithm establishes a two-dimensional score matrix, and the CLCS algorithm needs to establish a three-dimensional score matrix. The construction operation of the score matrix (`dp_matrix`), as a common function, contains two mutually exclusive subfeatures: two_dimen and three_dimen.

(4) The LCS algorithm can search for all the LCSs or determine the length of the LCS. For the former, the score source operation (`rmb_source`) needs to make a record of the source of values at each position in the score matrix for backtracking.

(5) The algorithm in the domain calculates the value in the score matrix in accordance with the scoring template and sets the scoring template (`scoring_temp`) operation as a common function, which contains two subfeatures, i.e., the LCS and CLCS scoring templates.

(6) The result output function (`result_op`), as a common function in the field, contains a subfeature output mode. The output model has two subfeature output sequence results (`seq_op`) and output score results (`score_op`), where the sequence output needs to be backtracked.

Taking the LCS search service as the main service in this field, the LCS domain feature model is established in accordance with the above analysis (Fig. 4).

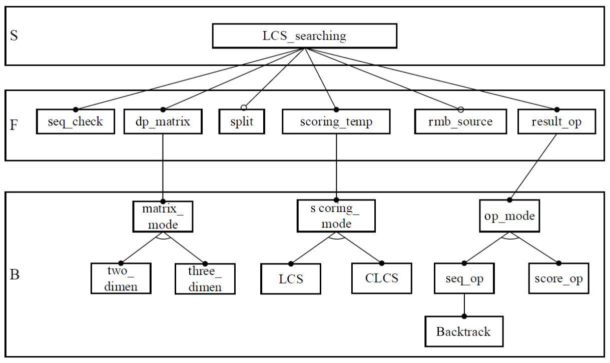


**Fig. 4 LCS domain feature model.**

### 4.2 Domain design and implementation

Here, the domain design and implementation are carried out based on the aforementioned LCS domain feature model.

#### 4.2.1 Component design

The functions and features in the LCS domain feature model are abstracted as components, and the dependency graph between components is established based on the running process of the algorithm and the data dependence between features. Here the six main algorithm functions in the LCS domain feature model are considered as the main components: sequence legitimacy check component (`seq_check` component), dynamic programming matrix pattern component (`dp_matrix_mode` component), split component (`split` component), scoring template pattern component (`scoring_temp_mode` component), remembering the score source component (`rmb_source` component), and result output component (`result_op` component). The dynamic programming matrix pattern component contains two sub-functional components: two-dimensional (`two_dimen` component) and three-dimensional matrix components (`three_dimen` component). The result output component includes two sub-functional components: the sequence result output component (`seq_op` component) and the score result output component (`score_op` component). The backtrack component is used as the auxiliary component of the `seq_op` component. The `seq_check` components and split component function before other components run and are not expressed in dependency diagrams. The connection with arrows represents the dependence of one component on another, and the dependence between components is shown in Fig. 5.

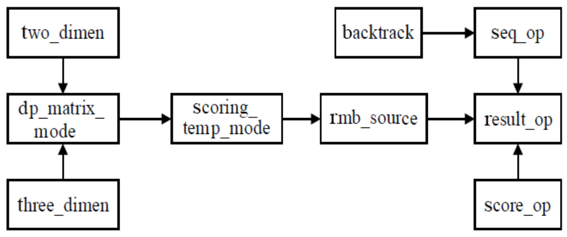


**Fig. 5 LCS component dependency diagram.**

### 4.2.2 Component implementation

After determining the dependencies between components using the LCS domain feature model, this paper applies the PAR method to formally implement the LCS algorithm component library. First, Radl is used to accurately describe the functional specification of algorithm components. Then, based on the data dependence between components, the components that depend on the same data source are integrated into an ADT. Finally, all ADTs are abstracted and realized using the Apla language. Here, the functional specification of the `seq_check` component is given as an example.

|[**in** seqs [$m$] [$n$]: alray of sequence; $a[s]$: array of character; $m$, $n$, $s$: integer; **out** flag: boolean]|

AQ: $m \geqslant 20$, $n \geqslant 20$, $s \geqslant 0$

AR: flag=$(\forall i, j{:}0 \leqslant S \leqslant i \leqslant m,\ 0 \leqslant j \leqslant n{:}(\exists k{:}0 \leqslant k \leqslant s{:}\ \text{seqs}[i][i] = a[k]))$

In the formal specification above, in and out are keywords defined in the PAR platform, representing input and output identifiers, respectively. Array, boolean, and integer types are predefined in the PAR platform. Here, AQ represents the pre-condition required by the component, and AR denotes the post-condition of the component. The `seq_check` component judges the legitimacy of the sequence by comparing the character types owned by the input sequence and the type of biological sequence. The seqs refers to the storage array of the input sequence, $a$ stands for the legitimate character array owned by the biological sequence (such as DNA is $A, C, G, T$), and the output flag is a boolean value (if true, it represents the sequence legitimate; otherwise, it is an illegal sequence).

The Apla language supports directly using ADTs and abstract processes to write algorithm programs, which can describe algorithm problems clearly and intuitively, and ensures program reliability by verifying program correctness. As the input language of the PAR platform program generation system, the Apla program can be easily converted to C++, Java, Python, and other programs. According to the LCS domain feature model and considering the dependencies between components and the relationship between the data types and functions involved in the field, the Apla language is used to define and implement LCS domain components, thus forming a highly abstract LCS algorithm component library. The ADT designed based on the data dependencies between the components are listed below, and the detailed implementation codes are omitted for the sake of space.

(1) sequence abstract data type: Seqs

```
define ADT Seqs (sometype elem)
   type Seqs = private;
   var
     seqs:Array [String];
     constraint_seqs:Array [ ];
     DNA:Array [ ];
     Protein:Array [ ];
     procedure read_seqs();
   function seq_check(seqs; type:
    integer): boolean;
   function split (seqs);
enddef
```

Considering that the seq_check component and split components are operated on the input sequence, ADT Seqs is used to encapsulate these components and their required data and auxiliary components. Four variables are defined: seqs for storage of input sequences, `constraint_seqs` for storage of constraint sequences, DNA and Protein for storage of legitimate characters of DNA and protein, respectively. The procedure `read_seqs` is used to read the input sequence. The `seq_check` function and the split function implement the `seq_check` component and split component, respectively.

(2) dynamic programming based matrix ADT: `dp_matrix`

```
define ADT dp_matrix
  type dp_matrix = private;
  procedure apply_memory(length_s:
   integer; length_t: integer; length_p:
   integer; proc init_score ());
  procedure memory_score (proc scoring
    (template_num));
  function max_score: integer;
  function get_score (i: integer; j:
    integer; k: integer): integer;
  procedure result_op(func
finally_score():integer; proc
backtrack(score_source))
enddef
```

As the backtracking function of the dynamic programming LCS algorithm is carried out on the score matrix constructed by the input sequence, the ADT `dp_matrix` encapsulates the components related to the score matrix operation and the result output operation. The `apply_memory` creates a storage space for the score matrix based on the lengths of the input and constraint sequences. The default length of the constraint sequence *p* is 0, and it contains a subroutine `init_score` to initialize the score matrix. The `memory_score` program stores the scores. It contains a subprogram scoring that selects different scoring templates based on the label of the scoring template. Function `max_score` and `get_score` return the highest and corresponding position scores, respectively. The `result_op` program realizes the function of the `result_op` component and its subcomponents. Function `finally_score` is the final result of the LCS algorithm for solving the LCS length in linear time; the subroutine backtrack outputs all LCS backtracking, which requires the support of ADT `Score_source`.

(3) source of score ADT: `Score_source`

```
define ADT Score_source
  type Score_source = private;
  procedure apply_memory(length_s:
integer; length_t: integer; length_p:
integer);
  function get_source(i: integer; j:
integer): integer;
  procedure set_source(i: integer; j:
integer);
enddef
```

The ADT `Score_source` remembers the score source of the value of each position in the score matrix, the `apply_memory` program dynamically creates storage space using the length of the input sequence, and `set_source` and `get_source` implement the access operation of the score source.

For example, the parallelization of cutoff pair interactions is mature in CPUs and typically employs a voxel-based method.

## 4.3 Algorithm assembly and experiment

### 4.3.1 Algorithm assembly

We use the established LCS component library to assemble the dynamic programming based LCS algorithm. The instantiation is shown below. The following Apla program can be converted to an executable C++ program through the Apla→C++ program generation system of the PAR platform.

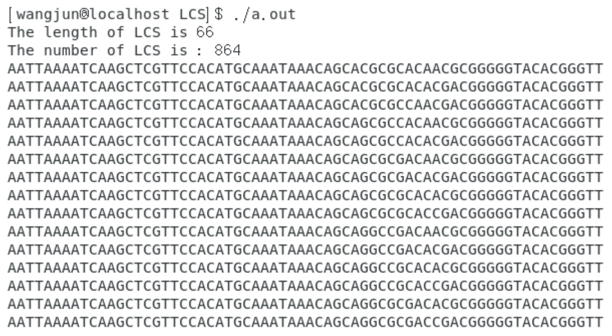
```
Program LCS:
  const path_infile: string; /* file
path of input sequence*/
  const path_outfile: string ; /* file
path of output LCS*/
var
  seqs: Array [string]; /*save the
input sequence*/
  constraint_seqs: Array [ ]; /*save
the constraint sequence */
  type: integer; /*The type of input
sequence*/
  template_num:string; /*The type of
scoring template */
  /*instantiation of functional
components used by LCS*/
1: ADT lcs_Seqs: new Seqs ();
2: ADT lcs_matrix: new dp_matrix(seqs;
constraint_ seqs);
3: ADT score_source: new
score_source(seqs; constraint_ seqs);
/*LCS Searching manipulate procedure*/
4: procedure LCS_search_mani (lcs_Seqs;
lcs_matrix; score_ source)
5: var
length_s, length_t, length_p: integer;
6: begin
7: lcs.Seqs.read_seqs (path_infile);
8: lcs.Seqs.seq_check (seqs;type);
9: lcs_matrix.apply_memory(length_s;
length_t; length_p);
10: lcs_matrix.memory_score(template_num);
11: lcs_matrix. result_op.finally_score;
12: lcs_matrix. result_op.backtrack;
13: end
```

### 4.3.2 Experiment

GenBank is a well-known nucleic acid database established and maintained by the National Center for Biotechnology, and it has a large number of biological gene sequences. We download the DNA sequence fragments of rice from the GenBank database as the experimental data, use the LCS algorithm based on dynamic programming generated by the above assembly to calculate the LCS of the two sequences, and carry out experiments on the DNA sequence fragments with lengths of 50, 100, 150, 200, 250, and 300. The experimental results are shown in Table 1. The LCS algorithm based on the dynamic programming generated by component library assembly is used to solve quickly and accurately the LCS subsequence of two input sequences with different lengths. Figure 6 shows the results of solving the LCS of two DNA sequences with

**Table 1 LCS experimental results of input sequences with different lengths.**

| Sequence length | Number of LCSs | LCS length |
|---|---|---|
| 50 | 1 | 28 |
| 100 | 15 | 61 |
| 150 | 510 | 91 |
| 200 | 384 | 121 |
| 250 | 46 048 | 150 |
| 300 | 39 552 | 187 |



```
[wangjun@localhost LCS]$ ./a.out
The length of LCS is 66
The number of LCS is : 864
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCACGCGCACAACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCACGCGCACACGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCACGCGCCAACGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCCACAACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCCACACGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCGACAACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCGACACGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCGCACACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGCGCGCACCGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCCGACAACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCCGACACGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCCGCACACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCCGCACCGACGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCGCGACACGCGGGGGTACACGGGTT
AATTAAAATCAAGCTCGTTCCACATGCAAATAAACAGCAGGCGCGACCGACGGGGGTACACGGGTT
```

**Fig. 6 Experimental result of LCS assembly algorithm.**

sequence numbers 30 229 045 and 30 229 047. The length of LCS shown in Fig. 6 is 66, and the number of LCS is 864.

Under the support of the PAR platform program generation system, we manually assemble components to obtain the LCS algorithmic program in Apla language and then convert it into the corresponding C++ program in a semi-automatic manner. By assembling the component library of the LCS domain to generate the LCS algorithm, it not only eases the writing of the algorithm, reduces the redundancy of the code, improves the reliability, reusability, and maintainability of the assembly algorithmic program, but also assembles and generates different types of LCS algorithms supported by the component library of the LCS domain based on different data requirements of users, thereby improving the universality of LCS algorithm components.

## 5 Conclusion

Based on the PAR platform, combined with domain engineering, formal method, generic programming, and other technologies, the paper presented a new development method for biological sequence analysis algorithms. The method attempted to design domain-oriented and highly reliable reusable algorithm components and mechanically generate specific algorithms by assembling such components. An example of the implementation dynamic programming based LCS algorithm family is given. The following points are worthy of mentioning.

(1) Based on previous studies and deep analysis of software automation, domain engineering, the PAR method, and the LCS problem, the paper provides a new development method for biological sequence analysis algorithms based on the PAR platform.

(2) For the correlation between LCS and CLCS problems and the similarity of their problem-solving ideas, this paper views them as a problem domain. Using the proposed method, the algorithm function features are extracted to generate the LCS domain feature model, and the dependency relationship among features is constructed. Then, with the support of the PAR method, the domain functional components are designed and implemented in accordance with the LCS domain feature model and feature dependency relationship. Further, the LCS domain component library is built. Finally, the particular LCS algorithm can be generated by assembling components from the component library.

The proposed method enables the improvement of the reliability, understandability, and development efficiency of particular algorithms. It is suitable for the rapid development of different algorithms for the same kind of problem. It is not only applicable to the problem of LCS, but also can be used to solve other problems. In our previous studies, we successfully applied the proposed method to the domain of multiple sequence alignment and pairwise sequence alignment problems.

Next, we will focus on the further application of the method in other bioinformatics research areas, such as gene discovery and recognition, to develop a systematic method and continue improving the component library of the biological sequence analysis algorithm. Future work also includes the development of publicly accessible web services for our method.

## Acknowledgment

This work was supported by the National Natural Science Foundation of China (No. 62062039) and Natural Science Foundation of Jiangxi Province (Nos. 20202BAB202024 and 20212BAB202017).

## References

[1] Y. Zhao, R. Gu, and S. Du, Current status and development trend of bioinformatics research, *Journal of Medical Informatics*, vol. 33, no. 5, pp. 2–6, 2012.

[2] S. Liu, Y. P. Wang, W. N. Tong, and S. W. Wei, A fast and memory efficient MLCS algorithm by character merging for DNA sequences alignment, *Bioinformatics*, vol. 36, no. 4, pp. 1066–1073, 2020.

[3] H. Shi and X. Zhang, Component-based design and

assembly of heuristic multiple sequence alignment algorithms, *Front. Genet*, vol. 11, p. 105, 2020.

[4] S. Liu, Design and implementation of first descending squad based flipping sorting algorithm, Master dissertation, School of Computer Science and Technology, Shandong University, Jinan, China, 2006.

[5] M. D. Cao, S. H. Nguyen, D. Ganesamoorthy, A. G. Elliott, M. Cooper, and L. J. M. Coin, Scaffolding and completing genome assemblies in real-time with nanopore sequencing, *Nature Communications*, vol. 8, p. 14515, 2017.

[6] R. A. Wagner and M. J. Fischer, The string-to-string correction problem, *Journal of ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[7] D. S. Hirschberg, Algorithms for the longest common subsequence problem, *Journal of the ACM*, vol. 24, no. 4, pp. 664–675, 1977.

[8] Y. Li, Y. Wang, Z. Zhang, Y. Wang, D. Ma, and J. Huang, A novel fast and memory efficient parallel MLCS algorithm for long and large-scale sequences alignments, in *Proc. 2016 IEEE 32$^{nd}$ International Conference on Data Engineering*, Helsinki, Finland, 2016, pp. 1170–1181.

[9] Y. Chen, A. Wan, and W. Liu, A fast parallel algorithm for finding the longest common sequence of multiple biosequences, *BMC Bioinformatics*, vol. 7, no. 4, p. S4, 2006.

[10] Z. Peng and Y. Wang, A novel efficient graph model for the multiple longest common subsequences (MLCS) problem, *Frontiers in Genetics*, vol. 8, p. 104, 2017.

[11] N. Nakatsu, Y. Kambayashi, and S. Yajima, A longest common subsequence algorithm suitable for similar text strings, *Acta Informatica*, vol. 18, no. 2, pp. 171–179, 1982.

[12] E. W. Myers and W. Miller, Optimal alignments in linear space, *Computer Applications in the Biosciences: CABIOS*, vol. 4, no. 1, pp. 11–17, 1988.

[13] Y. -T. Tsai, The constrained longest common subsequence problem, *Information Processing Letters*, vol. 88, pp. 173–176, 2003.

[14] C. L. Peng, An approach for solving the constrained longestcommon subsequence problem, http://etd.lib.nsysu.edu.tw/ETD-db/ETD-search/search, 2003.

[15] A. N. Arslan and Ö. Eğecioğlu, Algorithms for the constrained longest common subsequence problems, *International Journal of Foundations of Computer Science*, vol. 16, no. 6, pp. 1099–1109, 2005.

[16] Y. C. Chen and K. M. Chao, On the generalized constrained longest common subsequence problems, *Journal of Combinatorial Optimization*, vol. 21, no. 3, pp. 383–392, 2011.

[17] J. Xue, PAR method and its supporting platform, in *Proc. 1st Asian Working Conference on Verified Software* (*AWCVS 2006*), Macao, China, 2006, pp. 29–31.

[18] J. Xue, Y. Zheng, Q. Hu, Z. You, W. Xie, and Z. Cheng, PAR: A practicable formal method and its supporting platform, in *Proc. 20$^{th}$ International Conference on Formal Engineering Methods* (*ICFEM*), Gold Coast, Australia, 2018, pp. 70–86.

[19] H. Mili, F. Mili, and A. Mili, Reusing software: Issues and research directions, *IEEE Trans. on Software Engineering*, vol. 21, no. 6, pp. 528–562, 1995.

[20] E. A. Karlsson, *Software Reuse: A Holistic Approach*. New York, NY, USA: John Wiley and Sons Ltd, 1995.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. Rep. CMU/SEI-90-TR-21, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

**Haipeng Shi** received the PhD degree from Jiangxi University of Finance and Economics, Nanchang, China in 2022. She is currently an associate professor at the School of Software, Jiangxi Normal University. She has published about 10 papers in international journals and conferences. Her research interests include algorithm design and software engineering.

**Huan Chen** received the BEng degree from Jiangxi Normal University in 2020. He is currently pursuing the master degree at Jiangxi Normal University. His research interests are software engineering, bioinformatics, and deep learning.

**Jun Wang** received the BEng degree from Hunan Institute of Technology in 2017 and the MEng degree from Jiangxi Normal University in 2021. His research interests include bioinformatics algorithms and formal methods.

**Qinghong Yang** received the MEng degree from Jiangxi Normal University in 2003. She is currently a professor at the School of Computer and Information Engineering, Jiangxi Normal University and a professional member of China Computer Federation. She has published over 30 research papers in international journals and conferences. Her research interests include software formal methods and intelligent recommendation systems.

**Haihe Shi** received the PhD degree from Institute of Software, Chinese Academy of Sciences, China in 2012. She is currently a professor at the School of Computer and Information Engineering, Jiangxi Normal University, an expert in China Academic Degrees and Graduate Education Development Center, and a professional member of China Computer Federation. She has published three monographs and more than 40 research papers in international journals and conferences. Her research interests include bioinformatics, big data analysis, software formal method, and generative programming.