计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20180835
56(9): 1907-1917, 2019

基于动态规划的双序列比对算法构件设计与实现

石海鹤　　周卫星
(江西师范大学计算机信息工程学院　南昌　330022)
(haiheshi@jxnu.edu.cn)

Design and Implementation of Pairwise Sequence Alignment Algorithm Components Based on Dynamic Programming

Shi Haihe and Zhou Weixing
(*School of Computer and Information Engineering, Jiangxi Normal University, Nanchang* 330022)

Abstract　Pairwise sequence alignment algorithm is a key algorithm in bioinformatics, and it is widely used in sequence similarity analysis and genomic sequence database searching. The existing study mainly focuses on the optimization and use of relative alignment algorithms for specific application problems. To some extent, those studies lack a high-level algorithm framework that not only has led to the redundancy of the sequence alignment algorithms and the possible errors caused by the artificial selection algorithm, but also made the structure of algorithm difficult to be understood effectively. Through in-depth analysis of the dynamic programming-based pairwise sequence alignment algorithms domain(DPPSAA), a domain feature model and the corresponding algorithm component interactive model have been established, a DPPSAA component library has been formally implemented by the PAR platform, and a concrete algorithm has been assembled, thus the reliability of the algorithm for formal assembly is guaranteed, moreover a valuable reference for the application of sequence similarity analysis algorithms is provided. Finally, the C++ program generation system of PAR platform is used to transform the assembly alignment algorithm into C++ program and the running results show that the dynamic programming-based pairwise sequence alignment algorithm component library has certain practicability.

Key words　pairwise sequence alignment algorithm; dynamic programming; feature model; component interactive model; PAR platform

摘　要　双序列比对算法是生物信息学中的一个关键算法,广泛应用于序列相似性分析以及基因组序列数据库搜索.现有研究主要针对特定应用问题优化和使用相对应比对算法,缺乏高抽象层算法框架的细致研究,在一定程度上导致了序列比对算法的冗余性以及人为选择算法可能造成的误差等问题,也使得人们难以有效地了解算法结构.通过深入分析基于动态规划的双序列比对算法(dynamic programming-based pairwise sequence alignment algorithm, DPPSAA)领域,在建立该算法领域的特征模型以及对应算法构件交互模型基础上,利用 PAR 平台形式化实现双序列比对算法构件库,并装配生成具体算法,保证了形式化装配算法的可靠性,为序列相似性分析算法应用提供了一条有价值的参考途径.最后,利用

收稿日期:2018-12-18;**修回日期**:2019-02-25
基金项目:国家自然科学基金项目(61662035,61762049,61862033)
This work was supported by the National Natural Science Foundation of China (61662035, 61762049, 61862033).
通信作者:石海鹤(haiheshi@jxnu.edu.cn)

PAR 平台C++程序生成系统将组装的比对算法转换为C++程序,运行结果表明 DPPSAA 算法构件库具有一定的实用性.

关键词 双序列比对算法;动态规划;特征模型;构件交互模型;PAR 平台

中图法分类号 TP301.6

序列比对是一种通过排列基因组序列来识别序列相似性区域,从而获得待比对序列之间的功能、结构或进化关系的技术.随着人类基因组计划的实施,测序技术的发展产生了大量的有关生物分子的原始序列数据,例如,Illumina HiSeqX Ten 在 3 天之内可以产生大约 30 亿个 2×150 bp 的双端测序数据[1].面对如此丰富的基因组序列数据,如何高效处理和分析这些丰富的基因组序列数据,如比较两序列之间的相似区域和保守性位点,寻求序列同源结构,揭示生物遗传、变异和进化关系等,成为了序列比对算法研究的主要动力之一.

研究表明,计算机的微处理性能和存储设备容量平均每 18～24 个月增长 1 倍,而基因组测序数据则平均 4～5 月就增长 1 倍,这成为了高性能计算发展史中一个前所未有的挑战,因此序列比对算法的时空开销和精准度成为生物序列比对过程中的关键因素之一.由于序列比对属于 NP-hard 问题,而运筹学中动态规划策略多用于解决多阶段决策过程最优化问题,因此,动态规划策略被广泛应用于序列比对算法之中.这里我们主要针对基于动态规划的双序列比对算法(dynamic programming-based pairwise sequence alignment algorithm, DPPSAA)领域开展研究.

在双序列比对算法中最为经典的确定性双序列比对算法有基于全局比对的 NW 算法[2](Needleman-Wunsch algorithm)、基于局部比对的 SW 算法[3](Smith-Waterman algorithm)和准全局比对算法.在后续的研究中,发展出了基于以上算法进行优化[4-6]或作为主要比对策略[7-9]的系列算法.

目前,比对算法的研究大部分集中于序列相似性分析领域中的特定问题[10-13]或者特定算法优化[14-16],而较少面向于整个问题域,难以得到一个具有更高抽象层次且适用于整个序列相似性分析领域的算法构件库,在一定程度上导致了序列比对算法的冗余以及人为选择算法可能造成的误差等问题,也使得人们难以有效地了解算法结构,无法保证算法的正确使用,在一定程度上降低了序列相似性分析结果的准确性.由于现有算法的专用性和低抽象性,不仅导致研究人员需要花费大量时间去学习和使用该类算法,降低了算法的可维护性和复用性,而且难以定位和解决算法产生的错误,加重了序列相似性分析的负担.

通过深入分析 DPPSAA 领域,本文设计和实现了该领域抽象泛型算法构件库,提高了序列相似性分析领域算法可靠性和开发效率.首先对 DPPSAA 领域进行特征分析,提取出其中的通用和可变特征以及它们之间的约束与依赖关系,建立了一致的领域构件模型,基于此设计了算法构件交互模型,进一步利用新型高可靠软件开发平台 PAR[17-19]中高抽象程序设计语言 Apla(abstract programming language)进行形式化实现,形成了一个高抽象 DPPSAA 构件库,以期自动或半自动装配构件产生特定领域序列比对算法,甚至于装配出更为高效的新型序列比对算法.

1 比对算法的建模过程

1.1 领域分析

面向特征的领域分析[20](feature oriented domain analysis, FODA)是由软件工程研究所(Software Engineering Institute, SEI)于 1990 年首次提出的一种领域分析方法,主要包含上下文分析以及特征建模 2 个阶段[21].其中上下文分析主要由界定待研究领域需求范围以及领域的输出与输入构成,该阶段的成功实施能够保证特征建模过程中被选取特征的有效性和可靠性.特征建模阶段则包含了特征选取与建立、特征关系描述以及特征模型建立等过程,在该阶段需要对被研究领域进行分析,充分了解其功能性特征和非功能性特征,并找出特征之间的约束和依赖关系以及优先级等附加信息,进而获得该领域的主次要特征,建立一种具有更高抽象层次的特征模型或者构件框架.

特征建模阶段主要包含以下 4 种特征:强制特征(mandatory)、可变特征(optional)、OR 特征、XOR 特征.强制特征表示领域内的所有实例都必须

包含的特征,可变特征表示领域中实例的一个可选特征,XOR 特征表示领域中实例有且只能选择一组 XOR 特征中的一个特征,OR 特征表示领域中实例至少包含了一组 OR 特征中的一个特征,特征表示如图 1 所示.特征建模清晰地刻画了特征模型构建过程中所需的各类特征,是识别和捕捉差异性和可变性的关键技术.

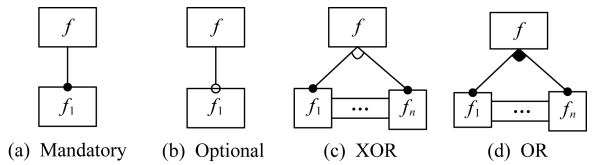

Fig. 1 Feature graph

图 1 特征图

图 1 中强制特征被末端为实心圆的边所指向,可变特征被末端为空心圆的边所指向,XOR 特征被一组用弧连接的边所指向,OR 特征被一组用实心弧连接的边所指向.这 4 种特征是特征建模中的主要属性,也是特征模型的重要组成部分,代表了领域中的通用和可变属性,同时它们之间的约束以及依赖关系则需通过特征内在的层次结构关系、领域业务逻辑设计的约束关系以及运行时依赖关系来表示.

1.2 特征建模流程

以 FODA 为基础,我们在其后增添了领域实现阶段,即将建立的特征作为构件进行构件交互模式设计和形式化实现,建立抽象构件库.需要注意的是,由于建立的构件之间的交互性,程序人员使用 C,C++或 Java 等低抽象层次高级编程语言进行代码编程时,不仅增加了特征之间的关联性与复杂性,而且难以让使用者对整个构件库有具体和全面的了解.因此需要使用一种具有高抽象性的编程语言来实现这些构件,以至不用在意算法构件的具体实现细节,而能够清晰地了解这些算法构件的功能以及它们之间的交互关系.例如在本文中主要利用 Apla 语言来实现 DPPSAA 领域构件.

基于以上介绍,对具体领域进行分析时,领域构件库建立流程图如图 2 所示:

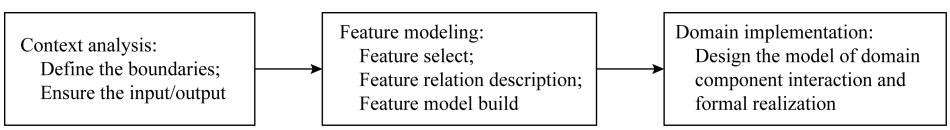

Fig. 2 Flow chart of domain analysis

图 2 领域分析流程图

1.3 DPPSAA 领域特征建模

特征模型描述了领域内实例的通用和可变特征以及它们之间的约束与依赖关系,是在特征建模过程中构建的,一般由描述特征层次关系的特征图和相关的文本描述组成.其中特征图通常使用具有不同层次的树状结构来表示,在该结构中包含了 1 组节点、1 组节点之间的定向边以及两边之间的关系描述.其中 1 个节点代表一个特征,具有唯一的标识符;节点之间的定向边将特征连结成树结构;两边之间的关系描述则代表特征之间的关系,是对特征节点的划分.同时特征模型中的文本描述则表示了特征的语义描述、原理以及约束和默认依赖规则等信息.

因此,针对基于动态规划的双序列比对算法领域,建模过程有 3 方面:

1) 上下文分析.该算法领域的范围被限制为一种在生物序列相似性分析领域中以动态规划为主要罚分策略、双序列比对为主要比对方式的算法形式.

2) 特征建模.我们利用特征建模方法[22]对该领域进行特征建模,即从分析领域中的服务(service)、功能(function)以及行为(behavior)特点入手构建特征模型.比对操作服务是该领域中的核心服务,通过控制序列比对过程中的比对方式和各算法特征之间的执行优先级以及组合方式来实现使用者定义的序列比对算法.通过分析双序列比对算法中的一些主要执行步骤,可以将该比对操作服务进一步划分为检查序列合法性、得分矩阵操作、动态规划算法方式选择、记住得分来源、回溯和比对结果输出等功能,同时得分矩阵初始化、动态规划方式选择以及检查序列合法性作为 3 个必选的功能.在上述分析的基础上,输出方式是比对结果输出的显著行为特点,包括比对序列输出和比对得分输出两种主要行为特点,且二者属于 OR 特征.动态规划方式可以分为全局动态规划方式、准全局动态规划方式、局部动态规划方式 3 种,三者为 XOR 特征.我们根据上述分析对该领域建立了特征模型,如图 3 所示:

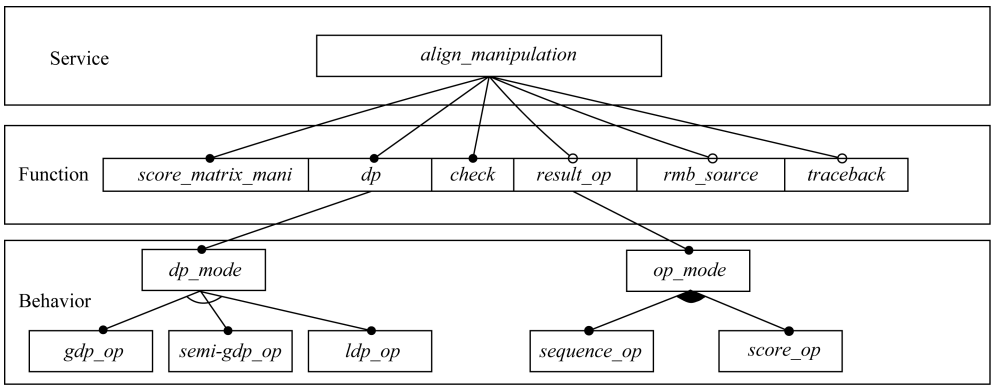

Fig. 3 DPPSAA feature model

图 3 DPPSAA 特征模型

3) 领域实现.该过程针对已建立的特征模型进行 DPPSAA 领域算法构件交互设计并使用 Apla 语言形式化实现.

2 比对算法构件的设计与实现

2.1 DPPSAA 领域算法构件交互设计

不同构件通过交互实现完整的算法构件库,而算法构件的交互则需要由其包含的特征之间的约束以及依赖关系来体现.因此针对 1.3 节建立的特征模型,我们对 DPPSAA 领域的算法构件交互模型进行了设计.

通过分析整个 DPPSAA 领域得知,算法主要包括 3 个主要变化过程特征:得分矩阵操作、动态规划算法方式以及比对结果输出.因此,我们将特征模型中的这些特征以及检查序列合法性作为主要构件,其他特征以及相关数据结构作为辅助构件,并根据其优先级建立了构件间的交互模型,如图 4 所示:

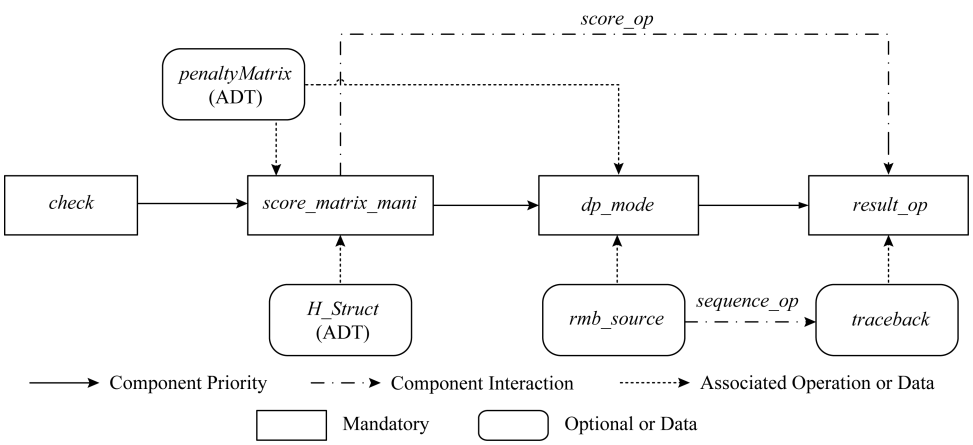

Fig. 4 Algorithm components interaction model

图 4 算法构件交互模型

图 4 中实线所连接的节点表示为 DPPSAA 领域中所必须含有的基本构件,即对应于特征模型中的 4 个强制特征,实线箭头所代表的方向表示 4 个构件的执行优先级为由高到低;点连线箭头表明在算法执行过程中 2 个构件之间的交互,如在图 4 中,当比对结果输出构件选择得分输出时,则需要使用得分矩阵操作构件中的获取元素得分操作;虚线箭头则代表在算法组装过程中所需的数据、结构以及关联操作等,如在得分矩阵操作构件中需要使用 2 个抽象数据类型(abstract data type, ADT):罚分模型 ADT 和得分矩阵元素 ADT,且罚分模型 ADT 同时作用于动态规划算法方式构件等.

2.2 Apla 形式化实现

Apla 语言可以直接使用抽象数据类型和抽象过程编写程序,因此能够更抽象地描述算法问题,并易于对其进行正确性验证,保证了程序正确性和可靠性.

PAR 方法和 PAR 平台包含循环不变式的新定义和新的开发策略、统一的算法程序设计方法、新的算法表示方法、自定义算法设计语言和抽象程序设计语言等关键技术.它集成涵盖了泛型、生成式、模型驱动和构件组装等新型软件开发技术,其系列程序自动生成系统可将一个正确的 Apla 程序自动转换成 C++,Java,Delphi 等高级语言程序.因此,本文利用该语言形式化实现并建立了 DPPSAA 领域算法构件库,不仅可高抽象表示算法程序的功能特性与非功能特性,而且能够直观地展示各算法构件之间的约束以及依赖关系.这样既减少了各构件之间的干扰,降低了算法程序的复杂性,提高了算法构件安全性,又消除了传统算法设计方式中算法与数据难以分离的问题,提高了构件装配产生算法的可复用性和可维护性,同时使用该算法构件库时,我们只需关注算法功能即可,而不用在意具体构件实现细节,从而提高了算法设计效率.

下面展示了对 DPPSAA 领域构件的 Apla 程序实现.

1) 罚分模型以及得分矩阵元素结构设计

首先我们将罚分模型设置为一个 ADT,这里我们为解释方便,只使用固定空位罚分策略,与扩展罚分时操作类似.其中罚分模型的 ADT 定义为

```
define ADT penaltyMatrix(sometype elem);
  match:integer;
  mismatch:integer;
  space:integer;
enddef.
```

其中 sometype 为 Apla 语言中的关键字,用来定义类型变量.其中的 *match*,*mismatch*,*space* 分别表示罚分模型中的匹配罚分值、错配罚分值以及空位罚分值.

在得分矩阵中,由于序列比对结果输出时需要进行回溯操作,因此我们将得分矩阵中的元素定义为一个 ADT,命名为 *H_Struct*,该 ADT 中包含了整型变量 *value* 和 boolean 型的数组 *dp_direct*,前者表示 2 个字符比对得分值,后者表示该得分值的来源(其中 true 代表对应得分来源为真,false 代表为假).并且数组内的元素含义如图 5 所示,4 个方框分别代表得分矩阵中的 4 个元素,3 个箭头分别表示(i,j)中的得分来源为上、左和对角元素,分别用数组 *dp_direct* 下标 0,1,2 对应的数组元素来表示.

该 ADT 定义为

```
define ADT H_Struct(sometype elem);
```

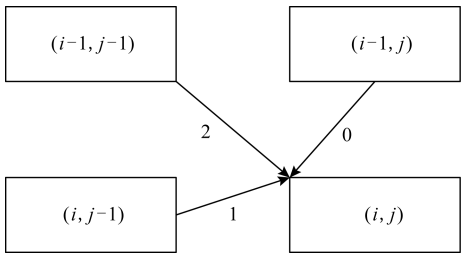

Fig. 5 Score source of score matrix element

图 5 得分矩阵元素的得分来源

```
  value:integer;
  dp_direct:array[0:3,boolean];
enddef.
```

2) 得分矩阵操作

该构件被定义为一个 ADT 类型,因为在不同的 DPPSAA 中得分矩阵初始化方式不同,因此在该 ADT 内部包含了一个泛型子程序 *Memory_Score_of_Matrix*,并将 *Init_score_matrix* 方法作为它的泛型参数,使得泛型子程序可以支持实例化具有不同得分矩阵初始化方式的比对算法,即当使用不同的方法参数实例化 *Init_score_matrix* 时,将返回不同的得分矩阵.同时在该 ADT 中还包含了一些常用的得分矩阵操作,如求矩阵最大得分、矩阵元素取值操作和赋值操作等.

```
define ADT score_matrix_mani (sometype
  elem);
  procedure apply_memory(length_s:integer,
    length_t:integer);
  procedure Memory_Score_of_Matrix (proc
    Init_score_matrix());
  function Max_score_of_Matrix:integer;
  function the_Last_element_score:integer;
  function get_value(i:integer;j:integer):
    integer;
  procedure set_value(i:integer;j:integer;
    score:integer);
enddef.
```

其中该 ADT 类型名为 *score_matrix_mani*,且带有一个类型参数 *elem*;*apply_memory* 泛型子程序的作用是根据整型变量 *length_s* 和 *length_t* 值为 *score_matrix_mani* 动态分配内存空间;函数 *Max_score_of_Matrix* 和 *the_Last_element_score* 分别表示获取 *score_matrix_mani* 中得分最大值以及最后一个元素的得分值;*get_value* 以及 *set_value* 分别为获取和设置 *score_matrix_mani* 中元素得分

值,$(i,j)(0\leqslant i\leqslant length\_s, 0\leqslant j\leqslant length\_t$ 表示对应元素的下标,$length\_s$ 和 $length\_t$ 分别表示两比对序列的长度.

3) 动态规划算法方式选择

该构件也被定义为泛型 ADT,可以支持不同的序列比对算法所使用的动态规划得分模式,得分模式的变化主要依靠泛型子程序 dp_align_score 来实现,同时该 ADT 类型中还包括求 2 字符的最大比对得分操作 $max\_score\_of\_char$,函数中的 3 个参数分别表示不同来源的罚分值.

```
define ADT dp_mode(sometype elemMatrix);
  function max_score_of_char(up_score:
    integer;left_score:integer;diag_score:
    integer):integer;
  procedure dp_align_scores(s:String;t:
    String;sM:penaltyMatrix;proc set_and
    _remember(sometype elemMatrix;length
    _s:integer;length_t:integer));
enddef.
```

这里 $elemMatrix$ 是一个类型参数;函数 $set\_and\_remember$ 是泛型子程序 dp_align_score 的泛型参数,该函数的功能为记录得分矩阵中各元素的得分值以及得分来源,可以被用于各种动态规划罚分模型的实例化.

4) 检查序列合法性

检查序列是否属于字符集{A,T,C,G}(默认为 DNA 序列,如果为其他序列,加入对应表示字符即可,如果为 RNA 序列,则将 G 换为 U 即可).

```
procedure check(s,t:String).
```

其中 s 和 t 都为字符串类型.

5) 记住得分来源

该构件表示记住(i,j)处得分的来源,即将(i,j)处中对应的方向标志赋值为 true.其为回溯阶段输出序列比对结果提供支持.

```
procedure rmb_source(i:integer;j:integer).
```

6) 回溯

该构件的 Apla 语言定义为

```
procedure traceback(proc print_align();proc
  print_extrude()=NULL).
```

在回溯过程中,一般由短序列比对区间回溯和两端突出回溯组成,其中短序列比对区间回溯表示从 2 序列匹配的第 1 个序列字符开始,到最后匹配的字符结束位置之间的字符区间序列输出,两端突出回溯则表示从头开始到第 1 个匹配字符的前一个字符区间序列输出或者最后一个匹配字符的下一个字符到最后所有字符序列对应输出.在 $trackback$ 泛型子程序中分别由 $print\_align$ 和 $print\_extrude$ 表示,并且在默认配置下 $print\_extrude$ 为空,即为全局比对.

7) 比对结果输出

这里将该构件定义成一个泛型子程序.

```
procedure op_mode(func finally_score():
  integer;proc traceback(proc print_align();
  proc print_extrude()=NULL)).
```

在此泛型子程序定义中,函数 $finally\_score$ 功能为输出最终比对得分.

8) 比对操作

为了能够实现现有的序列比对算法,需要对上述算法构件进行人工装配,因此将比对操作服务定义为一个泛型子程序,并将各功能作为其参数,使之能够支持装配产生 DPPSAA.

```
procedure align_manipulation(op_mode
  (func finally_score():integer;proc
  traceback(proc print_align();proc print_
  extrude()=NULL));ADT dp_mode(eM:
  elemMatrix);sometype elemMatrix;result:
  boolean;eM:elemMatrix;s:String;t:
  String)).
```

该算法构件 $align\_manipulation$ 主要包含 4 个参数:比对输出构件 op_mode、回溯构件 $traceback$、自定义泛型 ADT dp_mode 以及待比对序列 s 和 t.其中 $elemMatrix$ 表示为一个类型参数,并且后面 4 个变量参数为在主程序中实例化参数所需代入的.这样我们就可以通过手工装配该子程序,以达到实现相应比对算法的目的.

3 NW 算法的装配实现

通过第 2 节的介绍,可以较清晰地了解到整个 DPPSAA 领域构件库的建立过程,下面我们利用上述构件库来装配实现基于全局的双序列比对算法 NW,程序为

```
program para;
const
    /*序列 s,t 的输入;*/
procedure Init_score_matrixNW(sometype
  elemMatrix);                                ①
var
```

```
  i,j:integer;H:elemMatrix;
begin
  foreach(i,j:0≤i,j<length_s,
    length_t:…;);                                    ②
end;
procedure set_and_remember(length_s:
  integer;length_t:integer;sometype
  elemMatrix);                                       ③
var
  i,j:integer;
begin
  foreach(i,j:0≤i,j<length_s,
    length_t:…;);                                    ④
end;
ADT pM:new penaltyMatrix();
ADT struct:new H_struct();
ADT matrix:new score_matrix_mani(struct);
ADT dp_NW:new dp_mode(matrix);
var
  dp_g:dp_NW;
begin
  dp_g.dp_align_score(matrix,set_and_
    remember(sometype elemMatrix;
    length_s:integer;length_t:integer));             ⑤
end;
procedure align_manipulation(sometype
  elemMatrix;ADT dp_mode(eM:elemMatrix);
  op_mode(func finally_score():integer;
  proc traceback(proc print_align();
  proc print_extrude()=NULL);
  result:boolean;eM:elemMatrix;
  s:String;t:String);                                ⑥
begin
  if(result)
    write("The alignment score is:",
      finally_score());
  else
    traceback(print_align,print_extrude);
end;
procedure NW:new align_manipulation(score_
  matrix_mani,dp_g,op_mode(print_
  align));                                           ⑦
begin                                                ⑧
  check(s,t);
  matrix.apply_memory(s.length(),
    t.length());
  matrix.Memory_Score_of_Matrix(Init_
    score_matrixNW(matrix));
  NW(false,matrix,s,t);
end.
```

在上面程序中,过程①为得分矩阵初始化的不同实例化方式;代码块②和④由于得分矩阵初始化过程占用篇幅过大,因此用…表示;过程③则为记录比对得分值以及得分来源的实例化;⑤则表示 NW 算法的动态规划算法的实例化;泛型子程序⑥则表示在步骤⑦实例化 NW 算法对象时内部所要执行的算法构件间的关联操作;⑧以下的代码块为主程序.

4 实验及其结果分析

目前,由于 Apla 语言无法在 PAR 平台上直接运行,本节利用 PAR 平台 C++程序生成系统,将组装 NW 算法过程中所需的 Apla 算法构件代码转换为相对应的 C++代码.

Apla 代码中只包含数据成员的 ADT 算法构件被转换为 C++中的结构体,如 *penaltyMatrix* 以及 *H_Struct* 等,其结果为

```
struct penaltyMatrix
{
  int match;
  int mismatch;
  int space;
};
struct H_Struct
{
  int value;
  bool dp_direct[3];
}.
```

含有数据成员和成员函数的 ADT 则被转换为 C++函数中的类,如 *score_matrix_mani*,*dp_mode* 等,其中大括号内的省略号表示函数体,且 *Sequence* 类将 2 个待比对序列作为其数据成员.其转换的部分结果如图 6 所示.

同时,Apla 代码中的泛型子程序和泛型函数等被转换为 C++中独立的类成员函数,降低构件间的耦合性.其中,主调函数与主调函数泛型参数之间

的关系被转换为C++中的函数指针,即将泛型参数转换为主调函数的指针函数参数,从而具有与Apla程序同样的多态性特征,如 *traceback* 泛型子程序的C++转换结果如图7所示.

可将Apla算法构件代码转换成可执行的C++代码,最后利用转换后的算法构件进行手工装配NW算法(即将服务 *align_manipulation* 转换为主函数)如图8所示,并输出代码结果如图9所示.

```
class Score_matrix_mani : public Sequence
{
public:
    void apply_memory() {...}
    void Memory_Score_of_Matrix( void (Init_score_matrix::*Init_score_matrix_NW)(H_Struct**,penaltyMatrix ,int, int),
        H_Struct** H,penaltyMatrix pM,int length_s, int length_t)    {...}
    const int the_Last_element_score() {...}
    const int Max_score_of_Matrix() {...}
    void set_value( int i, int j, const int score) const {...}
    int get_value( int i, int j) const {...}
    Score_matrix_mani(const std::string& s, const std::string& t) : Sequence(s, t) {...}
    ~Score_matrix_mani() {...}
    void print_Matrix() {...}
};
class dp_mode
{
public:
    int max_score_of_char(int up_score, int left_score, int diag_score) {...}
    void dp_align_score(char s_a, char t_b, int i, int j, penaltyMatrix pM, Score_matrix_mani smm,
        void (set_and_remember::*set_and_remember_NW)(int, int,int,Score_matrix_mani,const bool*,dp_score)) {...}
    void align_and_score(Score_matrix_mani smm, void (set_and_remember::*set_and_remember_NW)(int, int,int,
        Score_matrix_mani,const bool*,dp_score)) {...}
    dp_mode(){...}
    ~dp_mode() {}
};
```

Fig. 6 Result of ADT transformation

图6 ADT转换结果

```
class Op_mode
{
public:
    void op_mode(Score_matrix_mani smm,Traceback tb, const int (Score_matrix_mani::*finally_score)())
};
```

Fig. 7 Result of generic program transformation

图7 泛型子程序转换结果

```
int main()
{
    string s = "ACTAG";
    string t = "ACTTCG";
    Check().check_dna(s,t);
    Score_matrix_mani matrix(s,t);
    matrix.apply_memory();
    matrix.Memory_Score_of_Matrix(&Init_score_matrix::Init_score_matrix1,matrix.get_Matrix(),
        matrix.getPenaltyMatrix(),matrix.get_length_s(),matrix.get_length_t());
    dp_mode dp_NW;
    dp_NW.align_and_score(matrix,&set_and_remember::set_and_remember1);
    matrix.print_Matrix();

    Traceback tb(&Print_align::print_align);
    Op_mode().op_mode(matrix,tb,&Score_matrix_mani::the_Last_element_score);
}
```

Fig. 8 C++ assembly process of NW

图8 NW的C++装配过程

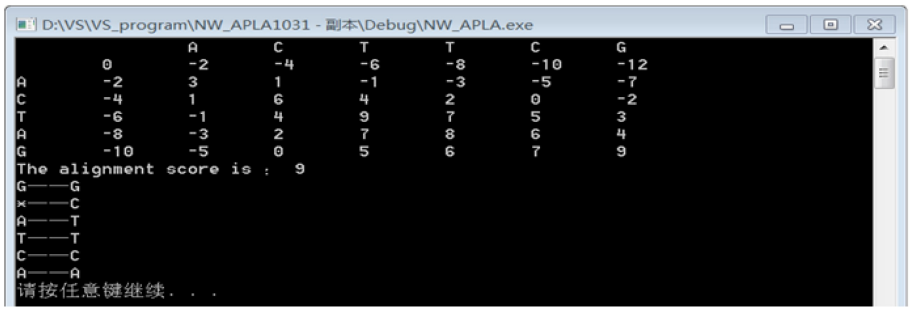

Fig. 9 NW alignment result

图9 NW比对结果

在 PAR 平台的支撑下,我们以半自动的方式将上述 Apla 语言编写的 DPPSAA 算法构件程序转换成 C++代码,并装配形成 NW 算法.经实际运行 NW 算法检测,算法的运行结果与原本 NW 算法结果一致.使用 DPPSAA 领域构件装配形成的具体双序列比对算法,不但提高了装配算法程序的可靠性、执行效率以及可维护性,而且可以根据客户需求进行手工装配形成指定算法,增强了 DPPSAA 算法构件的通用性.

在下一步研究工作中,为实现自动化装配 DPPSAA 领域算法,我们将使用产生式编程的相关方法学来开展工作,为进一步装配生成奠定了基础,可望装配形成一种相较于现有算法具有更高执行效率和更低内存消耗的新型双序列比对算法.

5 相关研究比较

针对序列比对算法研究的不同方面,国内外学者通过使用并融合相关方法学技术开展了较多研究工作,可分为 3 个方面:

1) 基于代数方法的研究.比勒费尔德大学的 Giegerich 等人[23-25]提出了一种面向系统族的基于代数结构的新型动态规划构造算法,通过利用代数数据类型概念将动态规划算法过程形式化定义为 2 个阶段:识别阶段和评价阶段.在具体应用过程中,算法通过迭代进行这 2 个阶段,可为序列比对算法领域内待生成算法提供一种更具一般性的生成范式,简化了算法开发过程,提高了生成算法的正确性和可靠性.

2) 基于有限状态机的研究.宾夕法尼亚大学的 Searls 等人[26]提出了一种基于有限状态机和具有自适应加权的序列比对算法生成方法.该算法利用有限状态机模型形式化表示序列比对过程中的罚分、比对以及动态规划过程,并对比对过程中的不同碱基字符比对结果(匹配、错配以及空位)进行自适应加权,能够快速地生成新型序列比对算法.算法能够支持可视化编程.

3) 基于算法优化的研究.算法优化是指为了提高已有算法处理问题能力而进行研究的过程.文献[27]采用经典的脉动阵列结构的并行加速器体系结构来提高序列加载、比对和结果收集效率的数据通路,并且利用 Hash 索引优化技术过滤比对过程中的非相似行序列,增强了序列比对算法的性能.为了提高序列比对算法的准确度,文献[28-29]等利用改进的隐马尔可夫模型对序列比对算法进行优化.

6 结束语

序列比对作为生物序列分析中的关键问题,其算法及应用研究受到广泛关注,然而,尚未有工作将其视为一个专门领域从高抽象层次开展研究,从而提高算法可靠性和开发效率、降低算法次优解及误差等问题出现的概率.我们采用特征建模,分析和提取出基于动态规划双序列比对算法领域的通用以及可变特征,并利用高抽象语言进行形式化实现,以期能够以自动或者半自动方式进行形式化组装生成特定问题的求解算法,从而降低人工选择算法进行序列相似性分析的错误发生率以及时间开销,提高算法执行效率,甚至于装配出一种更为高效的基于动态规划的新型序列比对算法.

本文首先提出了一种领域建模过程,包括上下文分析、特征建模以及领域实现 3 个主要阶段;其次通过分析 DPPSAA 领域的一般执行过程,抽取出算法的通用以及可变特征,并建立领域特征模型;然后对建立的构件特征模型进行构件交互设计,同时利用 Apla 语言形式化实现这些算法构件,并通过手工装配生成了 NW 算法实例;最后利用 PAR 平台 C++程序生成系统将 Apla 程序转换为 C++可运行程序,转换结果以及算法运行结果展示了其具有一定的实用性.由于 Apla 的高抽象性,建立的一系列泛型构件,如 *dp_mode*,*result_op* 等,保证了构件装配后的算法多样性,也能够较好地展示出算法特征之间的联系.

我们对整个 DPPSAA 领域进行了精确分析,对其各功能特性与非功能特性有充分的了解,同时在算法设计过程中建立了特征模型以及构件交互设计,从而有利于算法构件库的学习和使用;另外,使用高抽象语言 Apla 形式化实现算法构件,不仅易于对 Apla 算法程序进行正确性验证,保证算法构件库的可靠性,而且在使用过程中可利用泛型编程机制快速发现和定位算法中的错误,提高算法构件库的鲁棒性.

本文研究 DPPSAA 领域的主要方法学思想和成果,不仅适用于 DNA 序列比对算法,理论上对于一些其他的生物序列分析算法领域也具有参考价值和实用价值,例如基因组装过程中的基于 de bruijn graph 结构的组装算法[30-32].下一步的研究尚需扩充本文结果在生物序列分析算法领域的应用范围,同时基于 PAR 平台实现算法构件库的自动或半自动装配.

参 考 文 献

[1] Illumina. HiSeqX instrument performance parameters [EB/OL]. 2016 [2018-09-26]. https://www.illumina.com/systems/sequencing-platforms/hiseq-x/specifications.html

[2] Needleman S B, Wunsch C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins [J]. Journal of Molecular Biology, 1970, 48(3): 443-453

[3] Smith T F, Waterman M S. Identification of common molecular subsequences [J]. Journal of Molecular Biology, 1981, 147(1): 195-197

[4] Hirschberg D S. A linear space algorithm for computing maximal common subsequences [J]. Communications of the ACM, 1975, 18(18): 341-343

[5] Ukkonen E. Algorithms for approximate string matching [J]. Information & Control, 1985, 64(1/2/3): 100-118

[6] Abbasi M, Paquete L, Liefooghe A, et al. Improvements on bicriteria pairwise sequence alignment: Algorithms and applications [J]. Bioinformatics, 2013, 29(8): 996-1003

[7] Train C M, Glover N M, Gonnet G H, et al. Orthologous matrix (OMA) algorithm 2.0: More robust to asymmetric evolutionary rates and more scalable hierarchical orthologous group inference [J]. Bioinformatics, 2017, 33(14): i75-i82

[8] Altenhoff A M, Škunca N, Glover N, et al. The OMA orthology database in 2015: Function predictions, better plant support, synteny view and other improvements [J]. Nucleic Acids Research, 2015, 43(D1): 240-249

[9] Šoši M, Šiki M. Edlib: A C/C++ library for fast, exact sequence alignment using edit distance [J]. Bioinformatics, 2017, 33(9): 1394-1395

[10] Chattopadhyay A K, Nasiev D, Flower D R. A statistical physics perspective on alignment-independent protein sequence comparison [J]. Bioinformatics, 2015, 31(15): 2469-2674

[11] Cattaneo G, Petrillo U F, Giancarlo R, et al. Alignment-free sequence comparison over Hadoop for computational biology [C] //Proc of the 44th Int Conf on Parallel Processing Workshops. Piscataway, NJ: IEEE, 2015: 184-192

[12] Huo Hongwei, Sun Zhigang, Li Shuangjiang, et al. CS2A: A compressed suffix array-based method for short read alignment [C] //Proc of the 2016 IEEE Data Compression Conf. Piscataway, NJ: IEEE, 2016: 271-278

[13] Isa M N, Murad S A Z, Ismail R C, et al. An efficient processing element architecture for pairwise sequence alignment [C] // Proc of the 33rd Int Conf on Electronic Design. Piscataway, NJ: IEEE, 2015: 461-464

[14] Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations [J]. Bioinformatics, 2007, 23(2): 156-161

[15] Houtgast E, Sima V M, Al-Ars Z. High performance streaming Smith-Waterman implementation with implicit synchronization on Intel FPGA using OpenCL [C] //Proc of the 2018 IEEE Int Conf on Bioinformatics and Bioengineering. Piscataway, NJ: IEEE, 2018: 492-496

[16] Junid S A M A, Idros M F M, Razak A H A, et al. Parallel processing cell score design of linear gap penalty Smith-Waterman algorithm [C] //Proc of the 13th IEEE Int Colloquium on Signal Processing & ITS Applications. Piscataway, NJ: IEEE, 2017: 299-302

[17] Xue Jinyun. A unified approach for developing efficient algorithmic programs [J]. Journal of Computer Science & Technology, 1997, 12(4): 314-329

[18] Wang Changjing, Xue Jinyun. Formal derivation of a generic algorithmic program for solving a class of extremum problems [C] //Proc of the 10th ACIS Int Conf on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing. Piscataway, NJ: IEEE, 2009: 100-105

[19] Xue Jinyun. Genericity in PAR platform [C] //Proc of the 5th Int Workshop on Structured Object-Oriented Formal Language and Method. Berlin: Springer, 2015: 3-14

[20] Kang K C, Cohen S G, Hess J A, et al. Feature-oriented domain analysis (FODA) feasibility study, CMU/SEI-90-TR-21 [R]. Pittsburgh: Carnegie-Mellon University, 1990

[21] Czarnecki K, Eisenecker U W. Generative Programming: Methods, Tools, and Applications [M]. New York: ACM, 2000

[22] Zhang Wei, Mei Hong. A feature-oriented domain model and its modeling process [J]. Journal of Software, 2003, 14(8): 1345-1356 (in Chinese)
(张伟, 梅宏. 一种面向特征的领域模型及其建模过程[J]. 软件学报, 2003, 14(8): 1345-1356)

[23] Giegerich R. A systematic approach to dynamic programming in bioinformatics [J]. Bioinformatics, 2000, 16(8): 665-677

[24] Steffen P, Giegerich R, Giraud M. GPU parallelization of algebraic dynamic programming [C] //Proc of the 8th Int Conf on Parallel Processing and Applied Mathematics. Berlin: Springer, 2009: 290-299

[25] Saule C, Giegerich R. Pareto optimization in algebraic dynamic programming [J]. Algorithms for Molecular Biology, 2015, 10(1): 22

[26] Searls D B, Murphy K P. Automata-theoretic models of mutation and alignment [C] //Proc of the 3rd Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAI Press, 1995: 341-349

[27] Wang Wendi, Tang Wen, Duan Bo, et al. Parallel accelerator design for high-throughput DNA sequence alignment with Hash-index [J]. Journal of Computer Research and Development, 2013, 50(11): 2463-2471 (in Chinese)
(王文迪, 汤文, 段勃, 等. 基于 Hash 索引的高通量基因序列比对并行加速技术研究[J]. 计算机研究与发展, 2013, 50(11): 2463-2471)

[28] Ge Hongwei, Liang Yanchun. A multiple sequence alignment algorithm based on a hidden Markov model and immune particle swarm optimization [J]. Journal of Computer Research and Development, 2006, 43(8): 756-765 (in Chinese)
(葛宏伟, 梁艳春. 基于隐马尔可夫模型和免疫粒子群优化的多序列比对算法[J]. 计算机研究与发展, 2006, 43(8): 756-765)

[29] Orlando G, Raimondi D, Khan T, et al. SVM-dependent pairwise HMM: An application to protein pairwise alignments [J]. Bioinformatics, 2017, 33(24): 3902-3908

[30] Li Ruiqiang, Zhu Hongmei, Ruan Jue, et al. De novo assembly of human genomes with massively parallel short read sequencing [J]. Genome Research, 2010, 20(2): 265-272

[31] Peng Yu, Leung H C M, Yiu S M, et al. IDBA-UD: A de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth [J]. Bioinformatics, 2012, 28(11): 1420-1428

[32] Li Min, Liao Zhongxiang, He Yiming, et al. ISEA: Iterative seed-extension algorithm for de novo assembly using paired-end information and insert size distribution [J]. IEEE/ACM Transactions on Computational Biology & Bioinformatics, 2017, 14(4): 916-925

Shi Haihe, born in 1979. PhD, professor. Member of CCF. Her main research interests include bioinformatics, formal method and software engineering.

Zhou Weixing, born in 1994. Master. Student member of CCF. His main research interests include bioinformatics, formal method.