燕山大學
YANSHAN UNIVERSITY

# 硕 士 学 位 论 文

MASTER'S DISSERTATION

论文题目 不确定字符串的匹配方法研究

作者姓名 王丁

学科专业 计算机科学与技术

指导教师 王璿 副教授

**2018 年 5 月**

中图分类号：TP311　　学校代码：10216
UDC：654　　密级：公开

工学硕士学位论文

# 不确定字符串的匹配方法研究

**硕士研究生：**王丁
**导　　师：**王璿 副教授
**申请学位：**工学硕士
**学科专业：**计算机科学与技术
**所在单位：**信息科学与工程学院
**答辩日期：**2018 年 5 月
**授予学位单位：**燕山大学

A Dissertation in Computer Science and Technology

# RESEARCH ON MATCHING METHOD OF UNCERTAIN STRING

by Wang Ding

Supervisor: Associate Professor Wang Xuan

**Yanshan University**

May, 2018

# 燕山大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《不确定字符串的匹配方法研究》，是本人在导师指导下，在燕山大学攻读硕士学位期间独立进行研究工作所取得的成果。论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字：　　日期：　　年　月　日

# 燕山大学硕士学位论文使用授权书

《不确定字符串的匹配方法研究》系本人在燕山大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归燕山大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解燕山大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅。本人授权燕山大学，可以采用影印、缩印或其它复制手段保存论文，可以公布论文的全部或部分内容。

保密□，在　　年解密后适用本授权书。

本学位论文属于

不保密□。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

# 摘 要

给定一个文本串 *T* 和模式串 *P*，字符串匹配就是从一个 *T* 中找到所有和 *P* 相同的子串。字符串匹配的应用涉及到生物信息学、文本编辑、模式识别、自然语言处理和搜索引擎等领域。随着互联网技术的发展，数据来源的差异性导致了不确定数据的产生。本文重点研究生物信息学领域的不确定字符串匹配问题，研究内容如下。

首先，通过对现有方法进行深入分析，发现了现有方法存在索引构建时间长，索引规模大的问题。

其次，针对生物信息学领域中字符种类较少的特点，提出了空间代价为字符种类乘字符串长度的索引 USAL。在 USAL 上提出了积极字符和消极字符的概念，基于积极字符并结合贪心思想和范围最值查询 RMQ 提出了一种高效的字符串匹配算法 GUSM。 USAL 在每个位置记录了每种字符的出现概率，字符串匹配时，根据给定的查询概率阈值用 RMQ 以 $O(1)$时间返回当前划分区间中概率最大的积极字符位置，然后对该位置进行概率阈值的验证，最后返回正确的结果。为了提高积极字符位置的过滤效果，提出了基于最小概率字符和随机选择字符的过滤策略。

再次，为了进一步增强字符串匹配阶段的过滤效果，结合 Bitmap 思想提出了一种小规模位图索引 BI。BI 可以对 USAL 上的位置概率进行位映射，基于 BI 提出了一种高效字符串匹配算法 BUSM，字符串匹配时，将子串匹配操作转化为 BI 上的位操作，然后验证候选结果集合，最后返回正确的结果。为了提高 BUSM 在概率阈值验证阶段的效率，提出了多维位图索引 MBI，通过减少候选结果集合的规模，从而提高了算法执行效率。

最后，对于多个不同特征的真实数据集和人工合成数据集进行实验，验证了本文提出算法的高效性。

**关键词：**不确定字符串；索引构建；过滤策略；子串匹配

# Abstract

With a given text *T* and a pattern *P*, string matching refers to the procedure designed to find all the substrings identical to *P* in *T*. The application of string matching involves various subjects including bioinformatics, text editing, pattern recognition, natural language processing, search engine, etc. The rapid growth of Internet technologies has resulted in diversified sources of statistics, which leads to the generation of uncertain data. This paper focuses on the problem of uncertain string matching in the field of bioinformatics, and the research content is as follows.

Firstly, a deep analysis is adopted on the current methodologies, and problems like long-time and large-scaled construction of index are discovered.

Secondly, in view of the fact that there are few character types in bioinformatics, a index named USAL with space cost of character type multiplied by string length is proposed in this paper. Concepts of positive and negative character are proposed in USAL. Based on the positive character, an effective string matching algorithm, namely GUSM, is proposed combined with the Greedy algorithm and Range Maximum Query. The probability of each character occurrence in each position is recorded by USAL, therefore, when matching strings, the position of the positive character with the largest probability will be returned in the form of time $O(1)$ by RMQ according to the given probability threshold before being verified. At last, results that meet the threshold will be returned. In order to improve the filtering effect of positive character position, filtering strategies based on minimum probability character and randomly-selected character are proposed.

Thirdly, to enhance the process of matching, a small-scaled index named BI is adopted on the foundation of bitmap thought. The application of BI allows the probability of positons in USAL to be bitmapped, which promoted another highly-efficient string matching algorithm named BUSM. When matching strings, BUSM will turn the matching operation into bit manipulation via BI, and then verify the results through probability threshold before returning the final correct answer. To improve the efficiency of BI in the period of verifying, MBI is then introduced to decrease the size of the candidate results, which further upgrades

the algorithm.

Finally, experiments on real and synthetic datasets with different features are carried out to verify the efficientcy of the proposed algorithms.

**Keywords:** uncertain string; index construction; filtering strategy; substring matching

# 目 录

# 第 1 章 绪 论

## 1.1 研究背景及意义

字符串是计算机系统中的一种重要数据类型，一个字符串通常是指定义在一个有限字母表集合上的字符序列，模式串通常是指在一段长字符序列中出现的子序列并且任意两个相邻的字符之间没有出现间隔。例如，在生物信息学中，脱氧核糖核酸(DNA)是一种生物大分子，可以组成遗传指令，引导生物的发育以及生命机能的运作。DNA 作为一种长链聚合物，组成单位为四种脱氧核苷酸，分别是：腺嘌呤脱氧核苷酸(dAMP)、胸腺嘧啶脱氧核苷酸(dTMP)、胞嘧啶脱氧核苷酸(dCMP)以及鸟嘌呤脱氧核苷酸(dGMP)。可以定义一个有限字母表集合 $\Sigma$={A,T,C,G}来分别表示以上四种脱氧核苷酸，那么一个 DNA 的序列就可以简约表示为字符串，例如，AGCTCT。GCT 就可以作为这个 DNA 序列的一个模式串，反之，GTC 就不是一个模式串，因为 GTC 之间被 C 隔开了。

字符串匹配问题通常是指从一个文本字符串当中搜索某个给定模式串的所有出现位置，这个问题一直以来就是计算机科学的一个中心问题。字符串匹配问题之所以有这么重要的地位，是因为这个看起来似乎简单的问题是大量应用的核心操作。比如在文本编辑程序中，经常出现要在一段文本中找到某一个模式的全部出现位置。典型的情况是，一段文本是正在编辑的文件，所搜寻的模式通常是用户提供的一个特定单词，通过解决这个问题就可以极大地提高文本编辑程序的响应性能。又或者在网络入侵检测中[1-3]，需要给定一组“特征”，利用这一组“特征”与网络中的数据内容进行比较来进行入侵行为检测、区分病毒特征码等。这个给定的“特征”通常是指入侵检测、病毒码提取、垃圾邮件过滤等应用中定义的模式串，用来表示攻击行为、病毒或者垃圾邮件区别于正常网络流量的特征。此外，在网络内容审计方面，网络内容审计包括应用层协议感知的内容语义解析，实时提取内容语义，如 HTTP 的 URL、邮件的发件人和收件人、MSN 会话参与人、P2P 传输文件名以及关键字审计等，快速关键字匹配技术是实现实时审计传输内容的关键。除了以上枚举的例子，字符串匹配还广泛应用于生物信息学[4,5]、模式识别[6]、搜索引擎[7]、语言翻译[8]、数据压缩[9-11]等领域。

近年来，许多先进技术不断发展起来用于存储和记录大量的数据，数据来源的差

异性导致了大量不确定数据的产生。比如在许多情况下，数据可能存在错误或者可能只是部分完成。在另一些情况下，数据点可能对应于仅模糊指定的对象，因此它们的表示也被认为是不确定的。同样地，通过调查和估算技术产生的数据在性质上也会具备不确定性。例如，传感器网络和卫星由于底层设备的局限性以及某些因素对于通信的干扰，会固有地收集带有噪声和传输误差的数据，从而产生大量的不确定数据。又或者在移动对象数据库轨迹的知识发现中，因为数据的未来行为只能够近似地预测，导致物体的移动轨迹也可能是未知的，许多与时空相关的应用数据本身就包含不确定性。在生物信息学上，由于生物的遗传和进化过程中发生的突变等因素，DNA 与蛋白质的结合图谱原本就需要对其序列属性进行灵活的描述，给序列数据附加不确定属性能够更加贴合于实际应用[12,13]。另外，在实际生活中，随着互联网的发展，人与人之间的关系变得日益密切，从而产生了大量的社交网络，人们在社会生活中的实际关系往往可以通过生活中的信息来衡量，但是信息通常会存在一定误差，因此人与人之间的关系也会存在不确定性[14-16]。不确定数据相较于确定数据其表现更加灵活，含义也更加丰富，同时也加大了处理的难度。当数据产生了不确定性后，一般可以使用概率量化来描述数据的不确定性。

通过上述讨论可知，各式各样的不确定数据广泛存在于人们的实际生活中。正是由于不确定数据具备如此广阔的应用基础，从不确定数据中挖掘有价值的信息将会具备重大的研究意义和商业价值，因此人们也越来越重视在不确定数据上的数据挖掘研究，也取得了一系列的进步和成果。比如文献[17-19]从不确定数据中挖掘频繁项集，文献[20-22]在不确定数据上进行 top-*k* 查询，以及文献[23-25]在不确定数据上挖掘序列模式等。

随着互联网、数字图书馆、大基因组项目的发展，促成了数据的巨幅增长，导致需要处理的文本规模变得越来越大。一方面由于数据的迅速增长和技术成本的限制，无法产生完全干净的数据，使得带噪声的和不确定性的文本数据变得流行起来。另一方面实际应用中对于字符串匹配性能的要求也越来越高，使得人们对字符串匹配问题的研究兴趣也与日俱增。

长期以来，人们经过大量的理论研究与实验论证，在字符串匹配问题的研究方面取得了有目共睹的成果，提出了一系列简洁的数据结构和高效的匹配算法。然而，现有的大量工作和高效算法大部分都是基于确定字符串的，确定字符串一般特指字符

串序列中的每个字符位置不存在歧义，也就是每一个字符位置只包含一个来自某个字母表集合中的字符。一些适用于确定字符串的数据结构和算法往往对于不确定字符串而言是低效的，甚至是无法处理的。一方面由于不确定字符串匹配问题至今仍然探索较少，许多研究工作尚处于起步阶段，另一方面不确定字符串又在大量应用中作为普遍存在的数据，具备研究的价值和需求。本文正是基于这样的研究背景将重点研究不确定字符串匹配问题。

## 1.2 研究现状

经过前文对研究背景的介绍，可以根据字符串数据是否具备不确定性，将字符串匹配问题划分为两类：确定字符串匹配问题和不确定字符串匹配问题。

目前针对确定字符串匹配问题研究相对比较成熟，所以确定字符串匹配算法有很多，比较常见主要包括：BF(Brute Force)算法[26]、RK(Rabin-Karp)算法[27]、KMP(Knuth-Morris-Pratt)算法[28]、BM(Boyer-Moore)算法[29]和 Sunday 算法[30]等。BF 算法又称朴素字符串匹配算法，算法思想是从左至右依次尝试匹配文本中的每一个位置，其过程可以形象地看成用一个包含模式的“模板”沿着文本滑动，优点是实现相对简单，缺点是效率较低。RK 算法结合了哈希函数的思想，对文本中模式串长度的子串计算哈希值，通过哈希值的比较，实现了只用一次比较来判断两者是否相等，在实际应用中可以达到线性期望时间。KMP 算法首先要对模式串进行预处理，当发现文本中某一个位置不与模式串匹配时，不需要将文本位置回溯，而是利用已经“部分匹配”的结果将模式串向前移动尽可能远的距离，是能够达到线性时间的经典算法。BM 算法通过预处理文本串和模式串，并行使用好后缀规则(good-suffix shift)和坏字符规则(bad-character shift)来使得模式串在匹配失败时跳跃较远的距离，实际应用中可以达到相较于 KMP 算法 3 至 5 倍的效果[31]。Sunday 算法在匹配过程中，当发现模式串位置不匹配时，算法跳过尽可能多的字符来进行下一步的匹配，比较适合处理短模式串的匹配[32]，甚至达到比 BM 更优的效率。相较于每个位置都是一个字符的确定字符串而言，不确定字符串的每个位置都是由来自某个字母表集合中字符的概率分布表示，这样使得不确定字符串的表达含义更加灵活丰富，同时也使得不确定字符串匹配问题变得更加复杂，因此以上针对确定字符串提出的算法并不能直接应用于不确定字符串匹配问题。

不确定字符串匹配问题可以简单定义为：对于一个不确定字符串 $T$，$T$ 中每一个

位置都是一组概率分布的字符集合，给定一个模式查询$(P,\tau)$，其中 $P$ 表示一个确定模式串，$\tau$ 表示确定模式串 $P$ 在文本串 $T$ 中出现的概率阈值，不确定字符串匹配就是从 $T$ 中找到所有满足 $P$ 出现并且 $P$ 的概率至少为 $\tau$ 的位置。目前而言，解决不确定字符串匹配问题的方法主要有以下几类：

第一类是不确定字符串近似匹配算法。如文献[33]提出的算法，返回结果的位置与给定的模式串不一定完全匹配，通过定义文本误差的度量：Hamming distance，使得返回结果与模式串完全匹配的结果之间存在一定的匹配误差。

第二类是不建立索引的不确定字符串匹配算法。如文献[34]提到的可能域算法，这种算法思想有点类似确定字符串匹配中的 BF 算法，想法比较直接，从左至右依次尝试匹配文本中的每一个位置，由于随着不确定字符串长度的增加，其能够表现的字符串含义也会呈指数级别增长，所以用这种方法虽然能得到与模式串完全匹配的结果，但是效率比较低。

第三类是建立索引的不确定字符串匹配算法。这类方法不但可以找到与模式串完全匹配的结果而且还能够保证较好的查询效率，其基本思想是在文本上实现简洁的索引结构以支持高效的在线模式查询。在确定字符串的索引结构方面，人们提出了广泛使用的数据结构有后缀树和后缀数组[35]，并且这些数据结构也都陆续提出了构建时间为 $O(n)$的算法，其中 $n$ 表示文本的长度。但是由于数据不确定性的限制，以上介绍的数据结构无法直接适用于不确定字符串。文献[36-38]提出了一系列对不确定字符串构建索引的方法，其主要思想大都是利用一个预设概率阈值对不确定字符串进行枚举剪枝，保留满足预设概率阈值的部分内容构建一个树形结构的索引，在此基础上搜索满足模式的查询。

## 1.3 研究内容

由以上讨论内容可知，从实际应用的实用性和查询效率考虑，既能找到与模式串完全匹配的结果又能保证较好查询效率的第三类解决方法是一个比较好的研究方向。如果可以找到一种索引结构，既能保证较小的存储规模，又具备良好的查询效率，同时还方便编程实现，从而具备较好的实用性，本文正是基于这个目的提出了相关的数据结构和匹配算法。

经过观察发现大量应用所需要的字符种类通常较少。例如，4 种核苷酸就能组合成携带遗传信息且表现形式多样化的 DNA，20 种左右的氨基酸就能构成承担细胞生

命活动的各种蛋白质，7 个音阶就能谱写成悦耳动听的乐曲，26 个英文字符就能撰写出不朽的文学名著。具体地来说，本文的贡献主要如下：

(1)基于生物信息学领域中字符种类较少的特点，提出了空间代价为字符种类乘字符串长度的索引 USAL。在 USAL 索引上定义了积极字符和消极字符的概念，结合贪心思想和范围最值查询在 USAL 索引上提出了相应的不确定字符串匹配算法 GUSM 及两种过滤策略。

(2)结合 Bitmap 思想对 USAL 索引进行改进，提出了一种小规模位图索引 BI，BI 可以对 USAL 进行有效的位映射，减少了索引的规模，进一步提高了空间利用率。在 BI 索引上提出一种不确定字符串匹配算法 BUSM，将子串匹配操作转化为位操作，提高了不确定字符串匹配的效率。

(3)为了提高 BUSM 算法在概率阈值验证阶段的效率，提出了多维位图索引 MBI，有效减少了阈值验证阶段中候选结果集合的规模，进一步提高了算法执行效率。

(4)最后，基于不同特征的真实数据集和人工合成数据集，从索引构建规模、索引构建时间和模式匹配效率三个方面进行对比，客观地分析比较了实验结果，论证了本文提出算法的高效性与可扩展性。

## 1.4 本文结构

本文主要研究了不确定字符串匹配问题。文章的主体共划分为 5 章，除第 1 章外，结构安排如下。

第 2 章为基础知识概述，详细地介绍了与不确定字符串匹配问题相关的概念，同时也分析了现有不确定字符匹配算法的不足之处。

第 3 章针对现有不确定字符串匹配算法存在构建索引效率低的问题，介绍了不确定字符串索引 USAL 及相应的匹配算法 GUSM。为了进一步提高 GUSM 算法对于积极字符位置的过滤效果，提出了基于最小概率字符和随机选择字符的过滤策略，解决了现有算法中索引规模大，索引构建时间长的问题。

第 4 章介绍了小规模位图索引 BI 及相应的匹配算法 BUSM。BI 索引在 USAL 的基础上进一步减小了索引规模，BUSM 算法将子串匹配操作转化为高效的位操作，进一步提高了不确定字符串的匹配效率。为了增强 BUSM 算法在概率阈值验证阶段的效率，提出了多维位图索引 MBI，MBI 索引可以有效减少候选结果集合的规模，从而进一步提高了算法执行效率。

第5章为实验及结果分析，基于多个不同特征的真实数据集和人工合成数据集，将不同的不确定字符串匹配算法从索引规模大小，索引构建时间，模式匹配效率三个方面进行对比，论证了本文提出算法的高效性与可扩展性。

最后是结论，结论对本文研究内容做了总结，并对本文未来研究方向做了展望。

# 第 2 章 基础知识概述

本章主要介绍一些与不确定字符串匹配相关的基础理论知识。在 2.1 节将主要介绍与不确定字符串匹配相关的概念及问题定义；在 2.2 节将主要介绍与字符串索引相关的数据结构；在 2.3 节将主要介绍现有解决不确定字符串问题的几类方法，并分析了这些方法的不足；最后在 2.4 节将对本章内容进行小结。

## 2.1 相关概念

**定义 2.1 字符串**：给定一个有限字符集合 $\Sigma=\{s_1,s_2,\ldots,s_\sigma\}$，由零个或者多个来自 $\Sigma$ 中的字符构成的有限序列。一般记为：

$$S=s_1s_2\ldots s_n(n\geq 0) \tag{2-1}$$

公式(2-1)中，$S$ 表示字符串的名称，$s_i(1\leq i\leq n)$是来自有限字符集合 $\Sigma$ 中的字符，字符串中字符的数目 $n$ 称为串的长度。零个字符的串称为空串，它的长度为零。

**定义 2.2 子串**：字符串 $S$ 中任意个连续字符组成的子序列。包含子串的字符串 $S$ 相应地称为主串。通常称字符在序列中的序号为该字符在 $S$ 中的位置，子串在主串中的位置以子串的第一个字符的位置来表示。

例如，假设 $A$、$B$、$C$、$D$ 为如下 4 个字符串：

(1) $A$=HE;

(2) $B$=BEI;

(3) $C$=HEBEI;

(4) $D$=HE BEI。

它们的长度分别为 2、3、5 和 6。其中 $A$ 和 $B$ 都是 $C$ 和 $D$ 的子串，$A$ 在 $C$ 和 $D$ 中的位置都是 1，而 $B$ 在 $C$ 中的位置是 3，在 $D$ 中的位置则是 4。

**定义 2.3 有效偏移**：给定一个长度为 $n$ 的字符串 $T$ 和一个长度为 $m(m\leq n)$的字符串 $P$，如果有 $0\leq i\leq n-m$，$1\leq j\leq m$，使得 $T[i+j]=P[j]$，则称 $P$ 在 $T$ 中出现且有效偏移为 $i$。一般将 $T$ 称作文本串，$P$ 称作模式串。

**定义 2.4 字符串匹配**：从文本串 $T$ 中搜索到模式串 $P$ 的所有出现。当模式串 $P$ 相对于文本串 $T$ 的有效偏移集合不为空时，记录有效偏移集合的元素或者个数作为字符串匹配的返回结果。如图 2-1 所示，在偏移 $i$=3 处，模式串 $P$ 在文本串 $T$ 中出现

了一次，偏移 $i$=3 是一个有效偏移。字符串匹配问题是找出模式串 $P$ 在文本串 $T$ 中的所有出现，可以看到模式串 $P$ 在文本串 $T$ 中一共出现了两次，有效偏移集合为{3,9}。

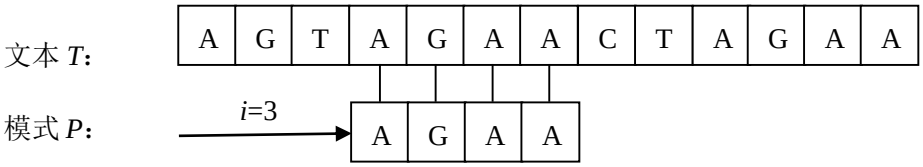



图 2-1 字符串匹配问题

当前文献对于不确定字符串的研究一般涉及到两种模型：字符串级模型和字符级模型[38]401。在字符串级模型中，一段完整的字符串被进行枚举处理和概率量化，另一种字符级模型是将每一个字符位置进行概率量化。本文的重点研究对象是应用中更加普遍出现的字符级模型。

**定义 2.5 不确定字符串**：给定一个有限的字符集合 $\Sigma=\{s_1,s_2,\ldots,s_\sigma\}$，由零个或者多个来自 $\Sigma$ 中的字符和其出现概率构成的有限序列。一般记为：

$$X=x_1x_2\ldots x_n\ (n\geq 0) \tag{2-2}$$

$$x_i=\{(s_j,\pi_i(s_j))\ :\ j\in\{1,2,\ldots,\sigma\}\}\ \ (1\leq i\leq n) \tag{2-3}$$

公式(2-2)中，$X$ 表示不确定字符串的名称，一个不确定字符串由有限个 $x_i$ 构成。

公式(2-3)中，$x_i$ 表示取自有限字符集合 $\Sigma$ 中的字符和其对应概率组成的组对集合，$s_j$ 表示取自 $\Sigma$ 中的字符，$\pi_i(s_j)$表示 $s_j$ 在位置 $i$ 的出现概率。

如图 2-2 所示，设定 $\Sigma$={A,C,G,T}，列举了一个长度为 5 的不确定字符串 $X$。通过观察，$X$ 中出现概率非零的字符数目为 9，与一个位置只能出现一个字符的确定字符串相比，一个较短长度的不确定字符串也可以包含大量的字符数目。虽然不确定字符串每个位置可以出现多个字符，但是要求每个位置字符出现概率的和为 1。

| $X$[1] | $X$[2] | $X$[3] | $X$[4] | $X$[5] |
|---|---|---|---|---|
| A 0.3 | A 0.6 | T 1 | A 0.5 | A 1 |
| C 0.4 | G 0.4 | | G 0.5 | |
| T 0.3 | | | | |

图 2-2 不确定字符串

在不确定字符串的每个位置选择一个非零概率字符，通过这种枚举方法可以将一个不确定字符串生成多个确定字符串。例如，对图 2-2 中的不确定字符串枚举可以得到表 2-1 中的结果。

表 2-1 不确定字符串 $X$ 的枚举结果

| ID | 枚举出的确定字符串 |
|---|---|
| 1 | AATAA |
| 2 | AATGA |
| 3 | AGTAA |
| 4 | AGTGA |
| 5 | CATAA |
| 6 | CATGA |
| 7 | CGTAA |
| 8 | CGTGA |
| 9 | TATAA |
| 10 | TATGA |
| 11 | TGTAA |
| 12 | TGTGA |

**定义 2.6 匹配概率：**不确定文本串 $X$ 中出现一个确定模式串 $P$ 的概率。在本文中，为了方便考虑问题，假设不同位置的字符出现概率具备独立性，用公式表示为：

$$Pr(P, X) = \prod_{i=1}^{|P|} \pi_i(P[i]) \tag{2-4}$$

公式(2-4)中，$Pr(P,X)$表示确定模式串 $P$ 在不确定文本串 $X$ 中出现的概率，在不确定文本串 $X$ 没有歧义的情况下，也可以简约写作 $Pr(P)$。公式等号右侧表示将确定模式串 $P$ 中出现的每一个字符的概率 $\pi_i(P[i])(1\le i\le|P|)$连续相乘起来。例如，使 $P$=AATAA，则依据图 2-2 和表 2-1 可知，$Pr(P)=0.3\times0.6\times1\times0.5\times1=0.09$。

**定义 2.7 不确定字符串模式有效偏移：**给定一个长度为 $n$ 的不确定文本串 $X$，一个长度为 $m(m\le n)$的确定模式串 $P$ 及一个查询概率阈值 $\tau(0<\tau\le1)$，如果有 $0\le i\le n-m$，$1\le j\le m$，使得 $X[i+j]=P[j]$且 $Pr(P)\ge\tau$，则称 $P$ 在 $X$ 中出现且有效偏移为 $i$。

**定义 2.8 不确定字符串匹配：**从不确定文本串 $X$ 中搜索确定模式串 $P$ 的所有出现。与确定字符串匹配问题同理，当确定模式串 $P$ 相对于不确定文本串 $X$ 的有效偏移集合不为空时，可以记录有效偏移集合的元素或者个数作为不确定字符串匹配的返回结果。

给定一个查询的确定模式串 $P$ 和概率阈值 $\tau$ 分别为：(AA,0.5)。由图 2-2 可知，确定模式串 $P$ 在不确定字符串 $X$ 中一共出现了一次，有效偏移集合为{3}。在偏移 $i$=0 时，确定模式串 $P$ 没有出现，因为 $Pr(P)=0.3\times0.6=0.18<0.5$。

## 2.2 字符串的索引结构

字符串的索引结构包含很多种，其中后缀树与后缀数组都是非常高效的工具。下面重点介绍一下这两种数据结构。

### 2.2.1 后缀树

后缀树的概念最早由 Weiner 于 1973 年提出，后缀树是一种树形的数据结构，可以快速解决许多关于字符串的问题[39]。在介绍后缀树之前，首先介绍另一种可以高效索引字符串的树形数据结构：字典树。

字典树，又称单词查找树、Trie 树。典型应用是用于统计、排序和保存大量的字符串，所以经常被搜索引擎系统用于文本词频统计，其优点是可以利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较。对于字符串集合{“bear”，“bell”，“bid”，“sell”，“stock”，“stop”}建立一棵字典树，如图 2-3 所示。

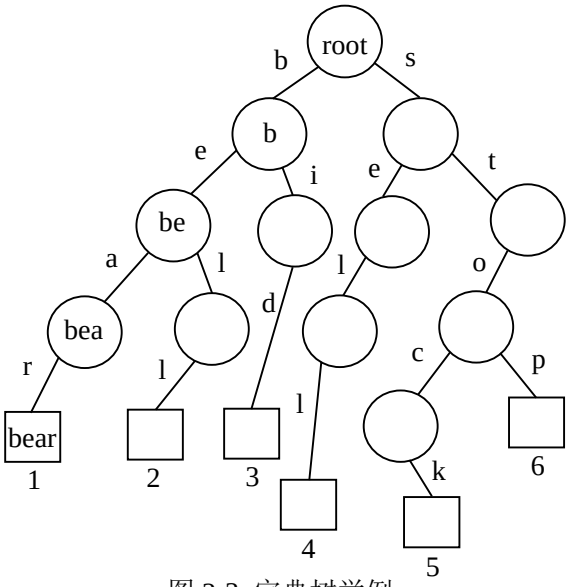



图 2-3 字典树举例

在图 2-3 中，字典树的根结点不包含任何字符，除根结点外每个结点只包含一个字符，每个结点的所有子结点包含的字符各不相同，从根结点到某一个结点，将路径上的字符连接起来得到的字符串为该结点所表示的字符串。例如，从根结点沿着 b→e→a→r 的路径到达叶子结点 1，叶子结点 1 所表示的字符串就是字符串集合中的“bear”。字典树可以实现字符串的快速检索，由根结点开始，以查找字符串中包含的字符为导向沿路径检索，如果经过路径上的字符连接起来可以得到查找字符串，则

字符串就包含在字典树中，反之查找失败。字典树也可以实现单词排序，一般每个字典树的结点都包含字母表大小|Σ|条指针，每条指针都按照字典顺序对应字母表 Σ 中一个字符，给定一个包含 $n$ 个单词的字符串集合，对这个字符串集合建立字典树，然后对字典树进行先根遍历就得到了排序好的单词表。

后缀树是包含了一个字符串所有后缀并压缩后的字典树，字典树的优点及应用同样适用于后缀树。例如，给定一个字符串 $S$=BANANA，$S$ 所包含的所有后缀为{“$”，“A$”，“NA$”，“ANA$”，“NANA$”，“ANANA$”，“BANANA$”}，其中的字符$不存在 $S$ 的字母表 Σ 中，用来表示空串。对 $S$ 的后缀集合建立一棵后缀树，如图 2-4 所示。

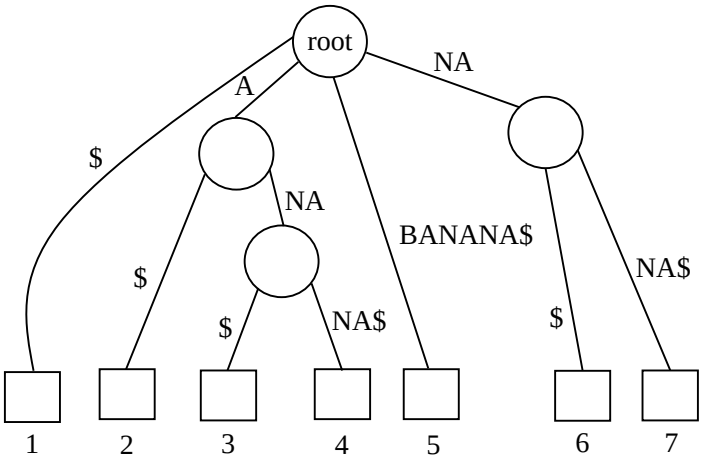



图 2-4 字符串 $S$=BANANA 的后缀树

在图 2-4 中，后缀树将单一路径到达某一个结点上的字符做了路径压缩处理，例如从根结点 root 到 5 号叶子结点的路径，如果想要在图 2-3 中的字典树上表示后缀“BANANA$”应该需要 7 个结点，但是在后缀树中将单一路径上的结点字符压缩后，只需要一个结点就可以表示，这样做相较于后缀的简单字典树而言更加节省空间，使后缀树的空间复杂度可以达到 $O(n)$。在 $S$ 的末尾加入字符$有两个目的，第一个目的是表示空串，因为空串可以作为任何字符串的后缀，第二个目的是用来将隐藏的后缀显示出来。例如，不添加$时，到达 2 号和 3 号叶子结点的路径会被到达 4 号叶子结点的路径所隐藏。

根据后缀树的定义，后缀树可以先对字符串的后缀集合建立字典树，然后再进行路径压缩后得到，但是这种方法效率较低，极大限制了后缀树的应用。McCreight 在 1976 年[40]和 Ukkonen 在 1995 年[41]分别对后缀树的构建算法改进完善，使得后缀树有了线性时间 $O(n)$构建完成的高效算法。后缀树线性构建算法中最大的改进就是提

出了后缀链接(suffix link)的概念，在图 2-4 的基础上添加后缀链接后，如图 2-5 所示。

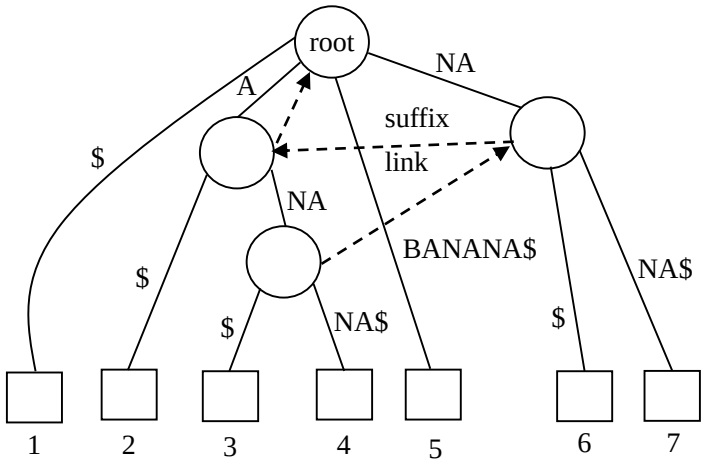



图 2-5 后缀指针示例

McCreight 提出的算法是有缺陷的，构建时需要逆序输入，就是从字符串的末端开始输入，这样很难作为在线算法，在一定程度上限制了应用，Ukkonen 在此算法的基础上做了改动，使之可以对字符串正序构建后缀树，所以一般用 Ukkonen 算法来构建后缀树。给定一个长度为 $n$ 的文本串 $T[1\ldots n]$，Ukkonen 算法的主要思想是假设文本串 $T[1\ldots i\text{-}1]$的后缀树已经构建完毕，在此基础上的每个后缀位置上加上字符 $T[i]$，就可以得到 $T[1\ldots i]$的后缀树。引入后缀链接后，在加入字符 $T[i]$时，只需要沿着后缀链接连接的结点进行处理，不用每一次都从根结点遍历到插入位置，从而提高了算法效率，使后缀树构建算法达到了 $O(n)$。

虽然后缀树可以在线性时间内构建完成，但是也存在着比较大的缺点：空间开销较大且构建算法过程复杂。

### 2.2.2 后缀数组

后缀数组可以作为后缀树的一个小巧的替代品，能够实现后缀树的许多功能而时间复杂度也不太逊色。后缀树的构建过程通常比较复杂导致其应用性不佳，后缀数组相较于后缀树更加容易编程实现，从而实用性更强，在一些信息学竞赛中经常会利用后缀数组来代替后缀树。后缀数组可以有效解决一些与字符串相关的问题，例如求解最长重复子串问题，重复出现子串计数问题，连续重复子串问题等。此外，后缀数组主要的优点是比后缀树所占用的空间要小很多，在实践中要比后缀树的空间少 3 至 5 倍[42]。给定字符串 $S$=BANANA，$S$ 的后缀集合为{“$”,“A$”,“NA$”,“ANA$”,“NANA$”,“ANANA$”,“BANANA$”}，对于 $S$ 的后缀集合按照字典顺序进行

排序，后缀集合排序后的结果如表 2-2 所示。

表 2-2 字符串 *S* 的后缀排序结果

| 排序后序号 | 字符串 *S* 后缀 | 后缀起始位置 |
|---|---|---|
| 1 | $ | 7 |
| 2 | A$ | 6 |
| 3 | ANA$ | 4 |
| 4 | ANANA$ | 2 |
| 5 | BANANA$ | 1 |
| 6 | NA$ | 5 |
| 7 | NANA$ | 3 |

对一个长度为 $n$ 的文本串 $T[1\ldots n]$建立后缀数组 $SA[1\ldots n]$，其中数组中的元素 $SA[i](1\le i\le n)$表示的是 $T$ 中按照字典顺序排序的第 $i$ 个后缀的起始位置。由此，对表 2-2 中的结果进行整理得到字符串 *S*=BANANA 的后缀数组 *SA*，如图 2-6 所示。

*SA*：

| 7 | 6 | 4 | 2 | 1 | 5 | 3 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

图 2-6 字符串 S = BANANA 的后缀数组

在图 2-6 中，*SA*[4]=2，表示了字符串 *S*=BANANA 的后缀集合中，按照字典顺序排名第 4 的后缀为“ANANA$”，它在字符串 *S* 中的起始位置为 2。

## 2.3 不确定字符串匹配算法

根据第一章对研究现状的讨论，可以将不确定字符串匹配算法大致划分为三类，分别为不确定字符串近似匹配算法，不建立索引的不确定字符串匹配算法以及建立索引的不确定字符串匹配算法。在本文中主要研究的是可以返回完全匹配模式串结果的算法，所以主要介绍后两种解决不确定字符串匹配问题的算法。

### 2.3.1 不建立索引的求解方法

不建立索引的求解方法最大的优点是容易编程实现。如文献[34]提到的可能域算法，这种算法思想有点类似确定字符串匹配中的 BF 算法，从左至右依次尝试匹配文本中的每一个位置，算法的时间复杂度为 $O(|\Sigma|nm)$，用这种方法虽然可以得到与模式串完全匹配的结果，但是效率比较差。文献[43]对于确定字符串匹配中的 BM 算法进

行了改良，使之可以应用于不确定字符串，这也提供了一种解决不确定字符串匹配问题的方向，就是针对现有成熟的确定字符串算法进行改良使其可以应对不确定字符串匹配问题。但是不建立索引的求解方法通病是查询效率较差，特别是针对固定文本内容进行反复模式查询的时候。

### 2.3.2 建立索引的求解方法

建立索引的不确定字符串匹配算法不但可以找到与模式串完全匹配的结果而且还能够保证较好查询效率，其基本思想是在文本上实现简洁的索引结构以支持高效的在线模式查询。后缀树和后缀数组是处理确定字符串索引的有力工具，但是因为不确定字符串的每个位置都是一组来自有限字母表 $\Sigma$ 中字符的概率分布，随着不确定字符串长度的增加，其所能代表的确定字符串含义也以指数级别增长，字符串的不确定特性导致了针对确定字符串提出的高效索引并不能够直接适用。

文献[36,37]都各自提出了一种树形结构的索引，在索引上实现了基于固定查询概率阈值 $\tau$ 的模式查询，当查询概率阈值 $\tau$ 发生了变化，就需要重新建立树形索引结构以应对当前 $\tau$ 的模式查询。由于对于查询具备一定的条件限制，导致实际应用中通用性不佳。文献[38]提出了一种相对通用性较好的索引方法，通过在索引构建阶段预设一个概率阈值 $\tau_{min}$，可以实现任意查询概率阈值 $\tau \geq \tau_{min}$ 的模式查询。下面主要介绍下这篇文献的方法。

在文献[38]中首先定义了一种特殊不确定字符串，特殊不确定字符串在每一个位置只包含一个来自有限字母表集合 $\Sigma$ 中的字符和其对应出现的概率。然后再对特殊不确定字符串建立索引，索引分为三个部分：后缀树，后缀数组和连乘概率数组。在建立索引的时候，可以将特殊不确定字符串看作两部分组成：字符部分以及概率部分。当忽视概率部分时，特殊不确定字符串就可以看作是确定字符串，那么就可以在这个基础上构建后缀树与后缀数组。正如前面介绍到的，后缀树可以对确定字符串的所有后缀进行字典顺序的索引，后缀数组可以按照字典顺序记录每个后缀的起始位置。最后对于字符的概率部分可以建立一个连乘概率数组 $C$，因为模式串的出现概率等于模式串中各个字符在相应位置出现概率的乘积，所以对于特殊不确定字符串一个子串 $X_i...X_{i+j}$ 的出现概率可以用 $C[i+j]/C[i-1]$求得。假设给定一个特殊不确定字符串 $X$={(B,0.4),(A,0.7),(N,0.5),(A,0.8),(N,0.9),(A,0.6)}，对 $X$ 建立索引如图 2-7 所示。

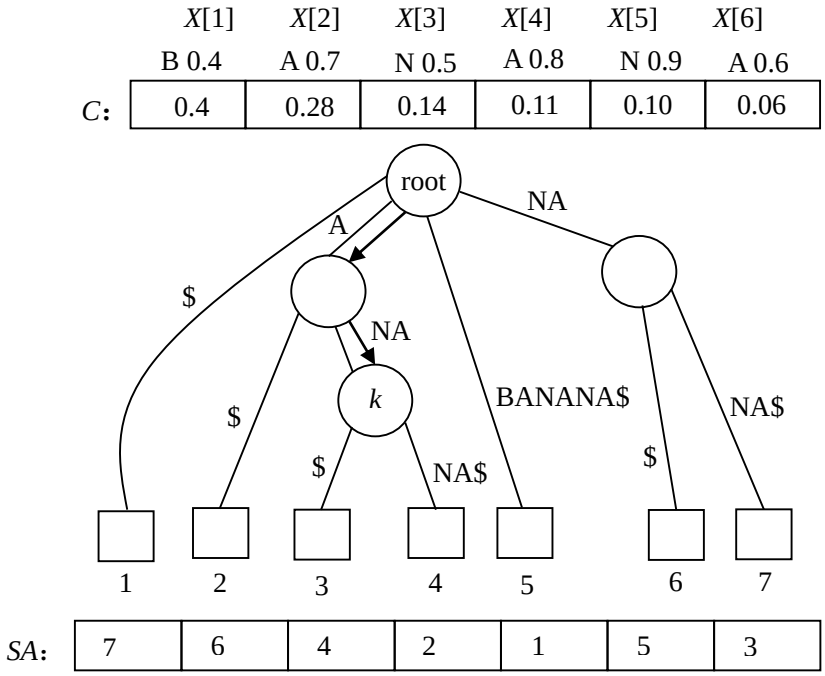



图 2-7 特殊不确定字符串 *X* 的索引

在图 2-7 中，给定一个模式查询(ANA,0.3)，通过给定的确定模式串 *P*=ANA 在后缀树中从根结点出发，沿着构成确定模式串 *P* 的字符导向路径向叶子结点方向寻找，可以找到 *k* 结点。后缀数组 *SA* 的一个重要性质就是可以将字符串的所有前缀相同的后缀的起始位置存储在数组中相邻的区域，后缀树中的 *k* 结点可以看作是一些叶子结点的最低公共祖先，这些叶子结点的共同点在于所表示的后缀中都包含了确定模式串 *P* 作为前缀，以 *k* 结点为祖先的叶子结点在 *SA* 中划定了一个后缀范围[3,4]。当 $i \in [3,4]$ 时，*SA*[*i*]就是指向需要验证的后缀的起始位置，以[3,4]为起始位置的后缀有{“ANA$”, “ANANA$”}，这些后缀都包含了确定模式串 *P*=ANA 作为前缀，根据后缀树和后缀数组的索引映射关系就可以得到包含了确定模式串 *P* 在不确定文本串中的位置信息。

以上的索引结构和匹配算法都是对于特殊不确定字符串而言的，并不能直接应用于每个位置对应于一组字符概率分布的一般不确定字符串。为了将以上的索引结构应用于一般不确定字符串，文献[38]提出索引构建阶段前设置一个概率阈值 $\tau_{min}$，通过使用文献[44]中的方法对一般不确定字符串进行预处理。预处理的过程相当于依据预设概率阈值 $\tau_{min}$，将一般不确定字符串中满足 $\tau_{min}$ 的极大子串枚举出，并且拼接起来，从而得到可以处理的特殊不确定字符串，在转换后的特殊不确定字符串的基础上，再建立后缀树和后缀数组构成的索引结构。

## 2.4 本章小结

本章主要介绍了与不确定字符串匹配相关的基础知识和现有解决方法。首先介绍了字符串的概念以及字符串匹配问题的定义，随后将相关概念和问题定义引申到了不确定字符串。其次介绍了字符串索引相关的数据结构：后缀树与后缀数组，随后简明介绍了这两种索引结构的相关特性和使用方法。最后着重介绍了现有的解决不确定字符串匹配问题的两类方法：不建立索引的不确定字符串匹配算法和建立索引的不确定字符串匹配算法。

# 第 3 章 基于贪心思想的不确定字符串匹配

设定一个有限字母表集合 $\Sigma=\{s_1,s_2,\ldots,s_\sigma\}$，不确定字符串 $X$ 就是由零个或者多个来自 $\Sigma$ 中的字符和其出现概率构成的有限序列。给定一个确定模式串 $P$ 和一个查询概率阈值 $\tau$，当确定模式串 $P$ 在不确定字符串 $X$ 中的出现概率 $Pr(P)\geq\tau$ 时，称确定模式串 $P$ 在不确定字符串 $X$ 中出现，其中$|X|=n$，$|P|=m$，$m\leq n$。不确定字符串匹配问题就是从一个不确定文本串 $X$ 中搜索确定模式串 $P$ 的所有出现。

## 3.1 问题分析

不确定字符串由于数据的不确定性使其具备了以下几个较难处理的方面。首先，不确定字符串在每一个位置可以出现多个来自有限字母表集合 $\Sigma$ 中的字符。随着不确定字符串长度的增长，不确定字符串所可以表达的确定字符串数量也以$|\Sigma|$的指数级别增长。例如，设 $\Sigma=\{A,C,G,T\}$，当以 $\Sigma$ 构建一个长度为 $n$ 的不确定字符串 $X$ 时，其所能表现的确定字符串数量可以达到 $4^n$。其次，从不确定字符串 $X$ 中的相同位置可以枚举出多个出现概率不等的确定子串，使得直接对不确定字符串 $X$ 构建索引变得困难。再次，不确定字符串匹配时给定的查询概率阈值 $\tau\in(0,1]$也是可以变化的，所以根据变化范围内任意的查询概率阈值 $\tau$ 都在不确定字符串 $X$ 上枚举出确定字符串并建立索引将会导致空间的剧烈消耗。最后，在不确定字符串 $X$ 上任意指定一个位置 $i$，以位置 $i$ 起始的子串出现概率也是会随着子串长度的增加呈现没有规律的非递增变化，所以这个特性同样会导致对于变化范围内任意的查询概率阈值 $\tau$ 从 $X$ 中枚举确定字符串并建立索引的想法变得困难。

字符串作为计算机系统中的一种重要数据类型，计算机所处理的信息都可以存储表示为 0 和 1 的二进制序列。随着互联网和计算机技术的发展，促成了数据的巨幅增长，需要处理的文本规模变得越来越大。一方面，数据增长和技术成本等因素导致带有噪声的和不确定性的数据大量出现，另一方面，字符串匹配问题涉及到了大量应用的核心操作，为了提高这些应用的性能效率，从而对于字符串匹配的性能要求也越来越高。目前大量对于字符串匹配问题的研究工作都是以确定字符串为考虑前提，基于确定字符串提出的算法和数据结构由于没有考虑文本数据的不确定性，不能直接适用于不确定字符串。虽然近年来人们对于不确定字符串匹配也做了一些研究工作，提出了一些解决方法，但是因为不确定文本数据难以处理的特性，相关理论仍然

不够成熟。现有的解决不确定字符串匹配问题的方法主要有三类，第一类是不确定字符串近似匹配算法，这类算法只能返回与给定确定模式串 $P$ 有一定匹配误差的结果，第二类是不建立索引的不确定字符串匹配算法，这类算法特点在于实现简单但是匹配效率较差，第三类是建立索引的不确定字符串匹配算法，这类算法通常具备较好的匹配效率，但是索引构建复杂且时间与空间消耗较大。

本文主要研究的是可以返回完全匹配模式串结果的算法，通过对相关文献的研究和对比，认为建立索引的不确定字符串匹配算法是一个解决问题比较好的方向。文献[38]提出了一种有效解决一个位置只有一个字符和其出现概率的特殊不确定字符串匹配算法，随后通过将不确定字符串转换为特殊不确定字符串，间接地解决了不确定字符串匹配问题。其主要思想分为三个阶段：第一个阶段，将不确定字符串预处理成为文中定义的特殊不确定字符串。预处理过程中通过一个预设概率阈值 $\tau_{min}$ 从不确定字符串中将所有出现概率满足 $\tau_{min}$ 且可保留长度最长的子串枚举出，并用不属于有限字母表集合 $\Sigma$ 的连接符\$连接起来，这样就得到了一个转换后的特殊不确定字符串。这样的枚举过程相当于对不确定字符串每个字符位置进行压缩，只保留满足预设概率阈值 $\tau_{min}$ 的子串，势必造成不确定字符串长度规模的增长，对一个长度为 $n$ 的不确定字符串 $X$ 进行如上过程的转换，其转换后的长度上限可达 $\Omega(n^2)$[38]407。为了缩短转换后特殊不确定字符串的长度规模，引用了文献[44]中提到的一种扩展转换方法，可以将转换后的长度规模限定为 $O((1/\tau_{min})^2 n)$[44]308。第二阶段，在已经转换的不确定字符串的基础上构建后缀树、后缀数组等索引部件。第三阶段，通过后缀树、后缀数组的特性，在不考虑确定模式串 $P$ 的出现概率的前提下，找到不确定字符串中确定模式串 $P$ 可能出现的位置集合，从中剔除不满足查询概率阈值 $\tau$ 的结果。

通过对文献[38]中的方法进行分析，其主要存在如下几个问题：第一，不能直接处理不确定字符串，只能先将不确定字符串转换为其文中所定义的特殊不确定字符串后再构建后缀树、后缀数组等索引部件。这样做的原因是后缀树、后缀数组是基于确定字符串提出的索引结构，不能直接适用于不确定字符串。第二，不确定字符串转换为特殊不确定字符串后，使得原来相对较短的长度增加，对于长文本字符串建立后缀树和后缀数组会引发时间和空间的大量消耗。第三，由于预处理阶段需要一个预设概率阈值 $\tau_{min}$，$\tau_{min}$ 带来的优点是可以对不确定字符串所表示的确定字符串进行剪枝，缩小需要构建索引的字符串规模。但是确定模式串 $P$ 的查询概率阈值 $\tau$ 可以在 0 至

1 的范围内任意变化，一旦出现 $\tau<\tau_{min}$ 时，就会无法返回查询结果，只能设置一个新的 $\tau_{min}$ 重新建立索引。

根据上述存在的问题，本章提出了一种新的索引结构，并在索引结构上提出了相应的不确定字符串匹配算法。新的索引结构采用了数组形式来存储不确定字符串中的字符及其出现概率，不需要对不确定字符串中表示的确定字符串进行枚举转换，从而不确定字符串的处理长度规模不会增加，同时也可以应对确定模式串 $P$ 的查询概率阈值 $\tau$ 在 0 至 1 范围内任意变化的查询。下面首先介绍本文提出的索引，随后介绍其相应的不确定字符串匹配算法。

## 3.2 USAL 索引

通过分析现有方法存在的问题，发现不确定字符串在每一个位置可以出现多个来自有限字母表集合 $\Sigma$ 中的字符，这个特性使得直接对不确定字符串建立索引变得困难，前面介绍过的对不确定字符串建立的索引方法[36-38]中都通常会预设一个概率阈值 $\tau_{min}$，通过 $\tau_{min}$ 对不确定字符串枚举出符合条件的极大子串，再对这些子串建立索引，这样做通常会带来两个弊端：第一，因为不确定字符串中相同的位置可以枚举出多个确定子串，那么相同位置满足 $\tau_{min}$ 的确定子串通常也会出现多个，对于这些枚举出来的子串再建立索引，势必会导致原有不确定字符串长度规模的增加，引起索引结构空间规模的剧烈消耗。第二，由于给定的确定模式串 $P$ 的查询概率阈值 $\tau$ 可以在 0 至 1 范围内任意变化，通过预设概率阈值 $\tau_{min}$ 预处理后，就建立了确定模式串 $P$ 的出现概率 $\tau\geq\tau_{min}$ 的依赖性，使得在预处理阶段选择一个合适的 $\tau_{min}$ 变得困难，一旦出现 $\tau<\tau_{min}$ 的情况，就需要重新建立索引带来额外的时间开销。

基于对这两个弊端的改善，本文提出了一种新的索引结构：不确定字符串邻接表(Uncertain String Adjacency List,USAL)，这种索引结构主要是从数据结构中图的存储结构邻接表得到的启发。邻接表可以作为一种数组和链表的组合，由于不确定字符串的字母表集合 $\Sigma$ 一般都是确定的，所以可以用一个一维数组来表示不确定字符串的字母表信息，再把所有在不确定字符串中出现的字符概率链接到其对应的数组元素上。因此，只要有一个字符在不确定字符串中出现过，数组中对应的位置就会存在一条链表。vector 在许多高级语言中作为一种可以存储任意类型的容器，vector 相对于普通数组有两个优点，第一个优点是可以动态增长，使用时不需要对 vector 初始化一个长度值，其长度会随着元素的插入自动变化。第二个优点是 vector 通常

作为标准模板库存在，其包含了许多现成的操作方法，在插入、删除与检索等方面比普通数组更加高效与方便。本文实验部分采用的是 C++编程语言，所以选择 vector 作为邻接表中链表部分的替代品，其中 vector 中的元素保存了字符在对应位置的出现概率。

有限字母表集合 $\Sigma$ 在 USAL 中对应了数组部分，数组作为一段连续存储的空间，最大的优势在于可以提供对于数组元素 $O(1)$时间的随机访问，为了实现通过字符对有限字母表集合 $\Sigma$ 对应的数组元素随机访问，从哈希表的直接寻址方式得到启发，最直接的想法就是通过字符之间的差值来定位数组中元素的位置。以生物信息学中 DNA 的有限字母表集合 $\Sigma$={A,C,G,T}为例，发现有限字母表集合 $\Sigma$ 中的字符不一定是连续的，为了保持通过字符来随机访问有限字母表集合 $\Sigma$ 中相应数组元素的特性，下面将分别介绍两种解决方案。

第一种对于类似 DNA 的有限字母表集合 $\Sigma$={A,C,G,T}而言，其特点是包含的字符数量较少，不同字符之间的差值跨度较大，可以将跨度差值较大的字符进行转换，使字符之间的差值变得紧凑。例如将 $\Sigma$ 中字符差值较大的字符 G 和 T 分别转换为 B 和 D，得到一个新的有限字母表集合 $\Sigma$'={A,B,C,D}。以一段 DNA 的不确定序列 $X$ 为例，构建不确定字符串索引 USAL，如图 3-1 所示。

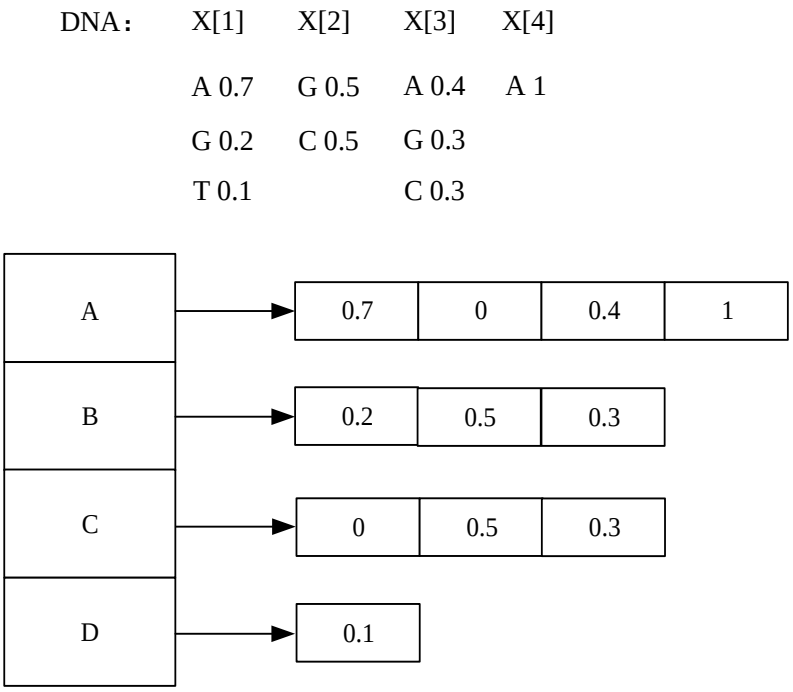



图3-1 DNA不确定序列构建USAL

在图 3-1 中，一段 DNA 的不确定字符串 $X$ 的长度为 4，对其构建不确定索引 USAL 后，左边的数组表示有限字母表集合 $\Sigma$'={A,B,C,D}，其中 B 和 D 分别是由脱氧核苷酸字符集合 $\Sigma$ 中的 G 和 T 转换得到，每个字母表数组元素后面都链接了一个字符出现概率表，其中存储了各个位置字符的出现概率，当字符出现概率表中相应位置没有字符出现时，其概率用 0 填充。

蛋白质(protein)是生物信息学经常研究的另一种生物大分子，蛋白质是构成生物体细胞的基本有机物，是生命活动的主要承担者。构成蛋白质的基本单位是氨基酸，其种类约为 20 种，可以表示为一个有限字母表集合 $\Sigma$={A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}，其特点是包含的字符数量较多，不同字符之间的差值跨度不大，可以使用第二种解决方案，申请一段较大的空间，例如申请空间大小为 26，使其可以完全包含 $\Sigma$。以一段蛋白质的不确定序列 $X$ 为例，构建不确定字符串索引 USAL，如图 3-2 所示。

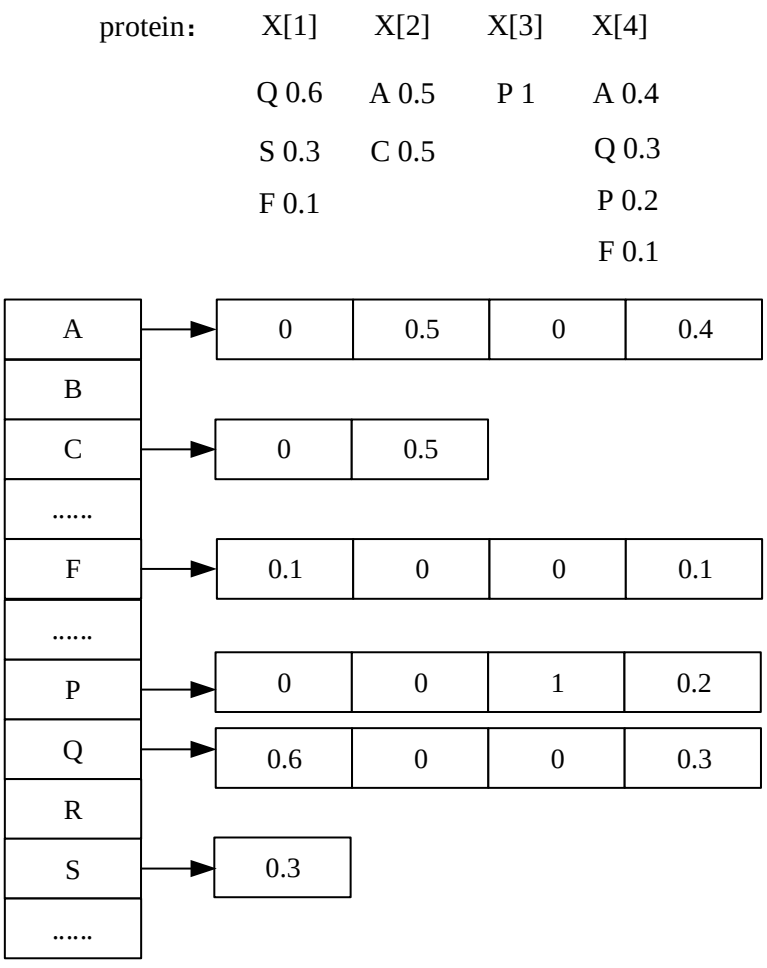



图3-2 蛋白质不确定序列构建USAL

USAL 主要由字母表数组和字符出现概率表两部分组成，构建的步骤也可以分成两个阶段，算法具体过程如算法 3.1 所描述。第一个阶段首先构建字母表数组部分，对应算法3.1的第1行。在构建字母表数组前，可以根据之前提到的两种方案，选择一种较为合适的方案来限定数组的大小。第二阶段根据数据集里字符的出现概率信息将概率值插入到对应字符的数组元素后面，对应算法 3.1 的第 3-11 行。

**算法**3.1　USALC算法

**Input:** the size of $\Sigma$ *sigma*, uncertain string *X*

**Output:** uncertain string adjacency list *USAL*

1: build an array *alphabet*[*sigma*] for $\Sigma$

2: *length* ← the size of *X*

3: **for** *i* ← 0 to *length*-1 **do**

4: *position* ← *i*

5: *letter* ← *X*[*i*].*char* //记录*X*[*i*]中的字符

6: *probability* ←*X*[*i*].*pro* //记录*X*[*i*]中的字符出现概率

7: **for** *j* ← the size of *alphabet*[*letter* – $\Sigma$[0]] to *position* **do**

8: **if** *j* is equal to *position* **then** //插入字符出现概率

9: add *probability* to *alphabet*[*letter* – $\Sigma$[0]].*link*

10: **else**

11: add 0 to *alphabet*[*letter* – $\Sigma$[0]].*link*

12: **return** *USAL*

## 3.3 GUSM 算法

在 3.1 节提出了现有建立索引结构解决不确定字符串匹配问题时存在的三个不足点，第一是无法直接处理不确定字符串数据，需要预处理转换。第二是预处理过程会造成需要建立索引的字符串长度规模增加，导致空间和时间的剧烈消耗。第三是由于预处理阶段预设概率阈值 $\tau_{min}$ 的限制，一旦出现查询概率阈值 $\tau<\tau_{min}$，就需要重新建立索引，增加额外时间开销。在 3.2 节基于现存问题的考虑，受到数据结构中图的存储结构邻接表的启发，提出了一种新的索引结构：不确定字符串邻接表 USAL。USAL 可以在不确定字符串 $X$ 的基础上直接建立，不需要依据预设阈值 $\tau_{min}$ 从 $X$ 中枚举极大子串，从而不确定字符串的处理规模不会增加，同时也能够应对查

询概率阈值 $\tau$ 在范围 0 至 1 任意变化的查询。本节主要介绍如何利用 USAL 索引回答不确定字符串匹配问题。

### 3.3.1 GUSM 算法思想

本节在 USAL 索引的基础上结合贪心思想和范围最值查询提出了一种高效的字符串匹配算法(Greedy Unertain String Matching,GUSM)。当对于一个不确定字符串 $X$ 建立完成 USAL 后，给定一个模式查询$(P,\tau)$，可以依据确定模式串 $P$ 的首字符 $P[0]$ 在 USAL 上定位到其所对应的字母表数组元素，在该数组元素后链接了 $P[0]$在每个偏移处出现的概率，最朴素的想法就是自左至右依次去检验 $P$ 是否在该偏移处出现，但是通过对前面介绍的 USAL 索引观察后发现，有两类偏移位置是不需要检验的，分别是在偏移处满足条件 $Pr(P[0])=0$ 或者 $Pr(P[0])<\tau$ 时。由以上分析可知，当偏移处满足 $Pr(P[0])\geq\tau$ 时，这个偏移才是需要检验的偏移，进而在 USAL 索引上提出了积极字符和消极字符的概念。

**定义 3.1 积极字符和消极字符**：给定一个长度为 $n$ 的不确定文本串 $X$，一个长度为 $m(m\leq n)$的确定模式串 $P$ 及一个查询概率阈值 $\tau(0<\tau\leq 1)$，如果有 $0\leq i\leq n-m$，$1\leq j\leq m$，使得 $X[i+j]=P[j]$ 且 $Pr(P[j])\geq\tau$，则称 $P[j]$ 在 $X$ 上出现，$P[j]$ 为积极字符。反之，当 $X[i+j]=P[j]$ 且 $Pr(P[j])<\tau$ 时，则称 $P[j]$ 在 $X$ 上不出现，$P[j]$ 为消极字符。

在图 2-2 中，给定一个查询的确定模式串 $P$ 和概率阈值 $\tau$ 分别为：(AA,0.5)。当忽略查询概率阈值 $\tau$ 时，确定模式串 $P$ 在不确定字符串 $X$ 中一共出现了两次，偏移集合为{0,3}。当偏移 $i=0$ 时，其中 $Pr(P[0])=Pr(A)=0.3<\tau$，此时 $P[0]$是一个消极字符，因为 $Pr(P)=Pr(P[0])\times Pr(P[1])=0.3\times 0.6=0.18<\tau$，所以 $i=0$ 不是一个有效偏移。当偏移 $i=3$ 时，$Pr(P)=Pr(P[0])\times Pr(P[1])=0.5\times 1=0.5\geq\tau$，则 $i=3$ 是一个有效偏移，此时 $P[0]$和 $P[1]$均为积极字符。综上所述，有效偏移集合为{3}。

**定理 3.1**：给定一个长度为 $n$ 的不确定文本串 $X$，一个长度为 $m(m\leq n)$的确定模式串 $P$ 及一个查询概率阈值 $\tau(0<\tau\leq 1)$，如果有 $0\leq i\leq n-m$，$1\leq j\leq m$，使得 $X[i+j]=P[j]$，如果 $i$ 是一个有效偏移，则有效偏移 $i$ 处出现的确定模式串 $P$ 只包含积极字符。

证明：依据匹配概率公式(2-4)，$P$ 的出现概率 $Pr(P)$的乘积因子都是对应字符出现的概率 $\pi_i(P[i])$。因为 $\pi_i(P[i])\in[0,1]$，所以出现概率 $Pr(P)$随着乘积因子的增加呈非递增变化，假设 $P$ 中包含了一个消极字符，有 $Pr(P[j])<\tau$，那么一定有 $Pr(P)<\tau$。当 $i$ 是一个有效偏移时，满足 $Pr(P)\geq\tau$，因此 $P$ 中只可以包含积极字符。

上文提到过了两类不需要检验的无效偏移位置，即确定模式串 $P$ 中的首字符满足 $Pr(P[0])=0$ 或者 $Pr(P[0])<\tau$ 的位置。依据定义 3.1，可以将这两类字符位置归类为消极字符的位置，不确定字符串匹配问题就进一步转换为寻找确定模式串 $P$ 出现的有效偏移 $i$ 的集合，并且在有效偏移 $i$ 处，保证确定模式串 $P$ 的首字符 $P[0]$是一个积极字符。依据定理 3.1，在不确定字符串匹配时，只需要检验在偏移 $i$ 处，确定模式串 $P$ 的首字符 $P[0]$是否是积极字符就可以过滤大量无效偏移。

一个朴素想法就是在 USAL 索引中找到确定模式串 $P$ 的首字符 $P[0]$对应的数组元素后面链接的字符出现概率表，在字符出现概率表上从左至右依次尝试检验，消极字符出现的位置一定是无效位置，积极字符出现的位置有可能是有效偏移位置，仍需要进行下一步的验证。这样做有一个缺陷，就是如果简单从左至右扫描，积极字符与消极字符通常会穿插出现，这样消极字符也需要比较，需要花费一定的时间。有一种方法可以避免这个缺陷，就是对 USAL 索引中的字符出现概率表进行排序，在排序后的字符出现概率表上扫描，当出现消极字符时，就可以提前结束扫描过程。排序算法有许多种，但是在一般情况下，对一组无序的数排序最快花费的时间为 $O(n\log_2 n)$。

贪心思想是在每一步都做出当时看起来最佳的选择，也就是说，它总是做出局部最优的选择，寄希望这样的选择能导致全局最优解。通过结合贪心思想可以对于前文提到缺陷提出另一种解决方案。由于概率 $Pr(P)$随着乘积因子的增加呈非递增变化，可以在 USAL 索引中的字符出现概率表上选择当前区间中 $Pr(P[0])$最大的位置，并将这个选择看作当时最佳的选择。当选择了一个位置后，该位置就将当前的区间划分为两个较小区间，再分别从两个较小区间内选择 $Pr(P[0])$最大的位置，直到遇见消极字符的位置，算法的执行过程就可以提前结束，这样就能找到一组 $Pr(P[0])\geq\tau$ 的偏移集合。依据定理 3.1，很显然不确定字符串匹配问题的有效偏移集合一定包含在根据上述方法得到的候选结果集合内。完成这个算法步骤的关键一步，就是如何才能从一个区间内快速定位到元素值最大的位置，这个问题可以通过范围最值查询(Range Maximum Query, RMQ)来解决。

给定一个长度为 $n$ 的数组 $A$，范围最值查询问题是指以 $O(1)$时间返回指定下标区间范围内的最值。例如，给定一个数组 $A=\{5,3,4,1,7,6\}$，当指定查询数组下标区间范围为[2,5]时，由于数组的下标从 0 开始，范围内包含的元素有{4,1,7,6}，则很明显数组元素 1 的下标位置就是本次查询的答案，结果为 RMQ(2,5)=3。解决给定区间内

最值查询可以有许多种方法，但是 RMQ 问题困难的地方在于如何以 $O(1)$时间来回复查询，大部分解决 RMQ 问题方法的思想是首先对数组 $A$ 进行预处理，建立一个简洁的索引结构，然后当给定查询范围后，通过对索引结构进行检索来快速回复查询结果的位置。目前解决 RMQ 问题的最佳解决方法是 optimalRMQ[45,46]，详细方法见参考文献，由此给出以下定理 3.2。

**定理** 3.2：给定一个长度为 $n$ 的数组 $A$，只需要对 $A$ 进行 $O(n)$时间的预处理，建立一个占用空间为 $O(n)$的索引结构，就能够以 $O(1)$时间回答 $A$ 上的任意区间最值查询 $RMQ_A(l,r)$。其中 $0\le l\le r<n$。

将贪心思想和范围最值查询结合后就得到了不确定字符串匹配算法 GUSM，相对于排序后扫描的解决方法，GUSM 算法通过建立一个辅助数组，提高了算法的执行效率。依据定理 3.2，可以通过辅助查询数组在 USAL 中的字符出现概率表上以 $O(1)$时间回答当前划分区间内 $Pr(P[0])$最大的位置。经过上述过程处理后可以得到一个候选结果集合，依据定理 3.1 可知，有效偏移位置 $i$ 处，出现的确定模式串 $P$ 只能包含积极字符，值得注意的是，这并不是一个充要条件，即如果在偏移 $i'$处，确定模式串 $P$ 包含的都是积极字符，偏移 $i'$不一定是一个有效偏移。例如，给定模式查询(AT,0.4)，在一个偏移 $i'$处，出现{(A,0.4),(T,0.5)}，但是由于 $Pr(P)=0.4\times0.5=0.2<0.4$，所以偏移 $i'$不是一个有效偏移。所以需要对候选结果集合进行一个验证，剔除掉干扰项，最后返回有效偏移集合。

### 3.3.2 GUSM 过滤策略

通过 3.2 节提出的索引结构 USAL 和 3.3 节提出的匹配算法 GUSM，给定一个模式查询$(P,\tau)$，经过 GUSM 处理后可以得到一个候选结果集合，因为里面还有一些不是有效偏移的干扰项，可能会使验证阶段花费较多的时间，所以需要一些过滤策略，使得进入验证阶段的结果集合规模尽量小一些，本节针对候选结果集合中的干扰项提出一些过滤策略。例如给定模式查询(AT,0.4)，通过 GUSM 算法处理完毕后，当在偏移 $i'$出现{(A,1),(T,0.1)}或者{(A,1),(G,1)}时，很明显偏移 $i'$是需要剔除的干扰项。其原因是上节提出的 GUSM 只能保证在偏移 $i$ 处，确定模式串 $P$ 的首字符是一个积极字符，并没有考虑到 $P$ 除首字符以外的字符和相应出现概率，这就导致了候选结果集合规模相对较大，对于这个问题在稍后的第 4 章会提出一个更有效的解决方法，在本节先提出两种基于 GUSM 设计的过滤策略。

本节提出的过滤策略主要目的是在进入概率阈值验证阶段前，使得通过 GUSM 算法得到的候选结果集合规模变小，前面分析了候选结果集合中干扰项的出现原因，提出了两种过滤策略：最小概率字符过滤和随机选择字符过滤。

最小概率字符过滤依据这样一个想法，不确定字符串$X$是从有限字母表集合$\Sigma$中枚举得到的，$X$中包含的字符种类数量有很大可能是不均匀的，当给定一个确定模式串$P$后，可以将$P$中出现次数最少的字符位置视作一个特殊位置，当考虑$P$是否在一个偏移$i$处出现时，优先考虑这个特殊位置的字符是否是积极字符，有很大的可能只需进行一次比对就可以判断出偏移$i$是否是一个有效偏移。可以在USAL中的字母表数组里添加一个计数域*count*，*count*统计了数组元素后链接的字符出现概率表中非零概率字符的个数，在图3-1的基础上添加计数域*count*后，如图3-3所示。当给定一个模式查询$(P,\tau)$后，找到$P$中出现概率最小的字符位置，在添加一个偏移$i$进入到候选结果集合前，先对最小概率字符出现的位置进行检验，这样就能有效过滤一部分干扰项。

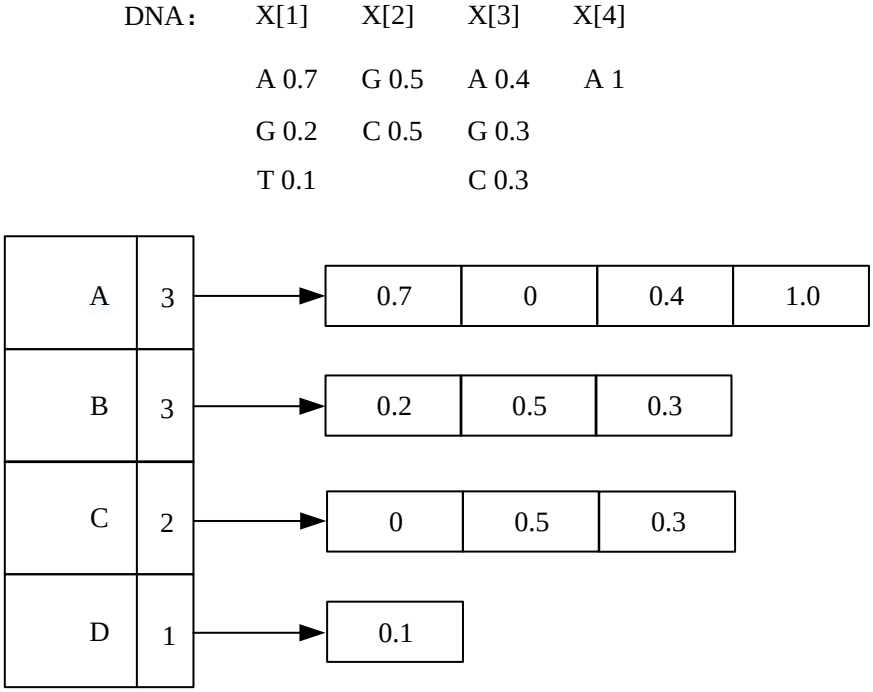



图3-3 在USAL上添加*count*域

随机选择字符过滤是在最小概率字符过滤的基础上结合随机思想得到的。最小概率字符过滤是以确定模式串 $P$ 中相对于不确定字符串 $X$ 中出现次数最少的字符位置作为优先考虑的位置，随机选择字符过滤是将这个优先考虑的位置随机化，将 $P$ 中除首字符 $P[0]$以外剩余子串的位置中随机选择一个作为优先考虑的位置。随机思

想秉承的依据是避免最坏的情况发生，一个经典的例子就是将随机思想与快速排序算法结合的随机快速排序算法。快速排序是众多排序算法中较为便捷的一种，最好情况的时间复杂度为 $O(n\log_2 n)$，最坏情况的时间复杂度为 $O(n^2)$，制约快速排序算法的因素有两个：排序的数据和枢轴的选取。这两个因素与随机都有一定的关系，排序的数据越接近无序，枢轴将数据划分得越均匀，排序的速度就越快。应用随机思想后，可以有效避免不利于快排处理的情况，提升算法执行效率。与最小概率字符过滤同理，随机选择字符过滤很大可能进行一次比对就可以判断出偏移 $i$ 是否是一个有效偏移，从而有效过滤一部分干扰项。

### 3.3.3 GUSM 算法描述

通过 3.3.1 节对 GUSM 算法思想的介绍，了解到 GUSM 算法需要一个预处理阶段来构建解决范围最值查询问题的辅助查询数组 RMQArray。RMQArray 中的每个数组元素是利用 optimalRMQ 算法构建的对应于 USAL 索引中字符出现概率表部分的索引对象，依据定理 3.2，通过该索引对象在 USAL 索引中的字符出现概率表上能够以 $O(1)$时间回答当前划分区间内 $Pr(P[0])$最大的位置。辅助查询数组的构建方法如算法 3.2 所示。

**算法**3.2　getRMQArray算法

**Input:** *sigma* ← the size of *USAL*. *alphabet*

**Output:** auxiliary query array *RMQArray*

1: build an array *RMQArray*[*sigma*]

2: **for** *i* ← 0 to *sigma*-1 **do**

3: **if** *alphabet*[*sigma*].*link.size* is not equal to 0 **then** //相应字符出现概率表非空

4: *RMQArray*[*i*] ← *optimalRMQ*(*alphabet*[*sigma*].*link*) //构建RMQ索引

5: **return** *RMQArray*

根据定理 3.1，在有效偏移处，确定模式串 $P$ 只能包含积极字符，但是这并不是一个充要条件，所以用 GUSM 算法匹配处理得到的偏移结果集合只是一个候选结果集合，对于这个候选结果集合还需要进一步验证，将干扰项从候选结果集合中剔除掉，从而得到有效偏移结果集合。为了在进入验证阶段前，减小候选结果集合的规模，在 3.3.2 节提出了两种过滤策略：最小概率字符过滤和随机选择字符过滤。最小概率字符过滤需要在 USAL 索引中的数组部分添加一个计数域 *count*，*count* 用来

统计不确定字符串 $X$ 中不同字符种类的非零概率字符数量，当向 USAL 索引中的字符出现概率表插入字符出现概率时，也要更新对应数组位置的 *count*，即在算法 3.1 中第 9 行后添加“*++alphabet*[*letter*–*Σ*[0]].*count*”。最小概率字符过滤的具体过程可以分为两个阶段，第一阶段是在确定模式串 $P$ 中寻找到最小概率字符的出现位置，第二阶段是将找到的最小概率字符出现位置作为优先位置进行积极字符验证，分别如算法 3.3 和算法 3.4 所示。

**算法**3.3　findMinIndex算法

**Input:** query pattern *p*, query probability threshold $\tau$

**Output:** minimum probability position of *p minIndex*

1:　*min* ← *alphabet*[*p*[0] – *Σ*[0]].*count*

2:　*minIndex* ← 0

3:　**for** *i* ← 1 to *p*.size – 1 **do**

4:　　**if** *min* > *alphabet*[*p*[*i*] – *Σ*[0]].*count* **then**

5:　　　*min* ← *alphabet*[*p*[*i*] – *Σ*[0]].*count*　　//更新最少出现字符个数

6:　　　*minIndex* ← *i*　　//更新最小概率字符在*p*中位置

9:　**return** *minIndex*

**算法**3.4　MinProFiltering算法

**Input:** query pattern *p*, query probability threshold $\tau$, Maximum probability position of *Pr*(*p* [0]) *max*, minimum probability position of *p minIndex*

**Output:** bool variable *flag*

1:　*flag* ← false

2:　**if** *alphabet*[*p*[*minIndex*] - *Σ*[0]].*link*[*max* + *minIndex*] is positive **then**

3:　　*flag* ← true

4:　**return** *flag*

在算法 3.3 中，最小概率字符过滤的第一阶段是根据给定确定模式串 $P$ 在 USAL 中的计数域 *count* 上进行比较，对应算法 3.3 的第 3-6 行，返回出现概率最小字符的位置。在算法 3.4 中，最小概率字符过滤的第二阶段是对 findMinIndex 算法返回的位置进行验证，确定该位置的字符是否是一个积极字符，并将判断的结果返回。当判断

的结果为 false 时，就可以确定该位置是一个无效偏移并忽略，否则，需要保存这个偏移结果。

随机选择字符过滤是将最小概率字符过滤中的优先考虑位置随机化，将确定模式串 $P$ 中除首字符 $P[0]$以外剩余子串的位置中随机选择一个作为优先考虑的位置，对该位置进行积极字符的验证，其原理是应用了随机的思想来尽量避免最坏情况的发生，其具体过程如算法 3.5 所示。

**算法**3.5　RandomFiltering算法

**Input:** query pattern *p*, query probability threshold *τ*, Maximum probability position of *Pr*(*p* [0]) *max*

**Output:** bool variable *flag*

1: *flag* ← false
2: *length* ← the size of *p*
3: *checkBit* ← randomly select a position from *p*[1] to *p*[*length* - 1]　//随机选择位置
4: **if** *alphabet*[*p*[*checkBit*] - *Σ*[0]].*link*[*max* + *checkBit*] is positive **then**
5: 　*flag* ← true
6: **return** *flag*

3.3.1 节介绍的 GUSM 算法思想是基于贪心思想的，贪心思想是每一步都做出当时看起来最佳的选择，需要逐步返回 USAL 索引中字符出现概率表当前划分区间中出现概率最大的位置，很明显这是一个递归的过程，如算法 3.6 所示。

**算法**3.6　recursiveRMQ算法

**Input:** left boundary *l*, right boundary *r*, query pattern *p*, query probability threshold *τ*, auxiliary query array *RMQArray*, minimum probability position of *p minIndex*

**Output:** approximate occurrence set *result'*

1: *max* ← *RMQArray*[*p*[0] - *Σ*[0]].*query*(*l*, *r*)
2: **if** *alphabet*[*p*[0] - *Σ*[0]].*link*[*max*] is positive **then**
3: 　**if** *MinProFiltering*(*p, τ, max, minIndex*) and *RandomFiltering*(*p, τ, max*)
4: 　　*result'* ← *max*
5: 　*recursiveRMQ*(*l*, *max*-1, *p, τ, RMQArray, minIndex*)
6: 　*recursiveRMQ*(*max*+1, *r*, *p, τ, RMQArray, minIndex*)

在算法3.6中，算法的第1行是根据给定确定模式串*P*的首字符找到RMQ辅助查询

数组中对应的索引对象，返回当前划分区间内的最大出现概率字符位置。第2行是对该位置是否是一个积极字符做出判断，如果不是积极字符，意味着继续递归的子区间内也不会包含积极字符，整个算法过程可以提前结束。第3行应用了前面介绍过的最小概率字符过滤和随机选择字符过滤两种过滤策略对候选结果的位置做一个筛选，从而达到减小候选结果集合规模的目的。第5,6行对应了向由*max*位置划分的两个子区间去递归执行算法。

将前面介绍的算法结合起来就得到了基于贪心思想的不确定字符串匹配算法GUSM，其具体过程如算法3.7所示。

**算法**3.7　GUSM算法

**Input:** query pattern *p*, query probability threshold $\tau$, auxiliary query array *RMQArray*

**Output:** exact occurrence set *result*

1: *length* ← *alphabet*[*p*[0] - $\Sigma$[0]].*link.size* – 1

2: *minIndex* ← *findMinIndex*(*p,* $\tau$)

3: *result'* ← *recursiveRMQ*(0, *length, p,* $\tau$, *minIndex*)

4: **for** each element *i* in *result'* **do**　　//验证候选结果集合

5: 　　**if** *i* is a occurrence **then**

6: 　　*result* ← *i*

7: **return** *result*

在算法 3.7 中，GUSM 算法可以划分为两个阶段：模式匹配阶段和有效偏移验证阶段。模式匹配阶段是根据前面介绍过的算法过程，利用贪心思想每次在 USAL 的字符出现概率表上当前划分区间中选择 *Pr*(*P*[0])最大的位置，在选择的过程中应用两种过滤策略忽略一部分无效偏移，最后得到使确定模式串 *P* 包含积极字符的候选结果集合，有效偏移验证阶段是从找到的候选结果集合中，剔除掉干扰项，找到正确的有效偏移集合。

模式匹配阶段具体对应于算法 3.7 中的第 2-3 行，有效偏移验证阶段对应于算法的第 4-6 行，具体的执行过程如下。第 2 行是用 findMinIndex 算法从确定模式串 *P* 中找到出现概率最小字符的位置，第 3 行是用 recursiveRMQ 算法从不确定字符串 *X* 中当前划分的区间内递归枚举出确定模式串 *P* 的首字符出现概率最大的位置。如算法 3.6 所示，递归枚举的过程中应用了最小概率字符过滤和随机选择字符过滤，通过

这两种过滤策略可以有效减小候选结果集合的规模，第 4-6 行是对候选结果集合中包含的有效偏移进行验证，最后返回有效偏移集合。

例 3.1，给定一段 DNA 的不确定序列 $X$={{(C,0.7),(T,0.3)}, {(A,0.8),(T,0.2)}, {(A,0.1),(G,0.1),(T,0.8)}, {(A,0.1),(G,0.2),(T,0.7)}, {(A,1)}}，使用不确定字符串匹配算法 GUSM 匹配模式查询(TA,0.5)。

首先，应用第一种字母表处理方案将有限字母表集合 $\Sigma$ 中的字符差值较大的字符 G 和 T 分别转换为 B 和 D，替换后的字符序列 $X$，如图 3-4 所示。

| DNA： | X[1] | X[2] | X[3] | X[4] | X[5] |
|---|---|---|---|---|---|
| | C 0.7 | A 0.8 | A 0.1 | A 0.1 | A 1 |
| | D 0.3 | D 0.2 | B 0.1 | B 0.2 | |
| | | | D 0.8 | D 0.7 | |

图3-4 替换字符后的DNA不确定序列

为转换后的 $X$ 建立不确定字符串索引 USAL，建立完毕的 USAL 如图 3-5 所示。

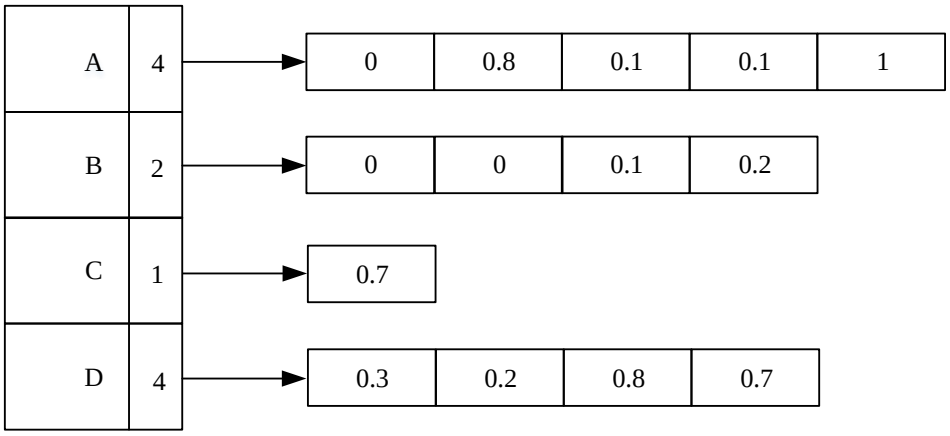



图3-5 替换字符后的DNA序列$X$构建USAL

最后，使用不确定字符串匹配算法 GUSM 在 USAL 上匹配模式查询(TA,0.5)。依据算法 3.7，首先用 findMinIndex 算法找到确定模式串 $P$=TA 中出现概率最小字符 T，其在 $P$ 中的位置是 0。当算法 3.7 执行到第 3 行时，从由 T 转换得到的 D 所对应的字符出现概率表中，返回当前区间内概率最大值 0.8 的偏移，即偏移 $i=2$。由于当偏移 $i$=2 时，A 不是一个积极字符，所以偏移 $i$=2 能够通过随机选择字符过滤策略判断为无效偏移。偏移 $i$=2 将当前长度区间划分为两个子区间，分别是{0.3,0.2}和{0.7}。依据算法 3.6 所示，首先递归处理左区间{0.3,0.2}，发现当前区间内最大值为 0.3<0.5，

所以可以提前结束对左区间的执行过程，然后递归处理右区间{0.7}，可以将 0.7 的偏移加入到候选结果集合 *result'*中，即 *result'*={3}。最后在算法 3.7 的 4-6 行进行验证，因为 $Pr(P)=0.7\times1=0.7>0.5$，从而得到有效偏移集合 *result*={3}。

### 3.3.4 GUSM 算法分析

如算法 3.1 所示，索引结构 USAL 建立的过程需要将不确定字符串 $X$ 扫描一遍，因此其最坏时间复杂度为 $O(|\Sigma|n)$，其中$|\Sigma|$表示 $X$ 的字母表集合大小，$n$ 表示 $X$ 的长度。USAL 索引由字母表数组部分和其后链接的字符出现概率表两部分组成，所以其最坏空间复杂度同为 $O(|\Sigma|n)$。在许多字符串匹配的应用中，比如生物信息学，字母表的大小是远远小于其长度的，在这种情况下，可以将建立 USAL 的时间和空间复杂度近似看作 $O(n)$。GUSM 算法为了实现范围最值查询需要构建一个 RMQ 辅助查询数组，依据定理 3.2，最坏情况下构建 RMQ 辅助查询数组的时间和空间复杂度均为 $O(|\Sigma|n)$，同样在小字母表的情况下，可以近似看作 $O(n)$。以上可以看作是 GUSM 算法的索引构建阶段，该阶段整体的时间和空间复杂度可以近似看作 $O(n)$。GUSM 算法的时间复杂度为 $O(m+occ\ m)$，其中 $m$ 表示确定模式串 $P$ 的长度，$occ$ 表示候选偏移出现的次数。如算法 3.7 所示，首先需要用 findMinIndex 算法找到其中小概率字符的出现位置，时间复杂度为 $O(m)$，随后用 RMQ 可以 $O(1)$时间返回当前区间内概率最大的积极字符位置，最后对得到的候选偏移结果集合进行验证，最坏情况下时间复杂度为 $O(occ\ m)$。

## 3.4 本章小结

本章首先分析了现有不确定字符串匹配算法存在建立索引困难、索引规模大和查询受限的问题，进而提出了新的索引结构 USAL。其次，在 USAL 上提出了积极字符与消极字符的概念，并结合贪心思想和范围最值查询提出了一种高效的字符串匹配算法 GUSM。再次，逐步介绍了 GUSM 的算法思想及步骤，并给出了一个演示样例详细说明算法的执行过程。最后，给出了 USAL 索引和 GUSM 算法的复杂度分析。

# 第 4 章 基于位映射的不确定字符串匹配

第 3 章首先分析了现有不确定字符串索引方法存在的问题，提出了一种新的索引结构 USAL，并在 USAL 上结合贪心思想和范围最值查询提出了一种高效的不确定字符串匹配算法 GUSM。依据定理 3.1，可以将不确定字符串匹配问题进一步转换为寻找使确定模式串 $P$ 只包含积极字符的偏移集合问题，GUSM 可以在 USAL 中的字符出现概率表上选择当前区间中 $Pr(P[0])$最大的位置，这样可以得到一个包含干扰项的候选结果集合。为了使候选结果集合的规模减小，提高积极字符位置的过滤效果，进一步基于 GUSM 设计了两个过滤策略：最小概率字符过滤和随机选择字符过滤，通过应用这两种策略可以进一步过滤掉使 $P$ 包含了消极字符的无效偏移。

## 4.1 问题分析

在 3.1 节的问题分析中，列举了不确定字符串若干较难处理的特性，其中一个就是在不确定字符串 $X$ 上任意指定一个位置 $i$，以位置 $i$ 起始的子串出现概率也是会随着长度的增加呈现没有规律的非递增变化。由于这个特性使得 GUSM 应用贪心思想时也变得困难，因为确定模式串 $P$ 是一个连续的子串，如果只是考虑从 USAL 中的字符出现概率表上选择 $Pr(P[i])(0\leq i<m)$最大的位置，这些位置很可能是离散的，这也就是 GUSM 只将贪心思想应用于 $P$ 的首字符 $P[0]$的原因。GUSM 可以保证在偏移 $i$ 处，$P$ 的首字符是一个积极字符，但是不能保证 $P$ 除 $P[0]$以外不包含消极字符，从而导致候选结果集合规模相对较大，基于 GUSM 设计的两个过滤策略可以在一定程度上缓解这个问题，但是不能从根本解决这个问题。

根据上述存在的问题， 为了进一步过滤掉候选结果集合中使 $P$ 包含消极字符的无效偏移，达到减小候选结果集合规模的目的，本章结合 Bitmap 思想对 USAL 进行了改进，并在改进后的索引上提出了对应的不确定字符串匹配算法。下面首先介绍本文改进的索引，随后介绍其相应的不确定字符串匹配算法。

## 4.2 BI 索引

Bitmap 是一种(*key*,*value*)的映射关系，用一个位(bit)来标记某个元素对应的 *value*，而将该元素作为 *key*。bit 即比特，是目前计算机系统里面数据的最小单位，以 bit 作为存储数据的单位，可以大大节省存储空间规模。例如，需要存储从 0 到 7 的十个

整数，用整型数组实现时，至少需要 8×4=32Byte(以 C++语言标准为例，一个整型数据占用 4 个字节)。但是当使用 Bitmap 来存储时，只需要一个长度为 8 的位数组，将插入的整数对应的 bit 位置 1 即可，整体的存储空间只需要 8 个 bit，8bit 等于 1Byte，可见应用了 Bitmap 后可以使内存节省 32 倍。

一个 bit 位只有两种状态 0 或 1，在 USAL 索引中的字符出现概率表中，当字符的概率不为 0 时，表示该位置有字符出现，用状态 1 表示，否则，用状态 0 表示。在许多高级程序设计语言中也有现成的类库来方便程序员管理一系列的 bit 位，例如：C++的 bitset 和 Java 的 BitSet。将图 3-1 中的 USAL 用 Bitmap 思想改进后，就得到了基于位映射的不确定字符串索引：位图索引(Bitmap Index,BI)，如图 4-1 所示。

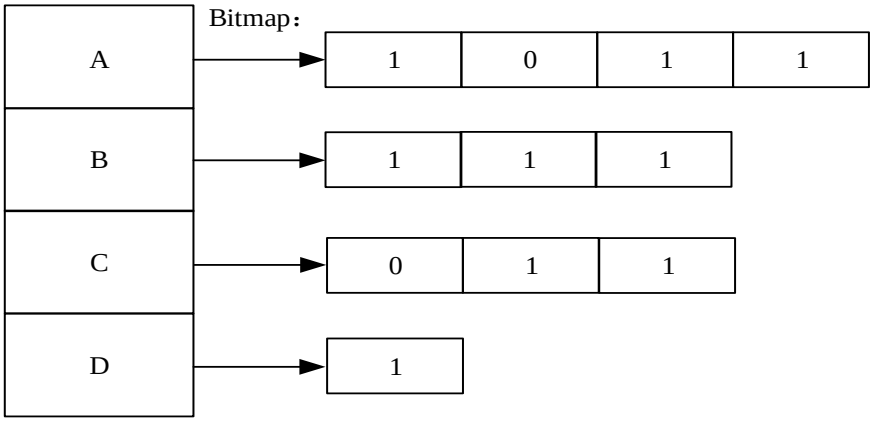



图4-1 用Bitmap对USAL改进后的BI

## 4.3 BUSM 算法

在 4.1 节提出了 GUSM 算法在解决不确定字符串匹配问题时存在的一个问题，因为 GUSM 应用了贪心思想，每一步都做出当时看起来最佳的选择，由于不确定字符串子串的出现概率随着长度增加会呈现没有规律的非递增变化，给定一个确定模式串 $P$，简单地从 USAL 的字符出现概率表中选择 $Pr(P[i])(0\leq i<m)$最大的位置，会导致选择位置是离散的，由于没有考虑 $P$ 作为子串的连续特性，使得 GUSM 只将贪心思想应用于 $P$ 的首字符 $P[0]$。

依据定理 3.1，在有效偏移 $i$ 处，$P$ 一定不会包含消极字符。因此，需要考虑一种方法能够更加有效过滤掉使 $P$ 包含消极字符的无效偏移，进一步增强字符串匹配阶段的过滤效果，达到减少概率阈值验证阶段候选结果集合规模的目的。通过结合

Bitmap 思想，可以改善 USAL 的存储空间规模，得到一个小规模的位图索引 BI，本节主要介绍如何利用 BI 回答不确定字符串匹配问题。

### 4.3.1 BUSM 算法思想

通过对 GUSM 算法存在问题的分析，如果想更加有效过滤掉使确定模式串 $P$ 包含消极字符的无效偏移，依据定理 3.1，除了要考虑在一个偏移 $i$ 处，$P$[0]位置是积极字符，还需要尽可能保证 $P$ 中包含的其它字符也是积极字符。解决这个问题的关键在于不但要求 $P$ 中只能出现积极字符，由于子串的连续特性，这些积极字符的位置也必须是连续的。Bitmap 除了前面介绍的可以有效节省存储规模外，还有另一个优势，那就是可以做高性能的位运算。以寻找积极字符位置、模式子串的连续性和位运算的想法作为出发点，可以想到一种可以在 BI 上进行不确定字符串匹配的算法(Bitmap Uncertain String Matching,BUSM)。

在改进后的不确定字符串索引结构 BI 中，BI 的字符出现概率表与 USAL 略有不同，其中使用一个 bit 来表示该位置是否有字符出现，bit 只有两种状态：0 和 1，当字符在该位置出现概率不为 0，则用 1 来表示，否则，用 0 来表示。当给定一个长度为 $m$ 的确定模式串 $P$=$P$[0] $P$[1]…$P$[$m$-1]时，根据 $P$ 中的末尾字符 $P$[$m$-1]可以从 BI 中确定其字符概率出现表，首先将该表中的每个 bit 都向左移动一位，然后与 $P$[$m$-2]的字符概率出现表做相与运算，迭代这个位运算的过程，直到和 $P$ 中的首字符 $P$[0]的字符概率出现表相与运算，经过上述过程处理后可以在 $P$[0]的字符概率出现表得到一个候选结果集合，这个候选结果集合可以有效过滤掉使确定模式串 $P$ 中包含了出现概率为 0 的无效偏移，在这个候选结果集合的基础上进行验证，剔除掉干扰项，最后就可以返回有效偏移集合。

### 4.3.2 BUSM 优化策略

通过前面介绍的索引结构 BI 和不确定字符串匹配算法 BUSM，给定一个模式查询($P$,$\tau$)，可以通过 BUSM 算法处理得到一个候选结果集合。这个候选结果集合可以有效过滤掉使 $P$ 中包含字符出现概率为的无效偏移，但是仍然会包含一些不是有效偏移的干扰项，干扰项过多会导致候选结果集合验证阶段的效率变差，为了使进入验证阶段的候选结果集合的规模变小，与 GUSM 算法类似，本节对 BUSM 算法设计优化策略。例如给定模式查询(AT,0.4)，当在偏移 $i$’出现类似{(A,1), (T,0)}时，经过 BUSM

算法处理后，可以将这样的无效偏移过滤掉。但是当在偏移 $i'$出现{(A,1), (T,0.2)}时，此时 $Pr(P)=1\times0.2=0.2<0.5$，偏移 $i'$也是可以剔除的无效偏移，类似这样的无效偏移不能被前面的处理过程过滤。其原因是上节介绍的 BUSM 只能够处理得到使 $P$ 出现概率 $Pr(P)\neq0$ 的偏移集合，并没有考虑模式查询中的查询概率阈值 $\tau$。

BI 按照 bit 位来存储数据，相较于初始的 USAL 可以有效节省存储空间规模，由于 bit 位只存在两种状态：0 和 1，利用这两种状态可以对字符在该位置的出现概率是否为 0 进行区分，但是这样做没有考虑字符出现概率与查询概率阈值 $\tau$ 的关系，导致大量无效偏移的干扰项包含在 BUSM 算法处理得到的候选结果结合中。由于 BI 存储规模小的特性，使得考虑牺牲一些空间来提高概率阈值验证阶段的效率的想法变得合理。综合考虑 BI 的小规模特性与在匹配阶段利用查询概率阈值 $\tau$ 两个方面，本节提出了一种多维位图索引(Multiple Bitmap Index,MBI)。

多维位图索引 MBI 依据这样一个想法，BI 的字符概率出现表中的 bit 位只能包含两种状态：0 和 1，对于在区间(0,1]内任意变化的查询概率阈值 $\tau$，简单地使用这两种状态在一个 BI 中来标记区分消极字符的位置是困难的，这是因为一个字符位置是否是积极字符会随着着查询概率阈值 $\tau$ 发生改变。例如，在一个不确定字符串 $X$ 中的偏移 $i$ 处，包含一个不确定字符(A,0.5)。如果给定模式查询(A,0.4)，由于 0.5>0.4，所以这个偏移 $i$ 可以作为一个有效偏移结果保留，这就要求在 BI 中这个字符位置的 bit 位要置 1。但是当模式查询变化为(A,0.7)时，由于 0.5<0.7，所以这个偏移 $i$ 就作为一个无效偏移需要舍弃，要求 BI 中该字符位置的 bit 位要置 0。使用一个 BI 去应对不同的查询概率阈值 $\tau$ 是不现实的，因为不同的 $\tau$ 要求 BI 的字符概率出现表的 bit 位状态不尽相同。基于这种情况，提出了多维位图索引 MBI，其思想是可以设定一个维度值 $d$，在每一个维度根据不同的概率阈值 $\tau_d$ 去建立 BI，当给定一个模式查询$(P,\tau)$后，根据查询概率阈值 $\tau$ 从多维位图索引 MBI 中选择一个适合 BI，再用 BUSM 算法去处理，这样就可以根据给定的查询概率阈值 $\tau$ 过滤掉使确定模式串 $P$ 包含消极字符的无效偏移，达到减小候选结果集合规模的目的。

### 4.3.3 BUSM 算法描述

BI 是对 USAL 应用 Bitmap 改进后得到的，BI 也看作数组和链表两个部分，数组对应了不确定字符串的字母表集合 $\Sigma$，数组元素后面链接了其对应表示字符的字符出现概率表。BI 的构建过程也与 USAL 类似，其构建过程分为两个阶段，先构建

字母表数组部分，其后根据不确定字符串 $X$ 中的字符概率信息，在字符出现概率表相应 bit 位置 1。上一节提出的多维位图索引 MBI 要求可以为指定的维度概率阈值 $\tau_d$ 建立 BI，在算法 3.1 的基础上修改后，具体过程如算法 4.1 所示。在 BI 的实现过程中，字符出现概率表部分可以使用 bitset 替代 vector(以 C++语言为例)。

**算法**4.1 BIC算法

**Input:** the size of $\Sigma$ *sigma*, uncertain string *X*, probability threshold $\tau_d$

**Output:** Bitmap Index *BI*

```
1:    build an array alphabet[sigma] for Σ
2:    length ← the size of X
3:    for i ← 0 to length-1 do
4:        position ← i
5:        letter ← X[i].char        //记录X[i]中的字符
6:        probability ←X[i].pro     //记录X[i]中的字符出现概率
7:        for j ← the size of alphabet[letter – Σ[0]] to position do
8:            if j is equal to position and probability ≥ τd then     //插入字符出现概率
9:                add 1 to alphabet[letter – Σ[0]].link
10:           else
11:               add 0 to alphabet[letter – Σ[0]].link
12:   return BI
```

相较于算法 3.1，算法 4.1 主要修改的地方在于的第 8 行和第 9 行，在第 8 行插入字符的出现概率时，需要用维度概率阈值 $\tau_d$ 进行一个条件判断，由于 BI 中的字符出现概率出现表用 bit 位存储，所以将第 9 行原本插入字符出现概率值的操作改为置 1 操作。在算法 4.1 的基础上，应用前面小节提到的优化策略，建立一个多维位图索引 MBI，具体过程如算法 4.2 所示。

**算法** 4.2 MBIC 算法

**Input:** the size of $\Sigma$ *sigma*, uncertain string *X*, dimension *d*

**Output:** Mulitple Bitmap Index *MBI*

```
1:    τd ← 0
2:    Δτ ← 1.0/d      //计算相邻维度的概率阈值差值
3:    for i ← 0 to d and τd ≤ 1.0 do
```

4: $BI \leftarrow$ BIC($sigma$, $X$, $\tau_d$)

5: $\tau_d \leftarrow \tau_d + \Delta\tau$

6: add $BI$ to $MBI$

7: **return** $MBI$

在算法 4.2 中，第 2 行根据设定的维度值 $d$ 计算相邻维度的概率阈值差值，例如设定维度值 $d$=10，则相邻维度的概率阈值差值为 1.0/10=0.1。根据第 3-6 行可以构建 $d$+1=11 个维度的 MBI 索引，其不同维度对应的 $\tau_d$={0,0.1,0.2,…,1.0}。当给定一个模式查询($P$,$\tau$)时，可以根据查询概率阈值 $\tau$ 从 MBI 中选择一个合适的 BI 作为索引，例如设定 $\tau$=0.7，可以选择 $\tau_d$=0.7 所对应的 BI 来用 BUSM 算法处理。在建立多维位图索引 MBI 后，应用 BUSM 算法回答模式查询($P$,$\tau$)，其具体过程如算法 4.3 所示。

**算法** 4.3　BUSM 算法

**Input:** query pattern $p$, query probability threshold $\tau$

**Output:** exact occurrence set *result*

1: $length \leftarrow$ the size of $p$

2: $d \leftarrow MBI.size$ - 1

3: $\Delta\tau \leftarrow 1.0/d$　　//计算相邻维度的概率阈值差值

4: **if** $\tau \geq \tau_d$ **then**　　//根据查询概率阈值 $\tau$ 选择合适维度

5: $d \leftarrow \tau/\Delta\tau$

6: **else**

7: $d \leftarrow 0$

8: $result' \leftarrow MBI[d].alphabet[p[0] - \Sigma[0]].link$

9: **for** $i \leftarrow length - 1$ to 1 **do**

10: $tmp \leftarrow MBI[d].alphabet[p[i] - \Sigma[0]].link$

11: $tmp \leftarrow tmp >> i$

12: $result' \leftarrow result' \,\&\, tmp$

13: $occ \leftarrow$ the *position* of each 1 in result′　　//收集 1 的位置信息作为候选结果

14: **for** each element $i \in occ$ **do**　　//验证候选结果集合

15: **if** $i$ is a occurrence **then**

16: $result \leftarrow i$

17: **return** *result*

在算法 4.3 中，第 4-7 行是根据模式查询$(P,\tau)$中的查询概率阈值 $\tau$ 从多维位图索引 MBI 中选择一个合适的 BI 来作为索引，第 9-13 行是在 BI 上经过高效的位运算来得到一个候选结果集合 *occ*，最后的 14-16 行对将候选结果集合进行验证，只保留有效偏移的位置。

例 4.1，给定一段 DNA 的不确定序列 $X$={{(C,0.7),(T,0.3)}, {(A,0.8),(T,0.2)}, {(A,0.1),(G,0.1),(T,0.8)}, {(A,0.1),(G,0.2),(T,0.7)}, {(A,1)}}，使用不确定字符串匹配算法 BUSM 匹配模式查询(TA,0.5)。

与构建索引 USAL 类似，首先应用第一种字符表处理方案将有限字母表集合 $\Sigma$ 中的 G 和 T 分别转换为 B 和 D，替换后的字符序列 $X$，见图 3-4。其次为转换后的 $X$ 建立位图索引 BI，建立完毕的 BI 如图 4-2 所示。

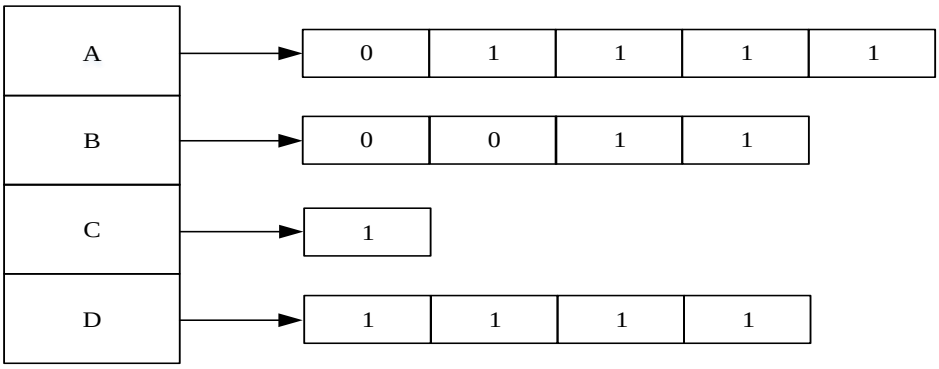



图4-2 替换字符后的DNA序列$X$构建BI

应用前面对于 BI 设计的优化策略后，可以进一步建立一个多维位图索引 MBI。设定维度值 $d$=10，则相邻维度的概率阈值差值 $\Delta\tau$=1.0/10=0.1，则 MBI 一共有 11 个维度(包含 0 号)，建立完整的 MBI 后，如图 4-3 所示。

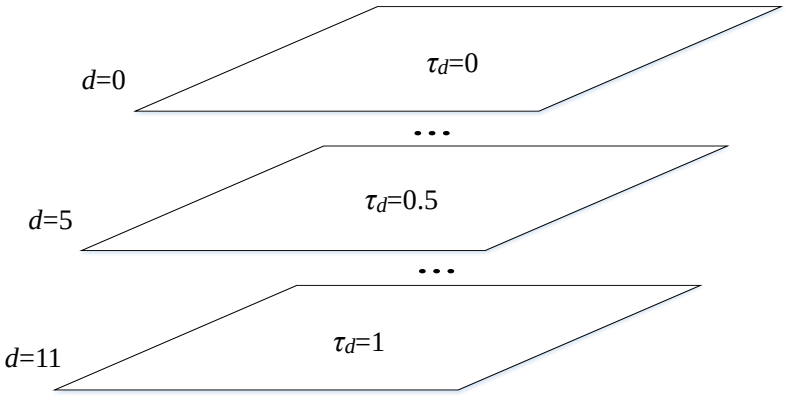



图4-3 $d$=10时，建立的MBI

因为模式查询为(TA,0.5)，由查询概率阈值$\tau$可知，需要选择维度概率阈值$\tau_d$=0.5所对应的BI，如图4-4所示。

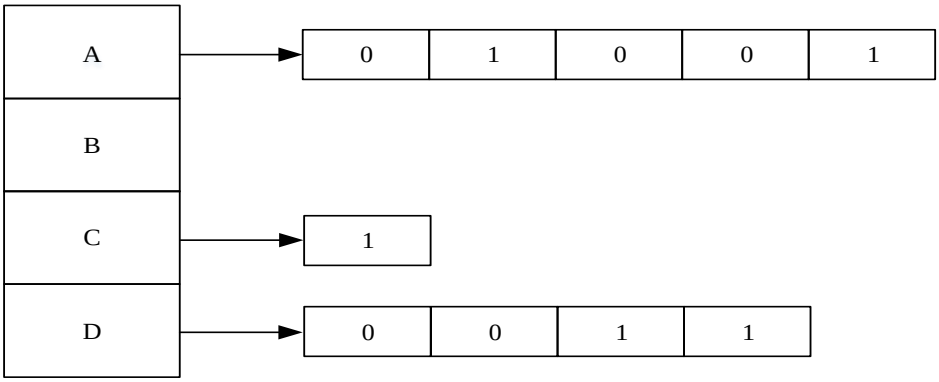



图4-4 $\tau_d$=0.5时，从MBI中选择的BI

最后，使用不确定字符串匹配算法 BUSM 在 MBI 中 $\tau_d$=0.5 的 BI 上匹配模式查询(TA,0.5)。依据算法 4.3，当 BUSM 算法执行完第 4-7 行后，由于 $d=\tau/\Delta\tau$=0.5/0.1=5，所以选择 MBI 中 5 号 BI 作为索引页。第 8-12 行对 BI 中与确定模式串 $P$ 相关联的索引行执行高效的位运算，依照从末尾字符开始的顺序，由图 4-4 所示，从字符 A 所对应的字符出现概率表中，取出二进制序列“01001”，向低位方向移动 1 位后得到序列“10010”，与由字符 T 转换得到的 D 所对应的字符出现概率表中的二进制序列“00110”相与后，得到序列“00010”，第 13 行将位运算后得到的二进制序列转换为对应的候选结果集合为{3}。在 14-16 行进行查询概率阈值 $\tau$ 的验证，因为 $Pr(P)$=0.7×1=0.7>0.5，从而得到有效偏移集合 $result$={3}。

### 4.3.4 BUSM 算法分析

如算法 4.1 所示，位图索引 BI 的构建过程与 USAL 类似，都只需要将不确定字符串 $X$ 扫描一遍，因此最坏时间复杂度为 $O(|\Sigma|n)$，其中$|\Sigma|$表示 $X$ 的字母表集合大小，$n$ 表示 $X$ 的长度。空间复杂度也与 USAL 一致，最坏情况为 $O(|\Sigma|n)$。与 USAL 每个位置都存储每种字符出现的概率不同，BI 在字符概率出现表中对于字符出现的概率用 bit 位来标记，节省了索引的空间规模。针对类似生物信息学的情况，其字母表大小远小于其长度的情况，建立 BI 的时间和空间复杂度都可以近似看作 $O(n)$。多维位图索引 MBI 是以 BI 存储规模较小为前提考虑得到的一种优化策略，维度值 $d$ 决定了 MBI 包含 BI 的索引页数，所以 MBI 的时间和空间复杂度为 $O(|\Sigma|nd)$。综合考虑查询概率阈值 $\tau$ 出现的范围与存储规模的大小可以选取一个合适的 $d$，但通常情况下

*d* 不会很大。与 BI 同理，可以将 MBI 的时间和空间复杂度近似看作 *O*(*n*)。BUSM 算法对比于 GUSM 算法而言，在字符串匹配阶段考虑了更多使确定模式串 *P* 出现积极字符的条件，使得进入概率阈值阶段的候选结果集合规模更小，并在匹配过程中使用了高效的位运算，显然 BUSM 算法性能会更加高效。

## 4.4 本章小结

本章主要介绍了基于位映射思想的不确定字符串匹配算法 BUSM。为了更加有效过滤掉使确定模式串 P 包含消极字符的无效偏移，综合考虑子串的连续特性，对索引结构 USAL 用 Bitmap 的思想进行改进，提出了一种小规模索引 BI。由于 BI 小规模的特性，提出了优化策略 MBI，然后在应用 MBI 的基础上逐步介绍了 BUSM 算法思想及步骤，最后通过一个演示样例来详细说明了算法的执行过程。

# 第 5 章 实验及结果分析

## 5.1 引言

本文通过对现有的不确定字符串匹配算法进行深入分析，发现了现有方法在索引构建阶段会由于不确定字符串数据难以处理的特性，导致存在索引构建时间长与规模大的问题，并且现有的不确定索引构建方法通常会在建立索引阶段使用一个预设概率阈值 $\tau_{min}$，当给定模式查询$(P,\tau)$后，一旦出现查询概率阈值 $\tau<\tau_{min}$ 的情况，就需要重新建立索引从而带来额外的时间开销。针对上述问题，提出了空间代价为字符种类乘字符串长度的不确定字符串邻接表 USAL，并对 USAL 应用 Bitmap 思想进一步改进得到小规模位图索引 BI，并在两种索引结构上分别提出了高效的不确定字符串匹配算法 GUSM 和 BUSM 以及相应的优化策略。本文实验的重点是对不确定字符串匹配算法 GUSM 和 BUSM 在多个不同特征的真实数据集和人工合成数据集进行实验。下面首先介绍实验环境和数据集的相关情况，然后与现有基于后缀树与后缀数组设计的不确定字符串匹配算法分别从索引规模大小，索引构建时间，模式匹配效率三个方面进行对比，以验证本文所提出的不确定字符串匹配算法的高效性。

## 5.2 实验环境和数据集

本文实验基于的计算机硬件平台是 Intel Core i5 主频为 2.6GHz 的 CPU，运行内存大小为 8GB，操作系统为 Windows10。实验代码编程语言为 C++，运行环境为 Microsoft Visual Studio 2017。

由于本文所设计的数据结构和匹配算法的出发点是针对生物信息学领域中字符种类较少的特点，所以真实数据集采用的是 DNA 和蛋白质序列。所用的数据集相关信息如表 5-1 所示。

表 5-1 源数据集

| 数据集 | 类型 | $\|\Sigma\|$ | 规模大小 |
|---|---|---|---|
| Pa_dna | DNA 序列 | 4 | 6.14MB |
| Pa_protein | 蛋白质序列 | 20 | 1.81MB |
| Random | 随机序列 | 26 | 1MB |

在表 5-1 中，Pa_dna 是从 NCBI(National Center for Biotechnology Information)下载的铜绿假单胞菌 PAO 1 染色体全基因组的 DNA 序列，DNA 的字母表集合 $\Sigma$={A,T,C,G}，字母表大小为 4，每一个字符表示一种脱氧核苷酸。Pa_protein 是用数据集 Pa_dna 翻译得到的蛋白质序列，蛋白质的字母表集合 $\Sigma$={A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}，字母表大小为 20，每一个字符表示一种氨基酸。Random 是人工随机生成的字母数据集，其字母表集合 $\Sigma$={A-Z}，字母表大小为 26。

从以上三个源数据集中随机抽取出长度 $n$=320K 的字符串作为实验数据集，然后分别对其字符位置进行随机模糊处理，得到本文实验中所使用的 8 个不确定字符串数据集，如表 5-2 所示。

表 5-2 不确定字符串数据集

| 数据集 | 类型 | $\|\Sigma\|$ | 模糊系数 | 规模大小 |
|---|---|---|---|---|
| Unc2_pa_dna | DNA 序列 | 4 | 2 | 7.56MB |
| Unc3_pa_dna | DNA 序列 | 4 | 3 | 10.4MB |
| Unc3_pa_protein | 蛋白质序列 | 20 | 3 | 10.8MB |
| Unc5_pa_protein | 蛋白质序列 | 20 | 5 | 16.4MB |
| Unc9_pa_protein | 蛋白质序列 | 20 | 9 | 27.7MB |
| Unc3_random | 随机序列 | 26 | 3 | 10.9MB |
| Unc5_random | 随机序列 | 26 | 5 | 16.6MB |
| Unc9_random | 随机序列 | 26 | 9 | 28MB |

在表 5-2 中，明确列出了每个不确定字符串数据集对应的类型和字母表大小，模糊系数表示经过模糊处理后会在一个字符位置出现相应字符数量的概率分布。例如，设定模糊系数为 3，表示在数据集里的某个位置字符可能出现的个数为 3。本文实验所涉及到的实验参数信息，如表 5-3 所示。

表 5-3 实验相关参数

| 实验参数 | 说明 |
|---|---|
| $n$ | 不确定字符串长度 |
| $m$ | 查询确定模式串长度 |
| $\tau_{min}$ | 现有方法索引阶段预设概率阈值 |
| $d$ | 多维位图索引 MBI 的维度值 |

## 5.3 性能比较与分析

本节主要从索引规模大小、索引构建时间和模式匹配时间三个方面对比了现有方法和本文所提出的方法。实验中涉及到的相关算法主要有：基于后缀树与后缀数组的不确定字符串索引构建算法(General Substring Index Construction,GSIC)和不确定字符串匹配算法(General Substring Query Answering,GSQA)，本文提出的不确定字符串索引 USAL 的构建算法 USALC 和不确定字符串匹配算法 GUSM，位图索引 BI 的构建算法 BIC 和不确定字符串匹配算法 BUSM。当 GUSM 算法不使用所设计的两种过滤策略时记为 GUSM-，当 BUSM 算法使用 MBI 优化策略时记为 BUSM+。

### 5.3.1 索引构建规模分析

现有的 GSIC 算法需要对不确定字符串建立由后缀树和后缀数组构成的索引结构，后缀树和后缀数组都是查找效率比较高的数据结构，但是它们都是基于确定字符串提出的。但是不确定字符串由于一个位置中可以出现多个字符的概率分布，使得这两种数据结构都无法基于不确定字符串直接构建。文献[38]中定义了一种特殊不确定字符串，该特殊不确定字符串相较于一般处理的不确定字符串而言，一个位置只限定出现一个字符的概率分布，并引用文献[44]中的方法对一般不确定字符串进行预处理，预处理的过程是首先设定一个概率阈值 $\tau_{min}$，然后将一般不确定字符串中满足 $\tau_{min}$ 的极大子串枚举并拼接，从而得到其定义特殊不确定字符串。这样做会造成不确定字符串长度规模的增加，如图 5-1 所示。

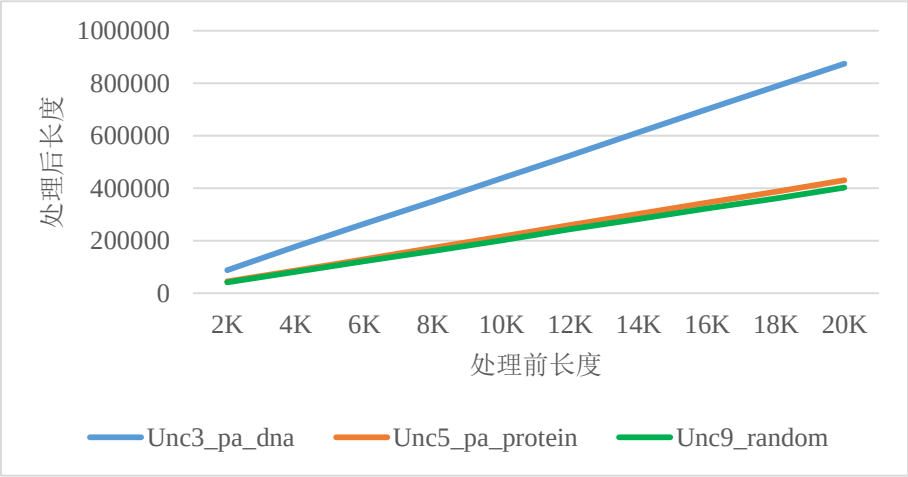



图5-1 不确定字符串预处理枚举长度变化($\tau_{min}$=0.1)

在图 5-1 中，分别从三个不同特征的数据集截取范围 2K-20K 长度的不确定字符串，使用文献[44]中的方法来预处理枚举，可以看到随着不确定字符串处理规模的增加，处理得到的特殊不确定字符串的规模也近似呈线性增长，因为该方法可以将转换后的长度规模限定为 $O((1/\tau_{min})^2 n)$[44] 308，不同特征的数据集即使在相同范围内长度的增加规模也不同。在字符串长度规模变大的基础上建立后缀树和后缀数组会造成时间和空间剧烈消耗。本文提出的不确定字符串索引结构 USAL 和 BI 的空间代价均为字符种类乘字符串长度，当字符种类较少时，长度规模可以近似为 $O(n)$，不会进一步使不确定字符串长度规模增加，从而保证了索引结构良好的空间效率。

GUSM 算法的索引部分由两部分组成：不确定字符串邻接表 USAL 和范围最值查询辅助数组 RMQArray，索引结构占用的空间等于这两部分的总和。BUSM 算法只需要位图索引 BI，所以只需要考虑 BI 占用的空间即可。当 8 个数据集长度规模达到 320K 时，GSIC 算法在本文实验环境下已经无法得出结果，原因在前文已经作了说明，本文所提出的两种不确定字符串索引空间占用情况如图 5-2 所示。

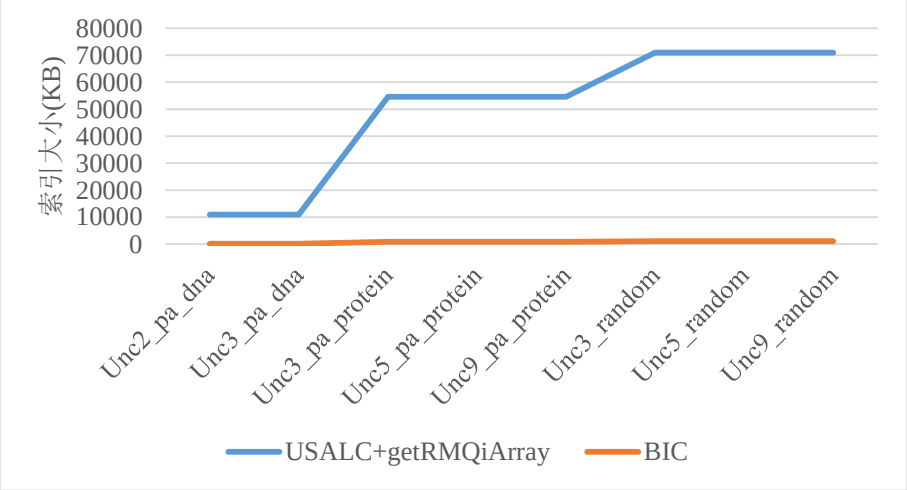



图5-2 本文提出索引的空间占用对比($n$=320K)

在图 5-2 中，8 个数据集的长度规模都是 320K，随着字符种类的增加，GUSM 算法的索引部分明显呈梯度上升，因为所使用的 USAL 索引的空间代价为字符种类乘字符串长度，当数据集的字符种类确定后，不同的模糊系数变化对于构建的索引没有影响。BI 索引由于使用了 Bitmap 思想，大大缩减了索引的存储空间，由于 C++中 bitset 模板类的缺陷性，即在声明时需要指定其大小，所以设定 BI 中每个字符出现概率表的长度都是 320K，即使这样处理字符种类和模糊系数最大的 Unc9_random 时，其索引空间占用也才达到 1040KB，而建立 USAL 和 RMQArray 占用的空间大约是

BI 的 68 倍左右，因此 BI 相较于 USAL 而言具备更好的可扩展性。

### 5.3.2 索引构建时间分析

GSIC 算法索引构建时间主要由三部分组成：后缀树、后缀数组和连乘概率数组，所以 GSIC 算法的构建时间是以上三部分构建时间的总和。GUSM 算法的索引由两部分组成：不确定字符串邻接表 USAL 和范围最值查询辅助数组 RMQArray，所以 GUSM 算法的索引构建时间计算等于这两部分构建时间的总和。BUSM 算法只需要位图索引 BI，所以只需要计算 BI 的构建时间。分别从 8 个数据集截取 20K 长度，GSIC 算法的索引构建时间变化如图 5-3 所示。

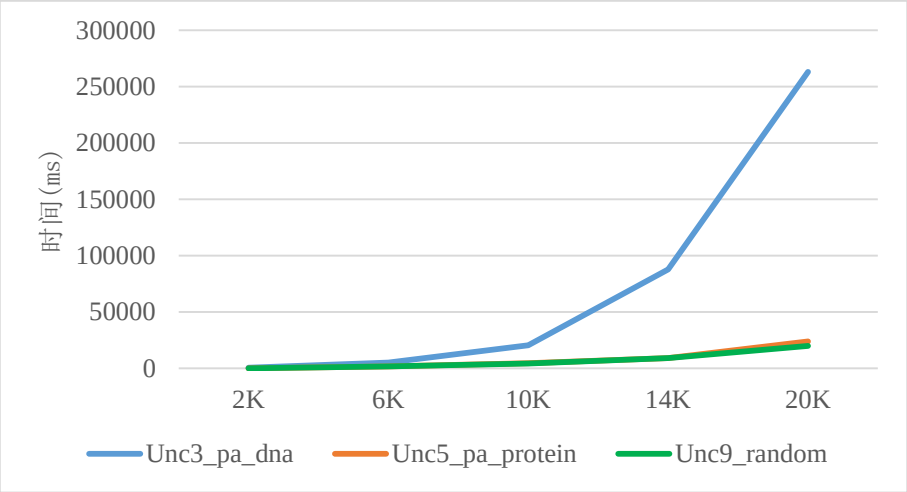



图5-3 GSIC算法索引构建时间变化($\tau_{min}$=0.1)

由图 5-3 所示，GSIC 算法的随着长度增加会大幅消耗时间，三个数据集索引构建时间变化与图 5-1 中不确定字符串预处理后规模的趋势相同，由于相同范围内不同特征的数据集在索引构建时间上也有较大差别，说明 GSIC 算法的索引构建时间取决于其预处理后不确定字符串的长度规模。从 8 个数据集中截取 20K 长度，索引构建时间对比结果如表 5-4 所示，其中“—”表示时间超过 5 分钟。

表 5-4 不同算法索引构建时间对比（$n$=20K） (ms)

| 数据集 | GSIC | USALC+getRMQArray | BIC |
|---|---|---|---|
| Unc2_pa_dna | — | 64+25=89 | 66 |
| Unc3_pa_dna | 263221 | 82+25=107 | 86 |
| Unc3_pa_protein | 247695 | 108+141=249 | 114 |
| Unc5_pa_protein | 20719 | 154+148=302 | 157 |

表 5-4(续表)

| 数据集 | GSIC | USALC+getRMQArray | BIC |
|---|---|---|---|
| Unc9_pa_protein | 22694 | 253+126=379 | 270 |
| Unc3_random | 175536 | 137+155=292 | 149 |
| Unc5_random | 22824 | 189+170=359 | 187 |
| Unc9_random | 21624 | 256+156=412 | 276 |

在表 5-4 中，可见基于后缀树和后缀数组索引的 GSIC 算法在索引构建时间上远远大于本文提出的索引构建算法，这是由于经过预处理后会导致原本的不确定字符串规模增加。在 $\tau_{min}$=0.1 的情况下，经由文献[44]中的方法预处理后，8 个数据集处理后的规模由 20K 依次增长为：1984640，874712，714725，429895，408384，691511，417450 和 402698。由于对长文本建立后缀树会导致效率较低，本文提出的 GUSM 算法的索引结构是基于不确定字符串邻接表 USAL 和范围最值查询数组 RMQArray 两个部分，由表 5-4 中可知，其索引构建时间要远远优于现有的 GSIC 算法，从索引构建时间的分配比重上看，构建 RMQArray 的时间一般会小于构建 USAL 的时间。在 GUSM 算法的索引构建总时间中去除去构建 RMQArray 的时间后，剩余的 USALC 算法执行时间很接近于 BIC 算法的执行时间，这是因为 BI 是对 USAL 中的字符概率出现表部分用 Bitmap 改进后得到的，其索引的构建过程类似。当不确定字符串被预处理后，导致了字符串规模的增长，GSIC 算法的构建时间效果会变差，下面分别在 8 个长度规模为 320K 的完整数据集上对比本文提出索引构建算法，也会得到与上文提到类似特征的结果，如表 5-5 所示。

表 5-5 本文提出索引构建算法时间对比（$n$=320K） (ms)

| 数据集 | USALC+getRMQArray | BIC |
|---|---|---|
| Unc2_pa_dna | 1064+387=1451 | 1125 |
| Unc3_pa_dna | 1491+371=1862 | 1539 |
| Unc3_pa_protein | 1815+1738=3553 | 1835 |
| Unc5_pa_protein | 2459+1784=4243 | 2485 |
| Unc9_pa_protein | 3810+1829=5639 | 4043 |
| Unc3_random | 1867+2247=4114 | 2053 |
| Unc5_random | 2526+2274=4800 | 2794 |
| Unc9_random | 3959+2374=6333 | 4062 |

### 5.3.3 模式匹配效率分析

经过前面的分析，本文提出的不确定字符串匹配算法在索引构建时间与空间上都要优于基于后缀树和后缀数组的不确定字符串匹配算法 GSQA。当对数据集建立完索引后，给定一个模式查询$(P,\tau)$，模式匹配时间就是指除去建立索引结构时间外，找到所有有效偏移的时间。当模式查询不同时，不同特征的确定模式串 $P$ 和出现概率 $\tau$ 多少都会影响一次查询的时间，为了更好衡量不同不确定字符串匹配算法之间的匹配性能，使用从相应数据集的字母表中随机选择字符并加权后生成 100 个查询模式，如表 5-6 所示。

表 5-6 查询模式测试用例

| 测试用例 | 类别 | $\|\Sigma\|$ | $\|P\|$ |
|---|---|---|---|
| 3_query100_dna | DNA 序列 | 4 | 3 |
| 3_query100_protein | 蛋白质序列 | 20 | 3 |
| 5_query100_random | 随机序列 | 26 | 5 |

在表 5-6 中，列出了相应测试用例的特征。例如，测试用例 3_query100_dna 表示是从 DNA 序列的字母表集合中随机生成的 100 个 $m$=3 的模式查询集合。在没有额外说明的情况下，设 $\tau_{min}$=0.1，$d$=10。分别从 8 个数据集中截取 20K 长度，将 100 个查询模式的时间总和作为衡量标准，对比结果如表 5-7 所示，其中“—”表示构建索引时间超过 5 分钟。

表 5-7 不同算法对 3_query100_dna 查询模式时间对比（$n$=20K） (ms)

| 数据集 | GSQA | GUSM | GUSM- | BUSM | BUSM+ |
|---|---|---|---|---|---|
| Unc2_pa_dna | — | 2304 | 2484 | 27 | 13 |
| Unc3_pa_dna | 68 | 1806 | 2062 | 76 | 20 |
| Unc3_pa_protein | 1 | 537 | 569 | 5 | 3 |
| Unc5_pa_protein | 2 | 529 | 553 | 12 | 3 |
| Unc9_pa_protein | 2 | 524 | 546 | 11 | 2 |
| Unc3_random | 1 | 313 | 317 | 6 | 3 |
| Unc5_random | 1 | 288 | 307 | 7 | 2 |
| Unc9_random | 1 | 284 | 296 | 7 | 2 |

在表 5-7 中，将各类算法纵向比较，各类算法的匹配时间都呈递减的趋势，其原因是随着数据集模糊系数和字母表规模的增加，每次查询中符合条件的结果规模在减少，从而导致整体模式匹配时间的减少。将各类算法横向比较，发现 GSQA 算法和 BUSM+模式匹配效率最高，之所以 GSQA 算法在数据集 Unc3_pa_dna 上花费了 68ms，是因为在该数据集上满足测试用例 3_query100_dna 的有效偏移结果较多。值得注意的是 BUSM+算法可以根据查询模式中的概率阈值 $\tau$，过滤更多包含消极字符的候选偏移，如果设置一个稍大些的 $d$，增加多维位图索引 MBI 的索引维度，BUSM+算法的匹配效率还会进一步提升。对比两种本文提出的不确定字符串匹配算法 GUSM 和 BUSM，发现 BUSM 算法整体的模式匹配性能要好，这是因为 BUSM 在匹配过程中考虑了确定模式串 $P$ 作为子串的连续特性，并在匹配过程中使用了高效的位运算，从而达到了更好的模式匹配性能。两种算法在使用对应设计的优化策略后，匹配效果上都会显著提升，特别是 BUSM+可以将 BUSM 的性能提高一倍以上。

测试用例 3_query100_protein 表示该是从蛋白质序列的字母表集合中随机生成的 100 个长度为 3 的模式查询集合，测试用例 5_query100_random 表示是从随机序列的字母表集合中随机生成的 100 个长度为 5 的模式查询集合，在该两个测试用例上也得到了与表 5-7 类似的结果，如表 5-8 和 5-9 所示。

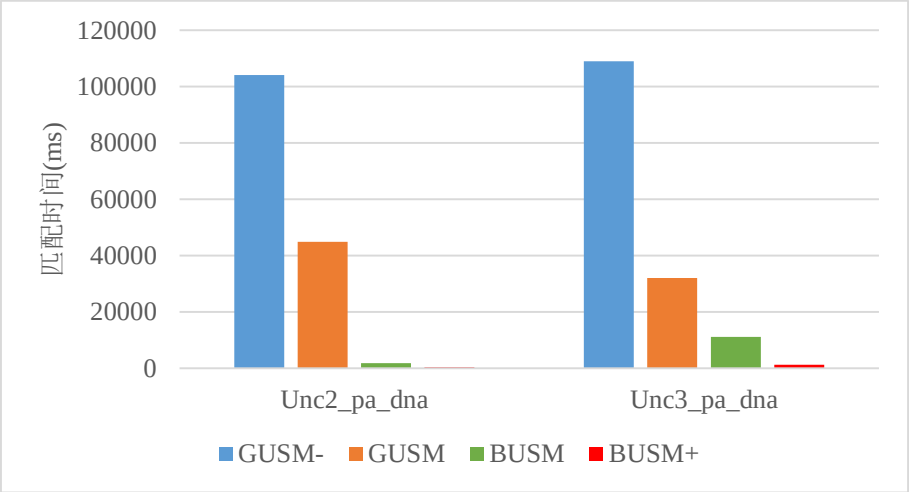
表 5-8 不同算法对 3_query100_protein 查询模式时间对比（$n$=20K） (ms)

| 数据集 | GSQA | GUSM | GUSM- | BUSM | BUSM+ |
|---|---|---|---|---|---|
| Unc3_pa_protein | 1 | 464 | 490 | 6 | 2 |
| Unc5_pa_protein | 1 | 465 | 483 | 9 | 3 |
| Unc9_pa_protein | 2 | 460 | 484 | 13 | 3 |
| Unc3_random | 1 | 325 | 319 | 7 | 3 |
| Unc5_random | 1 | 313 | 332 | 9 | 3 |
| Unc9_random | 1 | 305 | 323 | 9 | 2 |

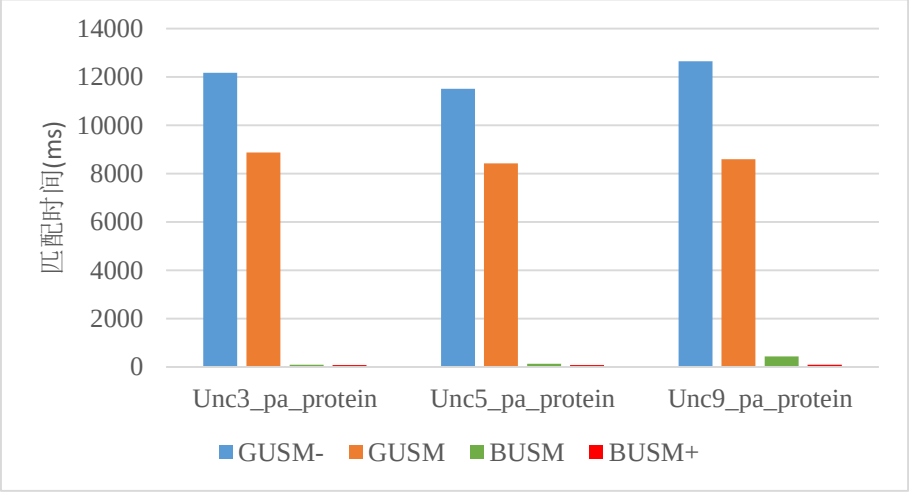
表 5-9 不同算法对 5_query100_random 查询模式时间对比（$n$=20K） (ms)

| 数据集 | GSQA | GUSM | GUSM- | BUSM | BUSM+ |
|---|---|---|---|---|---|
| Unc3_random | 1 | 338 | 364 | 1 | 1 |
| Unc5_random | 1 | 346 | 396 | 3 | 1 |
| Unc9_random | 1 | 355 | 374 | 5 | 1 |

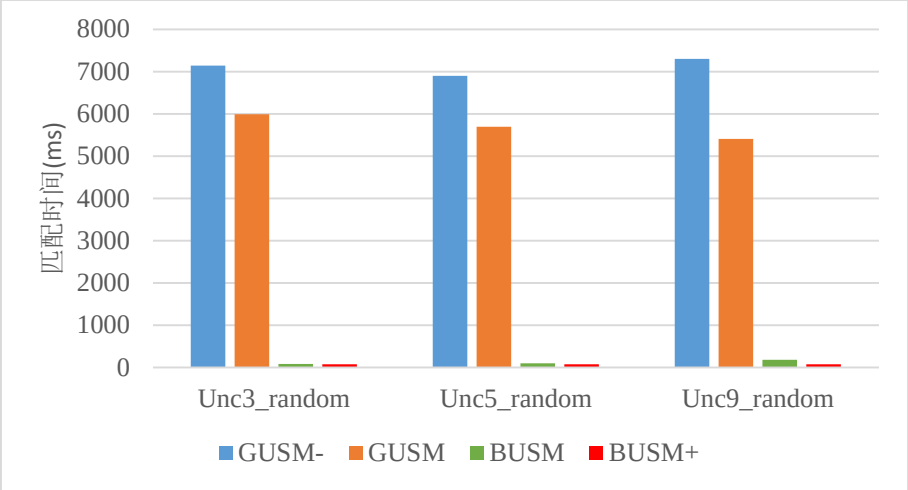
根据以上测试得到的结果，GSQA 算法和本文提出的 BUSM 及 BUSM+算法都可以达到很好的模式匹配效果。但是 GSQA 算法的匹配效果好的前提是在预处理后字符串长度规模增长的基础上建立后缀树与后缀数组，其需要建立的索引空间较大，时间较长，用大量空间换取了模式匹配时间。本文所提出的两种算法都不会改变原来字符串的长度规模，在字符种类较少的情况下，GUSM 算法构建索引的时间和空间复杂度可以近似为 $O(n)$，采用少量的空间换取了模式匹配时间，尤其是 BUSM 算法使用了 Bitmap 的思想，进一步提高了空间利用率，同时位运算保持了较好的模式匹配效率，因此 GUSM 算法和 BUSM 算法具备更好的可扩展性。将测试用例 3_query100_dna 在 8 个长度规模为 320K 的完整数据集上用本文提出的两种算法进行模式匹配对比，结果如图 5-4 所示。




a) DNA 序列数据集




b) 蛋白质序列数据集




c) 随机序列数据集

图 5-4 本文算法对 3_query100_dna 查询模式时间对比（$n$=320K）

在图 5-4 中，将数据集规模扩大后，本文提出的两种匹配算法的匹配时间整体上依然呈现出表 5-7、表 5-8 和表 5-9 的趋势。BUSM 算法对比 GUSM 算法依然保持优势，BUSM 算法平均要比 GUSM 算法快几十倍到近百倍。GUSM 算法过滤策略的效果进一步明显，尤其在 DNA 数据集上，模式匹配时间提高了一倍以上。

## 5.4 本章小结

本章在 Microsoft Visual Studio 2017 环境下用 C++编程实现了基于后缀树和后缀数组的不确定字符串匹配算法和本文提出的两种算法，并在 5 个生物信息学序列的真实数据集和 3 个人工合成数据集上进行了对比测试。主要从索引构建规模、索引构建时间和模式匹配效率三个方面进行了对比，并且对其产生差别的原因进行了详细分析。通过对实验结果进行客观分析比较，结果显示随着数据集的增长，本文提出的算法依然具备高效性和可扩展性。

# 结　论

随着互联网、数字图书馆、大基因组项目的发展，不确定文本数据的规模不断变大。近年来，不确定字符串匹配问题逐渐成为数据挖掘领域中的热点之一。本文针对现有不确定字符串匹配方法存在的索引构建时间长和索引规模大的问题展开研究，提出了相应解决方案，具体研究成果如下：

(1)针对现有方法对不确定字符串枚举预处理后，在字符串长度规模增加的基础上构建后缀树与后缀数组所造成的时间和空间效率低下的问题，提出了空间代价为字符种类乘字符串长度的索引 USAL。在 USAL 索引上定义了积极字符与消极字符的概念，并结合贪心思想和范围最值查询 RMQ 提出了不确定字符串匹配算法 GUSM。GUSM 算法在 USAL 索引上当前划分的区间范围内收集概率最大的积极字符位置作为候选结果集合，再对候选结果集合进行概率阈值的验证，最后返回正确的有效偏移集合。为了提高积极字符位置的过滤效果，给 GUSM 算法设计了两种过滤策略，实验结果表明 GUSM 算法相较于现有方法提高了索引构建的时间和空间效率，两种过滤策略对 GUSM 算法匹配效率的提高达一倍以上。

(2)为了进一步加强字符串匹配阶段的过滤效果，结合 Bitmap 思想对 USAL 索引改进后得到小规模位图索引 BI。BI 索引可以对 USAL 索引上的概率信息进行位映射，进一步提高了 USAL 索引的空间利用率。在 BI 索引上提出了高效的不确定字符串匹配算法 BUSM，BUSM 算法将子串匹配操作转换为高效的位运算，实验结果表明 BUSM 算法可以进一步提高不确定字符串匹配的效率。

(3)为了提高 BUSM 算法在概率阈值验证阶段的效率，提出了多维位图索引 MBI，有效减少了阈值验证阶段候选结果集合的规模，进一步提高了算法执行效率。

本文所提出的不确定字符串匹配算法在索引构建规模、索引构建时间和模式匹配时间方面均取得了一定成果，完成了预期的研究目标，但仍存在有待改进的地方。首先，本文考虑问题的出发点是基于字符种类较少的情况，当字符种类变得很大时，就需要提出更佳的方法来设计索引结构。其次，为了便于分析问题，假设了字符出现概率是互相独立的，现实生活中可能情况更加复杂，字符出现的概率可能会存在关联，这两个问题可以在后续工作中继续研究。

# 参考文献

[1] Liu Y, Liu Q, Liu P, et al. A Factor-Searching-Based Multiple String Matching Algorithm for Intrusion Detection[C]//Proceeding of International Conference on Communications, Sydney, Australia, 2014: 653-658.

[2] Sheikh N, Mustafi K, Mukhopadhyay I. A Unique Approach to Design an Intrusion Detection System using an Innovative String Searching Algorithm and DNA Sequence[C]//Proceeding of the 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference, NY, USA, 2016: 1-9.

[3] Vakili S, Langlois J M P, Boughzala B, et al. Memory-Efficient String Matching for Intrusion Detection Systems using a High-Precision Pattern Grouping Algorithm[C]//Proceeding of the 2016 Symposium on Architectures for Networking and Communications Systems, ANCS 2016, Santa Clara, USA, 2016: 37-42.

[4] Gudur V Y, Acharyya A. Accelerated Reconfigurable String Matching using Hardware-Software Codesign for Computational Bioinformatics Applications[C]//Proceeding of European Conference on Circuit Theory and Design, Catania, Italy, 2017: 1-4.

[5] 唐四薪，尹军，刘艳波. 串匹配算法在生物信息学中的应用[J]. 中国科教创新导刊, 2007(21): 123-123.

[6] 沈学利，张家明. 模式识别的改进算法研究[J]. 微型机与应用, 2009, 28(10): 16-18.

[7] 吴楠，朱怀宏，夏黎春. 一种应用于现代网络搜索引擎的快速串匹配算法[J]. 计算机与现代化, 2003(11): 7-8.

[8] 屠晓，陈国跃. 信函上英文地址的自动识别和翻译[J]. 华东师范大学学报, 2008(3): 83-91.

[9] Imura H, Tanaka Y. Compression and String Matching Method for Printed Document Images[C]//Proceeding of International Conference on Document Analysis and Recognition, Barcelona, Spain, 2009: 291-295.

[10] Peel A, Wirth A, Zobel J. Collection-Based Compression using Discovered Long Matching Strings[C]//Proceeding of the 20th ACM Conference on Information and Knowledge Management, Glasgow, UK, 2011: 2361-2364.

[11] Zhang M, Zhang W, Bu S. A Compression Scheme allowing Direct String Matching on Compressed

Binary Files and Its Applications[J]. International Journal of Wireless and Mobile Computing, 2017, 12(2): 142-146.

[12] Alvarez J R, Skachkov D, Massey S E, et al. DNA/RNA Transverse Current Sequencing: Intrinsic Structural Noise from Neighboring Bases[J]. Frontiers in Genetics, 2015, 6(14):213.

[13] Greenblatt M S. Sequence Variants of Uncertain Significance: What to Do When Genetic Test Results Are Not Definitive[J]. Surgical Oncology Clinics of North America, 2015, 24(4):833.

[14] Sendi M, Omri M N, Abed M. Possibilistic Interest Discovery from Uncertain Information in Social Networks[J]. Intelligent Data Analysis, 2017, 21(6): 1425-1442.

[15] Xu G, Zhang M, Jin H H, et al. Research on the Topological Evolution of Uncertain Social Relations in Opportunistic Networks[C]//Proceeding of IEEE International Conference on Edge Computing. IEEE Computer Society, Honolulu, USA, 2017:202-205.

[16] Ahmed N M, Chen L. An Efficient Algorithm for Link Prediction in Temporal Uncertain Social Networks[J]. Information Sciences, 2016, 331(10):120-136.

[17] Tong Y, Chen L, She J. Mining Frequent Itemsets in Correlated Uncertain Databases[J]. Journal of Computer Science and Technology, 2015, 30(4): 696-712.

[18] Shintani T, Ohmori T, Fujita H. Skip Search Approach for Mining Probabilistic Frequent Itemsets from Uncertain Data[C]//Proceeding of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, Porto, Portugal, 2016: 174-180.

[19] Nie L, Li Z, Qi H, et al. Probabilistic Frequent Itemsets Mining Based on Expectation Bound over Uncertain Database[C]//Proceeding of International Symposium on Pervasive Systems, Algorithms and Networks, Exeter, UK, 2017: 14-19.

[20] Hua M, Pei J, Zhang W, et al. Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data[C]//Proceeding of the 24th International Conference on Data Engineering, Cancun, Mexico, 2008: 1403-1405.

[21] Mo L, Cheng R, Li X, et al. Cleaning Uncertain Data for Top-k Queries[C]//Proceeding of International Conference on Data Engineering, Brisbane, Australia, 2013: 134-145.

[22] Chen T, Chen L, Ozsu M T, et al. Optimizing Multi-Top-k Queries over Uncertain Data Streams[J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(8): 1814-1829.

[23] Li Y, Bailey J, Kulik L, et al. Mining Probabilistic Frequent Spatio-Temporal Sequential Patterns

with Gap Constraints from Uncertain Databases[C]//Proceeding of International Conference on Data Mining, Dallas, USA, 2013: 448-457.

[24] Ge J, Xia Y, Wang J. Towards Efficient Sequential Pattern Mining in Temporal Uncertain Databases[C]//Proceeding of Advances in Knowledge Discovery and Data Mining – 19th Pacific-Asia Conference, Ho Chi Minh Cith, Vietnam, 2015: 268-279.

[25] Muzammal M, Raman R. Mining Sequential Patterns from Probabilistic Databases [J]. Knowledge and Information Systems, 2015, 44(2): 325-358.

[26] Thomas H C, Charles E L, Ronald L R, et al. 算法导论[M]. 潘金贵译, 北京: 机械工业出版社, 2006: 558-559.

[27] Gonnet G H, Baeza-Yates R A. An Analysis of the Karp-Rabin String Matching Algorithm[J]. Information Processing Letters, 1990, 34(5): 271-274.

[28] Knuth D E, Jr J H M, Pratt V R. Fast Pattern Matching in Strings[J]. SIAM Journal on Computing, 1977, 6(2): 323-350.

[29] Boyer R S, Moore J S. A Fast String Searching Algorithm[J]. Communications of the ACM, 1977, 20(10): 762-772.

[30] Sunday D M. A Very Fast Substring Search Algorithm[J]. Communications of the ACM, 1990, 33(8): 132-142.

[31] 胡金柱, 熊春秀, 舒江波, 等. 一种改进的字符串模式匹配算法[J]. 模式识别与人工智能, 2010, 23(1): 103-106.

[32] 王成, 刘金刚. 一种改进的字符串匹配算法[J]. 计算机工程, 2006, 32(2): 62-64.

[33] Amir A, IIiopoulos C, Kapah O, et al. Approximate Matching in Weighted Sequences[C]//Proceeding of the 17th Annual Symposium on Combinatorial Pattern Matching, Barcelona, Spain, 2006: 365-376.

[34] 段枫凯, 吴爱华. 不确定序列子串匹配的概率矩阵算法[J]. 现代计算机, 2016(14): 32-38.

[35] Crochemore M, Hancart C, Lecroq T. Algorithms on Strings[M]. NY, USA: Cambridge University Press, 2007: 146-192.

[36] Iliopoulos C S, Makris C, Panagis Y, et al. The Weighted Suffix Tree: An Efficient Data Structure for Handling Molecular Weighted Sequences and its Applications[J]. Fundamenta Informaticae, 2006, 71(2,3):259-277.

[37] Barton C, Kociumaka T, Pissis S P, et al. Efficient Index for Weighted Sequences[C]//Proceeding of

the 27th Annual Symposium on Combinatorial Pattern Matching, Tel Aviv, Israel, 2016: 4:1-4:13.

[38] Thankachan S V, Patil M, Biswas S, et al. Probabilistic Threshold Indexing for Uncertain Strings[C]//Proceeding of the 19th International Conference on Extending Database Techology, Bordeaux, France, 2016: 401-412.

[39] Weiner P. Linear Pattern Matching Algorithms[C]//Proceeding of the 14th Annual Symposium on Switching and Automata Theory, lowa City, USA, 1973: 1-11.

[40] McCreight E M. A Space-Economical Suffix Tree Construction Algorithm[J]. Journal of the ACM, 1976, 23(2): 262-272.

[41] Ukkonen E. On-Line Construction of Suffix Trees[J]. Algorithmica, 1995, 14(3): 249-260.

[42] Manber U, Myers G. Suffix Arrays: A New Method for On-Line String Searches[J]. SIAM Journal on Computing, 1993, 22(5): 935-948.

[43] Lee I. A Simple Pattern Matching Algorithm for Weighted Sequences[C]//Proceeding of Research in Applied Computation Symposium, San Anatonio, USA, 2012: 65-67.

[44] Amir A, Chencinski E, IIiopoulos C, et al. Property Matching and Weighted Matching[J]. Theoretical Computer Science, 2008, 395(2-3): 298-310.

[45] Fischer J, Heun V. A New Succinct Representation of RMQ-Information and Improvements in the Enhanced Suffix Array[C]//Proceeding of Symposium On Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, Hangzhou, China, 2007: 459-470.

[46] Fisher J. Optimal Succinctness for Range Minimum Queries[C]//Proceeding of Latin American Theoretical Informatics, Oaxaca, Mexico, 2010: 158-169.

# 攻读硕士学位期间承担的科研任务与主要成果

## （一） 参与的科研项目

[1] 陈子阳, 周军锋, 王丁. 面向 XML 数据的关键字查询算法辅助生成技术研究, 国家自然科学基金资助项目. 课题编号: 61272124.

# 致　谢

时光似箭，行文至此，意味着三年研究生的学习生活马上就要结束了。在这三年的研究生学习和生活中，首先要感谢的是我的导师王璿副教授。王老师严谨的治学态度，以及精益求精的工作精神时刻激励着我，让我在学习生活中受益匪浅。王老师不但在学业上对我细心指导，而且在生活上也给了我无微不至的关怀，她让我认识到自己在学习上的不足，让我明白搞学术必须全身心投入的道理，也教导我无论生活中遇到多大的困难，也都要有积极乐观的心态。在王老师的帮助下，我无论是在阅读外文文献方面，还是在动手编程方面都有了显著的提升。同时还要感谢同课题组的周军锋教授和陈子阳教授，两位老师不仅严谨治学，而且为人亲善，在我遇到学习上的问题时，会和我积极地交流，给予我细心的指导，并且在百忙中抽出时间对我的毕业论文给出了至关重要的指导意见。在此特地感谢周老师和陈老师，让我度过了一段充实而美好的研究生生活。最后，向三位老师表示由衷的感谢和崇高的敬意！

同时，我还要感谢三年来和我共同学习奋斗的实验室的兄弟姐妹们。从三年前的互不相识，到三年后的相见恨晚，是你们在我消沉的时候鼓励我，是你们在我遇到困难时给予我无私的帮助，是你们陪我一起走过学习和生活中的点点滴滴。三年来所有的欢笑和泪水汇聚成一幅丰富多彩的回忆画卷，未来的路还很长，虽然我们可能会短暂别离，但是“海内存知己，天涯若比邻”，相信未来有你们的陪伴，生活依然美好。

另外，我还要感谢燕山大学信息学院的全体教职员工，感谢各位老师为我提供了一个安心舒适的学习生活环境，能让我在一个积极向上的校园氛围中逐步成长。

最后，感谢各位专家和学者在百忙中审阅我的论文，并给予悉心的指导，在此向各位专家和学者献上诚挚的谢意。在这里，向所有在学习生活中给予我帮助的老师、同学们表示最诚挚的感谢！