文章编号: 1000-5641(2018)05-0154-10

Levenshtein 算法优化及在题库判重中的应用

张 衡, 陈良育

(华东师范大学 上海市高可信计算重点实验室, 上海 200062)

摘要: 为了解决 Levenshtein 距离算法在长文本和大规模匹配效率的不足, 本文针对 Levenshtein 距离算法提出一种提前终止的优化策略. 首先根据 Levenshtein 距离矩阵中元素内在的联系, 归纳总结出一个递推关系式. 再依据此递推关系式, 提出一种提前终止策略, 可提前判断两个文本是否满足预先设定的相似度阈值. 经过多个学科题库判重实验的佐证, 本文的提前终止策略能显著减少计算时间.

关键词: 题库匹配; 文本相似度; Levenshtein 编辑距离

中图分类号: TP311.5 **文献标志码**: A **DOI:** 10.3969/j.issn.1000-5641.2018.05.013

Optimization of the Levenshtein algorithm and its application in repeatability judgment for test bank

ZHANG Heng, CHEN Liang-yu

(*Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai* 200062*, China*)

Abstract: In order to overcome the disadvantages of the Levenshtein distance algorithm for long text and large-scale matching, we propose an early termination strategy for the Levenshtein distance algorithm. Firstly, according to the intrinsic relationship between elements in the Levenshtein distance matrix, we sum up a recurrence relation. Based on this relation, an early termination strategy is proposed to determine early-on whether two texts satisfy the predefined similarity threshold. Through several tests on different subjects, it is demonstrated that the early termination strategy can significantly reduce calculation time.

Keywords: bank match; text similarity; Levenshtein edit distance

0 引 言

随着计算技术日益提升, 全球教育信息化蓬勃发展. 尤其是在线教育市场呈现出爆发式增长. 题目测试一直是教育和学习的重要组成部分. 在题目测试中, 题库一直处于在线教育的中心位置. 借助于大数据存储、计算和分析等技术, 题库的规模日益扩大, 但也暴露出一

收稿日期: 2018-07-04

基金项目: 国家自然科学基金(11471209)

第一作者: 张 衡, 男, 硕士研究生, 研究方向为自然语言处理. E-mail: 51174500154@stu.ecnu.edu.cn.

通信作者: 陈良育, 男, 副教授, 研究方向为计算机软件与理论. E-mail: lychen@sei.ecnu.edu.cn.

些问题, 题目的质量良莠不齐, 重复或高度相似题目日益增多, 给教育行业的使用带来很大的困扰. 因此, 为了保证题库及教学质量, 快速找出这些重复或高度相似题目就显得十分重要.

题目的主要表现形式是文本. 现有的计算字符串相似度的方法按照计算所需的特征的不同, 可以划分为 3 种方法: 基于字面相似的方法、基于统计关联的方法、基于语义相似的方法, 以及综合这 3 种方法的多层特征方法[1-3]. 题目与普通短文本的主要区别在于, 题目表述的内容更趋向于字面化, 且文本长度较长. 因此本文采用基于字面相似的计算方法来判断两个题目的相似度. 基于字面相似的计算方法主要有基于编辑距离的计算方法[4]和基于相同字或词的方法[5]. 其中, 编辑距离法应用广泛, 计算方法相对成熟. 本文将一个题目看作为一个字符串, 通过计算 2 个字符串的编辑距离来判断 2 个题目彼此是否相似[6]. 文献 [7] 通过修改操作代价, 提高了算法的精准度, 文献 [8] 通过扩展交换操作, 提高了算法的精确度, 虽然文献 [7] 和文献 [8] 都提高了算法的精准度, 但同时也都增加了计算量.

本文分析 Levenshtein 距离矩阵中元素的内在关系, 提出一种提前终止的优化策略, 可预判编辑距离是否满足预设阈值, 并将该优化策略用于大规模题目匹配. 本文的主要贡献在于提出并证明了定理1, 并根据定理 1 提出了提前终止的计算优化策略. 该策略不仅可以提前判断两字符串是否不相似, 还可以提前判断两字符串是否相似, 并且实现了并行程序加速匹配过程. 经过大量的实验, 测试结果表明: 根据该优化算法得出的结果与原算法得出的结果相同, 且显著减少计算时间.

本文组织结构如下: 第 1 节先介绍原 Levenshtein 算法的相关内容; 第 2 节提出对原 Levenshtein 算法的优化, 和优化策略的理论证明, 以及对应的时间复杂度的分析; 第 3 节介绍在实验过程中使用的两种优化策略, 以及对比优化前和优化后的编辑算法在处理相同数据集下的效率. 第 4 节是总结.

1　Levenshtein 算法简介

1.1　Levenshtein 距离基本思想

Levenshtein 距离又称编辑距离, 由俄国科学家 Vladimir Levenshtein 于 1965 年提出[9], 指从源字符串 S 转成目标字符串 T 最少需要进行多少次字符的修改、添加和删除操作[10]. 这 3 种操作次数的总和即定义为两个字符串的编辑距离. 显然, 编辑距离越小, 表明两个字符串的相似程度越高.

1.2　Levenshtein 距离算法

设有 2 个字符串 S 和 T, 其中 $S=s_1s_2\cdots s_m, T=t_1t_2\cdots t_n$ (不失一般性可设 $n\geqslant m$). 其中 s_i, t_j 是 S 和 T 经过分词的词项. 建立一个字符串 S 与 T 的 $(m+1)\times(n+1)$ 阶关系矩阵 LD(默认 LD 矩阵的每一行对应 S 的一个下标, 每一列对应 T 的一个下标), 按式(1)填充矩阵 LD(矩阵元素也称单元或单元格)[11].

$$d_{i,j}=\begin{cases} i, & j=0,\\ j, & i=0,\\ \min\begin{cases} d_{i-1,j-1}+a_{i,j},\\ d_{i,j-1}+1,\\ d_{i-1,j}+1, \end{cases} & i>0, j>0, \end{cases} \tag{1}$$

其中

$$a_{i,j}=\begin{cases}0 & s_i=t_j\\ 1 & s_i\neq t_j\end{cases}(i=1,2,\cdots,m;j=1,2,\cdots,n).$$

矩阵 $LD_{(m+1)\times(n+1)}$最右下角的元素 $d_{m,n}$ 即为字符串 S 和字符串 T 之间的 Levenshtein 距离, 简称编辑距离, 记为 ld, 表示字符串 S 变化到字符串 T 所需的最少编辑操作次数.

根据编辑距离 ld, 定义字符串 S 与字符串 T 的相似度[12],

$$Sim=1-\frac{ld}{\max(n,m)}. \tag{2}$$

式(2)中 m, n 分别为 S 和 T 的长度. Sim 值越大, 表示两个字符串相似度越高. 显然, ld 越小, Sim 值越大, 两个字符串越相似.

以 S="$ABCD$", T="$EABC$"举例, 根据式(1)计算可得表 1.

表 1 字符串 S 与 T 的距离矩阵表

Tab. 1 Distance matrix of strings S and T

	Null	A	B	C	D
Null	0	1	2	3	4
E	1	1	2	3	4
A	2	1	2	3	4
B	3	2	1	2	3
C	4	3	2	1	2

从表 1 可知, $d_{4,4}=2$, 即 $ld=2$. 显然, 字符串 S 转换成字符串 T 需要进行 2 次修改操作. 再根据式(2)得到 $Sim=1-\frac{2}{4}=\frac{1}{2}$, 最终得出字符串 S 与 T 的相似度约为 50%.

2 Levenshtein 算法优化及应用

2.1 编辑距离矩阵特征分析与证明

根据第 1.2 节可知, $d_{m,n}$ 表示为 $S=s_1s_2\cdots s_m$ 与 $T=t_1t_2\cdots t_n$ 的 LD 距离.

因此, $\forall i,j,1\leqslant i\leqslant m,1\leqslant j\leqslant n,d_{i,j}$ 表示为 $S_1=s_1s_2\cdots s_i$ 与 $T_1=t_1t_2\cdots t_j$ 的编辑距离. 如表 2 左上角阴影部分所示.

表 2 $d_{i,j}$ 的 LD 距离矩阵元素含义

Tab. 1 Meaning of $d_{i,j}$ in LD distance matrix

	Null	t_1	$\cdots$	t_j	$\cdots$	t_n
Null	0	1	$\cdots$	j	$\cdots$	n
s_1	1					
$\cdots$	$\cdots$					
s_i	i			$d_{i,j}$		
$\cdots$	$\cdots$					
s_m	m					$d_{m,n}$

根据表 1 可知, $d_{1,1}=1$表示字符串"A"与"E"的编辑距离为 1, $d_{2,2}=2$ 表示为字符串"AB"与"EA"的编辑距离为 2, 通过观察发现如下规律,

$$\forall i,0\leqslant i<m,d_{m,n}\geqslant d_{m-i,n-i}.$$

根据如上的观察, 提出如下定理.

定理 1　$\forall i,j,0\leqslant i<m,0\leqslant j<n,d_{i+1,j+1}\geqslant d_{i,j}$, 恒成立.

证明　设

$$d_{i,j}=X;d_{i+1,j+1}=Y;d_{i+1,j}=Z.$$

$$S_1=s_1s_2\cdots s_i;T_1=t_1t_2\cdots t_j;S_2=s_1\cdots s_is_{i+1}=S_1s_{i+1};T_2=t_1\cdots t_jt_{j+1}=T_1t_{j+1}.$$

根据式(1)可知, $d_{i+1,j+1}$ 的值与 $d_{i,j}$, $d_{i+1,j}$, 和 $d_{i,j+1}$ 相关, 故存在以下 3 种可能.

①若 $d_{i,j}$ 为最小值(若$d_{i,j}=d_{i+1,j}$ 或者 $d_{i,j}=d_{i,j+1}$, 也认为 $d_{i,j}$ 为最小值)

根据式(1)中的 $a_{i,j}$ 取值可知, 若 $s_{i+1}=t_{j+1}$, 则 $d_{i+1,j+1}=X=d_{i,j}$, 反之 $d_{i+1,j+1}=X+1$. 故当 $d_{i,j}$为最小值时, $d_{i+1,j+1}\geqslant d_{i,j}$.

②若 $d_{i+1,j}$ 为最小值 ($d_{i+1,j}<d_{i,j}$)

因为 $d_{i+1,j}<d_{i,j}$, 则可推出以下结论:

$$\exists k,0<k\leqslant i+1,\text{使得}S_k=s_k\cdots s_is_{i+1}\text{为 }T_1\text{ 的子串;}\tag{3}$$

反之不一定成立.

若使用反证法证明这种情况: 若不存在 S_k 是 T_1 的子串, 则 T_1 中肯定不存在 s_{i+1}. 因此当 S_1 添加 s_{i+1} 后, T_1 只能插入 s_{i+1}, 或将 t_j 修改成 s_{i+1} 才能转换成 S_2, 故 $d_{i+1,j}=d_{i,j}+1$, 与 $d_{i,j}>d_{i+1,j}$ 产生矛盾, 故结论(3)成立.

需要指出的是, 若结论(3)能够成立, $d_{i+1,j}<d_{i,j}$ 不一定成立. 例如表 2 中 $d_{1,2}=2$, 即将 $T_1=$“E”转换成$S_1=$“AB” 最少需要两次编辑操作, 但是当 T_1 插入“A”之后虽然满足结论的要求, 但 $d_{2,2}=2=d_{1,2}$, 故结论(3)成立并不能说明 $d_{i+1,j}<d_{i,j}$ 也成立.

若想 $d_{i+1,j}<d_{i,j}$ 成立, 新插入的 s_{i+1} 还必须满足如下条件,

$$s_{i+1}\text{是 }S_1\text{ 转换成 }T_1\text{ 所需的编辑操作之一.}\tag{4}$$

例如在表 1 中, $d_{2,2}=2$, 若将 $T_1=$“EA”转换成$S_1=$“AB”有两种转换方法, 两种方法都至少需要进行两次编辑操作. 第一种是在 T_1 的最后插入“B”再将“E”删除, 第二种是将 T_1 两个字符逐个修改成 S_1.

当 T_1 添加“B”之后, 发现新添加的“B”正是 T_1 转换成 S_1 所需的编辑操作之一. 所以新添加的“B”反而相当于减少了一次转换所需的编辑操作, 故 $d_{2,2}=d_{3,2}+1$.

因此, 只有当新插入的 s_{i+1} 满足条件(4)时, 才能使 $d_{i+1,j}<d_{i,j}$. 但由于 S_1 只插入一个字符, 所以 $d_{i+1,j}$ 只会比 $d_{i,j}$ 少一次编辑操作, 故 $d_{i,j}=d_{i+1,j}+1$.

又根据式(1)可知, 若 $d_{i,j+1}$ 为最小值, 则 $d_{i+1,j+1}=d_{i,j+1}+1$, 所以 $d_{i+1,j+1}=d_{i,j}$. 因此, 当 $d_{i+1,j}$ 为最小值时也能满足 $d_{i+1,j+1}\geqslant d_{i,j}$.

③ 若 $d_{i,j+1}$ 为最小值 ($d_{i,j+1}<d_{i,j}$)

$d_{i,j+1}$ 与 $d_{i,j}$ 的关系, 与 $d_{i+1,j}$ 和 $d_{i,j}$ 的关系相似, 故证明同第二种可能. 所以 $d_{i+1,j+1}\geqslant d_{i,j}$ 也成立.

结合以上 3 种可能, 证明定理 1 是成立的. 证毕.

令 ($i=m,j=n$), 对定理 1 进行递推展开可得:

$$d_{m,n}\geqslant d_{m-1,n-1}\geqslant\cdots\geqslant d_{1,n-m+1}.\tag{5}$$

2.2 改进编辑距离算法与示例

根据式(5)递推关系式可以对应得到下表 3.

表 3 递推关系式对应 LD 矩阵表

Tab. 3 Recursive relation in LD distance matrix

	Null	t_1	$\cdots$	t_{n-m-1}	$\cdots$	t_{n-1}	t_n
Null	0	1	$\cdots$	$n-m-1$	$\cdots$	$n-1$	n
s_1	1			$d_{1,n-m-1}$			
$\cdots$	$\cdots$				$\cdots$		
s_{m-1}	$m-1$					$d_{m-1,n-1}$	
s_m	m						$d_{m,n}$

根据表 3, 可总结出如下结论,

$$\forall i, 0 \leqslant i < m, d_{m,n} \geqslant d_{m-i,n-i}\text{恒成立.} \tag{6}$$

因此, 在计算 Levenshtein 距离的过程中, $\exists_{0\leqslant i<m}$, 使得 $d_{m-i,n-i} > \alpha$, 则 $d_{m,n} > \alpha$ 恒成立. 在理想的状态下式(6)等价于 $\forall_{0\leqslant i<m}$, $d_{m,n} = d_{m-i,n-i} + 1$. 因此, 若 $\exists i, 0 \leqslant i < m$, 使得 $d_{m-i,n-i} \leqslant \alpha - (m-i)$, 则 $d_{m,n} \leqslant \alpha$恒成立.

在题目匹配中, 判定两个字符串是否相似, 需要对两个字符串的相似度设定一个阈值 Sim. 因此若两个题目相似, 则必满足如下,

$$1 - \frac{ld}{\max(n,m)} \geqslant Sim. \tag{7}$$

简化式(7)可得如下,

$$n(1-Sim) \geqslant ld = d_{m,n.} \tag{8}$$

综合式(6)和式(8), 得出如下,

$$\exists i, 0 \leqslant i < m, \text{使得 } d_{m-i,n-i} > n(1-Sim), \text{则 } d_{m,n} > n(1-Sim) \text{ 恒成立.} \tag{9}$$

根据式(9)可知, 计算 Levenshtein 编辑距离时, 如 $d_{m-i,n-i} > n(1-Sim)$ 时, 则可认为两个题目的编辑距离必大于设定的阈值, 两个题目必不相似.

$$\text{同理, } \exists i, 0 \leqslant i < m, \text{使得} d_{m-i,n-i} \leqslant n(1-Sim) - (m-i), \text{则} d_{m,n} \leqslant n(1-Sim)\text{恒成立.} \tag{10}$$

根据式(10)可知, 如 $d_{m-i,n-i} \leqslant n(1-Sim) - (m-i)$ 时, 则可认为两个题目的编辑距离必小于等于设定的阈值, 两个题目必相似.

算法 1 改进 Levenshtein 算法
输入: 字符串 S 与字符串 T, 相似度阈值 Sim
输出: 字符串 S 与字符串 T 是否相似
1: 比较两个字符串的长度, 将较长字符串的作为行, 较短的作为列, 保证 $n \geqslant m$
2: 初始化 LD 矩阵 $LD[m+1, n+1]$, 按照式(1)中 $i=0$ 与 $j=0$ 初始化第一行和第一列, 并初始化变量 $line=1$
3: 使用 Levenshtein 算法计算第 $line$ 行数据
4: 若 $LD[line, n-m-line]$ 大于(1-Sim)n, 转步骤 9, 否则转步骤 5
5: 若 $LD[line, n-m-line] + m - line$ 小于等于(1-Sim)n, 转步骤 8, 否则转步骤 6
6: 若 $line$ 小于 m, 转步骤 7，否则转步骤 8
7: $line++$, 转步骤 3
8: 返回两字符串相似
9: 返回两字符串不相似

因此, 改进后的 Levenshtein 算法步骤如下, 改进编辑距离算法计算编辑距离流程图见图 1.

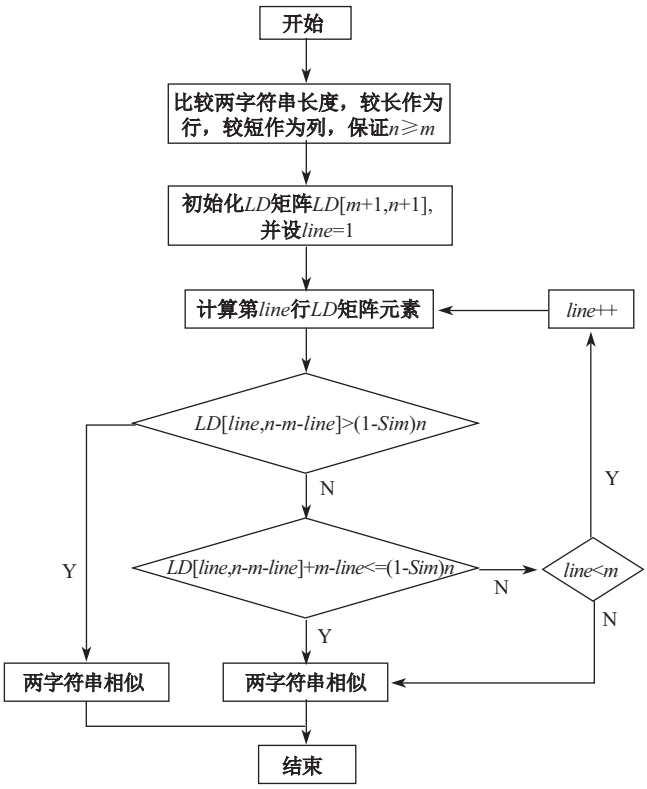

图 1　改进编辑距离的重复度计算过程

Fig. 1　The flow of repeatability calculation for improved editing distance

以表 1 所示的 S 和 T 为例说明改进后的编辑距离算法. 假设若 S 与 T 的相似度大于等于 80%, 则认为 S 与 T 是相似的两个题目.

将 $Sim = 80\%$ 代入式(8), 得 $ld \leqslant 0.2n$. 则 S 与 T 的编辑距离阈值为 $0.2n$. 根据结论(9)可知, $\exists i, 0 \leqslant i < m$, 使得 $d_{m-i,n-i} > 0.2n$, 则认为 S 与 T 不相似.

由结论(9)可知, 只需关注表 1 矩阵中 $d_{1,1}$, $d_{2,2}$, $d_{3,3}$ 以及 $d_{4,4}$ 4 个矩阵元素.

由式(8)得出 ld 的阈值为 $0.2n = 0.8$, 当计算得出 $d_{1,1}$时, 发现$d_{1,1} > ld$, 则认为两个字符串相似度必低于 80%, 因此后续计算就可省略.

由于该改进算法并不修改原矩阵中的数值, 而是在计算过程中根据预设的阈值进行重复度提前判定, 减少不必要的计算过程. 因此改进算法匹配出的重复记录与原算法匹配出的结果是相同的, 因此该改进算法的准确性得到保证.

在文献 [13] 中, 提前终止思想已经使用在了两个大字符串集合中找到了相似的字符串中. 在文献 [13] 中, 设 $\triangle = n - m$(两个字符串的长度差), 设 τ 为两字符串相似度是阈值, 即为式 (8) 中的 $n(1-Sim)$. 在文献 [13] 中关注的是矩阵中 $d_{i,j}(i = 1, 2, 3 \cdots; i - \lfloor \frac{\tau - \triangle}{2} \rfloor \leqslant j \leqslant i + \lfloor \frac{\tau + \triangle}{2} \rfloor)$的值, 若$\exists i, j$, 使得 $d_{i,j} \geqslant \tau$, 则认为两个字符串不相似, 以此来实现提前终止. 本文直接关注的是 $d_{i,i+\triangle}$, 并证明 $d_{i+1,i+\triangle+1} \geqslant d_{i,i+\triangle}$. 本文的效率要比文献 [13] 高, 主要表现在式(10)的提前终止策略是文献 [13] 未涉及的. 且本文每行只需要比较一次, 而文献 [13] 则需要比较 $\tau - \triangle$ 次. 只有当 $\triangle = 0$ (两个字符串的长度相同), 且两个字符串不相似的时候, 本文的效率和文献 [13] 的效

率相同.

2.3 改进编辑算法分析

优化前算法的时间复杂度为 $O(n^2)$, 优化后算法在最坏情况下需要计算的矩阵和原矩阵一样, 所以优化后算法的时间复杂度也为 $O(n^2)$, 因此在时间复杂度上, 与优化前后的时间复杂度相同.

由于原算法本质上是一个矩阵计算, 本身就能够终止, 本文基于原算法提出了一种提前终止的策略, 所以该改进算法也具有终止性.

根据第 2.1 节的证明, 可保证该改进算法的正确性.

3 实验优化与结果分析

3.1 实验优化

在实验过程中增加了两个优化过程, 一个是根据题目长度来决定两个题目是否进行匹配, 还有一个是将进行匹配的题目按照字符出现的频率由低到高重新排列.

在实验过程中发现, 若两个题目相似, 其字符串长度也很相近. 如: S=“AB”与 T=“$AHNSUDIUS$”. 显然, 根据两个字符串的长度就可以认为字符串 S 与 T 是不可能相似的两个题目.

根据上述特点, 设定常数$\varepsilon(0<\varepsilon<1)$, 只有满足如下时, 才进行相似度匹配.

$$(1-\varepsilon)\leqslant\frac{S\text{ 的长度}}{T\text{ 的长度}}\leqslant(1+\varepsilon). \tag{11}$$

实验中, 假设 $\varepsilon=0.25$, 经过分析, 减少了 70% 的无关计算. 在理想状态下, 希望 LD 矩阵元素能尽量满足如下,

$$d_{m,n}>d_{m-1,n-1}>\cdots>d_{1,n-m+1}. \tag{12}$$

这样就能在最少的计算内满足结论(9), 可以最大幅度的提升计算效率. 但当 S=“$AAAB$”, T=“$AAAC$”时, 根据式(1)的计算结果见表 4.

表 4 字符串 S 和 T 的原始 LD 矩阵

Tab. 4 LD distance matrix for S and T

	Null	A	A	A	B
Null	0	1	2	3	4
A	1	0	1	2	3
A	2	1	0	1	2
A	3	2	1	0	1
C	4	3	2	1	1

由结论(10)可知, 关注 LD 距离矩阵中 $d_{1,1},d_{2,2},d_{3,3}$ 以及 $d_{4,4}$ 这 4 个 LD 矩阵元素, 但发现只有当程序计算到 $d_{4,4}$ 时才能得出结果, 与改进算法前的效率相同. 算法优化效率提升不明显, 故需要对数据进行预处理.

在进行匹配前, 先将字符串 S 与 T 中的字符按照各自的出现频率进行重新组合, 按照频率由低到高排序, 组成新的字符串 S' 和 T'.

在新的字符串 S' 与 T'中. 字符越靠前, 表明该字符出现的频率越低, 出现与该字符相同字符的概率也就越低. 所以字符越靠前, 对式(12)的吻合度越高, 因此计算效率也就越高.

将上述的字符串 S 和 T, 根据字符出现频率由低到高排序, 生成新的字符串 S'=“$BAAA$”和 T'=“$CAAA$”, 详见表 5.

表 5　字符串 S' 和 T' 的 LD 矩阵

Tab. 5　LD distance matrix for S' and T'

	Null	B	A	A	A
Null	0	1	2	3	4
C	1	1	2	3	4
A	2	2	1	2	3
A	3	3	2	1	2
A	4	4	3	2	1

根据重新排完序之后的字符串 S'与 T' 的 LD 矩阵. 当计算得出 $d_{1,1}$ 时, 发现 $d_{1,1} > ld$, 表明两个字符串的相似程度不满足设定的阈值, 所以程序只需要计算第一行便可以结束.

3.2　实验结果及分析

为了验证本文改进算法的有效性, 选取一系列真实的 K12 题目进行测试. 实验所用的计算机配置 CPU: i7-7700 8 核 16 线程 3.6 GHz、内存: 16 G. 判重程序中开启 16 个线程进行并行匹配计算. 所有学科题目的重复度阈值设置为 80%. 首先, 选取了 10 万生物试题进行详细匹配, 匹配时根据题目长度进行, 长度差别过大的试题, 默认两个题目不相似, 所以记录了相应题目长度范围内的题目个数、算法改进前后的计算量以及计算时间, 并给出相应的计算量节省百分比和计算时间节省百分比. 结果详见表 6.

根据表 6, 将计算量节省百分比和计算时间节省百分比绘制折线图, 结果详见图 2.

表 6　10 万生物试题的各项数据指标表(80% 重复度阈值)

Tab. 6　Calculation statistics for 100 000 biology questions (repeatablity: 80%)

长度	个数	计算量 改进前	改进后 计算量	计算量节省 百分比/%	改进前 计算时间/s	改进后 计算时间/s	计算时间节省 百分比/%
0—14	11 616	1.63×10^8	2.89×10^7	82.2	14	4	71.4
10—18	16 044	2.17×10^8	3.98×10^7	81.7	18	5	72.2
14—23	18 194	4.20×10^8	6.84×10^7	83.7	36	10	72.2
18—29	19 651	6.88×10^8	1.01×10^8	85.3	50	13	74.0
23—37	20 775	1.03×10^9	1.52×10^8	85.3	91	23	74.7
29—47	22 264	1.28×10^9	1.76×10^8	86.2	141	32	77.3
37—59	22 345	1.51×10^9	2.06×10^8	86.3	206	48	76.6
47—74	20 315	1.44×10^9	1.82×10^8	87.3	224	48	78.6
59—93	15 966	7.83×10^8	1.04×10^8	86.7	179	35	80.4
74—117	11 176	5.34×10^8	7.25×10^7	86.4	185	33	82.2
93—147	7 720	5.50×10^8	6.99×10^7	87.3	246	36	85.4
117—∞	7 474	9.93×10^8	1.56×10^8	88.7	511	78	84.7

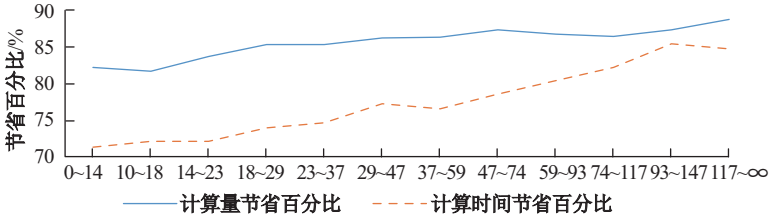

图 2　10 万生物试题计算量和计算时间节省对比图(80% 重复度阈值)

Fig. 2　Performance comparison of repeatability caculation saved for 100 000 biology questions (repeatability: 80%)

从图 2 中, 发现计算量节省百分比是比较稳定的, 而计算时间节省百分比却有着上升的趋势.

出现这种趋势主要是因为在题目长度较短时, 每行计算量较小, 程序运行时间较短, 因此在程序运行中的题目分配、函数调用等程序执行过程导致的延迟不能忽略不计. 因而才导致了题目长度在较短时, 节约耗时的百分比与实际减少的计算量差距较大. 在折线图中也可以清楚地看出随着题目长度的增长, 节约计算量百分比与节约耗时百分比的差距在不断缩小, 由于程序执行过程而导致的延迟所占比例越小.

接着选取了生物的 10 万、20 万、30 万、40 万 和 50 万不同数量的试题进行匹配, 记录了改进前后的计算时间, 以及计算时间节省百分比, 结果详见表 7.

根据表 7 的计算时间节省百分比, 绘制折线图, 结果详见图 3.

表 7 不同生物试题数量实验对比表(80% 重复度阈值)

Tab. 7 Performance comparison of repeatability calculation for biology questions in different sizes (repeatability: 80%)

试题数量/万	改进前的计算时间/s	改进后的计算时间/s	计算时间节省百分比/s
10	1 913	376	80.3
20	4 816	1 033	78.7
30	7 178	1 393	80.6
40	11 099	2 051	81.5
50	19 998	3 743	81.3

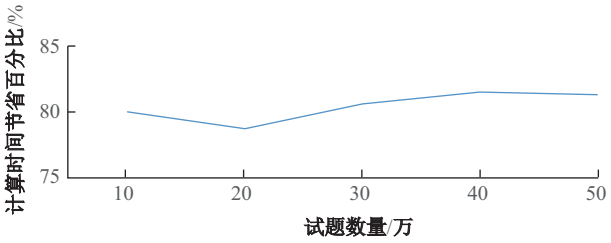

图 3 不同生物试题数量计算时间节省对比图(80% 重复度阈值)

Fig. 3 Comparison of calculation time saved for biology questions in different sizes (repeatability: 80%)

从图 3 中发现在各数据量中, 计算时间节省百分比时区域稳定的, 在 80% 上下浮动.

最后选取语文、数学、物理、化学、地理、生物、历史和政治各 50 万数量的题目进行匹配, 记录了改进前后的计算时间, 以及计算时间节省百分比, 结果详见表 8.

表 8 不同学科实验对比表(80% 重复度阈值)

Tab. 8 Performance comparison of repeatability calculation for multiple disciplines

学科	改进前的计算时间/s	改进后的计算时间/s	计算时间节约百分比/%
语文	223 942	43 626	80.5
数学	131 779	25 838	80.4
物理	50 631	10 355	79.5
化学	35 975	6 372	82.3
生物	19 998	3 743	81.3
地理	25 709	4 944	80.8
政治	44 603	9 462	78.8
历史	27 729	6 075	78.1

根据表 8 的计算时间节省百分比, 绘制折线图, 结果详见图 4.

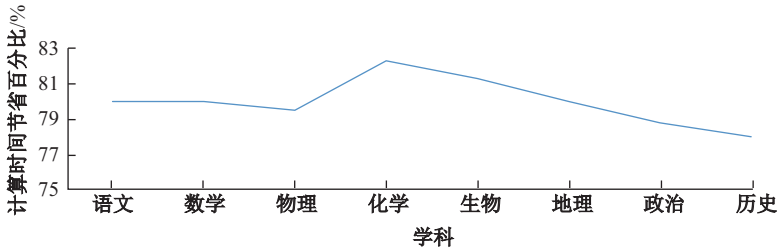

图 4　不同学科计算时间节省对比图

Fig. 4　Comparison of calculation time saved for different disciplines

分析以上结果发现, 使用改进编辑距离算法的平均计算时间节省百分比在 80% 左右. 且两个算法所匹配出的相似题目在数量和匹配上完全相同.

由于各个学科的试题内容和知识点的不同, 所以各学科在匹配时间上各有不同, 比如在语文试题中长文本的个数居多, 所以匹配时间也是最长, 但各学科的平均计算时间节省百分比是比较稳定的, 所以本文提出的提前中止策略的效率是稳定的.

4　结　　论

本文分析了经典 Levenshtein 算法生成的矩阵中内在的联系, 提出一种提前终止策略的改进算法. 该改进算法不仅不影响原算法的准确性, 而且能更快速地判断出两个题目是否相似. 实验结果表明, 本文提出的改进算法, 能够显著减少计算时间, 适用于大规模题库重复判定.

致谢　感谢精锐教育(www.onesmart.org)提供的各学科试题库.

[参　考　文　献]

[1] HALL P, DOWLING G. Approximate string matching[J]. ACM Computing Survey, 1980, 12(4): 381-402.
[2] WAGNER R, FISCHER M. The string-to-string correction problem[J]. Journal of the ACM, 1974, 21(1): 168-173.
[3] 章成志. 基于多层特征的字符串相似度计算模型[J]. 情报学报, 2005, 24(6): 696-701.
[4] MONGE A, ELKAN C. The field matching problem: Algorithms and applications[C]//Proc of ACM International Conference on Knowledge Discovery and Data Mining. New York: ACM, 1996: 267-270.
[5] NIRENBURG S, DOMASHNEV C, GRANNES D. Two approaches to matching in example-based machine translation[C]//Proc of the 5th International Conference on Theoretical and Methodological Issues in Machine Translation. Kyoto: TMI, 1993: 47-57.
[6] JIN L, LI C, MEHROTRA S. Efficient record linkage in large data sets[C]//Proc of the 8th International Conference on Database System for Advanced Application. Kyoto: DASFAA, 2003: 137-146.
[7] 邵清, 叶琨. 基于编辑距离和相似度改进的汉字字符串匹配[J]. 电子科技, 2016, 29(9): 7-11.
[8] 廖宏建, 杨玉宝, 唐连章.改进的编辑距离计算及其在自动评分中的应用[J]. 广州大学学报(自然科学版), 2012, 11(4): 79-83.
[9] LEVENSHTEIN V. Binary codes capable of correcting deletions, insertions and reversals[J]. Soviet Physics Doklady, 1966, 10(1): 845-848.
[10] 玛依热·依布拉音, 米吉提 · 阿不里米提, 艾斯卡尔·艾木都拉.基于最小编辑距离的维语词语检错与纠错研究[J]. 中文信息学报, 2008, 22(3): 110-114.
[11] 姜华, 韩安琪, 王美佳, 等. 基于改进编辑距离的字符串相似度求解算法[J]. 计算机工程, 2014, 40(1): 222-227.
[12] 何锋, 谷锁林, 陈彦辉. 基于编辑距离相似度的文本校验技术研究与应用[J]. 飞行器测控学报, 2015, 34(4): 389-394.
[13] LI G, DENG D, WANG J, et al. Pass-join: A partition-based method for similarity joins[J]. Proceedings of the VLDB Endowment, 2011, 5(3): 253-264.

(责任编辑: 张　晶)