LCS 分词扩展与求解子序列算法优化研究

莫若玉，张秀娟，杨婷，朱洲森*

四川师范大学物理与电子工程学院，四川成都 610101

摘要： 相似度计算作为自然语言处理（Natural Language Processing, NLP）中一项最常见且关键的任务，在查重检测、信息检索等领域有着广泛的应用。为提高文本相似度计算的准确性，在深入分析传统 LCS 算法的基础上，本文提出一种基于分词与同义词匹配的 LCS 扩展算法。该算法结合自然语言处理研究的新成果，针对 LCS 用于文本相似度比较时无法甄别同义词替换等常见的抄袭、套改等手段，以及在求解最长公共子序列时采用的回溯算法时间复杂度高、性能较弱等问题，在分词的基础上通过同义词词林对词语间的相似度进行计算，实现序列间同义词的匹配，达到甄别对原文进行同义词替换等抄袭、套改手段。同时，该算法对求解 LCS 子序列的传统算法进行改进，通过记录最长公共子序列的字符在每个序列中的关联位置，适当增加空间复杂度，实现共有序列链式标记。实验结果表明，本文提出的 LCS 扩展算法可以准确识别文本中同义词的替换，文本相似度的计算结果更加准确，同时使求解 LCS 子序列的时间复杂度由 $O(2^{max(m,n)})$降低至线性级别 $O(n)$。

关键词： 最长公共子序列；动态规划；中文分词；同义词匹配；链式标记

DOI: 10.57237/j.cst.2023.01.006

Research on LCS Word Segmentation Expansion and Optimization of Subsequence Solving Algorithm

Mo Ruoyu, Zhang Xiujuan, Yang Ting, Zhu Zhousen*

Department of Physics and Electronic Engineering, Sichuan Normal University, Chengdu 610101, China

Abstract: As one of the most common and critical tasks in natural language processing (NLP), similarity calculation has a wide range of applications in fields such as censorship detection and information retrieval. In order to improve the accuracy of text similarity calculation, based on the in-depth analysis of traditional LCS algorithms, this paper proposes an extended LCS algorithm based on word separation and synonym matching. The algorithm combines the new achievements in natural language processing research, and addresses the problems that LCS cannot screen the common means of plagiarism and nesting when used for text similarity comparison, as well as the high time complexity and weak performance of the backtracking algorithm used in solving the longest common subsequence, and realizes the matching of synonyms between sequences by calculating the similarity between words through synonym word forest on the basis of word separation. It achieves the screening of plagiarism and copying means such as synonym substitution to the original text. At the same time, the algorithm

基金项目：国家社科基金一般项目《新疆少数民族手工织物图案构图基因理论与应用研究》(20BMZ092).

*通信作者：朱洲森, 992827658@qq.com

收稿日期: 2023-01-07; 接受日期: 2023-02-25; 在线出版日期: 2023-03-02

improves the traditional algorithm for solving LCS subsequences by recording the associated positions of the characters of the longest common subsequence in each sequence and appropriately increasing the spatial complexity to realize the chain marking of common sequences. The experimental results show that the LCS extension algorithm proposed in this paper can accurately identify the substitution of synonyms in the text, and the calculation result of text similarity is more accurate, while the time complexity of solving the LCS subsequence is reduced from $O(2^{max(m,n)})$ to linear level $O(n)$.

Keywords: Longest Common Subsequence; Dynamic Programming; Chinese Participle; Matching of Synonyms; Chain Marking

1 引言

LCS 问题是计算机科学领域的一个经典问题，用于返回两个比较序列间的最长公共子序列[1]。其表述如下：一个序列 S，如果分别是两个已知序列的子序列，且是所有子序列中最长的，则称 S 为两个已知序列的最长公共子序列[2]。例如：序列 A =“堆和栈在计算机内存中是动态变化的”与序列 B=“堆与栈在虚拟机中的存在方式是随时变化的”的最长公共子序列为 C =“堆栈在机中是变化的”。最长公共子序列在序列相似性比对、基因检测、代码克隆检测[3-6]领域都有着重要的应用。

目前，LCS 研究已经相对广泛很多学者都对 LCS 进行了大量的研究，并主要集中在求解 LCS 长度和求解具体子序列两个方面。求解 LCS 长度方面，Wagner 和 Fisher[7]最早提出利用动态规划思想求取序列的长度，其时间复杂度和空间复杂度均为 $O(mn)$。主要思想是采用 $O(m+1)(n+1)$ 的矩阵 dp 来存储对应序列间的计算结果[8]，矩阵 dp 的值可由式(1)迭代计算。Masek [9]等提出运用快速计算字符串编辑距离的方法，使算法时间复杂度降至 $O(n^{2/\log n})$。文献[10]中提出将 LCS 问题转换成求单调子序列问题，将算法时间复杂度降至 $O(m+n\log n)$。Hirschberg [11]提出一种基于分治的思想，对基于动态规划的最长公共子序列算法进行改进和优化，改进后的算法空间复杂度由原来的 $O(mn)$ 降低到了 $O(m+n)$。

$$dp[i][j]=\begin{cases}0, i=0\text{或}j=0\\ dp[i-1][j-1]+1, s_1[i]=s_2[j]\\ max(dp[i][j-1], dp[i-1][j]), s_1[i]\neq s_2[j]\end{cases} \quad (1)$$

LCS 可以很好地应用于文本的相似度比对和查重等，但却无法纠正因同义词替换造成相似度计算结果出现偏差，这恰恰是最常见的规避查重的抄袭和论文套改手段。本文针对这一问题，在动态规划思想的基础上提出一种基于分词与同义词匹配的 LCS 延申算法，通过同义词词林计算分词后词语间的相似度，实现一种甄别同义词替换后文本间的相似度的扩展应用。其次，到目前为止，虽然查找 LCS 子序列长度算法时间复杂度和空间复杂度已经得到了显著改善，但求解实际公共子序列采用的回溯算法时间开销在最坏的情况下达到 $O(2^{max(m,n)})$ [12]，性能较弱，尤其在含批量比对的实际工程应用中，这些算法不能达到令人满意的程度，资源消耗和响应时间都不理想。在深入分析传统 LCS 算法基础上，经过反复研究和工程实践探索，本文提出一种基于位置链式标记的 LCS 子序列扩展算法，实现序列间共有序列的链式标记，适当增加空间复杂度，使查找 LCS 子序列的时间复杂度降低至线性级别 $O(n)$。

2 基于分词与同义词匹配策略的 LCS 扩展研究

2.1 基于分词的 LCS 算法研究

分词是指通过特定的语法规则将连续的字符串或序列按照一定的规范重新组合成词序列的过程[13]，分词作为自然语言处理（NLP）的基础任务之一，其质量好坏会直接影响下游任务的性能[14]。中文分词器作为 NLP 的一项主要研究成果也得到了充分的发展。中文语言特性复杂，分词也复杂得多，为选取合适的中文分词器，通过对主流的中文分词组件 word 分词器、IK 分词器，Ansj 分词器等对比分析得出：Ansj 中文分词功能强大，在能够达到其他分词器效果的同时，还支

持词典拓展，并且分词速度极快，并且 Ansj 官方推荐的精准分词方式，在易用性、稳定性、准确性以及分词效率上，都取得较好的平衡[15]，能够满足中文文本分词需求，故本文采用 Ansj 的精准分词 ToAnalysis 进行分词处理。

分词后，LCS 不再以字符为单位计算最长公共子序列的长度，而是将序列分词结果中的词作为计算单位，字符的长度为 1，而词的长度不固定，分词后如果仍然按照经典 LCS 算法迭代计算矩阵 dp 的值，矩阵最右下角的值则不能再表示两序列 LCS 的长度，只能表示两序列分词后相同词的数量。例如：序列 r_1=“自然的四时交替是亘古的现象”、r_2=“自然季节更迭是自古以来的规律”，分词的结果为 r_1={“自然”，“的”，“四时”，“交替”，“是”，“亘古”，“的”，“现象”}，r_2={“自然”，“季节”，“更迭”，“是”，“自古以来”，“的”，“规律”}，利用式(1)计算得到序列 r_1、r_2 分词后的最长公共子序列的长度为 3，即分词后 r_1、r_2 相同词的数量为 3，为使矩阵 dp 最右下角的值能直接代表序列的 LCS 长度，必须对式(1)做出改进。

在本文中，词语表示序列分词后产生的字符串数组中的每一个元素，flag 表示相同词语的长度，如果词 $s_1[i]$=词 $s_2[j]$，则

$$flag=\frac{1}{2}(l_1+l_2) \tag{2}$$

否则 $flag=0$，l_1、l_2 分别代表词 $s_1[i]$ 与词 s_2[j] 的长度，对 LCS 算法改进后的递推公式如式(3)所示。

$$dp[i]][j]=\begin{cases}0, i=0或j=0\\ dp[i-1][j-1]+flag, s_1[i]=s_2[j]\\ max(dp[i][j-1], dp[i-1][j]), s_1[i]\neq s_2[j]\end{cases} \tag{3}$$

利用式(3)计算序列 r_1、r_2 分词后的最长公共子序列的求解过程如图 1，当计算 dp[1][1] 时，序列分词后 r_1[1]、r_2[1] 对应词语均为“自然”，利用式(2)计算得到 flag 的值为 2，因此 dp[1][1] 的值为 dp[0][0] + $flag$，即 dp[1][1] 的值为 2，最终得到两序列的最长公共子序列的长度为 4。

通过对经典 LCS 进行改进，即使在分词后，改进后的 LCS 也能直接得出序列间 LCS 的长度，但由于分词存在歧义切分问题，即待切分的句子在切分时容易产生切分的歧义[16]，导致 LCS 计算结果出现误差。例如：序列 x_1=“汉字是汉语的记录符号”，x_2=“中国字是中文的记录符号”，分词结果为 x_1={“汉字”、“是”、“汉语”、“的”、“记录”、“符号”}，x_2={“中国”、“字”、“是”、“中文”、“的”、“记录”、“符号”}，计算得到 x_1、x_2 的 LCS 长度为 6，而分词前 LCS 的长度为 7。另外，“汉语”和“中文”互为同义词，无论是经典的 LCS 算法还是基于分词的改进 LCS 算法，都不能识别同义词。基于此，将在分词的基础上，增加序列间同义词匹配，得到序列间广义 LCS 长度，从而提高序列间文本相似度计算的准确性。

		自然	的	四时	交替	是	亘古	的	现象
	0	0	0	0	0	0	0	0	0
自然	0	2	2	2	2	2	2	2	2
季节	0	2	2	2	2	2	2	2	2
更迭	0	2	2	2	2	2	2	2	2
是	0	2	2	2	2	3	3	3	3
自古以来	0	2	2	2	2	3	3	3	3
的	0	2	3	3	3	3	3	4	4
规律	0	2	3	3	3	3	3	4	4

图 1 改进后 r_1、r_2 最长公共子序列求解过程

2.2 基于同义词匹配的相似度计算方法研究

经典 LCS 算法在进行相似度比对时只对序列间相同的字符进行计算，没有考虑字或词之间的相关性，对文本相似度计算结果的准确度影响较大。因此，本节将根据同义词词林的结构及编码特点，利用词语在同义词词林中的词义编码，来量化词语间的语义距离，根据语义距离求解出词语间的相似度，再将词语相似度转换成词语间广义的最长公共子序列，从而提高文本相似度计算的准确性。

在同义词词林中，词义编码由大类、中类、小类、词群五部分组成，分别代表词义编码的一至五级。词义编码的大类由一位大写英文字母表示，中类由一位小写英文字母表示，小类由 2 位数字组成，词群由一位大写英文字母表示，原子词群由两位数字表示。在同义词词林的基础下，词语间的语义距离可由式(4)表示，n 表示从词义编码左边的第一位开始，分支层级数对应编码连续相同的个数。例如：词义编码“Aa01A01=”与“Aa01A02=”，则表示在同义词词林中，两词义编码在第一至四级相同，第五级不同。

$$d(A,B)=5-n \tag{4}$$

在同义词词林中，词语的词义编码可能不唯一，如果在计算词语间的语义距离时，同一词语有多个词义编码，则需将两个词语的词义编码分别两两进行计算，取计算结果的最小值作为两词语的语义距离，可由式(5)表示。其中 X、Y 分别表示包含词 A、B 的词义编码集合。例如：“继承”词义编码包括“Hj10C01=”和“Ig03B01=”“进行”的词义编码“Ig03A01=”，对词义编码分别两两计算得到的语义距离分别为 5、2。根据式(5)得到“继承”与“进行”之间的词义编码距离为 2。

$$dis(A,B)=\min_{a\in X,b\in Y} d(A,B) \tag{5}$$

为了获取词语间准确的相似度，还需要计算词语间的词义编码在不相同的层级分支的总节点数 n 以及两词语对应分支的距离 k，将式(6)的结果 p 作为调节参数。

$$p=1-\frac{k}{n} \tag{6}$$

词义编码的标识位是用来判断词语间的关系，若词语对应词义编码的标识位为“=”，则表示互为同义词，如果为“#”，则表示互为近义词，如果为“@”，则表示没有相关的词。为得到更加准确的词语相似度，需要根据不同的编码标记位乘以不同的权重参数 w，对于三种不同的标志位，权重参数 w 的权值设置分为 1、0.5、0。结合式(5)、(6)，词语间的相似度计算公式如下：

$$S(A,B)=p\times w\times(1-\frac{dis(A,B)}{5}) \tag{7}$$

举例说明如下：

Ga01A01= 高兴 愉快 欢快 …

Ga01A02= 兴高采烈 欢天喜地 …

Ga01A03= 开颜 喜形于色 春风满面…

词语“高兴”和“兴高采烈”的相似度为

$$S(A,B)=(1-\frac{1}{3})\times 1\times(1-0.2)=0.53$$

结合以上内容，本文对广义最长公共子序列长度规定如下：

若词 $s_1[i]$ 与词 $s_2[j]$ 相等，那么两词语间的广义最长公共子序列长度为：

$$L(s_1[i],s_2[j])=\frac{1}{2}(l_1+l_2) \tag{8}$$

若词 $s_1[i]$ 与词 $s_2[j]$ 不相等，且词 $s_1[i]$ 与词 $s_2[j]$ 的相似度 $S(s_1[i],s_2[j])\in(0,1]$，那么两词语的广义最长公共子序列长度为：

$$L(s_1[i],s_2[j])=\frac{1}{2}\times S(s_1[i],s_2[j])\times(l_1+l_2) \tag{9}$$

否则，$L(s_1[i],s_2[j])=0$。其中 l_1、l_2 分别表示 $s_1[i]$、$s_2[j]$ 的长度。

当比较的两词语相等时，词语的相似度为 1，因此可将式(8)合并至式(9)，则基于同义词匹配的改进 LCS 算法的递推公式可由式(10)表示。

$$dp[i][j]=\begin{cases}0, i=0\text{或}j=0\\ dp[i-1][j-1]+L(s_1[i],s_2[j]), S(s_1[i],s_2[j])\neq 0\\ max(dp[i][j-1],dp[i-1][j]), S(s_1[i],s_2[j])=0\end{cases} \tag{10}$$

在分词基础上利用式(10)计算序列 r_1、r_2 的广义最长公共子序列长度的求解过程如图 2。当计算 $dp[4][3]$ 时，$r_1[4]$、$r_2[3]$ 对应的词分别为“交替”与“更迭”，两词语不相等，但由式(9)得到 $L(r_1[4],r_2[3])=2$，因此，$dp[4][3]=dp[3][2]+2$，得到 $dp[4][3]$ 的值为 4，计算 $dp[6][5]$ 时，$r_1[6]$、$r_2[5]$ 对应的词分别为“亘古”与“自古以来”，由式(9)得到 $L(r_1[6],r_2[5])=3$，$dp[6][5]=dp[5][4]+3$，值为 8，最后得到序列 r_1、r_2 广义最长公共子序列的长度为 9。

		自然	的	四时	交替	是	亘古	的	现象
	0	0	0	0	0	0	0	0	0
自然	0	2	2	2	2	2	2	2	2
季节	0	2	2	2	2	2	2	2	2
更迭	0	2	2	2	4	4	4	4	4
是	0	2	2	2	4	5	5	5	5
自古以来	0	2	2	2	4	5	8	8	8
的	0	2	3	3	4	5	8	9	9
规律	0	2	3	3	4	5	8	9	9

图 2 基于同义词匹配的 r_1、r_2 最长公共子序列求解过程

序列间的相似度可以由式(11)计算，Lcs 为序列间广义最长公共子序列的长度，L_1、L_2 分别为原序列的长度。利用图(1)的结果计算 r_1、r_2 间的相似度为 0.3，

利用图(2)的结果计算 r_1、r_2 间的相似度为 0.67。由此可以得出，本文提出的基于同义词匹配的相似度计算方法，可以甄别序列间同义词的替换，提高了文本相似度计算的准确性。

$$sim(s_1,s_2)=\frac{2Lcs}{L_1+L_2} \tag{11}$$

3 基于位置链式标记的 LCS 子序列改进算法

为提高求解 LCS 子序列的效率，本文将在经典 LCS 算法基础上，通过记录相同词语（在本文中，词语相似度为 1 的词也视为相同的词语）在各序列出现的位置，实现共有序列链式标记，得到一种新的改进算法。为与相似度计算结果保持一致，仍将分词后的序列作为标记对象。

为实现序列间共有序列链式标记，本文对 LCS 算法进行扩展，扩展后的算法 LCSFullCN 的数据结构如表 1 所示。

表 1 LCSFullCN 算法数据结构

类型	标识	描述
int[][]	dp	动态规划数组
int	m,n	分词后字符串 1、2 对应词的数量
String[]	s_1,s_2	分词后字符串 1、2 对应的字符数组
class	PathFrom	标记类
PathFrom[]	pathFrom	共有序列链式标记数组

算法 LCSFullCN API 如表 2 所示。

表 2 LCSFullCN 算法 API

方法名称	描述
getLCS	获取 LCS 的长度
LCSFullCN	在构造方法种计算 LCS 的长度
getSequence	获取最长公共子序列

PathFrom 类为扩展算法新增的标记类，有 m、n 两个属性，分别表示最长公共子序列中的词在序列 1、序列 2 分词后产生的字符串数组中的位置。PathFrom 类型的二维数组的大小与矩阵 dp 的大小一致，用于在求解最长公共子序列长度时同步记录对应位置的标记值，值为 PathFrom 类对象。例如：序列 $L_C=AB$、$L_D=CA$ 标记实现如图 3 所示，对两序列共有子序列 A 的标记为（0,1），即表示 A 位于序列 L_C 中 0 的位置，位于序列 L_D 中 1 的位置。

		C	A
	(-1,-1)	(-1,-1)	(-1,-1)
A	(-1,-1)	(-1,-1)	(0,1)
B	(-1,-1)	(-1,-1)	(0,1)

图 3 序列 L_C、L_D 共有序列标记图

基于位置链式标记的 LCS 扩展算法的核心思想是在计算矩阵 *pathFrom* 的值时，同时计算对应位置中二维数组的值，即对比到当前位置，两序列最长公共子序列最后一个词在分词后产生的字符串数组中的位置。标记具体核心步骤如下：

(1) 创建 PathFrom 类型的二维数组 *pathFrom*，大小为$(m+1)(n+1)$，创建 PathFrom 类型的对象 *pathZero*，其属性 m、n 的值均为-1。

(2) 初始化 *pathFrom*。按照式(12)初始化第 0 行和第 0 列的值。

$$\begin{cases} pathFrom[i][0]=pathZero, 0\le i\le m \\ pathFrom[0][j]=pathZero, 0\le i\le n \end{cases} \tag{12}$$

(3) 利用式(13)迭代计算二维数组 *pathFrom* 的值。

$$pathFrom[i][j]=\begin{cases} B(i,j), S(s_1[i],s_2[j])=1 \\ B1(dp[i][j-1],dp[i-1][j]), S(s_1[i],s_2[j])\ne 1 \end{cases} \tag{13}$$

若词 $s_1[i]$ 与词 $s_2[j]$ 的相似度为 1，则通过 B 方法获取 $pathFrom[i][j]$ 的值，否则通过 $B1$ 方法获取 $pathFrom[i][j]$ 的值。其中，B 方法返回属性 m、n 分别为 $i-1$、$j-1$ 的 PathFrom 类对象。在 $B1$ 方法中，需要比较 $dp[i][j-1]$ 与 $dp[i-1][j]$ 的大小，若 $dp[i][j-1]$ 的值大于 $dp[i-1][j]$ 的值，则返回 $pathFrom[i][j-1]$，反之，则返回 $pathFrom[i-1][j]$。例如：t_1 =“汉字是记录汉语的书写符号体系”，t_2 =“汉字是汉语的记录符号”，分词的结果为t_1 ={“汉字”，“是”，“记录”，“汉语”，“的”，“书写”，“符号”，“体系”}，t_2 ={“汉字”，“是”，“汉语”，“的”，“记录”，“符号”}，对序列 、 的最长公共子序列的标记过程如下：

(1) 创建大小为 9×7 的 PathFrom 类型二维数组

pathFrom，并初始化二维数组的值。

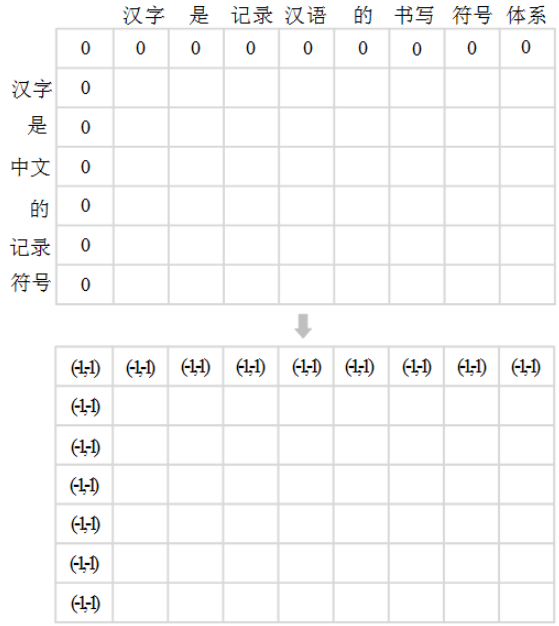

图 4 初始化 pathFrom 的值

(2) 按照式(13)计算二维数组 *pathFrom* 所有的值。

当 $t_1[i]$ 与 $t_2[j]$ 相似度为 1 时，*pathFrom*[*i*][*j*] 的值为属性 m、n 的值为 $i-1$，$j-1$ 的 PathFrom 类对象，即 $pathFrom[1][1]=(0,0)$。

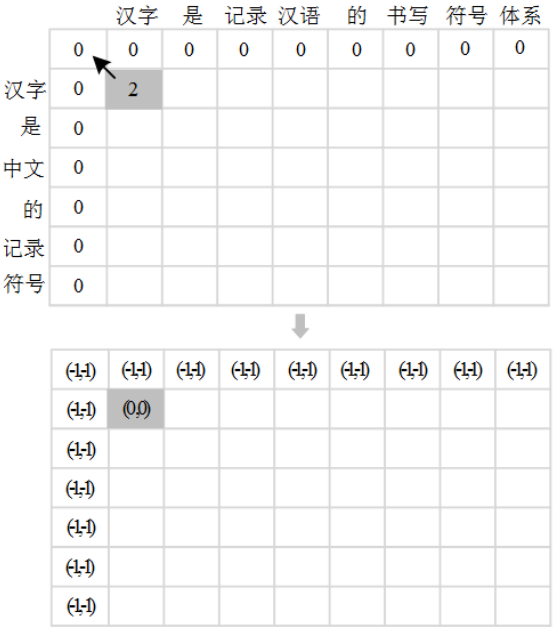

图 5 序列 t_1[i]与 t_2[j]相似度为 1 时标记方式

当 $t_1[i]$ 与 $t_2[j]$ 相似度不为 1，且 $dp[i][j-1]$ 大于 $dp[i-1][j]$ 时，$pathFrom[i][j]=pathFrom[i][j-1]$，即 $pathFrom[1][2]=pathFrom[1][1]=(0,0)$。

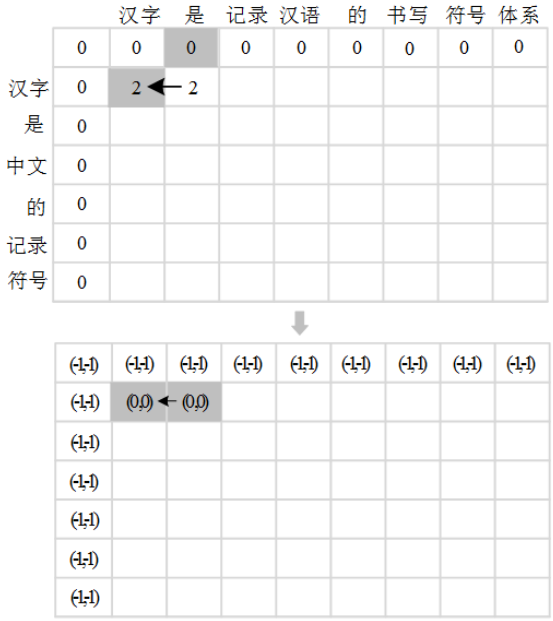

图 6 序列 t_1[i]与 t_2[j]相似度不为 1 且 dp[i][j-1]>dp[i-1][j]时标记方式

当 $t_1[i]$ 与 $t_2[j]$ 相似度不为 1，且 $dp[i][j-1]$ 不大于 $dp[i-1][j]$ 时，$pathFrom[i][j]=pathFrom[i-1][j]$，即 $pathFrom[3][2]=pathFrom[2][2]=(1,1)$。

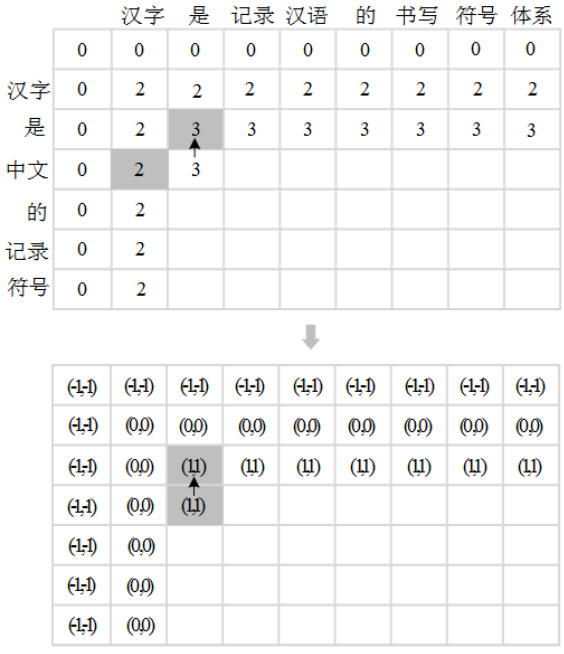

图 7 t_1[i]与 t_2[j]相似度不为 1 且 dp[i][j-1]≤dp[i-1][j]时标记方式

(3) 标记完成后如图 8。

		汉字	是	记录	汉语	的	书写	符号	体系
	0	0	0	0	0	0	0	0	0
汉字	0	2	2	2	2	2	2	2	2
是	0	2	3	3	3	3	3	3	3
中文	0	2	3	3	5	5	5	5	5
的	0	2	3	3	5	6	6	6	6
记录	0	2	3	5	5	6	6	6	6
符号	0	2	3	5	5	6	6	8	8

(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)	(-1,-1)
(-1,-1)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(-1,-1)	(0,0)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
(-1,-1)	(0,0)	(1,1)	(1,1)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)
(-1,-1)	(0,0)	(1,1)	(1,1)	(2,3)	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)
(-1,-1)	(0,0)	(1,1)	(4,2)	(2,3)	(3,4)	(3,4)	(3,4)	(3,4)	(3,4)
(-1,-1)	(0,0)	(1,1)	(4,2)	(2,3)	(3,4)	(3,4)	(5,6)	(5,6)	(5,6)

图 8 序列 t1、t2 共有序列链式标记过程

标记完成后，得到最长公共子序列的词在分词后各序列中的具体位置，即 *pathFrom* 表示的二维数组。根据算法 *getSequence* 获取比对文本间链式标记的共有序列，即比对文本间具体的最长公共子序列，该算法具体实现流程描述如下：

(1) 创建 List 类型集合 *list*，用于存放序列间公共词语，初始化 i、j 的值为 m、n，以便从 LCS 中的最后一个词开始遍历。

(2) 获取属性 m、n 分别为 i、j 的 PathFrom 类型的对象 *pFrom*。

(3) 判断对象 *pFrom* 的属性 m 和 n 的值是否都大于 0，以及 s_1、s_2 分词后 $s_1[pFrom.m]$ 与 $s_2[pFrom.n]$ 的相似度是否等于 1，如果以上条件都满足，则将 $s_1[pFrom.m]$（或 $s_2[pFrom]$）加入 lis 中，并将 $pFrom$、$pFrom.n$ 的值分别赋值给 i、j。

(4) 循环步骤(3)，直到 i 或 j 等于或小于 0。

(5) 由于获取公共子序列是从序列末尾开始遍历，根据以上步骤得到的序列为词序相反的序列，最后通过式(14)得到序列最长公共子序列。

$$Lcs(s_1,s_2)=reverse(list) \tag{14}$$

4 结果比对与应用

4.1 基于同义词匹配的文本相似度准确率测试

本文以某职业技能等级认定平台题库为背景，进行文本相似度比对测试，以甄别大型题库中的雷同题或相似题。首先，从该题库的同一科目同一类型的题目（例：焊工一级判断题）中随机抽取 100 道题目，并对题目进行同义词替换，当向题库新添加题目时，将新加的题目与选取的 100 道题目逐一进行相似度比较，本文相似度的阈值设置为 0.7，若题目相似度大于 0.7，则两题目相似。最后，通过召回率比较经典 LCS 算法与改进后算法文本相似度比对的准确性。召回率的定义

$$召回率=\frac{相似题目数量}{测试题目数量}\times 100\% \tag{15}$$

表 3 测试结果

方法	题目数量	相似题目数量	召回率
经典 LCS 方法	100	4	4%
改进后算法	100	9	9%

按照上述方法，测试结果如表 3 所示。利用改进后的算法测得 9 道题目与新加题目相似，召回率为 9%。而利用经典 LCS 算法测试得到 4 道题目与新加题目相似，召回率为 4%。由此可由得出，本文提出的基于分词与同义词匹配的 LCS 优化算法能够更加准确识别雷同或相似文本。进一步延申应用可以甄别因同义词替换等规避查重的抄袭和论文套改手段，提高了文本相似度比对的准确性和实用性。

4.2 共有序列链式标记时间代价测试

本小节主要测试共有序列链式标记方法、递归回溯方法在求解 LCS 子序列时间消耗与序列长度的关系，共取 5 种不同长度的序列进行测试，序列长度分别为 100、500、1000、2000、3000。考虑特定情况下执行时间存在一定的随机性，为得到更准确的实验比对结果，对不同长度的序列重复执行 10 次，取其平均值作为最后实验分析数据。

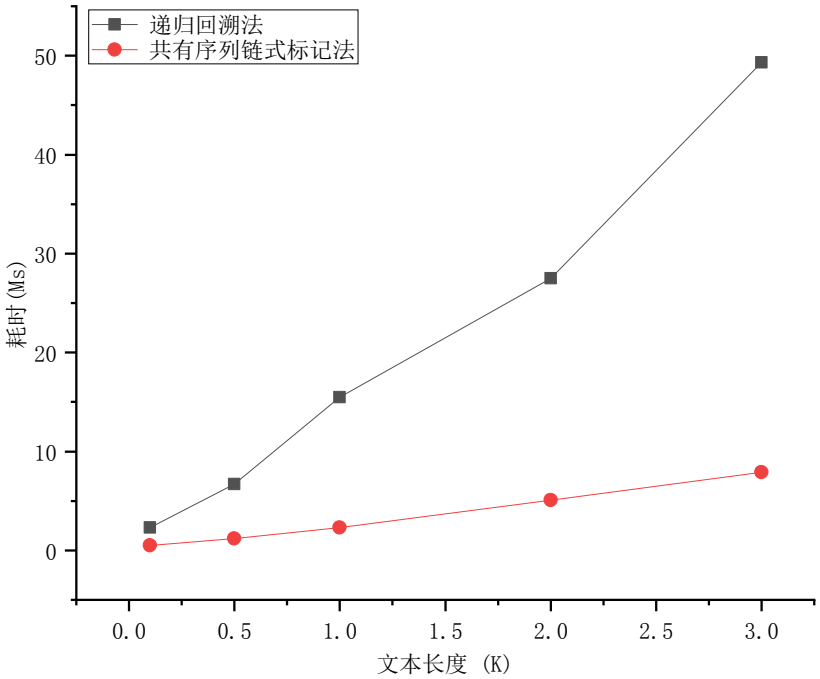

图 9 不同长度的文本与算法耗时的关系图

得到的序列长度与算法耗时关系如图 9 所示，从算法耗时比对可知，随着序列长度的增加，共有序列链式标记算法和递归回溯算法的耗时均有所增加，但共有序列链式标记算法消耗的时间始终比递归回溯算法少，且本文提出的共有序列链式标记法的耗时基本保持线性增长，对于递归回溯法，序列长度增加的越快，其耗时增加的幅度越大。本文提出的基于位置链式标记的 LCS 子序列扩展算法的性能明显优于递归回溯法，目前该成果已经应用于实际工程中。

5 结束语

本文针对 LCS 在解决文本相似度比对存在的不足进行了改进，提出一种基于中文分词与同义词匹配的 LCS 扩展算法，提高了 LCS 在文本相似度比对的准确性和实用性，并通过共有序列进行链式标记，较大幅度提升了求解具体 LCS 子序列的效率。以上研究尚有不足之处，在同义词匹配的 LCS 扩展算法中，算法的准确性较大程度依赖中文分词的准确性，由于同义词词林没有囊括分词后的所有词语，在分词后当比较的词语不相等时，得到的词语间广义最长公共子序列的长度不够准确，导致相似度计算结果存在一些误差，这将依赖于分词器研究的完善，也是我们下一步研究的重点。

参考文献

[1] 翟璐莎. 最长公共子序列查询算法研究 [D]. 燕山大学, 2018.

[2] Hunt J W, MacIlroy M D. An algorithm for differential file comparison [M]. Murray Hill: Bell Laboratories, 1976.

[3] 徐雅静, 李通, 刘玉涛. 基于代码相似度的隐含学生行为模式挖掘 [J]. 计算机教育, 2017, (06): 90-94.

[4] 李明. 文本文件差异对比算法研究 [J]. 软件, 2017, 38 (12): 216-219.

[5] Dayhoff M O. Computer aids to protein sequence determination [J]. Journal of Theoretical Biology, 1965, 8 (1): 97-112.

[6] Hofacker I L, Huynen M A, Stadler P F, et al. Knowledge Discovery in RNA Sequence Families of HIV Using Scalable Computers [J]. KDD, 1996, 7 (2): 1460-1483.

[7] Robert A. Wagner and Michael J. Fischer. 1974. The String-to-String Correction Problem. J. ACM 21, 1 (Jan. 1974), 168–173.

[8] Tseng C T, Yang C B, Ann H Y. Efficient Algorith ms for the Longest Common Subsequence Problem with Squential Substring Constraints [J]. Journal of Complexity, 2013, 29 (1): 44-52.

[9] Masek W J, Paterson M S. A faster algorithm computing string edit distances☆ [J]. Journal of Computer & System Sciences, 1980, 20 (1): 18-31.

[10] 林清波, 吴锤红. 求最长公共子序列长度的一个新方法 [J]. 福建农业大学学报, 1998 (04): 122-126.

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

[12] 郑翠玲. 最长公共子序列算法的分析与实现 [J]. 武夷学院学报, 2010, 29 (02): 44-48.

[13] 朱巧明, 李培峰, 吴娴, 朱晓旭. 中文信息处理技术教程 [M]. 北京: 清华大学出版社, 2005.

[14] 韩士洋, 马致远, 杨芳艳, 李想, 汪伟. 针对中文分词的带标签注意力的成词记忆网络 [J]. 计算机应用研究, 2022, 39 (06): 1651-1655.

[15] 霍晨鹏. 科技专家遴选系统关键技术研究与实现 [D]. 华南理工大学, 2020.

[16] 王洪信, 何爱元, 陈新, 张楠. 搜索引擎之中文分词技术研究 [J]. 信息技术与信息化, 2015 (10): 189-190.