doi:10.3969/j.issn.1000-436x.2014.12.002

高效的 top-*k* 相似字符串查询算法

陈子阳[1,2]，韩玉俊[1]，王璿[1,2]，周军锋[1,2]

（1. 燕山大学 信息科学与工程学院，河北 秦皇岛 066004；
2. 河北省计算机虚拟技术与系统集成重点实验室，河北 秦皇岛 066004）

摘 要：研究基于编辑距离的 top-*k* 相似字符串查询处理方法，即对于给定的字符串集合 *S* 和查询串 σ，返回 *S* 中前 *k* 个与 σ 编辑距离最小的字符串。首先提出了基于长度跳跃索引的 2 种自适应过滤策略来减少字符串之间编辑距离的计算次数；其次提出了查询字符串与不匹配字符串集合的编辑距离下界，以便在处理和 σ 无公共特征的字符串时，进一步减少编辑距离的计算次数；最后给出了基于上述过滤策略的高效 top-*k* 相似字符串查询算法，并在 3 个真实的数据集上进行了实验，实验结果验证了所提算法的高效性。

关键词：字符串相似性；非对称特征方案；长度跳跃索引

中图分类号：TP311 **文献标识码**：A **文章编号**：1000-436X(2014)12-0010-11

Efficient top-*k* string similarity query algorithms

CHEN Zi-yang[1,2], HAN Yu-jun[1], WANG Xuan[1,2], ZHOU Jun-feng[1,2]

(1. School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China;
2. Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China)

Abstract: Computing top-*k* similar strings based on edit distance, i.e., given a query string σ and string set *S*, finding *k* similar strings to σ based on edit distance from *S*. Firstly, two adaptive filter strategies based on length-skip index are proposed, such that to reduce the times of edit distance computation between two strings. Then the lower bound of edit distance between query string and unmatched string set is proposed, such that to further reduce the times of edit distance computation when processing strings that do not have common signatures with the query string. Finally efficient algorithms to return top-*k* similar strings are proposed. Experimental results on three real datasets verify the benefits over the state-of-the-art algorithm.

Key words: string similarity; asymmetric signature scheme; length-skip index

1 引言

top-*k* 字符串相似性查询在很多领域有重要应用，如数据清洗、数据集成、生物信息学和模式识别等。一个典型的例子是对文本进行拼写检查时，spellchecker需要从词典库里查找*k*个与用户输入最相似的单词[1]。在生物信息领域，字符串相似性查询用来找到 top-*k* 相似基因序列，用于疾病预测和新药品研发[2]。数据清洗会从一个实体集合找到 top-*k* 个与给定实体最相似的实体[1]。

已有的基于阈值的字符串相似性查询方法[1,3~5]需要用户提前指定相似性函数(如编辑距离)和阈值，然后返回与查询字符串相似性以及在给定阈值内的字符串。当应用该方法求解 top-*k* 相似字符串

收稿日期：2014-08-12；**修回日期**：2014-10-20

基金项目：国家自然科学基金资助项目（61040023, 61272124, 61303040）；河北省教育厅研究计划基金资助项目（Y2012014）；河北省科学技术研究与发展计划科技支撑计划基金资助项目（11213578）

Foundation Items: The National Natural Science Foundation of China (61040023, 61272124, 61303040); The Research Funds From Education Department of Hebei Province(Y2012014); The Science and Technology Research and Development Program of Hebei Province (11213578)

时，用户很难提前确定一个合适的阈值，当返回结果为空或者返回大量结果时，需要多次根据不同的阈值重复计算，查询效率低。文献[6]使用 Trie 树索引来组织集合中的所有字符串，通过递增的方式计算查询字符串与 Trie 树节点所对应的字符串之间的编辑距离来求解 top-k 结果，由于 Trie 树节点所对应的字符串仅是字符串集合中字符串的前缀，当字符串长度变大时，由于长字符串的公共前缀较短，该方法会处理很多不相似字符串的前缀，导致系统性能下降。

针对已有方法在求解 top-k 相似字符串时效率低的问题，提出了一种长度跳跃索引以及基于该索引的长度过滤策略和计数过滤策略，用以减少需要处理的字符串数量；提出无匹配特征的字符串之间的编辑距离下界，用以进一步减少需要处理的候选字符串数量。基于以上策略，提出了相应的 top-k 字符串相似性查询算法，并在不同特征的数据集上进行实验，实验结果验证了本文算法的高效性和扩展性。

2　背景知识

2.1　问题定义

用 Σ 表示一个有限字符的集合，字符串 s 可以看成由 Σ 中字符组成的有序序列，$|s|$表示字符串 s 的长度。

现有广泛使用的衡量字符串相似性的标准之一是编辑距离[1~8]。2 个字符串的编辑距离是将一个字符串转换成另一个字符串所需要的最少单字符编辑操作次数，允许的单字符编辑操作有：插入、删除和替换。符号 ed(r, s) 表示字符串 r 和 s 之间的编辑距离。假设 r =“srajit”, s =“seraji”，则 ed(r, s) = 2 表示将字符串“srajit”变为“seraji”所需的最少单字符编辑操作次数为 2。

问题定义(top-k 字符串相似性查询)：给定字符串集合 S 和查询字符串 σ，top-k 字符串相似性查询返回集合 $R \subseteq S$，其中 $|R|=k$，且对 $\forall r \in R$ 和 $\forall s \in S-R$，有 ed(r, σ) ≤ ed(s, σ)。

例 1　表 1 是字符串集合 S 中的字符串，假定查询字符串 σ=“geometric”，top-3 字符串相似性查询返回的结果 R={“geometrics”，“isometric”，“biometric”}，σ 与这 3 个字符串的编辑距离分别是 1、2、2，集合中其他字符串与 σ 的编辑距离都不小于 2。

表 1　字符串集合 S 中的字符串

ID	字符串
s_1	emetic
s_2	genetic
s_3	geometry
s_4	isometric
s_5	biometric
s_6	geocentric
s_7	geometrics
s_8	symmetrical

2.2　基本概念

q-gram 和 q-chunk：q-gram 是字符串中所有长度为 q 的子串，$g_q(s)$表示字符串 s 的 q-gram 集合，为保证 s 中每个字符都有对应的 q-gram，需要在字符串 s 结尾添加 $q-1$ 个\$字符。q-chunk 是指字符串中长度为 q 且起始位置为 $iq(i \geqslant 0)$的子串，字符串 s 的 q-chunk 集合是对 s 完整且不相交的划分，用 $c_q(s)$表示。为了保证字符串最后一个 q-chunk 有 q 个字符，需要在 s 结尾添加$(q-(|s|\%q))\%q$ 个\$字符。以字符串“genetic”为例，q = 2，g_q(“genetic”) = {“ge”,“en”,“ne”,“et”, “ti”, “ic”,“c\$”}，c_q (“genetic”) = {“ge”,“ne”,“ti”,“c\$”}。

字符串 s 的任意 q-gram 和 q-chunk 都称为 s 的特征(signature)。由于直接计算字符串之间的编辑距离代价较高，已有方法[1,3,5,8~12]在计算与给定查询串相似的字符串时，第一步通过字符串特征进行过滤，以便缩小候选字符串的数量，进而降低第二步编辑距离计算的代价。

为了在第一步执行过滤，文献[1,4,5,8~10,13,14]都会为字符串集合中的所有字符串特征(q-gram)建立倒排表。倒排表中的每个元素是一个二元组<id, pos>，其中 id 表示相应的 q-gram 在 id 对应的字符串中出现，pos 表示相应的 q-gram 在该字符串中出现的起始位置。每一个倒排表按照 id 升序有序，其中 id 值相等的元组按照 pos 升序有序。预处理时，可以按照字符串的长度排序，进而设定每个字符串的 id，因此本文的后续讨论中，id 小的字符串长度也较短。图 1 展示了表 1 中字符串集合的部分 q-gram 对应的倒排表。对于 q-gram =“eo”，由图 1 可知，“eo”在字符串 s_3、s_6、s_7 中出现，且起始位置都是 1。

2.3　相关工作

在 top-k 字符串相似性查询方面，除了第 1 部分介绍的文献[6]，文献[7]通过动态调整 q-gram 的

长度来解决 top-k 字符串相似性查询，但是动态调整 q-gram 长度非常费时，过滤代价很大，导致查询效率降低[6]，且该方法需要为不同长度的 q-gram 建立索引，所需的存储空间大。文献[3]使用 B 树来索引 S 中字符串的特征(如 q-gram)，通过迭代方式遍历 B 树中的节点，动态计算节点下的字符串与 σ 的编辑距离下界，并基于此更新阈值进行过滤，最后通过计算编辑距离求得 top-k 相似性结果。这种方法需要枚举很多字符串来动态更新阈值，尤其当 k 较小时，阈值不紧确，剪枝效果不好[6]。

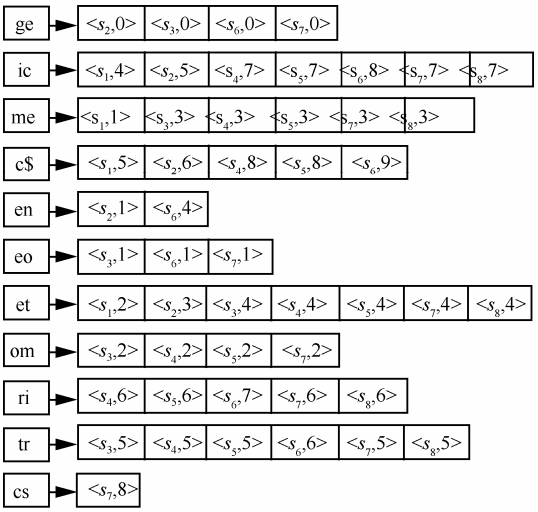


图 1　倒排索引

除以上工作外，和 top-k 字符串相似性查询相关的工作还体现在基于阈值的字符串相似性查询和相似性连接。

基于阈值的字符串相似性查询的问题定义可表述为：给定字符串集合 S、查询字符串 σ、字符串相似性函数(如编辑距离)以及一个阈值 τ，返回 S 中与 σ 相似性小于等于 τ 的字符串。

已有基于阈值的方法[1,2,4,5,7]在过滤阶段利用 q-gram 倒排表得到和查询串的公共 q-gram 不小于 $|\sigma|-q+1-q\tau$ 的候选字符串；验证阶段调用动态规划算法求解每个候选串和查询串的编辑距离，并根据阈值 τ 确定该候选是否满足条件。当应用该方法来求解 top-k 相似字符串时，由于很难根据 k 值确定合适的阈值 τ，因而可能存在返回结果过多和过少的问题，进而导致根据不同的阈值重复计算、效率较低。

以上基于阈值的方法对查询串和被查询串使用的都是 q-gram，称之为对称特征方案，与此相反，文献[10]研究基于非对称特征方案的字符串相似性查询问题(非对称特征方案指查询串用 q-chunk，被查询串用 q-gram，或者反过来也称为非对称特征方案)，并给出了非对称特征方案下 2 个字符串相似的公共特征下界。即给定 2 个字符串 s 和 σ，假设 ed(s, σ) $\leqslant \tau$，如下 2 个不等式成立。

$$|\{(g,c)\,|\,g=c,g\in g_q(s),c\in c_q(\sigma)\}|\geqslant\lceil|\sigma|/q\rceil-\tau \quad (1)$$

$$|\{(g,c)\,|\,g=c,g\in g_q(\sigma),c\in c_q(s)\}|\geqslant\lceil|s|/q\rceil-\tau \quad (2)$$

g 和 c 表示相同的子串，对于式(1)和式(2)中二者匹配(即 g=c)，当且仅当 g 和 c 的位置差不大于 τ。

相似性连接方面，对于给定的 2 个字符串集合 S 和 R、字符串相似性函数(如编辑距离)以及相似性阈值，文献[8~10, 12~18]返回所有分属 2 个集合且相似性在阈值之内的(top-k)相似字符串对，感兴趣的读者可以参考相关文献获取详细信息。

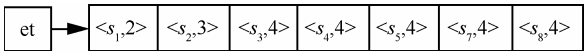
3　top-k 字符串相似查询算法

3.1　top-kLength 算法

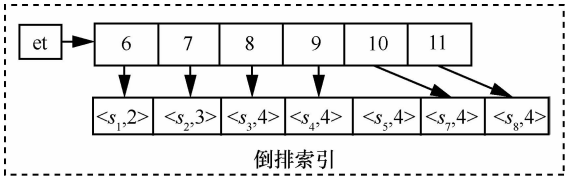
本节首先提出了一种长度跳跃索引，基于该索引提出了一种提前终止策略，根据该策略只需扫描局部倒排表；然后提出了查询字符串与字符串集合之间编辑距离的紧确下界，根据该下界，可以确定是否需要处理与查询串 σ 没有公共特征的字符串。基于上述 2 种策略，给出了基本的 top-k 字符串相似性查询算法，称为 top-kLength 算法。

3.1.1　基于长度跳跃索引的长度过滤策略

给定查询串 σ 和字符串 s，根据编辑距离的定义，如果 ed(σ, s)$\leqslant\tau$，则$||s|-|\sigma||\leqslant\tau$。根据该结论，将原始的 q-gram 倒排表(如图 2(a)所示)按照字符串长度分段组织，如图 2(b)所示，其中长度用来快速定位特定长度的字符串，在求解基于阈值的相似字符串时，通过该索引可以快速定位倒排表中和 σ 长度差不大于 τ 的字符串，从而实现基于长度的过滤策略，将该索引称为长度跳跃索引。



(a) 原始 q-gram 倒排表



(b) 长度跳跃索引

图 2　长度跳跃索引举例

以图 2(b)为例，倒排表中长度为 9 的字符串有 s_4 和 s_5。

显然，基于长度跳跃索引可以高效支持基于阈值的相似字符串查询问题，但是当应用基于阈值的方法来解决 top-k 相似字符串查询时，存在冗余计算问题。例如，给定查询串的长度$|\sigma| = 9, k = 5$，假设 τ = 1，则根据长度过滤的要求，需要处理的字符串长度应在[8,10]内，即 s_3, s_4, s_5, s_7。由于被处理字符串个数小于 5，需要增大阈值。当增大 τ 的取值来增加被处理字符串的个数时，需要重复计算 s_3, s_4, s_5, s_7，显然存在重复计算问题。当$|\sigma| = 9, k = 1, \tau$ = 1，假设 $\mathrm{ed}(\sigma, s_5) = 1$，则不需要计算 s_3, s_7，而基于阈值的方法会处理 s_3, s_7，显然仍然存在冗余计算问题。

针对以上方法存在的冗余计算问题，提出一种基于长度跳跃索引的高效过滤策略，和以上方法相比，处理的字符串数量更少，且只需要一遍处理。其总体思路是根据被处理字符串和查询串的长度差从小到大进行处理。

定理 1　假设当前 top-k 结果中与查询串 σ 编辑距离最大的字符串为 s_k，则对任意尚未处理的字符串 r，如果$||\sigma|-|r|| \geqslant \mathrm{ed}(\sigma, s_k)$，则有 $\mathrm{ed}(\sigma, r) \geqslant \mathrm{ed}(\sigma, s_k)$。

证明　假定 $\mathrm{ed}(\sigma, s_k) = \tau_k$，因为$||\sigma|-|r|| \geqslant \tau_k$，即 σ 和 r 的长度之差不小于 τ_k，根据编辑距离的定义，必有 $\mathrm{ed}(\sigma, r) \geqslant \tau_k$。证毕。

根据定理 1，当按照与查询串的长度差递增方式处理被查询串时，如果长度差满足定理 1 的条件，则可以提前终止，无需处理剩余字符串。

例 2　假定查询 σ = “geometrics”，k = 1，以表 1 中的字符串集合为例，σ 的 q-chunk 所对应的长度跳跃索引如图 3 所示。因为$|\sigma|$=10，所以首先处理长度为 10 的字符串，即字符串 s_6,s_7，此时 $\mathrm{ed}(\sigma, s_7) = 0$；对于倒排表中剩余的任意字符串 r 有$||\sigma|-|r|| \geqslant 1$，根据定理 1，只需处理倒排表中的字符串 s_6,s_7，其他字符串无需处理。

3.1.2　无匹配特征的字符串的处理策略

定理 2　假设 S 是字符串集合，S_c^+ 是 S 中与 $c_q(\sigma)$ 的 q-chunk 至少有一个公共 q-gram 的字符串集合，$S_c^- = S - S_c^+$，则对 $\forall s \in S_c^-$，必有 $\mathrm{ed}(s, \sigma) \geqslant |c_q(\sigma)|$。

证明　对 $\forall s \in S_c^-$，由于 $c_q(\sigma)$是 σ 的不相交的划分，当把 σ 转换为 s 时，每次编辑操作最多影响 1 个 q-chunk，又因为 $g_q(s) \cap c_q(\sigma) = \varnothing$，因此将 σ 转换为 s，至少需要 $|c_q(\sigma)|$ 编辑操作，即 $\mathrm{ed}(s, \sigma) \geqslant |c_q(\sigma)|$。证毕。

当处理完 σ 的 q-chunk 对应的倒排表后，根据定理 2，假设当前 top-k 结果中与 σ 编辑距离最大的字符串为 s_k，当 $\mathrm{ed}(s_k, \sigma) \leqslant |c_q(\sigma)|$时，可以保证当前 top-$k$ 结果即最终结果；反之，当大于$|c_q(\sigma)|$时，仅依赖 σ 对应的倒排表不能保证 top-k 结果的正确性，为了保证结果的正确性，这时仍然需要处理字符串集合中与查询串 σ 没有公共特征的字符串。

例 3　假设 σ=“geometric”，q = 2, k = 3，则 $|c_q(\sigma)|$=5。当处理完查询串 σ 的 q-chunk 对应的倒排表后，假设当前 top-k 结果与 σ 的编辑距离分别为 0, 1, 2, 由于 2 < 5; 根据定理 2，当前 top-k 结果即最终结果。假设当前 top-k 结果与 σ 的编辑距离分别为 0, 1, 7，则仍需要处理字符串集合 S 中与 σ 没有公共特征的字符串，且这些字符串的长度应满足$||s|-|\sigma|| < \mathrm{ed}(s_k, \sigma)$。

3.1.3　top-kLength 算法描述

基于以上讨论，给出了 top-kLength 算法。在算法的执行过程中，使用一个优先队列 $Q_{\text{top-}k}$ 来保存当前所求的 top-k 结果，优先队列中的元素个数上限为 k。

算法 1　top-kLength 算法

输入：字符串集合 S，查询字符串 σ，结果个数 k

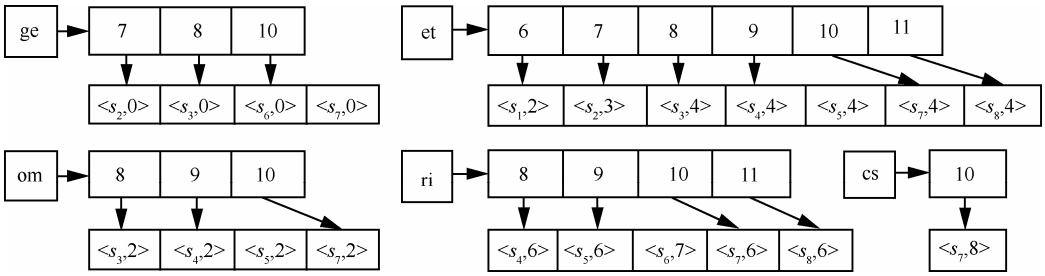


图 3　提前终止策略举例

输出：top-k 相似字符串

1) 初始化优先队列 $Q_{\text{top-}k}$ /*$Q_{\text{top-}k}$ 的每个元素是二元组<s_{id}, d>，其中，s_{id} 表示字符串的 id，d 是 s_{id} 和 σ 的编辑距离，所有元组按照 d 降序排序*/

2) ld←0

3) INV_f← $c_q(\sigma)$中 q-chunk 对应的倒排表

4) while true do

5) INV_p←INV_f 中满足$||s|-|\sigma|| = ld$ 的部分倒排表

6) processPartialInvList(INV_p)

7) if $|Q_{\text{top-}k}| \geqslant k$ and head($Q_{\text{top-}k}$).dis < ld+1 then

8) break

9) else

10) ld ←ld+1

11) if head($Q_{\text{top\_}k}$).dis >$|c_q(\sigma)|$ then

12) processNoCommonStr()

13) return $Q_{\text{top-}k}$

函数 processPartialInvList(INV_p)

1) for each $s\in INV_p$ do

2) d ←verifyED(s, σ)

3) $s.flag$ ← true

4) if $|Q_{\text{top-}k}| < k$ then

5) insert ($Q_{\text{top-}k}$, <s, d>)

6) else

7) if d < head($Q_{\text{top-}k}$).dis then

8) deleteHead($Q_{\text{top-}k}$)

9) insert ($Q_{\text{top-}k}$, <s, d>)

函数 processNoCommonStr()

1) P←S 中满足$||s|-|\sigma|| <$ head($Q_{\text{top-}k}$).dis 的字符串

2) for each $s\in P$ and $s.flag$ = false do

3) d←verifyED (s, σ)

4) if d < head($Q_{\text{top-}k}$).dis then

5) deleteHead($Q_{\text{top-}k}$)

6) insert ($Q_{\text{top-}k}$, <s, d>)

算法 1 按照与查询字符串长度差递增的方式来处理倒排表中的字符串，初始化最初的长度差为 ld=0(第 2)行)。算法首先得到查询字符串 σ 的 q-chunk 对应的倒排表(第 3)行)，在第 4)~10)行，根据长度差及长度跳跃索引确定满足$||s|-|\sigma||=ld$ 的局部倒排表(第 5)行)，然后调用 processPartialInvList()来计算局部的 top-k 结果(第 6)行)。处理完局部的倒排表后，如果 $Q_{\text{top-}k}$ 中有 k 个结果且 head($Q_{\text{top-}k}$).dis< ld+1 (第 7)行)，则根据定理 1 可以提前终止，不需要遍历剩余的倒排表；否则按照与查询字符串长度差递增的方式来处理剩余的倒排表。当处理完 σ 的 q-chunk 对应的倒排表后，算法得到当前的 top-k 结果。最坏情况下，当 head($Q_{\text{top-}k}$)dis > $|c_q(\sigma)|$时，根据定理 2，仅依赖查询串对应的倒排表不能保证当前 top-k 结果等于最终的结果。为了保证结果的正确性，这时仍然需要处理字符串集合中与查询串 σ 没有公共特征的字符串(第 11)和 12)行)。本文使用动态规划计算 2 个字符串的编辑距离[19]。

3.2 top-kLengthCount 算法

与用基于阈值的方法来计算 top-k 相似字符串相比，虽然 top-kLength 算法避免了对字符串的重复处理，且减少了需要处理的字符串数量，但对于需要处理的每个字符串，都需要计算编辑距离，代价较高。针对以上问题，提出了自适应计数过滤策略，进一步减少需要计算编辑距离的字符串数量。

3.2.1 自适应计数过滤策略

定理 3 假设当前 top-k 结果中与查询串 σ 编辑距离最大的字符串为 s_k，$\text{ed}(\sigma, s_k) = \tau_k$,对于下一个要处理的字符串 s，如果 $\text{ed}(s, \sigma) \leqslant \tau_k$，则有下面的不等式(3)和式(4)成立

$$|\{(g,c)\,|\,g=c, g\in g_q(s), c\in c_q(\sigma)\}| \geqslant \lceil |\sigma|/q \rceil - \tau_k \tag{3}$$

$$|\{(g,c)\,|\,g=c, g\in g_q(\sigma), c\in c_q(s)\}| \geqslant \lceil |s|/q \rceil - \tau_k \tag{4}$$

证明 首先证明式(3)，式(4)根据对称性可得。

令 t=$\lceil |s|/q \rceil$，即 t 为 σ 的 q-chunk 个数。考虑在字符串 σ 上执行编辑操作，将其转换为字符串 s。因为 t 个 q-chunk 是互不相交的，则对于不匹配的 q-chunk，至少需要一个编辑操作将其转换为 s 的子串。由于每个编辑操作只能影响一个 q-chunk，在 τ_k 的限制下，根据鸽子洞原理[18]，σ 中至少有 t–τ_k 个 q-chunk 与 s 中的 q-gram 匹配。证毕。

定理 3 说明，下一个被处理的字符串是否能被过滤掉，依赖于当前所求得的 s_k 与查询串 σ 的匹配特征个数。与文献[11]不同，对于 top-k 相似性查询，应用定理 3 时，阈值是动态调整的。在处理字符串的过程中，τ_k 是单调递减的，所以公共特征匹配个数的下界在不断增大，因此可以过滤更多的字符串。

当应用定理 3 时，关键的操作是快速统计查询串的 q-chunk 与被查询串的 q-gram 的匹配个数。提

出一种自适应的归并跳跃策略来求解查询串的 q-chunk 与被查询串的 q-gram 的匹配个数，如函数 adaptiveMergeSkip 所示。

假设当前 top-k 结果中与查询串 σ 编辑距离最大的字符串为 s_k，$\mathrm{ed}(\sigma, s_k) = \tau_k$，则根据定理 3，后续处理的字符串和 σ 的匹配个数不能少于 $\lceil |\sigma| / q \rceil - \tau_k$，函数 adaptiveMergeSkip 中用 T_k 表示该阈值。第 2 个输入参数 INV_p 表示根据长度跳跃索引得到的对应特定长度的局部倒排表集合。

输入：INV_p, 查询字符串 σ，结果个数 k，特征匹配阈值 T_k

输出：top-k 相似字符串

1) 初始化优先队列 Q_p /*按照字符串 id 升序排序*/

2) 将 INV_p 中每个倒排表的第一个待处理元素插入 Q_p

3) while(Q_p 非空) do

4)　　s←head(Q_p)

5)　　将 Q_p 中表示 s 的全部 n 个元素出队列

6)　　if $|Q_{\text{top-}k}| < k$ then

7)　　　　d←verifyED(s, σ)

8)　　　　insert ($Q_{\text{top-}k}$, <s, d>)

9)　　　　pushNext(INV_p, $Q_{p,}$, n)

10)　　else

11)　　　　T_k←$|c_q(\sigma)|$ ← head($Q_{\text{top-}k}$).dis

12)　　　　if($n < T_k$)

13)　　　　　　pushBinSearch(INV_p, Q_p, n)

14)　　　　else

15)　　　　　　m←出队的元素中匹配的特征个数

16)　　　　　　if $m \geqslant T_k$ then

17)　　　　　　　　d←verifyEDThreshold(s, σ, head($Q_{\text{top-}k}$).dis)

18)　　　　　　　　if $d <$ head($Q_{\text{top-}k}$).dis do

19)　　　　　　　　　　deleteHead($Q_{\text{top-}k}$)

20)　　　　　　　　　　insert ($Q_{\text{top-}k}$, <s, d>)

21)　　　　　　pushNext(INV_p, $Q_{p,}$, n)

函数 pushBinSearch(INV_p, Q_p, n)

1) Q_p 中 T_k−1−n 个元素出队

2) s'←head(Q_p)

3) for　T_k−1 个弹出元素涉及的倒排表 do

4)　　r←binSearch(s', INV_{pi}) /* INV_{pi} 表示 T_k−1 个倒排表中的第 i 个倒排表*/

5)　　insert(Q_p, r)

函数 pushNext(INV_p, Q_p, n)

1) for　n 个弹出元素涉及的倒排表 do

2)　　r←INV_{pi} 中的下一个元素 /* INV_{pi} 表示 n 个倒排表中的第 i 个倒排表*/

3)　　insert(Q_p, r)

函数 adaptiveMergeSkip 使用优先队列 Q_p 来存储当前需要处理的元素，Q_p 中元素按照 id 升序排序，并在第 2)行将 INV_p 中每个倒排表的第一个待处理元素插入 Q_p。在第 3)~21)行，只要 Q_p 不空，算法首先将与队头元素表示的字符串相同的所有元素出队列(第 5)行)，如果当前 $Q_{\text{top-}k}$ 中少于 k 个结果，则算法直接计算队头元素所表示的字符串 s 和查询串的编辑距离，并将计算结果插入 $Q_{\text{top-}k}$(第 6)~8)行)，并将弹出元素涉及的倒排表中的下一个元素入 Q_p(第 9)行)。反之，如果$|Q_{\text{top-}k}| > k$，算法首先计算下一个候选字符串 s 和 σ 的公共匹配个数的下界 T_k(第 11)行)，然后判断出队列元素个数 n 和 T_k 的关系。如果 $n<T_k$(第 12)行满足)，则调用 pushBinSearch()(第 13)行)，从队列中进一步弹出 T_k−1−n 个元素，加上前面处理的元素，总共从优先队列中出来 T_k−1 个元素。因为所有的倒排表中按照字符串 id 递增排序，令 s'表示当前的队头元素，对于元素 id 小于 s'的元素对应的字符串，其匹配的特征个数最好的情况下是 T_k−1，必小于 T_k，所以对于弹出元素的 T_k−1 个倒排表，直接定位到大于等于 s'的最小元素 r，中间的元素表示的字符串不需要处理，然后将 r 插入优先队列 Q_p。如果 $n \geqslant T_k$，则根据位置进一步计算 n 个元素中匹配的特征个数 m(第 15)行)，如果 $m \geqslant T_k$，调用 verifyEDThreshold( )函数计算 s 和 σ 的编辑距离 d(第 17)行)，当 d 小于 τ_k，即 s 和 σ 的编辑距离小于当前 top-k 结果和 σ 的最大编辑距离，则用 s 替换 $Q_{\text{top-}k}$ 中编辑距离最大的字符串（第 18)~20)行)。并将弹出元素涉及倒排表中的下一个元素入 Q_p。

例 4　假设经过长度过滤得到的局部倒排表如图 4(a)的虚线框所示，且 T_k=4。为了处理虚线框中的字符串，函数 adaptiveMergeSkip()首先将每个倒排表的第一个待处理元素，即 36,100,50,115,115 进入优先队列，如图 4(b)所示。然后 36 出队列，这时 n=1，由于 1<T_k=4，所以再从优先队列中弹出 T_k−1−n =2 个元素，即弹出 50 和 100，这时共有 T_k−1=3 个元素出队列，50 和 100 不可能是满足条件的结果，可以直接跳过，无需计算编辑距离。之后当前优先

队列中最小的元素为 115。对于刚刚弹出元素的 $T_k-1=3$ 个倒排表，直接定位大于等于 115 的最小元素，这样可以跳过很多不可能成为 top−k 结果的元素，队列状态如图 4(c)所示。下次处理时，当前优先队列的头元素为 115，个数为 4，调用基于阈值的编辑距离函数 verifyEDThreshold()来验证查询串 σ 和 115 对应的候选串的编辑距离。如果二者的编辑距离小于当前 top-k 结果的最大编辑距离，则更新当前的 top-k 结果，并根据定理 3 更新匹配的 q-gram 和 q-chunk 的阈值为 T_k。

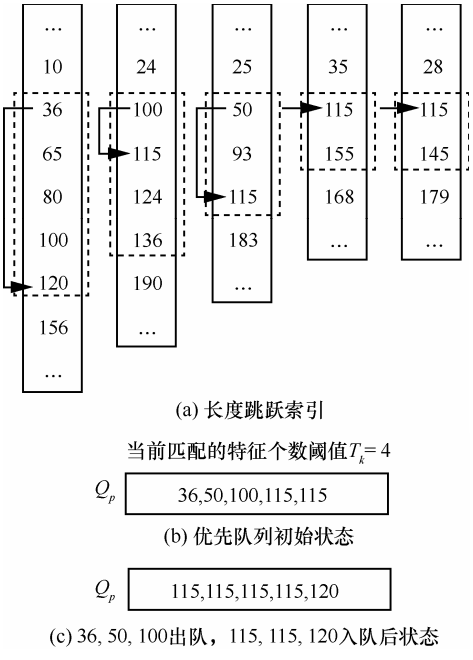


图 4　adaptiveMergeSkip 函数举例

假设 top-k 结果中与查询串 σ 编辑距离第 k 大的字符串为 s_k, ed(σ, s_k) = τ_k，在给定 τ_k 情况下只需判断字符串 σ 与 s 的编辑距离是否大于 τ_k ，根据文献[15]中的 length pruning 策略，可以降低计算编辑距离的时间复杂度，如果 2 个字符串的编辑距离大于 τ_k，则返回 τ_k+1，否则返回实际编辑距离，用 verifyEDThreshold(s_1, s_2, τ_k)表示基于阈值的编辑距离算法，verifyED(s_1,s_2)表示计算 2 个字符串的实际编辑距离的动态规划算法。

与 merge-skip[1] 使用固定阈值的方法不同，adaptiveMergeSkip 函数的阈值是动态调整的。可以看出，首先使用长度跳跃索引定位特定长度的字符串，并使用 adaptiveMergeSkip 函数对局部倒排表进行处理，可以首先定位与查询字符串较相似的字符串，所以 τ_k 可以快速降低，相应的 q-gram 和 q-chunk 的匹配个数快速增加，从而加速扫描查询相关的倒排表。

3.2.2　top-kLengthCount 算法描述

基于以上介绍的自适应计数过滤策略，给出了 top-kLengthCount 算法，如算法 2 所示。

算法 2　top-kLengthCount 算法

输入：字符串集合 S，查询字符串 σ，结果个数 k

输出：top-k 相似字符串

1) 初始化优先队列 $Q_{\text{top-}k}$ /*$Q_{\text{top-}k}$ 的每个元素是二元组<s_{id}, d>，其中 s_{id} 表示字符串 id，d 示 s_{id} 和 σ 的编辑距离，所有元组按照 d 降序排序*/

2) T_k←1, ld ←0

3) INV_f←$c_q(\sigma)$中 q-chunk 对应的倒排表

4) while true do

5)　　INV_p←INV_f 中满足$||s|-|\sigma|| = ld$ 的部分倒排表

6)　　adaptiveMergeSkip (T_k, INV_p, σ, k)

7)　　if $|Q_{\text{top-}k}| \geqslant k$ and head($Q_{\text{top-}k}$).dis < ld+1 then

8)　　break

9)　　else

10)　　ld←ld +1, T_k = head($Q_{\text{top-}k}$).dis

11) if head($Q_{\text{top-}k}$).dis >| $c_q(\sigma)$| then

12)　　processNoCommonStr()

13) return $Q_{\text{top-}k}$

函数 processNoCommonStr()

1) P←S 中满足$||s|-|\sigma|| <$ head($Q_{\text{top-}k}$).dis 的字符串

2) for each $s\in P$ and s.$flag$ = false do

3)　　$com|\{(g,c)\,|\,g=c, g\in g_q(\sigma), c\in c_q(s)\}|$

4)　　if com - head($Q_{\text{top-}k}$).dis then

5)　　d←verifyEDThreshold (s_{id}, σ, head ($Q_{\text{top-}k}$).dis)

6)　　if d < head($Q_{\text{top-}k}$).dis do

7)　　deleteHead($Q_{\text{top-}k}$)

8)　　insert ($Q_{\text{top-}k}$, <s, d>)

top-kLengthCount 算法与 top-kLength 算法的不同之处体现在：1)在第 6)行调用 adaptiveMergeSkip 函数对局部倒排表进行处理，并在处理完局部倒排表后，根据定理 2 更新 T_k(第 10)行)；2)与 top-kLength 算法处理字符串集合中与查询串 σ 没有公共特征的字符串不同，在 processNoCommonStr()中，虽然 σ 的 q-chunk 集合与未处理的字符串的 q-gram 集合的交集为空，根据式(4)，通过计算 σ 的 q-gram 集合

与未处理的字符串的 *q*-chunk 集合的交集(第 3)行)，进一步减少需要计算编辑距离的字符串。如果匹配的特征个数大于下界(第 4)行)，则调用 verifyEDThreshold()进行编辑距离的计算(第 5)行)，如果二者的编辑距离小于当前 top-*k* 结果的最大编辑距离，则更新当前的 top-*k* 结果(第 6)~8)行)。

4　实验评估

4.1　实验环境

实验所用的机器配置为 Intel Pentium G2020，2.90 GHz CPU，4 GB 内存，操作系统为 Ubuntu 12.04 LTS。

根据文献[6]的实验结果，Range 算法比 AQ[3]、B^{ed}-tree[4]和 Flamingo[1]更高效，因此实验中只和 Range 算法进行比较，Range 算法源码由原文作者提供[注1]。基于 C++实现了本文中提出的算法，并使用-O3flag 的 GCC4.8.2 进行编译。实验环境与 Range 算法完全相同。

所用数据来自 3 个真实数据集，分别是：1) word 数据集[注2]，包含常用的英语单词字符串；2) author 数据集，包含作者名称的字符串，从期刊 pubMed [注3]提取；3) DBLP 数据集[注4]，包含作者名字与文章名字连接起来的长字符串。表 2 给出了 3 个数据集的基本统计信息。图 5 中展示了 3 个数据集中字符串的长度分布情况。由图 5 可知，所有数据集的字符串长度都呈近似的正态分布，其中 word 和 author 数据集的字符串长度较短，而 DBLP 数据集的字符串长度较长。

对于每一个数据集，从数据集中随机选择了 100 组查询，然后比较平均运行时间。

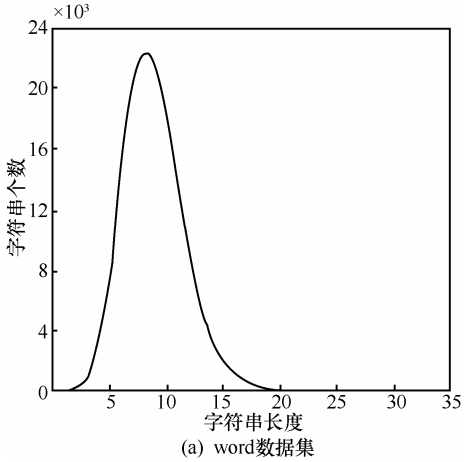
表 2　数据集基本信息

数据集	个数	平均	最大	最小
word	146 033	8.76	35	1
author	1 266 150	13.92	263	8
DBLP	156 506	151.9	793	25

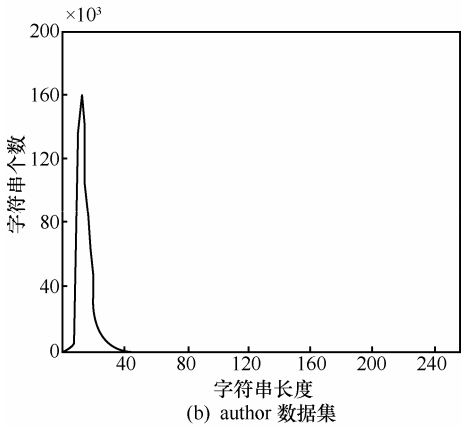
4.2　性能比较

4.2.1　*q*-gram 长度的确定

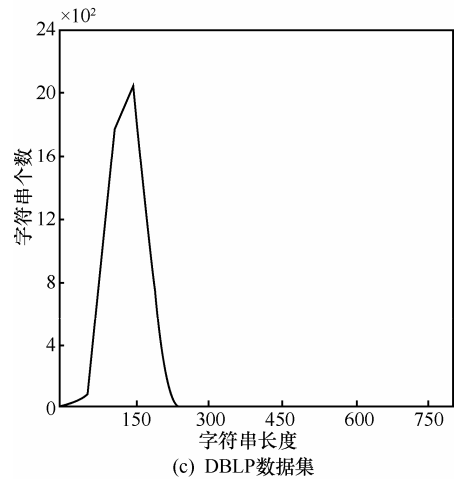
由于本文算法的性能与 *q*-gram 的长度相关，这部分首先通过实验为每个数据集确定合适的 *q*-gram 长度。在每个数据集上运行 100 个查询，每个查询返回 top-100 结果。图 6 展示了在 *q*-gram 长度变化的情况下，本文算法的性能波动情况。由图 6 可知，本文算法在 word/author/DBLP 数据集上性能最好时 *q*-gram 的长度分别为 2、2、6，因此，在后续实验中，本文算法的运行时间都是基于这里确定的 *q*-gram 长度。



(a) word数据集



(b) author 数据集



(c) DBLP数据集

图 5　数据集字符串长度分布

注1　http://dbgroup.cs.tsinghua.edu.cn/dd/projects/topksearch/index.html
注2　http://dbgroup.cs.tsinghua.edu.cn/dd/data/data.tar
注3　http://www.ncbi.nlm.nih.gov/pubmed/
注4　http://www.cse.unsw.edu.au/~weiw/project/simjoin.html#_download

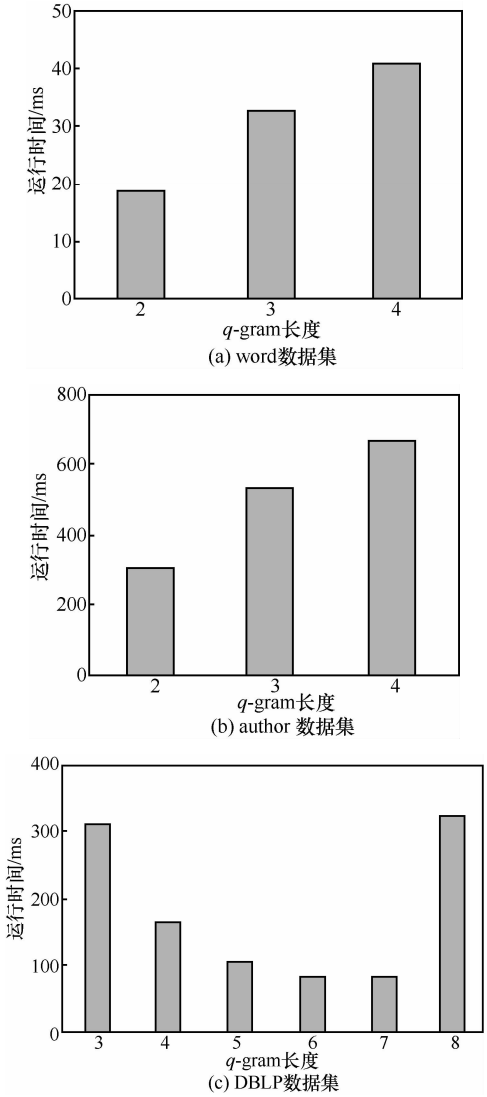


图 6　不同 q-gram 长度运行时间

4.2.2　过滤效果比较

由于本文得到的 Range 算法是二进制可执行代码，没有算法运行过程中的统计信息，因此这里的过滤效果仅展示本文方法之间的比较。

由表 3 可知，通过长度跳跃索引进行过滤，在求解 top-10 结果时，本文算法无需计算所有字符串(每个数据集的字符串数量见表 2)的编辑距离。对于 word、author、DBLP 数据集来说，需要计算编辑距离的字符串数量分别为 22 488、279 046、976（表 3 第 5 列），比例分别为相应数据集中字符串总量的 3/20, 1/5, 1/100。进而，通过提出的自适应计数过滤技术，在验证结果时计算编辑距离的字符串数量分别为 9 266、134 560、141，和不使用计数过滤技术相比，需要计算编辑距离的字符串数量分别为原来的 2/5, 12/25, 7/50。

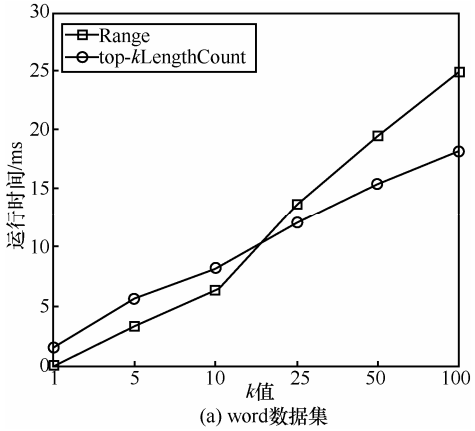
另外，从表 3 的第 3 列可知，算法 top-kLengthCount 需要的过滤时间比算法 top-kLength 稍长，但是验证阶段所用时间得到了显著的改善（表 3 第 4 列），原因在于通过计数过滤，减少了需要计算编辑距离的字符串数量（表 3 第 5 列）。

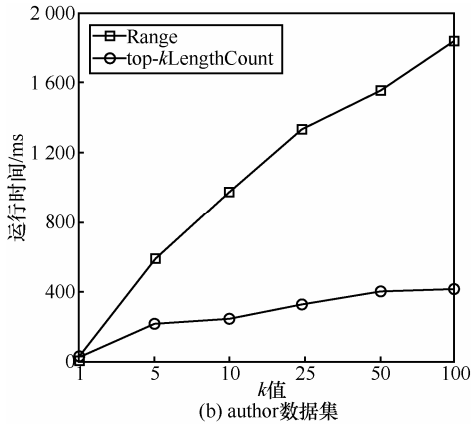
4.2.3　查询时间比较

图 7 给出本文的算法 top-kLengthCount 和 Range 算法的运行时间比较。由图 7(a)可知，对于 word 数据集而言，算法 top-kLengthCount 在 k 小于 18 的时候要比 Range 算法慢，但是随着 k 值的增大本文的算法要比 Range 算法更高效。原因在于 Range 算法采用一种递进的方式来计算 top-k 结果，需要计算所有字符串的前缀，当 k 值较小的时候，需要计算的字符串前缀的数量少，运行时间比较短；但随着 k 值逐步增大，需要计算前缀字符串数量迅速增多(根据文献[6]，当 k 从 10 变到 100 时，需要处理的字符串前缀数量增长了 5 倍多)，需要的运行时间相应变长，而本文的算法 top-kLengthCount 按照与查询串长度差递增的方式访问倒排表中的字符串，同时使用计数过滤并结合提前终止条件来减少需要处理的字符串，当 k 值增大时，需要处理的字符串数量的增长没有 Range 算法增长的字符串前缀数量(当 k 从 10 变到 100 时，需要处理的字符串前缀数量增长了 3 倍)，因而当 k 值增大时，可以获得比 Range 算法更好的处理性能。

表 3　　**3 个数据集上处理 top-10 结果时本文算法的性能比较**

数据集	算法	过滤时间/ms	验证时间/ms	验证阶段处理的字符串数量
word	top-kLength	4.408	18	22 488
	top-kLengthCount	6.406	12.135	9 266
author	top-kLength	81.477	381	279 046
	top-kLengthCount	134.677	260.211	134 560
DBLP	top-kLength	10.397	180.533	976
	top-kLengthCount	23.553	60.167	141





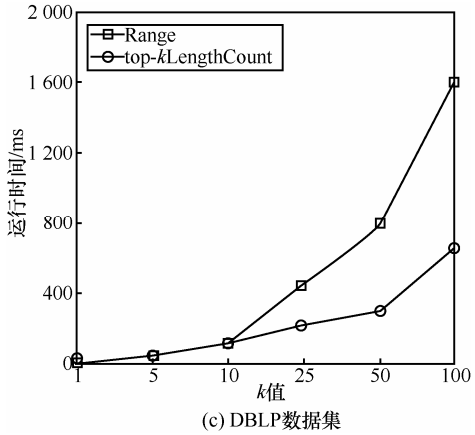
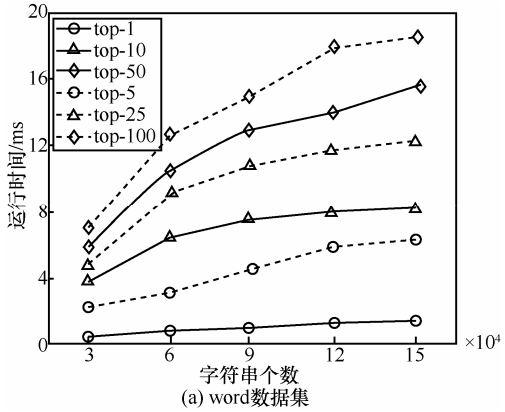


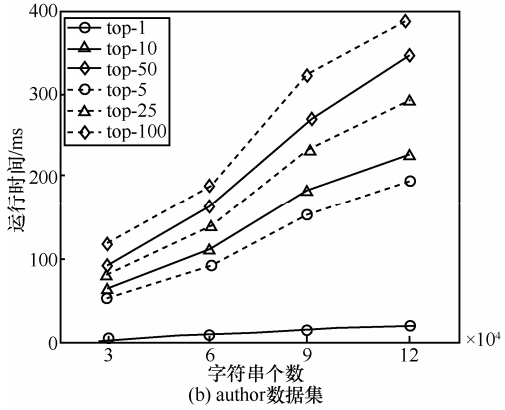
图 7　top-kLengthCount 算法与 Range 算法性能对比

随着字符串数量和平均长度的同时增长，本文的算法 top-kLengthCount 在 author 和 DBLP 数据集上较 Range 算法的优势更明显(如图 7(b)和图 7(c)所示)。原因在于，字符串数量以及字符串长度同时增加(word 数据集包含 146 033 个字符串，而 author 和 DBLP 数据集分别包含 1 266 150 和 156 506 个字符串，长度由 8.76 增长到 151.9)会导致字符串前缀数量的成倍增加，加之相应的查询串变长，Range 需要处理的字符串前缀数量的增长速度更快，所需时间更长。与之对应，本文算法基于自适应的长度和计数过滤，可以过滤很多字符串而不用处理。由图 7(b)和图 7(c)可知，当 k 取 100 时，本文的算法在 author 数据集上比 Range 算法快 4.5 倍，在 DBLP 数据集上比 Range 算法快 2.5 倍。

4.3　扩展性

图 8 展示的是本文算法 top-kLengthCount 在不同 k 值和不同大小的数据集上处理 100 个查询的平均运行时间。





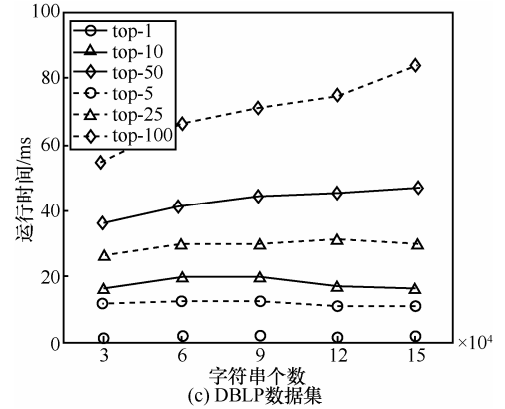


图 8　可扩展性

由图 8 可知，随着数据集和 k 值的增大，算法 top-kLengthCount 体现出了较好的扩展性。以 author

数据集为例，当 k=100、字符串个数是 900 000 时，运行时间是 326 ms，当数据集是 1 200 000 个字符串时，运行时间是 390 ms。在 DBLP 数据集上，当 k=10 时，基于 150 000 字符串的运行时间比基于 120 000 和 90 000 个字符串的运行时间还要少，原因在于算法 top-kLengthCount 按照与查询字符串的长度差递增的方式访问倒排表，随着数据集的增大，在局部倒排表中有更多的相似字符串。按照早终止策略，可以更早地得到 top-k 结果，所以在数据集增大的时候，运行时间会更短。

5　结束语

针对已有 top-k 相似字符串算法存在的冗余计算问题，提出了基于长度跳跃索引的 2 种高效的自适应过滤策略，包括基于字符串长度差递增方式的过滤策略和基于当前 top-k 结果的动态计数过滤策略，以减少字符串之间的编辑距离计算次数，针对查询串对应的倒排表存在不能正确求解 top-k 结果的问题，提出了查询字符串与不匹配字符串集合的编辑距离下界来进一步减少字符串之间编辑距离的计算次数。实验结果表明，被处理的字符串越长，本文方法的优势越明显；k=100 时，所提算法比现有的最好算法快 2~4 倍。

参考文献：

[1] LI C, LU J, LU Y. Efficient merging and filtering algorithms for approximate string queries[A]. ICDE[C]. 2008. 257-266.

[2] KAHVECI T, SINGH A K. Efficient index structures for string databases[A]. VLDB[C]. 2001.351-360.

[3] ZHANG Z, HADJIELEFTHERIOU M, OOI B C, *et al*. Bed-tree: an all-purpose index structure for string similarity query based on edit distance[A]. SIGMOD[C]. 2010.915-926.

[4] CHAUDHURI S, GANJAM K, GANTI V, *et al*. Robust and efficient fuzzy match for online data cleaning[A]. SIGMOD[C]. 2003.313-324.

[5] HADJIELEFTHERIOU M, KOUDAS N, SRIVASTAVA D. Incremental maintenance of length normalized indexes for approximate string matching[A]. SIGMOD[C]. 2009.429-440.

[6] LI G, DENG D, FENG J, *et al*. Top-k String Similarity Search with Edit-Distance Constraints[A]. ICDE[C]. 2013.925-936.

[7] YANG Z, YU J, KITSUREGAWA M. Fast algorithms for top-k approximate string matching[A]. AAAI[C]. 2010.1467-1463.

[8] GRAVANO L, IPEIROTIS P G, JAGADISH H V, *et al*. Approximate string joins in a database (almost) for free[A]. VLDB[C]. 2001. 491-500.

[9] XIAO C, WANG W, LIN X. Ed-join: an efficient algorithm for similarity joins with edit distance constraints[A]. VLDB[C]. 2008.933-944.

[10] XIAO C, WANG W, LIN X, *et al*. Top-k set similarity joins[A]. ICDE[C]. 2009.916-927.

[11] QIN J, WANG W, LU Y, *et al*. Efficient exact edit similarity query processing with the asymmetric signature scheme[A]. SIGMOD[C]. 2011.1033-1044.

[12] ARASU A, GANTI V, KAUSHIK R. Efficient exact set-similarity joins[A]. VLDB[C]. 2006. 918-929

[13] CHAUDHURI S, GANTI V, KAUSHIK R. A primitive operator for similarity joins in data cleaning[A]. ICDE[C]. 2006.5-16.

[14] SARAWAGI S, KIRPAL A. Efficient set joins on similarity predicates[A]. SIGMOD[C]. 2004.743-754.

[15] BAYARDO R J, MA Y, SRIKANT R. Scaling up all pairs similarity query[A]. WWW[C]. 2007.131-140

[16] WANG J, LI G, FENG J. Trie-join: efficient trie-based string similarity joins with edit-distance constraints[A]. VLDB[C]. 2010. 1219-1230.

[17] WANG J, LI G, FENG J. Fast-join: An efficient method for fuzzy token matching based string similarity join[A]. ICDE[C]. 2011. 458-469.

[18] LI G, DENG D, WANG J, *et al*. Pass-join: A partition-based method for similarity joins[A]. VLDB[C]. 2011.253-264.

[19] WAGNER R A, ANDFISCHER M J. The string-to-string correction problem[A]. ACM[C]. 1974.168-173.

作者简介：

陈子阳（1973-），男，黑龙江五常人，博士，燕山大学教授、博士生导师，主要研究方向为数据库理论与系统等。

韩玉俊（1988-），男，河北邯郸人，燕山大学硕士生，主要研究方向为字符串相似性匹配。

王璿[通信作者]（1977-），女，黑龙江齐齐哈尔人，博士，燕山大学副教授，主要研究方向为高性能计算、数据库等。E-mail：wangxuan@ysu.edu.cn。

周军锋（1977-），男，陕西西安人，博士，燕山大学副教授，主要研究方向为 XML 数据库、字符串相似匹配等。