doi: 10.12068/j.issn.1005-3026.2018.07.010

一种带有长度和位置约束的字符串索引方法

于长永[1], 高 明[1], 柏禄一[1], 赵宇海[2]
(1. 东北大学秦皇岛分校 计算机与通信工程学院, 河北 秦皇岛 066004; 2. 东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

摘 要: 提出了一种基于BWT(Burrows-wheeler-transform)的字符串集合的索引方法,以解决带有匹配字符串长度和匹配子串位置约束的子串确切匹配查找问题. 讨论了BWT和基于BWT索引进行确切子串查找的基本原理. 分析了字符串集合、匹配字符串长度和匹配子串位置约束对原BWT索引的影响. 重点解决了快速地从匹配后缀位置到字符串ID和匹配子串位置的计算问题. 在3个真实的数据集上进行了比对实验,结果表明:所提出的基于BWT索引方法在没有增加原索引大小的情况下,大大提升了带有匹配字符串长度和匹配位置约束的确切子串的查找的性能,因此该算法更加适用于大规模的字符串集合的索引进行近似字符串匹配和连接.

关 键 词: BWT;字符串索引;倒排链表;字符串近似匹配;序列比对
中图分类号: TP 311.131 **文献标志码**: A **文章编号**: 1005-3026(2018)07-0959-05

A String Collection Indexing Method with String Length and Position Constraint

YU Chang-yong[1], *GAO Ming*[1], *BAI Lu-yi*[1], *ZHAO Yu-hai*[2]
(1. School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao 066004, China; 2. School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: YU Chang-yong, E-mail: yuchangyong1981@126.com)

Abstract: An index method of string collection was proposed based on BWT (Burrows-wheeler-transform) for solving the exact substring queries with string length and matching position constraints. Firstly, the BWT and exact string query based on it were discussed. Then the impact of string collection, string length and substring position upon the original BWT index was analyzed. Finally, the fast calculation problem was discussed and solved from the position of the matching suffix to the string ID and position on the string of the matching substring. The approximate string matching was conducted on three real string collections and compared the results of index method proposed and the original one. The experimental results showed that the method proposed based on BWT speeded up the process of exact substring queries with string length and matching position constraints considerably in the case of keeping the index size. Therefore, the proposed method was suitable for indexing large-scale string collection for string similarity match and joint.

Key words: BWT (Burrows-Wheeler-transform); string index; inverted list; string similarity match; sequence alignment

近似字符串匹配和连接技术,即在给定的字符串集合中搜索相似的字符串和在两个字符串集合中搜索近似的字符串对,在数据清洗[1]、拼写检查[2]、Web检索[3]、近似拷贝检测[4]、近似命名实体识别[5]、信息检索和生物信息学[6-7]中具有十分重要的应用. 因此,近年来近似字符串匹配[8-11]和近似字符串连接技术[12-13]受到了数据库领域研究人员的高度关注.

收稿日期: 2017-05-24
基金项目: 国家自然科学基金资助项目(61772124, 61332014, 61401080, 61402087); 河北省自然科学基金资助项目(F2015501049); 河北省教育厅项目(QN2014339); 中央高校基本科研业务费专项资金资助项目(N150402002).
作者简介: 于长永(1981-),男,辽宁海城人,东北大学副教授.

字符串索引技术是近似字符串匹配算法的基础和算法高效性的保障. 后缀树、后缀数组、BWT、倒排链表及其变形和 hash 表技术等字符串索引技术被广泛应用. B + 树和小波树等数据结构也被应用到 BWT 及其索引字符串的技术中. 其中,基于 BWT 的字符串索引技术及基于该技术的确切子串的查找算法因其高压缩、灵活和适应大规模数据的特点被广泛地应用到生物信息学中来解决 read mapping 问题,取得了良好的效果[5-7].

本文研究基于 BWT 的支持字符串集合上带有匹配字符串长度和匹配子串位置约束的子串查找问题的高效大规模索引方法. 该问题较基因组上的 BWT 索引结构增加了字符串集合中字符串长度、字符串上子串位置等限制条件. 本文在保持基于 BWT 的索引方法的优点的同时,提出了针对上述约束条件的优化方法. 实验结果表明,本文的索引方法能够高效地解决带有上述约束条件的子串的精确匹配查找,为进一步解决字符串近似匹配算法中最优种子选择或最优 segment 分段提供了快速的频率计算方法.

1 问题定义

定义 1(带有长度和位置约束的子串的查找问题):设 $S = \{s_1, s_2, \cdots, s_n\}$ 表示一个字符串集合;r 表示一个查询字符串;r_{sub} 是 r 的一个子串;τ 表示查找的编辑距离门限. 下面的公式表示在字符串集合 S 上的带有长度和位置约束条件的确切的子串的查找结果,

$$A(r_{\text{sub}}, \tau, S) = \{(\text{ID}_i, \text{pos}_i) \mid i = 1, 2, \cdots, k\}. \tag{1}$$

式(1)满足如下关系:

1) $S_{\text{ID}}[\text{pos}_i, \text{pos}_i + |r_{\text{sub}}| - 1] = r_{\text{sub}}$;

2) $\text{abs}(|s_{\text{ID}}(-|r|)) \leqslant \tau$;

3) $\text{abs}(\text{pos}_i - \text{pos}(r_{\text{sub}})) \leqslant \tau$.

定义 1 给出了带有字符串长度和位置约束的子串确切查找问题,该问题是现存的大多数近似字符串匹配及生物序列比对方法的基础问题. 为了快速地解决定义 1 中的问题,现有算法较多利用倒排链表对字符串集合进行索引;然而,倒排链表索引具有如下的问题:①索引的子串长度通常是固定的,无法实现对任意长度的子串进行索引;②索引的大小随着支持的子串的长度迅速增长,从而无法应用到大规模的字符串集合. 另一方面,基于 BWT 和后缀数组的索引结构被大量地应用到基因组的序列比对问题中. 该索引能够应对任意长度子串的查找,并以其良好的压缩性能能够应对大规模的字符串数据.

2 索引算法

2.1 BWT 和后缀数组

设 Σ 表示字符串 X 对应的字母表. \$ 表示一个不属于 Σ,且比 Σ 中所有字符都要小的字符. 字符串 $X = a_0 a_1 \cdots a_{n-1}$ 以字符 \$ 结尾,即 $a_{n-1} =$ '\$'. $X[i] = a_i$ 表示 X 的第 i 个字符. $X[i,j] = a_i, \cdots, a_j$ 表示 X 的一个子串,$X_i = a_i, \cdots, a_{n-1}$ 表示 X 中从 a_i 开始的后缀. 字符串 X 的后缀数组 SA 是一个由 n 个数字 $\{0, 1, \cdots, n-1\}$ 的一个排列构成的整数数组. $\text{SA}[i]$ 等于 X 中第 i 小的后缀的开始位置. 字符串 X 的 BWT 把 X 变成字符串 B,字符串 B 满足条件 $B[i] = X[\text{SA}[i] - 1]$,如果 $\text{SA}[i] \neq 0$,则 $B[i] =$ '\$'.

2.2 确切子串查找

设 $C(a)$ 表示在 $X[0, n-2]$ 中比 a 小的字符出现的总的次数. $\text{OCC}(a, i)$ 表示 $B[0, i]$ 中字符 a 出现的总的次数. 假设 $[L(W), H(W)]$ 表示子串 W 在 X 出现的后缀数组区间. Ferragina 和 Manzini 证明了 aW 的后缀数组区间可以由下面的公式进行计算[14]:

$$L(\text{aW}) = C(a) + \text{OCC}(a, L(W) - 1) + 1; \tag{2}$$

$$H(\text{aW}) = C(a) + \text{OCC}(a, H(W) - 1). \tag{3}$$

其中,aW 表示字符串 W 前面再加一个字符形成的字符串. 若 $L(\text{aW}) > H(\text{aW})$,表明 aW 不是 X 的子串. 因此,子串 W 在 X 出现的后缀数组区间可以通过反复调用上面的式(2)和式(3),从 W 的最后一个字符开始求解. 如果计算到某一步出现 $L(\text{aW}) > H(\text{aW})$,计算可以终止,并且可以得出 aW 不是 X 的子串的结论.

2.3 利用 BWT 索引字符串数组

BWT 和后缀数组是对一个字符串的变换和索引结构. 当直接利用该方法解决定义 1 中的问题时,将字符串集合 S 中的所有字符串连接成一条长的字符串,记为 $\text{Sup}(S)$,并对 $\text{Sup}(S)$ 建立索引和应用. 2.2 节中给出了确切子串查找方法,这样做出现了如下的问题:①索引到很多假阳性结果. 例如:在字符串集合 S 中存在相邻的两个字符串 $a_i =$ "$abcd$" 和 $a_{i+1} =$ "$efgh$",当查找子串 $r_{\text{sub}} =$ "$cdef$" 时,会查找到上面两个字符串给出的假阳性结果. ②给出的结果并不一定满足字符串

长度和子串位置的约束条件，并且进一步计算比较耗时. 例如：在一个例子中，处理 url 字符串集合，该集合中字符串绝大多数包含“http”，“com”和“www”等子串，若要查找包含“http”的长度不超过 10 的字符串，上面的方法返回的却是所有的包含“http”的字符串. 对于子串的位置，结果也是一样的. ③上述方法给出的后缀数组区间表示 Sup(S)上的位置. 如何将该位置映射到具体的字符串 ID 和该字符串上的位置，由于在近似字符串查找算法中该操作是十分频繁的，因此即使高效的 hash 表的方法，也是十分耗时的. 很好地解决以上具体应用中的 3 个问题是利用 BWT 方法高效地索引字符串集合来回答带有字符串长度和子串位置等约束条件的子串查找问题的关键. 为了克服上述的问题，本文提出了如下方法：

1）将字符串集合中的字符串按照长度进行分组. 首先将所有的字符串按照长度进行排序，然后从首个字符串开始按照 L_g 个不同长度的字符串分为一组的方式进行分组，假设分组后的字符串组分别为 $G_1, G_2, \cdots, G_k$. $\text{len}_{\text{start}}(i)$ 和 $\text{len}_{\text{end}}(i)$ 表示 G_i 组的字符串的最小长度和最大长度. 为了解决定义 1 中的问题，首先验证 $(\text{len}_{\text{start}}(i), \text{len}_{\text{end}}(i))$ 与 $(|r|-\tau, |r|+\tau)$ 是否有交集；如果交集不为空，利用式(2)，式(3)计算该组的后缀数组区间，否则不进行计算. 这样实现在较小的长度范围内进行确切的子串查找.

2）设 Π 表示一个不属于 Σ，且比 Σ 中所有字符都要大的字符. 对于每一组字符串 G_i，在每个字符串的尾部都插入一个字符 Π.

3）设 $\text{len}_{\text{end}}(i)$ 表示第 i 组字符串集合中最长的字符串的长度. 经过步骤 2)，所有的字符串的长度均增加一个，最长的字符串的长度变为 $\text{len}_{\text{end}}(i)+1$. 在 G_i 中所有长度小于 $\text{len}_{\text{end}}(i)+1$ 的字符串的尾部插入字符 Π，使得所有的字符串的长度均为 $\text{len}_{\text{end}}(i)+1$. 这样每组字符串集合中的字符串长度都相等，且均为 $\text{len}_{\text{end}}(i)+1$.

经过以上 3 个步骤，将每组字符串按照原来的排列顺序进行连接，形成一个长串，记为 Sup(G_i). 每组建立字符串 Sup(G_i)的 BWT 索引，记为 $\text{index\_BWT}(G_i)=\{\text{SA}, \text{OCC}_i, C_i, B_i\}$，将多组字符串的索引称为分组补全分隔符的 BWT 索引，记作 $\text{G\_BWT}(S)=\{\text{index\_BWT}(G_i)\}$.

2.4　G_BWT(S)性能分析

1）利用字符串集合 S 的分组补全分隔符的 BWT 索引 G_BWT(S)进行确切子串 r_{sub} 的查找时调用式(2)和式(3)的最小和最大次数分别为 $|r_{\text{sub}}|\times\lfloor(2\tau+1)/L_g\rfloor+\lambda$ 和 $|r_{\text{sub}}|\times\lceil(2\tau+1)/L_g\rceil+\lambda$，其中 $\lambda=1$；如果 $(2\tau+1)/L_g<1$，$\lambda=2$. 其次，假设共查找 n 组，则检索到的不符合条件的字符串的长度数量占符合条件的长度的比例为 $nL_g/(2\tau+1)-1$.

2）利用 G_BWT(S)检索到的结果不会出现假阳性结果. 原因是任意两个相邻的字符串之间至少有一个分隔符连接. 其次，假设 r_{sub} 在 G_i 中检索到的后缀数组区间为 $[1,h]$，那么 $\forall j\in[1,h]$，位置 $\text{SA}_i[j]$ 对应的 G_i 中字符串的 ID 为 $\text{SA}_i[j]/(\text{len}_{\text{end}}(i)+1)$，该位置所对应在这个字符串上的位置为 $\text{SA}_i[j]/\%(\text{len}_{\text{end}}(i)+1)$，即通过一次除法和一次取余运算即可得到 $(\text{ID}_i, \text{pos}_i)$ 信息.

3）对于没有压缩的 BWT 索引，即 SA_i，OCC_i 完全存储的情况下，分组补全分隔符的索引与不做任何操作直接建立 BWT 索引的存储空间花费是相同的. 因为每个组只需要存储字母表 Σ 中字符所对应的后缀数组部分，补全的分隔符部分位于整个后缀数组的最后，不需要进行存储. OCC_i, B_i 也是仅需要存储 Σ 中字符对应的行和 W 串. 但是，当索引进行压缩存储时，由于需要利用 lf-mapping 来计算后缀数组的值，会用到字符 Π 对应的后缀位置和 OCC 的值，因此需要每个组存储补全分隔符连接长串的完整 index. 这时索引的大小要有所增加，但也可通过分组的大小和索引的压缩参数来控制索引的大小，对索引性能的影响很小.

3　实验结果与讨论

3.1　数据集和实验环境

本文方法在 Win7x64 位操作系统下 Visual Studio2010 编程环境下利用 C++语言实现. CPU 配置为 Intel(R) Core(TM) i7-2600 CPU processor @3.40 GHz 并配备 16 GB 内存. 数据集见表 1.

表 1　数据集
Table 1　Dataset

数据集	N	n	Avg-Len	Size/MB
DBLP	860 751	1 000	106	88
PubMed	4 000 000	400	100.6	383
URL	1 000 000	200	28	26.7

表 1 中，N 和 n 分别表示各个数据集中包含

的待查询字符串和查询字符串的数量. 本文所有的测试结果均在单线程的设置条件下测得.

3.2 构建索引 G_BWT(S)

对每个字符串集合数据进行分组. 在分组实验中考察两个指标:①每组集合的大小;②每组字符串覆盖的长度范围. 由于分组字符串集合的大小直接决定构建 G_BWT(S)过程中对内存的需求,所以将最大的字符串集合分组的大小设置为 50 MB. 另外,尝试了各种长度范围 L_g 的取值,在查询范围 $1 \leqslant \tau \leqslant 12$ 的情况下,通过实验测得 $L_g = 5$对应的实验结果是最优的. 在这样的条件下,DBLP,PubMed 和 URL 数据集分别被分成了 80,61 和 20 个组.

对于每个组,首先计算 SA_i,然后利用公式 $B[i] = X[SA[i] - 1]$计算 $B[i]$,最后计算 OCC_i 和 C_i 两个数组. 其中 SA_i 是最耗时的,计算 SA_i 的过程实质是一个排序过程. 由于排序的后缀数量太大,并且有些后缀需要比较多次才能得出大小关系,所以,通常花费很长的时间. 本文利用 C++语言编写程序,利用 B+Tree 数据结构计算 SA_i. 得到的最终 G_BWT(S)信息见表 2 和表 3. OCC 和 SA 分别需要 $|\Sigma| \cdot |X|$ 和 $|X|$ 个整数的存储空间. 如果按照一个整数需要 4 个 Bytes,一个字符需要 1 个 Byte 来计算,OCC 和 SA 分别需要原字符串 X 的 104 和 4 倍的存储空间. 对于 PubMed 和 DBLP 数据集,分别需要大约 9 GB 和 33 GB的存储空间. 因此,对其进行压缩存储. 表 2 中, OCC_{gap}和 SA_{gap}分别表示 OCC 和 SA 压缩存储的参数.

表 2 G_BWT(S)参数
Table 2 Parameters of G_BWT(S)

数据集	OCC_{gap}	SA_{gap}	N_g	t/s
PubMed	50	5	61	1306
DBLP	10	4	80	208
URL	10	0	20	202

表 3 G_BWT(S)大小
Table 3 Size of G_BWT(S)

数据集	OCC/MB	SA/MB	C/KB	B/MB
PubMed	1 124	557	11	395
DBLP	999	125	12	77
URL	331	162	3	24

3.3 基于 G_BWT(S)确切子串查找

为了验证 G_BWT(S)的可用性和高效性,利用其进行近似字符串查找. 表 1 中给出了各个数据集上的查询个数. 在 PubMed 和 DBLP 数据集上的查询范围为 $1 \leqslant \tau \leqslant 12$,URL 数据集上的查询范围为 $1 \leqslant \tau \leqslant 5$. 为了验证本文检索方法的高效性,利用直接连接数据集,构建 BWT 索引的方法进行了对比实验. 由于直接连接字符串时,各个字符串的长度不同,利用哈希算法实现从后缀位置到字符串 ID 和该位置在该字符串上的位置的计算. 实验中,对于查询范围 τ,将查询字符串分为 $\tau+1$段 segments,利用本文提出的方法和上述对比方法计算每段 segments 的后缀数组区间,并对其中每个位置计算对应字符串的 ID 和位置 POS. 图 1 给出了确切子串的查询时间. 随着 τ 的增加,每个 query 字符串被分成 $\tau+1$ 个 segments,segment 的长度变得更短,因此,查询更加频繁. 要计算的后缀位置的数量迅速增加. 图 1 给出了在每个 τ 值下,计算所有 segments 匹配的后缀位置及映射成数据库中字符串的 ID 和 POS 的总的计算时间. 因此,随着 τ 的增加,时间花费快速增加. 图 1 中"*_a"表示本文提出的索引方法. "*_b"表示直接连接数据库中字符串并构建 BWT 索引的方法. 图 1 表明,本文所提方法具有较高的实用性和高效性.

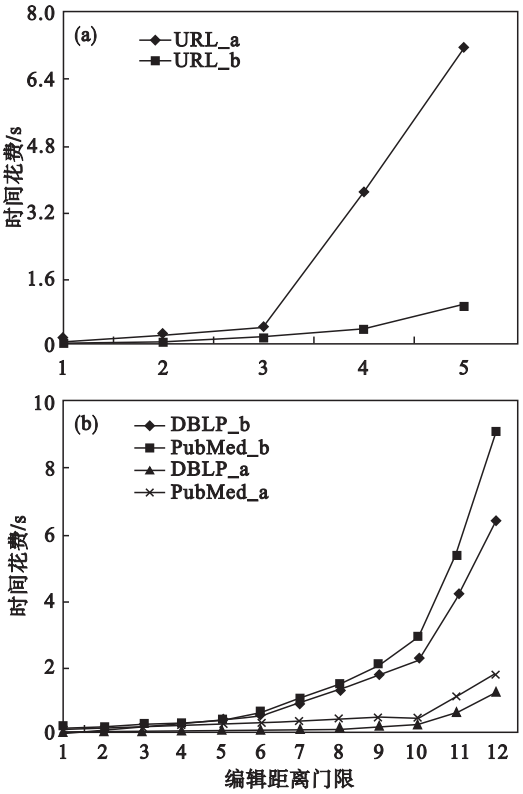

图 1 确切子串查询时间
Fig. 1 Time cost for exact substring query
(a)—URL 数据集; (b)—DBLP 和 PubMed 数据集.

4 结　论

1）优化后索引，在没有压缩存储的情况下和元索引大小相同. 当进行压缩存储时，索引大小可以通过分组和压缩参数来进行控制.

2）克服了结果中出现假阳性的问题，并仅用一次除法和取余运算实现从后缀数组位置到字符串 ID 和字符串上子串匹配位置的计算.

3）在 DBLP，PubMed 和 URL 三个数据集上，在没有增加索引大小的情况下，索引速度均有所加快，索引速度提升 3 ~ 5 倍，实现了更加高效的索引.

参考文献：

[1] Chaudhuri S, Ganti V, Kaushik R. A primitive operator for similarity joins in data cleaning [C]// International Conference on Data Engineering. Atlanta, 2006: 5 – 17.

[2] Xiao C, Wang W, Lin X, et al. Efficient similarity joins for near duplicate detection [C]// International Conference on World Wide Web. Beijing, 2008: 131 – 140.

[3] Chaudhuri S, Ganjam K, Ganti V, et al. Robust and efficient fuzzy match for online data cleaning [C]//ACM SIGMOD International Conference on Management of Data. San Diego, 2003: 313 – 324.

[4] Deng D, Li G, Feng J. A pivotal prefix based filtering algorithm for string similarity search [C]//ACM SIGMOD International Conference on Management of Data. San Diego, 2014: 673 – 684.

[5] Li C, Wang B, Yang X. VGRAM: improving performance of approximate queries on string collections using variable-length grams [C]// International Conference on Very Large Data Bases. Vienna, 2007: 303 – 314.

[6] Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform [J]. *Bioinformatics*, 2009, 25 (14): 1754 – 1760.

[7] Qin J, Wang W, Lu Y, et al. Efficient exact edit similarity query processing with the asymmetric signature scheme [C]// ACM SIGMOD International Conference on Management of Data. Athens, 2011: 1033 – 1044.

[8] Sarawagi S, Kirpal A. Efficient set joins on similarity predicates [C]// ACM SIGMOD International Conference on Management of Data. Paris, 2004: 743 – 754.

[9] Sokol D, Benson G, Tojeira J. Tandem repeats over the edit distance [J]. *Bioinformatics*, 2007, 23(2): e30 – e35.

[10] Wang W, Xiao C, Lin X, et al. Efficient approximate entity extraction with edit distance constraints [C]// ACM SIGMOD International Conference on Management of Data. Rhode, 2009: 759 – 770.

[11] Wang W, Qin J, Xiao C, et al. VChunkJoin: an efficient algorithm for edit similarity joins [J]. *Transactions on Knowledge & Data Engineering*, 2013, 25(8): 1916 – 1929.

[12] Xiao C, Wang W, Lin X. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints [J]. *Proceedings of the VLDB Endowment*, 2008, 1(1): 933 – 944.

[13] Yang X, Wang Y, Wang B, et al. Local filtering: improving the performance of approximate queries on string collections [C]// ACM SIGMOD International Conference on Management of Data. Victoria, 2015: 377 – 392.

[14] Ferragina P, Manzini G. Opportunistic data structures with applications [C]// Foundations of Computer Science. Beijing, 2002: 390 – 399.