软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(8):2175−2195 [doi: 10.13328/j.cnki.jos.005124]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于数字签名与 Trie 的保序子矩阵约束查询*

姜 涛, 李战怀, 尚学群, 陈伯林, 李卫榜, 殷知磊

(西北工业大学 计算机学院,陕西 西安 710072)

通讯作者: 李卫榜, E-mail: wbli2003@163.com

摘 要: 目前,基因芯片技术飞速发展,促使生物学家积累了大量的不同实验条件下的基因表达数据.事实证明,基因芯片数据分析在理解基因功能、基因调控和分子生命过程中发挥着重要作用.保序子矩阵(order-preserving submatrix,简称 OPSM)是基因芯片数据分析技术中的一种有效模型,其可以发现在部分基因和不同实验条件下具有相同表达趋势的聚类.在分析基因表达机理的过程中,OPSM 的检索无疑节省了生物学家的时间与精力.目前,OPSM 的查询主要是基于关键词的检索方法,但是分析者对结果具有微弱的控制力.通常,分析者所能决定的临时的参数设置往往偏离其领域知识,致使检索结果与真实想要的结果相去甚远.为了解决上述问题,提出两类基于数字签名与 Trie 的 OPSM 索引与约束查询方法.在真实数据上进行了大量的实验,实验结果表明,所提出的方法具有良好的有效性与可扩展性.

关键词: 基因表达数据;OPSM(order-preserving submatrix);约束查询;数字签名;Trie;枚举序列
中图法分类号: TP311

中文引用格式: 姜涛,李战怀,尚学群,陈伯林,李卫榜,殷知磊.基于数字签名与 Trie 的保序子矩阵约束查询.软件学报,2017,28(8):2175−2195. http://www.jos.org.cn/1000-9825/5124.htm

英文引用格式: Jiang T, Li ZH, Shang XQ, Chen BL, Li WB, Yin ZL. Constrained query of order-preserving submatrix based on signature and Trie. Ruan Jian Xue Bao/Journal of Software, 2017,28(8):2175−2195 (in Chinese). http://www.jos.org.cn/1000-9825/5124.htm

Constrained Query of Order-Preserving Submatrix Based on Signature and Trie

JIANG Tao, LI Zhan-Huai, SHANG Xue-Qun, CHEN Bo-Lin, LI Wei-Bang, YIN Zhi-Lei

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: The advances of microarray technology have made large amount of gene expression data available from a variety of different experimental conditions. Analyzing the microarray data plays a key role in understanding gene functions, gene regulation and cellular process. Order-Preserving Submatrix (OPSM) is an important model in microarray data analysis, which captures the identical tendency of gene expressions across a subset of conditions. In the process of analyzing mechanism of gene expression, OPSM search undoubtedly saves the time and effort of biologists. However, OPSM retrieval mainly depends on keyword search, resulting a weak control on the obtained clusters. Typically, the analyst can determine the ad-hoc parameters which are far from the declarative specification of desired properties on operation and concept. Motivated by obtaining much more accurate query relevancy, this paper proposes two types of OPSM

* 基金项目: 国家重点基础研究发展计划(973)(2012CB316203); 国家自然科学基金(61033007, 61272121, 61332014, 61572367, 61472321, 61502390); 国家高技术研究发展计划(863)(2015AA015307); 中央高校基本科研业务费专项资金(3102015JSJ0011); 西北工业大学研究生创业种子基金(Z2012128)

Foundation item: National Program on Key Basic Research Project of China (973) (2012CB316203); National Natural Science Foundation of China (61033007, 61272121, 61332014, 61572367, 61472321, 61502390); National High-Tech R&D Program of China (863) (2015AA015307); Fundamental Research Funds for the Central Universities (3102015JSJ0011); Graduate Starting Seed Fund of the Northwestern Polytechnical University (Z2012128)

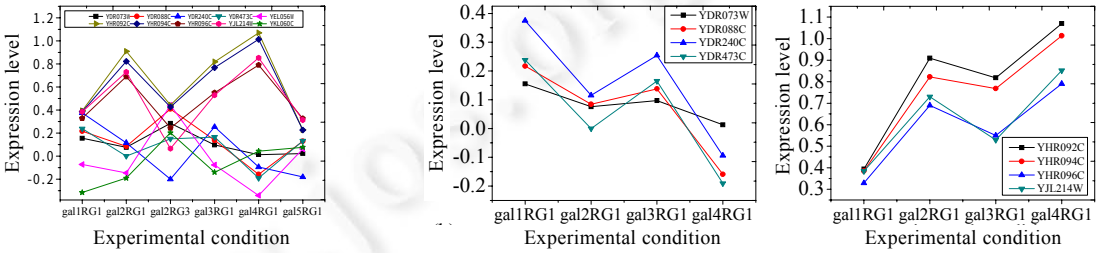
收稿时间: 2016-01-20; 修改时间: 2016-05-20; 采用时间: 2016-07-19; jos 在线出版时间: 2016-10-11

CNKI 网络优先出版: 2016-10-12 16:26:34, http://www.cnki.net/kcms/detail/11.2560.TP.20161012.1626.003.html

indexing and constrained query methods based on signature and Trie. Extensive experiments conducted on real datasets demonstrate the proposed methods have better behaviors than the state-of-the-art methods on efficiency and effectiveness.

Key words: gene expression data; order-preserving submatrix (OPSM); constrained query; signature; Trie; enumerated sequence

基因芯片技术加快了基因表达数据的积累,数据的积累又促进了大量用来挖掘新的有用知识的高性能算法的设计与实现.基因表达数据广泛应用于基因组研究,因为其方便了疾病亚型的发现与诊断.基因表达数据一般组织成矩阵的形式,其中的行代表基因,列代表实验条件,每个元素代表某个基因在特定实验条件下的表达水平.图 1(a)给出一个真实的基因表达数据集(http://genomebiology.com/content/supplementary/gb-2003-4-5-r34-s8.txt),其中,横轴表示实验条件,纵轴表示表达水平,每一条曲线代表某个基因在若干个实验条件下的表达水平的变化趋势.因为基因之间更趋向于在部分条件下有一致的上升下降趋势,所以图 1(a)中的升降趋势并不明显.图 1(b)是从图 1(a)中挖掘出来的两个 OPSM,其按照原始实验条件的先后顺序来表示.实际上,在发现过程中,先将每个基因的表达值按大小排序,接着替换为列标签,最后寻找列标签序列的最长公共子序列.两图清晰、明了地展示了相对应的一致表达趋势.通常,从中挖掘出来的子矩阵称为保序子矩阵(order-preserving submatrix,简称 OPSM).该模型在基因表达数据的挖掘与分析中发挥着重要作用[1].



(a) 基因表达数据示例(10 个基因,6 个实验条件) (b) OPSM

Fig.1 Toy example

图 1 举例

目前,聚类方法已广泛应用于基因表达数据分析领域,但是其要求所有实验条件必须在同一聚类中.然而,相关基因不一定在所有实验条件下共表达.基于上述观点与现实,聚类不必过分强调聚类中包含全部的实验条件或者基因.相反地,两个维度可以同时聚类,该方法称为双聚类或者子空间聚类.它可以挖掘出在部分实验条件下具有相似表达水平或趋势的若干基因的集合.双聚类的概念最初由 Hartigan 等人[2]提出,其作为对矩阵中的行与列同时聚类的一种方法,并将其命名为 Direct 聚类.Cheng 等人[3]提出了基因表达数据的双聚类,并引入了元素残差以及子矩阵的均方残差的概念.Ben-Dor 等人[1]介绍了一种特殊的双聚类模型 OPSM,并证明了它是 NP 难问题.随后,研究者们提出了基于定量测度[4−14]和定性测度[15−33]的 OPSM 挖掘方法.定量测度包括均方残差 MSR[3]、平均相关值 ACV[10]、平均斯皮尔曼秩相关系数 ASR[10]、平均一致性相关指数 ACSI[10]等.定性测度包括上升、下降和无变化等衡量标准[24,28].上述方法大多数是批量挖掘多种类型的 OPSM,在搜索或者查询特定的某个 OPSM 时显然太过笨拙.因为其或者经过多轮迭代或者批量挖掘之后从中选取目标聚类,所以其在查询 OPSM 的过程中性能较差.尽管基于查询的双聚类方法[34]可以在一定程度上满足需求,但是其输入的关键词和参数并不能真正表达分析者的领域知识.从文献中可知,分析者的领域知识可以用约束来表示.现有的约束在基因表达数据分析[35−37]中也有应用,其主要辅助共表达聚类的批量挖掘,目前,约束与查询[38]还没有共同使用.基于约束查询的 OPSM 分析对探索基因协同表达、调控网络的创建具有重要意义.本文主要解决的问题是,从 OPSM 数据集中搜索与查询特定的符合分析者要求的若干 OPSM.

基于约束条件的 OPSM 查询存在以下挑战.

(1) OPSM 数据量庞大,如何快速遍历如此大的数据具有难度.因为 OPSM 间具有交叉重叠,所以基因或实

验条件会出现在多个 OPSM 中.虽然具有冗余,但又是必须的,因为其代表了不同的生物学功能团.

(2) 如何规范而有效地将分析者的领域知识转化为自定义约束条件以及如何规范有效地利用这些规范,都是需要解决的问题.现有文献中的自定义约束主要有 must-link 与 cannot-link 两类[37].如果用户主观上要求基因 g_4 与 g_7 必须同时出现在一个功能团中,就可以应用 must-link 加以约束,以减少不相关的搜索.同样地,若生物学家意在搜寻含有基因 g_4 但无 g_1 的功能团,就可以应用 cannot-link 加以限制.另外,我们观察到本研究与现有方法的不同点,即对表达值进行排序与实验条件(列)标签的替换,所以可以在实验条件的表达值上加偏序之后,定义一个 interval 约束(实验条件之间的间隔数目的限制)来减少搜索空间.若同时对 OPSM 的行列数目加以约束,则同样能够加速搜索速度.

(3) 如何建立数据量小且查询速度快的索引具有挑战性.我们观察到,每个基因名序列编码较长,若在匹配过程中直接利用,会极大地降低搜索速度.另外,实验条件序列有序且较长,也不能简单地通过变换编码的方式来解决.为此,亟待设计高效的索引与搜索算法来解决 OPSM 的检索问题.

为了解决上述问题,本文首先将 OPSM 中的基因名序列转换成 0/1 数字签名,并将其放入 Tire 树中建立 sTrie 索引,减少了内存的消耗.又对 sTrie 中的单分支节点进行压缩,设计了 cTrie 索引.接着,针对 OPSM 中的实验条件序列提出了基于原始或枚举实验条件的索引 tTrie.然后,为了快速通过自定义约束进行搜索,提出了基于上述两类索引的自顶向下与自底向上搜索方式相结合的查询方法.其旨在利用一类索引从另一类索引中搜索出来的候选集,从而加速搜索.以自顶向下搜索的 sTrie(或 cTrie)与以自底向上的 tTrie 搜索方式相结合后,比蛮力等初步方法在性能上有较大的提升.其搜索时间复杂度为 $O(\zeta b+\omega m)$,其中的 ζ 为平均搜索分支数,b 为数字签名位数,ω 为候选分支数,m 为平均分支长度.为了进一步解决 sTrie 和 cTrie 所产生的候选集数目庞大以及遍历效率低的问题,提出了以自顶向下搜索的 tTrie 与以自底向上的 sTrie(或 cTrie)搜索方式相结合的方法.其搜索时间复杂度为 $O(\varepsilon+\zeta b)$,其中,ε 为平均搜索节点数.

本文的主要工作和创新点如下.

(1) 提出了基于数字签名和 Trie 的基因序列与实验条件序列索引方法(第 2 节),其中包括数字签名位数的计算方法、0/1 Trie 的分支/节点的压缩方法以及表头的创建方法.

(2) 提出了自顶向下与自底向上相结合的查询方法(第 3 节),它能够保证不同情况下的查询效率.

(3) 在真实数据集上做了大量实验,实验结果表明,所提出的方法具有良好的查询精确性和可扩展性(第 4 节).

1 问题描述

本节主要介绍相关概念与解决 OPSM 约束型查询所用到的索引和查询框架,用到的相关符号见表 1.

Table 1 Notations used throughout the paper

表 **1** 本文用到的相关符号

符号	描述	符号	描述
G	基因集合	T	实验条件集合
g	部分基因	t	部分实验条件
g_i	基因 i	t_i	实验条件 i
$D(G,T)$	源数据集	e_{ij}	g_i 在 t_i 下的表达值
r_i	行聚类	l_i	列聚类
M	OPSMs	M_i	一个 OPSM
M^g	M 中的行	M^t	M 中的列

如果后文中没有特殊说明,基因与行、实验条件和列将交替使用,表示同样的含义.

定义 **1**(保序子矩阵(**OPSM**)). 给定数据 $D(n\times m$ 的矩阵),$M_i(g,t)$ 是 D 中的一个子矩阵,且 $g\subseteq G,t\subseteq T$.若 M_i 是一个保序子矩阵,则有 g 中的每一行数据 e 关于列标签子集 t 的排列严格单调递增,即 $e_{i1}\le e_{i2}\le\ldots\le e_{ij}\le\ldots\le e_{ik}$,其中,$(i1,..ij,\ldots,ik)$ 是列标签 t 的一个排列.

例 1:图 1(b)和图 1(c)是从图 1(a)中挖掘出来的两个 OPSM,其按照原始实验条件的先后顺序来表示.实际上,在发现过程中,先将每个基因的表达值按大小排序,接着替换为列标签,最后寻找列标签序列的最长公共子序列.

定义 **2**(保序子矩阵的约束查询). 给定自定义约束集合 C(例如必须连接、不能连接、间隔约束和数量约束等)与 OPSM 的集合 M,保序子矩阵的约束查询是在 M 中寻找符合 C 的保序子矩阵集合 Q_C 的过程.

定义 **3**(必须连接(**must-link**)). 若行 g_a 和 g_b(或列 t_a 和 t_b)参与必须连接约束,表示为 $c_{must}(g_a,g_b)$(或 $c_{must}(t_a,t_b)$),那么必须返回处于同一个行聚类 M^g(或同一个列聚类 M^t)的 OPSM.

例 2:从表 2 数据中查询 g_4,g_5,g_7 所在的某功能团,将记住的基因名 g_5,g_7 作为必须连接约束(即 $c_{mst}(g_5,g_7)$)提前输入.在查询时,将需要遍历的数据从 8 条减少到 2 条(即 M_0 和 M_2),减少了不相关数据的干扰,降低了计算量.

Table 2 A toy example of OPSM dataset

表 **2** OPSM 数据集举例

OPSM 代号	基因名	实验条件	基因名的签名
M_0	$g_1\ g_3\ g_5\ g_7$	4 1 3 0	0101
M_1	$g_1\ g_2\ g_3$	0 3 1 4	0111
M_2	$g_4\ g_5\ g_7$	2 4 5 3 0	1101
M_3	$g_0\ g_1\ g_3\ g_4$	1 2 0 4 5	1101
M_4	$g_1\ g_3\ g_5$	2 5 6	0101
M_5	$g_1\ g_2\ g_3\ g_5$	2 3 4 5	0111
M_6	$g_0\ g_1\ g_2\ g_6$	4 6 2 3	1110
M_7	$g_1\ g_4\ g_6$	5 3 6 1 0	1110

定义 **4**(不能连接(**cannot-link**)). 若行 g_a 和 g_b(或列 t_a 和 t_b)参与不能连接约束,表示为 $c_{cant}(g_a,g_b)$(或 $c_{cant}(t_a,t_b)$),那么必须返回不在同一个行聚类 M^g(或同一个列聚类 M^t)的 OPSM.

例 3:从表 2 数据中查询 g_4 但非 g_1 所在的某个功能团,将基因名 g_4,g_1 作为不能连接约束(即 $c_{cant}(g_4,g_1)$)提前输入.在查询时,将需遍历的数据从 8 条减少到 1 条(即 M_2),减少了不相关数据的干扰,降低了计算量.

定义 **5**(间隔约束(**interval**)). 若在列集上定义一个偏序(<),则该集合上的间隔约束表示为 $c_{int}(t_a,t_b,i)$,i 为非负整数,表示 t_a 和 t_b 在列聚类 M^t 的每一个子集 l_j 中的间隔必须为 i.如果 i 为 0,则表示 t_a 和 t_b 紧挨着;如果 i 为正整数,表示 t_a 和 t_b 间距为 i.

例 4:从表 2 数据中查询间隔为 1 的实验条件 4 和条件 3 所在的某个功能团,将其转化为间隔约束 $c_{int}(t_4,t_3,1)$(即间隔 1 的条件 4 和条件 3 在聚类 M^t 中).在查询时,将候选集从 8 条减少到 2 条(即 M_0 和 M_2),减少了数据的干扰与计算量.

定义 **6**(数量约束(**count**)). 给定行阈值$|g|$、列阈值$|t|$以及二元关系 $u,v\in\{\neq,=,\leq,\geq,<,>,\not<,\not>\}$,则行列集上的数量约束表示为 $c_{cnt}(|g|_u,|t|_v)$,其中,$|g|_u$ 表示查询的 M_i 的行数要 u 于$|g|$,$|t|_v$ 表示查询的 M_i 的列数要 v 于$|t|$.

例 5:从表 2 数据中查询行列数目分别为 3 和不小于 5 的某个功能团,将其转化为数量约束 $c_{cnt}(3_=,5_\geq)$.在查询时,将需要遍历的数据从 8 条减少到 2 条(即 M_2 和 M_7),减少了候选集的数量,也降低了计算量.

通常,序列查询可以转换为集合包含问题[39].然而,OPSM 查询问题转化为集合包含问题之后,其中的实验条件还需要鉴定顺序.为此,针对 OPSM 基因部分的查询,利用基于数字签名(生成方法见第 2.1 节)的 Trie 来解决.对于 OPSM 实验条件部分的查询,利用基于枚举序列的前缀树索引来解决.OPSM 查询分为以下两个主要步骤.

1) 创建索引.这是一个对 OPSM 中基因名与实验条件部分进行预处理的过程.
   (i) 首先,将基因名转换为数字签名,如果关键词 g 包含于某个 OPSM,那么 g 的数字签名也包含于该 OPSM 的数字签名之中,且利用数字签名的查询更快捷;然后,将数字签名放在 Trie 中,这样可以节省部分内存空间.
   (ii) 首先,枚举不超过一定长度的实验条件序列;然后,将该序列放置于前缀树之中;最后,建立一个辅助表头,方便对索引的遍历与定位.
2) 查询处理.包含搜索与验证两个子步骤.

(i) 搜索,利用基因关键词的数字签名或者实验条件序列或者子序列来遍历 Trie 与前缀树,并获取相应的聚类 M_i 作为候选集 C_q 中的元素.

(ii) 验证,检测候选集 C_q 中的每个聚类 M_i 是否真正满足所有约束.

2 索引方法

实现 OPSM 约束查询的最简单的方法是蛮力搜索,即每次查询都完整地扫描一遍数据集,从中选择符合条件的结果.该方法只适用于小数据集,对大规模数据集却无能为力.为了使搜索更高效,首先分析如表 2 所示的 OPSM 数据集,知道完成 OPSM 的约束查询需要对基因名(行)与实验条件(列)两部分分别进行处理与匹配.为此,根据基因名与实验条件序列各自的特点,分别提出了两种索引方法.由于基因名序列没有元素的先后之分,设计了基于数字签名与 *Trie* 的索引方法.根据实验条件有先后顺序的特点,提出了基于枚举序列的索引策略.

2.1 基于数字签名与**Trie**的索引(**sTrie**)

数字签名 signature[39]是一个长度为 b 的位域,其用来代表或者近似代表一个集合.假如一个集合 t 中有 x 个元素,且每一个元素用 0 到 x–1 之间的整数来表示,那么这个集合的数字签名的生成方法为:将第(i mod b)位置置为 1.生成的数字签名即为集合 t 映射到 b 位上的一个压缩位图.表 2 数字签名列展示了为每一个基因名序列设计的 4 位数字签名.本文将数字签名的最左侧作为第 0 位,左侧第 2 位作为第 1 位,依次下去.在映射过程中,g_0 映射到第 0 位,g_1 映射到第 1 位,以此类推.注意,M_2 与 M_3 中的不同基因名序列具有相同的数字签名 1101.

为了寻找最优的数字签名位数,利用实验部分表 3 所示的数据,测试在 12 000,6 000,3 000,1 000,600,300,150,75,50,25 位数字签名情况下的候选集中元素的数目.从图 2 中可以看出:由列数较少的基因表达数据生成的 OPSM 数据集 D_1 和 D_2,随着数字签名位数的减少,候选集中元素数目增加的速度相对较快.在 600 位数字签名之前,候选集数目增加相对平稳,而在其后急速增加.由列数较多的基因表达数据生成的 OPSM 数据集(D_3,D_4,D_5,D_6)则在 150 位数字签名之后才快速增加.由此得出如下结论:对由列数较少的基因表达数据生成的 OPSM 数据集进行数字签名转化时,数字签名的位数应该稍多一些.

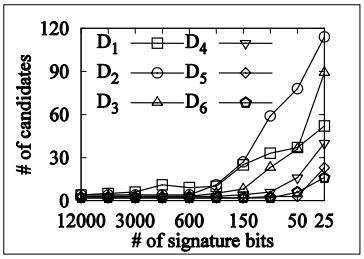


Fig.2 Candidate numbers with different bits of signature

图 2 不同位数的数字签名下的候选集中元素数目

基于数字签名的 Trie 索引的创建过程(算法 1)如下:对于每一个 OPSM 中的基因名序列,首先将数字签名 *sig* 中的相应位置置为 1,并将当前节点指向根节点(第 2 行);接着,对于 *sig* 中的每一位(第 3 行),如果当前节点的孩子中没有等于该值的节点,就创建一个具有该值的孩子节点,并将当前节点指向该节点(第 4 行、第 5 行),否则只需指向该孩子节点即可(第 5 行);直到 *sig* 的最后一位时,将当前节点设为叶子节点并存储 OPSM 的 ID 号集合(第 6 行).对于每一个 id 号(第 7 行),如果 *opsmIdTable* 表头中存在该 id,则将当前节点加入以 id 为键的集合中;否则,新建一个分别以 id 为键、当前节点地址为值的键值对放入 *opsmIdTable* 表头中(第 8 行).创建表头的目的是为了方便后文的通过获取到的 id 编号以自底向上的方式来检验候选 id 是否符合条件.

算法 **1**. 基于数字签名的 Trie(sTrie).

输入:OPSM 数据集 M.

输出:基于数字签名的 Trie sTrie.

1. **while** *M* 中存在下一条数据 **do**
2. *sig*←0; *sig*⊨1<<*m_sig*(*g*[*i*]); *curNode*←*gRoot*;
3. **for** *i*←0 to *sig.length*−1 **do**
4. **if** *curNode* 的孩子节点中不存在 *sig*[*i*] **then** 将 *sig*[*i*]加入到 *curNode* 的孩子节点中;
5. *curNode*←*curNode* 指向孩子节点 *sig*[*i*];
6. 将 *curNode* 设为末节点;将 *M* 中该 OPSM 的 ID 存入 *curNode*;
7. **for** ID 中的每一个 id **do**
8. **if** *opsmIdTable* 中存在 id **then** 将 *curNode* 添加到 *opsmIdTable* 中 id 所在单元;
9. **else** 将(*id*,*curNode*)加入 *opsmIdTable*;

例 6(sTrie 索引的创建):表 2 展示了一个 OPSM 样例数据集,该数据将用来作为后文创建索引与演示查询方法的示例.基于数字签名的 Trie 的创建结果如图 3(a)所示.

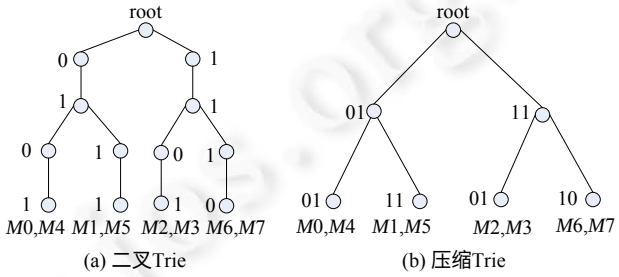


Fig.3 0/1 Trie example

图 3 0/1 Trie 例子

引理 **1**. 基于数字签名的索引方法能够保证查询结果的完整性.

证明:根据数字签名的性质,如果基因名约束转换后的集合 *c* 包含于某一个 OPSM 的基因名集合 *g* 中,即 $c \subseteq g$,则有 $sig_c \subseteq sig_g$,其中,$sig_c \subseteq sig_g :=sig_c \& \sim sig_g=0$,&与~分别是二进制中的按位与和求补操作.定理得证. □

因为索引(包括后文索引)的更新方法与索引的创建方法类似,所以在更新索引时只需调用类似于索引创建的更新方法即可.由于索引的创建方法已有详细介绍,这里不再给出具体的算法.

2.2 基于数字签名与**Trie**的压缩索引(**cTrie**)

第 2.1 节的基于数字签名的 sTrie 索引方法的一个缺点是其生成了许多不必要的节点,这些节点仅有 1 个孩子,也叫做单分支节点.从图 3(a)中也可以发现上述现象.对于 *k* 个拥有 *b* 位的数字签名,如果没有上述单分支节点,那么该 sTire 应该包含 2*k* 个节点.然而通常情况下,数字签名越长,sTire 的单分支节点就越多,节点越多,需要占据的内存也就越多.同时,在遍历过程中也可能消耗更多的 CPU 时间.为了解决上述问题,对上述方法进行改进,将每个节点的单分支节点压缩到一个节点中.

为了避免单节点分支的出现,将二分 sTrie 中的单分支节点放入一个节点中,这样就保证了除叶子节点之外的所有节点都有两个分支.下面用一个例子来说明如何建立压缩 Trie(即 cTrie).图 3(b)中展示了一个图 3(a)中对应的压缩索引 cTire.对于左分支,只有第 2 个位置有分裂,所以将第 1 节点、第 2 节点合并成一个节点 01,其后的两个分支分别合并成 01 和 11.右分支也只有第 2 个位置有分裂,所以将前两个节点合并成一个节点 11,其后的两个分支分别合并成 01 和 10.这样就将图 3(a)中的 13 个节点减少到图 3(b)中的 7 个节点.

下面给出基于数字签名的压缩 Trie 索引(cTrie)的创建过程(算法 2)与算法 1 的不同之处,其中,算法 1 的第 3 行~第 5 行替换为算法 2 的第 3 行~第 16 行.在处理第 1 个 *sig* 时,直接将该 *sig* 序列放入根节点的一个孩子中(第 4 行、第 5 行).对于之后的 *sig* 序列,首先获取当前节点(首先为根节点)的孩子节点 *child*,并找到节点 *child*

中所存储的 sig_{old} 序列与新的 sig_{new} 最长公共前缀 *lcp*(第 8 行).如果 *lcp* 的长度为 0,则将该 sig_{new} 作为当前节点的另外一个孩子节点(第 15 行).如果节点 *child* 中的某个孩子节点为 *lcp*,则将当前节点指向 *child* 节点的 *lcp* 节点(第 14 行).如果 *lcp* 长度大于 1 且 sig_{old} 片段有剩余,则将现有节点的 *key* 换为 *lcp*,同时创建一个新的节点作为现有节点的孩子节点(第 11 行~第 13 行).如果 *lcp* 长度大于 0,则将标志位置为 false(第 16 行).*opsmIdTable* 表头的创建过程与 sTrie 类似(第 18 行、第 19 行).

算法 **2**. 基于数字签名的压缩 Trie(cTrie).

输入:OPSM 数据集 *M*.

输出:基于数字签名的压缩 Trie cTrie.

1. **while** *M* 中存在下一条数据 **do**
2. *sig*←0; *sig*|=1<<*m_sig*(*g*[*i*]); *curNode*←*gRoot*;
3. **while** *sig* 不为空 **do**
4. **if** *curNode* 为根节点且没有孩子节点 **then**
5. 将 *sig*[*i*]加入到 *curNode* 的孩子节点中;*curNode*←*curNode* 指向孩子节点 *sig*[*i*];*sig*←"Φ";
6. **else if** *curNode* 有孩子节点 **then** *flag*←true;
7. **while** *flag* 为真且 *curNode* 中存在下一个孩子 **do**
8. *key*←获取 *currentChild* 基因名;*lcp*←寻找 *LongestCommonPrefix*(*key*,*sig*);
9. **if** *lcp.length*=*sig.length* **then** *sig*←"Φ"; **else** *sig*←*sig* 中截取从 *lcp.length* 到 *sig.length* 序列;
10. **if** *lcp.length*<*key.length* **then** *key_remain*←*key* 中截取从 *lcp.length* 到 *key.length* 的序列;
11. **if** *lcp.length*>0 & *key_remain.length*>0 **then** *val*←*currentChild*;*curNode* 孩子节点去除 *key*;
12. 将 *lcp* 加入 *curNode* 孩子节点;将(*key_remain*,*val*)放入 *curNode* 中孩子 *lcp* 的孩子;
13. *curNode*←*curNode* 中的孩子节点 *lcp*;
14. **else if** *lcp.length*>0 & *key_remain.length*=0 **then** *curNode*←*curNode* 中的孩子节点 *lcp*;
15. **else if** *lcp.length*=0 **then** 将 *sig* 加入 *curNode* 孩子节点;*curNode* 指向孩子节点 *lcp*;*sig*←"Φ";
16. **if** *lcp.length*>0 **then** *flag*←false;
17. 将 *curNode* 设为末节点;将 *M* 中该 OPSM 的 ID 存入 *curNode*;
18. **for** ID 中的每一个 id **do**
19. **if** *opsmIdTable* 中存在 *id* **then** 将 *curNode* 添加到 *opsmIdTable* 中 id 所在单元;
20. **else** 将(*id*,*curNode*)加入 *opsmIdTable*;

例 7(cTrie 索引的创建):利用表 2 所示的数据集创建的基于数字签名的压缩树 cTrie 如图 3(b)所示.

2.3 基于序列的索引(tTrie)

若直接将实验条件序列放入前缀树中,遍历过程则会比较耗时.为了减少根据约束条件等关键词的定位时间,本文直接枚举每个 OPSM 的实验条件部分,将枚举的片段作为前缀树中的中间节点,而 id 号放入枚举序列片段最后一个序列所在的节点中.虽然枚举序列可以加速定位,但是枚举序列的个数随着元素数目的增加成指数级增长.通过观察得知,经常用到的约束中关键词的长度不超过 4,所以只枚举不超过 4 个元素的序列.

引理 **2**. 如果将长度为 *m* 的序列枚举成长度不大于 *l*(*l*<*m*)的子序列,那么其空间复杂度为 $O(m^l)$.

证明:对于每一序列,其长度不大于 *l* 的序列片段的个数为 $A_m^1+A_m^2+\ldots+A_m^l=\sum_{i=1}^{l}A_m^i$, 那么空间复杂度为

$$O\left(\sum_{i=1}^{l}A_m^i\right)=O(lm^l)=O(m^l).$$

实验条件 tTrie 索引(tTrie)的创建过程(算法 3)如下.对于每一个枚举出来的特征 *f*(第 1 行),首先将当前节点指向根节点 *tRoot*,并取出该特征中的实验条件(第 2 行);接着,对于 *f* 中的每个实验条件(第 3 行),如果当前节点的孩子中没有等于该值的节点,就创建一个具有该值的孩子节点,并将当前节点指向该节点(第 4 行、第 5 行),否则只需指向该孩子节点即可(第 5 行);直到 *f* 的最后一个实验条件时,取出 *f* 中的 OPSM 的 ID 号集合(第 6 行);

然后,对于 *f* 中的每个 id 号(第 7 行),如果 *opsmIdTable* 中存在该 id,就将当前节点加入该 id 键所在的值中(第 8 行),否则将该 id 作为键而当前节点作为值,放入 *opsmIdTable* 中(第 9 行).

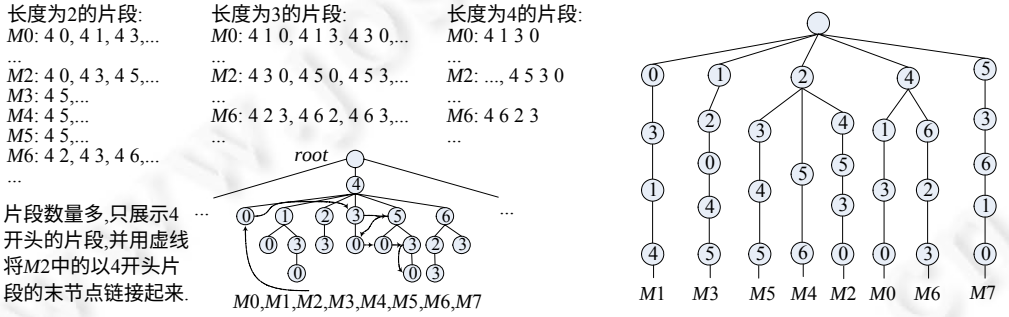
算法 **3**. 实验条件 Trie(tTrie).

输入:枚举序列 *F*.

输出:实验条件 tire tTrie.

1. **while** *F* 中存在下一条数据 *f* **do**
2. *curNode*←*tRoot*; *cols*←*f.cols*;
3. **for** *i*←0 to *cols.length*−1 **do**
4. **if** *curNode* 中不存在孩子 *cols*[*i*] **then** 将 *cols*[*i*]加入 *curNode* 的孩子节点中;
5. *curNode*←*curNode* 中的孩子节点 *cols*[*i*];
6. 将 *curNode* 设为末节点;将 *f* 中该 OPSM 的 ID 存入 ID;
7. **for** *i*←0 to *ID.size*−1 **do**
8. **if** *opsmIdTable* 中存在 *ID*[*i*] **then** 将 *curNode* 添加到 *opsmIdTable* 中 *ID*[*i*]所在单元;
9. **else** 将(*ID*[*i*],*curNode*)加入 *opsmIdTable*;

例 8(tTrie 索引的创建):利用表 2 中的数据集,枚举的长度不大于 4 的序列见图 4(a).由于数量大,只展示了以 4 开头的序列片段.基于枚举序列和原始序列的 tTrie 索引如图 4(a)和图 4(b)所示,后者用于后文自底向上的遍历方法中.



(a) 索引枚举序列的实验条件 Trie(tTrie) (b) 索引源数据的实验条件 Trie

Fig.4 tTrie with different datasets

图 4 不同数据集下的 tTrie

2.4 代价分析

代价分析:在创建索引过程中,主要关注的是预处理与索引的创建时间:

$$T_{index}=T_{sig}+T_{sig\_Trie}+T_{enum}+T_{seq\_tree} \quad (1)$$

其中,T_{sig} 是数字签名的时间代价,T_{sig_Trie} 是数字签名 Trie 的生成时间,T_{enum} 是实验条件序列的枚举时间,T_{seq_tree} 是索引枚举序列的时间.显然,提高公式(1)的性能需要依靠减少数字签名的生成时间和降低序列的枚举时间.

3 查询方法

提出自顶向下和自底向上两类查询方法,前者适用于给定关键词来定位候选集的情形,后者适用于给定候选集后进而缩小候选集的情况.二者结合使用能够提高查询效率,同时,两类索引的前后顺序也影响搜索速度.以上两类方法经过组合得到以下 4 种方法.

(1) 基于 sTrie 的自顶向下与 tTrie 的自底向上相结合的查询方法(即 GT);

(2) 基于 cTrie 的自顶向下与 tTrie 的自底向上相结合的查询方法(即 GcT);

(3) 基于 tTrie 的自顶向下与 sTrie 的自底向上相结合的查询方法(TG);

(4) 基于 tTrie 的自顶向下与 cTrie 的自底向上相结合的查询方法(即 TGc).

3.1 自顶向下的查询

自顶向下的查询包括基于基因名数字签名索引(即 sTrie 和 cTrie)和实验条件枚举枚举序列索引(即 tTrie)这两类查询方法.

3.1.1 基于 sTrie 的自顶向下 OPSM 约束查询

基于 sTrie 的自顶向下 OPSM 约束查询过程(算法 4)如下:算法的输入为 sTrie 索引的某个节点,由约束转化而来的 sig_c 以及 sig_c 的某一个下标.如果当前所遍历的节点未到叶子节点且 sig_c 的当前位点为 1,那么继续遍历该节点的右子树(第 1 行);如果当前所遍历的节点未到叶子节点且 sig_c 的当前位点为 0,那么继续遍历该节点的左右子树(第 2 行、第 3 行);如果当前所遍历的节点为叶子节点,则将该节点中的 OPSM 的 ID 号放入候选集 R 中(第 4 行).

算法 **4**. 基于 sTrie 的自顶向下 OPSM 约束查询(sTrie-TD).

输入:sTrie 的某个节点,由约束转化的数字签名 sig_c,sig_c 的位标签.

输出:OPSM ID 列表 R.

1. **if** *node.level*<*sig.length* & sig_c[*sig_idx*]=1 **then** *sTrie-TQ*(*node.right*,sig_c,*sig_idx*+1);
2. **else if** *node.level*<*sig.length* & sig_c[*sig_idx*]=0 **then**
3. *sTrie-TQ*(*node.left*,sig_c,*sig_idx*+1);*sTrie-TQ*(*node.right*,sig_c,*sig_idx*+1);
4. **else if** *node.level*=*sig.length*−1 **then** 将节点 *node* 中保存的 ID 加入 R;

例 9(基于 sTrie 索引的自顶向下查询):给定关键词 g_4,g_5,g_7,其转化成的 sig_c 为 1101,那么第 1 次迭代时的算法输入就是根节点、1101(sig_c)以及 0(sig_c 的第 1 位).由于当前节点是根节点,取出其左右孩子(0 和 1).因为 sig_c 的第 1 位为 1,所以将当前节点指向右孩子.然后,类似地经过 3 次迭代后,得到 M_2 和 M_3 所在分支符合条件.

定义 **7**(投入(**drop**)). 给定两个集合 c 与 g 以及相应的数字签名 sig_c 与 sig_g,其上的预测 θ,即 $sig_c\theta sig_g$,称为一个投入 drop.如果 $sig_c\theta sig_g$ 与 $c\theta g$ 同时成立,则该预测为正确的投入(right drop).如果 $sig_c\theta sig_g$ 成立但 $c\theta g$ 不成立,则该预测为错误的投入(false drop).

引理 **3**. 给定两个集合 c 与 g 以及相应的数字签名 sig_c 与 sig_g,错误投入的概率为 $(1-e^{-k/b})^k$,其中,b 为数字签名的位数,k 为集合的基数(本文中 c 和 g 的基数相同).

证明:具体方法类似于文献[39],此处从略. □

3.1.2 基于 cTrie 索引的自顶向下 OPSM 约束查询

基于 cTrie 索引的自顶向下 OPSM 约束查询过程(算法 5)如下:采用迭代的方式处理,其输入为 cTrie 索引的某个节点和由约束转化而来的 sig_c.若迭代中 *sig* 为空,则返回该节点中存储的 OPSM ID 号(第 6 行);否则(第 1 行),若当前节点有孩子节点,则返回这些孩子节点(第 2 行).对于每一个孩子节点,获取其基因名序列 sig_{old}、节点地址以及与 sig_{old} 同长度的 sig_c,并将 *or* 初始化,用来存储 sig_{old} 与 sig_c 的比对结果(第 3 行).对于其中的每一位,若前者不小于后者,则存储前者;否则,存储后者(第 4 行).接下来比对 *or* 与 sig_{old}:若相同,则进入新一轮的迭代(第 5 行).

算法 **5**. 基于 cTrie 索引的自顶向下 OPSM 约束查询(cTrie-TD).

输入:cTrie 的某个节点,由约束转化的数字签名 sig_c.

输出:OPSM ID 列表 R.

1. **if** *sig* 不为空 **then**
2. **if** *node* 中存在孩子节点 **then** *all*←*node* 的所有孩子节点;
3. **for** *child*: *all* **do** *key*←*node* 中的基因名;*val*←*node* 节点;*str*←*sig* 从 0 到 *key.length* 的序列;*or*←"Φ";
4. **for** *i*←0 to *key.length*−1 **do if** *key*[*i*]>=*str*[*i*] **then** *or*←*or*+*key*[*i*]; **else** *or*←*or*+*str*[*i*];
5. **if** *or* 的值与 *key* 相同 **then** *cTrie-TQ*(*val*,*sig.substring*(*key.length*,*sig.length*));

6. **else** 将节点 *node* 中保存的 ID 加入 *R*;

例 10(基于 cTrie 索引的自顶向下查询):查询的输入同例 9(根节点、1101(sig_c)以及 0(sig_c 的第 1 位)).由于当前节点是根节点,且有两个孩子节点(01 和 11).因为 sig_c 的前两位为 11,所以经过两次比较之后,将当前节点指向 11 节点.接着进入第 2 次迭代,经过类似的比较之后,最后得到 M_2 和 M_3 所在的分支符合条件.

引理 **4**. 基于 sTrie 与 cTrie 的自顶向下的搜索时间复杂度都为 $O(\zeta b)$,其中,ζ为平均搜索分支数.

3.1.3 基于 tTrie 索引的自顶向下 OPSM 约束查询

基于 tTrie 索引的自顶向下 OPSM 约束查询过程(算法 6)如下:算法的输入为必须连接约束,用到的数据为枚举的特征数据.首先将当前节点指向根节点,必须连接的游标初始化为 0,并将分支是否遍历的标志位设为真(第 1 行).对于每个分支,如果标志位为真,则判断该节点是否含有第 *idx* 个必须约束的孩子:若有且游标 *idx* 没有到达最后一位,则将当前节点指向该孩子节点(第 3 行、第 4 行);如果 *idx* 指向最后一位,则将当前节点中存储的 OPSM 的 ID 号放入结果集合 *R* 中,并将标志位设为假(第 5 行).

算法 **6**. 基于 tTrie 索引的自顶向下 OPSM 约束查询(tTrie-TD).

输入:必须连接 *tMust*.

输出:OPSM ID 列表 *R*.

1. *curNode*←tTrie 的根节点,*idx*←0,*flag*←true;
2. **while** *flag* 为真 **do**
3. **if** *curNode* 的孩子中存在 *tMust.get*(*idx*)且 *idx* 不等于|*tMust*|−1 **then**
4. *curNode*←*urNode* 的孩子节点 *tMust.get*(*idx*);
5. *idx*++; **if** *idx* 等于|*tMust*|−1 **then** 将节点 *curNode* 中保存的 ID 加入 *R*;*flag*←false;

例 11(基于 tTrie 索引的自顶向下查询):给定关键词 4,5,3(必须连接),首先将当前节点指向根节点,接着从其孩子节点中找到 4 节点,然后找到 4 的孩子节点 5,接下来找到 5 的孩子节点 3.此时关键词已用完,就从当前节点中取出 ID 号(M_2)作为候选集.

引理 **5**. 基于 tTrie 的自顶向下的搜索时间复杂度为 $O(\varepsilon)$,其中,ε为平均搜索节点数.

3.2 自底向上的查询

包括基于基因名数字签名索引(即 sTrie 和 cTrie)和实验条件枚举枚举序列索引(即 tTrie)的两类查询方法.

3.2.1 基于 sTrie 的自底向上 OPSM 约束查询

基于 sTrie 的自底向上 OPSM 约束查询过程(算法 7)如下:算法的输入为由约束转化而来的 sig_c 和候选集 ID.对于候选集中的每个 id,首先初始化支持它的分支的个数,并将数字签名中为 1 的位置数字放入链表 *rows* 中(第 1 行).如果 id 在 *opsmIdTable* 中不为空,则取出所有分支末节点(不一定是叶子节点),并以自底向上的方式检验该分支是否符合条件(第 2 行).对于取出来的每个分支节点,首先进行初始化工作(第 3 行);从该分支节点自底向上遍历(第 4 行),当节点标号 *idx_brc* 等于关键词 *rows* 中的第 *idx_key* 个元素(第 5 行)时,若检测到当前节点中的元素为 0,则将标志位 *flag* 置为 flase,且终止对该分支的遍历,若 *rows* 没有检测完(*idx_key* 大于 0),将 *idx_key* 减少 1(第 6 行).如果节点标号 *idx_brc* 等于关键词 *rows* 中的第 *idx_key* 个元素(第 5 行),则将 *idx_brc* 减少 1,并将当前节点指向父节点(第 7 行).在遍历完每一分支后,若 *flag* 为真,则将支持 id 的分支数目 *idBrcCnt* 加 1(第 8 行).在遍历完每一 id 的所有分支后,若其支持数量 *idBrcCnt* 为 0,则从 ID 中删除(第 9 行).最后,将 ID 加入 *R*(第 10 行).

算法 **7**. 基于 sTrie 的自底向上 OPSM 约束查询(sTrie-BU).

输入:由约束转化的数字签名 sig_c,OPSM 候选集 ID.

输出:OPSM ID 列表 *R*.

1. **for** id: ID **do for** *i*←0 to $sig_c.length$−1 **do if** $sig_c[i]$为 1 **then** 将 *i* 加入 *rows*;
2. **if** *opsmIdTable* 中存在 id **then** *nodes*←*opsmIdTable* 表 id 单元中的节点;
3. **for** *node*: *nodes* **do** *curNodes*←*node*; *idx_brc*←$sig_c.length$−1; *idx_key*←*rows.length*−1; *flag*←true;
4. **while** *curNode* 不为 sTrie 的根节点 **do**

5. **if** *idx_brc* 等于 *rows* 中第 *idx_key* 个元素 **then**
6. **if** *curNode.getgName*=0 **then** *flag*←false; *break*; **if** *idx_key*>0 **then** *idx_key*−−;
7. *idx_brc*−−; *curNode*←*curNode* 的父节点;
8. **if** *flag* **then** *IdBrcCnt*++;
9. **if** *idBrcCnt*=0 **then** 从 ID 中去除 id;
10. 将 ID 中的所有元素加入 *R*;

例 12(基于 sTrie 索引的自底向上查询):给定与例 9 相同的关键词 g_4,g_5,g_7(sig_c 为 1101)以及例 11 获取到的候选集 M_2.首先,将 sig_c 中为 1 的位数记录在 *rows* 链表中,则 *rows* 存储的为 0,1,3;接着,从 *opsmIdTable* 表头中取出 M_2 所在分支的叶节点;然后,以自底向上的顺序检测到该分支 3,1,0 所在位置为 1,所以 M_2 符合条件.

3.2.2 基于 cTrie 的自底向上 OPSM 约束查询

基于 cTrie 索引的自底向上 OPSM 约束查询(算法 8)与算法 7 大致相同,不同之处如下:第 3 行中加入了判断 *rows* 中关键词是否用完的标志位 *flag_key*,并在第 4 行的 while 中判断该标志位;第 5 行~第 7 行替换为首先判断当前节点中所存储位点的左分支是否小于 *idx_key*,若小于等于 *idx_key*,则取出当前节点中 *idx_key* 所对应的位点;接着,如果该位点为 0,则将标志位 *flag* 置为 flase,且终止对该分支的遍历;若 *rows* 中的关键词没有检测完(*idx_key* 大于 0),则将 *idx_key* 减 1(第 6 行);若 *rows* 中的关键词已经检测完(*idx_key* 等于 0),则将标志位 *flag_key* 置为 flase,且终止对该分支的遍历.如果节点标号 *idx_brc_left* 小于关键词 *rows* 中的第 *idx_key* 个元素(第 5 行),则将 *idx_brc_left* 赋值为 *idx_brc_left* 减少该节点存储的数字签名位数后的值,并将当前节点指向父节点(第 7 行).

算法 **8**. 基于 cTrie 的自底向上 OPSM 约束查询(cTrie-BU).

输入:由约束转化的数字签名 sig_c,OPSM 候选集 ID.

输出:OPSM ID 列表 *R*.

1~4. 同算法 7.

5. **if** *idx_brc_left*≤*row* 的第 *idx_key* 个元素 **then** *bit*←*curNode* 中基因名数字签名的第 *idx_key* 位;
6. **if** *bit*=0 **then** *flag*←false; *break*; **if** *idx_key*>0 **then** *idx_key*−−; **if** *idx_key*=0 **then** *flag_key*←false; *break*;
7. **else** *idx_brc_left*←*idx_brc_left*$\tilde{}$*curNode* 基因名的长度;*curNode*←*curNode* 的父节点;

8~10. 同算法 7.

例 13(基于 cTrie 索引的自底向上查询):给定与例 9 相同的关键词 g_4,g_5,g_7(sig_c 为 1101)以及例 11 获取到的候选集 M_2.搜索与比对方式与例 12 类似,不同之处为:本方法是每次获取到的数字签名位数不确定,需要在比对过程中做额外的对齐下标的处理.最后得到的结果同样为 M_2.

引理 **6**. 基于 sTrie 与 cTrie 的自底向上的搜索时间复杂度都为 $O(\zeta b)$,其中,ζ为平均搜索分支数.

3.2.3 基于 tTrie 的自底向上 OPSM 约束查询

基于 tTrie 索引的自底向上 OPSM 约束查询过程(算法 9)如下:算法的输入为 OPSM 的集合 ID、必须连接和不能连接约束条件.用到的数据为完整的实验条件序列,而不是枚举的实验条件序列.原因是可以减少候选集中元素的个数,减少后续的验证工作.对于每个由 tTrie 索引得到的 OPSM(第 1 行),如果 *opsmIdTable* 表头中存在该 id,就返回包含该 id 的节点(第 2 行).对于上述获取到的每个节点,首先获取每个节点为当前节点(第 3 行),接着判断该节点中的列标签是否在不能连接约束中:如果在,则终止该分支的遍历;否则,将当前列标签放入最长公共子序列 *lcs* 中.之后,指向当前节点的父节点,直到根节点(第 4 行、第 5 行).如果获取到的 *lcs* 与必须连接约束的反序列的 *lcs* 的长度大于 1,则将该 id 的 *lcs* 计数加 1(第 6 行).对于 *lcs* 计数为 0 的 id 号,从 ID 中删除,并将 ID 的长度与下标 *i* 的数值减 1(第 7 行).最后,将 ID 赋值给 OPSM 列表 *R*,并作为结果返回(第 8 行).

算法 **9**. 基于 tTrie 的自底向上 OPSM 约束查询(tTrie-BU).

输入:OPSM 候选集 ID,必须连接 *tMust*,不能连接 *tNot*.

输出:OPSM ID 列表 *R*.

1. **for** i←0, len←$ID.length$ to len−1 **do**
2. **if** *opsmIdTable* 中存在 $ID[i]$ **then** *nodes*←*opsmIdTable* 获取 $ID[i]$中的所有节点;
3. **for** j←0 to *nodes.size*−1 **do** *curNode*←*nodes* 的第 j 个元素;
4. **while** *curNode* 不为根节点 **do if** *tNot* 包含 *curNode.col* **then** *break*; **else** 将 *curNode.col* 加入 *lcs*;
5. *curNode*←*curNode* 的父节点;
6. **if** *lcs* 与 *tMust* 的反序列的 LCS 长度>1 **then** *count*++;
7. **if** *count*=0 **then** 从 ID 中去除 $ID[i]$; −−*len*; −−i;
8. R←ID;

例 14(基于 tTrie 索引的自底向上查询):给定与例 11 相同的关键词 4,5,3(必须连接),1,6(不能连接)以及例 9(例 10)获取到的候选集 M_2,M_3.首先找到 M_2 所在分支的节点,接着以自底向上的方式比对关键词,依次遍历节点 0,3,5,4,2,其中包括关键词 4,5,3,且没有关键词 1,6,所以该分支符合条件.接下来搜索 M_2 所在的分支.由于在 5 之前没检测到 3(自底向上方式,所以关键词顺序相反),所以提前终止该分支的遍历.最后只有 M_2 符合条件.

引理 **7**. 基于 tTrie 的自底向上搜索的时间复杂度为 $O(\omega m)$,其中,ω为候选分支数,m 为平均分支长度.

3.3 性能优化

规则 **1**(基于关键词顺序的剪枝). 在遍历 sTrie 与 cTrie 的过程中,如果某一位关键词位置为 0,那么此分支中剩下的元素就不必遍历.同样,在遍历 tTrie 的过程中,按照输入的列关键词顺序和相应分支中的元素比对.如果此分支中的元素顺序和关键词顺序不一致,那么此分支中剩下的元素就不必遍历.

搜索方法给出相关结果之后,还不能保证结果相关性较高的排在前面.为此,给出了对结果排名用到的标准与定义.

定义 **8**(排名). 对于符合所有约束的结果,按照每个 M_i 中行列数目和数目约束的接近程度来定义其排名,即越接近数目约束的排名越靠前.其计算方法如公式(2)所示.

$$R_{M_i}=\left\| M_i^r \right|-\left| C_{cnt}^r \right\|^2+\left\| M_i^t \right|-\left| C_{cnt}^t \right\|^2 \tag{2}$$

例 15:利用表 2 中的数据和例 14 中的查询结果为 M_2.如果还有另外一个结果 M_8(g_4 g_5 g_7:2 4 0 3 6 5),那么根据定义 8 有,M_0 排在 M_6 前边,因为 $R_{M_0}=|3-3|^2+|5-5|^2=0, R_{M_8}=|3-3|^2+|6-5|^2=1, R_{M_0}<R_{M_8}$, 则 M_0 排名高.

3.4 代价分析

代价分析:在 OPSM 约束查询处理中,主要关注的是查询响应时间:

$$T_{\text{response}}=T_{\text{search}}+|C_q|\times T_{\text{cst\_test}} \tag{3}$$

其中,T_{search} 是搜索部分的耗时,$T_{\text{cst\_test}}$ 是约束型检测所需要的平均时间.在验证部分,剔除候选集 C_q 中的所有假阳性 M_i 需要$|C_q|\times T_{\text{cst_test}}$ 时间.通常,验证时间占据响应时间公式(3)的主要部分,主要是因为$|C_q|$的数目比较庞大.另外,对于不同的约束查询,$T_{\text{cst\_test}}$ 的变化并不明显.因此,减小约束查询响应时间的关键工作就是最小化候选集 C_q 的大小.如果 OPSM 数据集太大而不能存储于内存,T_{search} 是将占据查询相应时间的重大部分.

4 实验评估

本节主要评估以下 6 种方法的有效性与可扩展性:(1) 暴力搜索法(BF);(2) 基于 sTrie 的自顶向下与 tTrie 的自底向上相结合的查询方法(GT);(3) 基于 cTrie 的自顶向下与 tTrie 的自底向上相结合的查询方法(GcT);(4) 基于 tTrie 的自顶向下与 sTrie 的自底向上相结合的查询方法(TG);(5) 基于 tTrie 的自顶向下与 cTrie 的自底向上相结合的查询方法(TGc);(6) KiWi[40].实验中用到了两种数据:真实数据和生成数据.大多数实验是在真实数据上进行的,因为它是真实需求的来源.实验用到的机器的是 1.87GHz 频率,16GB 内存,且运行着 Ubuntu 12.04 的浪潮服务器(在分布式并行情况下,可利用的最大节点数为 9).3 种方法用 Java 语言编写,由 Eclipse 4.3 编译运行.

数据生成方法:先从网站[41]上下载如表 3 所示的 6 个数据集;接着,利用快速排序法对每一行的表达值排序;然后,将每一个表达值替换成对应的列标签,使其变成序列数据[42];最后,利用文献[38,43]中的列模糊查询方法(关键词为相应数据集中所有的列标签,列和行阈值都为 2)生成表 4 中描述的 6 个数据集.

枚举序列(特征)见表 4.

Table 3 Details of gene expression datasets

表 3 基因表达数据集的详细情况

文件名	行数	OPSM	文件名	行数	OPSM
adenoma	12 488	D_1	a549	22 283	D_2
5q_GCT_file	22 278	D_3	krasla	12 422	D_4
bostonlungstatus	12 625	D_5	bostonlungsubclasses	12 625	D_6

Table 4 Details of OPSM datasets

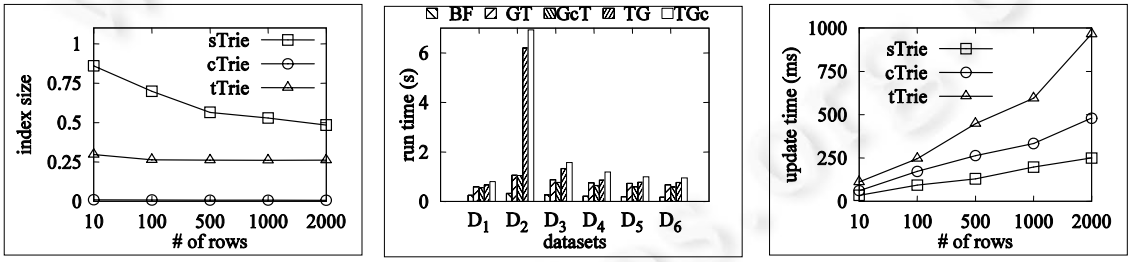
表 4 OPSM 数据集的详细情况

数据集	行数	特征数	数据集	行数	特征数
D_1	849	673 657	D_2	6 327	2 043 972
D_3	4 028	1 571 613	D_4	3 034	879 653
D_5	2 491	437 080	D_6	2 019	244 652

4.1 单机性能

4.1.1 索引性能

首先测试 sTrie,cTrie 和 tTrie 这 3 种方法在索引 10 行,100 行,500 行,1 000 行和 2 000 行的 D_2 数据集的情况下的索引大小,其中,sTrie 和 cTrie 用到的数据为表 3 中的 D_2 数据集,tTrie 用到的是表 4 中的 D_2 数据集.索引大小定义为索引的数据量与原始数据量的比值.3 种索引的索引大小如图 5(a)所示.当索引的 D_2 中的数据量由 10 行增加到 2 000 行的过程中,sTrie 索引的数据量占原始数据的比重大小由 86%下降到 48%,cTrie 索引的数据量占原始数据的比重大小由 1%下降到 0.67%,tTrie 索引的数据量占原始数据的比重大小由 30%下降到 26%.sTrie 与 cTrie 利用的是同样的数据而索引压缩比却有很大差别,这是因为 cTrie 对 sTrie 中的单分支节点进行了大幅度的压缩.tTrie 索引较小的原因是 tTrie 利用的数据为实验条件枚举序列,其最大长度为 4,所以会有较多的可重复利用节点,因而索引较小.



(a) 索引大小 (b) io/索引时间 (c) 索引更新时间

Fig.5 Performance evaluation of index methods on a single machine

图 5 索引方法在单机上的性能评估

接着评估 5 种方法在表 3 和表 4 所示的数据集上的性能.5 种方法在 6 种数据集上所消耗的 I/O 或者索引创建时间如图 5(b)所示,BF 的 I/O 在 0.2s 左右,虽然不大,但是由于其没有索引,致使在每次执行查询时这一耗时都是必须的.GT 创建索引耗时在 0.5s~1s 之间,其压缩索引 GcT 索引耗时也在 0.5s~1s 之间.TG 创建索引耗时在 0.6s~6s 之间,其压缩索引 TcG 索引耗时也在 0.8s~6s 之间.TG 和 TcG 耗时大于 GT 和 GcT 的原因是:前者索引的是实验条件的枚举序列(见表 4),数据量相对较大;后者索引的是原始的实验条件序列(见表 3),数据量相对较小.TG 与 TcG 索引中用到的特征数据分别来自 6 个 OPSM 数据集合中的实验条件部分,且实验条件的数目由少

到多,数据量 $D_2>D_3>D_4>D_5>D_6>D_1$.

最后测试 sTrie,cTrie 和 tTrie 这 3 种索引在插入 10 行,100 行,500 行,1 000 行和 2 000 行的 D_2 数据集的情况下的性能.索引更新时间如图 5(c)所示,sTrie 由 35ms 平缓增加到 250ms,cTrie 由 62ms 缓慢增加到 480ms,tTrie 由 112ms 快速增加到 966ms.由于 cTrie 是 sTrie 的压缩索引,其在节点的压缩方面要比后者多付出一些代价,所以 cTire 耗时多于 sTrie.因为 tTrie 索引的是特征数据,而特征数据的 key 较短但 value 较多,每次都要检测分支的末节点中是否已经含有 value 中的每个值,所以 tTrie 索引更新耗时相对较多.

4.1.2 查询性能

首先测试 5 种方法在枚举表 4 所示数据时的性能,如图 6(a)所示.D_2 数据上的枚举时间最长,因为其数据量远远多于其他数据.D_1 数据上的枚举时间最短,因为其数据量远远少于其他数据.其他数据上的枚举时间基本相同,这是因为虽然数据量的顺序为 $D_3>D_4>D_5>D_6$,但是列数顺序则恰好相反.5 种方法在 6 种数据集上进行同一种查询(3 个 must-link、3 个 cannot-link、2 个 interval 和 2 个 count 约束)所消耗的时间如图 6(b)所示,BF 的查询时间在 158ms~206ms 之间,GT 的查询时间在 36ms~115ms 之间,GcT 的查询时间略高于 GT,在 74ms~143ms 之间,TG 与 TcG 的查询耗时基本相同,分别在 1ms~17ms 与 0ms~14ms 之间.TG 与 TcG 的查询性能要优于另外 3 种方法,原因是 TG 与 TcG 的定位速度与候选集数目都要小于另外 3 种方法.

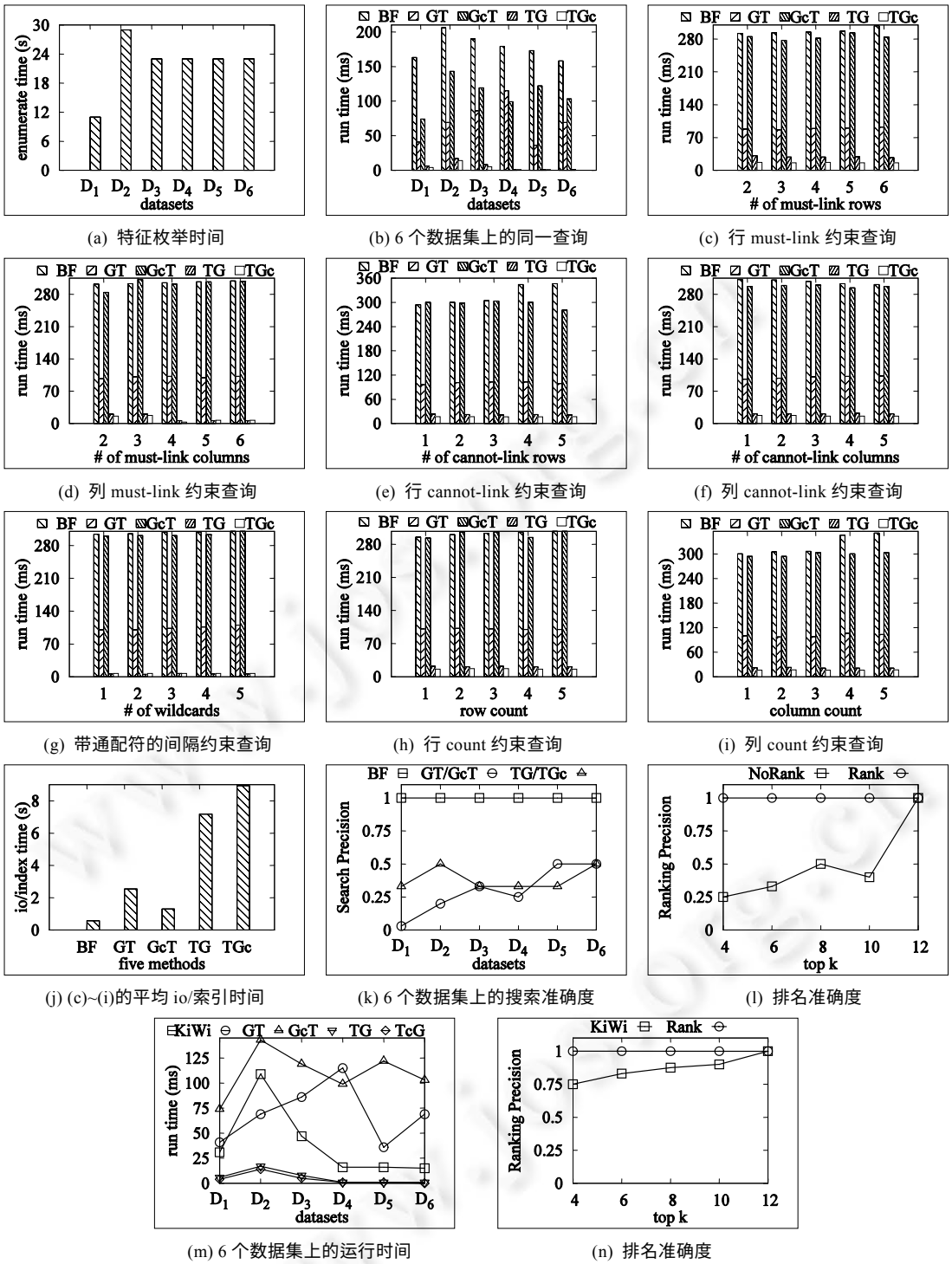
接着进行 must-link 查询的性能评估,其性能如图 6(c)和图 6(d)所示.当行的 must-link 关键词数目由 2 增加到 6 的过程中,BF 的查询时间由 292ms 增加到 308ms,GT 的查询时间由 86 增加到 92ms,GcT 的查询时间在 277ms~293ms 之间徘徊.GcT 是 GT 的压缩索引,但 GcT 的查询耗时却远大于 GT,基本与 BF 相当.原因是 GcT 失去了 GT 的二分查找特性,基本要将所有分支都遍历一下.TG 的查询时间在 27ms~31ms 之间,TcG 的查询耗时在 16ms~17ms 之间.TG 与 TcG 的性能远远优于 BF,GT 与 GcT.同样地,当列的 must-link 关键词数目由 2 增加到 6 的过程中,BF 的查询时间由 302ms 增加到 309ms,GT 的查询时间由 97ms 增加到 102ms,GcT 的查询时间在 284ms~312ms 之间徘徊.GT 性能优于 GcT 的原因同上.TG 的查询时间在 6ms~20ms 之间,TcG 的查询耗时在 3ms~17ms 之间.同样地,TG 与 TcG 的性能远远优于 BF,GT 与 GcT.

下面进行 cannot-link 查询的性能评估,其性能如图 6(e)和图 6(f)所示.在行的 cannot-link 关键词数目由 1 增加到 5 的过程中,BF 的查询时间由 292 增加到 308ms,GT 的查询耗时在 96ms~103ms 之间,GcT 的查询时间在 281ms~303ms 之间.TG 的查询时间在 21ms~23ms 之间,TcG 的查询耗时在 16ms~17ms 之间.TG 与 TcG 的性能远远优于 BF,GT 与 GcT.在列的 cannot-link 关键词数目由 1 增加到 5 的过程中,性能和行关键词上的测试相同,且 TG 与 TcG 的性能远远优于 BF,GT 与 GcT.

接下来评估 interval 约束查询中通配符的使用对查询性能的影响,其性能如图 6(g)所示.在 interval 约束中,通配符的数目由 1 增加到 5 的过程中,BF 的查询时间由 304ms 增加到 310ms,其原因是通配符越多,需要遍历的数据就越多,进而耗时也就越多;GT 的查询耗时由 100 增加至 110ms,GcT 的查询时间由 300 到 310ms.TG 的查询时间在 5ms~7ms 之间,TcG 的查询耗时维持在 7ms.同样地,TG 与 TcG 的性能远远优于 BF,GT 与 GcT.

接着评估 count 约束查询中行列 count 的多少对查询性能的影响,其性能如图 6(h)和图 6(i)所示.在行 count 由 1 增加到 5 的过程中,BF 的查询时间由 295ms 增加到 307ms,GT 的查询耗时在 100ms~102ms,GcT 的查询时间在 293ms~307ms 之间,TG 的查询时间在 21ms~22ms 之间,TcG 的查询耗时维持在 16ms~17ms.

图 6(j)展示了上述 4 种约束查询之前扫描数据或者对数据进行索引的耗时.BF 扫描一遍数据所需要的平均时间约为 0.6s,但每次查询都要扫描数据.假如数据量非常庞大,那么这部分耗时会急剧增加,所以此方法不可取.GT 方法在 0/1 数字签名与实验条件枚举序列数据上建立索引所消耗的平均时间约为 2.54s,实验条件序列枚举代价是对最大长度为 4 的特征进行索引所耗费的时间.如果全部长度的特征都要索引,则所需耗时会更长.GcT 索引没有单节点分支,即每个节点存储较多数据使得开辟新节点个数更少,所以耗时约为 GT 的一半(1.3s).TG 和 TcG 索引的数据同为原始实验条件序列与 0/1 数字签名序列,二者耗时基本相同,分别为 7.2s 与 8.9s.



(a) 特征枚举时间 (b) 6 个数据集上的同一查询 (c) 行 must-link 约束查询

(d) 列 must-link 约束查询 (e) 行 cannot-link 约束查询 (f) 列 cannot-link 约束查询

(g) 带通配符的间隔约束查询 (h) 行 count 约束查询 (i) 列 count 约束查询

(j) (c)~(i)的平均 io/索引时间 (k) 6 个数据集上的搜索准确度 (l) 排名准确度

(m) 6 个数据集上的运行时间 (n) 排名准确度

Fig.6 Performance evaluation of 5 methods on a single machine

图 6 5 种方法在单机上的性能评估

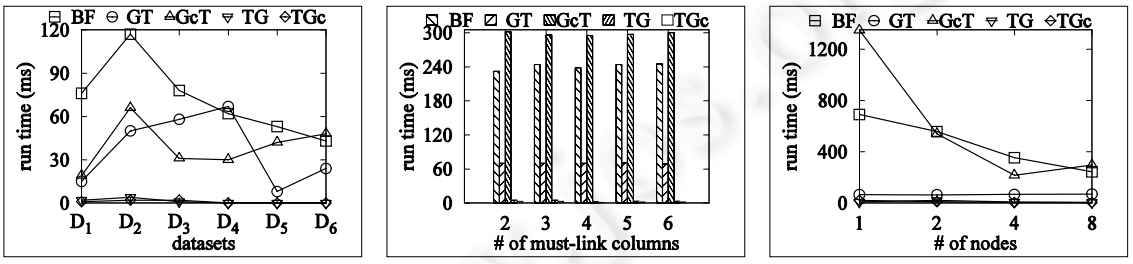
然后,评估所提出方法的准确性.图 6(k)展示了 BF,GT,GcT,TG 和 TcG 这 5 种方法在 6 种不同数据集上执行

同一查询所返回候选集的准确度.GT 和 GcT 方法在 6 种数据集上的准确度由 0.03 提高到 0.5,TG 和 TcG 方法则维持在 0.33 和 0.5 之间,而 BF 方法则一直为 1.这是因为 BF 方法是对每个 OPSM 利用 4 种约束进行搜索,所以搜索到的结果都是正确的,而另外 4 种方法是没有完全利用 4 种约束做搜索,剩余的约束在后续验证中才用到.图 6(l)给出了 ES 和 CQ 两种方法在有无利用排名技术的情况下所返回最终结果排名的准确度.由于前者(没有利用排名技术)返回的结果是随机排列的,而后者(利用排名技术)是将最重要的首先返回,所以后者对最终结果的排名更为准确.

最后,比较文献[20,29]中的 KiWi 方法(代码下载地址见文献[40])与本文所提出所有方法在时间效率和准确度方面的性能.图 6(m)展示了 KiWi 和 GT 等 5 种方法在 6 种数据集上执行同一查询的运行时间.KiWi 在 D_2 数据集上的运行时间最长(109ms),在 D_3 和 D_1 数据集上的运行时间稍短(分别为 47ms 和 31ms),在 D_4,D_5 和 D_6 数据集上的运行时间最短(基本为 16ms).同样地,TG 和 TcG 在 D_2 数据集上的运行时间最长(17ms 和 14ms),在 D_3 和 D_1 数据集上的运行时间稍短(7ms),在 D_4,D_5 和 D_6 数据集上的运行时间最短(1ms).GcT 的趋势稍有不同,在 D_2 数据集上的运行时间最长(143ms),在 D_3 和 D_5 数据集上的运行时间稍短(120ms),在 D_4 和 D_5 数据集上的运行时间更短些(100ms),在 D_1 数据集上的运行时间最短(74ms).GT 的趋势则有较大的不同,在 D_4 数据集上的运行时间最长(115ms),在 D_3 数据集上的运行时间稍短(86ms),在 D_2 和 D_6 数据集上的运行时间更短些(69ms),在 D_1 和 D_5 数据集上的运行时间最短(40ms).图 6(n)给出了 KiWi 和 Rank 两种方法在前 k 个结果中的准确度.在 top-k 中的 k 由 4 增加到 12 的过程中,KiWi 所返回结果的准确度由 0.75 逐步提高到 1,且接近于 Rank 的准确度 1.因为前者首先返回列长的结果再返回行多的结果,而后者要综合考虑行和列的数目,先返回行和列数目相对较多的,再返回较少的.

4.2 并行性能

上一节已经验证了 5 种方法在单机上的性能,本节将测试 3 种方法在分布式并行平台下(包括 1 个 master 节点、8 个 slave 节点)的表现.图 7(a)给出了 BF,GT,GcT,TG 和 TcG 这 5 种方法执行同样的约束查询(3 个 must-link、3 个 cannot-link、2 个 interval 和 2 个 count 约束)在 6 种数据集上的运行时间.BF 在 D_2 数据集上的运行时间最长(117ms),原因是 D_2 数据集中的行数是最多的.D_3,D_4 和 D_5 数据集上的查询时间随着行数的减少而减少.D_1 数据集的行数是最少的,但是其上的查询时间却排名前三位,原因是行所拥有的数据量远多于其他数据集.GT 方法基本与 BF 类似,但是在 D_5 和 D_6 数据集上例外.同样地,GcT 方法基本与 BF 类似,但是在 D_3 和 D_4 数据集上例外. TG 和 TcG 两种方法运行速度很快,最长耗时为 4ms,最短耗时为 0ms.此实验证明了 5 种方法在不同数据集上执行查询的可扩展性.TG 和 TcG 方法的查询性能最好,因为索引的结合顺序减少了候选集数目、缩短了查询时间.



(a) 6 个数据集上的同一查询 (b) 列 must-link 约束查询 (c) 不同节点数下的同一查询

Fig.7 Performance evaluation of 5 methods on Hadoop

图 7 5 种方法在 Hadoop 上的性能评估

图 7(b)展示了 BF,GT,GcT,TG 和 TcG 这 5 种方法在 18 748 行数据上执行不同的列 must-link 关键词而其他关键词约束相同的查询所消耗的时间.BF 方法最小运行时间为 232ms,最大运行时间为 245ms,基本符合随着关键词的增加,查询时间在增加的预测.GT 方法受关键词影响较小,基本稳定在 70ms.虽然 GcT 方法只是对 GT 方

法索引做了压缩,但是其失去了 GT 方法的遍历优势,所以非常耗时,甚至超过了 BF 方法.由于 TG 和 TcG 方法具有对候选集的良好过滤能力,所以二者的查询时间基本维持在 3ms 左右.此实验证明了 5 种方法在不同数目的关键词下执行查询的可扩展性.

最后验证 BF,GT,GcT,TG 和 TcG 这 5 种方法在不同集群节点(1,2,4,8 个节点)上执行同一种约束查询的可扩展性.如图 7(c)所示,随着节点数目的增加,BF 方法的查询时间由 690ms 减少到 244ms,性能提高了将近 3 倍;GT 方法的查询时间基本维持在 65ms~70ms 范围内;GcT 方法的查询时间由 1 347ms 减少到 216ms,性能提高了 6 倍以上;TG 方法的查询时间由 21ms 减少到 5ms,性能提高了 4 倍以上;TcG 方法的查询时间由 21ms 减少到 3ms,性能提高了 7 倍.此实验说明,BF,GT,GcT,TG 和 TcG 这 5 种方法随着节点的增加,都具备有良好的可扩展性.

5 相关工作

本节主要回顾和分析与约束型查询有关的文献,主要分 3 类:(1) 基于定量和定性测度的子空间聚类;(2) 基于查询的双聚类;(3) 约束型双聚类.对每一类工作只做简单介绍,更详细的描述可查阅 Sim 等人的综述[44−48].

1) 基于定量和定性测度的子空间聚类

双聚类的概念最初由 Hartigan 等人[2]提出,作为对矩阵中的行与列同时聚类的一种方法,并将其命名为 Direct 聚类.Cheng 等人[3]提出了基因表达数据的双聚类,并引入了元素残差以及子矩阵的均方残差 MSR[3]的概念.他们提出了一种贪婪方法.首先,将整个数据矩阵作为初始化数据;接着,删除元素残差或者均方残差最大元素或者行列,依次递归下去,直到剩余矩阵的 MSR 低于某个阈值;然后,增加部分元素或者行列,保证所得矩阵的 MSR 也低于该阈值.该方法效率较低,因为一次只能挖掘 1 个双聚类.Ben-Dor[1]介绍了一种特殊的双聚类模型 OPSM,并证明了其是 NP 难问题.随后,研究者们提出了基于定量测度[4−14]和定性测度[15−33]的 OPSM 挖掘方法.定量测度包括均方残差 MSR[1]、平均相关值 ACV[10]、平均斯皮尔曼秩相关系数 ASR[10]、平均一致性相关指数 ACSI[10]等.定性测度包括上升、下降和无变化[24,28].

(1) 基于定量测度[4−14]的 OPSM 挖掘方法

基于 Cheng 等人[2]提出的δ-bicluster 模型,Yang 等人[4]为了减少数据缺失值的影响,给出一种δ-cluster 模型.Cho 等人[5]介绍了两种与 MSR 相似的平方残差测度,同时提出了两种有效的基于 k-means 的双聚类算法.Divina 等人[6]给出了一种基于进化计算的双向聚类方法,用来发现尺寸较大、重叠较少且 MSR 小于某阈值的双聚类.Deodhar 等人[7]提出了一种鲁棒的有重叠的双聚类方法 ROCC,能够有效地从大量的含有噪声的数据中挖掘出稠密的、任意位置的有覆盖的聚类.Cho 等人[8]给出了数据转换的方法,以解决现有的平方残差和测度方法只能有效地挖掘出在数值上具有偏移的双聚类,却不能很好地解决在数值上有缩放的双聚类问题.Odibat 等人[9]发现,现有方法并不能有效地挖掘矩阵数据中任意位置有重叠的双聚类,提出了确定性双聚类算法.该算法可以有效地发现正负相关的任意位置上有重叠的双聚类.Ayadi 等人[10]利用平均一致性相关指数 ACSI 来评估相干双聚类,并利用有向无环图组建这些双聚类.Truong 等人[11]提出了一种算法,用来生成若干个覆盖度小于阈值的双聚类,在一定程度上能够发现无冗余的双聚类.Ayadi 等人[12]给出了模因双聚类算法 MBA,以发现生物学意义上的重要的负相关双聚类.Chen 等人[13]利用最小均方错误 MMSE 测度来鉴别所有类型的线性模式(偏移、缩放、偏移与缩放联合模式).Denitto 等人[14]利用 Max-Sum 测度来提升双聚类的质量.

(2) 基于定性测度[15−33]的 OPSM 挖掘方法

Wang 等人[15]提出并设计了基于 pScore 测度和 pCluster 模型的方法,以挖掘具有相似升降趋势的模式.Liu 等人[16]通过寻找在部分维度下表达值排序相同的基因等对象来挖掘 OPSM.Wang 等人[17]给出了一种基于最近邻的新的测度方法来挖掘相似模式.Kriegel 等人[18]提出了一种局部密度阈值的 OPSM 挖掘方法,试图改变现有的基于全局密度阈值的方法并不能适用于每个 OPSM 的现状.Jiang 等人[19]给出了一种质量驱动的 top-k 模式挖掘方法,以提升发现的有重叠的 OPSM 的质量.Gao 等人[20,29]提出了一种 KiWi 框架,利用 k 和 w 两个参数来约束计算资源和搜索空间.Zhang 等人[21]发现,现有的方法都假设基因表达数据是同质的,给出了称为 F-cluster 的模型来挖掘异质数据中的相干模式.Yan 等人[22]为挖掘非线性相关的模式,设计了适用于时序基因表达数据

的联合聚类方法 MI-TSB.Zhang 等人[23]提出了一种近似保序聚类模型 AOPC,以减少数据中噪声的影响.Chui 等人和 Yip 等人[24,30]利用多份数据模型 OPSM-RM 来消除数据噪声的影响.Zhao 等人[25]提出了一种最大化子空间聚类算法,来挖掘具有正相关和负相关的共调控基因聚类.Trapp 等人[26]为挖掘最优 OPSM,给出了一种基于线性规划的挖掘方法.Fang 等人[27]为挖掘放松的 OPSM,提出了包含以行或列为中心的 OPSM-Growth 方法.随后,Fang 等人[28,32]提出了基于桶和概率的方法,挖掘出放松的 OPSM.An 等人[31]利用互信息和核密度进行双聚类.Cho 等人[33]给出了一种基于坏字符规则的 KMP 算法,试图快速匹配保序模式.

2) 基于查询的双聚类[49]

该方法来自生物信息领域[50−53],应用对象是基因表达数据.首先,由用户根据经验来提供功能相关或共表达的种子基因;接着,利用该种子对搜索空间剪枝或双聚类进行指导.为了使现有挖掘方法能利用先验知识并回答指定的问题,Dhollander 等人[54]提出了基于贝叶斯的查询驱动的双聚类方法 QDB.同时,给出了一种基于实验条件列表的联合方法,以实现关键词的多样性并免除必须事先定义阈值等问题.随后,Zhao 等人[55]对 QDB 方法进行了改进,提出了 ProBic 方法.虽然二者在概念上相似,但是也有不同之处.QDB 方法利用概率关系模型扩展贝叶斯框架,并用基于期望最大化的直接指定方法来学习该概率模型.Alqadah 等人[34]提出了一种利用低方差和形式概念分析优势相组合的方法,以发现在部分实验条件下具有相同表达趋势的基因.为了便于 OPSM 的查询,Jiang 等人[38]提出了带有行列表头的前缀树索引 pIndex,同时给出 4 种 OPSM 查询方法.

3) 约束型双聚类

目前,该问题的相关研究相对较少,是一种对挖掘与分析基因表达数据的新方法.Pensa 等人[35,36]提出了一种从局部到整体的方法来建立间隔约束的二分分区.该方法是通过扩展从 0/1 数据集中提取出来的一些局部模式来实现的.其基本思想是:首先,将间隔约束转换成一个放松的局部模式;然后,利用 k 均值算法来获得一个局部模式的分区;最后,对上述分区做后续处理来确定数据之上的协同聚类结构.随后,Pensa 等人[37]对文献[35,36]进行了扩展,主要的不同点有:(i) 作者同时在行列之上应用目标函数来评价双聚类的好坏;(ii) 新工作[37]将文献[35,36]中的数据从 0/1 矩阵扩展到了实数数据;(iii) 提升了 must-link 与 cannot-link 两类约束在行列之上的处理性能.Tseng 等人[56]提出了基于相关约束完整链接的约束型双聚类方法.

6 结 论

针对基因表达数据中保序子矩阵的约束查询问题,提出了适用于不同情形的索引和查询方法,即提出了基于数字签名和 Trie 的基因序列与实验条件序列索引方法,以及自顶向下与自底向上相结合的查询方法.索引方法极大地减小了索引的数据量并提升了索引速度;查询方法方便了不同类型的查询目的,且提升了查询性能.在真实数据集上的实验结果证明了所提出的方法具有很好的查询精确性和良好的可扩展性.

致谢 在此,向百忙之中评审我们论文的专家学者和对本文的工作给予支持与建议的同行表示感谢.

References:

[1] Ben-Dor A, Chor B, Karp R, Yakhini Z. Discovering local structure in gene expression data: The order-preserving submatrix problem. In: Proc. of the 6th Annual Int'l Conf. on Computational Biology (RECOMB). ACM Press, 2002. 49−57. [doi: 10.1145/565196.565203]

[2] Hartigan JA. Direct clustering of a data matrix. Journal of the American Statistical Association, 1972,67(337):123−129. [doi: 10.1080/01621459.1972.10481214]

[3] Cheng Y, Church GM. Biclustering of expression data. In: Proc. of the 8th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB). AAAI Press, 2000. 93−103.

[4] Yang J, Wang W, Wang H, Yu PS. δ-Clusters: Capturing subspace correlation in a large data sets. In: Proc. of the 18th Int'l Conf. on Data Engineering (ICDE). IEEE Press, 2002. 517−528. [doi: 10.1109/ICDE.2002.994771]

[5] Cho H, Dhillon IS, Guan Y, Sra S. Minimum sum-squared residue co-clustering of gene expression data. In: Proc. of the 12th SIAM Int'l Conf. on Data Mining (SDM). SIAM Press, 2004. 114−125. [doi: 10.1137/1.9781611972740.11]

[6] Divina F, Aguilar-Ruiz JS. Biclustering of expression data with evolutionary computation. IEEE Trans. on Knowledge and Data Engineering, 2006,18(5):590−602. [doi: 10.1109/TKDE.2006.74]

[7] Deodhar M, Gupta G, Ghosh J, Cho H, Dhillon IS. A scalable framework for discovering coherent co-clusters in noisy data. In: Proc. of the 26th Annual Int'l Conf. on Machine Learning (ICML). ACM Press, 2009. 241−248. [doi: 10.1145/1553374.155 3405]

[8] Cho H. Data transformation for sum squared residue. In: Proc. of the 14th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD). Berlin, Heidelberg: Springer-Verlag, 2010. 48−55. [doi: 10.1007/978-3-642-13657-3_8]

[9] Odibat O, Reddy CK. A generalized framework for mining arbitrarily positioned overlapping co-clusters. In: Proc. of the 19th SIAM Int'l Conf. on Data Mining (SDM). SIAM Press, 2011. 343−354. [doi: 10.1137/1.9781611972818.30]

[10] Ayadi W, Elloumi M, Hao JK. BicFinder: A biclustering algorithm for microarray data analysis. Knowledge and Information Systems, 2012,30(2):341−358. [doi: 10.1007/s10115-011-0383-7]

[11] Truong DT, Battiti R, Brunato M. Discovering non-redundant overlapping biclusters on gene expression data. In: Proc. of the 13th IEEE Int'l Conf. on Data Mining (ICDM). IEEE Press, 2013. 747−756. [doi: 10.1109/ICDM.2013.36]

[12] Ayadi W, Hao JK. A memetic algorithm for discovering negative correlation biclusters of DNA microarray data. Neurocomputing, 2014,145:14−22. [doi: 10.1016/j.neucom.2014.05.074]

[13] Chen S, Liu J, Zeng T. MMSE: A generalized coherence measure for identifying linear patterns. In: Proc. of the IEEE Int'l Conf. on Bioinformatics and Biomedicine (BIBM). IEEE Press, 2014. 489−492. [doi: 10.1109/BIBM.2014.6999206]

[14] Denitto M, Farinelli A, Bicego M. Biclustering gene expressions using factor graphs and the max-sum algorithm. In: Proc. of the 24th Int'l Conf. on Artificial Intelligence (IJCAI). AAAI Press, 2015. 925−931.

[15] Wang H, Wang W, Yang J, Yu PS. Clustering by pattern similarity in large data sets. In: Proc. of the 28th ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). ACM Press, 2002. 394−405. [doi: 10.1145/564691.564737]

[16] Liu J, Wang W. OP-Clustering by tendency in high dimensional space. In: Proc. of the 3th IEEE Int'l Conf. on Data Mining (ICDM). IEEE Press, 2003. 187−194. [doi: 10.1109/ICDM.2003.1250919]

[17] Wang H, Pei J, Yu PS. Pattern-Based similarity search for microarray data. In: Proc. of the 11th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD). ACM Press, 2005. 814−819. [doi: 10.1145/1081870.1081978]

[18] Kriegel HP, Kroger P, Renz M, Wurst SHR. A generic framework for efficient subspace clustering of high-dimensional data. In: Proc. of the 5th IEEE Int'l Conf. on Data Mining (ICDM). IEEE Press, 2005. 250−257. [doi: 10.1109/ICDM.2005.5]

[19] Jiang D, Pei J, Zang A. A general approach to mining quality pattern-based clusters from expression data. In: Proc. of the 10th Int'l Conf. on Database Systems for Advanced Applications (DASFAA). Springer-Verlag, 2005. 188−200. [doi: 10.1007/11408079_18]

[20] Gao BJ, Griffith OL, Ester M, Jones SJM. Discovering significant OPSM subspace clusters in massive gene expression data. In: Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD). ACM Press, 2006. 922−928. [doi: 10.1145/1150402.1150529]

[21] Zhang X, Wang W. An efficient algorithm for mining coherent patterns from heterogeneous microarrays. In: Proc. of the 19th Int'l Conf. on Scientific and Statistical Database Management (SSDBM). IEEE Computer Society Press, 2007. 32. [doi: 10.1109/SSDBM.2007.30]

[22] Yan L, Sun Z, Wu Y, Zhang B. Biclustering nonlinearly correlated time series gene expression data. Journal of Computer Research and Development, 2008,45(11):1865−1873 (in Chinese with English abstract).

[23] Zhang M, Wang W, Liu J. Mining approximate order preserving clusters in the presence of noise. In: Proc. of the 24th Int'l Conf. on Data Engineering (ICDE). IEEE Press, 2008. 160−168. [doi: 10.1109/ICDE.2008.4497424]

[24] Chui CK, Kao B, Yip KY, Lee SD. Mining order-preserving submatrices from data with repeated measurements. In: Proc. of the 8th IEEE Int'l Conf. on Data Mining (ICDM). IEEE Press, 2008. 133−142. [doi: 10.1109/ICDM.2008.12]

[25] Zhao Y, Yu J, Wang G, Chen L, Wang B, Yu G. Maximal subspace coregulated gene clustering. IEEE Trans. on Knowledge and Data Engineering, 2008,20(1):83−98. [doi: 10.1109/TKDE.2007.190670]

[26] Trapp AC, Prokopyev OA. Solving the order-preserving submatrix problem via integer programming. INFORMS Journal on Computing, 2010,22(3):387−400. [doi: 10.1287/ijoc.1090.0358]

[27] Fang Q, Ng W, Feng J. Discovering significant relaxed order-preserving submatrices. In: Proc. of the 16th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2010. 433−442. [doi: 10.1145/1835804.1835861]

[28] Fang Q, Ng W, Feng J, Li Y. Mining bucket order-preserving submatrices in gene expression data. IEEE Trans. on Knowledge and Data Engineering, 2012,24(12):2218−2231. [doi: 10.1109/TKDE.2011.180]

[29] Gao BJ, Griffith OL, Ester M, Xiong H, Zhao Q, Jones SJM. On the deep order-preserving submatrix problem: A best effort approach. IEEE Trans. on Knowledge and Data Engineering, 2012,24(2):309−325. [doi: 10.1109/TKDE.2010.244]

[30] Yip KY, Kao B, Zhu X, Chui CK, Lee SD, Cheung DW. Mining order-preserving submatrices from data with repeated measurements. IEEE Trans. on Knowledge and Data Engineering, 2013,25(7):1587−1600. [doi: 10.1109/TKDE.2011.167]

[31] An P. Research on biclustering methods for gene expression data analysis [MS. Thesis]. Suzhou: Soochow University, 2013 (in Chinese with English abstract).

[32] Fang Q, Ng W, Feng J, Li Y. Mining order-preserving submatrices from probabilistic matrices. ACM Trans. on Database Systems, 2014,39(1):No.6. [doi: 10.1145/2533712]

[33] Cho S, Na JC, Park K, Sim JS. A fast algorithm for order-preserving pattern matching. Information Processing Letters, 2015,115(2): 397−402. [doi: 10.1016/j.ipl.2014.10.018]

[34] Alqadah F, Bader JS, Anand R, Reddy CK. Query-Based biclustering using formal concept analysis. In: Proc. of the 12th SIAM Int'l Conf. on Data Mining (SDM). SIAM Press, 2012. 648−659. [doi: 10.1137/1.9781611972825.56]

[35] Pensa RG, Robardet C, Boulicaut JF. Towards constrained co-clustering in ordered 0/1 data sets. In: Proc. of the 16th Int'l Symp. on Methodologies for Intelligent Systems (ISMIS). Springer-Verlag, 2006. 425−434. [doi: 10.1007/11875604_49]

[36] Pensa RG, Robardet C, Boulicaut JF. Constraint-Driven co-clustering of 0/1 data. In: Proc. of the Constrained Clustering: Advances in Algorithms, Data Mining and Knowledge Discovery Series. 2008. 123−148. [doi: 10.1201/9781584889977.ch6]

[37] Pensa RG, Boulicaut JF. Constrained co-clustering of gene expression data. In: Proc. of the 8th SIAM Int'l Conf. on Data Mining (SDM). SIAM Press, 2008. 25−36. [doi: 10.1137/1.9781611972788.3]

[38] Jiang T, Li Z, Chen Q, Li K, Wang Z, Pan W. Towards order-preserving submatrix search and indexing. In: Proc. of the 20th Int'l Conf. on Database Systems for Advanced Applications (DASFAA). Part II. Berlin, Heidelberg: Springer-Verlag, 2015. 309−326. [doi: 10.1007/978-3-319-18123-3_19]

[39] Helmer S, Moerkotte G. Evaluation of main memory join algorithm for joins with subset join predicates. In: Proc. of the 23rd Int'l Conf. on Very Large Database (VLDB). ACM Press, 1997. 386−395.

[40] KiWi Software 1.0. 2012. http://www.bcgsc.ca/platform/bioinfo/ge/kiwi/

[41] Broad Institute. Datasets.rar and 5q_gct_file.gct. 1999. http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi

[42] Jiang T, Li Z, Chen Q, Wang Z, Li K, Wang Z. Parallel partitioning and mining gene expression data with butterfly network. In: Proc. of the 24th Int'l Conf. on Database and Expert Systems Applications (DEXA). Part I. Berlin, Heidelberg: Springer-Verlag, 2013. 129−144. [doi: 10.1007/978-3-642-40285-2_13]

[43] Jiang T, Li Z, Chen Q, Wang Z, Li K, Pan W. OMEGA: An order-preserving submatrix mining, indexing and search. In: Proc. of the European Conf. on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD). Part III. Berlin, Heidelberg: Springer-Verlag, 2015. 303−307. [doi: 10.1007/978-3-319-23461-8_35]

[44] Sim K, Gopalkrishnan V, Zimek A, Cong G. A survey on enhanced subspace clustering. Data Mining and Knowledge Discovery, 2013,26:332−397. [doi: 10.1007/s10618-012-0258-x]

[45] Kriegel HP, Kroger P, Zimek A. Clustering of high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans. on Knowledge Discovery from Data, 2009,3(1):1−58. [doi: 10.1145/1497577.1497578]

[46] Yue F, Sun L, Wang K, Wang Y, Zuo W. State-of-the-Art of cluster analysis of gene expression data. Acta Automatica Sinica, 2008,34(2):113−120 (in Chinese with English abstract). [doi: 10.3724/SP.J.1004.2008.00113]

[47] Jiang D, Tang C, Zhang AD. Cluster analysis for gene expression data: A survey. IEEE Trans. on Knowledge and Data Engineering, 2004,16(11):1370−1386. [doi: 10.1109/TKDE.2004.68]

[48] Madeira SC, Oliveira AL. Biclustering algorithms for biological data analysis: A survey. IEEE Trans. on Computational Biology and Bioinformatics, 2004,1(1):24−45. [doi: 10.1109/TCBB.2004.2]

[49] Smet RD, Marchal K. An ensemble method for querying gene expression compendia with experimental lists. In: Proc. of the 2010 IEEE Int'l Conf. on Bioinformatics and Biomedicine (BIBM). IEEE Press, 2010. 314−318. [doi: 10.1109/BIBM.2010. 5706583]

[50] Zou Q, Guo M, Liu Y, Wang J. A classification method for class-imbalanced data and its application on bioinformatics. Journal of Computer Research and Development, 2010,47(8):1407−1414 (in Chinese with English abstract).

[51] Zou Q, Li X, Jiang W, Lin Z, Li G, Chen K. Survey of mapreduce frame operation in bioinformatics. Briefings in Bioinformatics, 2014,15(4):637−647. [doi: 10.1093/bib/bbs088]

[52] Chen W, Cheng Y, Zhang S, Pan Q. Heuristic clustering method based on neighbor-seeds for 454 sequencing data. Ruan Jian Xue Bao/Journal of Software, 2014,25(5):929−938 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4547.htm [doi: 10.13328/j.cnki.jos.004547]

[53] Zou Q, Hu Q, Guo M, Wang G. HAlign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy. Bioinformatics, 2015,31(15):2475−2481. [doi: 10.1093/bioinformatics/btv177]

[54] Dhollander T, Sheng QZ, Lemmens K, Moor BD, Marchal K, Moreau Y. Query-Driven module discovery in microarray data. Bioinformatics, 2007,23(19):2573−2580. [doi: 10.1093/bioinformatics/btm387]

[55] Zhao H, Cloots L, Bulcke TV, Wu Y, Smet RD, Storms V, Meysman P, Engleen K, Marchal K. Query-Based biclustering of gene expression data using probabilistic relational models. BMC Bioinformatics, 2011,12(s1):S37. [doi: 10.1186/1471-2105-12-S1-S37]

[56] Tseng VS, Chen LC, Kao CP. Constrained clustering for gene expression data mining. In: Proc. of the 12th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD). Berlin, Heidelberg: Springer-Verlag, 2008. 759−766. [doi: 10.1007/978-3-540-68125-0_73]

附中文参考文献:

[22] 闫雷鸣,孙志挥,吴英杰,张柏礼.联合聚类非线性相关的时序基因表达数据.计算机研究与发展,2008,45(11):1865−1873.

[31] 安平.基因表达数据的双聚类分析方法研究[硕士学位论文].苏州:苏州大学,2013.

[46] 岳峰,孙亮,王宽全,王永吉,左旺孟.基因表达数据的聚类分析研究进展.自动化学报,2008,34(2):113−120. [doi: 10.3724/SP.J.1004.2008.00113]

[50] 邹权,郭茂祖,刘扬,王峻.类别不平衡的分类方法及在生物信息学中的应用.计算机研究与发展,2010,47(8):1407−1414.

[52] 陈伟,程咏梅,张绍武,潘泉.邻域种子的启发式 454 序列聚类方法.软件学报,2014,25(5):929−938. http://www.jos.org.cn/1000-9825/25/929.htm [doi: 10.13328/j.cnki.jos.004547]

姜涛(1983－),男,河南滑县人,博士,讲师,CCF 专业会员,主要研究领域为生物信息检索,数据管理,数据挖掘.

陈伯林(1985－),男,博士,副教授,CCF 专业会员,主要研究领域为生物信息学,数据挖掘,数据管理.

李战怀(1961－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据管理,数据挖掘.

李卫榜(1979－),男,博士生,主要研究领域为数据质量,云计算,数据管理.

尚学群(1973－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据挖掘,生物信息学,数据管理.

殷知磊(1986－),男,博士生,讲师,主要研究领域为生物信息学,数据挖掘,数据库管理.