第9卷第3期
1998年3月

软件学报
JOURNAL OF SOFTWARE

Vol. 9, No. 3
Mar. 1998

多维索引hB树的改进方法——hB*树*

金树东 冯玉才 孙小薇

（华中理工大学计算机科学与工程系 武汉 430074）

摘要 本文在hB树基础上提出多属性索引方法——hB*树. hB*树索引结点溢出时先寻求避免分裂，以期得到较好的空间利用率；通过避免和消除多父结点，使hB*树成为严格的树形结构. 本文表明hB*树提高了空间利用率，树形化的代价也不高.

关键词 存取方法，多维索引，B树，hB树，空间利用率.

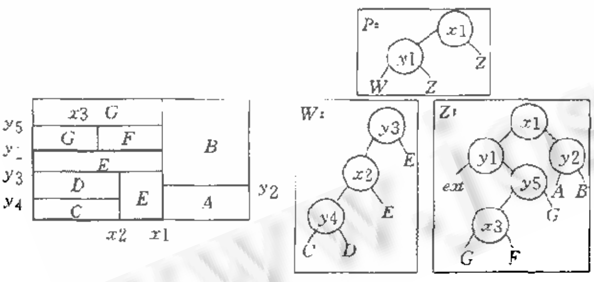
中图法分类号 TP311.13

为了有效地支持新应用如GIS、多媒体和CAD工具等，需采用新的存储结构和存取方法，特别是多维索引和搜索方法. 好的搜索方法应满足：(1) 高空间利用率和扇出；(2) 数据增加时索引树能健壮成长；(3) 适应多种存取操作，包括精确匹配和范围搜索. 许多有效的多属性和空间索引方法，如hB树[1~3]、各种R树[4~6]、Quad树[7]、TV树[8]、k-D-B树[9]和X树[10]等已经提出.

Lomet和Salzberg提出的hB树结点间搜索和增长方式类似于一维B树，结点内组织采用k维树，分裂时抽取k维树的一棵子树形成新的hB树索引结点. hB树具有较好的平衡性，但不能保证结点的单父性，因而可能成为一个有向无环图DAG. 另外，其空间利用率不高. 本文在hB树基础上提出一种多属性索引方法——hB*树[11]，在索引结点溢出时不立即分裂，而是先寻求与相邻结点的平衡. 它还采取措施避免和消除DAG结构，进行严格的树形化. 本文表明，避免分裂能提高空间利用率. DAG的避免和消除改善了hB*树的结构，且代价不高.

1 hB树简介

hB树是一种有效的多维动态索引结构，其结点间搜索和增长过程模拟B树的处理方法. 结点内采用k维树组织和进行高效搜索. 结点分裂要求k维树分裂，于是抽取一个大小介于1/3~2/3间并尽可能平衡的k维子树，并以此子树表示一个新结点. 因此hB树结点空间是有孔(holey)的k方体，即每个结点空间是被抽取若干个小k方体的k方体.



（a）二维空间的一个划分　　（b）对应的hB树结构

图1 hB表示空间划分的一个例子

图1是二维空间的一个划分和hB树结构. 图1(b)中结点W的k维树是从结点Z的k维树抽取的，如Z的k维树中的ext标识，它相当于图1(a)中粗线框内的矩形从整个大矩形中抽取. 结点P是该hB树的根，其k维树表示下层hB结点W，Z的导向路径，意为满足$x<x_1$且$y<y_1$的记录应存于结点W，其它记录则存于结点Z.

hB树能很好地支持精确匹配搜索和范围搜索. 进行精确匹配搜索，要经过自hB树根到一叶子的单一路径，访问的结点数是hB树的高度. 在hB树结点内通过比较k维树结点值与搜索的相应属性值，最终确定唯一的下层hB树结点. 范围搜索可能涉及hB树每层的多个结点. 范围的标界是在若干个属性上取值的上下界. 结点内搜索时将k维树结点的比较值与搜索范围的相应属性上下界进行比较，以决定转向k维树结点的左子树、右子树或两者.

---

* 作者金树东，1969年生，博士，主要研究领域为并行与分布数据库系统. 冯玉才，1945年生，教授，博士导师，主要研究领域为数据库，多媒体，人工智能，图象识别. 孙小薇，女，1970年生，博士生，主要研究领域为数据库，多维索引方法，地理信息系统.

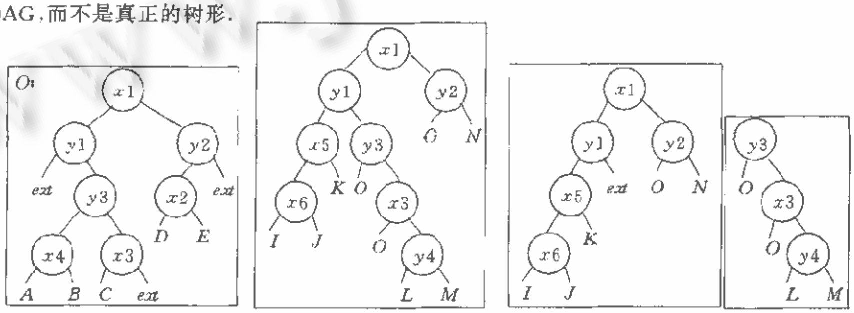
本文通讯联系人：冯玉才，武汉 430074，华中理工大学计算机科学与工程系

本文1996-12-17收到原稿，1997-04-28收到修改稿

在hB树中,插入记录与一维B树相似:首先搜索应插入记录的hB树叶子,若空间足够则插入即可,否则,从k维树中抽取一棵尽可能接近1/2的k维子树,并形成一新结点,然后将标志新结点的索引项插入到上层hB结点,这一插入可能导致再次分裂.由于必能从k维树中抽取1/3~2/3间的子树,hB树分裂保证不差于2:1的平衡.新结点的索引项是从原k维树根到抽取子树间的路径得到的,可压缩至不长于$2k$的路径,这样,在hB上层结点加入至多$2k$个k维树结点.

hB树的删除操作是插入操作的逆过程.如果删除不导致结点空间利用低于一个下限,则删除完成,否则,试图与相邻结点合并.这里,"相邻结点"是空间上包含它的结点,或从它分割出去的结点.合并的结果是两个结点的k维树组成单棵k维树,然后删除上层hB结点中的索引项,它可能导致上层又一次合并.

尽管hB树对于大多数情况能取得较好效果,但它有两个问题:(1)空间利用率不理想;(2)可能出现多父结点,使hB树不再是严格的树结构而成为DAG.问题(1)产生的原因是采用1/3~2/3的k维树分裂,结点分裂只保证不低于2:1的平衡,最坏空间利用率仅为1/3,而一维B^+树最坏为1/2.按文献[1]的估价,hB树平均空间利用率只有0.637,无法与一维B^+树的0.693相比.而高空间利用率意味着高扇出数,能降低搜索树高度和叶子层宽度,减少搜索时所需的I/O次数.问题(2)产生的原因在于hB树结点分裂出的k维子树数目不受限制.分裂时要将包含结点和抽取结点间的索引标界登记到其父结点中,如果有任意多个结点从包含结点中分离出来,则包含结点的k维树标界就很复杂.这样,父结点分裂时,不得不分割这一标界,分裂得到的两个上层结点成为包含结点的父结点.多父结点使hB树成为一个DAG,而不是真正的树形.



(a) 包含结点O的k维树 (b) 分裂前的父结点k维树 (c) 分裂后的两个父结点k维树

图2 hB树产生多父结点

图2给出一例加以说明,其中图2(a)是包含结点O的k维树,它被抽取3棵子树,以ext分别代表抽取结点K,L,N.图2(b)是O的父结点k维树,指明7个下层结点的划分范围.注意I,J是K的抽取结点,M则可看作L的抽取结点.现假定父结点溢出并分裂得到图2(c)的两个上层结点,可以看到它们都是结点O的父结点.多父结点使hB树要考虑到更复杂的DAG问题,因此产生了其它问题;而避免产生DAG结构要求不分隔包含结点在父结点中的索引,这会影响非叶子结点的分裂效果.

2 hB*树的基本特点

除了以下两点外,hB*树与hB树基本相同.①hB*树结点分裂前,首先尝试与相邻结点平衡以避免分裂;②hB*树中不存在多父结点.特性①模拟一维B*树的行为,尝试与相邻结点平衡数据量而不立即分裂,从而提高hB*树的空间利用率.特性②使hB*树成为真正的树,为此,采取DAG避免和消除方法.DAG避免意为减少多父结点出现的可能性,DAG消除则指产生多父结点时,对它进一步分裂及合并.

首先说明,本文沿用文献[1~3]中hB树的有关概念,与hB*树差别无关的性质也仍适用.为方便后续描述,还要定义几个概念.

定义1. 若hB*结点A的k维树被抽取一子树,形成结点B的k维树,则A称为B的包含结点,B称为A的抽取结点.

定义2. 在结点A的k维树中,导航至A的某抽取结点B对应的ext标志的路径,称为A到抽取结点B的路径,记作$Path(A,B)$.

定义3. 在结点A的父结点k维树中,导航至A的树结构片段,称为A的超路径,记为$SuperPath(A)$.

性质1. 有一个抽取结点的hB*树结点,其超路径长度$\leqslant 2k$.引自文献[1].

性质 2. 有 j 个抽取结点的 hB* 树结点，其超路径长度$\leqslant 2kj$. 由性质 1 推导出.

3 hB* 树的插入和删除算法

对 hB 树的一般插入和删除算法作改动，以适应 hB* 树的特点，插入算法描述如下：

算法 1. hB* 树一般插入算法

(1) 从 hB* 树根开始寻找记录关键字，若找到则出错返回，否则，得到记录应插入的叶子位置；

(2) 在要插入的结点中有两种可能性：

(a) 结点有足够空间存放插入数据，则插入完成并返回；

(b) 结点不足以存入数据，此时，

(Ⅰ) 若结点能与相邻结点平衡数据量，则在平衡后再重新插入；

(Ⅱ) 寻找满足平衡要求且不分割超路径的子树. 若成功则分裂结点，然后重新插入，最后在父结点中插入新结点索引项；

(Ⅲ) 否则，找到满足平衡要求的子树，产生下层多父结点，故进行一调整过程：对多父结点的再次分裂及合并. 最后分裂本层结点，然后重新插入，并在父结点中插入新结点索引项.

结点分裂总是会产生的. 若上述算法中结点溢出，而又不能如(Ⅰ)那样避免分裂，则要考虑结点分裂. 算法中(Ⅱ),(Ⅲ)两种情形都要进行分裂. 与 hB 树相似，分裂时总是找出一棵 k 维子树，分裂得到两个结点，包括新生成的抽取结点和更新后的包含结点. 最后在父结点中进行索引标志，它意味着在上层结点进行另一次插入并也有可能分裂. 按我们的术语，在父结点中的索引项称为超路径，它是一个 k 维树片段. 新结点的索引项是从包含结点导航至它的压缩路径，分裂时在父结点中插入一索引项要求至多增加 $2k$ 个 k 维树结点. 结点分裂的具体算法见文献[1]，这里不再详述.

下面给出 hB* 树的一般删除算法. 它与 hB 树的主要差别是要将稀疏结点与其包含结点平衡.

算法 2. hB* 树一般删除算法

(1) 从 hB* 树根开始搜索记录，若未找到，则出错返回；

(2) 否则，删除该记录，然后判断该结点空间利用率是否低于一下限，有两种可能性：

(a) 结点利用率仍不低于下限，则删除完成并返回；

(b) 结点利用率低于下限，此时：

(Ⅰ) 读取其包含结点(前提是包含结点与稀疏结点同父)，判断能否合并到包含结点. 若能，则完成合并，然后修改父结点中包含结点的索引项、删除原稀疏结点的索引项，这要求对父结点进行删除操作(即对其执行本算法的第 2 步)；

(Ⅱ) 若稀疏结点不能被合并至包含结点，则判断其能否与包含结点进行平衡，若不能，则算法也结束，否则，进行平衡并修改父结点中的索引项.

这里用于判断是否稀疏结点的下限应高于 hB 树的下限，如 0.5 或介于 0.4 和 0.5 间的某值. 另一差别是 hB* 树在不能合并时也考虑平衡. 但考虑到这种平衡的费用较大，要更新 3 个结点，因此，用于决定是否进行平衡的一个准则是，只有当平衡效果相当好时才进行. 由于 hB* 树与 hB 树的删除算法差别不大，我们不再详述.

4 分裂的避免

当结点剩余空间不足以插入时，hB 树的方法是立即分裂它. 这种方法使最坏和平均空间利用率不高. 因此，hB* 树模拟一维 B* 树的行为，试图在溢出结点及其相邻结点间平衡数据，来避免分裂而提高总的空间利用率. 平衡方法是在两结点间移动部分数据量，并改动它们的父结点中的索引项. 对一维 B* 树来讲，移动数据量的最小单位是单个键值，在父结点中改变的是结点的键值标界. 对 hB* 树来讲，移动数据量的最小单位是结点中 k 维树的分枝，改动的是父结点中的超路径.

B* 树被提出用于改善 B+ 树的空间利用率，但其费用较大，且 B+ 树本身就能保证很好的空间利用率，故 B* 树的好处不明显. 与 B+ 树相比，hB 树的空间利用率不理想，特别是在最坏情况下仅为 1/3. 如果通过插入时较少的费用而改善 hB 树的空间利用情况，特别是避免不平衡的分裂，则可期望获得较好的效果. hB* 树的插入算法就是以此为出发点的. 为保证 hB* 树插入时的花费较少而获益较大，我们提出以下准则.

准则 1. 只有当分裂将产生低平衡度时，才进行平衡尝试.

准则 2. 只有当分裂避免的可能性较大时，才进行平衡尝试.

针对 hB 树与 B 树的分裂平衡度不同，我们提出了准则 1. hB 树空间利用率低的原因是其分裂的平衡度差，但很多情况下它具有较平衡的分裂，此时我们应允许其进行分裂，而总是进行平衡尝试，花费很大. 针对 hB* 树与 B* 树得以平衡的概率不同，我们提出了准则 2. hB* 树与 B* 树移动数据单位有差别，B* 树邻近结点有空就可避免分裂的效果. 而在 hB* 树中则不然，分裂得以避免的可能性变小.

考虑平衡时的相邻结点选择及处理方法，相邻结点可选择包含结点和抽取结点.

4.1　与包含结点平衡

仅当溢出结点k维树很倾斜，即其左右子树之一很小时，才试图与包含结点平衡. 设hB*树的结点空间利用率不超过U的概率为p，U增加，则p增加. 判断溢出结点的左右子树之一大小是否小于$(1-U)$，若是，则避免分裂的可能性至少为p. 一般我们要求p大于0.5，如$p=0.8$.

进行平衡的直接方法是：将溢出结点及其包含结点k维树结合在一起，再重新抽取尽可能接近一半的子树，判断是否形成较平衡的两个结点. 容易证明最后抽取的子树必定也是溢出结点k维树的子树，因为查找不溢出且平衡的子树的过程必然要搜索到溢出结点的k维树根. 故可用以下步骤平衡两结点.

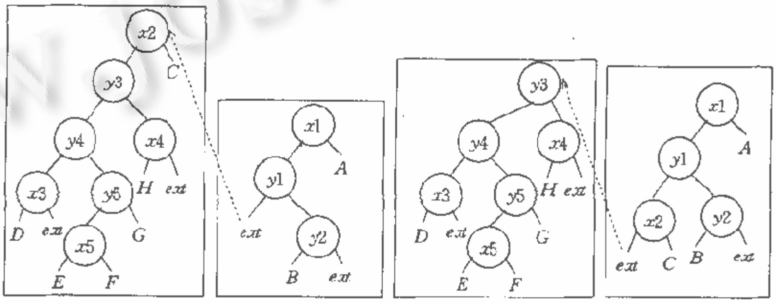
算法3. 与包含结点平衡的算法

(1) r=溢出结点的k维树根；

(2) 重复以下平衡步骤(a),(b),(c)，直到产生结果不比原情形更平衡：(a) 切割r的左右子树之小者；(b) 合并到包含树；(c) r=r的另一子树；

(3) 否则，平衡结束，判断结果是否不再溢出.

图3给出了一个与包含结点平衡的例子. 设hB*树索引结点的容量为N个k维树内结点，这里$N=6$，在抽取结点中插入数据溢出，因此启动以上平衡过程. 第1次判定将比较$x2$的k维树结点移至包含树. 第2次试图将比较$y3$和$x4$的结点合并到包含树，但产生结果不比原方案更好，因此平衡结束. 最后得到的结果如图3(b)所示.



(a) 平衡前　　(b) 平衡后

图3　与包含结点平衡

选择包含结点时考虑的一个特例是：若父结点超路径中，A和B是同一k维树结点的左右两边，而该k维树结点的更上层属结点C的超路径. 此时结点A的包含结点有两种选择，结点C或B均可. 结点B的包含结点也有两种选择，结点C或A均可.

4.2　与抽取结点平衡

当溢出结点有多个抽取结点时，需决定与哪个进行数据平衡. 若对多个结点都进行试验，为了获得较高的空间利用，花费将很大，这与追求总体高效的原则相背离. 因此我们的策略是，选择最有可能进行平衡的抽取结点. 这样不论是否能平衡，至多需读取一个结点.

考虑如何确定最有可能平衡的抽取结点. 进行平衡时，要将与抽取结点*ext*标识邻近的k维树分枝合并到抽取树. 若该分枝较大，则平衡成功的可能性就较小. 因此我们选择的抽取结点，与其相邻的k维树分枝是最小的. 当然这一分枝同样要满足小于$(1-U)$，若不满足，则仍将放弃平衡尝试. 平衡的方法是合并两个k维树并重新抽取尽可能接近一半的子树，判断其结果是否不再溢出.

图4是溢出结点与其抽取结点平衡的一个例子. 设结点容量$N=6$，在图4(a)中右边的包含结点中插入溢出，判定与其唯一的抽取结点相邻的k维树分枝很小，故与抽取结点平衡. 合并两个k维树并重新选取k维子树，得到图4(b)的结果. 以上探讨了与包含结点和抽取结点平衡的方法，为进一步限制其费用，我们还提出：

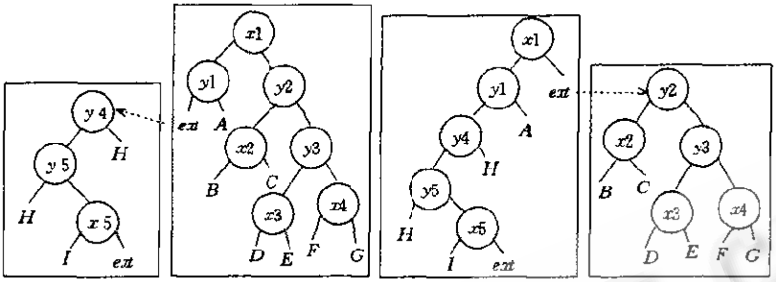
准则3. 只与一个相邻结点平衡，即从包含结点和所有抽取结点中选择最可能平衡的那个.

5　DAG的避免和消除

5.1　不分割超路径

显然多父结点只在hB*树非叶子结点分裂时才可能产生，因此，叶子结点分裂时不必考虑这一问题. 当内结点A分裂时，为使得hB*树成为真正的树结构，要求在分裂时不分割下层结点的超路径，这样就消除了多父结点产生的可能性. 如果合理选择抽取k维子树，往往能避免超路径被分隔到不同结点中. 从满足以下条件的所有候选分裂点中选择最平衡的分裂点.

(1) 若A不是叶子结点，则对A的任意子结点C，分裂点不分割$SuperPath(C)$.

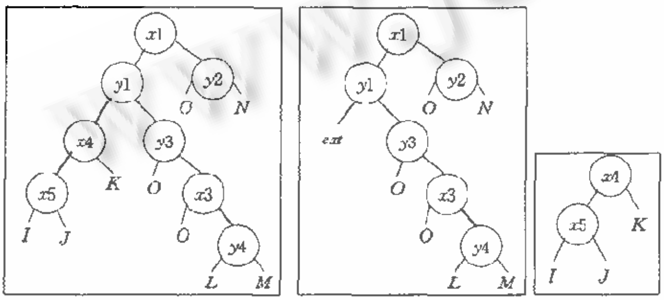


(a) 平衡前 (b) 平衡后

图4 与抽取结点平衡

(2) 保证至少 2:1 的平衡度.

保持超路径的整体性要求小心选择分裂点.选择的分裂点应当是下层结点超路径间的交界点.图 5 是选择抽取 k 维子树,而不分割超路径的的一个例子.图 5(a)中粗线为可选的超路径交界点,图 5(b)是最平衡的一个分裂方案,但其结果不满足 2:1 的平衡度.



(a) 分割前k维树 (b) 一个不分割超路径的分裂方法

图5 分裂时选择不分割超路径的k维子树与包含结点平衡

由于抽取点仅限于超路径交界,这样,当超路径很大时,找不到平衡分裂方案,即抽取子树远大于或小于 1/2.我们来检验超路径大小与分裂的平衡度间的关系.我们总可以找到一个介于 1/3～2/3 间的子树,但它不一定满足"不分割超路径",因此对其进行调整,将分割点移至超路径交界点.移动过程调整必少于 $2kj$ 个 k 维树结点,其中 j 为该超路径对应包含结点 k 维树被抽取的次数.若要求至少$(1/2+b)$:$(1/2-b)$的平衡度,则允许的超路径大小为 $N\times 2b$.即要求满足下式: $N\times 2b>2kj$ (1)

由于从 k 维树中抽取子树的平衡度只满足 2:1,因此这里也假定要求 2:1 的平衡度,从而 $b=1/6$,须满足 $N/3>2kj$.

一个 k 维树结点包括比较值、左右指针及一些标志位,约占 8～16 字节.以 12 字节估算,典型的页尺寸为 4K 的系统中,除去索引结点的其它信息,N 约为 340.为满足 $N/3>2kj$,k 的几个较小取值与 j 的关系如表 1.

表 1

k	2	3	4	5	6	…
j	<28.3	<18.9	<14.2	<11.3	<9.4	……

该估算仍是悲观的,因为每次抽取要求 $2k$ 个 k 维树结点的估计往往过多.即使如此,仍可看出,当维数较低时,允许每个 hB*结点 k 维树被抽取很多次,而不影响分裂时较好的平衡要求.

5.2 次平衡分裂

抽取结点数的增加,导致在上层结点分裂时难以选择平衡,且不分割超路径的方案.因此,要在下层结点分裂时即考虑减少抽取结点数.如果满溢结点的原有抽取结点数仍低于按式(1)得到的 j 值,则本次分裂暂不必考虑限制抽取结点数,而选择尽可能平衡的分裂方案.否则,要寻求不坏于 2:1 平衡度,而又不增加抽取结点的次平衡方案.

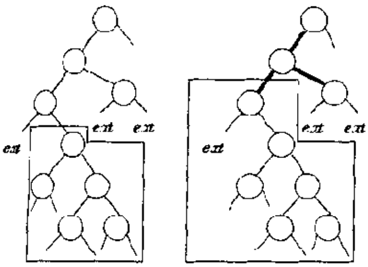
为了不增加抽取结点,需要在溢出结点 A 的 k 维树根到所有 ext 标志的路径上寻找分割点.即这些候选分隔点满足:

(1) 若 A 不是叶子结点,则对 A 的任意子结点 C,分裂点不分割 $SuperPath(C)$.

(2) 存在某个抽取结点 B,分裂点在 $Path(A,B)$上.

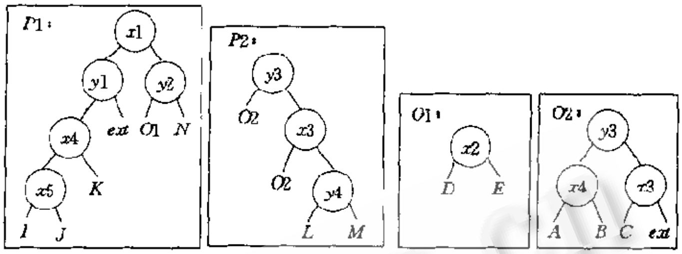
(3) 保证至少 2:1 的平衡度.

从所有候选分割点中,选择最接近 1/2 的分割点.如图 6 所示的分裂例子,设 $j=3$,不能任选最平衡的抽取子树,否则,如图 6(a)所示,得到包含结点的抽取结点过多.故从图 6(b)中粗线标明的可能的分隔点中选择最好的分隔点,方框内为实际选择的抽取子树.



(a) 最平衡而抽取过多　(b) 选择不增加抽取的分割点

图6　限制抽取结点数的分裂



(a) 分裂后的两个父结点 k 维树　(b) 导出分裂的两个子结点 k 维树

图7　hB*树消除DAG结构

hB树中也体现了减少抽取结点数的思想，但只考虑到在 k 维树根上分裂，若 k 维树根的左右子树满足不低于2:1的平衡，则抽取左子树或右子树，选择的范围较小. hB*的方法可看成是对hB树这种策略的一般化，可选择到所有 ext 的路径上的任意点进行 k 维子树抽取.

5.3　DAG的消除

然而，当没有满足最低平衡要求，而又需保证超路径不被分割的分裂方案时，仍然会不可避免地产生多父结点. DAG结构使hB树更复杂，故有必要消除出现的多父结点. hB*树中消除DAG结构的方法是：导出对下层结点的再次分裂和合并. 设 P 为一个hB*树内结点，在 P 中插入溢出，并设不得不分隔下层结点 A 的超路径.

算法4. 消除多父结点的分裂算法

1. 首先结点 P 分裂形成两个结点 $P1$，$P2$.
2. 然后按 P 的 k 维树中超路径的分隔点对 A 也进行分裂，得到两个下层结点 $A1$，$A2$. $A1$ 是 $P1$ 的子结点，$A2$ 是 $P2$ 的子结点. 不妨设 $A1$ 为包含结点，$A2$ 为抽取结点.
3. 若 $A1$ 的包含结点也是 $P1$ 的子结点(经常如此)，且 $A1$ 的空间利用低于一下限，则试图将 $A1$ 与其包含结点合并.
4. 若 $A2$ 有抽取结点，且 $A2$ 的空间利用低于同一下限，则试图将 $A2$ 与其抽取结点合并.

图2中的上层结点分裂形成DAG结构. hB*树的方法则是将上层结点 P 分裂后形成图7(a)的 $P1$，$P2$，对结点 O 也进行导出的分裂，得到图7(b)的两个结点 $O1$，$O2$. 这里 $O1$ 可试图与同父的包含结点 N 进行合并. $O2$ 有两个抽取结点 L，M，因此，它与其中之一试图合并.

6　效果分析

(1) 避免分裂的损益

我们提出了避免分裂时应考虑的3个准则. 第1个准则保证只有当分裂平衡度坏时才考虑避免分裂. 若hB*树分裂总是相当平衡，则它与hB树相同，不需额外费用. 第2个准则使平衡尝试往往能成功. 第3个准则进一步限制平衡尝试的费用，又能期望最大收获.

与包含结点平衡时，只有当溢出结点的 k 维树根的左右子树之一很小时($<1-U$)，才读取包含结点，这时得以合并的可能性高于 p. 若 $p=0.5$，则每两次索引结点读取至少节约1个结点的空间，往往 p 更大. 与抽取结点平衡时，只有当溢出结点的 k 维树的很小分枝有 p 的可能性合并到邻近抽取结点时，才进行尝试. 其时空损益与前者相同.

(2) 避免和消除DAG的效果

不分隔超路径使抽取子树的选择范围变小，因而使分裂平衡性受到影响. 但考虑到：①抽取结点数往往很少；②与hB树一样保证不低于2:1的平衡. 因此，对平衡性影响很小.

在抽取子树选择方面，hB*树明显优于hB树. hB树选择在 k 维树根上分裂，若根的两子树的平衡度不低于2:1，即使溢出结点的抽取数很小甚至没有时，也采用次平衡分裂方案. hB*树则在有限次抽取内，总是寻找最优的平衡分裂；而如果抽取过多，则选择根到所有 ext 结点上的候选分裂点，当然根本身也成为候选分裂点. 因此，hB*树往往具有更平衡的效果.

当hB树内结点分裂产生DAG结构时，不消除它，因此，分裂时的费用是写3个结点，包括包含结点、抽取结点及父结点. hB*树对下层多父结点进行导出的分裂，因此又多更新两个结点，并且使hB*树索引结点数增加. 这两个结点还能尝试与各自相邻结点合并以改善空间利用，每次尝试不论是否成功，均需额外的一次读取. 如果任一合并可行，则hB*的空间利用率不受损，此时消除DAG的额外费用仅为写两个结点和读1～2个结点.

（3）与R树的比较

在各种多维索引方法中，R树类是最广泛使用且高效的一种. R树中的主要问题是其最小化边界矩形会出现重叠，特别是当维数增加时重叠更严重，其性能迅速下降.[10]而hB树和hB*树对维数不敏感. 为减少R树中的空间重叠，有很多策略提出，其中一个方法是，为达到高度聚簇化而采用非平衡的分裂. 与hB树相似，它也可能导致低空间利用率. 但是R树中的获得高空间利用率与降低重叠是相互冲突的[6]，而在hB*树中高空间利用率必然意味着查询高效.

7 结 论

本文简单介绍了多维动态索引方法hB树，对其改进得到hB*树. hB*树提高了hB树的空间利用率，并对其进行严格的树形化. 通过模拟一维B*树在分裂时的行为，hB*树可望得到较高的空间利用率，而其花费又有限. 在hB*树索引结点分裂时，在保证好的平衡性的前提下，尽可能选择不引发DAG结构的分裂方案. 一旦分裂产生多父结点，则立即进行导出式的对多父结点进行分裂. 本文表明，避免分裂有益于提高空间利用率，避免DAG结构对分裂平衡性影响很小，消除DAG结构只需极小的费用. 进一步的研究包括：在实际系统中实现hB*树，hB*树与hB树及其它多维索引的实验比较.

参考文献

1 Lomet D, Salzberg B. The hB-tree: a multiattribute indexing method with good guaranteed performance. ACM Transactions on Database Systems, 1990,15(4):625～658

2 Salzberg B, Lomet D. Spatial database access methods. ACM SIGMOD Record, 1991,20(3):5～15

3 Evangelidis G, Lomet D, Salzberg B. The hB-tree: a modified hB-tree supporting concurrency, recovery and node consolidation. In: Dayal U eds. Proceedings of International Conference on Very Large Data Base. Zürich. Morgan Kaufmann Publishers, 1995. 551～561

4 Guttman A. R-trees: a dynamic index structure for spatial searching. In: Yormark B eds. Proceedings of ACM SIGMOD. Boston, MA: ACM Press, 1984. 47～57

5 Sellis A, Roussopoulos, Faloutsos C. The R+-tree: a dynamic index for multi-dimensional objects. In: Stocker P ed. Proceedings of International Conference on Very Large Data Base. Brighton, England: Morgan Kaufmann Publishers, 1987. 1～24

6 Bechmann N, Kriegel H, Schneider R *et al*. The R*-tree: an efficient and robust access method for points an rectangles. In: Molina H *et al* eds. Proceedings of ACM SIGMOD Conference on Management of Data. Atalantic City, NJ: ACM Press, 1990. 322～331

7 Samet H. The quadtree and related hierarchical data structures. ACM Computing Surveys, 1984,16(2):187～260

8 Lin K, Jagadish H V, Faloutsos C. The TV-tree: An index structure for high-dimensional data. VLDB Journal, 1994,3:517～542

9 Robinson J T. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In: Lien Y eds. Proceedings of ACM SIGMOD Conference on Management of Data. Ann Arbor: ACM Press, 1981. 10～18

10 Berchtold S, Keim D A, Kriegel H. The X-tree: an index structure for high-dimensional data. In: Vijayaraman T *et al* eds. Proceedings of International Conference on Very Large Data Base. Bombay, India: Morgan Kaufmann Publishers, 1996. 28～39

11 Jin S, Feng Y, Sun X. Improved hB-tree with higher space utilization and eliminating DAG. In: Shoval P *et al* eds. Proceedings of International Workshop on Next-Generation Information Technologies and Systems. Israel, 1997. 162～171

The hB* Tree——an Improved Multidimensional Indexing Method of hB-Tree

JIN Shu-dong FENG Yu-cai SUN Xiao-wei

(*Department of Computer Science and Engineering Huazhong University of Science and Technology Wuhan* 430074)

Abstract This paper is proposed a new multiattribute index method named hB*-tree on the basis of hB-tree. When an index node overflows, the first step is to avoid splitting if splitting will lead to poor balance degree. Therefore the node utilization of hB*-tree is improved. The DAG problem of hB-tree is also reduced by careful selection of extracted k-d-subtree. If a splitting still produces DAG structure, the hB*-tree is reorganized to be a strict tree. The authors show that hB* tree has reasonable space utilization and access costs.

Key words Access methods, multidimensional index, B-tree, hB-tree, space utilization.

Class number TP311.13