## OMT：重叠最小化自上而下R树的批量加载算法

序言：多维数据空间中的有效索引构建对于许多数据库应用程序都很重要，因为这样的索引构建时间和查询处理中的性能提升都很重要。为了在科学和数据仓库环境中有效地管理多维数据，基于R树的索引结构已被广泛使用。在本文提出了一种基于R树的自顶向下的批量加载算法称为OMT，可最大程度地减少目录节点之间的重叠区域从而极大地提高了查询性能。我们工作的主要思想是树的顶层节点之间的重叠比叶子节点更为关键。我们首先通过几次计算确定所得R树的拓扑，然后对数据项进行分组以创建根节点的条目。由于我们从上到下创建树，因此我们控制目录节点之间的重叠区域。之后，我们递归地划分每个条目以创建遵循100％空间利用率约束的较低级别的节点。这种方法可以通过减少不必要的节点访问来提高所得R树的查询性能。我们提出了OMT的详细算法。

### 1 介绍

R树是多维数据的常见索引技术，广泛用于空间和多维数据库。通常，我们通过一次插入一个对象来构建一棵R树。但是，这是一个缓慢的操作，并且会产生空间利用率低且重叠大的R树。

批量加载方法用于在相对较短的时间内加载大量数据集并生成结构良好的R树。它产生的R树的空间利用率接近100％，并且查询时间得到了改善（由于执行查询时需要访问较少的节点）。

已有的学者已经提出了几种批量加载方法，其中大多数试图在实现100％空间利用率的同时最小化节点之间的重叠区域[2，3]。他们首先对N个数据项的数据文件进行预处理，以使数据项在M个项的N / M个连续组中排序，其中M是可以放置在同一节点中的最大条目数。然后，它们将这些节点递归打包，将每个节点视为下一个数据项，直到创建根节点为止。

由于它们使用自下而上的方法，因此上层节点的创建要晚于叶节点。这通常导致目录节点之间的重叠很大，结果，R树在查询评估中的性能较差。这将在第2节中详细说明。

在本文中，我们提出了一种自顶向下的批量加载算法，该算法将目录节点之间的重叠最小化。由于我们可以通过100％的空间利用率约束来确定生成的R树的拓扑，因此我们可以从上到下高效地创建R树。结果，与以前的方法相比，我们可以实现相同的空间利用率和更好的查询性能。

在以下部分中，我们将介绍我们的工作动机和OMB的详细算法。

### 2 OMT：重叠最小化自上而下的批量加载算法

在本节中，我们描述了我们的工作动机，然后详细描述了我们的算法。在本节的最后部分，我们讨论了R树的构建成本。

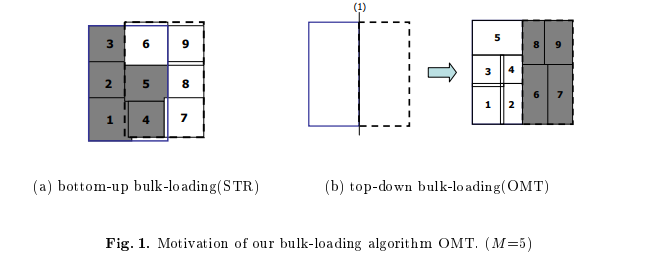
### 2.1 我们的工作动机

在STR方法中（这是批量加载区域中的最新技术），它们使用N / M个垂直切片“平铺”数据空间，以便每个切片包含足够的数据项以打包大约个节点[3]。

为此，它们按x坐标对数据项进行排序，并将它们划分为个垂直切片。然后，他们通过y坐标对每个切片的数据项进行排序，并通过将其分组为长度为M的行将这些数据项打包为节点。结果，生成了所有叶子节点。通过将叶子节点的每个MBR视为数据项，并将这些MBR递归打包到下一级节点，直到创建根节点，才能创建目录节点。

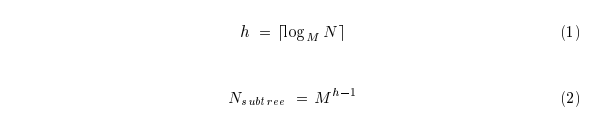
如图1（a）所示，由于100％的空间利用率约束，目录节点的MBR往往具有较大的重叠。例如，如果我们假设图1（a）中的最大条目数（= M）为5，则第一个目录节点应覆盖5个叶节点，以满足空间利用约束。结果，不可避免地要与另一个目录节点有很大的重叠。为了执行查询，我们检索并检查在每个级别[1]与查询区域Q相交的每个矩形。从根开始可以遵循几个路径。因此，重要的是最小化R树顶部的重叠区域。如果可能的话，最根本的是减少在树的顶层进行查询处理期间进行多个路径查找的可能性。

如图1（b）所示，自顶向下方法可以减少目录节点之间的重叠，因为它首先从上到下创建了节点。

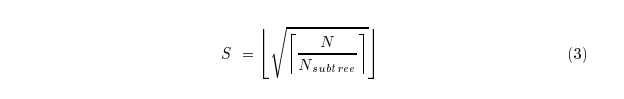


### 2.2 OMT算法的细节描述

算法的第一步是确定由批量加载产生的树的拓扑。由于100％的空间利用率限制，因此这是可能的。为此，我们首先通过等式（1）确定R树的高度。据此，我们可以确定根节点条目为根的一个子树中数据项的数量。由于我们将以100％的空间利用率打包数据项，因此除少数几个节点外，大多数节点将具有M个条目。据此，我们可以计算子树的大小。计算方式见等式（2）。



现在，我们可以确定根节点的条目数和顶层的垂直切片数。这在等式（3）中给出。这也表示要从一个垂直切片（最后一个切片除外）生成的节点数。请注意，最后一个垂直切片可能会导致超过S个条目。我们可以通过将数据项划分为S组来创建根节点的条目。



使用等式（3），我们可以通过将数据项划分为S组来创建根节点的条目。每个垂直切片被划分为S组，每个组构成根节点的条目。

接下来，我们将根的每个条目划分为M组。每个组又将是较低级别的条目。我们将较低级别的每个组递归地划分为M个组，直到每个组仅包含M个数据项。

### 2.3 我们的批量装载方法的建设成本

我们假设叶子节点级别为1，根级别为h。在我们的算法中，除根级别外，每个级别中的节点都分为M组。为此，我们需要按x和y坐标在彼此级别上对数据项进行排序。当我们开始按x坐标对它们进行排序时，在级别h-2中，我们将数据项按y坐标进行分类，而在级别h-4中，我们将其按x坐标进行排序，依此类推。

这似乎代价很大。但是，与STR方法相比，该方法花费不多，该STR方法通过x坐标对整个数据进行一次排序，并通过y坐标对每个垂直切片中的数据项进行排序。由于我们尚未进行任何实验，因此无法准确与STR比较建造成本。通过我们的分析评估，它表明我们的算法构建R树的成本是STR的两倍。但是，它仍然比传统的一对一插入方法快得多。

尽管对于批量加载算法而言，尽快构建R树很重要，但是批量加载的更重要的作用是构建可以有效执行查询的R树。我们的算法实现了与STR相同的空间利用率，并且通过大大减少目录节点之间的重叠来提高查询性能。

### 3 总结以及未来展望

在本文中，我们提出了一种自顶向下的批量加载算法，以提高R树的查询性能。先前提出的批量加载方法是自下而上地构建R树。 这不可避免地导致目录节点之间的高重叠区域。结果，它在搜索查询操作期间进行了不必要的遍历，从而降低了R树的性能。

我们工作的关键思想是，上级节点之间的重叠比叶级中节点的重叠更为关键。因此，我们设计了一种算法，该算法首先创建顶级节点。

由于我们尚未进行任何实验，因此我们需要在不久的将来证明我们的想法。我们认为查询性能将足以抵消较高建造成本的缺陷。

对于未来的工作，我们将开发改进的方法以有效地构建R树。

### 引用

1. Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. InProceedings of the 1984 ACM-SIGMOD Conference, pages 47-57, June 1984.
2. Ibrahim Kamel and Christos Faloutsos. On packing R-trees. In Proceedings of thesecond international conference on Information and knowledge management, pages490-499, 1993.
3. ST Leutenegger, J.M Edgington, and M.A Lopez. STR: A Simple and EfficientAlgorithm for R-Tree Packing. In Proceedings of the IEEE Data Engineering, pages497-506, 1997.