目录

1 索引架构

2 论文优势

3 查询更新

4 实验结果

1　索引架构

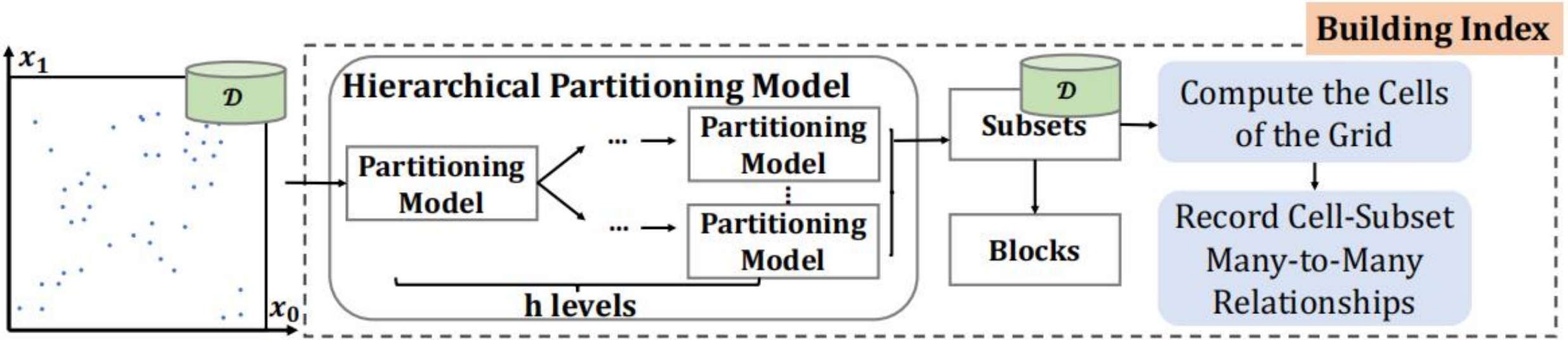


Fig. 1: FHSIE index building process overview

1）　给定分层数h，首先根节点使用Kmeans聚类方法对整个数据集进行聚类，然后自顶向下缩小聚类空间，递归划分空间直至到达底层层数h。对于底层（叶子）聚类，如果数据量超过块大小，那么就存储在多个块中，并保存分割点（指的是每一块的第一维起始值）。

1　索引架构

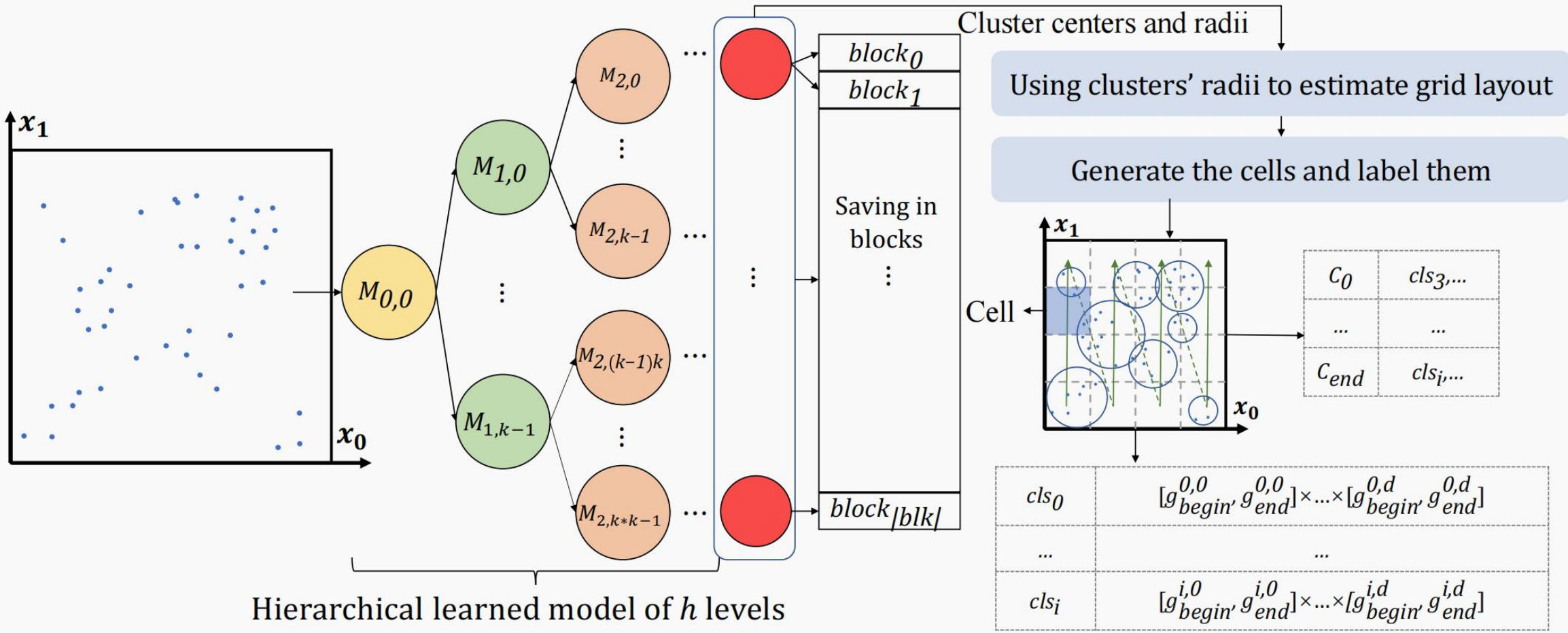


Fig. 2: FHSIE index structure.

1　索引架构

2）　划分网格：由于网格的布局会影响到查询效率，论文给出一种简单但有效的网格布局方法，把cell的边长设置为前5%的聚类半径大小。然后在底层聚类和cell之间建立一个多对多的联系，对于每个cell，维护一个和它相交的底层聚类的列表。

3）　内层保存聚类中心，叶子保存聚类中心和聚类半径，并保存聚类对应的块信息（块ID和分裂点）。

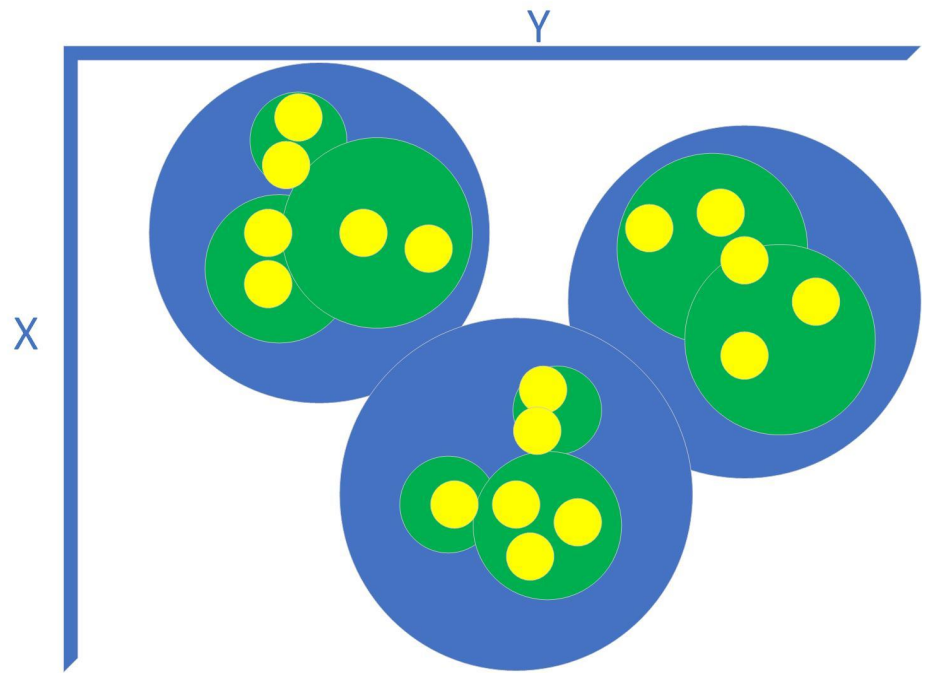
2　论文优势

相比以前的学习索引，论文给出的优势是对于以前基于监督学习（回归）的索引，它们固有的弊端是不够精确，而这个基于无监督的索引召回率是1。

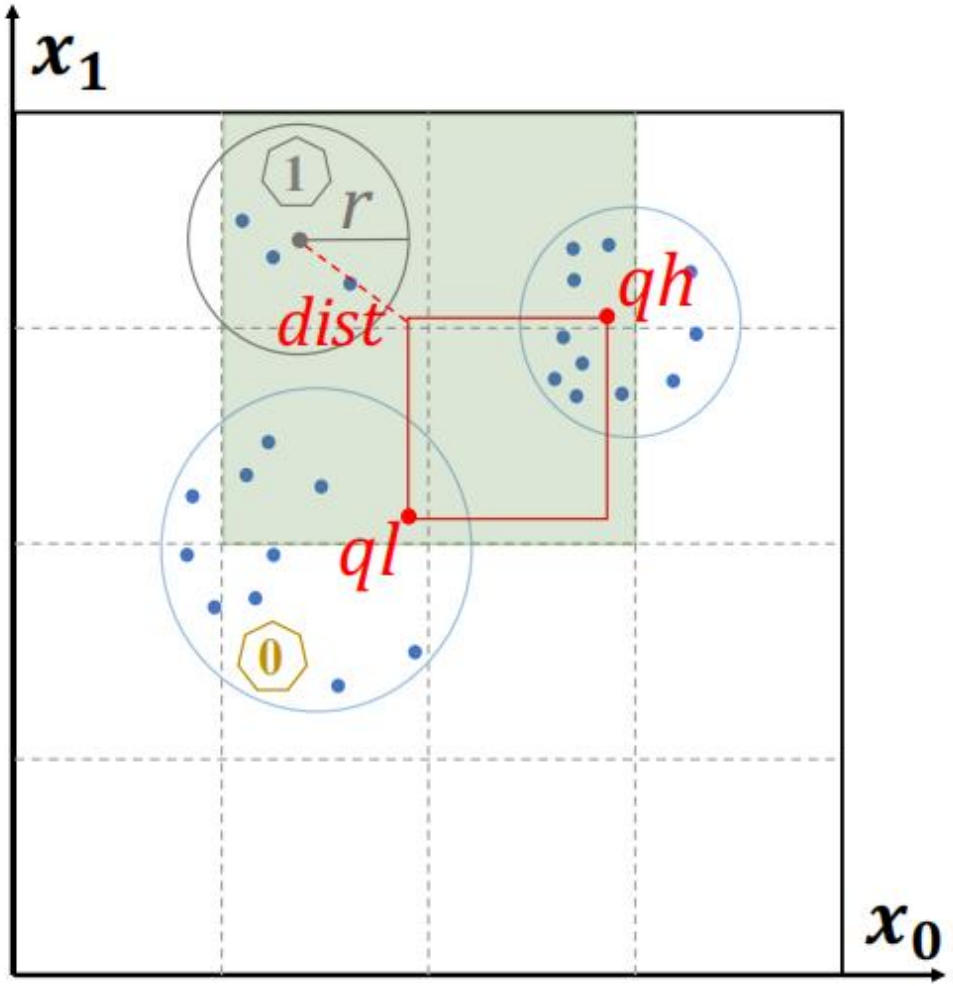
那么其实构造R树本质上也是聚类。这个优势其实不够充分。

3　查询更新

1）　点查询：对于已经训练好的模型，从根节点开始，从上到下递归选择聚类（标准是最近的欧氏距离），直至底层聚类，然后在聚类中寻找包含查询点的数据块，由于在底层保存了各个块的第一维起始值，所以可以对第一维执行二分查找来过滤目标块，然后扫描目标块。



3　查询更新



(a) Window query

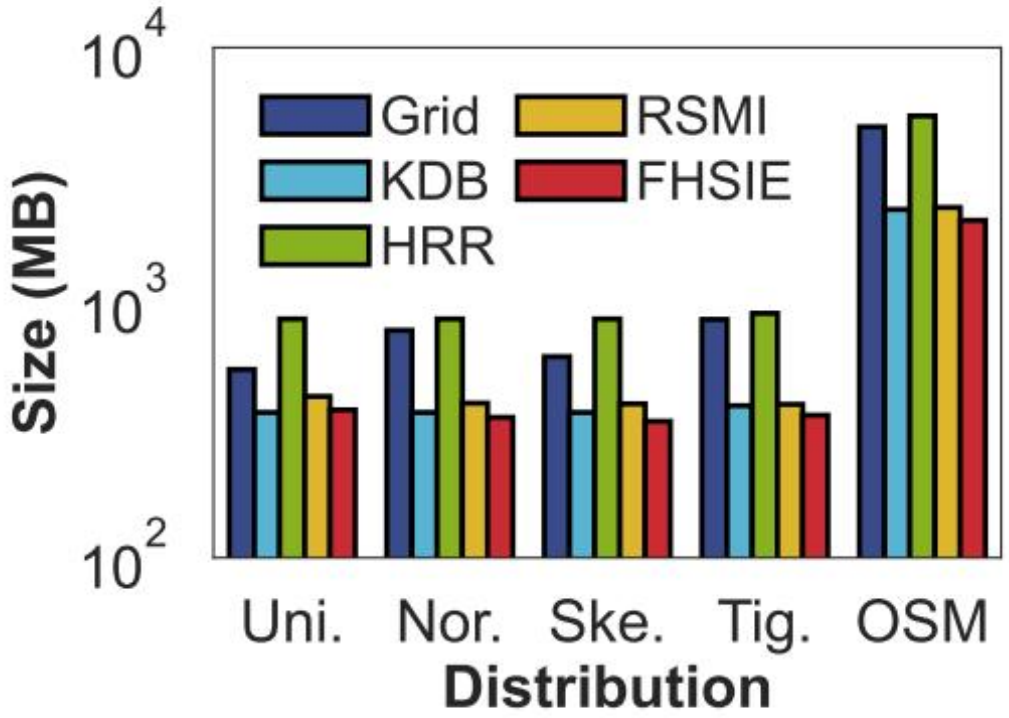
2）　范围查询和点查询不一样，首先找到和查询范围相交的cell，由于保存了和cell相交的底层聚类，所以可以很轻松得到它们（称为候选聚类），然后通过比较候选聚类中心到查询框的距离和聚类半径的大小来选择聚类，最后在聚类中过滤和查询相交的数据块，并扫描数据块。

3　查询更新

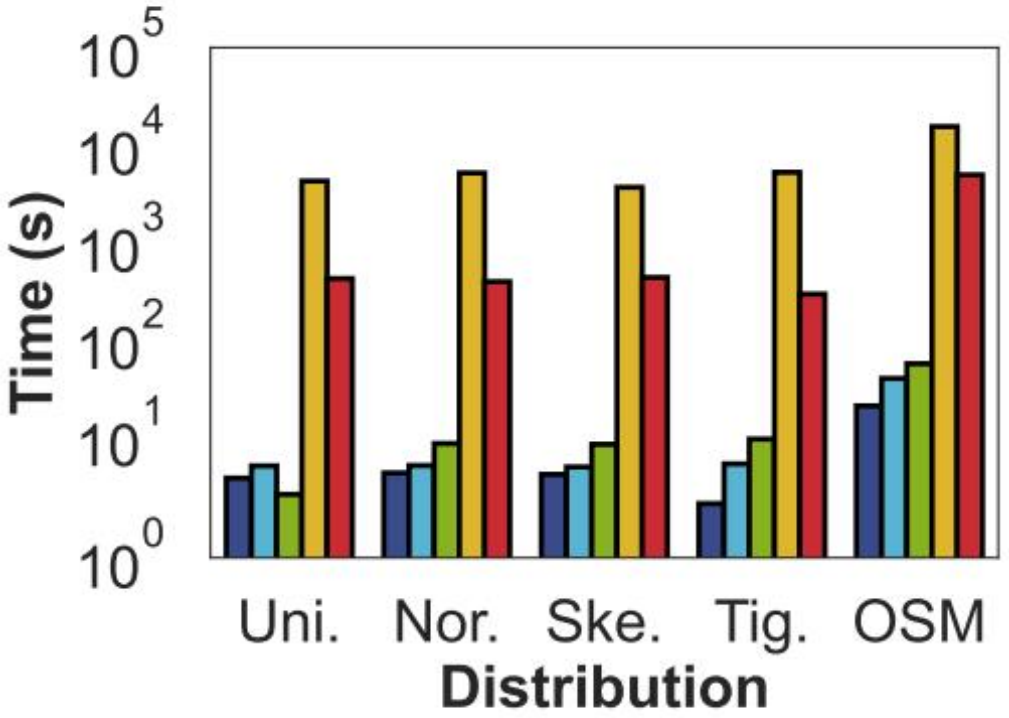
3）　对于更新，过程和点查询类似，找到底层聚类，然后过滤数据块，并将点插入数据块，如果块没有溢出，则直接更新分裂点；如果块溢出那么就简单分为两块，每一块数据是B/2，然后根据第一维值来插入到两块中合适的块，同时更新分裂点。然后需要考虑底层聚类的半径是否增大，如果增大，则需要更新和cell相交的聚类列表。

4 实验结果

合成数据（前三个）大小是2.56亿，Tig数据是1600万，OSM数据是1亿



(a) Index Size

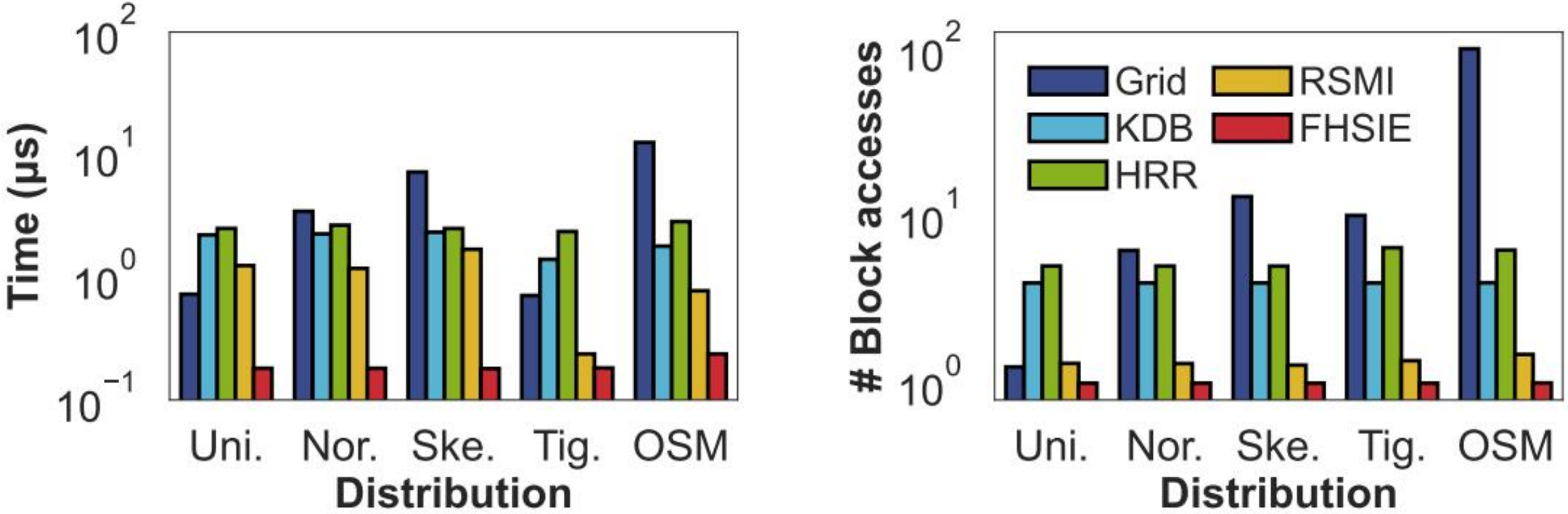


(b) Construction Time

Fig. 5: Index size and construction time vs. distribution

4 实验结果

点查询：对数据集中每个点执行点查询，块容量为100

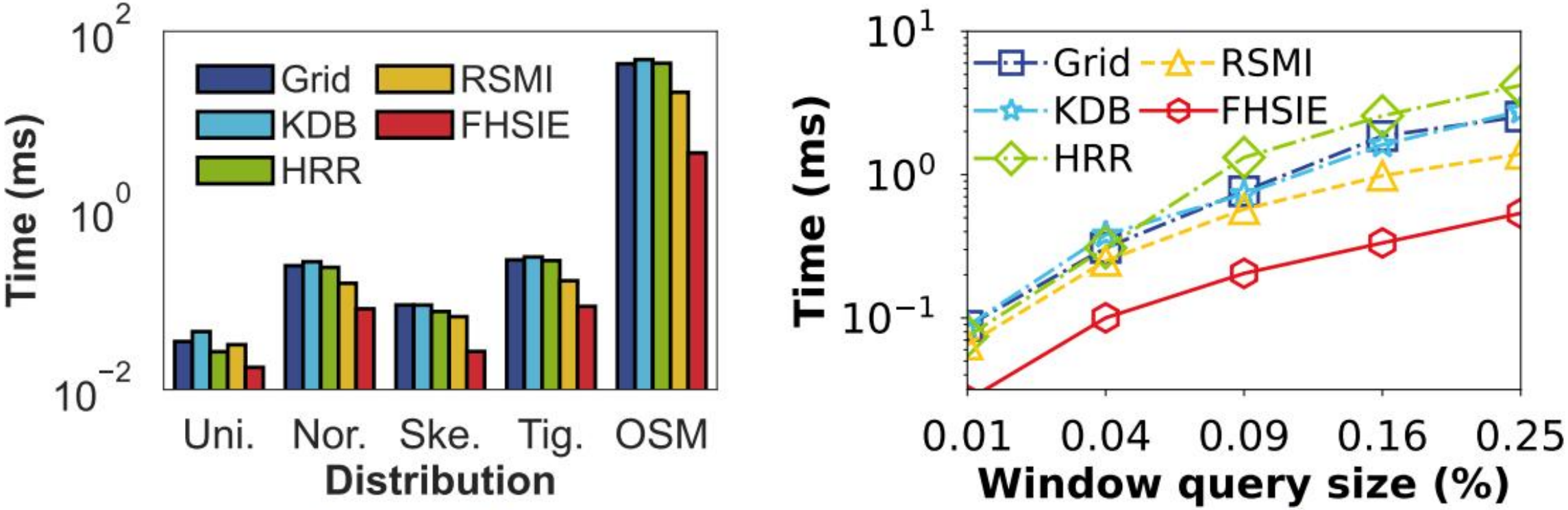


(a) Query time (b) Block access

Fig. 6: Point query vs. distribution

4 实验结果

内存中执行范围查询：在每个查询窗口下执行1000个范围查询，共5000个查询

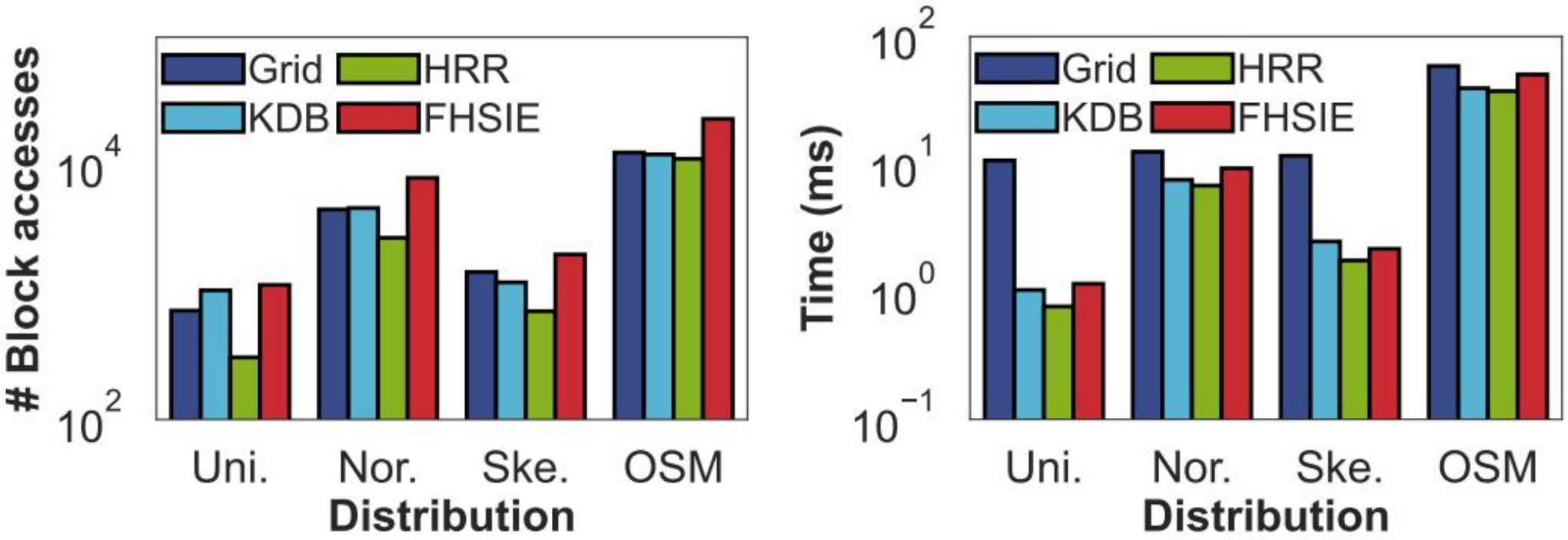


(a) Query time (b) Query time

Fig. 7: Window query vs. distribution vs. window size

4　实验结果

外存中执行范围查询：RSMI没有外存版本



This is because FHSIE uses only dimension-0 coordinates to prune the data blocks in the final pruning step, which suffers in the number of block accesses compared with using MBRs by the traditional indices。

4 实验结果

TABLE III: Window Query Performance against LISA

	Skewed		OSM	
Model	Response time (ms)	Recall	Response time (ms)	Recall
LISA	0.26	99.99%	30.75	99.88%
FHSIE	**0.17**	**100%**	**12.37**	**100%**

谢谢观看！