实现自适应空间文本分割树

王梓涵　周昕逸　刘权

1　理论介绍

在日常的信息传播中，常常需要处理用户所提供的一系列包含空间和关键词的查询项。例如：一些基于位置信息的推荐系统可根据用户的需求推送相关产品的信息，Twitter 等社交网络可分析用户发布的内容及附带的地理位置标签推送他们感兴趣的内容。本文实现的自适应空间文本分割树（Adaptive spatial-textual Partition Tree，以下简称 AP 树）可以通过自动计算关键词和空间因子两种分割方式的代价选择合适的匹配方式。

AP 树的实现有三大挑战。第一，在实际应用中的数据量极大，算法效率的提升可以节省很多成本。第二，算法吞吐量要大以应对源源不断的包含空间文本信息的对象流。第三，需要建立新的空间因子和关键词的索引机制用于不同的分配方法。

AP 树的基本运算包含三个部分。匹配对象和查询项的算法，计算关键词和空间因子分割代价的模型和建立索引的算法。

AP 树包含三种类型的结点：关键词结点（k-node），空间结点（s-node）和查询项结点（q-node）。一棵 AP 树的叶子结点都是 q-node，每一个查询项都可根据其关键词和空间因子被分配入一个或多个叶子结点中。运用关键词分割方法的结点称为 k-node。我们认为所有的关键词都包含于一个有序的词典中，它们的相对位置是确定的，因此可以比较大小。每一层的 k-node 中包含了其叶子结点对应的查询项的第结点层次个关键词的划分（cuts）。运用空间面积分割方法的结点称为 s-node。每一层的 s-node 中包含了其叶子结点对应的查询项的面积划分（cells）。如果一个查询项没有足够的关键词，无法寻找到一个对应的 cut，就用一个 dummy cut 保存；如果一个查询项的覆盖面积已经包含一个 s-node 的覆盖面积，就用一个 dummy cell 保存。

在匹配对象时，使用深度优先搜索法。如果当前结点是 q-node，则判断该对象是否与 q-node 中的查询项匹配，如果匹配则可等待输出。如果当前结点是 s-node，则访问它的 cell 和 dummy cell。如果当前结点是 k-node，则访问它的 cut 和 dummy cut。

AP 树还提供了一个计算两种分割匹配算法的代价的模型。匹配代价 $C(P)$ 的定义为：

$$C(P) = \sum_{i=1}^{f} \mathbf{w}(B_i) \times \mathbf{p}(B_i)$$

其中 B 是一种分割方式，$\mathbf{w}(B)$ 是与 B 有关的查询项的个数，$\mathbf{p}(B)$ 是 B 的击中概率，即在对象匹配过程中 B 被访问的可能性。在关键词分割法中，$\mathbf{p}(B) = \sum_{w \in B} \mathbf{p}(w)$，其中 $\mathbf{p}(w) = \frac{freq(w)}{\sum_{w \in P} freq(w)}$，

$freq(w)$ 为关键词 w 在所有查询项中出现的频率。在空间分割法中 $\mathbf{p}(B) = \frac{Area(B)}{Area(N)}$，$Area(B)$ 是 B 的面积，$Area(N)$ 是结点 N 的区域面积。接着使用启发式算法寻找使得匹配代价最小的关键词的分割和空间的分割。

AP 树还提供了建立索引的方法。使用两个标志 kP 和 sP 来表示所有的查询项是否可以进一步地进行关键词的分割和空间的分割，并通过计算两种分割方式的代价 C_k 和 C_s 确定当前结点使用哪种分割方式。

2 代码实现

2.1 接口设计

首先定义所涉及的两个结构体：空间文本对象 `STObject` 以及查询项 `Query`。`STObject` 有两个成员：`Pointf` 类型的位置 `location` 以及 `std::set<std::string>` 类型的关键词集合 `keywords`。`STObject` 也有两个成员：、`Boundf` 类型的方形区域 `region` 和 `std::set<std::string>` 类型的关键词集合 `keywords`，其中 `Boundf` 为由两个 `Pointf` 确定的轴对齐包围区域。类的使用者需要通过这两个结构来和类进行交互。

本文所实现的 AP 树，是一种空间文本分割树，定义抽象的空间文本分割树类 `STTree`，其有一个纯虚函数 `std::vector<Query> Match(const STObject &) const`。用户通过输入 `STObject`，可获得查询项，存放于一个 `std::vector` 中。AP 树类 `APTree` 继承自 `STTree`。`APTree` 的构造函数声明为 `APTree(const std::vector<std::string> &vocab, const std::vector<Query> &queries, size_t nCuts, size_t threshold)`，其中 `vocab` 为所有的关键词，`queries` 为需要注册的查询项，`nCuts` 即论文中的 f，为每个 k-node 或 s-node 的建议分割数目，`threshold` 即论文中的 θ_q，为每个 q-node 的最大查询项容量。

2.2 存储实现

由于 AP 树存储的数据体量大，树的构建和查询算法较为复杂，为了能够发挥 AP 树的优势，在内部存储方式的实现上需要一定的构思。

首先介绍 AP 树内部对于空间文本对象和查询项的存储。在 `APTree` 内部，使用 `STObjectNested` 和 `QueryNested` 来分别表示这两种结构。在两种结构中，位置信息，即 `location` 和 `region` 保持原样，而 `keywords` 变为 `std::vector<size_t>` 类型，其中 `std::vector` 所存储的是该关键词在关键词词汇 `vocab` 中的索引。考虑到字符串的比较较为耗时，而整数的比较效率较高，在建树和查找的过程中可以直接对序号进行操作，而非对字符串。同时，由于 AP 树算法中有对 `keywords` 随机访问的需求，应该将其从原来的 `std::set` 转为 `std::vector` 以实现高效的随机访问。

然后介绍关键词分割方案和空间分割方案的存储表示。关键词分割方案用 `KeywordPartition` 结构体表示，其成员包括：类型为 `std::vector<KeywordCut>` 的分割序列 `cuts`，其中 `KeywordCut` 为关键词序号的闭区间；类型为 `std::map<KeywordCut, std::vector<QueryNested *>>` 的查询项查找表

queries，用来表示每个关键词闭区间对应哪些查询项，注意到这里使用的是 QueryNested 的指针类型，这是为了节约运行时内存占用，避免 QueryNested 的多次复制；类型为 std::vector<QueryNested *> 的 dummy cut 表 dummy；还有一个表示该分割方案代价的浮点数 cost。空间分割方案用 SpatialPartition 结构体表示，其成员包括：类型为 size_t，在 X、Y 轴上个各自的分割数目 nPartX、nPartY；类型为 std::vector<double> 的分割位置 partX，partY，注意这两个 std::vector 的 size() 各自比 nPartX，nPartY 多一个，为了代码清晰，不使用它们的 size() 信息；类型为 std::unique_ptr <std::vector<QueryNested *>> 的单元格查询项序列 cells，注意这是一个一维的数组，一个二维的单元格索引需要通过运算转为一维的数组索引；剩下的 dummy 和 cost 与 KeywordPartition 相同，不作赘述。

2.3 运算实现

3 性能测试

4 评价

4.1 优势

4.2 不足

5 未来工作

A 源代码

B 数据集来源

C 分工

- **代码实现：**王梓涵
- **测试：**刘权
- **报告撰写：**周昕逸　王梓涵　刘权