roadRank设计

**概述**

roadRank是基于集体动态连续博弈矩阵的实时路况预测系统。理论上我们可以通过动态矩阵来描述整个路网中位置状态随时间的变化关系。

在程序上，我们将这个矩阵描述成一个有向无环图（DAG）。根据当前采集的GPS信息和地理环境信息，也就是当前的观测变量，然后利用概率建立一种随机型的时序模型，即概率转移矩阵。然后通过迭代，即可进行趋势分析与预测。

对于一个有向图，在计算机的世界就是一个庞大的二维数组：

a00, a01, …... a0j

a10, a11, …… a1

………

ai0, ai1, …… aij

A =

那么，aij 表示从i节点到j节点有一条道路，并且aij存储的是这条道路的静态属性和动态属性。静态属性包括道路的本身属性和静态环境数据，动态属性包括当前道路上行驶的车辆采集的数据，以及随时间变化的环境数据。

在完成概率矩阵的计算之前，首先需要解决的是存储资源消耗问题。在传统的图论计算中，一般会使用邻接矩阵或者邻接表来存储。但是对于目前全国1500万条道路，即使不考虑迭代产生的中间矩阵，也无法在单机上构建这个巨型有向图。

另外，我们还需要考虑计算的时间成本，这直接关系到数据的实时性。目前矩阵乘法最好的时间复杂度是O(nlog7) ,大概是O(n^2.81)。接近立方级的时间复杂度，加上多次迭代，很难保证实时性。

基于这些计算问题，我们希望通过同步并行计算模型。

同步并行计算模型在一次计算过程中由一系列的超步组成，每个超步由并发计算、通信和数据同步三个步骤组成。

该模型首先解决了数据的存储问题。对于这个巨型图，采用分布式的方式存储，但是还必须保证图的完整性，因为图数据的顶点和边之间是强耦合的。所以，在roadRank的计算中，采用顶点分割的方式来对图进行存储，保证任意一条边（属性数据，原始顶点和目标顶点）只被分配到一台机器上。对于每个顶点的存储，有一个Master顶点，其他为该Master顶点的镜像。Master顶点保存所有镜像顶点的地址信息，以便后续计算进行通信和数据同步。

有了图的分布式存储，就可将roadRank的计算转换成矩阵的并行化问题。

并行同步计算模型通过Master节点和Worker节点协调完成。所有的Worker之间同步执行，Master节点通过超步来协调工作。每个Worker有自己的消息队列，超步开始，会从消息队列读取消息，并完成特定业务的计算，并读取上一次迭代中其他Worker发送给该Worker的消息，并在本次迭代完成后，根据业务，将消息发送给特定Worker。而Master负责超步的同步，在每次超步开始时，汇总所有的Worker运行结束时发送的消息。直到某次超步之后，所有的Worker状态都标记为结束，迭代结束。

上面说到并行同步计算模型每次的超步都会因为消息的传递带来很大的网络开销，这也必然会影响计算性能，而且容错性不高。所以这里的消息传递只是数据计算的一个内存抽象，这个抽象数据集有足够的信息描述如何生成需要计算的数据，而不是数据本身。我们可以看做是分布式环境下的指针。因为roadRank计算中，迭代计算的过程中需要多次访问相同的数据集，通过分布式数据集指针，只需要在真正需要数据的时候将数据分发到不同的机器上。

通过这样的方式对图的计算，最终在逻辑上，等价于一系列的数据集的转换过程。数据只需要一份物理存储。

而且在roadRank的计算场景中，数据遵循幂律分布，也就是不同道路上的数据更新频率相差悬殊。

**图的存储**

1.邻接矩阵

邻接矩阵是直接利用一个二维数组对边的关系进行存储，矩阵的第i行第j列的值 表示 i -> j 这条边的权值；特殊的，如果不存在这条边，用一个特殊标记来表示；如果i == j，则权值为0。它的优点是实现非常简单，而且很容易理解；缺点也很明显，如果这个图是一个非常稀疏的图，图中边很少，但是点很多，就会造成非常大的内存浪费，点数过大的时候根本就无法存储。

2.邻接表

邻接表是图中常用的存储结构之一，每个顶点都有一个链表，这个链表的数据表示和当前顶点直接相邻的顶点。邻接表的优点是对于稀疏图不会有数据浪费，缺点就是实现相对麻烦，而且对于roadRank的计算，我们需要频繁的得到每条道路的进入道路和出去道路，也就是出度信息和入度信息，而邻接表查找比较麻烦。

3.链式前向星

链式前向星和邻接表类似，也是链式结构和线性结构的结合，每个结点i都有一个链表，链表的所有数据是从i出发的所有边的集合（对比邻接表存的是顶点集合），边的表示为一个四元组(u, v, w, next)，其中(u, v)代表该条边的有向顶点对，w代表边上的属性，next指向下一条边。  
具体的，我们需要一个边的结构体数组 edge[MAXM]，MAXM表示边的总数，所有边都存储在这个结构体数组中，并且用head[i]来指向 i 结点的第一条边。

边的结构体声明如下：

|  |  |
| --- | --- |
|  | struct EDGE {      int u, v, w, next;  }edge[MAXM]; |

初始化所有的head[i] = INF，当前边总数 edgeCount = 0  
每读入一条边，调用addEdge(u, v, w)，具体函数的实现如下：

|  |  |
| --- | --- |
|  | void addEdge(int u, int v, int w) {      edge[ edgeCount ] = EDGE(u, v, w, head[u]);      head[u] = edgeCount ++;  } |

这个函数的含义是每加入一条边(u, v)，就在原有的链表结构的首部插入这条边，使得每次插入的时间复杂度为O(1)，所以链表的边的顺序和读入顺序正好是逆序的。这种结构在无论是稠密的还是稀疏的图上都有非常好的表现，空间上没有浪费，时间上也是最小开销。

调用的时候只要通过head[i]就能访问到由 i 出发的第一条边的编号，通过编号到edge数组进行索引可以得到边的具体信息，然后根据这条边的next域可以得到第二条边的编号，以此类推，直到next域为INF（这里的INF即head数组初始化的那个值，一般取-1即可）。

另外，我们还可以在head[i]中记录从i出去的有多少条边。

因为我们在前面说到，为了保证图的完整性，需要对图进行点切分，保证不同的边被分配到不同的机器上处理，链式前向星的存储方式在存储和处理上更具有优势。

**数据加载**

在roadRank计算程序启动的时候，我们首先要加载道路拓扑关系到内存中，因为采取的是分片存储路网，所以拓扑关系加载的过程中也只需要加载对应片区的即可。另外，还需要加载静态地图数据，包括道路属性数据等。

**rank值计算**

roadRank的计算主要关注的是道路上的速度，而对于不同等级、不同限速的道路上的速度，无法直接进行对比，所以我们以道路的通行率作为评判准则。道路通行率R=V/V限速, 也就是通过道路上的行驶速度与该道路的限速的比值作为道路的通行率。

对于未来某一时刻的速度，它是由当前道路上的速度转移过去的；同样，对于下一条路上的行驶状态，也是由上一条道路转移过来的。所以，我们将整个路网中的状态转换看做是马尔科夫模型，它在时间和空间上是连续变化的，并且它的下一个状态仅由当前状态转换过去的。

计算出的rank值是从0到1闭区间上的一个有理数，rank值越接近1，说明通行率越高，道路越畅通，反之，道路越拥堵。

为了达到路况预测的目的，我们需要根据已知的信息制定出路况分流策略，这个就需要根据之前加载的拓扑关系，得到当前道路上的路况是由哪些道路汇聚而来，又会分流到哪些道路上去。同时，我们需要根据道路等级和限速信息来制定不同道路上分流的权重w。另外，根据当前道路上的拥堵情况，还需要动态调整这个分流策略的权重w。这样，就构建了分流策略矩阵P。

在进行分流的过程中，下一时刻的路况来源有两种可能性，一部分是源自本身的路况，另一部分是从连接到该条道路的所有上一条道路分流而来。所以下一时刻的分流矩阵可以表示为：

Q= mE + (1- m)P。P就是上面提到的分流策略矩阵，m是路况留在本身上的概率， E是单位矩阵。

基于马尔科夫的路况预测，其实就是求解概率转移矩阵的过程。下一时刻的概率矩阵Q(i+1) = AQ(i)， 其中A就是我们前面链式前向星加载的有向图，Q(i)为第i次概率转移矩阵的求解值。对于这样的一个迭代过程，概率矩阵必然会收敛到一个误差e范围内，即本次概率矩阵与上一次的矩阵之差小于e,则算法结束。

**roadRank中的博弈模型计算**

roadRank研究的是复杂条件下的路况，需要结合地理环境，道路属性，当前行驶状态等因素进行预测，本质上是多源信息的博弈融合。我们可以看做是由多个传感器信息在冲突环境下的策略交互过程。在多源信息存在冲突的环境下，信息融合系统包含着大量的不确定性，所以我们还必须得根据这些不确定性进行推理，以达到系统的决策与控制。

针对交通车流量、车速、天气、时段等多元动态参数,以及静态地理环境数据，构建每条道路上合适的博弈融合连通图。 每条道路上的构建的连通图可以看做是整个路网图的子图。

需要注意的是，多源信息组成的多连通图会导致信息在无向环路中循环传递而无法进入稳定状态，所以，对于多连通图，应该转换成多个单连通图的叠加。

设融合系统中多源信息为I=(S1,S2,…,Sk),博弈交互决策结果记为集F, 要判断信息S1,S2,…,Sk中对决策是否有影响,只需网络结构中判断由S1,S2,…,Sk代表的节点S1,S2,…,Sk是否存在可达集合F中节点的路径.

假设判断结果记为集合G,待判断节点集合为T,则算法的描述如下:

1. 初始化集合T=I,G=NULL;
2. 若集合T=NULL,则算法退出,结果为集合G;否则从T中任取出一个节点S作为当前判断节点;
3. 按照贝叶斯网络结构,生成节点S的所有子节点,若S没有子节点则算法转（2）,否则记S的子节点为Sci,i=1,2,…,n;
4. 若存在Sci∈F,则将S放入集合G中,并转第（2）步;
5. 对于任意的Sci,按照S的概率值和条件概率表计算其概率值P(Sci),并判断是否大于预先给定的阀值ε,若是则将Sci放入集合T中,算法转第（2）步.

在该算法中,判断的目的在于搜索对决策有影响节点的同时,尽可能寻找对融合决策有较高概率影响的节点,因而在算法的第（5）步中,利用一个事先给定的阈值,对概率小于预先给定值的节点进行剪枝。

求解的核心问题等价于求解P(S1,S2,…,Sk|Fi),实际上可以看作是概率密度函数的估计问题，在估计P(S1,S2,…,Sk|Fi)的值后,以最大者作为博弈的最终结果。

**同步并行计算的消息传递机制**

roadRank计算的过程是通过并行计算处理的，迭代和同步的过程中存在大量的通信。如果使用多线程状态共享机制去实现，性能会比较糟糕，并不能满足我们的实时性要求。这里，我们参考Erlang语言中的并发通信模型Actor Model。Actor模式是一个解决分布式计算的数学模型。Actor模型=数据+行为+消息。 Actor模型内部的状态由自己的行为维护，外部的线程不能直接调用对象的行为，必须通过消息才能激发行为。Actor通过异步事件驱动的方式发送和处理消息。