基于最短路算法的随机路径分析法

摘要

本文主要研究不同参数小区封闭与开放对周边道路交通的影响，从道路通行能力与出行效率角度出发建立道路通行综合评价体系，并利用基于流体力学一维流动运动微分方程的交通流理论模型，通过最短路算法的随机路径分析法定量比较各类型小区开放前后对道路通行的影响，根据结果向城市规划和交通管理部门提出关于小区开放的合理化的建议。

针对第一问，主要从道路通行能力和出行效率两个方面来进行评估，同时考虑道路服务水平和利用率两方面的综合性能，既考虑路网实际的承载能力，又考虑路网的容量的利用，程度，为第三问的影响分析提供合理的评价体系。

针对第二问，基于路网假设，为定量研究某路段上车辆行驶速度与车辆数的关系，我们采用交通流理论模型，借用速度-密度公式，计算服务水平通常为断面所在路段的平均行程车速 v，并以此计算路网运行效率，同时计算居民出行实际时间，为第一问的评价体系进一步完善。

针对第三问，首先对无小区情况进行十万次独立随机实验，将得到的道路通行能力作为标准，将小区由形状是否规则及大小分为四类，利用最短路算法的随机路径分析法，评估道路通行能力。

针对第四问，向城市规划部门提出建议尽可能修建形状规则、面积较小的小区，尽可能修建开放式小区，可以在一定程度上缓解道路交通压力。

关键词：最短路算法 随机路径分析法 交通流理论模型 网络流算法

一、问题重述

1.1 问题背景

近日，中国公布一份城市建设规划文件（全称为《中共中央国务院关于进一步加强城市规划建设管理工作的若干意见》），要求“新建住宅要推广街区制，原则上不再建设封闭住宅小区，已建成的住宅小区和单位大院要逐步打开。”这一规定受到人们就科学城市规划、物权产权、社会公平等问题的争议。

有人认为封闭式小区破坏了城市路网结构，堵塞了城市“毛细血管”，容易造成交通阻塞。因此小区开放后，路网密度提高，道路面积增加，通行能力自然会有提升。也有人认为这与小区面积、位置、外部及内部道路状况等诸多因素有关，不能一概认为开放小区更有利于提高通行能力。还有人认为小区开放后，虽然可通行道路增多了，相应地，小区周边主路上进出小区的交叉路口的车辆也会增多，也可能会影响主路的通行速度。

1.2 问题重述

第一问选取合适的评价指标体系用于评价小区开放对周边道路通行状况，第二问建立关于车辆通行的数学模型，用以研究小区开放对周边道路通行的影响。第三问从小区面积、位置、外部及内部道路状况等因素出发选取或构建不同类型的小区，应用建立的模型，定量比较各类型小区开放前后对道路通行的影响。第四问根据研究结果，从交通通行的角度，向城市规划和交通管理部门提出关于小区开放的合理化建议

二、问题分析

2.1 第一问

为建立合适的评价体系以评价小区开放对周边道路的影响，主要从道路通行能力和出行效率两个方面来进行评估。开放小区之后，为避免造成资源浪费，需要考虑新增道路的利用率问题，建立综合考虑路网实际的承载能力与路网的容量的利用程度的评价体系。

2.2 第二问

假设出行人员会优先选择总长度较短的路线，为定量研究某路段上车辆行驶速度与车辆数的关系，我们利用交通流理论模型，得到的相应于流体力学一维流动运动微分方程的交通流基本方程。进一步借用公式计算服务水平通常为断面所在路段的平均行程车速，并以此计算路网运行效率。

2.3 第三问

为具体探究小区结构对道路通行的影响，首先对无小区情况进行十万次独立随机实验，将得到的道路通行能力作为标准，进行对照。将小区由形状是否规则、面积大小分类，利用最短路随机路径分析法定量描述道路通行能力。

2.4 第四问

基于第三问结论，对小区规划及道路建设提出意见。

三、基本假设：

1. 居住小区为随机方形小区；
2. 小区外部城市街道为1-4车道，小区内部道路为1-2车道，街道均为方格网络，即沿南北东西方向按一定间距平行排列的干路与支路。
3. 交叉路口直行及左转弯的绿信比以及遭遇红灯的等待时间期望仅与相交叉两路的宽度有关。
4. 研究区域与外界接触路段等概率驶入车辆。
5. 出行人员会优先选择总长度较短的路线。

四、参数：

道路断面需求	q
道路通行能力	c

路段平均车速	v
平均车密度	k
路面输送效率	E_L
路段在时刻 t 的车辆数	n
路段长度	L
出行目标集合	$\mathcal{R}$
交通压力	p
粘性阻力	τ_w
自由流速度	v_f
自由流行驶最大密度	k_f
最大波速	mv_f
波速系数	m
速度为 0 的阻塞密度	k_j

五、评价体系建立

为了评价小区开放对周边道路通行的影响，我们主要从道路通行能力和出行效率两个方面来进行评估。

一般来说，定义道路断面交通需求 q 为单位时间内通过的车辆数。道路通行能力 c 为单位时间内可能通过的最大车辆数，服务水平通常用断面所在路段的平均行程车速 v 或平均车辆密度 k 表示。

开放小区之后，由于道路网络密度增加，区域的能承载的车辆数也会随之增加。为避免造成资源浪费，我们需要考虑新增道路的利用率问题。为了同时考虑道路服务水平和利用率两方面的综合性能，我们采取如下指标。

路面输送效率[1]是指单位时间内路段上车辆完成的总公里数，其定义为：

$$E_L = qLv$$

若定义 n 为路段在时刻 t 的车辆数，根据$q = kv$，$n = kL$，则有$E_L = nv^2$。也就是说，这个值实际上是路段上的车辆数与平均行程车速平方的乘积。

由于我们研究对象为区域道路网络，因此为了评估网络整体的运输效率，引入路网运行效率这一指标。路网运行效率实际上是路网各条路段的输送效率之和，

[1]

其定义为：

$$E_N = \sum_{i=1}^{N} E_{L_i}$$

通过这一指标，我们既可以考虑路网实际的承载能力，又可以考虑路网的容量的利用程度。

另一方面，我们从出行人员的角度，研究开放小区前后的出行效率。

为此，我们定义出行效率如下

$$A = \frac{\sum_{a,b\in\mathcal{O}} p_{ab} \cdot t_{ab}^{0}}{\sum_{a,b\in\mathcal{O}} p_{ab} \cdot t_{ab}}$$

其中$\mathcal{O}$为路网出入口的集合，m、n为任意两个出入口，p_{mn}为出行人员想要从 m到达 n的概率，t_{mn}^{0}为从 m 到达 n 所需的最短时间，即在完全没有遇到交通堵塞情况下的时间，t_{mn}为实际出行所用的时间。由于实际出行时间不恒定，与当时路网中整体情况有关，我们在实际评价时采用了如下的量：

$$B = \frac{\sum_{\alpha\in\mathcal{R}} t_{\alpha}^{0}}{\sum_{\alpha\in\mathcal{R}} t_{\alpha}}$$

其中，$\mathcal{R}$为随机模拟产生的出行目标集合，α为集合中某个出行目标，t_{α}^{0}为实现该目标的最短时间，t_{α}为模拟时所用的时间。这一指标越大，则表示出行人员浪费的时间越少，即出行效率越大。

六、模型建立：

假设小区周围及内部道路为如图所示的方形网络，小区外部的道路为 1-4 车道，小区内部道路为 1-2 车道。

假设出行人员会优先选择总长度较短的路线。某路段上车辆行驶速度受路段限速限制，并与该路段上车辆数相关，由经验知，车辆总数越多，车辆行驶速度越慢。为了方便计算，我们用某路段被选择的次数来代替实际该路段上车辆数 n。

为了定量研究某路段上车辆行驶速度与车辆数的关系，我们采用交通流理论模型，即认为交通流为连续流，得到的相应于流体力学一维流动运动微分方程的交通流基本方程[1]：

连续方程：

$$\frac{\partial k}{\partial t} + \frac{\partial (kv)}{\partial x} = 0$$

其中:t 为时间；x 为沿车流方向的距离；k 为密度;v 为速度

动量方程：

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+\frac{1}{k}\frac{\partial p}{\partial x}+\tau_w=0$$

式中：p 为交通压力，τ_w为粘性阻力

对于非粘性定常交通流，我们有：

$$p=p_1+q(v_1-v)$$

式中：q 为琉璃；下标 1 表示某个已知状态

当交通流的密度增加到一定值,相应的速度降低到一定值时,呈现出粘性特点,对于粘性阻力可采用如下假设

$$\tau_w=\begin{cases}0 & v>v_1,k<k_1\\ -2\dfrac{v_1-v}{k}\dfrac{\partial q}{\partial x} & v<v_1,k>k_1\end{cases}$$

由以上模型可以得到交通流的速度－密度特性：

$$v=\begin{cases}v_f\left[m+(1-m)\dfrac{k_f}{k}\right] & k_f\le k<\dfrac{1}{4}k_j-\dfrac{1-m}{m}k_f\\ \dfrac{1}{4}mv_f\dfrac{k_j}{k} & \dfrac{1}{4}k_j-\dfrac{1-m}{m}k_f\le k\le\dfrac{1}{2}k_j\\ mv_f\left(1-\dfrac{k}{k_j}\right) & \dfrac{1}{2}k_j<k\le k_j\end{cases}$$

以及流量－密度特性：

$$q=\begin{cases}v_f[mk+(1-m)k_f] & k_f\le k<\dfrac{1}{4}\ k_j-\dfrac{1-m}{m}k_f\\ \dfrac{1}{4}mv_fk_j & \dfrac{1}{4}k_j-\dfrac{1-m}{m}k_f\le k\le\dfrac{1}{2}k_j\\ mv_f\left(k-\dfrac{k^2}{k_j}\right) & \dfrac{1}{2}k_j<k\le k_j\end{cases}$$

式中：v_f为自由流速度，取道路的设计时速；k_f为以自由流速度行驶时的最大密度，可由实测数据回归得到。

$$v=\begin{cases}20+\dfrac{20k_f}{k} & k_f\le k<27.8-k_f\\ \dfrac{555.5}{k} & 27.8-k_f\le k\le 55.6\\ 0.5v_f\left(1-\dfrac{k}{111.1}\right) & 55.6<k\le 111.1\end{cases}$$

借用速度-密度公式，我们便可以计算服务水平通常为断面所在路段的平均行程车速 v，并以此计算路网运行效率。

同时我们也可以计算居民出行实际时间，假设某人选择的路线共经历 p 个路段，p-1 个交叉路口，则该人出行的实际时间为

$$t_\alpha = \sum_{k=1}^{p} \frac{L_p}{v_p} + \sum_{k=1}^{p-1} P_k \ T_k$$

其中，P_k为遇到红灯的概率，T_k为遭遇红灯的等待时间期望。

七、不同类型小区开放前后对道路通行的影响

首先对无小区情况进行十万次独立随机实验，将得到的道路通行能力作为标准，记作 100%，堵车概率为 2.13%。

将小区按形状是否规则，面积大小分为四类。其中形状规则是指小区为长方形或正方形，不规则是指小区为 U 字型，面积小是指小区占 6 个格子，面积大指小区占 16 个格子。

7.1. 形状规则面积较小的小区

在网格图中添加此类小区之后，经过十万次独立随机实验，道路通行能力是无小区情况的 91.11%， 堵车概率为 6.93%。

在开放此类小区后，经过十万次独立随机实验，道路通行能力是无小区情况的 96.81%，堵车概率为 4.47%。

7.2. 形状规则面积较大的小区

在网格图中添加此类小区之后，经过十万次独立随机实验，道路通行能力是无小区情况的 71.25%， 堵车概率为 27.45%。

在开放此类小区后，经过十万次独立随机实验，道路通行能力是无小区情况的 79.48%，堵车概率为 19.33%。

7.3. 形状不规则面积较小的小区

在网格图中添加此类小区之后，经过十万次独立随机实验，道路通行能力是无小区情况的 76.48%， 堵车概率为 16.40%。

在开放此类小区后，经过十万次独立随机实验，道路通行能力是无小区情况的 84.12%，堵车概率为 11.94%。

7.4.形状不规则面积较大的小区

在网格图中添加此类小区之后，经过十万次独立随机实验，道路通行能力是无小区情况的 56.35%， 堵车概率为 48.64%。

在开放此类小区后，经过十万次独立随机实验，道路通行能力是无小区情况的 67.47%，堵车概率为 42.28%。

7.5.结论

综上所述，四种类型小区影响道路通行能力的严重程度排序为：形状不规则面积较大的小区>形状规则面积较大的小区>形状不规则面积较小的小区>形状规则面积较小的小区。且每类小区开放后都能在一定程度上缓解道路交通压力。

八、合理化建议

8.1.

尽可能修建形状规则、面积较小的小区，这样对道路交通影响较小。

8.2.

尽可能修建开放式小区，可以在一定程度上缓解道路交通压力。

九、参考文献

[1] 叶彭姚, 陈小鸿. 基于交通效率的城市最佳路网密度研究[J]. 中国公路学报, 2008, 21(4): 94-98.

十、附录

9.1.最短路算法（Dijkstra 算法）

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<vector>
#include<queue>
#define MAXN 100000
#define INF 0x3f3f3f3f
using namespace std;
struct Edge{
    int from,to,dist;
};
struct HeapNode{
    int d,u;
    bool operator < (const HeapNode &rhs) const{
        return d>rhs.d;
    }
};
vector<int> G[MAXN];
vector<Edge> edges;
```

```
bool flag[MAXN];
int d[MAXN],n,m;
void AddEdge(int from,int to,int dist){
    edges.push_back((Edge){from,to,dist});
    G[from].push_back(edges.size()-1);
}
void init(){
    for(int i=0;i<MAXN;++i) G[i].clear();
    edges.clear();
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i){
        int x,y,w;
        scanf("%d%d%d",&x,&y,&w);
        AddEdge(x,y,w);
    }
}
void dijkstra(int s){
    priority_queue<HeapNode> Q;
    memset(d,INF,sizeof(d));
    d[s]=0;
    memset(flag,0,sizeof(flag));
    Q.push((HeapNode){0,s});
    while(!Q.empty()){
        HeapNode x=Q.top();Q.pop();
        int u=x.u;
        if (flag[u]) continue;
        flag[u]=1;
        for(int i=0;i<G[u].size();++i){
            Edge &e=edges[G[u][i]];
            if (d[e.to]>d[u]+e.dist){
                d[e.to]=d[u]+e.dist;
                Q.push((HeapNode){d[e.to],e.to});
```

```
            }
        }
    }
}
int main(){
    init();
    dijkstra(1);
    for(int i=1;i<=n;++i) printf("%d ",d[i]);
    return 0;
}
```

9.2.网络流算法（SAP 算法）

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<cstring>
#include<algorithm>
#include<vector>
#include<queue>
#define MAXN 110000
#define INF 1000000000
using namespace std;
struct Edge{
    int from,to,cap,flow;
};
int n,m,s,t;
vector<Edge> edges;
vector<int> G[MAXN];
bool vis[MAXN];
```

```
int d[MAXN],cur[MAXN],p[MAXN],num[MAXN];
void AddEdge(int from,int to,int cap){
    edges.push_back((Edge){from,to,cap,0});
    edges.push_back((Edge){to,from,0,0});
    m=edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
}
void init(){
    int M;
    scanf("%d%d",&n,&M);
    for (int i=1;i<=M;++i){
        int x,y,w;
        scanf("%d%d%d",&x,&y,&w);
        AddEdge(x,y,w);
    }
}
void BFS(){
    memset(vis,0,sizeof(vis));
    queue<int> Q;
    Q.push(t);
    memset(d,0x7f7f7f7f,sizeof(d));
    d[t]=0;
    memset(num,0,sizeof(0));
    num[0]=1;
    vis[t]=1;
    while (!Q.empty()){
        int x=Q.front();
        Q.pop();
        for (int i=0;i<G[x].size();++i){
            Edge e=edges[G[x][i]];
            if (G[x][i]%2==1&&!vis[e.to]){
```

```
                Q.push(e.to);
                num[d[e.to]=d[x]+1]++;
                vis[e.to]=1;
            }
        }
    }
}
int Augment(){
    int x=t,a=INF;
    while (x!=s){
        Edge e=edges[p[x]];
        a=min(a,e.cap-e.flow);
        x=e.from;
    }
    x=t;
    while (x!=s){
        edges[p[x]].flow+=a;
        edges[p[x]^1].flow-=a;
        x=edges[p[x]].from;
    }
    return a;
}
int Maxflow(int ss,int tt){
    s=ss;t=tt;
    int flow=0;
    BFS();
    memset(p,0,sizeof(p));
    int x=s;
    memset(cur,0,sizeof(cur));
    while(d[s]<n){
        if (x==t){
            flow+=Augment();
```

```
            x=s;
        }
        int ok=0;
        for (int i=cur[x];i<G[x].size();++i){
            Edge e=edges[G[x][i]];
            if (e.cap>e.flow&&d[x]==d[e.to]+1){
                ok=1;
                p[e.to]=G[x][i];
                cur[x]=i;
                x=e.to;
                break;
            }
        }
        if (!ok){
            int m=n-1;
            for (int i=0;i<G[x].size();++i){
                Edge e=edges[G[x][i]];
                if (e.cap>e.flow) m=min(m,d[e.to]);
            }
            if (--num[d[x]]==0) break;
            num[d[x]=m+1]++;
            cur[x]=0;
            if (x!=s) x=edges[p[x]].from;
        }
    }
    return flow;
}
int main(){
    init();
    printf("%d\n",Maxflow(1,n));
    return 0;
}
```