最小圆覆盖算法复杂度分析与效率评估

1. 问题背景

1.1 问题描述

无线网络基站在理想状况下有效信号覆盖范围是个圆形，而无线基站的功耗与圆的半径的平方成正比。现给出平面上若干网络用户的位置，请你选择一个合适的位置建设无线基站....就在你拿起键盘准备开始敲代码的时候，你的好朋友发明家 WYH 突然出现了。WYH 刚刚完成了他的新发明——无线信号增幅仪。增幅仪能够在不增加无线基站功耗的前提下，使得有效信号的覆盖范围在某一特定方向上伸长若干倍。即:使用了增幅仪的无线基站覆盖范围是个椭圆，其功耗正比于半短轴长的平方。现给出平面上若干网络用户的位置，请你选择一个合适的位置建设无线基站，并在增幅仪的帮助下使所有的用户都能接收到信号，且无线基站的功耗最小。注意:由于WYH 增幅仪的工作原理依赖地磁场，增幅的方向是恒定的。

1.2 问题简化

在一个二维平面上给定 N个点，请你画出一个最小的能够包含所有点的圆。圆的边上的点视作在圆的内部。即我们需要找到一个半径最小的圆使得能够覆盖平面上的所有点。

2. 算法描述

2.1 引理1

引理1: 对于一个平面点集，最小覆盖圆是唯一的。

证明：对于一个平面点集，若拥有两个或者两个以上的最小覆盖圆，则对于不同的两个圆心O和O'，都有$\circ O$和$\circ O'$能覆盖所有圆。又由于最小覆盖圆的边上至少有三个点，因此必然O和O'在某两个点构成的线段的垂直平分线上，则对于第三个点Q，由于$O \neq O'$，因此$OQ \neq O'Q$，得证。

2.2 引理2

引理2：对于目前给定的平面点集V，得到目前已知的最小覆盖圆O。对于新加进点集的点p构造出的新的最小覆盖圆O'，若p不在V的最小覆盖圆O内部，则p一定在$\{p\} \cup V$的最小覆盖圆O'的边上。

2.3 算法内容

> 1.将所有点随机化
>
> 2.`for(int i=2;i<=n;++i)`枚举每一个点，在点集中加入新点`p[i]`:
>
> > 1.在当前的最小覆盖原内的话直接continue
> >
> > 2.如果不在最小覆盖圆内的话，必定在最小覆盖圆的边上
> >
> > 圆<-`p[i]`
> >
> > `for(int j=1;j<i;++j)`
> >
> > `if j not in cir`则`j`一定在`1~j`，`i`这`j+1`个点且`i`在`cir`边上的最小覆盖圆
> >
> > `for(k=1;k<j;++k)`

> > if k not in cir则k一定在圆边上

2.4 最小圆覆盖核心代码

下面展示最小圆覆盖的核心代码

```
random_shuffle(q,q+n);
Cir c{q[0], 0};
for(int i=1;i<n;++i){
    if(sign(c.r-dist(c.p,q[i]))<0){
        c={q[i], 0};
        for(int j=0;j<i;++j){
            if(sign(c.r-dist(c.p,q[j]))<0){
                c={(q[i]+q[j])/2.0, dist(q[i],q[j])/2.0};
                for(int k=0;k<j;++k){
                    if(sign(c.r-dist(c.p,q[k]))<0)
                    c=get_cir(q[i],q[j],q[k]);
                }
            }
        }
    }
}
```

3. 算法分析

我们着重分析这个三层循环:

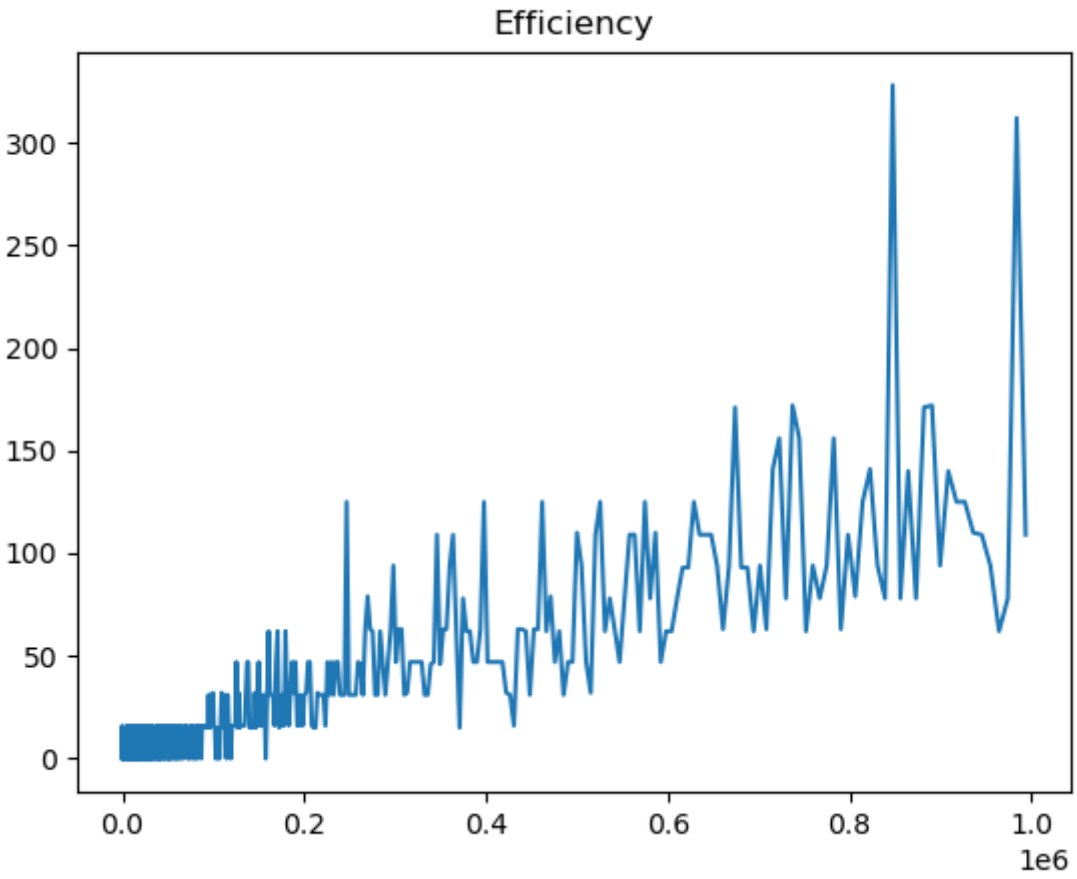
```
for i:1~n:
    if v[i] not in circle:
        for j in 1~i-1:
            if v[j] not in circle:
                for k in 1~j:
                    if v[k] not in circle:
                        c = get_circle(v[i],v[j],v[k])
```

首先，里面一层的复杂度是O(j)的，我们考虑第二层循环，当且仅当v[j]不在前j-1个点和i构成的最小覆盖圆中时才会进入第三层循环，由于点权随机，所以进入第三层循环的期望是$O(\frac{3}{j})$，最外层的循环复杂度是$O(n)$,所以总期望复杂度是$O(n \cdot \frac{3}{j} \cdot j) = O(3n)$的

这样，原本看似是$O(n^3)$的算法通过随机化的方式使得复杂度降低到了$O(n)$的级别。

4. 性能分析

我们选取1e2到1e6的1000种不同规模大小的数据，画出程序效率运行图:



发现在纯随机下的程序运行效率都非常的高，大概是O(n)的效率。

5. 基于规划的最小覆盖算法

上面我们从几何方面讨论了最小覆盖的问题，还可以从规划角度来讨论最小覆盖问题。而且为了在核空间更好地运算，将所有的运算都表示成内积运算。[1]

给定集合$K = \{x_1, x_2 \ldots, x_m\}$,求覆盖K的最小覆盖。求覆盖K的最小覆盖可以转为为以下规划问题:

> 目标函数：$max\{\theta\}$
>
> 约束条件：$< \omega, x_i > -\theta \geq \theta$ (i=1,...,m)
>
> $|w| = R^2$

此优化问题的解由下面的拉格朗日泛函的鞍点给出:

$L(\theta, \lambda, \omega, \alpha) = \theta + \sum_{i=1}^{m} \lambda_i(< \omega, x_i > -\theta) + \alpha(< \omega, \omega > -R^2)$

本篇文章在这里就不做过多赘述。此时转化为规划问题后的时间复杂度是$O(n^2)$的，效率不如几何做法。

6. 参考文献

[1]赵姝,张燕平,张铃,徐峰. 最小覆盖算法[C]//.第二十六届中国控制会议论文集.,2007:1909-1913.

7. 附录代码

```
#include<unordered_set>
#include<unordered_map>
#include<functional>
#include<windows.h>
#include<algorithm>
#include<string.h>
#include<iostream>
#include<iterator>
#include<cstring>
#include<numeric>
#include <ctime>
#include<cstdio>
#include<vector>
#include<bitset>
#include<queue>
#include<stack>
#include<cmath>
#include<set>
#include<map>
#define x first
#define y second
using namespace std;
//================================DEBUG
#define FASTIO cin.tie(nullptr) -> sync_with_stdio(false)
#define debug(a) cout << #a": " << a << endl;
#define rep(i, ll, rr) for(int i = ll; i <= rr; ++ i)
#define per(i, rr, ll) for(int i = rr; i >= ll; -- i)
//================================IO
typedef long long LL; typedef unsigned long long ULL; typedef long double LD;
inline LL read(){LL s=0,w=1;char ch=getchar();for(;!isdigit(ch); ch = getchar())if(ch == '-') w
= -1; for (; isdigit(ch);ch = getchar())s=(s<<1)+(s<<3)+(ch^48);return s*w;}
inline void print(LL x,int op=10){if(!x){putchar('0');if(op)putchar(op);return;}char F[40];LL
tmp=x>0?x:-x;if(x<0)putchar('-');int cnt=0;while(tmp>0)
{F[cnt++]=tmp%10+'0';tmp/=10;}while(cnt>0)putchar(F[--cnt]);if(op)putchar(op);}
inline void print128(__int128_t x){if(x < 0) {putchar('-');x = -x;}if(x/10)
print128(x/10);putchar(x%10+'0');}
template <typename T>void read(T &x){x=0;int f=1;char ch=getchar();while(!isdigit(ch)){if(ch=='-
')f=-1;ch=getchar();}while(isdigit(ch)){x=x*10+(ch^48);ch=getchar();}x*=f;return;}
template <typename T>void write(T x){if(x<0){putchar('-');x=-
x;}if(x>9)write(x/10);putchar(x%10+'0');return;}
//================================HABIT
LL fpower(LL a,LL b,LL mod) {LL ans = 1; while(b){ if(b & 1) ans = ans * (a % mod) % mod; a = a
% mod * (a % mod) % mod; b >>= 1;} return ans; }
LL Mod(LL a,LL mod){return (a%mod+mod)%mod;}
LL gcd(LL a,LL b) {return b?gcd(b,a%b):a;}
int mov[8][2]={1,0,0,1,-1,0,0,-1,1,1,-1,-1,1,-1,-1,1};
//================================DEFINE
// #define int LL
// #define double long double
typedef pair<int,int> pii;
typedef pair<double,double> pdd;
const double eps=1e-9,PI=acos(-1);
//================================GEOMETRY
int sign(double x) {if(fabs(x)<eps) return 0;return x>0?1:-1;}
struct Poi{
```

```
    double x,y;
    Poi operator-(Poi b){return {x-b.x,y-b.y};}
    Poi operator+(Poi b){return {x+b.x,y+b.y};}
    Poi operator*(double k){return {x*k,y*k};}
    Poi operator/(double k){return {x/k,y/k};}
    bool operator==(Poi b){return sign(x-b.x)==0&&sign(y-b.y)==0;}
    bool operator<(Poi b){return sign(x-b.x)<0||(sign(x-b.x)==0&&sign(y-b.y)<0);}
};
double cross(Poi a,Poi b){return a.x*b.y-a.y*b.x;}
double area(Poi a,Poi b,Poi c){return cross({b.x-a.x,b.y-a.y},{c.x-a.x,c.y-a.y});}
double dist(Poi a,Poi b){double dx=a.x-b.x;double dy=a.y-b.y;return sqrt(dx*dx+dy*dy);}
struct Cir{Poi p;double r;};
struct Line{Poi st,ed;};
double get_angle(const Line &a){ //获得直线的角度
    return atan2(a.ed.y-a.st.y,a.ed.x-a.st.x);
}
//直线按照角度的排序函数
bool cmp(Line &a,Line &b){  double A=get_angle(a),B=get_angle(b); if(sign(A-B)==0) return
sign(area(a.st,a.ed,b.ed))<0;return A<B;}
//求直线p+kv和直线q+kw的交点
Poi get_line_intersection(Poi p,Poi v,Poi q,Poi w){ auto u=p-q; double t=cross(w,u)/cross(v,w);
return {p.x+v.x*t,p.y+v.y*t};}
//两条线的交点
Poi get_line_intersection(Line a,Line b){
    return get_line_intersection(a.st,a.ed-a.st,b.st,b.ed-b.st);
}
//bc的交点是否再a的右侧
bool on_right(Line a,Line b,Line c){ auto jiao=get_line_intersection(b,c);return
sign(area(a.st,a.ed,jiao))<=0;}
//将一个点顺时针旋转d度
Poi rotate(Poi a,double b){
    return {a.x*cos(b)+a.y*sin(b),
    -a.x*sin(b)+a.y*cos(b)};
}
//获取中垂线
Line get_perpendicular_bisector(Poi a,Poi b){
    return {(a+b)/2.0,rotate(b-a,PI/2.0)};
}
//三点确定圆
Cir get_cir(Poi a,Poi b,Poi c){
    auto u=get_perpendicular_bisector(a,b),
         v=get_perpendicular_bisector(a,c);
    auto p=get_line_intersection(u.st,u.ed,v.st,v.ed);
    return {p, dist(p,a)};}
/*random_suffle(p+1,p+1+n);  点随机化*/
//================================
const int N=10000010,M=N*2,mod=1e9+7;
int n,m,k,a[N];
Poi p[N];
Cir c[N];
mt19937 rnd(time(0));

void solve(){
    // scanf("%d",&n);
    // rep(i,1,n) scanf("%lf%lf",&p[i].x,&p[i].y);
    random_shuffle(p+1,p+1+n);
    Cir q{p[1],0};
```

```
    rep(i,2,n){
        if(sign(q.r-dist(q.p, p[i]))<0){
            q={p[i], 0};
            for(int j=1;j<i;++j){
                if(sign(q.r-dist(q.p, p[j]))<0){
                    q={(p[i]+p[j])/2.0, dist(p[i], p[j])/2.0};
                    for(int k=1;k<j;++k){
                        if(sign(q.r-dist(q.p, p[k]))<0)
                            q=get_cir(p[i],p[j],p[k]);
                    }
                }
            }
        }
    }
    // printf("%.10lf\n",q.r);
    // printf("%.10lf %.10lf\n",q.p.x,q.p.y);
}

LL getrd(LL a,LL b){
    return (rand()%(b-a+1))+ a;
}

void work(){
    // n=100000;
    clock_t st, ed;
    for(n=100;n<=1000000;n+=n/100){
        for(int i=1;i<=n;++i){
            p[i]={(double)getrd(-1e6,1e6), (double)getrd(-1e9,1e9)};
            // cout << p[i].x << " " << p[i].y << endl;
        }
        st = GetTickCount();
        solve();
        ed = GetTickCount();
        cout << n << " " << ((ed-st)) << endl;
    }
}
//================================
signed main(){
    freopen("out.out","w",stdout);
    int _=1;
    // _=read();
    while(_--)  {
        work();
    }
    return 0;
}
```