不等圆 packing 问题

问题: 对于具有 NP 难度的不等圆 packing 问题，讨论其难解性和可计算性，并为此问题设计一个高效的求解算法。

1 不等圆 packing 问题难解性和可计算性

我们是这样描述不等圆 packing 问题：对于 n 个圆（$r_1, r_2, \cdots, r_n$）是否可以两两不重叠地放进到指定的一个大圆 R 中，且不和 R 重叠，如果可以应该如何放置。

首先我们定义“状态”是 n 个圆在大圆 R 中分布的一种情况，每个状态一一对应着 n 个圆和大圆 R 的圆心位置。当找到一种状态满足 n 个圆和大圆 R 互不重叠且所有 n 个圆都在大圆 R 内，则找到这个 packing 问题的一个解。因为大圆内空间的连续性，n 个不等圆中每个圆圆心在大圆 R 中的位置取值理论上是无限的，加上 n 个圆的排列，要找到满足互不重叠的状态是困难的。下面给出几个求解不等圆 packing 问题的一些解法：

最大穴度算法：[3] 中形式化定义了人工经验方法为最大穴度算法。在不考虑 n 个圆的各种排列情况下首先将第一个圆放进与大圆 R 相切的一个位置，接着把第二个圆放到与前两个圆相切的位置之一，然后把第三个圆也放置到与前三个圆中的两个圆相切的位置之一，就这样将下一个圆放到前 i 个圆中任意两个圆相切的位置，如果这个位置导致了新放入的圆与其他圆有重叠则将新圆放置在满足与其他两个圆相切的另一位置，如果发现所有位置都不满足不重叠的条件，则回溯到之前的一个圆，并重新选取之前圆的位置。按照这样的规则满足条件则继续添加新圆，如果不满足则回溯到上一个圆放置的位置，直到所有的圆都放入到了大圆 R 中，则找到问题的解，或者遍历所有情况仍旧找不到解，则认为该问题没有解。n 个圆的排列复杂度为 A_n^n，将新圆放置在 i 个圆和大圆中有 C_{i+1}^2 个位置，所以总计计算时间为 $A_n^n \cdot \prod_{i=1}^{n} 2 \cdot (1 + C_{i+1}^2)$，即 $A_n^n \cdot 2^n \prod_{i=1}^{n} (1 + C_{i+1}^2)$。

网格算法：在实际编程计算中，网格算法将大圆 R 用间隔为 step 正交的直线进行划分，横向和纵向直线的交点为格点，粗略估计这样的格点有 $N = \frac{R}{step}$ 个。这样就可以通过遍历所有 n 个圆在全部格点的位置找到满足 packing 问题的解，而对于每个圆都有 N^2 个位置选择，那么算法时间为 $(N^2)^n$，即 N^{2n}

拟人拟物算法：这是一个求解不等圆 packing 问题效率较高的方法，具体在 [1] 有详细介绍。拟物算法将大圆 R 看做封闭刚性不可改变空的圆形容器，将每个小圆看做光滑弹性的小

球，开始将这 n 个小球强行放到容器中，我们定义小球之间因为弹性挤压产生的能量为弹性势能（$U = U_1 + U_2 + \cdots + U_n$），接着小球由于相互间以及和容器间的作用力不断运动，每个小球都向受挤压的方向运动，直至所有小球受力到达平衡或者不再受力。拟物方法就是模拟这样一个过程。拟人方法是为了解决拟物方法中可能出现的一种“死锁”的情况，即 n 个小球在局部最小解的情况中反复移动无法跳出的情况（这时的弹性势能 U 不再变化且不为 0），“人为地”从容器中取出最拥挤的小球重新放入容器内进行拟物计算，这样就解决了拟物算法可能遇到的僵持平衡的状态。整个过程直到所有的小球都不受到弹性力，弹性势能为零（$U = 0$）则认为找到了解。

2 高效的不等圆 packing 问题算法实可能会遇到的问题

在 [1, 4] 中给出了拟人拟物算法的实现过程，程序可描述为：

1. 原点为圆心，在大圆 R 中随机选取 n 个点 $(x_1, y_1)\ (x_2, y_2)\ \cdots\ (x_n, y_n)$ 作为初始状态各个小圆圆心位置。令过去势能 $U_{old} = 0$，$t = 0$，$l_0 = 0$，$h = 0$；
2. 如果 $h < 10^{-30}$ 则转 5；
3. 计算 $U(X)$；
4. 若 $U(X) < 10^{-6}$，则成功停机；否则计算 U 的梯度向量 $GradU$；
   - 若 $U(X) < U_{old}$，则 $U_{old} \Leftarrow U(X)$，$X \Leftarrow X - (GradU) \cdot h$，转 2；
   - 若 $U(X) \geq U_{old}$，则 $U_{old} \Leftarrow U(X)$，$h \Leftarrow h \cdot 0.8$，$X \Leftarrow X - (GradU) \cdot h$，转 2；
5. 确认出相对挤压弹性势能最大的圆 l，$1 \leq l \leq M$；
6. 若 $l = l_0$，则 $t \Leftarrow t + 1$；
7. 若 $t < 1$，则在大圆内重新随机地选点 (x_l, y_l)，$l_0 \Leftarrow l$，$h \Leftarrow 1$，转 2；
8. 若 $t = 1$，则确认出绝对挤压弹性势能最小的圆 l，$1 \leq l \leq M$，在大圆内重新随机选择点 (x_l, y_l)，$t \Leftarrow 0$，$l_0 \Leftarrow 0$，$h \Leftarrow 1$，转 2；

针对上面给出的拟人拟物算法可以有几点改进：

1. h 的变化问题，h 程序中代表圆在单步移动中的距离，初值为 1 ，并随着系统势能发生一次逆转而减小一次，这是因为单步距离过大而走过了极值点，因此需要减少 h 的值并反向移动，而 h 的下降不能很快以避免走得过远而从极值点跳了出来。因此往往取下一次的单步距离是上次单步距离的 0.8 到 0.9 倍。
2. 在计算（势能/h）是否小于阈值可以判断是否到了局部最小点，这时候可采用的拟人策略有两种方法：一种是重新将 n 个圆重新随机放置在大圆内进行计算；第二种是选择最拥挤圆进行重新放置，最拥挤的圆可通过计算每个圆的势能（U_i）可得。

3. 最大势能圆的放置策略：最简单的一个方法就是随机选取位置，也可以通过将圆用等间隔分割线进行分划，计算每个格点对应的空缺大小，将这个最拥挤的圆放到最大的空缺中。这是因为往往最拥挤的圆最需要移动。

4. 结合前面几点如果找到了最大空缺放入最拥挤的圆可能会带来新的状况，如果最拥挤的圆小于最大空缺的话，那么这很可能不是一个好的选择，因为不仅这个小圆浪费了这个大空缺的空间，而且之后很难通过策略换出来所以选取的圆首先必须是拥挤的（如果把不拥挤的圆选择进去则可能破坏了之前好的格局/状态），其次选取的圆不能是半径最大的圆，最后选取的圆最好比最大空缺大。

通过 C 实现了拟人拟物算法，最拥挤圆随机放置策略，主程序如下所示：

```
init();                          /* 初始化参数和图形设备 */
sysU=energy_total();
while(sysU>0.0000001){
        get_gradU();                /* 计算势能 */
        if(sysU<sysU_old){        /* 小于之前势能继续 */
                sysU_old=sysU;
                for(k=1;k<M;k++){
                        x[k]=x[k]+gradU[k][0]*h;
                        y[k]=y[k]+gradU[k][1]*h;
                }
        }else{                           /* 大于之前势能 */
                sysU_old=sysU;
                h=h*0.9;
                for(k=1;k<M;k++){
                        x[k]=x[k]+gradU[k][0]*h;
                        y[k]=y[k]+gradU[k][1]*h;
                }
        }
        sysU=energy_total();
        if(sysU>sysU_old) j=sysU-sysU_old;
        else j=sysU_old-sysU;
        if((sysU/j)>10000){        /* 局部极值点 */
                block_pan=get_Dpan(0);
                if(block_pan=l0){
                        t=t+1;
                }
                block_pan=get_Dpan(t);                  /* 找到势能最大的圆 */
                        x[block_pan]=frandom(r[0]);        /* 随机选取 */
                        y[block_pan]=frandom(r[0]);
                if(t<1){
                        l0=block_pan;
                }else{
                        t=0;
                        l0=0;
                }
                h=1;
        }
}
```

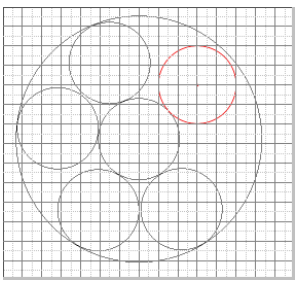
3 一个高效的不等圆 packing 问题解法

拟人过程中为了避免选取不合适的圆放到了不合适的位置，我提出了类似于 [6] 的启发式算法，在拟物拟人的基础上使用贪心策略，最简单的描述就是：**依次将余下最大的圆放到状态/格局中最大得空隙里**。即首先将 n 个需要放进去的圆进行由大到小进行排序，将最大的圆放到圆 R 中，然后进行拟物计算并移动，直至弹性势能（U）为零，接着把集合中余下最大的圆随机放置到圆 R 中，同样进行拟物计算并移动，直至弹性势能（U）为零，对每个要放进去的圆进行这个过程，直至所有圆都放入为止。如果当有一格局/状态不满足互不重叠的情况则使用拟人策略，将最拥挤的圆放到最大空隙或者随机放置，然后继续拟物计算并移动，待弹性势能（U）为零后继续添加剩下的圆。

主程序描述如下：

```
init();                                    /* 初始化参数和图形设备 */
rankpans();                               /* 对 n 个圆进行由大到小的排序 */
nx=frandom(rpans[0][0]);
ny=frandom(rpans[0][0]);
while(pans_in<=M){                          /* 还有圆没有放进去 */
        rpans[pans_in][1]=nx;              /* 将剩下最大的圆放入 */
        rpans[pans_in][2]=ny;
        sysU=energy_total();
        h=1;
        paint_pans();
        while(sysU>0.00001){               /* 没有找到解继续 */
                get_gradU();
                if(sysU<sysU_old){         /* 如果势能减小则继续移动 */
                        sysU_old=sysU;
                        for(k=1;k<pans_in;k++){
                                rpans[k][1]=rpans[k][1]+gradU[k][0]*h;
                                rpans[k][2]=rpans[k][2]+gradU[k][1]*h;
                        }
                }else{
                        sysU_old=sysU;     /* 势能变大则减少 h 并反向移动 */
                        h=h*0.8;
                        for(k=1;k<pans_in;k++){
                                rpans[k][1]=rpans[k][1]+gradU[k][0]*h;
                                rpans[k][2]=rpans[k][2]+gradU[k][1]*h;
                        }
                }
                sysU=energy_total();       /* 计算总体势能 */
                paint_pans();
        }
        maxhole();                         /* 计算最大的空隙 */
        pans_in=pans_in+1;
```

主程序中的 maxhole() 函数用于在大圆 R 中找到最大的圆形空隙，可以采用的方法有对圆用等距正交分割线进行划分，在每个正交分割线交点的格点上，用不断变大的测试圆与其他的圆计算势能，如果测试圆与其他圆无弹性势能则增大测试圆半径，直至有弹性势能则认为这时的测试圆为该格点的最大圆形空隙。如右图所示，该方法的缺点就是计算复杂度太高。



为改进计算效率，我使用的是在每个格点用下一个待放入的圆（由大到小排序号的序列中）

为测试圆，计算与其他圆（除了大圆 R ）的势能，如果有格点使得下一个放入的圆与其他圆无弹性势能（即无重叠），则返回第一个这样的格点坐标作为待放入圆圆心位置，如果没有，则选取使得待放入圆势能最小的格点为下一步该圆的放置位置。

maxhole() 函数描述如下：

```
void maxhole(){
        double i,j;
        int k;
        double min,current,temp1,temp2;
        double t,sum;/*sum 保存下个圆在格点与其他圆的弹性势能和 */
        double tempan;/* 保存待放进圆的半径 */
        double r0;
        double xk,yk;/* 待放进圆圆心坐标 */

        r0=rpans[0][0];/* 大圆半径 */
        tempan=rpans[pans_in+1][0];/* 待要放进去的圆 */
        min=65530;
        current=min;
        step=rpans[0][0]/100;/* 步长等于大圆半径的百分之一 */
        for(i=-r0;i<r0;i=i+step){/* 计算每一个格点对应的原形空隙大小 */
                for(j=-r0;j<r0;j=j+step){
                        if(incircle_or_pan(i,j)==1)
                                continue;/* 如果一个格点在圆中则继续下一个点 */
                        sum=0;
                        for(k=1;k<=pans_in;k++){/* 与每一个圆计算势能 */
                                xk=rpans[k][1];
                                yk=rpans[k][2];
                                if((sqrt((i-xk)*(i-xk)+(j-yk)*(j-yk)))>(rpans[k][0]+tempan))
                                        continue;/* 如果圆心距离大圆两个圆半径则无势能 */
                                t=tempan+rpans[k][0]-sqrt((i-xk)*(i-xk)+(j-yk)*(j-yk));
                                sum=sum+t*t;
                        }
                        temp1=(i-rpans[0][1])*(i-rpans[0][1]);
                        temp2=(j-rpans[0][2])*(j-rpans[0][2]);
                        t=sqrt(temp1+temp2)+tempan-rpans[0][0];
                        if(t<0)t=0;
                        sum=sum+t*t;
                        if(sum==0){/* 在这点势能为零 */
                                nx=i;
                                ny=j;
                                return;/* 找到一个势能为零的点就返回 */
                        }
                        current=sum;
                        if(current<min){/* 返回势能最小的点 */
                                min=current;
                                nx=i;
                                ny=j;
                        }
                }
        }
}
```

为验证该方法的优越性，实验对 [4, 5] 中的测试例子进行了实验（实验环境：VMWARE 虚拟机，单处理器，512 M 内存，编译器 TC），在很短时间内均得到了满足状态/格局，图 1 给出了几个不等圆 packing 问题解的图形表示：参照表 1 对比了 [4, 5] 所用的拟人拟物算法和本文的启发式算法，可以看出启发式算法有较大的效率优势，这是因为启发式算法在初态选取上采用了大圆先进占大空隙，小圆后进占小空隙的策略，使得空间分配合理，在为后面的拟人拟

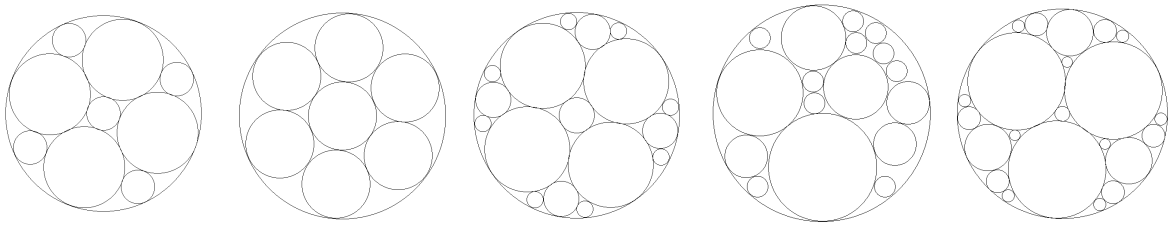


图 1: 五个算例的图形结果

物算法提供了好的状态/格局，避免了所有圆一起放入大圆 R 内造成的很多死锁的情况。不仅避免了初始随机布局后的不合理分布（拟人拟物算法无法预知随机的分布的 n 个坐标是否合理或者有解），而且加速了初始状态到极值点的下降速度，从而得到了好的实验结果。

表 1: 五个算例的结果比对

小圆	小圆半径	大圆半径	拟物拟人算法	拟物算法	启发式算法
9	$r_{1-4} = 1, r_{5-9} = 0.41415$	R=2.4143	>0.22 s	>15 s	≈ 0 s
7	$r_{1-7} = 20$	R=60	<0.05 s	<0.1 s	≈ 0 s
17	$r_{1-4} = 1$, $r_{5-9} = 0.41415$, $r_{10-17} = 0.2$	R=2.4143	>72 s	> 10800 s	<5 s
17	$r_1 = 25$, $r_2 = 20$, $r_{3,4} = 15$, $r_{5-7} = 10$, $r_{8-17} = 5$	R=50	>6.92 s	>29.01 s	<2 s
22	$r_{1-3} = 100$, $r_{4-6} = 48.26$, $r_{7-12} = 23.72$, $r_{13} = 15.47$, $r_{14-19} = 13.45$, $r_{20-22} = 11.61$	R=215.47	>540 s	—*	<5 s

* 无法计算出结果

参考文献

[1] 近世计算理论导引——NP 难度问题的背景、前景及其求解算法研究, 科学出版社

[2] 基于格局变换策略的不等圆 Packing 问题求解算法，计算机应用研究

[3] 解不等圆 packing 问题拟人拟物算法初态选取，华中理工大学学报

[4] 支持求解原型 packing 问题的两个拟人策略，中国科学

[5] 解 packing 及 CNF-SAT 问题的拟物拟人方法，华中理工大学学报

[6] 求解不等圆 packing 问题的一个启发式算法，计算机研究与发展