改进区域划分的圆 Packing 变分算法

余丽娟[1,3], 曹　娟[2], 陈中贵[1,3]*

[1] (福建省智慧城市感知与计算重点实验室(厦门大学)　厦门　361005)
[2] (厦门大学数学科学学院　厦门　361005)
[3] (厦门大学信息科学与技术学院　厦门　361005)
(chenzhonggui@xmu.edu.cn)

摘　要: 通过改进基于 Power 图的区域划分, 提出一种收敛速度更快的圆 packing 算法. 首先固定容器面积, 将输入圆缩小一定的倍数, 随机撒在容器中; 之后对圆心点进行三角化, 并根据相邻圆的半径比值对容器进行区域划分; 再让所有圆在不超出自己区域边界的条件下尽量等比例增长至最大; 最后将划分区域-长大的过程迭代下去, 得到最大增长倍数. 实验结果表明, 该算法能够使得圆 packing 的过程更快地达到收敛.

关键词: 圆 packing; 正则三角化; 最大内圆
中图法分类号: TP391.41　　**DOI:** 10.3724/SP.J.1089.2018.16777

Improved Domain Partitions for Variational Circle Packing

Yu Lijuan[1,3], Cao Juan[2], and Chen Zhonggui[1,3]*

[1] (*Fujian Key Laboratory of Sensing and Computing for Smart City, School of Information Science and Engineering, Xiamen University, Xiamen*　361005)
[2] (*School of Mathematical Sciences, Xiamen University, Xiamen*　361005)
[3] (*School of Information Science and Engineering, Xiamen University, Xiamen*　361005)

Abstract: This paper presents a circle packing algorithm which converges faster by improving the domain partitions based on power diagram. By setting the area of the container fixed, we firstly shrank the input circles so that they wouldn't overlap when scattered randomly in the container. Secondly, we triangulated the circle center points, and divided the container area into sub-regions according to the radius ratio of adjacent circles. And then all circles grew by the same ratio within their own regions. By repeating the "partition-and-grow" operations, we finally got the packing result. Experimental results show that our method can converge more quickly.

Key words: circle packing; regular triangulation; largest inner circle

圆 packing 问题指的是将一组已知个数和大小的圆不重叠地放进一个给定形状的容器中, 求容器面积的最小值. 这是一个 NP 难问题[1], 也就是说不能在确定的多项式时间内求得它的解. 圆 packing 是一个经典的数学问题, 它最早是由 Thurston[2]于 1985 年在一次会议上提出的. 经过近 30 年的研究, 该问题在理论上取得了很大的进展, 同时, 有关它的应用也日益广泛. 在自然科学、工程设计甚至在人们每天的日常生活中, 圆 packing 都有很多应用[3]. 例如, 它可以用来解决如何摆放一堆圆木, 使它们所占空间最小的问题[4]. Yu 等[5]利用圆 packing 提出一种拼图算法, 将图片的重要性和圆的半径联系起来, 根据圆 packing 的结果得到一种对平面的划分, 从而实现拼图的功能. Bai 等[6]利用圆 packing 来解决自组织无线网络中关于极大独立集的相关问题. 此外, 圆 packing 在布料裁剪、交通网络、

收稿日期: 2017-07-10; 修回日期: 2017-09-30. 基金项目: 国家自然科学基金(61472332); 福建省自然科学基金(2018J01104).
余丽娟(1994—), 女, 硕士研究生, 主要研究方向为计算机图形学; 曹　娟(1983—), 女, 博士, 副教授, 主要研究方向为 CAGD&CG; 陈中贵(1982—), 男, 博士, 副教授, CCF 会员, 论文通讯作者, 主要研究方向为计算机图形学、数字图像处理.

设施摆放和容器储物等工业领域都有很多应用[7].

通常，解决圆 packing 问题有 2 种思路：一种是先将所有输入圆不重叠地放进一个足够大的容器中，然后通过不断地缩小容器来求得容器面积最小值；另一种是先固定容器的大小，然后将输入圆缩小一定比例后不重叠地放入容器，再让输入圆在容器中不断长大，最后求得输入圆所能增长的最大比例. 本文算法采用的是第 2 种思路.

1 相关工作和背景

1.1 相关工作

对于圆 packing 问题，根据不同的分类标准有不同的分类. 根据容器形状的不同，可以分为在圆形容器中的圆 packing、在正方形容器中圆 packing 和在矩形容器中圆 packing 等；根据输入圆的半径大小是否相等，可以分为等圆 packing 和不等圆 packing.

近年来，人们提出了针对各种圆 packing 问题的算法. Stoyan 等[8-9]研究了如何将不等圆放进一个宽度固定的长条内，使得长条长度最短；其通过结合分支定界法和梯度下降法，尝试从一个局部最优解跳转到另一个局部最优解，以期望得到全局最优解；但是该算法把圆的半径作为一个变量，所以对等圆的 packing 并不适用. Hifi 等[10]利用模拟退火算法来解决矩形容器中的圆 packing 问题，首先从一个非可行解出发，通过将不满足条件的圆移出容器最后得到可行解. Huang 等[11-12]提出利用贪心算法在矩形容器中和利用启发式算法在圆形容器中进行圆 packing 的算法. Graham 等[13]提出等边三角形容器中的等圆 packing 算法，利用离散仿真法得到了输入圆个数为 22~34 的紧密 packing 结果. 刘景发等[14]结合模拟退火和二分搜索，也提出了正三角形容器内的等圆 packing 算法，其在模拟退火算法中融入了启发式格局更新策略和基于梯度法的局部搜索策略，是一种在三角形容器内进行等圆 packing 的有效算法. Akeb 等[15-163]利用集束搜索算法来解决在圆形容器中的圆 packing 问题，提出一种它的自适应改进算法. Grosso 等[173]基于种群进化提出在圆形容器中的等圆和不等圆 packing 算法. 何琨等[183]提出基于动作空间的拟物求解算法，通过把圆形容器和内部的圆近似看做矩形并结合若干拟人策略提高了连续优化的效率. Lu 等[19]提出基于 Power 图的圆 packing 算法，将重心 Voronoi 剖分的概念进行了推广，能够应用于不同形状的容器，而且对等圆和不等圆的 packing 都适用，其中的局部算法称为局部 Power 图算法 (local power diagram, LPD). LPD 是一种和 Lloyd 迭代[20]类似的迭代算法，此前的各种圆 packing 算法基本都只针对某种形状的容器或者只针对等圆或不等圆 packing，而 LPD 算法是解决更加一般化圆 packing 问题的算法. 本文算法是对 LPD 算法的一种改进.

1.2 背景知识

本文算法基于正则三角化. Delaunay 三角化是计算几何中的基础概念，其满足性质“任何一个三角形的外接圆内不再包含其他的种子点”的三角化就是 Delaunay 三角化. 对 Delaunay 三角化后每个三角形的每条边作垂直平分线，就得到了 Delaunay 三角化的对偶图，称做 Voronoi 图. 给定平面紧致区域 $\Omega\in\mathbb{R}^2$ 和区域内的一组不重合的点 $\{x_i\}_{i=1}^n$ (又称种子点)，则种子点 x_i 对应的 Voronoi 区域满足

$$V(x_i)=\left\{x\in\Omega\,|\,d(x,x_i)\leqslant d(x,x_j),\forall x_i\neq x_j\right\};$$

其中，$d(x,x_i)=\|x-x_i\|^2$. 于是，$\{V(x_i)\}_{i=1}^n$ 就是对区域 Ω 的 Voronoi 剖分.

Power 图是一个和正则三角化相关的概念. 与 Voronoi 图相似，Power 图也是一种关于种子点对平面区域的剖分，是一种加权的 Voronoi 剖分. 给定平面紧致区域 $\Omega\in\mathbb{R}^2$ 和区域内的一组不重合带权重的点 $\{x_i,w_i\}_{i=1}^n$，则种子点 $\{x_i,w_i\}$ 对应的 Power 区域满足

$$V_{\mathrm{w}}(x_i)=\left\{x\in\Omega\,|\,d_{\mathrm{w}}(x,x_i)\leqslant d_{\mathrm{w}}(x,x_j),\forall x_i\neq x_j\right\};$$

其中，$d_{\mathrm{w}}(x,x_i)=\|x-x_i\|^2-w_i$. Power 图可以看做是关于一组圆的对平面的剖分，这组圆的圆心在 $\{x_i\}_{i=1}^n$，半径为 $\left\{\sqrt{w_i}\right\}_{i=1}^n$. Power 图 $\{V_{\mathrm{w}}(x_i)\}_{i=1}^n$ 的对偶图也就是正则三角化.

2 本文算法

2.1 Packing 问题数学表示

原始的 packing 问题如下：在平面紧致区域 $\Omega\in\mathbb{R}^2$ 上，给定容器形状和 n 个已知半径为 $\{r_i\}_{i=1}^n$ 的圆 $\{C_i\}_{i=1}^n$，在满足“n 个圆在容器内且互不重叠”的约束条件下，求容器面积 $|\Omega|$ 的最小值. 设

$\{C_i\}_{i=1}^n$ 的圆心坐标为 $\{(x_i,y_i)\}_{i=1}^n$，那么圆 packing 问题可以表述为

$$\min \quad |\Omega|$$
$$\text{s.t.} \quad \begin{cases} C_i \subseteq \Omega, & 1 \leqslant i \leqslant n \\ C_i \cap C_j = \varnothing, & 1 \leqslant i, j \leqslant n, i \neq j \end{cases}.$$

将满足约束条件下圆 C_i 的圆心坐标 (x_i,y_i) 的集合 $X=\{(x_1,y_1),\cdots,(x_n,y_n)\}$ 称作格局.

为了便于计算求解，可以换个角度看这个问题. 首先固定容器的面积，使 $|\Omega|=m$ (m 为常数); 然后引入一个缩放因子 k 作用于所有的圆 $\{C_i\}_{i=1}^n$，使得它们可以互不重叠地放进容器. 那么圆 packing 问题就变成

$$\max \quad k$$
$$\text{s.t.} \quad \begin{cases} kC_i \subseteq \Omega, & 1 \leqslant i \leqslant n \\ kC_i \cap kC_j = \varnothing, & 1 \leqslant i, j \leqslant n, i \neq j. \\ |\Omega| = m \end{cases}$$

2.2 LPD 算法

基于 Power 图的 LPD 算法步骤如下:

输入. n 个半径分别为 $\{r_i\}_{i=1}^n$ 的圆 $\{C_i\}_{i=1}^n$，容器 Ω，$|\Omega|=m$ (m 为常数), 缩放因子 k 的初始值.

输出. k 的终值和圆 $\{kC_i\}_{i=1}^n$ 最终格局分布.

Step1. 将缩放因子 k 的初始值作用于圆 $\{C_i\}_{i=1}^n$，得到缩小后的圆 $\{kC_i\}_{i=1}^n$，随机生成圆 $\{kC_i\}_{i=1}^n$ 的初始格局分布 $X=\{(x_1,y_1),\cdots,(x_n,y_n)\}$.

Step2. 对圆心点 $\{(x_i,y_i)\}_{i=1}^n$ 作正则三角化，根据三角化结果构建每个圆的 Power 区域 Ω_i，得到 $\{\Omega_i\}_{i=1}^n$.

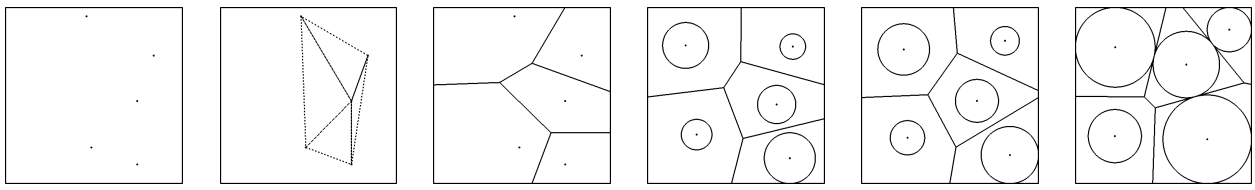
Step3. 对每个圆的 Power 区域 Ω_i，求它的最大内圆，得到最大内圆的圆心位置 $(\tilde{x}_i,\tilde{y}_i)$ 和最大内圆半径 $\tilde{r}_i$，并求最大内圆半径和原来圆半径的比值 $k_i=\tilde{r}_i/r_i$.

Step4. $k' \leftarrow k$，$k \leftarrow \min_i k_i$.

Step5. 如果 $k=k'$，则获得终止 k 值和终止格局分布 X，算法结束; 否则，更新当前圆圆心位置至 $(\tilde{x}_i,\tilde{y}_i)$，更新当前圆的半径为 kr_i，转 Step2.

图 1 所示为 LPD 算法的过程示意图. 这个例子中，容器是正方形，面积为 1，输入圆半径分别是 10, 12, 15, 18 和 20，初始 $k=5\text{E}-8$. 图 1a 所示为将输入圆缩小 k 倍之后随机撒在容器中所得格局分布; 图 1b 所示为图 1a 中圆的圆心正则三角化结果; 图 1c 所示为三角化对应的 Power 图，即每个圆得到自己的 Power 区域; 图 1d 所示为这组圆同步增长一次的结果以及对应的 Power 图，得到 $k=7.212\text{E}-3$; 图 1e 所示为第 2 次增长后的结果及对应的 Power 图，得到 $k=8.104\text{E}-3$; 图 1f 所示为迭代停止时的格局分布，$k=1.257\text{E}-2$. 实现过程中，迭代停止的条件不是 $k=k'$，而是它们的比值小于给定的阈值.

LPD 算法利用了 Power 图的优良性质: 2 个圆之间的 Power 分界线垂直于 2 个圆圆心的连线; 如果 2 个圆相离，则分界线位于 2 个圆之间且与 2 个圆不相交; 如果 2 个圆相切，则分界线过 2 个圆切点; 如果 2 个圆相交，则分界线过 2 个圆交点; 如果一个圆包含另一个圆，则分界线位于 2 个圆之外. 易知，LPD 算法的圆 packing 过程中只会出现 2 个圆相离或相切的情况. 另外，从 Power 图的定义可知 $\bigcup_{i=1}^n \Omega_i = \Omega$，即 LPD 算法能将容器划分成 n 块不重叠的区域.



a. 随机撒点　b. 正则三角化　c. Power 图　d. 第 1 次增长　e. 第 2 次增长　f. 最后结果

图 1　LPD 算法过程示意图

然而，这种划分区域的方法也有一个缺点: 当迭代进行到后期，圆半径比圆之间的缝隙大得多的时候，迭代会变得很慢. 参考图 2，具体分析如下:

$$|PO_1|^2 - r_1^2 = |PO_2|^2 - r_2^2 \Rightarrow (r_1+t_1)^2 - r_1^2 = (r_2+t_2)^2 - r_2^2 \Rightarrow 2r_1t_1 + t_1^2 = 2r_2t_2 + t_2^2 \Rightarrow \frac{2r_1+t_1}{2r_2+t_2} = \frac{t_2}{t_1} \tag{1}$$

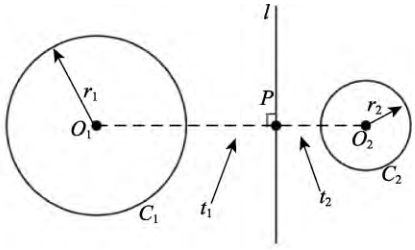


图 2 2 个圆之间的 Power 分界线

根据 Power 分界线的定义, P, O_1, O_2 的关系如式(1), 则当 r_1, r_2 远远大于 t_1, t_2 的时, 式(1)可以写成 $r_1 / r_2 \approx t_2 / t_1$, 这表明 2 个圆到 Power 分界线的距离会和圆的半径成反比. 可以想象, 如果一个大圆周边都是小圆, 那么大圆分得的可增长空间会比周围小圆分得的小得多; 又因为所有圆必须等比例同步增长, 所以小圆的增长会受到大圆增长的限制, 导致整体增长得比较慢、迭代次数比较多. 本文算法就是针对这个问题进行了改进.

2.3 本文算法

本文算法主要是针对 LPD 算法的区域划分方法进行改进. 算法步骤如下:

输入. n 个半径分别为 $\{r_i\}_{i=1}^n$ 的圆 $\{C_i\}_{i=1}^n$, 容器 Ω, $|\Omega| = m$ (m 为常数), 缩放因子 k 的初始值.

输出. k 的终值和圆 $\{kC_i\}_{i=1}^n$ 最终格局分布.

Step1. 将缩放因子 k 的初始值作用于圆 $\{C_i\}_{i=1}^n$, 得到缩小后的圆 $\{kC_i\}_{i=1}^n$, 随机生成圆 $\{kC_i\}_{i=1}^n$ 的初始格局分布 $X = \{(x_1, y_1), \cdots, (x_n, y_n)\}$.

Step2. 根据正则三角化的结果和圆的半径大小对每个圆构建一个自己的区域 Ω_i.

Step3. 对每个圆的区域 Ω_i 求它的最大内圆, 得到最大内圆的圆心位置 $(\tilde{x}_i, \tilde{y}_i)$ 和最大内圆半径 $\tilde{r}_i$, 并求最大内圆半径和原来半径的比值 $k_i = \tilde{r}_i / r_i$.

Step4. $k' \leftarrow k$, $k \leftarrow \min_i k_i$.

Step5. 如果 $k / k' \geqslant \delta$, 更新当前圆的位置至 $(\tilde{x}_i, \tilde{y}_i)$, 更新当前圆的半径为 kr_i, 转 Step2; 否则, 得到最大的增长倍数 k 和终止格局分布 X, 算法结束.

本文算法和 LPD 算法的不同之处主要在于 Step2 中构建各个圆的增长区域的方法不一样. 图 3 所示为本文算法的过程示意图, 算法输入和图 1 中的输入相同.

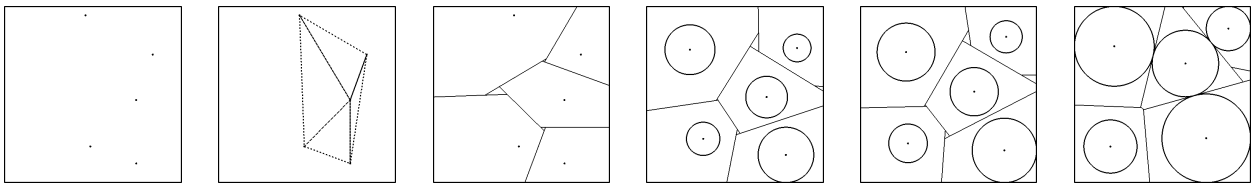
2.3.1 本文区域划分算法

通过对输入圆的圆心进行正则三角化, 在得到点与点之间的邻接关系的基础上, 根据圆的半径值对每个圆和它的 1-邻域圆之间划分一条分界线 l (简称为 1-邻域算法). 分界线 l 满足 2 条性质:

(1) l 垂直于 2 个圆圆心的连线;

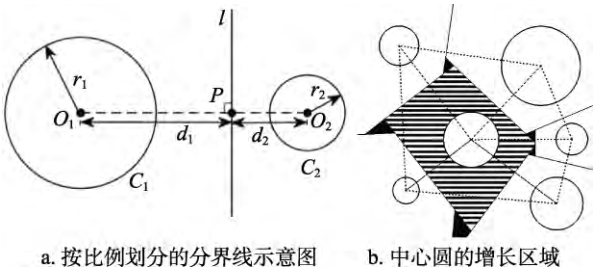
(2) 2 个圆圆心到分界线之间的距离比值等于 2 个圆的半径比值, 即 $d_1 / d_2 = r_1 / r_2$.

图 4a 所示为 2 个圆之间分界线的示意图, 图中 $l \perp O_1O_2$, $|PO_1| / |PO_2| = r_1 / r_2$, 即 $d_1 / d_2 = r_1 / r_2$. 图 4b 所示为单个圆的增长区域示意图, 其中, 虚直线表示三角化, 实直线表示区域划分的分界线, 阴影部分表示中心圆的可增长区域, 黑色部分是不属于任何一个圆的区域. 可以看出, 本文的区域划分方法虽然能够保证给不同大小的圆分配对应大小的可增长空间, 但它是一种对平面的不完全划分. 经过划分, 平面里存在 2 种区域: 一种是属于圆的增长区域; 另一种是冗余的区域, 这部分就是浪费掉的空间. 实验表明, 随着迭代的进行, 这种浪费的空间所占的比重会越来越小.



a. 随机撒点 b. 正则三角化 c. 区域划分 d. 第 1 次增长 e. 第 2 次增长 f. 最后结果

图 3 本文算法过程示意图



a. 按比例划分的分界线示意图 b. 中心圆的增长区域

图 4 本文算法区域划分方法示意图

另外, 本文在实现过程中发现, 1-邻域算法可能会导致圆的可增长区域相交. 如图 5a 所示, 阴影部分是 2 个圆的可增长区域, 中间一小块黑色部分就是这 2 块区域的相交部分. 因为圆会在自己的区域内尽量长大, 所以这种情况可能会导致圆长大后发生重叠. 但是在本文的实验中, 没有发现圆长大后重叠的情况.

为了使算法更加严密，本文对 1-邻域算法进行改进：除了进行 1-邻域的分界线划分外，还对中心圆和其 2-邻域圆也划分分界线. 对于图 5a 的格局分布，这种方法得到的划分结果如图 5b 所示. 实验表明，这种方法能够减少区域重叠的发生.

虽然上述的 2-邻域改进是有效的，但是会出现新的问题：当输入圆的个数很多时，中心圆的 2-邻域圆就会很多，所以运行时间是 1-邻域算法的 2~3 倍. 因此本文做了一个折中，当迭代次数小于 λ 时，进行 2-邻域划分，此后的迭代就只进行 1-邻域的划分(简称为 2-邻域算法)，其中 λ 是个经验值.

2.3.2　凸多边形的最大内圆算法

由于本文算法中圆所属的区域是由若干个半平面相交得到的，因此可以将它看成一个凸多边形围成的区域. 算法的 Step3 主要是求这个凸多边形的最大内圆. 本文采用郑梅生等[21]提出的算法，下面简要介绍这种算法.

具体算法步骤如下：

输入. 凸多边形和初始圆心位置 $O(x_0, y_0)$.

输出. 凸多边形内最大内圆的圆心坐标和半径.

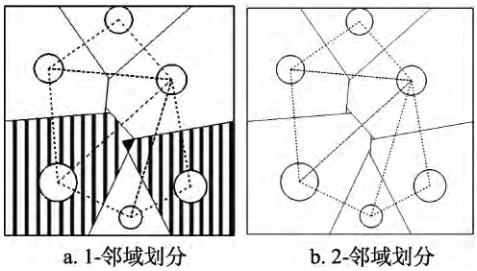
Step1. 计算圆心 $O(x_0, y_0)$ 到凸多边形各边的距离，得到其中 2 个最短的距离 d_1 和 d_2，且垂足分别为 $A(x_1, y_1)$ 和 $B(x_2, y_2)$，即 $|OA| = d_1, |OB| = d_2$. 注意：垂足必须落在多边形的边上而非延长线上才参与比较.

Step2. 取一个参考点 $C(x_c, y_c)$，C 满足下列条件 $|CA|/|CB| = |OA|/|OB|$. 如果初定圆心 O 与 A, B 共线，则取参考点为 A 点. C 的坐标计算公式为

$$\begin{cases} x_c = x_1 + (x_2 - x_1)d_1/(d_1 + d_2) \\ y_c = y_1 + (y_2 - y_1)d_1/(d_1 + d_2) \end{cases}.$$

Step3. 在 CO 延长线上取新圆心 $O'(x_0', y_0')$，新圆心的坐标计算公式为 $\begin{cases} x_0' = x_0 + a(x_0 - x_c) \\ y_0' = y_0 + a(y_0 - y_c) \end{cases}$，其中，$a$ 为步长因子.

Step4. 计算 $O'(x_0', y_0')$ 到凸多边形各边的最短距离 d'. 若 $d' \geqslant d_1$，则 $O \leftarrow O'$，转 Step1；若 $d' < d_1$，则调整步长因子 $a \leftarrow 0.5a$，转 Step3，重新计算新圆心坐标.



a. 1-邻域划分　　b. 2-邻域划分

图 5　容器区域划分示意图

Step5. 重复 Step1~Step4，直到 a 小于要求精度或者迭代次数大于给定次数则停止计算，得到凸多边形内圆圆心坐标 $O'(x_0', y_0')$ 和半径 d'.

图 6 所示为文献[21]算法的简要示意图. 本文算法实现过程中，初始圆心位置取的是凸多边形的重心位置.

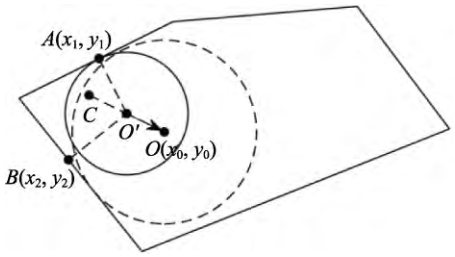


图 6　凸多边形最大内圆算法示意图

3　实验结果及分析

本文算法通过 C++实现，其中正则三角化的实现使用 CGAL(computational geometry algorithms library). 所有的实验数据都是在 Intel 的 Xeon 处理器，3.30 GHz CPU，12 GB 内存的电脑上获得的.

实验中的参数设置如下：2-邻域算法中 $\lambda = 50$；本文算法步骤 Step5 中的停止迭代阈值 $\delta = 1.0001$. 求凸多边形的最大内圆算法中 a 的初始值为 0.1，终止条件为 $a < 1\text{E}-5$ 或者迭代次数大于 500 次. 关于容器面积：当容器为正方形，正方形边长为 1，即 $m = 1$；当容器为圆形，圆的半径为 10，即 $m = 100\pi$.

本文 2-邻域算法、1-邻域算法的实验结果与 LPD 算法的收敛速度对比如图 7 所示，其中，图 7a~图 7c 容器为正方形，图 7d 容器为圆形. 输入圆取自 Huang 等[22]的 NR10-1*, NR30-1, NR60-1. 从图 7a 和图 7b 可以看出，2-邻域算法的收敛曲线与 1-邻域算法的基本重合，且都比 LPD 算法收敛快得多. 从图 7c 可以看出，在某些例子中，本文算法收敛速度和 LPD 算法相近. 图 7d 中，LPD 算法迭代到大约 41 次就停止了，而本文算法一直运行至收敛.

图 8~图 9 所示为 NR60-1 例子在正方形容器和圆形容器中迭代增长的最终结果. 图 10 所示为其他形状容器的 2-邻域算法结果图. 从图 8(对应的收敛速度曲线为图 7c)可以看出，2-邻域算法不仅迭代收敛得快，还能得到比 1-邻域算法和 LPD 算法更优的结果. 从图 9 及其对应的收敛速度对比曲线图 7d 可以看出，本文算法比 LPD 算法更不容易陷入局部最优解.

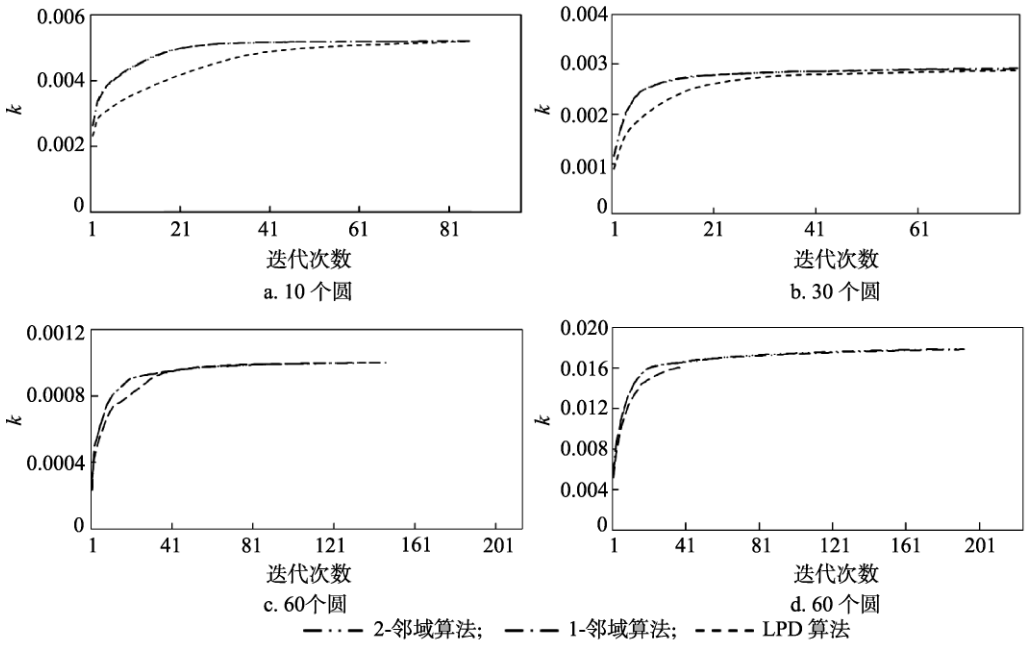
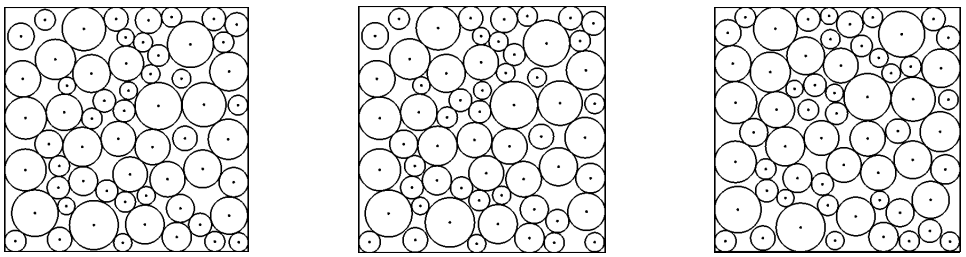
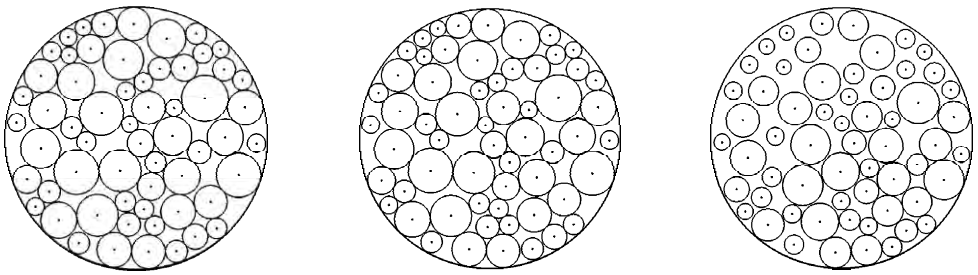


图 7 3 种算法收敛速度比较



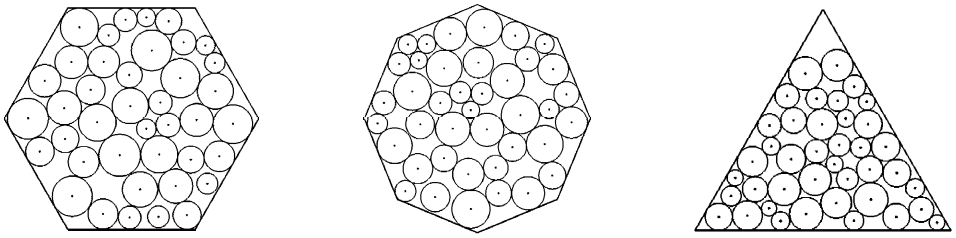
a. 2-邻域算法, k =1.00032 b. 1-邻域算法, k = 0.998107 c. LPD 算法, k = 0.99517

图 8 正方形容器结果比较图



a. 2-邻域算法, k = 0.01786 b. 1-邻域算法, k = 0.01782 c. LPD 算法, k = 0.01606

图 9 圆形容器结果比较图



a. 六边形容器 b. 八边形容器 c. 三角形容器

图 10 输入圆为 NR40-1 时 2-邻域算法不同形状容器结果展示

4 结　语

本文从“大圆分得可增长空间大，小圆分得可增长空间小”的角度出发，改进了 LPD 算法，该算法简单，收敛速度快. 图 8~图 10 所示例子中, 3 种算法的运行时间大致在 28 s~150 s，运行时间会因为输入圆的初始化位置不同而差异比较大. 整体上来说, 2-邻域划分会比 1-邻域划分稍慢, 1 邻域划分会比 LPD 算法稍慢，算法最耗时的部分为计算圆所属区域的多边形顶点的坐标. 另外，本文采用的求凸多边形最大内圆的算法也会随着输入圆的个数增多耗时增加很明显，今后可以通过采用更加高效的凸多边形最大内圆算法来加速计算.

本文算法是一种局部算法，得到的可能只是局部最优解，今后还可以在该算法的基础上进行全局算法的改进. 例如，给本文算法结果一个全局扰动，就可能使迭代继续下去，有望得到更好的解.

参考文献(References):

[1] Fowler R J, Paterson M S, Tanimoto S L. Optimal packing and covering in the plane are NP-complete[J]. Information Processing Letters, 1981, 12(3): 133-137

[2] Stephenson K. Introduction to circle packing: the theory of discrete analytic functions[M]. Cambridge: Cambridge University Press, 2005

[3] Hifi M, M'hallah R. A literature review on circle and sphere packing problems: models and methodologies[J]. Advances in Operations Research, 2009, 2009(2009): Article No.150624

[4] Szabó P G, Markót M C, Csendes T, *et al*. New approaches to circle packing in a square: with program codes[M]. New York: Springer Science & Business Media, 2007

[5] Yu Z Q, Lu L, Guo Y W, *et al*. Content-aware photo collage using circle packing[J]. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(2): 182-195

[6] Bai S, Che X J, Bai X, *et al*. Maximal independent sets in heterogeneous wireless Ad Hoc networks[J]. IEEE Transactions on Mobile Computing, 2016, 15(8): 2023-2033

[7] Castillo I, Kampas F J, Pintér J D. Solving circle packing problems by global optimization: numerical results and industrial applications[J]. European Journal of Operational Research, 2008, 191(3): 786-802

[8] Stoyan Y G, Yaskov G N. Mathematical model and solution method of optimization problem of placement of rectangles and circles taking into account special constraints[J]. International Transactions in Operational Research, 1998, 5(1): 45-57

[9] Stoyan Y G, YasKov G N. A mathematical model and a solution method for the problem of placing various-sized circles into a strip[J]. European Journal of Operational Research, 2004, 156(3): 590-600

[10] Hifi M, Paschos V T, Zissimopoulos V. A simulated annealing approach for the circular cutting problem[J]. European Journal of Operational Research, 2004, 159(2): 430-448

[11] Huang W, Li Y, Akeb H, *et al*. Greedy algorithms for packing unequal circles into a rectangular container[J]. Journal of the Operational Research Society, 2005, 56(5): 539-548

[12] Huang W Q, Li Y, Li C M, *et al*. New heuristics for packing unequal circles into a circular container[J]. Computers & Operations Research, 2006, 33(8): 2125-2142

[13] Graham R L, Lubachevsky B D. Dense packings of equal disks in an equilateral triangle: from 22 to 34 and beyond[J]. The Electronic Journal of Combinatorics, 1995, 2(1): Article No.39

[14] Liu Jingfa, Zhang Guojian, Liu Wenjie, *et al*. Heuristic algorithm for the equal circles packing problem in equilateral triangle container[J]. Journal of Computer-Aided Design & Computer Graphics, 2012, 24(6): 808-815(in Chinese)
(刘景发，张国建，刘文杰，等. 正三角形容器内等圆 Packing 问题的启发式算法[J]. 计算机辅助设计与图形学学报, 2012, 24(6): 808-815)

[15] Akeb H, Hifi M, M'Hallah R. A beam search algorithm for the circular packing problem[J]. Computers & Operations Research, 2009, 36(5): 1513-1528

[16] Akeb H, Hifi M, M'Hallah R. Adaptive beam search lookahead algorithms for the circular packing problem[J]. International Transactions in Operational Research, 2010, 17(5): 553-575

[17] Grosso A, Jamali A R M J U, Locatelli M, *et al*. Solving the problem of packing equal and unequal circles in a circular container[J]. Journal of Global Optimization, 2010, 47(1): 63-81

[18] He Kun, Yang Chenkai, Huang Menglong, *et al*. Quasi-physical algorithm based on action space for solving the circles packing problem with equilibrium constraints[J]. Journal of Software, 2016, 27(9): 2218-2229(in Chinese)
(何　琨，杨辰凯，黄梦龙，等. 动作空间带平衡约束圆形 Packing 问题的拟物求解算法[J]. 软件学报，2016，27(9): 2218-2229)

[19] Lu L, Choi Y K, Sun F, *et al*. Variational circle packing based on power diagram[R]. Hong Kong: The University of Hong Kong, 2011

[20] Lloyd S. Least squares quantization in PCM[J]. IEEE Transactions on Information Theory, 1982, 28(2): 129-137

[21] Zheng Meisheng, Chen Ning, Song Chao. An algorithm for determining the largest internal circle in arbitrary polygons[J]. Machinery Design & Manufacture, 2003(5): 84-85(in Chinese)
(郑梅生，陈　宁，宋　超. 计算任意多边形最大内圆的一种算法[J]. 机械设计与制造, 2003(5): 84-85)

[22] Huang W Q, Li Y, Li C M, *et al*. New heuristics for packing unequal circles into a circular container[J]. Computers & Operations Research, 2006, 33(8): 2125-2142