使用模拟退火算法求等圆 Packing 问题的数值解

面向科学问题求解的编程实践 报告

吴天铭 PB20000196

2021 年 7 月 11 日

1 背景介绍

等圆 Packing 问题 等圆 Packing 问题研究如何将一定数量的半径为 1 的圆形不重叠地装进一个尺寸最小的正方形中。寻找 Packing 问题的较优解和最优解，在工程上有着重要的优化意义。该问题可以由数学语言描述如下：

设每个圆形的半径为 1。给定 n，找出一组 $x_1, x_2, \ldots, x_n$ 和 $y_1, y_2, \ldots, y_n$ 满足约束

$$\forall i, j \qquad \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq 2 \qquad (1 \leq i \leq n, 1 \leq j \leq n)$$

在此约束下最小化函数 F：

$$F_x = \max_{1 \leq i \leq n} x_i - \min_{1 \leq i \leq n} x_i$$

$$F_y = \max_{1 \leq i \leq n} y_i - \min_{1 \leq i \leq n} y_i$$

$$F = \max\{F_x, F_y\}$$

该问题涉及在一定约束条件下求解某个多元函数的最值，因此属于全局优化问题。

模拟退火算法 让钢铁从高温状态缓慢退火回到低温状态的过程中，钢铁的内部结构会变得有序，从而实现应力最小、势能最低。模拟退火算法是从此物理过程中得到启发的一种全局优化的方法。

2 实验目的

该实验中，我将尝试使用模拟退火算法求等圆 Packing 问题的数值解。

等圆 Packing 问题中的待优化函数 F 是一个定义在 $2n$ 维向量空间上的函数。通过适当调整模拟退火算法中的各个参数，用启发式方法确保自变量时刻处于定义域中，并对一些细节部分的实现进行微调，可以用模拟退火算法求解出等圆 Packing 问题的最优数值解。

3 实验环境

- 使用语言：Python 3.9.2

- 使用模块：`numpy`, `matplotlib`
- 编写环境：VSCode

4 实验内容

4.1 搭建模型

用 `points = np.ndarray((n, 2))` 表示总共 n 个圆的圆心坐标，利用 `numpy` 库中的 `max()` 和 `min()` 函数可以快速求出点集横纵坐标的最大最小值，进而求出目标函数 F。

4.2 算法设计：模拟退火函数

最朴素的全局优化算法往往是沿着多元函数的负梯度方向行走，使得每次迭代之后当前位置的函数值降低，以求在一定次数迭代之后取得一个较小的值。

模拟退火算法受物理退火过程的启发，采用一种随机化的方式求解函数的全局最小值。该算法从一组初始设定的自变量取值出发进行若干次迭代，不断随机改变自变量的取值，并根据函数值变化对自变量进行取舍，直到迭代结束时得到最优的自变量，使得函数值最小化。迭代过程中有一重要参数 T 称为“**温度**”，用于决定随机运动的无序程度。最初温度被初始化为一个正浮点数；每次迭代时，做如下操作：

1. 从当前的自变量取值向量 $\mathbf{r}$ 随机生成一组近邻的新的自变量取值 $\mathbf{r}+\Delta\mathbf{r}$（“**随机游走**”）。相应地，函数值也会有一减小量 $\Delta F = F(\mathbf{r}) - F(\mathbf{r}+\Delta\mathbf{r})$。
2. 根据函数值减小量，判断是否接受这个新的自变量值：

   1° 若 $\Delta F \geq 0$，函数值变小，更加接近最小值的目标，因此立即接纳这个取值，从 $\mathbf{r}$ 移动到 $\mathbf{r}+\Delta\mathbf{r}$。

   2° 若 $\Delta F < 0$，函数值增大，这是不优的；但是此时我们并不直接放弃此次移动，而是视温度而定，以 $P(\Delta F, T) = \exp(\Delta F/T)$ 的概率接受这个移动。

   此时的温度越高，就更有可能“破格接受”随机游走中遇到的较劣的值，进而能够**跳出局部极小值**；温度越低，越可能沿着严格的梯度方向下降，找到精确的最小值。这是模拟了物理过程中的随机性。
3. 每经过一定次数的迭代，就让 T 的值乘以一个衰减率 α，使温度呈指数下降。当 T 的值足够小时，算法的行为会趋向严格的梯度下降，此时即可认为找到了一个足够优的全局最小值。

温度参数的设定 要让退火过程有效进行并收敛到全局最小值，需要合理设定温度起始值 $\{T_{begin}, T_{end}\}$、温度衰减率 α 等各项参数。为了让概率函数 $P(\Delta F, T) = \exp(\Delta F/T)$ 发挥作用，我们希望 ΔF 始终与 T 处在同一个量级。联系实际问题模型，状态函数 F 表征可以围住所有圆盘的最小正方形的边长，在退火过程中，F 的变化幅度应该从“圆半径的数倍”减少至“趋近 0”。

经过反复测试，最终取 $T_{begin} = 0.3, T_{end} = 0.0003, \alpha = 0.99$ 可以得到较好的效果和效率。

在退火函数核心部分之外，还需要设计实现**随机游走函数**。

4.3 算法设计：随机游走函数

待优化函数的自变量是 n 个圆心的坐标。为了实现模拟退火算法，需要让自变量在定义域中**随机游走**，产生邻近的新解。因此需要设计随机游走函数。设计该函数时，有几个需要注意的地方，下文分别讨论：

问题 (1) 游走幅度应当随温度降低而减小，这是为了在退火初期时有足够的几率走遍全部的解空间，且在退火后期能够稳定地收敛到一个解。

问题 (2) 圆盘两两之间不能重叠，这是自变量的定义域限制。

4.3.1 步长函数

对于问题 (1)，我设计了步长函数 `Step(T)`，根据温度计算出一个合适的步长作为游走幅度。我选择了 S 形曲线函数：

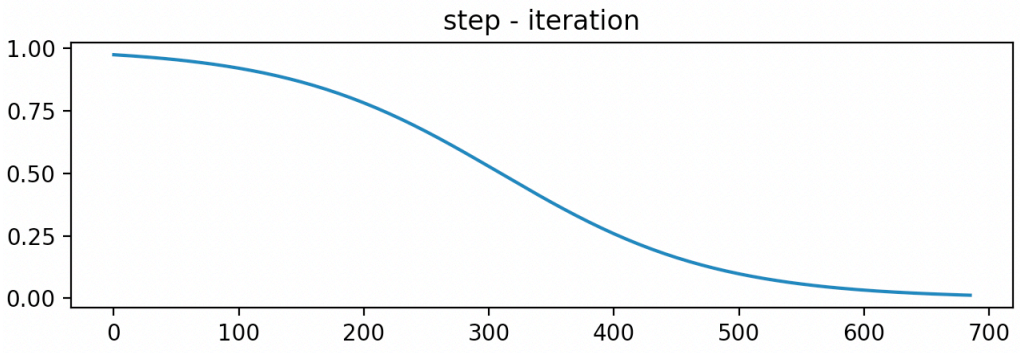

图 1: S 形曲线，游走幅度对迭代次数的函数。

$$Step(T) = \frac{Step_0}{1 + \exp(\text{倾斜程度} \times (\xi(T) - \xi_0))}$$

其中 ξ 为降温进程，$0 \le \xi \le 1$（与迭代次数成线性关系）：

$$\xi(T) = \frac{\ln(T/T_{begin})}{\ln(T_{end}/T_{begin})}$$

参数取值为 $Step_0 = 1.0$，倾斜程度 $= 8.0$，$\xi_0 = 0.45$ 时，对于 $n \le 16$ 可以取得不错的效果。

4.3.2 松弛函数

对于问题 (2)，我设计了 `Relax(points, T)` 函数，作用是确保圆盘两两不重叠。具体实现利用了**启发式方法**，模拟物理中弹性物品的弹力定律。令重叠的圆盘之间产生弹力，对所有的圆盘分别计算受到的力，并按照受力推动圆盘。这样可以让圆盘之间弹力减少，因此将函数命名为 “Relax”（松弛）。多次迭代调用 `Relax` 函数，直到其返回值为 0.00，即圆盘之间没有任何重叠部分为止。这样可以保证每次游走后得到的邻近解都处在定义域内。

对于两个圆盘 $\mathbf{r}_1 = (x_1, y_1), \mathbf{r}_2 = (x_2, y_2)$，记圆心之间的距离为

$$dist(\mathbf{r}_1, \mathbf{r}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

如果它们有重叠部分，即 $dist(\mathbf{r}_1, \mathbf{r}_2) \leq 2.00$，则令它们各自得到一个径向的位移，大小为

$$|d| = \frac{1}{5} \times (2.00 - dist(\mathbf{r}_1, \mathbf{r}_2) + T/10.00)$$

1/5 的因子起到了类似惯性的作用，保证 `Relax` 结束后圆盘之间仍有一定程度的紧凑性。T/10.00 是额外附加的一个力，是为了使圆盘更快分离，减少 `Relax` 函数重复调用次数，节省算法执行时间；同时这个附加力会随温度降低而快速衰减，因此不会对算法结果的准确性造成太大影响。

4.3.3 游走函数

由于总共有 n 个圆盘，在游走时可以考虑单独移动其中的一个圆的位置，也可以对所有的 n 个圆同时进行移动。在代码中使用了两种游走函数，分别是随机选择一个圆进行移动（“小步”）、移动所有圆（“大步”）。退火算法执行时，按照 $n:1$ 的次数比例交替执行小步、大步两种游走函数。

4.4 数据可视化

使用 `matplotlib` 库中的绘图函数，对模拟状态和参数曲线进行可视化，为观察程序运行状态、合理调节参数提供便利。

界面中一共显示三幅子图，从上至下依次是：

(1) **实时状态显示**：以最小的正方形为外边框，绘制出此时所有圆盘的摆放位置，每隔一段时间刷新以实时显示状态。

(2) **步长-迭代次数曲线**：以迭代次数 n 为横轴，游走幅度 Step 为纵轴，显示 `Step(T)` 函数的图像。其中，第 n 次迭代对应的温度为 $T = T_{begin} * \alpha^n$。

(3) **接受概率曲线**：

(a) 以蓝色绘制 $\Delta F = -0.01$ 时，接受概率 $P(T) = \exp\left(\frac{-0.01}{T}\right)$ 对 T 的关系曲线。可以一定程度上反映出温度 T 取不同的值时接受劣解的概率大小。

(b) 在模拟退火算法运行时，图上用绿色“×”同步打点，表示当前温度下有多少比例的邻近解被接受了（包括优解和按概率接受的劣解）。

子图 (1) 便于直观看见退火过程中圆盘位置的实时变化；子图 (2) 和 (3)(a) 显示的是静态的关系曲线，它们的横轴对齐，相同横轴位置大致能够表示一致的迭代次数。子图 (3)(b) 显示了邻近解的**接受率（Acceptance Rate）**：对于一个典型的退火算法，接受率在迭代开始时应处在 90% 以上，在迭代结束时应处在 10% 以下。

5 实验结果

对于 $\mathbf{2 \le n \le 16}$ 的各种情况，算法不仅能在误差 0.5% 以内得到已知的最优目标函数值，还能给出结果所对应的最优圆盘摆放方式。通过与已被严格证明的最优解 [1] 对比，可以验证这些结果的正确性。

对于 $\mathbf{n \ge 17}$ 的情况，算法能够给出一个比较优的目标函数值，但不能保证给出正确的圆盘摆放方式。当圆盘数量比较大时，圆盘的摆放接近于正六边形密铺，不同的摆放方式之间的差别显得更加细微，可能需要更先进的算法才能有效求解。

模拟退火相对于朴素梯度下降法的最大优点在于模拟退火能够跳出局部极小值。在本问题中，局部极小值往往表现为具有**较强对称性**的圆盘布局。例如 $n = 5$ 时，如左图所示是它的最优摆放方式，而右图是它的一种最常见的局部极小值的情况。

图 2: n=5 的最优方式和局部极小值举例。左侧摆放方式所需的正方形框比右边的更小

为了**验证模拟退火算法比梯度下降更优秀**，可以对比两种算法计算最优解的能力。在程序中，令概率函数 `P(e, e1, T)` 总返回 0，使模拟退火算法退化为朴素梯度下降算法。从随机摆放开始，利用两种算法分别计算 $n = 5$ 时的解。各重复 50 次得到如表所示数据。

算法	得到最优解的次数	仅得到局部极小值的次数	最优解占比
朴素梯度下降算法	19	31	38%
模拟退火算法	29	21	58%

表 1: $n = 5$ 时，重复 50 次运行算法得到的统计数据

可以看到，朴素梯度下降算法在大多数情况下最终收敛到了局部极小值，而不能得到最优解。模拟退火算法则有更大几率得到最优解。这说明**模拟退火算法确实具有更好的跳出局部极小值的能力**。

界面显示效果和计算过程如附图所示。另附 $n = 8, 11, 13, 15$ 时的运行截图于文件夹中。

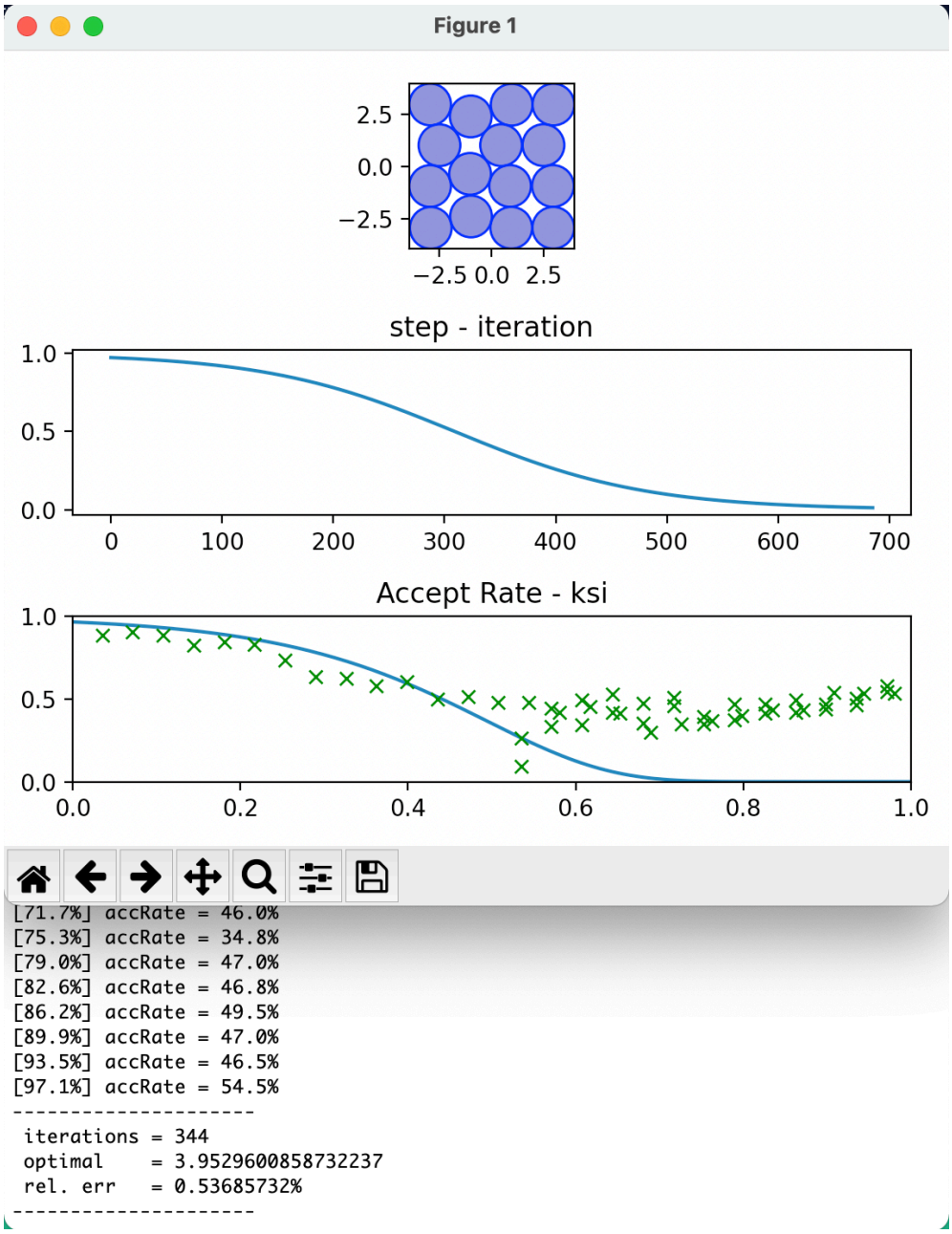

图 3: n=15 时的示例运行效果。在子图 (3) 中有多行绿色“×”，指示进行了多轮退火迭代。

程序输出结果对照如下：

`iterations` 总迭代次数

`optimal` 得到的最小函数值

`rel. err` 与理论最小值之间的百分误差

6 实验总结

本项目使用 Python 语言，利用 `numpy` 和 `matplotlib` 模块实现了求解等圆 Packing 问题的模拟退火算法，计算了 $n \leq 16$ 时的最优解和对应的摆放方式；并与朴素梯度下降算法比较，证实模拟退火算法具有能够跳出局部极小值的优点。

在实现算法过程中，遇到的最大的困难是**参数调整**。无论是温度的起始、终止、衰减系数，还是随机游走的步长，或是松弛过程中两个圆盘之间受力的系数，都需要调整成适当的定值或函数，使得模拟退火算法能够有效工作。假如任何一个参数的取值不合理，则圆盘要么分散得很远，要么剧烈抖动，最终导致算法无法收敛到一个很优的结果。实际上，实现初期的程序甚至在概率函数 `P(e, e1, T)` 中加入了可以调节的系数。这个系数在后期被移除，虽然因此需要重新调节其他的所有参数，但少了一个变化的因素，反而方便了将其他参数调到理想的范围内。

调整参数的工作目前是完全由人工估算和测试完成的。在这方面有巨大的提升空间，例如可以在统计学上进行更严格的数学分析，导出理想的参数取值或函数关系；或是可以利用另一个程序对算法参数的不同取值进行自动化的全面测试，以便“训练”、“挑选”出效果比较好的参数值。

在算法性能方面，我注意到，仅仅数百次迭代就需要长达分钟的运行时间。这可能是 Python 自身的效率问题造成的。可以预想，将算法核心部分使用 `C++` 语言实现，将会得到极大的速度提升。不过这样难以实现相同水平的数据**可视化**。另一方面，程序中使用的 `Relax` 函数需要多次反复调用才能完成功能，可能也是性能瓶颈之一。

收获 在本次实验中，我上手实现了模拟退火算法，熟悉了使用 Python 的程序设计、数据处理、数据可视化；锻炼了从实际问题中提炼建模的能力。通过实现算法、调节参数的过程，加深了对模拟退火等随机算法的理解。

参考文献

[1] E. Specht, Packomania website: `www.packomania.com`.