大规模圆内等圆装填问题的几何批量优化

建荣周[a]，昆何*[a]，炯志郑[a] 和楚敏李[b]

[a]华中科技大学计算机科学与技术学院，中国武汉，*80039*[b]*MIS*，皮卡第儒勒·凡尔纳大学，法国亚眠，*430074*

arXiv:2303.02650v2 [cs.CG] 8 Mar 2023

文章信息

关键词：全局优化 启发式方法 等圆装填 几何批量优化 解空间探索与下降

摘要

圆内等圆装箱问题是一个经典且著名的装填问题，在学术界得到了充分研究，并在工业界拥有多种应用。该问题在计算上颇具挑战性，文献中的研究者主要关注圆形物品数量 n 少于320的小规模实例。在本文中，我们旨在解决大规模的这一问题。具体而言，我们提出了一种新颖的几何批量优化方法，该方法不仅能够显著加快连续优化的收敛过程，还能降低程序运行期间的内存需求。随后，我们提出了一种称为解空间探索与下降的启发式搜索方法，能够在大规模情况下高效地发现可行解。此外，我们提出了一种自适应邻居对象维护方法，用于维护连续优化过程中所采用的邻居结构。通过这种方式，我们能够在合理的计算时间内，为大规模实例找到高质量解。在从 $n = 300$到1,000中抽样得到的基准实例上进行的大量实验表明，我们提出的算法优于最先进算法，并且在大规模实例上表现出色。特别是，我们的算法在21个经过充分研究的中等规模实例中发现了10个改进解，并在101个抽样的大规模实例中发现了95个改进解。此外，我们的几何批量优化、启发式搜索和自适应维护方法具有通用性，可适用于其他装填问题和连续优化问题。

1. 引言

装填问题是一类典型的优化问题，旨在将一组几何对象装入一个或多个容器中，目标是找到尽可能紧密的配置，或找到所需容器（箱子）数量尽可能少的方案。装填问题有着丰富的研究历史，已提出许多变体，也发表了大量用于求解此类问题的研究和方法，例如圆填充（Huang 和 Ye，2011年；He、Ye、Wang 和 Liu，2018年；Lai、Hao、Yue、Lü 和 Fu，2022年）、球体装填（Hartman、Mazá˘c 和 Rastelli，2019年；Hifi 和 Yousef，2019年）、正方形装填（Leung、Tam、Wong、Young 和 Chin，1990年；Fekete 和 Hoffmann，2017年）、立方体装填（Miyazawa 和 Wakabayashi，2003年；Epstein 和 van Stee，2005年）、不规则装填（Leao、Toledo、Oliveira、Carravilla 和 Alvarez-Valdés，2020年；Zhao、Jiang 和 Teo，2020年；Rao、Wang 和 Luo，2021年）以及装箱（Baldi、Manerba、Perboli 和 Tadei，2019年；Zhao、She、Zhu、Yang 和 Xu，2021年；He、Tole、Ni、Yuan 和 Liao，2021年）等。

作为最受欢迎的装填问题之一，圆装箱问题（CPP）在数学和计算机科学领域得到了广泛研究。给定 n 个半径固定的圆形物品，CPP旨在将所有 n 个物品装入容器中，使所有物品完全位于容器内且彼此不重叠，同时将容器尺寸降至最小。具体而言，如果容器是正方形，则目标是使方形容器的边长最小；如果容器是圆形，则目标是使圆形容器的半径最小。基于不同几何容器，还有其他圆装箱问题的变体（López 和 Beasley，2011年）。

CPP 在许多工业应用中都有用武之地，例如设施布局、圆柱体装箱、圆形切割、容器装载、仪表板布局（Castillo、Kampas 和 Pintér，2008年）、结构设计（Yanchevskyi、Lachmayer、Mozgova、Lippert、Yaskov、Romanova 和 Litvinchev，2020年）以及卫星封装（Wang、Wang、Sun、Huang 和 Zhang，2019年）。此外，CPP 的求解方法还可应用于数据可视化和数据分析（Wang、Wang、Dai 和 Wang，2006年；Murakami、Higo 和 Kusumoto，2015年；Görtler、Schulz、Weiskopf 和 Deussen，2017年）。另一方面，

*通讯作者。电子邮箱：brooklet60@hust.edu.cn
ORCID：

CPP 已被证明是 NP难问题（Demaine、Fekete 和 Lang，2010年），因此求解这一问题在计算上颇具挑战性。尤其是，CPP 有一个特殊变体，称为圆内等圆装箱（PECC），其目标是将 n 个单位圆装入半径尽可能小的圆形容器中。由于 PECC 具有代表性且形式简单，它已成为 CPP 问题族中最著名、研究最充分的问题，吸引了众多研究者投入大量精力。

由于 CPP 是 NP难问题，随着待打包圆形物品数量增加，获取高质量配置所需的计算资源和难度呈指数级增长，因此求解大规模 CPP 实例极其困难。因此，大多数 CPP 相关研究都聚焦于求解小规模或中等规模实例，只有少数工作致力于求解大规模实例。本文研究 CPP 的 PECC变体。基于经典弹性模型（Huang 和 Xu，1999年；He 等人，2018年）（也称为准物理准人类模型，QPQH），我们提出了一种几何批次优化（GBO）方法，可在大规模实例中构造可行解，或构造重叠尽可能少的不可行解。与现有方法不同，GBO 将待打包圆划分为若干几何批次。随后，在非凸连续优化过程中，GBO 交替更新各批次中的待打包圆，而不是更新所有圆。这样，GBO 降低了时间和空间复杂度，不仅显著加快非凸连续优化的收敛过程，还能减少程序运行期间的驻留内存需求。

此外，我们提出了一种自适应邻居对象维护（ANM）方法，用于在连续优化过程（He 等人，2018年）中维护邻居结构。ANM 使用两个变量“*counter*”和“*deferring length*”来实现自适应特性。当布局发生显著变化时，ANM 会在每次迭代中维护邻居；否则，ANM 会推迟维护过程。我们的 ANM 方法能够解决现有方法（He 等人，2018年；Lai 等人，2022年）的一些问题和缺点，并可用于解决动态装填问题或在线装填问题。

最后，我们基于 GBO 模块提出了一种先进的局部搜索启发式算法，称为解空间探索与下降（SED），用于求解 PECC问题。SED 对当前解进行扰动，以获得若干个扰动后的候选解。随后，SED 使用 GBO 模块最小化每个候选解的重叠，并从这些候选解中选取重叠最小的解来替换当前解。通过反复执行 SED 启发式算法过程，该算法能够高效地找到高质量解。

实验结果表明，我们的 GBO方法显著加快了非凸连续过程的收敛速度，并降低了内存消耗。具体而言，与典型的非批处理方法相比，在规模为 $n = 500$ 至 1,000 的大规模实例上，GBO 将收敛时间缩短了 32.74% 至 54.11%，并将运行时常驻内存降低了 36.00% 至 39.04%。将 GBO模块整合到 SED 中，以求解从 $500 \leq n \leq 1{,}000$ 中选取的 101 个大规模实例（包括 51 个规则数和 50 个非规则数）后，我们的方法改进了 Packomania 网站（Specht，2022年）报告的 95 个实例的已知最佳解。此外，我们的方法在中等规模实例上也表现高效：在 21 个经过充分研究的中等规模实例中，它改进了 10 个实例的已知最佳解，$300 \leq n \leq 320$这些实例由名为 IDTS 的最先进算法求解（Lai 等人，2022年）。通过这些实验，GBO 在求解大规模装箱实例方面展现出相较于现有方法的明显优势，而 SED 是一种高效的启发式算法。

本研究的主要贡献总结如下：

- 我们提出了一种新颖的方法——几何批次优化（GBO），将圆划分为若干几何批次，从而显著加快计算速度。
- 我们提出了一种用于求解 PECC问题的邻居结构自适应邻居对象维护（ANM）方法。
- 我们提出了一种高效的解空间探索与下降（SED）启发式算法，并将 GBO 作为子模块用于求解 PECC问题。
- 对从 $n = 500$ 到 1,000 的大规模实例以及从 $n = 300$ 到 320 的中等规模实例进行的大量实验表明，我们提出的算法性能优异、效率出色，并在许多实例上获得了新的最优解。

本文其余部分安排如下。第2 节介绍 CPP 和 PECC问题的相关研究，包括标志性模型、标志性启发式算法，以及近期求解 CPP 和 PECC 问题的研究。

第3 节介绍 PECC 问题的数学公式，以及本文采用的著名且强大的弹性模型（QPQH）。第4 节介绍连续优化方法，包括我们提出的 GBO方法、容器调整方法和 ANM方法。这些都是最终启发式算法中采用的核心模块。第5 节介绍离散优化方法（即 SED 启发式算法），这是我们用于求解 PECC问题的最终算法。第6 节介绍我们提出的算法与已知最佳结果、最先进算法的对比实验结果及参数研究。最后得出结论。

2. 相关工作

本节简要回顾 CPP 相关研究，包括一些典型研究、标志性模型和标志性启发式算法，随后回顾 PECC 问题的近期研究。

早期解决 CPP 的研究主要聚焦于通过数学分析求解 PECC 变体，旨在为小规模实例寻找并证明最优解。自 1967年起，Kravitz（1967年）首先给出了 $2 \le n \le 19$ 的解。随后，Graham 和 Peck（1968年）证明了 $2 \le n \le 7$ 的最优性。1969年，Pirl（1969年）进一步证明了 $2 \le n \le 10$ 的解的最优性，并给出了 $11 \le n \le 19$ 的解。1971年，Goldberg（1971年）改进了 $n = 14$ 的解，16,17 并进一步给出了 $n = 20$ 的解。1975年，Reis（1975年）给出了 $21 \le n \le 25$ 的解。Melissen（1994年）证明了 $n = 11$ 的最优性，而 Fodor（1999年、2000年、2003年）证明了 $n = 12$ 的最优性，13,19。总而言之，$2 \le n \le 13$ 和 $n = 19$ 的最优性已得到证明。对于更大规模的实例，利用数学分析寻找并证明一个解的最优性十分困难。此后，大多数研究致力于设计高效的启发式算法。

离散优化模型是求解 CPP 的常用技术之一，其思路是将圆逐个装入容器。如果存在一种放置顺序，使所有圆都能装入容器，则问题便得到解决。圆形放置启发式算法对该模型至关重要，因为它会直接影响放置策略和求解性能。针对不等圆装填问题，Huang、Li、Jurkowiak、Li 和 Xu（2003年）以及 Huang、Li、Li 和 Xu（2006年）提出了一种经典圆形放置启发式算法——角点占据放置法（COP），并提出了一种用于评估放置动作的度量函数——最大空穴度（MHD）。COP 要求每个当前放置的圆与两个已装填圆或容器边界相切，且不得与其他圆重叠；MHD 函数则可衡量圆形放置候选动作的质量。基于 COP 启发式算法和 MHD 函数，已有多项后续研究致力于求解其他变体问题或提升算法效率：Huang、Li、Akeb 和 Li（2005年）采用该启发式算法，在矩形容器中装填圆；Lü 和 Huang（2008年）将剪枝增强 Rosenbluth 方法（PERM）用于提升不等圆装填问题的求解性能；Akeb、Hifi 和 M’Hallah（2009年、2010年）进一步采用束搜索和自适应束搜索算法，以提升不等圆装填问题的求解性能；Chen、Tang、Song、Zeng、Peng 和 Liu（2018年）则提出了一种用于求解 PECC 问题的贪心启发式算法。此外，还有其他用于求解装填问题的圆形放置启发式算法，例如左下填充法（Martello、Monaci 和 Vigo，2003年；Burke、Hellier、Kendall 和 Whitwell，2006年）以及最佳局部位置（Hifi 和 M’Hallah，2004年；Hifi 和 M’Hallah，2007年）。

然而，由于离散优化模型的特点，在中等规模或大规模实例上获得稠密解需要耗费大量计算时间。因此，大多数研究人员倾向于采用非凸连续优化模型来求解中等规模的 CPP。受物理模型启发，研究人员提出了一些准物理模型来求解 CPP。Graham、Lubachevsky、Nurmela 和 Östergård (1998) 设计了分子排斥模型和台球模型，用于求解 PECC问题，这些模型也被用于求解其他装填问题变体（Nurmela 和 Östergård，1997年）。一些研究致力于基于弹性模型设计高效的局部搜索启发式算法，该模型称为准物理准人类（QPQH），由 Huang 和 Xu (1999年) 提出，用于求解 CPP 及其变体： Huang 和 Ye (2011年) 提出了一种基于吸引力模型的盆地跳跃启发式算法，用于求解 PECC问题；He 等人 (2018年) 提出了一种收缩盆地跳跃启发式算法，用于求解 PECC问题；Lai 等人 (2022年) 采用迭代动态阈值搜索方法求解 PECC问题；He、Mo、Ye 和 Huang (2013) 采用弹性模型求解带平衡约束的圆装填问题，而 Liu、Zhang、Yao、Xue 和 Guan (2016年) 采用弹性模型求解加权圆装填问题。

此外，还有其他求解 CPP 的方法和启发式方法，例如单调盆地跳跃和种群盆地跳跃启发式方法（Addis、Locatelli 和 Schoen，2008；Grosso、Jamali、Locatelli 和 Schoen，2010）、模拟退火方法（Hifi、Paschos 和 Zissimopoulos，2004）、基于遗传算法的方法（Hifi 和 M’Hallah，

2004）、禁忌搜索方法（Carrabs、Cerrone 和 Cerulli，2014）、基于非线性规划的方法（Mlade-novi´c、Plastria 和 Uroševi´c，2005；Birgin 和 Sobral，2008；Stoyan 和 Yaskov，2014）、混合策略（Stoyan、Yaskov、Romanova、Litvinchev、Yakovlev 和 Cantú，2020）等。

著名的 Packomania 网站（Specht，2022年）由 Specht 维护，展示了许多圆填充问题，并记录了这些问题目前已知的最佳解。根据解的数量、解的规模以及近期更新记录，我们发现，PECC 问题和矩形内圆填充问题都很受关注且研究充分；近期用于求解这两个问题的算法与上述方法相似，这表明这些方法和算法对于不同的圆填充问题及其变体具有良好的普适性和通用性。

根据 Packomania 上 PECC 的近期更新记录 $n \geq 100$，我们发现目前已知的最佳记录如下：Huang 和 Ye（2011年）拥有 $100 \leq n \leq 200$ 的若干个最佳已知解；Cantrell 通过未发表的方法拥有 $110 \leq n \leq 1039$ 的若干个最佳已知解；Donovan 通过未发表的方法拥有 $n = 85$、109 和 121 的三个最佳已知解；Lai 等人（2022年）拥有$126 \leq n \leq 319$的若干个最佳已知解；Stoyan 等人（2020年）拥有 $1077 \leq n \leq 5000$ 的 16 个最佳已知解；其余最佳已知解则由 Specht 通过未发表的方法获得。总之，基于弹性模型（QPQH）的方法（Huang 和 Ye，2011年；Lai 等人，2022年）以及混合策略方法（Stoyan 等人，2020年）可视为求解 PECC 问题的最先进算法。

3. 预备知识

3.1. 问题表述

PECC问题旨在将 n 个单位圆 $\{c_1, c_2, ..., c_n\}$ 装入半径尽可能小的圆形容器中，同时满足两个约束条件：(I) 任意两个圆不重叠；(II) 任意一个圆都不超出容器。该问题可在笛卡尔坐标系下表述为一个非线性约束优化问题：

$$\begin{aligned} \text{Minimize} \quad & R \\ s.t. \quad & \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq 2, \quad 1 \leq i, j \leq n,\ i \neq j, && (1) \\ & \sqrt{x_i^2 + y_i^2} + 1 \leq R, \quad 1 \leq i \leq n, && (2) \end{aligned}$$

其中，R 是以原点 $(0,0)$为圆心的圆形容器的半径，单位圆 c_i 的圆心位于(x_i, y_i)。公式（1）和（2）分别对应约束条件 (I) 和 (II)。

3.2. PECC问题的弹性模型（QPQH）

弹性模型（Huang 和 Xu，1999年；He 等人，2018年；Lai 等人，2022年）可视为 PECC问题的一种松弛形式，因为该模型认为每个圆都是弹性的。基于该模型的算法首先将所有圆放入容器中，此时圆与圆之间以及圆与容器之间可能存在重叠，并使用度量函数量化重叠程度。随后，该算法的主要工作是最小化度量函数的值（即最小化重叠面积）。通过这种方式，弹性模型将 PECC问题转化为一个无约束的非凸连续优化问题。

定义3.1 (重叠距离**)**。 两个单位圆 c_i 和 ^{c}*j*的重叠距离记为 d_{ij}，定义如下：

$$d_{ij} = \max\left(0, 2 - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right), \tag{3}$$

圆 c_i 到容器的重叠距离记为 d_{i0}，定义如下：

$$d_{i0} = \max\left(0, \sqrt{x_i^2 + y_i^2} + 1 - R\right).$$

图1 展示了两类重叠相互冲突的示例。根据弹性理论，两个弹性物体之间的弹性势能与嵌入距离的平方成正比，因此弹性能定义如下。

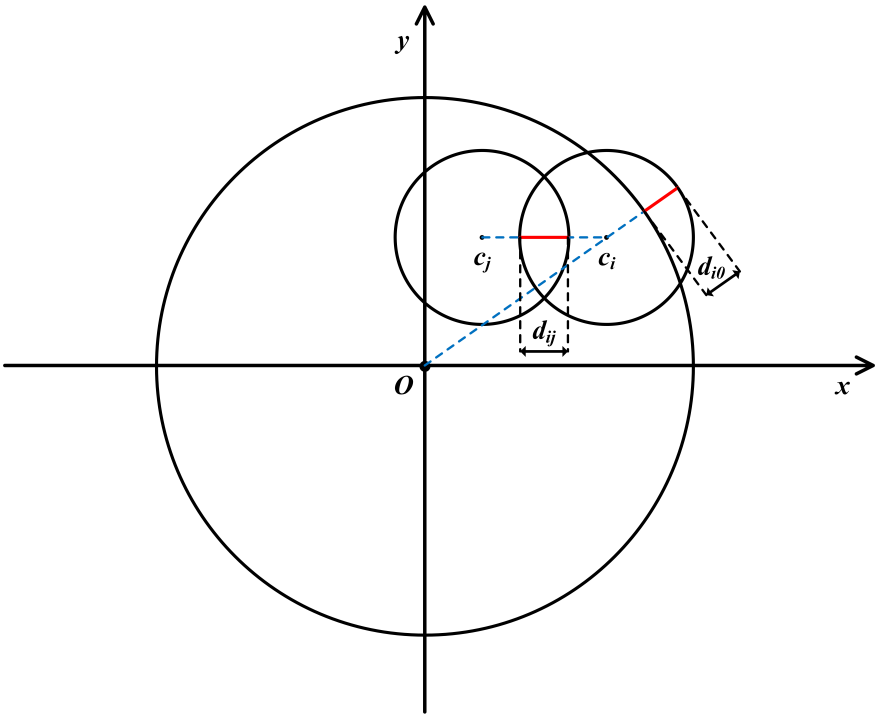


图1： 两种重叠类型的冲突示例示意图，其中 d_{ij} 表示圆-圆重叠距离， d_{i0} 表示圆-容器重叠距离。

定义 3.2（弹性能）。 PECC系统的总弹性能 E 定义如下：

$$E_R(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{ij}^2 + \sum_{i=1}^{n} d_{i0}^2, \tag{4}$$

其中， $\boldsymbol{x} = [x_1, y_1, x_2, y_2, ..., x_n, y_n]^{\mathrm{T}}$ 是一个向量， $\boldsymbol{x} \in \mathbb{R}^{2n}$表示一个候选解， R 是容器半径。

请注意，在 PECC 问题的弹性模型中，能量用于量化重叠程度。当能量 E，即式（4）等于零时，式（1）和（2）成立，且该解对于 PECC 问题是可行的。由于本文其余部分中容器的半径 R 是固定的，因此为便于阅读，我们省略 R 在 $E_R(\boldsymbol{x})$ 中的下标，并在本文其余部分将其简记为 $E(\boldsymbol{x})$ 。

4. 连续优化

由于我们采用弹性模型求解 PECC问题，目标转为最小化能量E ，以找到可通过连续优化算法求解的可行解。本节将介绍：1）我们提出的几何批次优化（GBO）方法，用于在固定容器半径下最小化冲突解的能量；该方法可视为一种批量非凸连续优化算法；2）通过优化容器半径来获得半径最小的可行解 R的方法；3）我们提出的自适应邻居对象维护方法（ANM），该方法能够在连续优化过程中自适应且高效地维护邻居结构；最后，4）我们对所提出的GBO方法进行复杂度分析。

4.1. 求解 PECC问题的GBO方法

设 k 为一个超参数。给定一个不可行解，其中已有 n 个弹性单位圆被强制装入容器，GBO 首先将这些圆均匀划分为 k 个互不相交的集合： $B_1, B_2, ..., B_k$，具有以下

算法1： GBO（$\boldsymbol{x}$, R）

输入：候选解 $\boldsymbol{x}$；以及固定容器半径 R
输出：局部最小解 $\boldsymbol{x}^*$

```
(x_1, x_2, ..., x_k) ← partition(x)
(H_1, H_2, ..., H_k) ← (I, I, ..., I)
cnt ← 0, len ← 1
4：构造当前邻域 Γ
对于 t 从 1 到 MaxIter 执行
    对于 p 从 1 到 k 执行
        计算 g(x_p)
        使用 α_p 公式（7）计算
        x_p ← x_p − α_p H_p g(x_p)
        更新 H_p 使用公式（8）和（9）
    结束循环
    (cnt, len, Γ) ← ANM(cnt, len, Γ)
    如果 Σ_{p=1}^{k} ||g(x_p)||_2 ≤ 10^{-12} 则
        中断
    结束 if
结束 for
x* ← merge(x_1, x_2, ..., x_k)
返回 x*
```

性质：

$$B_p \cap B_q = \emptyset, \quad 1 \le p,\ q \le k,\ p \ne q$$

$$\bigcup_{p=1}^{k} B_p = \{c_1, c_2, \dots, c_n\}.$$

每个圆的子集均被视为一个批次。批次 B_p 中的圆可表示为向量 $\boldsymbol{x}_p$，且批次 B_p 中圆的能量$E(\boldsymbol{x}_p)$ 可改写如下：

$$E(\boldsymbol{x}_p) = \sum_{i \in B_p} \sum_{j=1}^{n} d_{ij}^2 [j \notin B_p \vee j < i] + \sum_{i \in B_p} d_{i0}^2, \tag{5}$$

其中 $[]$是艾弗森括号，当 $[P] = 1$条件语句 P为真时取值为真，否则 $[P] = 0$。语句“$j \notin B_p \vee j < I$”保证批次 B_p 中的圆-圆重叠只计算一次。我们采用 Broyden–Fletcher–Goldfarb–Shanno（BFGS）算法（Ren-Pu 和 Powell，1983年）作为基本优化方法，通过迭代降低能量 $E(\boldsymbol{x}_p)$，具体如下：

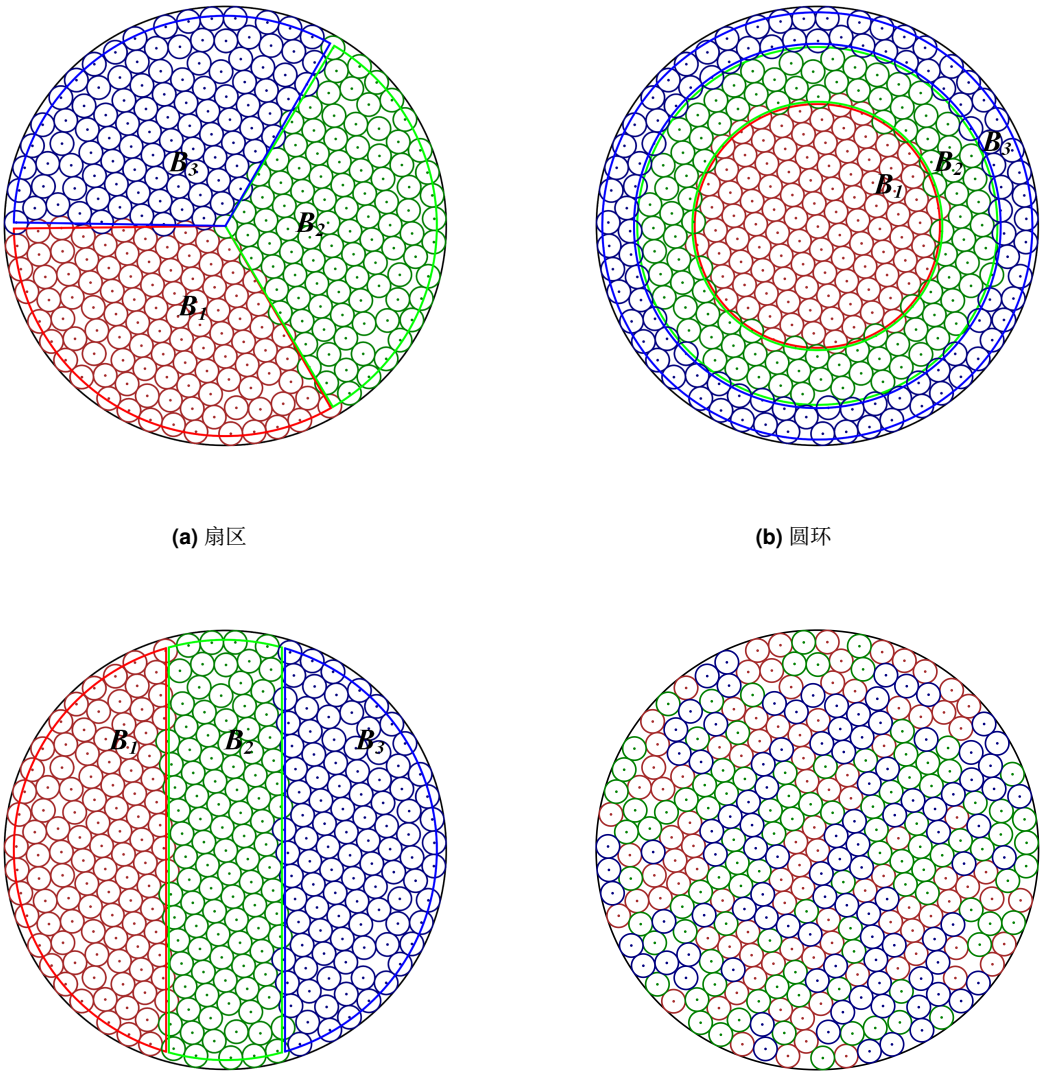
$$\boldsymbol{x}_p^{(t+1)} \leftarrow \boldsymbol{x}_p^{(t)} - \alpha_p^{(t)} \boldsymbol{H}_p^{(t)} g(\boldsymbol{x}_p^{(t)}), \tag{6}$$

其中，上标 t 表示第 t次迭代时的算法，$\boldsymbol{H}_p^{(t)}$是近似 Hessian 矩阵 $\left[\nabla^2 E(\boldsymbol{x}_p^{(t)})\right]^{-1}$，$\alpha$ 是通过线搜索方法获得的步长，函数 $g(\boldsymbol{x})$ 是能量的梯度，即 $\nabla E(\boldsymbol{x})$。$\boldsymbol{H}_p^{(t+1)}$的更新和 $\alpha_p^{(t)}$的计算如下所示：

$$\alpha_p^{(t)} = \underset{\alpha \in \mathbb{R}^+}{\arg\min}\ E\left(\boldsymbol{x}_p^{(t)} - \alpha \boldsymbol{H}_p^{(t)} g(\boldsymbol{x}_p^{(t)})\right), \tag{7}$$

$$\boldsymbol{u}_p^{(t)} = \boldsymbol{x}_p^{(t+1)} - \boldsymbol{x}_p^{(t)}, \quad \boldsymbol{v}_p^{(t)} = g(\boldsymbol{x}_p^{(t+1)}) - g(\boldsymbol{x}_p^{(t)}), \quad \beta = (\boldsymbol{v}_p^{(t)})^{\mathrm{T}} \boldsymbol{u}_p^{(t)}, \tag{8}$$

$$\boldsymbol{H}_p^{(t+1)} = \left(\boldsymbol{I} - \frac{\boldsymbol{u}_p^{(t)} (\boldsymbol{v}_p^{(t)})^{\mathrm{T}}}{\beta}\right) \boldsymbol{H}_p^{(t)} \left(\boldsymbol{I} - \frac{\boldsymbol{v}_p^{(t)} (\boldsymbol{u}_p^{(t)})^{\mathrm{T}}}{\beta}\right) + \frac{\boldsymbol{u}_p^{(t)} (\boldsymbol{u}_p^{(t)})^{\mathrm{T}}}{\beta}. \tag{9}$$



(a) 扇区　　**(b)** 圆环

(c) 栅栏　　**(d)** 随机

图 2： GBO方法的四种划分示意图。示例包含 300 个圆（$n = 300$），均匀划分为三个批次（$k = 3$），分别以三种不同颜色标示，每个批次包含 100 个圆。

这样，PECC问题便转化为一个固定容器半径下的无约束非凸连续优化问题。我们提出的 GBO算法的伪代码见算法 1。给定一个候选解 $\boldsymbol{x}$、固定容器半径 R，算法 GBO($\boldsymbol{x}, R$) 返回一个解 $\boldsymbol{x}^*$，使能量 $E(\boldsymbol{x}^*)$达到局部最小值。为此，GBO 首先将 n 个单位圆划分为 k 个批次（第 1 行），然后 GBO 使用 BFGS方法在每个批次上迭代地最小化能量（第 6–11 行）；当达到最大迭代步数（第 5 行），或 L_2 范数的 k批次梯度之和足够小（第 13–15 行）时，迭代过程终止。最后，GBO 返回一个通过合并 $\boldsymbol{x}^*$ 个批次中的圆得到的局部最小解 k（第 17 行）。尽管很难对所提出的 GBO算法的收敛性进行数学分析，但我们观察到，只要将最大迭代步数设得足够大，算法总能满足收敛条件（第 13 行）。

为使 GBO方法的总计算复杂度最小化，每个批次包含数量几乎相同的圆。具体而言，每个批次的大小为 $\lfloor \frac{n}{k} \rfloor$ 或 $\lceil \frac{n}{k} \rceil$。我们提供了几种用于求解 PECC问题的几何-$k$批次划分策略，描述如下：

- **1) 扇区（默认）**：按照角度 $\theta_i = \arctan \frac{y_i}{x_i}$ 对容器中的所有圆进行升序排序，然后依次将其划分为 k 个子集。
- **2) 圆环**：按照距离 $dis_i = \sqrt{x_i{}^2 + y_i{}^2}$对容器中的所有圆进行升序排序，然后依次将其划分为 k个子集。
- **3) 栅栏**：按照坐标 x_i 对容器中的所有圆进行升序排序，然后依次将其划分为 k 个子集。
- **4) 随机**：将所有圆随机划分为 k组。

我们的实验表明，扇区划分优于其他划分，因此本文将扇区设为默认划分。图2 展示了 GBO 方法的上述四种划分。

4.2. 调整容器的优化方法

从随机布局开始（所有圆都随机放置在容器中，且允许重叠），应用 GBO 方法可得到一个局部最小解，但该解可能包含重叠，因而不可行。获得可行解的一种策略是扩大容器半径并调整布局，直至消除所有重叠。实现这一目标最直观、最常用的方法是采用二分搜索（Huang 和 Ye，2011年；He 等人，2018年）。最近，文献（Lai 等人，2022年提出了一种智能且显著更快的方法，可找到容器半径为局部最小值的可行解 R，具体如下。

设向量 $z = [x_1, y_1, x_2, y_2, ..., x_n, y_n, R]^{\mathrm{T}}$, $z \in \mathbb{R}^{2n+1}$ 是一个候选解，容器半径 R 为变量。新的弹性能 U 可重写如下：

$$U(z, \lambda) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{ij}^2 + \sum_{i=1}^{n} d_{i0}^2 + \lambda R^2, \tag{10}$$

其中，λR^2 是惩罚项，λ 是惩罚系数。通过采用优化方法最小化能量 U，当 λ 增大时，容器半径倾向于缩小；当 λ 减小时，容器半径则倾向于扩大。因此，该模型将当前目标转化为无约束连续优化问题。通过调整 λ，在最小化能量 U 时，可以获得容器半径达到局部最小值的可行解。

我们采用（Lai 等人，2022年）调整容器半径的主要思路，并控制 λR^2 以最小化容器尺寸。我们调整容器半径的方法的伪代码如算法2所示。给定一个候选解$(\boldsymbol{x}, R)$，算法2 将系数初始化为 λ 一个根据经验设定的固定值 10^{-4}，并将 $(\boldsymbol{x}, R)$ 结合起来，得到一个新的候选解 $\boldsymbol{z}$；随后，算法执行若干次迭代以获得可行解。在每次迭代中，算法采用 BFGS方法最小化能量 $U(\boldsymbol{z}, \lambda)$，并更新候选解 $\boldsymbol{z}$。然后，算法将 λ 减半，并在下一次迭代中继续调整候选解 $\boldsymbol{z}$ 。经过若干次迭代后，能量 $U(\boldsymbol{z}, \lambda)$ 收敛至0，因此重叠足够小，候选解 $\boldsymbol{z}$ 可视为容器半径达到局部最小值的可行解。最后，该方法返回解 $(\boldsymbol{x}^*, R^*)$ 作为结果。

注意，经过足够的迭代步数后，λR^2 这一项足够小。该算法在容器半径不固定的情况下，迫使能量$E(\boldsymbol{x}) = 0$（U 退化为 E，不含惩罚项 λR^2）收敛。

4.3. 用于优化的圆的邻居结构

He 等人 (2018年) 首先提出了一种高效的邻居结构，该结构可以高效地计算候选解的能量 E 和U 及其梯度函数，最先进算法 IDTS（Lai 等人，2022年）也采用了该方法。在本小节中，我们首先介绍邻居结构。然后，我们提出 ANM方法，并讨论我们的方法相较于现有方法的优势。

令 l_{ij} 表示两个单位圆 c_i 和c_j的圆心之间的距离：

$$l_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

回顾一下， d_{ij} 表示两个单位圆之间的能量 c_i 和c_j ，其定义见公式（3）。显然，$d_{ij} > 0$ 当 $l_{ij} < 2$时成立，而 $d_{ij} = 0$在其他情况下成立。我们使用距离控制超参数 $\Gamma(i)$ ，定义圆 c_i 的邻居 n 为以下单位圆的子集： l_{cut}。

$$\Gamma(i) = \left\{ c_j \mid \forall j : 1 \le j \le n,\ i \ne j,\ l_{ij} < l_{cut} \right\},$$

当 $l_{cut} = 2$ 时，所有与圆 c_i 重叠的圆 $\{c_j\}$ 都包含在 $\Gamma(i)$中。因此，关于圆 c_i 的能量可以通过枚举 c_j 中的圆 $\Gamma(i)$ 来计算，而无需枚举所有打包圆。因此，批量能量 E （式（5））和能量 U （式（10））可重写如下:

$$E(\boldsymbol{x}_p) = \sum_{i \in B_p} \sum_{j \in \Gamma(i)} d_{ij}^2 [j \notin B_p \vee j < i] + \sum_{i=1}^{n} d_{i0}^2,$$

$$U(\boldsymbol{z}, \lambda) = \sum_{i=1}^{n} \sum_{j \in \Gamma(i)} d_{ij}^2 [j < i] + \sum_{i=1}^{n} d_{i0}^2 + \lambda R^2,$$

请注意，陈述“$j \notin B_p \vee j < i$”和“$j < i$”保证圆-圆重叠只计算一次。

然而，当 $l_{cut} = 2$ 时，即使圆在布局中仅发生轻微移动，也需要维护邻居结构，否则无法保证能量和梯度计算的正确性。因此，若适当增大 l_{cut} 的值，邻居中就会包含更多相邻圆，从而无需在圆发生轻微布局移动时维护邻居结构，也能保证正确性。但如果 l_{cut} 设置得过大，邻居中会包含许多不必要的圆，增加能量和梯度计算的耗时。因此，我们根据经验将 $l_{cut} = 4$ 设为一种折中设置。图 3 展示了一个圆的邻居以及 l_{cut} 的不同设置。请注意，通过枚举圆对来计算能量 E 和 U 及其梯度的复杂度为 $O(n^2)$。采用邻居结构后，复杂度可降低至 $O(n)$(He 等人，2018年）。

我们提出的ANM方法如算法3所示。ANM维护三个参数，即计数器 *cnt*、延迟长度 *len*和历史邻居 Γ。初始时，计数器设为0，长度设为1，算法计算初始邻居 Γ （参见算法1第3—4行）。在连续优化过程中，每次迭代后都会调用ANM（见算法1第12行），每当调用ANM时，计数器加1。如果计数器 *cnt* 等于延迟长度 *len*，ANM会重建一个新的邻居 Γ′ ，并将其与历史邻居 Γ进行比较。如果两个邻居 Γ′ 和 Γ 相同，则将计数器设为0，并将延迟长度 *len* 乘以2；否则，将计数器设为0，延迟长度设为1，并用 Γ′更新邻居 Γ 。

ANM延迟维护邻居结构的基本思想基于计数器 *cnt* 和延迟长度 *len*的运行机制。如果布局不稳定且邻居发生变化，ANM会在每次迭代中重建邻居。如果布局稳定，延迟长度 *len* 将呈指数增长，从而使ANM延迟维护邻居。值得注意的是，也可以使用Voronoi图方法构建邻居，从而获得优异的时间复杂度 $O(n \log n)$。在本研究中，我们采用扫描线方法（$O(n\sqrt{n})$）来构建邻居，该方法效率足够高且易于实现。

He 等人（2018年）采用了一种简单的策略来维护邻居，即每 10 次迭代重建一次邻居。然而，当布局稳定时，重建邻居并无必要，而且这会浪费

算法2： 调整_容器($\boldsymbol{x}$, R)

输入：候选解 $\boldsymbol{x}$和容器半径 R
输出：具有局部最小容器半径的可行解 $(\boldsymbol{x}^*, R^*)$

$\boldsymbol{z}^* \leftarrow \text{combine}(\boldsymbol{x}, R)$, $\lambda \leftarrow 10^{-4}$
对于 t 从 1 到 35 **执行**
  采用 BFGS 最小化能量 $U(\boldsymbol{z}^*, \lambda)$
  $\lambda \leftarrow 0.5 \times \lambda$
结束循环
$(\boldsymbol{x}^*, R^*) \leftarrow \text{divide}(\boldsymbol{z}^*)$
返回 $(\boldsymbol{x}^*, R^*)$

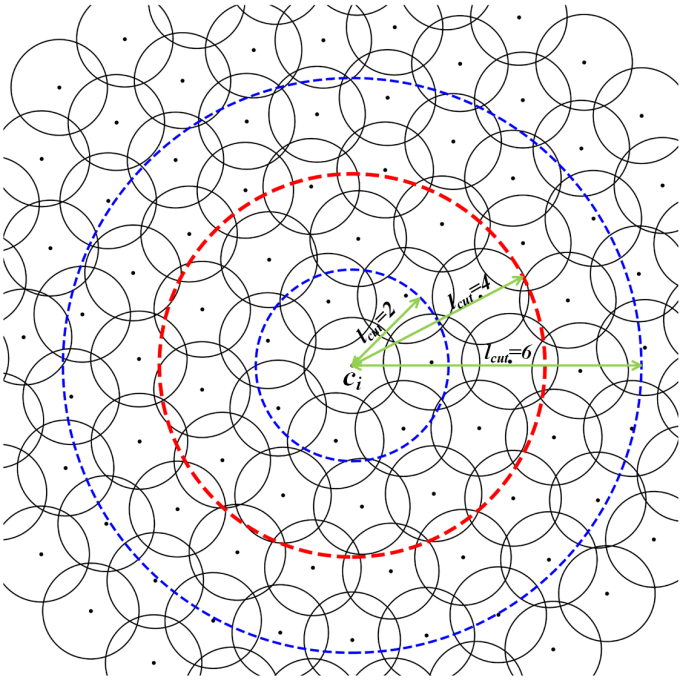


图3： 不同 l_{cut} 设置下一个圆的邻居示意图。此示例展示了冲突布局中的三种设置：$l_{cut}=2$、4 或 6。我们在本研究中通过经验将 $l_{cut}=4$ 设为一种折中设置。

算法3： ANM（cnt，len，Γ）

输入：延迟计数器 cnt；延迟长度 len；邻居 Γ
输出：更新后的计数器 cnt；更新后的长度 len；更新后的邻居 Γ

$cnt \leftarrow cnt+1$
如果 $cnt \geq len$ **则**
  构造当前邻域 Γ'
  如果 $\Gamma \neq \Gamma'$ **则**
    $cnt \leftarrow 0,\ len \leftarrow 1,\ \Gamma \leftarrow \Gamma'$
  else
    $cnt \leftarrow 0,\ len \leftarrow 2\times len$
  结束如果
结束如果
返回 (cnt, len, Γ)

在这种情况下的计算资源。Lai 等人（2022年）提出了一种两阶段策略。在第一阶段，他们通过枚举所有圆对来计算能量和梯度，而不使用邻居结构。当条件

$\|g(\boldsymbol{x})\|_\infty < 10^{-2}$ 满足条件后，他们便进入第二阶段。在第二阶段，他们只在阶段开始时构建邻居结构；随后利用该邻居结构计算能量和梯度，直到算法找到局部最小解。这种两阶段策略有三个缺点：1）第一阶段的枚举方法计算开销很大，尤其是在大规模实例上。2）阈值

条件需要针对不同规模的实例进行微调。3）第二阶段不更新邻居结构，因此，如果解陷入鞍点，就无法保证该策略的正确性。我们的 ANM 方法能够很好地解决这些问题。

值得一提的是，我们的 ANM 是一种自适应方法，能够处理装箱物品的数量和半径可变的动态问题，例如在线装箱问题（Hokama、Miyazawa 和 Schouery，2016年；Fekete 和 Hoffmann，2017年；Fekete、von Höveling 和 Scheffer，2019年；Lintzmayer、Miyazawa 和 Xavier，2019年）。

4.4. GBO 的复杂度分析

我们现在 p提供 基于 BFGS 优化方法的 GBO 时间和空间复杂度分析 d.

时间复杂度。我们分析每次迭代的时间复杂度。根据算法1第6至15行可知，每次迭代显然包含6个部分：批次梯度 $g(\boldsymbol{x}_p)$ 计算、批次步长α_p 计算、批次向量$\boldsymbol{x}_p$ 更新、批次海森矩阵 $\boldsymbol{H}_p$ 更新、维护模块ANM，以及 k-批次梯度范数之和。由于我们采用了高效的邻居结构（He等人，2018年）（见第4.3节），能量函数和梯度函数的时间复杂度均为 $O(n)$。因此，批次梯度计算、步长计算以及 k-批次梯度范数求和的时间复杂度分别为 $O(\binom{n}{k})$、 $O(\beta\binom{n}{k})$和$O(n)$，其中常数 β近似表示线搜索方法的递归深度。每个批次包含$\binom{n}{k}$个装箱物品，因此批次海森矩阵的规模为 $O((\binom{n}{k})^2)$。批次向量更新（式（6））和批次海森矩阵更新（式（8）和式（9））涉及矩阵与向量乘法，因此二者的时间复杂度均为$O((\binom{n}{k})^2)$。我们在ANM模块中采用扫描线方法构建邻居结构（见第4.3节），其时间复杂度为 $O(n\sqrt{n})$ （采用Voronoi图方法可获得更优的时间复杂度$O(n\log n)$）。最后，每次迭代的时间复杂度为 $O(k(\binom{n}{k}+\beta\binom{n}{k}+(\binom{n}{k})^2)+n+n\sqrt{n})$，可简化为 $O(\beta n+n\sqrt{n}+\binom{n^2}{k})$；若ANM模块延后执行维护过程，则时间复杂度为 $O(\beta n+\binom{n^2}{k})$。

空间复杂度。 GBO方法的内存需求主要用于存储 k 批次海森矩阵。因此，GBO方法的空间复杂度为 $O(k(\binom{n}{k})^2)$，可简化为 $O(\binom{n^2}{k})$。

如上所述，我们的GBO方法的时间和空间复杂度（$O(\beta n+\binom{n^2}{k})$ 和 $O(\binom{n^2}{k})$）低于经典BFGS方法（$O(n^2)$和 $O(n^2)$）。当 $k=1$时，它退化为BFGS方法。

5. 搜索启发式

基于弹性模型求解PECC问题的算法可分为两个阶段。在第一阶段，容器半径是固定的，目标是找到一个可行解，或找到一个重叠尽可能少的不可行解（即能量 E 尽可能小）。在第二阶段，算法扩大

算法4： 求解PECC的框架

输入：若干个单位圆 n；当前已知的最佳容器半径 R_b；截止时间 T_{cut}
输出：具有最小容器半径 $(\boldsymbol{x}^*, R^*)$的可行解

```
x* ← random_solution(n, R_b)
x* ← GBO(x*, R_b)
(x*, R*) ← adjust_container(x*, R_b)
当 time() ≤ T_cut do
    R ← min(R_b, R*)
    x ← SED(n, R)
    (x, R) ← adjust_container(x, R)
    如果 R < R* 则
        x* ← x, R* ← R
    结束如果
结束循环
返回 (x*, R*)
```

或缩小容器半径，以获得容器半径尽可能小的可行解。从一个随机候选解开始，虽然我们可以通过采用GBO方法（第4.1节）完成第一阶段，并采用容器调整方法（第4.2节）完成第二阶段，从而获得可行解。然而，以这种方式获得的解的质量仍不令人满意。

通过充分的实验，我们发现，最终解的质量会直接受到第一阶段所得解的影响。如果第一阶段所得解的能量 $E(\boldsymbol{x})$（对应于重叠面积）较大，那么第二阶段半径调整的扩大量也会较大。另一方面，如果第一阶段找到可行解，那么第二阶段就可以缩小半径。因此，第一阶段发现能量最小的解对于获得最终的高质量可行解至关重要。然而，如第3.2节所述，弹性模型将 PECC问题转化为非凸优化问题，因此极难找到全局最优解。因此，我们提出了一种高效的解空间探索与下降（SED）启发式算法，用于在第一阶段发现能量尽可能小的解。

5.1. 求解 PECC 的框架

我们首先介绍算法框架，其伪代码如算法4所示。

首先，算法采用已知最佳半径（Specht，2022年） R_b 作为初始固定容器半径，并生成随机布局作为初始解（第1行）。每个圆的中心 (x_i, y_i) 满足 $x_i,\ y_i \in \mathcal{U}(-R_b, R_b)$（$\mathcal{U}$ 表示连续均匀分布）。随后，算法采用我们的GBO方法（第4.1节）来最小化能量 $E(\boldsymbol{x}^*)$（第2行），并通过采用容器调整方法（第4.2节）获得初始可行解 $\boldsymbol{x}^*$（第3行）。接下来，算法执行迭代过程，以改进最佳记录解，直至达到截止时间 T_{cut}（第4—11行）。

在每次迭代中，算法将目标半径 R 设为$\min(R_b,\ R^*)$（第5行），其中 R_b 是已知最佳半径 R_b，而 R^* 是目前为止发现的最佳可行半径 R^*。然后，算法采用我们提出的SED启发式算法（第5.2节），以发现能量尽可能低的候选解 $\boldsymbol{x}$（第6行）。随后，算法对该候选解应用容器半径调整方法，计算新的可行解（第7行）。如果新的可行半径小于最佳记录半径，表明找到了更优的可行解，则更新最佳记录解（第8—10行）。最后，算法返回最佳记录解作为结果。

需要注意的是，我们的目标是求解大规模实例，这是一项极具挑战性的计算任务。以已知最佳半径而非近似半径或通过初始方法得到的半径作为基准，可以降低计算难度，快速找到高质量解，并提升算法性能。因此，这是一种高效的快速启动方法。

5.2. 解空间探索与下降启发式算法

我们提出的 SED 启发式算法旨在求解以下问题：给定一个固定容器半径，判断是否存在可行解；这本质上是一个判定 PECC 问题。如果 SED 找到可行解，就立即返回该解；否则，返回搜索过程中能量最小的不可行解。首先，我们定义一个新的度量函数 J，其公式如下：

$$J(\boldsymbol{x}) = \lceil -\log_{10} E(\boldsymbol{x}) \rceil,$$

其中，函数 J 是函数 E 的负对数的向上取整。它将能量 E 映射为一个整数。函数 J 的值用于控制启发式过程中的探索次数。

现在，我们介绍所提出的 SED 启发式算法，如算法 5 所示。从固定容器半径R开始，SED 首先生成一个随机配置作为初始解，并采用 GBO方法最小化初始解的能量$\boldsymbol{x}$（第 1–2 行）。随后，SED 执行迭代过程以寻找可行解（第 4–18 行）。在每次迭代中，SED 通过函数 J 获得一个值 m，将其作为扰动数（第 8 行）；随后创建候选解集 U、$|U| = m$，其中，通过扰动当前操作解 $\boldsymbol{x}$ 并采用 GBO 最小化能量 $E(\boldsymbol{x}')$，得到 U 中的候选解 $\boldsymbol{x}'$（第 9–13 行）。接着，SED 采用 *select* 算子，从集合 U 中选择一个候选解来替换解 $\boldsymbol{x}$。如果新解 $\boldsymbol{x}$ 的能量小于最佳记录解 $\boldsymbol{x}^*$的能量，则更新 $\boldsymbol{x}^*$（第 15–17 行）。如果 $\boldsymbol{x}^*$ 的能量足够小，表明已找到可行解，SED 会立即返回该可行解 $\boldsymbol{x}^*$（第 5–7 行）；否则，当达到最大迭代步数 S_{iter}时，SED 返回迭代过程中能量最小的不可行解。

算法 5： SED(n, R)

输入： n 个单位圆；容器半径为 R
输出： 迄今找到的最小能量解 $\boldsymbol{x}^*$

$\boldsymbol{x} \leftarrow \text{random_solution}(n, R)$
$\boldsymbol{x} \leftarrow \text{GBO}(\boldsymbol{x}, R)$
$\boldsymbol{x}^* \leftarrow \boldsymbol{x}$
对于 i 从 1 到 S_{iter} **do**
  如果 $E(\boldsymbol{x}^*) \le 10^{-25}$ **则**
    中断
  结束如果
  $m \leftarrow \max(1, J(\boldsymbol{x})), \quad U \leftarrow \emptyset$
  对于 j 从 1 到 m **执行**
    $\boldsymbol{x}' \leftarrow \text{perturbing}(\boldsymbol{x})$
    $\boldsymbol{x}' \leftarrow \text{GBO}(\boldsymbol{x}')$
    $U \leftarrow U \cup \{\boldsymbol{x}'\}$
  结束循环
  $\boldsymbol{x} \leftarrow \text{select}(U)$
  如果 $E(\boldsymbol{x}) < E(\boldsymbol{x}^*)$**则**
    $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$
  结束条件判断
结束循环
返回 $\boldsymbol{x}^*$

为了获得扰动后的解 $\boldsymbol{x}'$，我们随机移动操作解 $\boldsymbol{x}$ 中圆的坐标，具体描述如下：$x_i' \leftarrow x_i + r$ 和 $y_i' \leftarrow y_i + r$ $(1 \le i \le n)$，其中 r 是一个随机数，$r \in \mathcal{U}(-0.8, 0.8)$。*select* 算子的策略描述如下：

$$\text{select}(U) = \begin{cases} \arg\min\limits_{\boldsymbol{y} \in U} E(\boldsymbol{y}), & \text{if } \min\limits_{\boldsymbol{y} \in U} E(\boldsymbol{y}) < E(\boldsymbol{x}) \\ P\left(X = \boldsymbol{y} \mid p_{\boldsymbol{y}} = softmax(J(\boldsymbol{y}))\right), & \text{otherwise} \end{cases}$$

$$softmax(J(\boldsymbol{y})) = \frac{\exp(J(\boldsymbol{y}))}{\sum_{\boldsymbol{y}' \in U} \exp(J(\boldsymbol{y}'))}.$$

该算子首先将集合 U 中能量最小的候选解与当前操作解 $\boldsymbol{x}$ 进行比较。如果候选解的能量小于操作解，则该算子用候选解替换操作解；否则，该算子根据 Softmax 函数的概率从集合 U 中选择一个候选解。

6. 实验

在实验中，我们首先评估所提出的 GBO方法在不同批次数量和不同几何划分方法下的性能，然后展示结果之间的比较，并进行参数研究。

6.1. 实验设置

我们的算法采用 C++ 实现，并使用 g++ 5.4.0 编译。实验在一台配备 Intel®Xeon® E5-2650v3 CPU 和 256 GBytes 随机存取存储器的服务器上进行，运行 Linux 操作系统。由于算法具有随机性，我们使用不同的随机种子（CPU 时间戳）多次独立运行算法。为了全面评估算法的性能，我们选取三个实例规模作为基准，具体如下：

• **中等规模**： $300 \le n \le 320$，用于与最先进算法 IDTS（Lai 等人，2022年）进行比较。我们设定 $k = 3$ GBO 的批次数，整体搜索的截止时间设为 12 小时 T_{cut}。算法独立运行 20 次，截止时间设置和运行时间均与 IDTS 工作中的设置相同。

• **大规模 I**： $500 \le n < 800$，用于与已知最佳结果（Specht，2022年）进行比较。我们设置 $k = 5$ 作为批次数，截止时间设为 24 小时 T_{cut}。算法独立运行 10 次。

• **大规模 II**： $800 \le n \le 1000$，用于与已知最佳结果（Specht，2022年）进行比较。我们设置 $k = 5$ 作为批次数，截止时间设为 48 小时 T_{cut}。算法独立运行 10 次。

其余参数统一设置如下。几何划分方法采用扇区（第4.1节），GBO 的最大迭代步数为 $MaxIter = 5000$（第 4.1节），SED 的最大迭代步数为$S_{iter} = 500$（第 5.2节）。参数调优与分析见第 6.5节。

6.2. 中等规模实例的比较

我们在中等规模实例（$300 \le n \le 320$）上运行所提出的 SED（3批次 GBO）算法及其变体 SED（1批次 GBO）。两种算法与已知最佳结果（Specht，2022年）及 IDTS（Lai 等人，2022年）的比较结果见表1。请注意，SED（1批次 GBO）与 SED（3批次 GBO）的区别仅在于批次数量由 $k = 3$变为 $k = 1$，而 1批次 GBO 则退化为经典的 BFGS 方法。

在表1中，n 表示实例中的项数，R^* 表示 Packomania 网站上的已知最佳结果（Specht，2022年）（数据下载日期：2022/10/1），其后列出 IDTS 的最佳结果（Lai 等人，2022年）。SED（1批次 GBO）和 SED（3批次 GBO）对应我们方法的结果：R_{best} 表示 20 次独立运行中的最佳结果，R_{avg} 表示 20 次独立运行的平均结果，$R_{best} - R^*$ 表示 R_{best} 与 R^* 之间的差值（负值表示最佳结果有所改进），RR 表示优于或等于已知最佳结果的比率 R^*，HR 表示达到最佳值的比率 R_{best}，而 $time$ (s) 表示获得最优解的平均时间（秒）。表格底部的“#Improve”、“#Equal”和“#Worse”分别表示，对于我们的两种算法 SED（1批次 GBO）和 SED（3批次 GBO），算法取得优于、等于或劣于 R^* 和 IDTS 的最佳结果的实例数量。

从表1中，我们可以得出以下几点结论：

(1) 我们提出的结合 1批次 GBO（即经典 BFGS）的 SED 启发式算法，在 21 个中等规模实例中取得了 10 个优于 IDTS 的最佳结果、7 个与 IDTS 持平的最佳结果，以及 4 个逊于 IDTS 的最佳结果。这表明 SED 优于 IDTS。值得注意的是，除 $n = 320$ 外，已知最佳结果与 IDTS 相同；而 IDTS 采用 L-BFGS 作为优化方法，这同样是一种拟牛顿法。

(2) SED（3批次 GBO）在 21 个中等规模实例中取得了 6 个更优的最佳结果、8 个持平的最佳结果和 7 个更差的最佳结果。结果表明，3批次 GBO 不如 1批次 GBO（经典 BFGS 方法）。这说明多批次 GBO 在中等规模实例上效果不佳。

(3) 从命中最佳结果 R_{best}（HR）的比率来看，HR 高于 10/20 的比率很少，HR 的比率大多等于 1/20。从 IDTS 的研究中我们还发现，IDTS 研究中 HR 对于 $300 \le n \le 320$ 的所有比率均低于 6/20，且有 14 个 HR 的比率等于 1/20。这些结果表明，中等规模实例已得到充分研究，要获得更优的最佳结果非常困难。

6.3. 大规模实例比较

对于大规模实例，我们选取 $n = 500$、510、520、.....、990、1000 作为规则数的51个大规模实例，然后从 $500 \le n < 800$ 中随机抽取30个非规则数，并从$800 \le n \le 1000$ 中随机抽取20个非规则数，作为非规则数的50个大规模实例。我们在这些实例上运行SED（5批次GBO），规则数和非规则数的实验结果分别如表2 和表3所示。

在表2 和3中，我们提供了 n实例中的项数、R^* Packomania 网站上的已知最佳结果（Specht，2022年）（数据下载日期为2022/10/1）、R_{best} 10次独立运行中的最佳结果、R_{avg}10次独立运行的平均结果。$R_{best} - R^*$ 显示了 R_{best} 与 R^* 之间的差值（负值表示最佳结果有所改进）。RR 显示了结果等于或优于已知最佳结果的比率 R^*，以及 HR

Table 1: Comparison between the best-known results, IDTS and our proposed algorithms SED (1-batch GBO) and (5-batch GBO) on the 21 well-studied moderate scale instances. The improved best results R_{best} found by our proposed algorithms appear in bold.

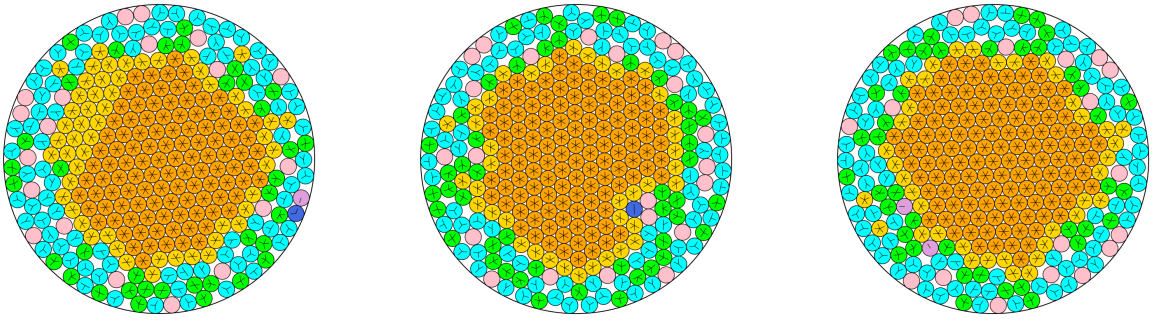
n	R^*	IDST	SED (1-batch GBO)						SED (3-batch GBO)					
			R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$	R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$
300	18.813153706	18.813153706	18.813157576	18.813191071	3.87E-06	0/20	1/20	26216	18.813189941	18.813198345	3.62E-05	0/20	12/20	33237
301	18.843463507	18.843463507	18.843463507	18.844498079	0.00E+00	2/20	2/20	26222	18.843463507	18.843551084	0.00E+00	1/20	1/20	38618
302	18.891782255	18.891782255	**18.891781604**	18.892064228	-6.51E-07	2/20	1/20	20792	18.891782255	18.892033313	0.00E+00	2/20	2/20	23987
303	18.929749153	18.929749153	18.929749153	18.930326363	0.00E+00	1/20	1/20	23549	18.929750618	18.930328723	1.47E-06	0/20	1/20	23652
304	18.964441751	18.964441751	**18.964297557**	18.964819754	-1.44E-04	3/20	1/20	24562	**18.963620323**	18.964664008	-8.21E-04	5/20	1/20	18852
305	19.001754565	19.001754565	**19.001744832**	19.002856526	-9.73E-06	1/20	1/20	24542	**19.001726813**	19.002734687	-2.78E-05	2/20	1/20	21922
306	19.030389407	19.030389407	19.031079983	19.031763629	6.91E-04	0/20	1/20	21131	19.030651242	19.031391719	2.62E-04	0/20	2/20	25226
307	19.060160922	19.060160922	19.061100857	19.062150163	9.40E-04	0/20	1/20	23579	19.060841920	19.061955433	6.81E-04	0/20	1/20	26981
308	19.104991437	19.104991437	19.109083748	19.110722905	4.09E-03	0/20	1/20	26804	19.107239953	19.109820607	2.25E-03	0/20	1/20	23291
309	19.142573165	19.142573165	**19.141335827**	19.143371324	-1.24E-03	5/20	1/20	25662	19.142625039	19.143620434	5.19E-05	0/20	1/20	24278
310	19.178928265	19.178928265	**19.178560517**	19.179857308	-3.68E-04	2/20	1/20	18906	**19.178419320**	19.179442843	-5.09E-04	6/20	1/20	24718
311	19.212365036	19.212365036	**19.210669807**	19.213148503	-7.17E-04	5/20	1/20	22109	**19.210564074**	19.212561780	-8.22E-04	4/20	2/20	21153
312	19.233585653	19.233585653	19.233585653	19.234863780	0.00E+00	1/20	1/20	23934	19.233585653	19.234529499	0.00E+00	4/20	4/20	30386
313	19.257103014	19.257103014	**19.256994660**	19.258015876	-1.08E-04	1/20	1/20	21251	19.257103014	19.258284406	0.00E+00	1/20	1/20	25457
314	19.286195141	19.286195141	**19.286190236**	19.286514591	-4.91E-06	4/20	4/20	25125	**19.286190236**	19.286476954	-4.91E-06	1/20	1/20	26386
315	19.302288067	19.302288067	**19.302273991**	19.302586043	-1.41E-05	17/20	11/20	25517	**19.302273991**	**19.302286175**	-1.41E-05	18/20	3/20	24488
316	19.334041754	19.334041754	19.334041754	19.334805176	0.00E+00	9/20	9/20	25370	19.334041754	19.334584281	0.00E+00	7/20	7/20	22840
317	19.367595672	19.367595672	19.367595672	19.367930543	0.00E+00	10/20	10/20	23713	19.367595672	19.367871121	0.00E+00	9/20	9/20	23129
318	19.391566091	19.391566091	19.391566091	19.391909306	0.00E+00	15/20	15/20	22836	19.391566091	19.391631572	0.00E+00	17/20	17/20	27084
319	19.424277830	19.424277830	19.424277830	19.424920970	0.00E+00	15/20	15/20	18382	19.424277830	19.425310867	0.00E+00	12/20	12/20	24782
320	19.456230764	19.451649630	**19.451583741**	19.453206681	-4.65E-03	20/20	1/20	27713	19.451734176	19.454455527	-4.50E-03	19/20	3/20	30039
#Improve			10						6					
#Equal			7						8					
#Worse			4						7					

Table 2: Comparison between the best-known results and SED (5-batch GBO) on the 51 large scale instances of the regular number. The improved best results of R_{best} and R_{avg} appear in bold.

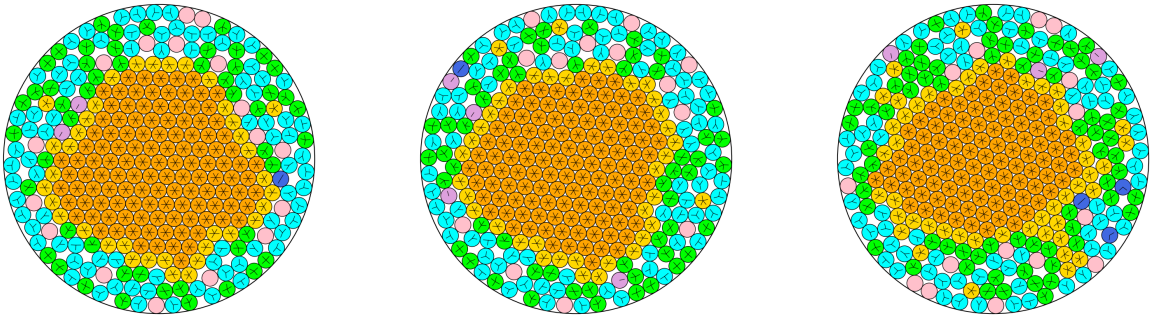
n	R^*	SED (5-batch GBO)						n	R^*	SED (5-batch GBO)					
		R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$			R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$
500	24.1329376240	**24.1312529426**	**24.1313788210**	-1.68E-03	10/10	1/10	39561	800	30.4212133790	**30.4198645288**	30.4250741893	-1.35E-03	2/10	1/10	132389
510	24.4365629292	**24.4210537570**	**24.4249135890**	-1.55E-02	10/10	1/10	60259	810	30.6017659657	**30.5956736421**	**30.6002028281**	-6.09E-03	6/10	1/10	92738
520	24.6609522831	**24.6258073657**	**24.6314124118**	-3.51E-02	10/10	1/10	69426	820	30.7666908826	**30.7489836784**	**30.7533477503**	-1.77E-02	10/10	1/10	125921
530	24.8482376878	**24.8455280225**	**24.8472603022**	-2.71E-03	7/10	1/10	44648	830	30.9418117602	**30.9297617239**	**30.9337956547**	-1.21E-02	10/10	1/10	116582
540	25.0884399378	**25.0855911217**	**25.0877127273**	-2.85E-03	8/10	1/10	45255	840	31.1208576445	**31.1194578886**	31.1224527712	-1.40E-03	2/10	1/10	88791
550	25.3384484709	**25.3342535707**	**25.3357628704**	-4.19E-03	10/10	1/10	43783	850	31.3552353388	**31.3432578820**	**31.3503256930**	-1.20E-02	9/10	1/10	117055
560	25.5167889934	**25.5122424885**	**25.5146796829**	-4.55E-03	9/10	1/10	59827	860	31.5114350783	**31.5070879233**	31.5135835953	-4.35E-03	5/10	1/10	114252
570	25.7224085766	**25.7134847249**	**25.7138542411**	-8.92E-03	10/10	1/10	63852	870	31.6807723726	**31.6747965238**	**31.6790476977**	-5.98E-03	8/10	1/10	125813
580	25.9623218516	**25.9511074848**	**25.9527379554**	-1.12E-02	10/10	1/10	69140	880	31.8536138755	**31.8463147084**	**31.8512855368**	-7.30E-03	8/10	1/10	107682
590	26.2105443770	**26.2025041590**	**26.2069156613**	-8.04E-03	9/10	1/10	62246	890	32.0482429714	**32.0438440391**	32.0485774225	-4.40E-03	6/10	1/10	88801
600	26.4274162694	**26.4176880113**	**26.4216830234**	-9.73E-03	10/10	1/10	58273	900	32.2330843545	**32.2199533426**	**32.2301042219**	-1.31E-02	8/10	1/10	92746
610	26.6310600018	**26.6227736610**	**26.6256150993**	-8.29E-03	10/10	1/10	55976	910	32.3661258161	**32.3658580937**	32.3778653220	-2.68E-04	1/10	1/10	79106
620	26.8618811252	**26.8431235737**	**26.8457652476**	-1.88E-02	10/10	1/10	65063	920	32.5489524357	**32.5347534619**	**32.5391088907**	-1.42E-02	10/10	1/10	121667
630	27.0340036487	27.0525778864	27.0618588183	1.86E-02	0/10	1/10	44900	930	32.7013004786	**32.6957714244**	32.7024754370	-5.53E-03	7/10	1/10	119341
640	27.2419589706	**27.2387282736**	27.2421894917	-3.23E-03	6/10	1/10	59358	940	32.9143189848	**32.9009771761**	**32.9052353405**	-1.33E-02	10/10	1/10	109627
650	27.4458279070	**27.4382286041**	**27.4422058882**	-7.60E-03	10/10	1/10	71570	950	33.1232153862	**33.1032839823**	**33.1093226461**	-1.99E-02	10/10	1/10	136179
660	27.6680891671	**27.6589093888**	**27.6622225342**	-9.18E-03	10/10	1/10	69611	960	33.2729042511	**33.2554924317**	**33.2639325269**	-1.74E-02	10/10	1/10	111880
670	27.9068676439	**27.9008215002**	**27.9030904780**	-6.05E-03	10/10	1/10	65422	970	33.4357278371	**33.4225668260**	**33.4271703375**	-1.32E-02	10/10	1/10	114131
680	28.0951980575	**28.0877536042**	**28.0903964995**	-7.44E-03	10/10	1/10	63392	980	33.6049866471	**33.5827959909**	**33.5885379042**	-2.22E-02	10/10	1/10	154687
690	28.2458655422	**28.2447186303**	28.2638787279	-1.15E-03	5/10	1/10	55853	990	33.7831428326	**33.7627857460**	**33.7698434841**	-2.04E-02	10/10	1/10	132277
700	28.4958443164	**28.4839888638**	**28.4877680865**	-1.19E-02	10/10	1/10	59161	1000	33.9571409147	**33.9457725483**	**33.9500559701**	-1.14E-02	10/10	1/10	115406
710	28.7110433153	**28.6956216568**	**28.7030160501**	-1.54E-02	10/10	1/10	51258								
720	28.8599089374	**28.8547583326**	**28.8594487318**	-5.15E-03	6/10	1/10	49792								
730	29.0380889370	**29.0368792578**	29.0438783072	-1.21E-03	1/10	1/10	39872								
740	29.2501613747	**29.2418887324**	**29.2440743866**	-8.27E-03	10/10	1/10	61698								
750	29.4806882503	**29.4704602348**	**29.4725052490**	-1.02E-02	10/10	1/10	51594								
760	29.6611069657	**29.6529205123**	29.6626249312	-8.19E-03	5/10	1/10	45894								
770	29.8480812041	**29.8415580215**	**29.8451997650**	-6.52E-03	9/10	1/10	47679								
780	30.0188056551	**30.0138757407**	**30.0164282850**	-4.93E-03	9/10	1/10	59540								
790	30.2195970061	**30.2169949810**	30.2208074501	-2.60E-03	4/10	1/10	44975								
#Improve		29	24							21	15				
#Equal		0	0							0	0				
#Worse		1	6							0	6				

Table 3: Comparison between the best-known results and SED (5-batch GBO) on the 50 large scale instances of the irregular number. The improved best results of R_{best} and R_{avg} appear in bold.

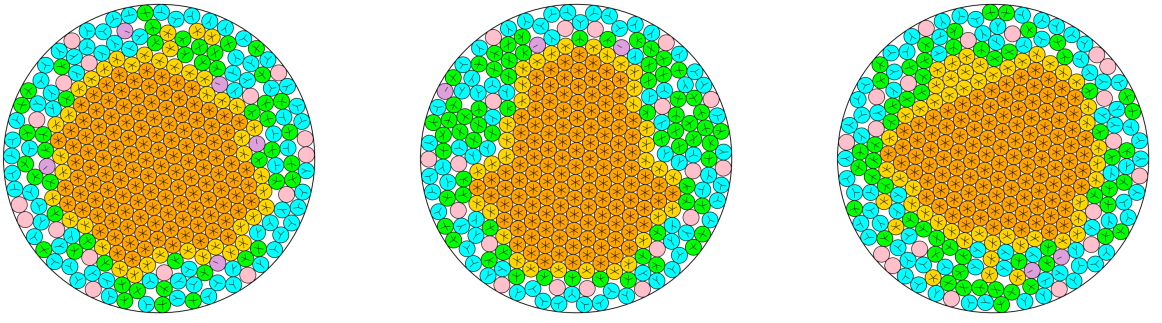
n	R^*	SED (5-batch GBO)						n	R^*	SED (5-batch GBO)					
		R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$			R_{best}	R_{avg}	$R_{best}-R^*$	RR	HR	$time\ (s)$
505	24.2933415273	**24.2888919603**	**24.2901907584**	-4.45E-03	10/10	1/10	41754	818	30.7297559207	**30.7181308589**	**30.7214168182**	-1.16E-02	10/10	1/10	96312
507	24.3570322874	**24.3483441023**	**24.3499568679**	-8.69E-03	10/10	1/10	56666	823	30.7951925938	**30.7907479760**	30.7953689952	-4.44E-03	7/10	1/10	102853
511	24.4539029279	**24.4418315424**	**24.4443899780**	-1.21E-02	10/10	1/10	65166	828	30.9026164659	**30.8916045494**	**30.8945942840**	-1.10E-02	10/10	1/10	129179
513	24.4944643498	**24.4803487672**	**24.4836476996**	-1.41E-02	10/10	1/10	53628	846	31.2210952935	**31.2169756294**	31.2320033828	-4.12E-03	7/10	1/10	77582
515	24.5352933431	**24.5160216754**	**24.5234661213**	-1.93E-02	10/10	1/10	64179	856	31.4404613224	31.4415597563	31.4451742962	1.10E-03	0/10	1/10	93393
517	24.5806822590	**24.5688597754**	**24.5708937360**	-1.18E-02	10/10	1/10	64080	861	31.5340220772	**31.5286770943**	**31.5304575449**	-5.34E-03	10/10	1/10	121885
539	25.0666591245	**25.0644157476**	**25.0649834706**	-2.24E-03	10/10	1/10	61702	872	31.7129332496	**31.7028455621**	**31.7105946951**	-1.01E-02	9/10	1/10	93714
547	25.2486649453	25.2630537774	25.2647995297	1.44E-02	0/10	1/10	39954	873	31.7297370844	**31.7220224328**	**31.7251146977**	-7.71E-03	9/10	1/10	95472
568	25.6768452576	**25.6742746832**	**25.6747142118**	-2.57E-03	9/10	2/10	63844	877	31.8055541388	**31.7939196678**	**31.7991773790**	-1.16E-02	10/10	1/10	123868
578	25.9195920293	**25.9105502154**	**25.9141526168**	-9.04E-03	10/10	1/10	57305	879	31.8366948955	**31.8265208472**	**31.8312690574**	-1.02E-02	10/10	1/10	110296
591	26.2324777614	**26.2247762629**	**26.2300329942**	-7.70E-03	8/10	1/10	63852	888	32.0091101899	**31.9999805643**	**32.0056588693**	-9.13E-03	10/10	1/10	93530
597	26.3542748538	**26.3529416759**	26.3549863877	-1.33E-03	5/10	1/10	48364	892	32.0884743996	**32.0850195925**	32.0888263967	-3.45E-03	5/10	1/10	118031
605	26.5365274699	**26.5233023305**	**26.5261325786**	-1.32E-02	10/10	1/10	72521	899	32.2281486010	**32.2157001450**	**32.2200915147**	-1.24E-02	10/10	1/10	130749
608	26.6012529572	**26.5828030363**	**26.5852665616**	-1.84E-02	10/10	1/10	73642	906	32.3065651824	32.3093777750	32.3220781255	2.81E-03	0/10	1/10	131780
613	26.6970192350	**26.6806757801**	**26.6857073661**	-1.63E-02	10/10	1/10	77051	911	32.3720611668	32.3751529633	32.3910623981	3.09E-03	0/10	1/10	90007
666	27.8222921562	**27.8095070987**	**27.8149840911**	-1.28E-02	10/10	1/10	62523	923	32.6092522621	**32.5891329001**	**32.5927036033**	-2.01E-02	10/10	1/10	117298
669	27.8866609962	**27.8807032094**	**27.8832913945**	-5.96E-03	10/10	1/10	50575	924	32.6279909132	**32.6035557397**	**32.6110620319**	-2.44E-02	10/10	1/10	131299
677	28.0279735048	**28.0223920966**	28.0294202784	-5.58E-03	3/10	1/10	50970	945	33.0363462137	**33.0028997187**	**33.0183695018**	-3.34E-02	10/10	1/10	138869
678	28.0545929509	**28.0452214253**	**28.0486293696**	-9.37E-03	10/10	1/10	70367	964	33.3393817195	**33.3204312890**	**33.3287459497**	-1.90E-02	10/10	1/10	115451
737	29.1885579873	**29.1805602263**	**29.1837249288**	-8.00E-03	10/10	1/10	59552	977	33.5337256998	**33.5256191705**	**33.5286656634**	-8.11E-03	10/10	1/10	117418
741	29.2673369351	**29.2630224724**	**29.2656945026**	-4.31E-03	8/10	1/10	52375								
743	29.2989651471	**29.2950962890**	29.3006429415	-3.87E-03	7/10	1/10	51812								
755	29.5864164492	**29.5781420320**	**29.5813137994**	-8.27E-03	10/10	1/10	66710								
763	29.7207712331	**29.7156118909**	29.7208072730	-5.16E-03	6/10	1/10	47539								
764	29.7349505495	**29.7321782541**	29.7356589694	-2.77E-03	7/10	1/10	42529								
774	29.9175478793	**29.9123498351**	**29.9157783799**	-5.20E-03	9/10	1/10	44943								
778	29.9893439763	**29.9830573225**	**29.9844978472**	-6.29E-03	10/10	1/10	55350								
781	30.0278742024	30.0301280216	30.0358959038	2.25E-03	0/10	1/10	65116								
796	30.3480735601	**30.3399225581**	**30.3447190305**	-8.15E-03	10/10	1/10	49815								
797	30.3755692236	**30.3676919043**	**30.3715277628**	-7.88E-03	9/10	1/10	46237								
#Improve		28	23							17	14				
#Equal		0	0							0	0				
#Worse		2	7							3	6				



(a) $n = 302$ **(b)** $n = 304$ **(c)** $n = 305$



(d) $n = 309$ **(e)** $n = 310$ **(f)** $n = 311$



(g) $n = 313$ **(h)** $n = 314$ **(i)** $n = 315$

图4： 我们提出的算法在中等规模实例上找到的新改进解 ($300 \leq n \leq 320$)。根据接触圆的数量，圆被涂成不同的颜色；当两个圆心之间的距离 $d(c_i, c_j)$ 满足 $d(c_i, c_j) \leq 2 + 10^{-10}$时，两个圆 c_i 和c_j 被认为彼此接触。

显示命中最佳值 R_{best}的比率。 *time* (*s*) 的最后一列显示获得最优解的平均时间。在表格底部，“#Improve”、“#Equal”和“#Worse”分别表示与已知最佳结果相比，SED（5批次GBO）获得改进结果、相同结果和更差结果的实例数量。

根据结果，我们可以得出以下结论：

表4在选取的20个大规模实例上，1批次GBO与5批次GBO的比较。 R'_{best} 和 R_{best} 的最佳结果以及 R'_{avg} 和 R_{avg} 的最佳结果以粗体显示。

n	SED（1批次GBO）			SED（5批次GBO）				
	R'_{best}	R'_{avg}	$time\ (s)$	R_{best}	R_{avg}	$time\ (s)$	$R_{best}-R'_{best}$	$R_{avg}-R'_{avg}$
510	24.4854461416	24.5081091304	48297	**24.4210537570**	**24.4249135890**	60259	-6.44E-02	-8.32E-02
610	26.6992073479	26.7150332110	58763	**26.6227736610**	**26.6256150993**	55976	-7.64E-02	-8.94E-02
700	28.5679045373	28.5899851016	52511	**28.4839888638**	**28.4877680865**	59161	-8.39E-02	-1.02E-01
740	29.2443717879	29.2477209407	56338	**29.2418887324**	**29.2440743866**	61698	-2.48E-03	-3.65E-03
760	29.7452134712	29.7964289470	37391	**29.6529205123**	**29.6626249312**	45894	-9.23E-02	-1.34E-01
780	30.1284909797	30.1454731784	36161	**30.0138757407**	**30.0164282850**	59540	-1.15E-01	-1.29E-01
820	30.8413365443	30.8831339363	87278	**30.7489836784**	**30.7533477503**	125921	-9.24E-02	-1.30E-01
890	32.1297407398	32.1758794016	80371	**32.0438440391**	**32.0485774225**	88801	-8.59E-02	-1.27E-01
930	32.8196370611	32.8494260033	104544	**32.6957714244**	**32.7024754370**	119341	-1.24E-01	-1.47E-01
960	33.3779501763	33.4204902518	100149	**33.2554924317**	**33.2639325269**	111880	-1.22E-01	-1.57E-01
513	24.4833011422	24.4880257346	69720	**24.4803487672**	**24.4836476996**	53628	-2.95E-03	-4.38E-03
568	**25.6742746832**	25.6750224839	58749	**25.6742746832**	**25.6747142118**	63844	0.00E+00	-3.08E-04
608	26.5872589451	26.5914219805	63320	**26.5828030363**	**26.5852665616**	73642	-4.46E-03	-6.16E-03
678	28.0507101923	28.0533386356	49399	**28.0452214253**	**28.0486293696**	70367	-5.49E-03	-4.71E-03
737	29.2746711273	29.3098829156	46403	**29.1805602263**	**29.1837249288**	59552	-9.41E-02	-1.26E-01
774	**29.9122799819**	29.9166270797	46759	29.9123498351	**29.9157783799**	44943	6.99E-05	-8.49E-04
846	31.2194588931	31.2658324893	96767	**31.2169756294**	**31.2320033828**	77582	-2.48E-03	-3.38E-02
877	31.7991261202	31.8019948228	105985	**31.7939196678**	**31.7991773790**	123868	-5.21E-03	-2.82E-03
923	32.5978399965	32.6013970912	128923	**32.5891329001**	**32.5927036033**	117298	-8.71E-03	-8.69E-03
964	33.3301336049	33.3336095372	110927	**33.3204312890**	**33.3287459497**	115451	-9.70E-03	-4.86E-03

(1) 在规则数的大规模实例中，SED（5批次GBO）在51个实例中取得了50个更优、0个持平和1个较差的最佳结果；在不规则数的大规模实例中，它在50个实例中取得了45个更优、0个持平和5个较差的最佳结果。这些结果表明，我们提出的SED（5批次GBO）算法在大规模实例上表现出色。

(2) RR 的比率大多高于 5/10，其中许多等于 10/10。对于规则数的大规模实例，SED（5批次GBO）的平均结果中有 39 个有所改善、0 个持平、12 个变差；对于不规则数的大规模实例，其 50 个实例的平均结果中有 37 个有所改善、0 个持平、13 个变差。这些结果表明，SED（5批次GBO）的许多次运行结果优于已知最佳结果。这也说明该算法在大规模实例上表现出色。

(3) 除了 $n = 568$，HR 的所有比率都等于 1/10。这表明，获得最佳结果极其困难，而且大规模 PECC问题在计算上具有挑战性。

6.4. GBO（多批次）与非批处理的比较

我们进一步进行比较，以评估所提出的GBO模块在大规模实例上的性能。我们从大规模实例中随机选取10个规则数和10个非规则数，然后在这20个选定实例上运行SED（5批次GBO）及其变体SED（1批次GBO）。对于SED（5批次GBO），其批次数从 $k = 5$ 改为 $k = 1$ 。两个算法分别在每个实例上独立运行10次。结果如表4所示。请注意，1批次（非批处理）GBO退化为经典BFGS优化方法。

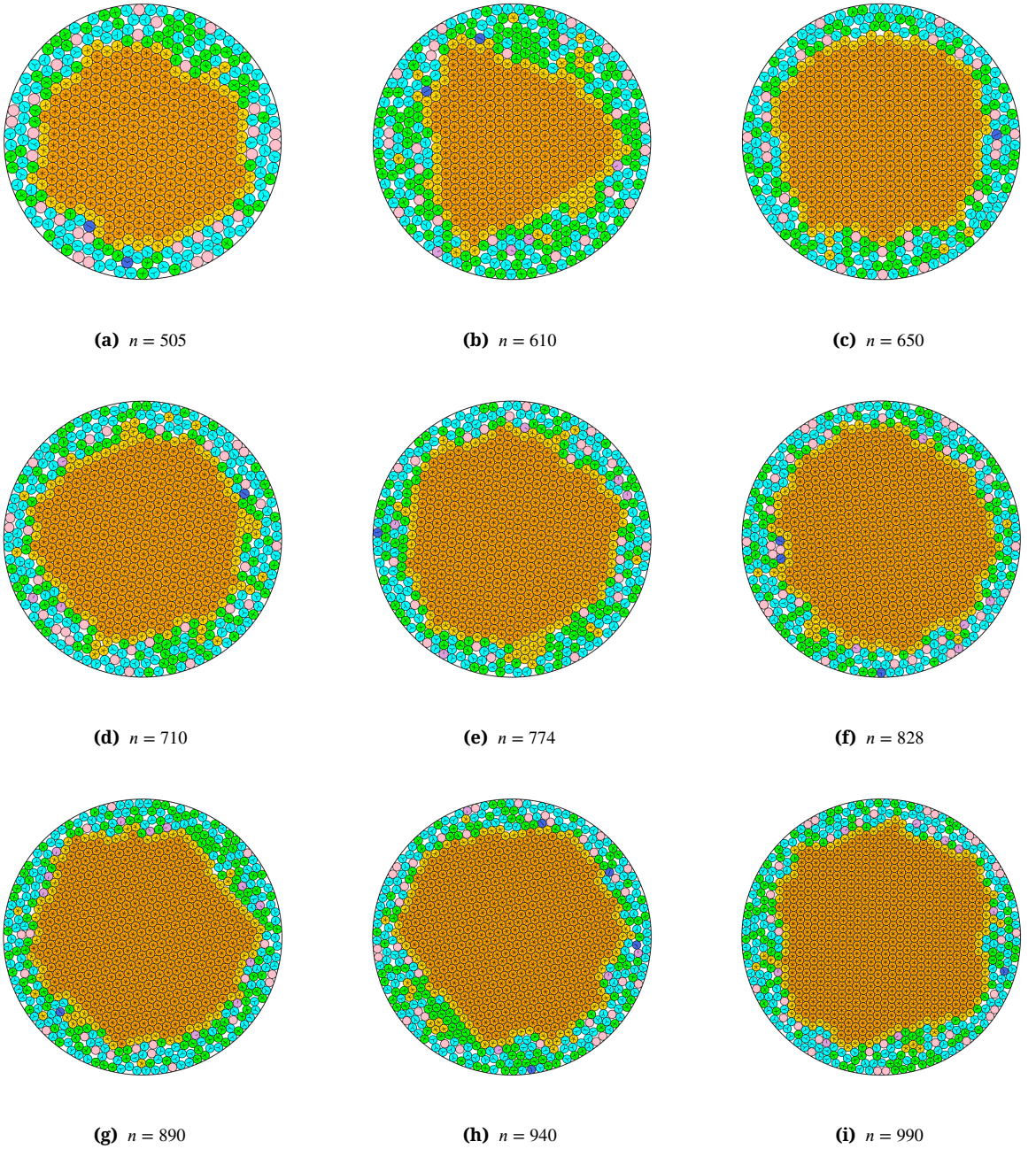
表中， n 表示实例数量； R'_{best} 和 R'_{avg} 分别表示SED（1批次GBO）运行10次得到的最优结果和平均结果； R_{best} 和 R_{avg} 分别表示SED（5批次GBO）运行10次得到的最优结果和平均结果； $time\ (s)$ 表示获得最优解的平均时间。 $R_{best}-R'_{best}$ 和 $R_{avg}-R'_{avg}$ 表示两类结果之间的差异，其中负值表示SED（5批次GBO）取得的结果优于SED（1批次GBO）。

从结果中可以看出，SED（5批次GBO）有18个最佳结果优于SED（1批次GBO），有1个最佳结果与后者相同，还有1个最佳结果较差。SED（5批次GBO）的所有平均结果都优于SED（1批次GBO）。这清楚地表明，在大规模实例上，多批次方法优于非批处理方法，而且我们提出的GBO方法在大规模实例上表现出色。

我们还比较了5批次GBO与非批处理（即1批次）的运行时内存需求。我们对实例 $n = 500$、550、600、...、1000 运行SED（5批次GBO）和SED（1批次GBO），并且

记录了这两个程序运行期间的驻留内存需求。比较结果如图7所示。

从图 7中可以观察到，SED（5批次 GBO）所需的常驻运行时内存低于 SED（1-批次 GBO）。具体而言，对于实例 $n = 500$、550、600、.....、1000，5批次 GBO仅需1批次 GBO（即经典BFGS）运行时内存的60.96%至64.00%。5批次 GBO与1批次 GBO的内存比率

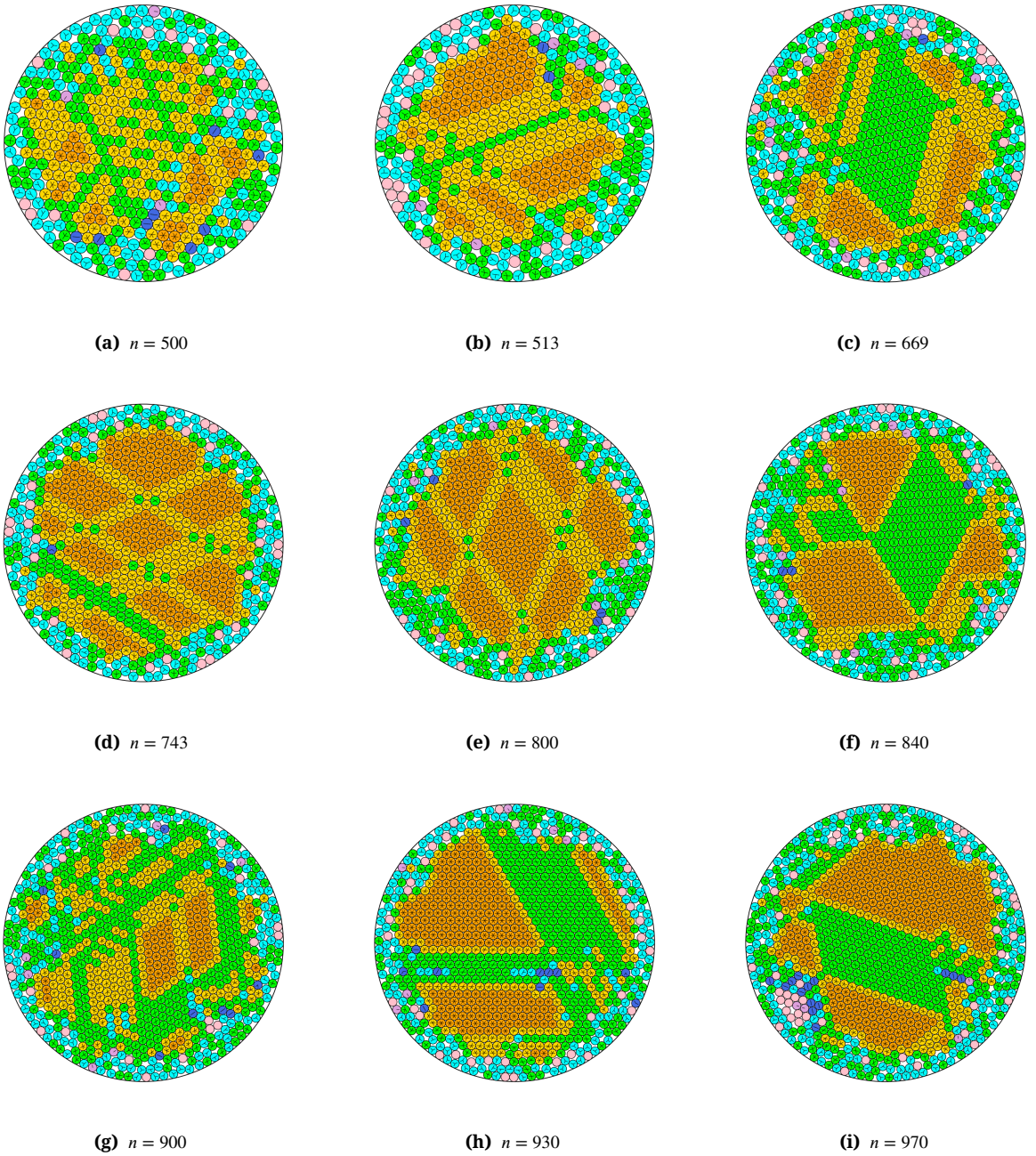


(a) $n = 505$ **(b)** $n = 610$ **(c)** $n = 650$

(d) $n = 710$ **(e)** $n = 774$ **(f)** $n = 828$

(g) $n = 890$ **(h)** $n = 940$ **(i)** $n = 990$

图5： 我们的算法针对大规模实例中的一些代表性实例找到的新改进解，这些实例在中心区域具有最密堆积。



(a) $n = 500$ **(b)** $n = 513$ **(c)** $n = 669$

(d) $n = 743$ **(e)** $n = 800$ **(f)** $n = 840$

(g) $n = 900$ **(h)** $n = 930$ **(i)** $n = 970$

图 6：我们算法针对大规模实例中部分代表性大规模实例找到的新改进解，这些实例的中心区域内并非最密堆积。

随着 n 的增加而降低。这表明，多批次 GBO在大规模实例的运行时内存需求方面具有优势。

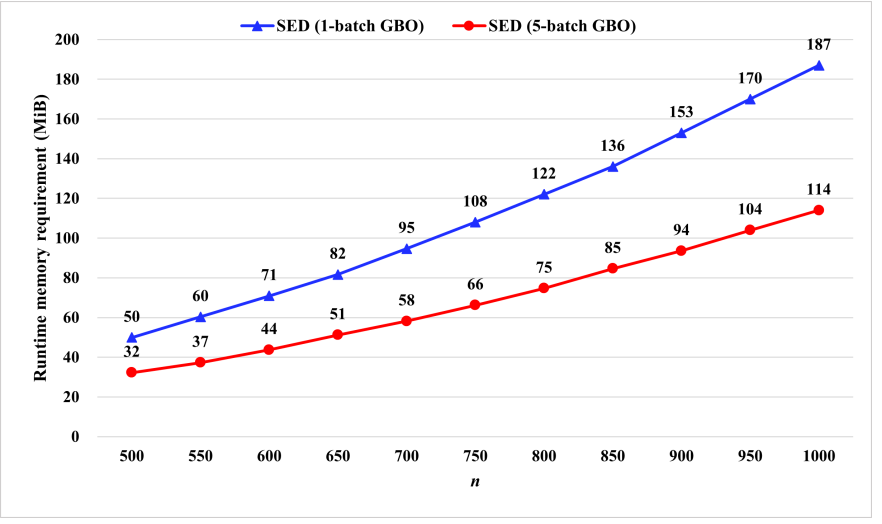
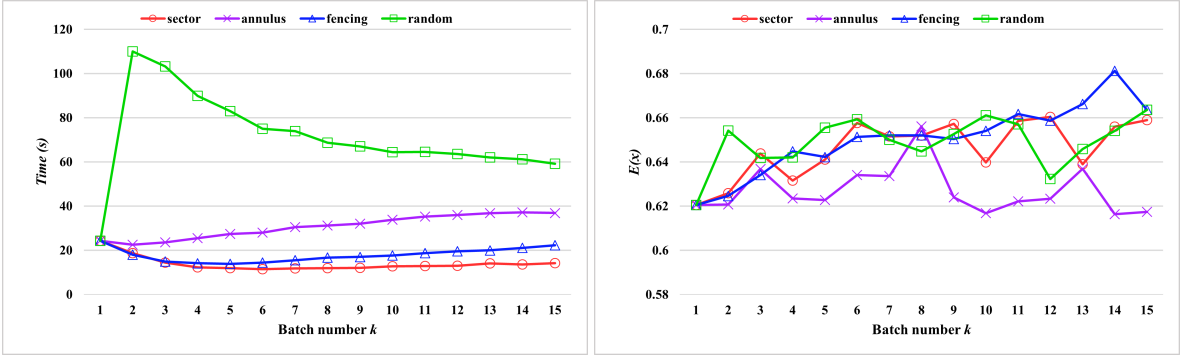


图 7： 比较SED（5批次GBO）和SED（1批次GBO）在实例 $n = 500、550、600、……、1000$上的运行时常驻内存需求，其中内存需求以MiB（即兆字节）表示。



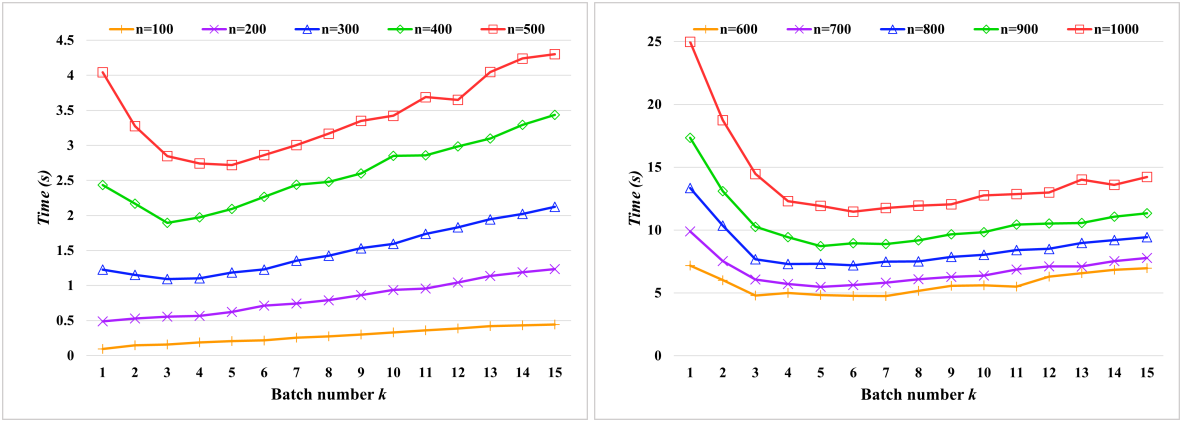
Fig图 8： 大规模实例上GBO方法四种划分之间的比较。

6.5. 参数研究

我们提出的算法需要调整三个参数，即GBO 分区的批量策略（第4.1节）、GBO的批次数 k （第4.1节）以及SED启发式算法的迭代步数 S_{iter} （第5.2节）。在本小节中，我们给出实验设计和比较结果，以确定合适的参数设置。

关于批量划分策略。 由于GBO是一种连续优化方法，因此它对特定实例不敏感，但对实例规模较为敏感。因此，我们针对 $n = 1000$ 规模、批次数从 $k = 1$ 到15，分别采用四种批量划分策略（扇区、圆环、栅栏和随机），以研究这些划分策略在大规模实例上的性能。每种设置均独立运行1,000次，每次运行均从随机初始布局开始，并在能量收敛或达到最大迭代步数时终止（见算法1）；平均耗时和平均收敛能量的实验结果如图8所示。注意，当批次数 $k = 1$时，GBO方法退化为经典BFGS方法（即非批处理方法）。

图8a 展示了四种划分的平均耗时比较，其中X轴表示批次数 k，Y轴表示1,000次运行的平均耗时。从图中可以观察到：



(a) (b)

图9：不同规模实例下，GBO方法在不同批次数设置之间的比较。

表5GBO在 $n = 100$、200、.....1000 下的最优批次数设置与非批次设置的比较。

n	$T_{non.}(s)$	$opt.$	$T_{opt.}(s)$	比率(%)	n	$T_{non.}(s)$	$opt.$	$T_{opt.}(s)$	比率(%)
100	0.09	$k = 1$	0.09	100.00%	600	7.17	$k = 7$	4.74	66.16%
200	0.49	$k = 1$	0.49	100.00%	700	9.90	$k = 5$	5.48	55.32%
300	1.23	$k = 3$	1.09	88.91%	800	13.35	$k = 6$	7.19	53.81%
400	2.43	$k = 3$	1.89	77.78%	900	17.35	$k = 5$	8.73	50.33%
500	4.04	$k = 5$	2.72	67.26%	1000	24.97	$k = 6$	11.46	45.89%

（1）随机划分的耗时显著高于非批处理（即 $k = 1$），这表明在GBO中采用随机划分会使性能变差。不过，随着批次数增加，随机划分的平均耗时可以降低。

(2) 三种几何批次划分策略呈现出相似的趋势：随着批次数增加，三种策略的平均耗时均先下降后上升。当 $k = 2$ and 3 时，环带划分的耗时略低于非批处理；当 $k \geq 4$ 时，则高于非批处理。$k \geq 2$ 时，扇区划分和栅栏划分的耗时均低于非批处理，且扇区划分在大规模问题上表现最佳。

(3) 将随机划分与扇区划分、环带划分和栅栏划分进行比较后可以发现，求解 PECC问题需要采用合理的几何划分策略，而非随机划分；划分策略会直接影响 GBO方法的性能。

图8b 展示了四种划分方式的平均收敛能量对比，其中 X轴表示批次数 k，Y轴表示平均收敛能量 $E(\boldsymbol{x})$，数据来自 1,000 次运行。从图中可以看出，对于大多数批次划分设置，四种划分方式的平均收敛能量均略高于非批处理。不过，这些平均收敛能量大多介于0.62 和 0.66之间。我们认为这种差异属于实验误差，因为与非批处理相比，这些平均收敛能量的差异不超过 7%。

根据上述讨论，GBO方法的扇区划分显然能够加快收敛速度，而且扇区划分与非批处理在收敛结果方面没有本质区别。因此，我们选择扇区划分作为批处理策略的最优设置。

关于不同实例规模下的批次数。为评估 GBO 在不同规模实例上的性能，我们对规模为 $n = 100$，200，.....，1000 的实例使用扇区划分的 GBO方法，批次数从 $k = 1$ 到 $k = 15$。我们对每种设置独立运行 1,000 次，每次运行均从

表6参数 S_{iter} 在12个选定实例上的平均结果（R_{avg}）及计算结果和比较，其中测试参数值中获得的最佳结果以粗体显示。

n/ S_{iter}	R_{avg} 100	200	300	400	500	600
305	19.0029499894	19.0022928525	19.0023365854	**19.0022767126**	19.0027346869	19.0026950563
316	19.3350583426	19.3348928193	19.3352210818	19.3347673471	**19.3345842813**	19.3346305894
513	24.4871818937	24.4850574870	24.4858380045	24.4843239209	**24.4836476996**	24.4850158014
568	25.6753984748	25.6746996472	25.6746268914	25.6749397842	25.6747142118	**25.6746092173**
608	26.5878240724	26.5866285288	26.5860789992	26.5865544926	**26.5852665616**	26.5867207111
678	28.0509853708	28.0507532028	28.0503173118	28.0509759006	**28.0486293696**	28.0498132121
740	29.2459673340	29.2469306141	29.2451838453	29.2451220192	**29.2440743866**	29.2458209932
774	29.9177876254	29.9143968413	**29.9117005742**	29.9143441746	29.9157783799	29.9153122016
846	31.2425443239	31.2333675583	31.2304459481	31.2320033828	31.2275469287	**31.2261977327**
877	31.8025272145	31.7999463301	31.8016759748	**31.7975804456**	31.7991773790	31.7998976078
923	32.5975528505	32.5942894739	32.5935176512	32.5949283036	**32.5927036033**	32.5947280012
964	33.3335745251	33.3300879802	33.3314171856	33.3271953322	33.3287459497	**33.3265773865**
平均值	27.6066126681	27.6044452780	27.6040300044	27.6037509847	**27.6031336198**	27.6035015425

随机初始布局开始，并在能量收敛或达到最大迭代步数时终止。平均时间开销的实验结果如图9所示。请注意，当批次数 k 设为 1 时，GBO方法退化为经典 BFGS方法。

图 9a 和 9b 分别给出了规模 $n = 100$、200.....500 和规模 $n = 600$、700.....1000的平均耗时。X轴表示批次数 k，Y轴表示运行1,000次的平均耗时。表5 展示了非批处理和最优批量设置的平均耗时比较，其中表格第一列给出了 n 实例，$T_{non.}$ 以及 $T_{opt.}$ 非批处理（即 $k = 1$）和最优批量设置，$opt.$ 批次数的最优设置 k 和比率（%）$= \frac{T_{opt.}}{T_{non.}}$ 最优批量方法与非批处理方法的平均耗时比率。

根据图 9a 和 9b以及表 5，我们得出以下观察结果：

(1) $n = 100$ 和 $n = 200$ 对应的规模曲线表明，GBO 在非常小的实例上效果不佳；采用批量方法会增加收敛时间，并且随着批次数增加，GBO 需要更长时间才能获得收敛解。

(2) $n = 300$对应的规模曲线表明，在中等规模实例上，GBO 相较于非批处理方法略有优势。$k = 2$、3、4 和 5 的平均耗时略低于非批处理方法（即 $k = 1$）。规模为 $400 \le n \le 1000$ 的曲线表明，设置适当的批次数可以缩短收敛时间，并显著加快收敛过程。表中“比率（%）”列显示，比率会随着规模增大而降低，这表明随着实例规模增大，GBO 的加速效果优势更加明显。实验结果表明，GBO 在大规模实例上表现出色。

（3）大规模实例 $500 \le n \le 1000$ 的曲线表明，随着批次数 $k \ge 8$ 的增加，平均时间成本也在增加。这说明对于较大的取值，批次数并非总是越多越好。大规模实例的最优批次数 k 位于区间 $[5, 7]$ 内。

根据上述讨论，我们将批次数 $k = 3$ 作为中等规模实例 $300 \le n \le 320$ 的最优设置，并将批次数 $k = 5$ 作为大规模实例 $500 \le n \le 1000$ 的折中设置。

关于 SED 的迭代步数。接下来要分析的参数是 SED 的迭代步数。我们采用若干不同的迭代步数 $S_{iter} = 100$、200，.....，600 对 12 个实例进行 SED（5批次GBO），其中包括分别从中等规模、大规模 I 和 II 中随机选取的 2、6 和 4 个实例。迭代步数的比较结果见表6。 n 表示实例中的项数，第 2 至第 7 列显示每个测试迭代步数下运行 10 次或 20 次（中等规模为 20 次，大规模为 10 次）的平均结果，R_{avg}，底部的“平均值”行显示每列中 12 个实例结果的平均值。

表6 显示，对于所测试的 12 个实例中的 6 个，采用 $S_{iter} = 500$ 的算法在 R_{avg} 方面表现最佳，数量远高于其他 5 种经过测试的迭代步数。它在 6 种经过测试的迭代步数中，也取得了 12 个实例结果的最佳平均值。因此，我们将 S_{iter} 的默认值设为 500。

7. 结论

本文旨在解决最具代表性的装填问题——大规模圆内等圆装填问题。我们提出了一种新颖的几何批量优化方法，不仅能显著加快连续优化过程，还能降低寻找局部最优装填配置所需的内存需求。为此，我们还提出了一种解空间探索与下降搜索启发式方法，用于搜索整体装填优化的全局最优解。此外，我们提出了一种自适应邻居对象维护方法，解决了现有邻居结构维护方法中的一些问题，并且适用于动态装填问题和在线装箱问题。对 21 个中等规模实例（$n = 300$ 至 320）和 101 个采样的大规模实例（$n = 500$ 至 1000）进行的大量实验，证明了所提方法的有效性和效率。我们的算法通常能够找到优于当前最佳记录的新装填结果。此外，我们的几何批量优化、启发式搜索和自适应维护方法具有通用性，也可用于其他优化问题。未来，我们将扩展这些方法，以解决其他装填问题。

参考文献

Addis, B., Locatelli, M., Schoen, F., 2008年。正方形中的圆盘装填：一种新的全局优化方法。《INFORMS计算杂志》20, 516–524。Akeb, H., Hifi, M., M'Hallah, R., 2010年。圆形装填问题的自适应束搜索前瞻算法。《国际运筹学汇刊》17, 553–575。Akeb, H., Hifi, M., M'Hallah, R., 2009年。圆形装填问题的束搜索算法。《计算机与运筹学》36, 1513–1528。Baldi, M.M., Manerba, D., Perboli, G., Tadei, R., 2019年。最后一公里物流中的广义装箱问题。《欧洲运筹学杂志》274, 990–999。Birgin, E.G., Sobral, F., 2008年。最小化圆与球体装填问题中的物体尺寸。《计算机与运筹学》35, 2357–2375。Burke, E., Hellier, R., Kendall, G., Whitwell, G., 2006年。二维不规则装填问题的一种新的左下角填充启发式算法。《运筹学》54, 587–601。Carrabs, F., Cerrone, C., Cerulli, R., 2014年。圆装填问题的禁忌搜索方法，载于：第17届网络信息系统国际会议，IEEE。第165–171页。Castillo, I., Kampas, F.J., Pintér, J.D., 2008年。通过全局优化求解圆装填问题：数值结果与工业应用。《欧洲运筹学杂志》191, 786–802。Chen, M., Tang, X., Song, T., Zeng, Z., Peng, X., Liu, S., 2018年。将等圆装入圆形容器的贪心启发式算法。《计算机与工业工程》119, 114–120。Demaine, E.D., Fekete, S.P., Lang, R.J., 2010年。折纸设计中的圆装填问题很难。arXiv预印本 arXiv:1008.1224。Epstein, L., van Stee, R., 2005年。在线正方形与立方体装填。《信息学报》41, 595–606。Fekete, S.P., Hoffmann, H.F., 2017年。在线正方形内正方形装填。《算法学》77, 867–901。Fekete, S.P., von Höveling, S., Scheffer, C., 2019年。在线圆装填，载于：《算法与数据结构：第16届国际研讨会，WADS 2019，加拿大阿尔伯塔省埃德蒙顿，2019年8月5–7日，论文集》第16卷，Springer。第366–379页。Fodor, F., 1999年。圆内19个全等圆的最密装填。《几何学专刊》74, 139–145。Fodor, F., 2000年。圆内12个全等圆的最密装填。《代数与几何论文集》41, 401–409。Fodor, F., 2003年。圆内13个全等圆的最密装填。《代数与几何论文集》44, 431–440。Goldberg, M., 1971年。圆内14、16、17和20个圆的装填。《数学杂志》44, 134–139。Görtler, J., Schulz, C., Weiskopf, D., Deussen, O., 2017年。用于不确定性可视化的气泡树状图。《IEEE可视化与计算机图形学汇刊》24, 719–728。Graham, R., Peck, C., 1968年。具有给定最大间距的点集（问题e1921）。《美国数学月刊》75, 80–81。Graham, R.L., Lubachevsky, B.D., Nurmela, K.J., Östergård, P.R., 1998年。圆内全等圆的致密装箱。《离散数学》181, 139–154。Grosso, A., Jamali, A., Locatelli, M., Schoen, F., 2010年。求解圆形容器中等圆与不等圆的装填问题。《全球优化杂志》47, 63–81。Hartman, T., Mazáˇc, D., Rastelli, L., 2019年。球体装箱与量子引力。《高能物理学杂志》2019, 1–68。He, K., Mo, D., Ye, T., Huang, W., 2013年。用于求解带平衡约束的圆装填问题的由粗到细的准物理优化方法。《计算机与工业工程》66, 1049–1060。He, K., Tole, K., Ni, F., Yuan, Y., Liao, L., 2021年。求解圆形装箱问题的自适应大邻域搜索。《计算机与运筹学》127, 105140。

He, K., Ye, H., Wang, Z., Liu, J., 2018年。在圆形容器中装填等圆的高效准物理准人类算法。《计算机与运筹学》92, 26–36。Hifi, M., M'Hallah, R., 2004年。约束圆形切割问题的近似算法。《计算机与运筹学》31, 675–694。Hifi, M., M'Hallah, R., 2007年。圆形装填问题的动态自适应局部搜索算法。《欧洲运筹学杂志》183, 1280–1294。Hifi, M., M'Hallah, R., Zissimopoulos, V., 2004年。圆形切割问题的模拟退火方法。《欧洲运筹学杂志》159, 430–448。Hifi, M., Yousef, L., 2019年。球体装箱问题的基于局部搜索的方法。《欧洲运筹学杂志》274, 482–500。Hokama, P., Miyazawa, F.K., Schouery, R.C., 2016年116, 337 342。一种在线圆装填的有界空间算法。《信息处理快报》–。Huang, W., Li, Y., Akeb, H., Li, C., 2005年。将不等圆装入矩形容器的贪心算法。《运筹学会杂志》56, 539–548。Huang, W., Xu, R., 1999年。求解圆形装箱问题的两种拟人化策略。《中国科学：E辑：技术科学》42, 595–602。Huang, W., Ye, T., 2011年。寻找圆内等圆致密装填的全局优化方法。《欧洲运筹学杂志》210, 474–481。Huang, W.Q., Li, Y., Jurkowiak, B., Li, C.M., Xu, R.C., 2003年。将不等圆装入圆形容器的两层搜索策略，载于：约束规划原理与实践国际会议，Springer。第868–872页。Huang, W.Q., Li, Y., Li, C.M., Xu, R.C., 2006年。将不等圆装入圆形容器的新启发式方法。《计算机与运筹学》33, 2125–2142。Kravitz, S., 1967年。将圆柱体装入圆柱形容器。《数学杂志》40, 65–71。Lai, X., Hao, J.K., Yue, D., Lü, Z., Fu, Z.H., 2022年。将等圆装入圆形容器的迭代动态阈值搜索。《欧洲运筹学杂志》299, 137–153。Leao, A.A., Toledo, F.M., Oliveira, J.F., Carravilla, M.A., Alvarez-Valdés, R., 2020年。不规则装箱问题：数学模型综述。《欧洲运筹学杂志》282, 803–822。Leung, J.Y., Tam, T.W., Wong, C.S., Young, G.H., Chin, F.Y., 1990年。将正方形装入正方形。《并行与分布式计算杂志》10, 271–275。Lintzmayer, C.N., Miyazawa, F.K., Xavier, E.C., 2019年。在线圆与球体装填。《理论计算机科学》776, 75–94。Liu, J., Zhang, K., Yao, Y., Xue, Y., Guan, T., 2016年。采用粗调与细调的圆装填问题启发式准物理算法。《计算机与工业工程》101, 416–426。López, C.O., Beasley, J.E., 2011年。具有多种容器的圆装填问题启发式方法。《欧洲运筹学杂志》214, 512–525。Lü, Z., Huang, W., 2008年。用于求解圆装填问题的PERM。《计算机与运筹学》35, 1742–1755。Martello, S., Monaci, M., Vigo, D., 2003年。条带装箱问题的精确方法。《INFORMS计算杂志》15, 310–319。Melissen, H., 1994年。圆内十一个全等圆的最致密装箱。《几何学专刊》50, 15–25。Miyazawa, F.K., Wakabayashi, Y., 2003年。立方体装填。《理论计算机科学》297, 355–366。Mladenovi´ ´ 2005 32, c, 2419–2434 N., Plastria, F., Uroševic, D.,。应用于圆装填问题的重构下降法。《计算机与运筹学》–。Murakami, H., Higo, Y., Kusumoto, S., 2015年。Clonepacker：用于克隆集可视化的工具，载于：IEEE第22届软件分析、演化与再工程国际会议（SANER），IEEE。第474–478页。Nurmela, K.J., Östergård, P.R., 1997年。在正方形中装填最多50个等圆。《离散与计算几何》18, 111–120。Pirl, U., 1969年。单位圆盘内n个点之间的最小间距。《数学通讯》40, 111–124。Rao, Y., Wang, P., Luo, Q., 2021年。波束搜索与禁忌搜索混合求解不规则装填问题。《工程数学问题》2021。Reis, G.E., 1975年。圆内等圆的致密装填。《数学杂志》48, 33–37。Ren-Pu, G., Powell, M.J., 1983年27, 123 143。无约束优化中变度量矩阵的收敛性。《数学规划》–。Specht, E., 2022年。Packomania 网站：http://www.packomania.com。Stoyan, Y., Yaskov, G., 2014年。使用跳跃算法将不等圆装入最短条带。《优化快报》8, 949–970。Stoyan, Y., Yaskov, G., Romanova, T., Litvinchev, I., Yakovlev, S., Cantú, J.M.V., 2020年。多维超球体的优化装填：一种统一方法。《数学与工程生物科学》17, 6601–6630。Wang, W., Wang, H., Dai, G., Wang, H., 2006年。通过圆装填实现大型层次数据的可视化，载于：SIGCHI计算机系统中的人为因素会议论文集，第517–520页。Wang, Y., Wang, Y., Sun, J., Huang, C., Zhang, X., 2019年。针对带平衡约束的圆装填问题的基于刺激—响应的分配方法。《物理学A：统计力学及其应用》522, 232–247。Yanchevskyi, I., Lachmayer, R., Mozgova, I., Lippert, R.B., Yaskov, G., Romanova, T., Litvinchev, I., 2020年。用于无支撑结构的圆形装填。《EAI能源网络认可期刊》7, e3–e3。Zhao, C., Jiang, L., Teo, K.L., 2020年。三维不规则装箱问题的混沌萤火虫混合算法。《工业与管理优化杂志》16, 409。Zhao, H., She, Q., Zhu, C., Yang, Y., Xu, K., 2021年。采用受约束深度强化学习的在线三维装箱，载于：AAAI人工智能会议论文集，第741–749页。