论 文

基于禁忌搜索的启发式算法求解带平衡约束的圆形装填问题

李刚^①, 刘景发^{②③*}

① 南京信息工程大学数理学院, 南京 210044

② 南京信息工程大学网络信息中心, 南京 210044

③ 南京信息工程大学计算机与软件学院, 南京 210044

* 通信作者. E-mail: jfliu@nuist.edu.cn

收稿日期: 2010-12-17; 接受日期: 2011-02-13

国家公益性行业科研专项 (批准号: GYHY200906006)、江苏省自然科学基金 (批准号: BK2010570)、中国博士后科学基金 (批准号: 20100471350)、江苏省博士后科研资助计划 (批准号: 1001030B)、江苏省高校自然科学研究 (批准号: 09KJB520008) 和江苏省“青蓝工程”(苏教(2008)30号)资助项目

摘要 带平衡约束的圆形装填 (Packing) 问题是一类简化的卫星舱布局优化问题. 现提出一个基于禁忌搜索的启发式 (TSH) 算法对该问题进行求解. 算法从任一初始格局出发, 应用基于自适应步长的梯度法进行能量极小化. 为了使计算能有效地逃离局部极小点的陷阱且避免迂回搜索, 算法采用了禁忌搜索的策略. 在禁忌搜索的过程中, 我们对传统的邻域解、禁忌对象以及当前解接受原则进行了有效的改进. 对两组共 11 个有代表性的算例进行了实算. 计算结果表明, TSH 算法刷新了其中 7 个算例的当今国际上的最好纪录, 对于其余 4 个算例, 该算法均达到问题的最优解.

关键词 平衡约束 装填问题 启发式算法 禁忌搜索 布局优化

1 引言

装填 (Packing) 问题研究的是将多个物体互不嵌入地放置在有限空间的容器内, 要求尽量提高容器的利用率, 降低空间的浪费. 装填问题被广泛应用于货物装载、工业切割、网络通信、航天等领域^[1–3]. 对此问题的解决能够极大地节约资源、降低生产成本和运输费用等.

装填问题是一个 NP 难度问题^[4]. 在国内, 对装填问题的研究比较少, 工作主要集中在圆形和矩形装填问题^[5–9], 而且多为不带性能 (如平衡性、惯性、稳定性等) 约束的装填布局问题. 本文主要研究带平衡约束的圆形装填问题. 该问题除了像通常的圆形装填问题一样, 要求圆形待布物之间不嵌入和容器利用率高之外, 还要求布局后的系统满足静不平衡约束^[10].

近年来, 一些学者对带平衡约束的圆形装填问题进行了研究, 并提出了一些求解算法. Teng 等^[10]采用模式迭换法和主布模法给出了该约束布局优化问题的启发式算法; Tang 等^[11]提出了一种十进制编码控制参数的自适应遗传算法; Qian 等^[12]通过将人工设计的个体加入到染色体群体中, 提出了人机交互的遗传算法; Yu 等^[13]通过增加学习算子, 并构造拟 Newton 方向进行局部寻优, 提出了基

引用格式: 李刚, 刘景发. 基于禁忌搜索的启发式算法求解带平衡约束的圆形装填问题. 中国科学: 信息科学, 2011, 41: 1076–1088

于学习的遗传算法; Li 等^[14]提出了基于带变异算子的粒子群优化算法; Zhou 等^[15]提出了适合粒子群优化的约束处理机制, 并通过与直接搜索法相结合得到了一种混合算法; Lei 等^[16]通过对传统的粒子群优化算法进行改进, 提出了一种自适应粒子群优化算法; Huang 等^[17]在拟物算法^[18]的基础上提出了加速搜索过程的改进的拟物算法; Wang 等^[19]以散射搜索法为基础, 集成两种局部搜索算法, 梯度下降法和 Nelder-Mead 直接搜索法, 提出了改进的散射搜索法.

这些算法大致可分为两类: 即随机算法和启发式算法. 虽然这些算法在求解带平衡约束的圆形装填问题时取得了较大的进展, 但是它们各有优缺点. 启发式算法 (如模式迭换法^[10]、改进的拟物算法^[17]等) 较之于随机算法 (如遗传算法^[11-13]、粒子群算法^[14-16]、散射搜索法^[19]等), 针对性强, 但普适性不强; 随机算法具有全局搜索且适应面广的优点, 但缺乏有效的局部搜索机制, 且收敛速度较慢; 另外, 确定性局部极小化搜索算法 (如梯度法、共轭梯度法、拟 Newton 法等) 也可进行全局优化, 其具有搜索速度快、精度高的优点, 但却易于陷入局部极小点的陷阱. 文献 [19-21] 指出, 如何将具有全局搜索能力的随机算法与能精确搜索的局部搜索算法以及一些启发式策略相结合, 是构造具体领域一般问题的高效求解算法的有效机制. 在文献 [21] 中, Liu 等^[22]通过对势能曲面变平法 (ELP, energy landscape paving) 进行改进, 进而将改进的 ELP 法与局部搜索的梯度法和一些启发式的格局更新机制相结合, 提出了一种新的全局搜索算法 — 吸引盘填充算法. 吸引盘填充算法的高效性通过求解一组公认的带平衡性能约束的圆形装填算例得到了验证. 然而, 由于吸引盘填充算法中 ELP 方法的本质是基于 Metropolis 抽样的随机方法, 所以 ELP 方法在搜索的过程中, 还是容易错过一些有“前景”的优良格局. 因此, 如何针对具体问题构造一种高效的全局随机搜索方法 (该方法最好具有既能集中搜索, 又能分散搜索, 且能在合理的时间内有效地遍历整个解空间的能力), 以及充分发挥全局随机搜索方法与局部极小化方法的各自特长, 使它们能很好地有机结合, 互相取长补短、相辅相成, 可能是设计高效算法的关键所在.

禁忌搜索 (tabu search, TS) 方法^[23-25]是一种具有局部搜索能力的全局寻优的智能搜索算法. 针对局部邻域搜索易于陷入局部极小点的特点, TS 利用禁忌表采用禁忌策略尽量避免迂回搜索, 从而保证对不同的有效搜索途径的探索, 跳出局部极小点. TS 同时又采用藐视准则避免遗失优良状态, 激励对优良状态的局部搜索, 实现全局优化, 所以 TS 方法是一种搜索能力极强的全局寻优方法. 本文通过将局部极小化方法与 TS 算法相结合, 提出了一种基于禁忌搜索的启发式 (TSH) 算法. TSH 算法对传统的 TS 算法进行了有效的改进, 提出了所求问题的有针对性的邻域解构造方式, 定义了新的禁忌对象、当前解接受原则等. 实验结果表明, 该方法是求解带平衡约束的圆形装填问题的有效算法. 对两组共 11 个有代表性的算例进行了实算, TSH 算法刷新了其中 7 个算例的当今国际上的最好纪录, 对于其余 4 个算例, 该算法均达到了问题的最优解.

2 问题的描述及转化

以人造卫星舱布局设计为背景^[21]的带平衡约束的圆形装填问题, 也就是一个所谓转动圆桌平衡摆盘问题^[10]. 如图 1 所示, 在一个半径为 R_0 , 以角速度 ω 转动的圆桌上布置 n 个圆形待布物 $C_i (i \in N = \{1, 2, \dots, n\})$, 要求这 n 个圆形待布物尽量趋于圆桌的中心, 并满足如下约束:

- (1) 各圆形待布物之间不干涉 (即互相不嵌入);
- (2) 各圆形待布物不超出桌面;
- (3) 布局后的系统的静不平衡量应小于某一允许值 $\delta_J (> 0)$.

图 1 Descartes 坐标系下的圆形待布物布局示意图

Figure 1 The schematic diagram of packing of the circular objects in Cartesian coordinate system

建立如图 1 所示 Descartes 坐标系. 设任一圆形待布物 C_i 的半径为 r_i , 质量为 m_i , C_i 的圆心坐标为 (x_i, y_i) . 令 $X = (x_1, y_1, x_2, y_2, \dots, x_n, y_n)$ 表示布局的一个方案, 也就是一个格局. 那么带平衡约束的圆形装填问题实际上就是如何将这 n 个圆形待布物互不干涉地放置于圆形容器 C_0 内, 使得 C_0 的半径 R_0 尽可能小, 也就是下面的约束优化问题:

$$\begin{cases} \text{minimize} & R_0 = \max_{i \in N} \left\{ \sqrt{x_i^2 + y_i^2} + r_i \right\}; \\ \text{subject to} & \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq r_i + r_j, \quad i \neq j, i, j \in N; \\ & \sqrt{x_i^2 + y_i^2} \leq R_0 - r_i, \quad i \in N; \\ & J = \sqrt{\left(\sum_{i=1}^n m_i x_i \right)^2 + \left(\sum_{i=1}^n m_i y_i \right)^2} \leq \delta_J. \end{cases} \quad (1)$$

这里 J 表示布局后的系统的静不平衡量. 静不平衡约束, 即 $J \leq \delta_J$ 的物理意义是, 由圆形容器 (即卫星舱的旋转隔板) 内各圆形待布物的偏心质量所产生的静不平衡量 J (或不平衡的离心力的大小) 是在允许值范围内. J 的值越小, 整个系统就越满足静平衡约束; 当 $J=0$ 时, 整个系统处于静平衡.

我们将所有 n 个圆形待布物和圆形容器想象为光滑的弹性实体, 按照文献 [21] 的拟物策略和罚函数法, 求解约束优化问题 (1) 的可行解可转化为下面的无约束优化问题 (2) 的求解:

$$\text{minimize} \quad U(X) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n d_{ij}^2 + l \left[\left(\sum_{i=1}^n m_i x_i \right)^2 + \left(\sum_{i=1}^n m_i y_i \right)^2 \right]. \quad (2)$$

这里 $d_{ij} = d_{ji} = \max(0, r_i + r_j - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2})$ 是圆形待布物 C_i 和 C_j 之间的嵌入深度, $d_{i0} = d_{0i} = \max(0, r_i + \sqrt{x_i^2 + y_i^2} - R_0)$ 是圆形待布物 C_i 与圆形容器 C_0 之间的嵌入深度; l 为惩罚项系数, 是一个小的正数. 如果能找到一个有效算法求解优化问题 $\text{minimize } U(X)$, 那么我们就能够通过二分法等有效方法得到原来带平衡约束的圆形装填问题 (1) 的最优解或近似最优解. 因此, 下面我们将主要讨论优化问题 $\text{minimize } U(X)$ 的求解.

3 禁忌搜索算法

禁忌搜索 (tabu search, TS) 法由 Glover^[23-25] 于 1986 年提出, 它是局部邻域搜索算法的推广, 是人工智能在组合优化问题中的一个成功应用. 所谓禁忌就是努力禁止重复前面的工作. 为了尽量避免算法迂回搜索, 探索不同的有效的搜索途径, 同时避免算法陷入局部极小点, TS 算法用一个禁忌表记录下已经到达过的局部极小点, 在下一搜索中, 利用禁忌表中的信息不再或有选择地搜索这些点. TS 算法的基本流程是: 给定一个初始解, 构建当前解的一个邻域, 从邻域解中确定若干候选解; 如果最佳候选解的目标函数值优于当前最优解的目标值, 则忽视其禁忌属性, 将其替换为当前最优解和当前解, 并将其加入禁忌表中, 同时修改禁忌表中各禁忌对象的任期, 并释放任期为 0 的对象; 若不存在这样的候选解, 则在候选解中选择非禁忌的最优解为新的当前解, 同时将其加入禁忌表中, 并修改禁忌表中各对象的任期, 释放任期为 0 的对象; 如此重复上述过程, 直到算法满足终止条件. 对于 TS 算法更详细的描述, 读者可参考文献 [25].

4 基于禁忌搜索的启发式算法

4.1 初始格局的产生

TS 算法是基于局部搜索的全局寻优方法. 通常, TS 算法从好的初始解出发易于搜索到好的解, 而从差的初始解出发会降低算法的收敛速度, 所以基于禁忌搜索的多数算法选择好的初始解来提高算法效率. 然而, 在本文提出的基于禁忌搜索的启发式 (TSH) 算法中, 我们不考虑初始解的获取方式, 仅仅从任一随机的初始格局出发, 甚至可将所有圆形待布物均放在圆形容器的中心亦可. 这是由于即使从任一初始格局出发, 在紧接着执行邻域结构更新格局并运行梯度法后, 所有嵌入的圆形待布物将在弹性力的作用下不断地进行运动, 最终使整个系统达到平衡, 也很有可能得到一个能量极小的好的格局.

4.2 邻域格局的产生

定义 1 动作: 对于当前格局 $X=(x_1, y_1, \dots, x_i, y_i, \dots, x_n, y_n)$, 称将势能半径比 E_i/r_i 最大的圆形待布物 C_i 放在圆形容器内的空位点 (即圆形容器的空白区域内的一点) 上的手续为一个动作, 这里 $E_i = \sum_{j=0, j \neq i}^n d_{ij}^2$ 表示第 i 个圆形待布物所受到的其他 $n-1$ 个圆形待布物和圆形容器 C_0 施加于它的挤压弹性势能之和.

确定圆形容器内的空位点的方法如下: 在圆形容器内随机产生一个点, 如果该点与容器内任一圆形待布物的中心之间的距离均大于相应的圆形待布物的半径, 则该点为圆形容器内的一个空位点; 否则, 重新在圆形容器内随机产生一个新点, 此过程重复进行, 直到新产生的点为圆形容器内的空位点.

定义 2 邻域格局与邻域: 对于当前格局 $X=(x_1, y_1, \dots, x_i, y_i, \dots, x_n, y_n)$, 称 $X'=(x_1, y_1, \dots, x'_i, y'_i, \dots, x_n, y_n)$ 是 X 的一个邻域格局, 这里 (x'_i, y'_i) 是第 i 个圆形待布物 C_i 做一个动作后得到的新的圆心坐标. X 的所有邻域格局的集合称为 X 的邻域, 记为 $N(X)$.

为了提高 TS 方法的搜索效率, 同时使 TS 搜索更有效地逃离局部极小点的陷阱, 我们采用启发式的邻域结构来更新格局, 即首先挑出当前格局中势能半径比 E_i/r_i 最大的圆形待布物 C_i ; 然后将 C_i 在圆形容器内“试做”10 个随机动作, 这里“试做”是指 C_i 只是暂时放置, 一旦计算出 C_i 放在该位置的挤压弹性势能后就将 C_i 从该位置移走; 最后将 C_i 正式放在挤压弹性势能最小的位置, 同时保持其

他圆形待布物的位置不变而得到一个新的格局 X' .

定义 3 候选邻域: 对于当前格局 $X=(x_1, y_1, \dots, x_i, y_i, \dots, x_n, y_n)$, 若 C_i 是势能半径比最大的圆形待布物, 称将 C_i 在圆形容器内“试做”10 个随机动作得到的 10 个邻域格局为 X 的基于 C_i 的候选邻域, 记为 $N(X, C_i)$. 显然, $N(X, C_i) \subseteq N(X)$.

4.3 局部搜索策略

当用邻域结构产生新的格局 X' 后, 为了进一步搜索 X' 附近能量更低的格局, 避免搜索与全局最优解擦肩而过, 本文从 X' 出发, 进一步执行精细的局部搜索算法——基于自适应步长的梯度法 (GM, gradient method)^[21], 以加速对全局最优格局的搜索.

4.4 禁忌表

禁忌表包括禁忌对象和禁忌长度. 禁忌对象, 顾名思义, 是指那些被禁在禁忌表中的对象. 当采用邻域结构更新格局时, 为了避免算法迂回搜索, 防止某些圆形待布物被反反复复挑出而找不到合适的放置位置, 本文对某些圆形待布物进行禁忌. 然而, 不像通常的禁忌搜索方法那样设置禁忌对象: 从候选解中选择最佳解, 若其目标函数值优于当前最优解的目标值, 则无视该圆形待布物的禁忌属性, 将该解设置为当前最优解和当前解, 并把该圆形待布物作为禁忌对象重新加入禁忌表中; 否则从候选解中选择非禁忌的最佳解为新的当前解, 并将其对应的圆形待布物作为禁忌对象加入禁忌表中. 现在, 我们不管该圆形待布物 C_i 原来是否是禁忌对象, 只要其被挑出且正式做 5 次动作, 并在每次做动作后执行局部搜索 (见 4.3 小节) 所得到的格局仍未被接受 (接受原则见 4.5 小节), 就将 C_i 的势能半径比设置为 0, 并将其加入禁忌表作为禁忌对象. 这样, 当再次采用邻域结构从当前格局中挑选势能半径比最大的圆形待布物做动作时, 就不会再次挑选 C_i , 从而能有效地避免搜索再次回到已经搜索过的局部极小值点, 而且这种改进策略也能给当前能量极小的格局 (包括接近最优解的优良格局) 多次寻优机会, 也就是从它出发, 通过邻域结构更新格局后, 进一步执行局部搜索策略来对全局最优解进行有效搜索.

禁忌长度是指禁忌对象在不考虑藐视准则的情况下不允许被选取的最大次数. 本文设置禁忌长度为 3, 并采用单向链表记录搜索过程中被禁忌的对象, 同时采用先进先出的队列数据结构来更新禁忌表.

4.5 藐视准则与当前格局的接受原则

藐视准则是指若某禁忌对象对应的格局的目标函数值优于当前最优格局的目标函数值时, 则无视其禁忌属性, 仍接受该格局为当前格局. 藐视准则是算法避免遗失优良状态, 进而实现全局优化的关键步骤. 在通常的禁忌搜索方法中, 当不存在某禁忌对象的目标函数值优于当前最优格局的目标值时, 就从候选邻域中选择非禁忌的最优格局为当前格局, 这种策略显然没有给前一能量极小的格局, 特别是接近最优解的优良格局足够的机会去从它出发进一步搜索能量更低的格局, 而且很容易接受能量高的格局为当前格局, 这就存在缺陷. 本文在保留藐视准则的前提下, 对这种当前格局的接受原则作了如下改进:

1) 若前一格局 X 中被挑出需要做动作的圆形待布物 C_i 是禁忌对象, 且将其做动作并执行局部搜索后得到的格局 X' 的能量 $U(X')$ 小于当前最优格局的能量 U_{opt} 时, 则将 C_i 强行解禁, 同时修改禁忌表中各对象的任期, 释放任期为 0 的对象, 更新当前最优格局, 并将 X' 设置为当前格局.

2) 如果 C_i 是禁忌对象, 且将其做动作并执行局部搜索后得到的格局 X' 的能量 $U(X')$ 不小于 U_{opt} , 则不接受 X' 为当前格局, 此时, 恢复 X 为当前格局.

3) 如果 C_i 不是禁忌对象, 但将 C_i 做动作并执行局部搜索后得到的格局 X' 的能量 $U(X')$ 小于前一格局 X 的能量 $U(X)$, 则仍然接受 X' 为当前格局, 同时修改禁忌表中各对象的任期, 释放任期为 0 的对象; 若此时该格局 X' 的能量 $U(X')$ 小于当前最优格局的能量 U_{opt} , 则进一步将 X' 更新为当前最优格局.

4) 如果 C_i 不是禁忌对象, 且将 C_i 做动作并执行局部搜索后得到的格局 X' 的能量 $U(X')$ 不小于前一格局 X 的能量 $U(X)$, 则不接受 X' 为当前格局, 此时, 恢复 X 为当前格局.

不难看出, 这种改进策略更有利于 TSH 算法在迭代过程中从好的有前景的格局出发, 进一步搜索全局最优格局.

4.6 结束准则

当下列条件之一满足时, TSH 算法就结束: (1) 算法找到全局最优解, 即当 $U(X) \leq 10^{-20}$ 时, 算法成功退出; (2) 当迭代步数 $t > 10^5$ 时, 则算法失败退出.

4.7 算法描述

通过将基于自适应步长的梯度法融入改进的禁忌搜索过程, 便得到求解带平衡约束的圆形装填问题的一种新的全局搜索算法 — 基于禁忌搜索的启发式 (TSH) 算法. TSH 算法的计算步骤如下:

1) 在半径为 R_0 的圆形容器 C_0 中, 随机给出 n 个圆形待布物 $C_i (i=1, 2, \dots, n)$ 的圆心坐标 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, 并将之作为初始格局 X , 即 $X = (x_1, y_1, x_2, y_2, \dots, x_n, y_n)$. 令 $t=1, \varepsilon = 10^{-20}, U_{\text{opt}}=10^{10}$.

2) 计算 $U(X, t)$.

3) 挑出当前格局 X 中势能半径比 E_i/r_i 最大的圆形待布物 C_i , 备份 X , 令 $j=1$;

4) 将 C_i 在圆形容器内随机“试做”10 个动作得到 X 的基于 C_i 的候选邻域 $N(X, C_i)$, 再从 $N(X, C_i)$ 中挑选一个最佳候选邻域格局, 记为 X' ;

5) 调用基于自适应步长的梯度法 (GM), 令 $X' = \text{GM}(X')$;

6) 如果 $U(X', t) < \varepsilon$, 则输出 X' 且成功退出;

否则 如果 C_i 是禁忌表中的对象, 则

如果 $U(X', t) < U_{\text{opt}}$, 则

将 C_i 从禁忌表中删除, 令 $U_{\text{opt}} = U(X', t)$, 转 7);

否则 不接受 X' , 恢复 X 为当前格局, 令 $j = j+1$, 转 8);

否则 如果 $U(X', t) < U_{\text{opt}}$, 则令 $U_{\text{opt}} = U(X', t)$;

如果 $U(X', t) < U(X, t)$, 则转 7);

否则 不接受 X' , 恢复 X 为当前格局, 令 $j=j+1$, 转 8).

7) 接受 X' 为当前格局, 即令 $X = X', U(X, t) = U(X', t)$, 同时将禁忌表中各禁忌对象的任期减 1, 释放任期为 0 的对象, 转 9);

8) 如果 $j > 5$, 则令 C_i 的势能半径比为 0, 且将 C_i 加入禁忌表中, 转 3);

否则转 4).

9) 如果 $t < 10^5$, 则令 $t = t + 1$, 转 3);

否则 失败退出.

表 1 第一组测试算例

Table 1 The first set of test instances

Instance	n	δ_J (g·mm)	r_i (mm) and m_i (g), $i=1, 2, \dots, n$
Instance 1.1	$n=5$	0.1	$r_1 = m_1=20.71, r_i = m_i=50, i=2, 3, 4, 5$
Instance 1.2	$n=9$	0.1	$r_1 = r_2 = r_3 = r_4 = (\sqrt{2}-1), r_5 = r_6 = r_7 = r_8 = r_9 = (3 - 2\sqrt{2}), m_i = r_i, i=1, 2, \dots, 9$
Instance 1.3	$n=5$	1.0	$r_1=6, r_2=7, r_3=8, r_4=9, r_5=11, m_i = r_i^2, i=1, 2, \dots, 5$
Instance 1.4	$n=7$	3.4	$r_1=10, r_2=11, r_3=12, r_4=11.5, r_5=9.5, r_6=8.5, r_7=10.5, m_i = r_i^2, i=1, 2, \dots, 7$
Instance 1.5	$n=40$	20	$r_1=106, m_1=11, r_2=112, m_2=12, r_3=98, m_3=9, r_4=105, m_4=11, r_5=93, m_5=8,$ $r_6=103, m_6=10, r_7=82, m_7=6, r_8=93, m_8=8, r_9=117, m_9=13, r_{10}=81, m_{10}=6,$ $r_{11}=89, m_{11}=7, r_{12}=92, m_{12}=8, r_{13}=109, m_{13}=11, r_{14}=104, m_{14}=10, r_{15}=115, m_{15}=13,$ $r_{16}=110, m_{16}=12, r_{17}=114, m_{17}=12, r_{18}=89, m_{18}=7, r_{19}=82, m_{19}=6, r_{20}=120, m_{20}=14,$ $r_{21}=108, m_{21}=11, r_{22}=86, m_{22}=7, r_{23}=93, m_{23}=8, r_{24}=100, m_{24}=10, r_{25}=102, m_{25}=10,$ $r_{26}=106, m_{26}=11, r_{27}=111, m_{27}=12, r_{28}=107, m_{28}=11, r_{29}=109, m_{29}=11, r_{30}=91, m_{30}=8,$ $r_{31}=111, m_{31}=12, r_{32}=91, m_{32}=8, r_{33}=101, m_{33}=10, r_{34}=91, m_{34}=8, r_{35}=108, m_{35}=11,$ $r_{36}=114, m_{36}=12, r_{37}=118, m_{37}=13, r_{38}=85, m_{38}=7, r_{39}=87, m_{39}=7, r_{40}=98, m_{40}=9$

5 计算结果与分析

我们将基于禁忌搜索的启发式 (TSH) 算法用 Java 语言编程, 且在 Intel Core 2 Duo, 1.6 GHz CPU, 内存为 1 GB 的笔记本上运行. 我们测试了文献中的二组算例.

5.1 测试算例

第一组算例有 5 个. 算例 1.1 和 1.5 引自于文献 [11], 算例 1.2 引自于文献 [12], 算例 1.3 和 1.4 引自于文献 [10]. 这 5 个算例被目前文献经常使用作为带平衡约束的圆形装填问题的测试实例 (Benchmark)^[10-19,21]. 算例 1.1 和 1.2 的最优解已知, 分别是 120.7106782 和 1.0000000. 对于每一个算例, 圆形待布物的个数 n , 允许的静不平衡量 δ_J (g·mm) 以及每个算例的各个圆形待布物的半径 r_i (mm) 和质量 m_i (g) 分别列于表 1 中.

第二组算例有 6 个, 这 6 个算例均引自于文献 [17]. 对于算例 2.1 和 2.2, 不难计算得到它们的最优圆形容器半径分别是 60.0000000 和 215.4700539. 这组算例通常用作不带性能约束的纯粹装填问题的测试实例^[26-28]. 然而, 由于这组算例既包含规则的, 也包含不规则的; 既包含等圆算例, 也包含不等圆算例; 既有小规模的, 也有大规模的算例, 所以被经常用作测试算法效率的 Benchmark. 本文应用这组算例去进一步测试 TSH 算法在带平衡约束的圆形装填问题上的效率. 对于第二组算例, 每个算例的圆形待布物的个数 n , 允许的静不平衡量 δ_J (g·mm) 以及各个圆形待布物的半径 r_i (mm) 和质量 m_i (g) 分别列于表 2 中.

5.2 实验结果与比较

对于第一组算例, 我们分别设置静不平衡量前的惩罚因子为 $l=10^{-4}, 10^{-4}, 10^{-5}, 10^{-7}$ 和 10^{-4} . 对于每个算例, 独立运行 TSH 算法 5 次. 在 5 次运行中, 我们将获得的最小圆形容器半径 R_0 (mm) 以及静不平衡量 J (g·mm) 和运行时间 T (s) 的平均值分别列于表 3 中, 并与基于带变异算子的粒子群算法

表 2 第二组测试算例

Table 2 The second set of test instances

Instance	n	$\delta_J(\text{g}\cdot\text{mm})$	$r_i(\text{mm})$ and $m_i(\text{g}), i=1, 2, \dots, n$
Instance 2.1	$n=7$	0.01	$r_i=20, m_i=r_i^2, i=1, 2, \dots, 7$
Instance 2.2	$n=12$	0.01	$r_1=r_2=r_3=100, r_4=r_5=r_6=48.26, r_7=\dots=r_{12}=23.72, m_i=r_i^2, i=1, 2, \dots, 12$
Instance 2.3	$n=15$	0.01	$r_1=1, r_{i+1}=r_i+1, i=1, 2, \dots, 14, m_i=r_i^2, i=1, 2, \dots, 15$
Instance 2.4	$n=17$	0.01	$r_1=25, r_2=20, r_3=r_4=15, r_5=r_6=r_7=10, r_8=r_9=\dots=r_{17}=5, m_i=r_i^2, i=1, 2, \dots, 17$
Instance 2.5	$n=37$	0.01	$r_i=20, m_i=r_i^2, i=1, 2, \dots, 37$
Instance 2.6	$n=50$	0.01	$r_i=20, m_i=r_i^2, i=1, 2, \dots, 50$

图 2 TSH 算法求得的第一组 5 个算例的最优布局图

Figure 2 Geometric configurations of the optimal solutions obtained by TSH for the first set of five instances.

(a) Instance 1.1; (b) instance 1.2; (c) instance 1.3; (d) instance 1.4; (e) instance 1.5

(MPSO, mutation particle swarm optimization)^[14], 自适应粒子群算法 (APSO, adaptive particle swarm optimizer)^[16], 改进的拟物算法 (IQP, improved quasi-physical)^[17], 改进的散射搜索算法 (ISS, improved scatter search)^[19] 以及吸引盘填充算法 (BF, basin filling)^[21] 的计算结果形成对比. 从表 3 可以看出, 对于算例 1.1 和 1.2, 本文算法得到了理论上的最优圆形容器半径. 需要指出的是, 文献 [17, 19, 21] 以及本文均采用了 Huang 等^[18,29] 提出的拟物法的能量函数作为优化的目标函数, 但文献 [17, 19] 中的算法求得公式 (2) 的目标函数值分别只满足 $U(X) < 5 \times 10^{-6}$ 和 $U(X) < 10^{-8}$, 而文献 [21] 和本文算法求得最终解的目标函数值满足 $U(X) < 10^{-20}$, 也就是说, 文献 [21] 和本文算法求得的解为问题的精确解. 对于算例 1.3–1.5, 本文得到的圆形容器的半径比其他 5 种算法的结果明显要优 (用粗体显示), 而且, 随着问题规模的增大, TSH 算法的优势更加明显. 事实上, 对于算例 1.3, TSH 算法求得的圆形容器半径比目前文献最好结果减少了 $(22.249 - 22.2462033) / 22.249 \times 100\% = 0.01\%$; 对于算例 1.4, TSH 算法求得的结果比目前最好结果减少了 $(31.8546 - 31.8411336) / 31.8546 \times 100\% = 0.04\%$; 对于算例 1.5, TSH 算法求得的结果比目前最好结果减少了 $(725.0435 - 716.6109319) / 725.0435 \times 100\% = 1.16\%$. 另外, 对于各个算例的结果的静不平衡量, 本文算法与 BF 的结果基本相当, 但比 MPSO, APSO, IQP 和 ISS 的结果普遍要好, 只有算例 1.4 的静不平衡量比 ISS 的结果要差. TSH 算法求得的第一组 5 个算例的最优布局如图 2 所示.

对于各个算例的计算时间, 由于所用机器的性能和编程的语言不同, 所以不便作详细的比较, 但从表 3 不难看出, TSH 算法在各个算例上的运行时间是较快的, 至少与其他几个算法的运行时间是相当的.

对于第二组 6 个算例, 我们分别取惩罚因子 $l=10^{-5}, 5 \times 10^{-9}, 10^{-6}, 10^{-6}, 10^{-6}$ 和 10^{-6} . 对于每个算例, 独立运行 TSH 算法 5 次, 并将获得的最小圆形容器半径 $R_0(\text{mm})$ 以及静不平衡量 $J(\text{g}\cdot\text{mm})$

表 3 TSH 算法与 MPSO^[14]、APSO^[16]、IQP^[17]、ISS^[19] 和 BF^[21] 在第一组算例上的计算结果比较**Table 3** Comparison of the results obtained by HTS, MPSO [14], APSO [16], IQP [17], ISS [19] and BF [21] for the first set of instances

Instance	Results of MPSO (2004) ^{a)}			Results of APSO (2006) ^{b)}			Results of IQP (2006) ^{c)}		
	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$
Instance 1.1	120.711	2.712×10^{-3}	287	120.7107	7.03×10^{-4}	263	120.710	7.63×10^{-4}	3.82
Instance 1.2	—	—	—	—	—	—	—	—	—
Instance 1.3	—	—	—	—	—	—	22.327	3.9×10^{-4}	2.91
Instance 1.4	31.985	1.82×10^{-2}	1002	31.885	2×10^{-6}	982	31.981	4.11×10^{-3}	76.33
Instance 1.5	843.94	3.9×10^{-3}	2523	820.987	7.26×10^{-4}	2318	742.75	5.4×10^{-4}	12.27
Instance	Results of ISS (2009) ^{d)}			Results of BF (2010) ^{e)}			Results of TSH in this paper ^{f)}		
	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$
Instance 1.1	120.7102	8.45×10^{-8}	1.275	120.7106782	3.89×10^{-9}	0.594	120.7106782	3.25×10^{-9}	0.350
Instance 1.2	1.00005	1.78×10^{-4}	4.63	1.000000	9.70×10^{-9}	2.891	1.0000000	2.38×10^{-9}	0.225
Instance 1.3	—	—	—	22.2490	3.12×10^{-9}	1.313	22.2462033	2.63×10^{-9}	0.506
Instance 1.4	31.954	5.68×10^{-14}	20.18	31.8546	1.50×10^{-9}	54.828	31.8411336	5.89×10^{-9}	27.766
Instance 1.5	740.58	4.01×10^{-5}	20.21	725.0435	2.47×10^{-9}	7.187	716.6109319	1.77×10^{-9}	63.263

a) MPSO is run on a PC with 166 MBHz CPU, and the memory and the language of programming are not reported

b) APSO is run on a Pentium III PC with 2.0 GHz CPU, 256 MB of RAM, and coded in MATLAB language

c) and d) IQP and ISS are run on a Pentium IV PC with 2.4 GHz CPU, 512 MB of RAM, and coded in C language

e) and f) BF and TSH are run on a Notebook PC with Intel Core 2 Duo, 1.6 GHz CPU, 1 G of RAM, and coded in

Java language

表 4 算法 TSH 和 IQP 在第二组算例上的计算结果比较**Table 4** Comparison of the results obtained by TSH and IQP for the second set of instances

Instance	Results of IQP (2006) ^{a)}			Results of TSH in this paper ^{b)}		
	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$	$R_0(\text{mm})$	$J(\text{g}\cdot\text{mm})$	$T(\text{s})$
Instance 2.1	60	3.6×10^{-3}	7.78	60.0000000	7.38×10^{-9}	0.206
Instance 2.2	215.47	9.5×10^{-3}	444.78	215.4700539	3.77×10^{-7}	3.042
Instance 2.3	39.78	7.6×10^{-3}	91.92	39.0651048	4.13×10^{-8}	365.797
Instance 2.4	49.72	5.1×10^{-3}	157.92	49.3681942	2.50×10^{-8}	597.719
Instance 2.5	135.176	6.7×10^{-3}	18.29	135.1754097	4.91×10^{-9}	0.425
Instance 2.6	159.57	8.0×10^{-3}	348.97	158.9676981	3.49×10^{-9}	1078.386

a) IQP is run on a Pentium IV PC with 2.4 GHz CPU, 512 MB of RAM, and coded in C language

b) TSH is run on a Notebook PC with Intel Core 2 Duo, 1.6 GHz CPU, 1 G of RAM, and coded in Java language

和运行时间 $T(\text{s})$ 的平均值分别列于表 4 中, 并与改进的拟物算法 (IQP)^[17] 的计算结果形成对比. 从表 4 可以看出, 对于算例 2.1 和 2.2, 本文算法得到了理论上的最优圆形容器半径. 对于算例 2.3–2.6, 本文 TSH 算法找到了比 IQP 算法更小的圆形容器半径 (用粗体显示). 另外, 对于各个算例的结果的静不平衡量 J , TSH 算法的结果比 IQP 的结果明显要好. 由于上面所提的原因, 在此我们也不详细地比较 TSH 和 IQP 的运行时间. TSH 算法求得的第二组 6 个算例的最优布局如图 3 所示.

图 3 TSH 算法求得的第二组 6 个算例的最优布局图

Figure 3 Geometric configurations of the optimal solutions obtained by TSH for the second set of six instances
(a) Instance 2.1; (b) instance 2.2; (c) instance 2.3; (d) instance 2.4; (e) instance 2.5; (f) instance 2.6

表 5 TSH 算法得到的算例 1.5 的最优格局的坐标

Table 5 The coordinates of optimal configuration found by TSH for instance 1.5

No.	x	y	No.	x	y
1	-233.71720864089102	131.01400188236994	21	94.51648076930135	-601.2269964756864
2	300.6716238118119	-225.37520710383689	22	537.2594606823667	330.18543174871854
3	-0.057693393835633336	253.13963628620334	23	-14.764167938050669	443.57261558988995
4	399.82648784594323	462.82470940558983	24	-38.5174405795553	-239.54022355932284
5	35.734535862199785	622.5862489100716	25	224.13511425028005	572.2849361734114
6	-16.142459543226476	52.784251287652374	26	352.1741895013926	256.20394103026155
7	-394.807375170643	239.15475419533522	27	129.05063776873783	-367.76634850526693
8	-106.07475849909758	-614.039284669713	28	171.61718209614327	369.7607697660905
9	-243.76293454049215	-309.99082475418874	29	-162.81159165854615	585.3916895429936
10	-519.6697395369367	365.9844512549746	30	625.5863928808128	-5.541042786772165
11	-417.49113082009103	65.80552446667465	31	479.8699242479468	-369.44479511702764
12	449.6787771621979	83.8745693421458	32	96.9198639141864	-104.86380932334733
13	266.44099766190544	1.2615185342280424	33	-581.5180569575209	194.81557081159627
14	-604.854648552889	-8.851813648231888	34	-62.13368538808253	-432.9779623410568
15	-361.0917866884603	481.1757394254895	35	-186.92598743425185	339.83589158263777
16	-291.70178546953247	-531.8711225987728	36	-184.99034390212572	-83.52199195719405
17	-406.65283762306007	-136.90493752425215	37	303.75596733608626	-515.8171770093842
18	169.26518108722482	173.77488228626726	38	-599.8444028071675	-197.7853930817572
19	439.45983274485945	-89.82509437368967	39	605.8427781452938	171.3605956374431
20	-475.2939217617528	-360.6109981669278	40	588.4338895441839	-190.85345871138736

表 6 TSH 算法得到的算例 2.6 的最优格局的坐标

Table 6 The coordinates of optimal configuration found by TSH for instance 2.6

No.	x	y	No.	x	y
1	-35.332462034864605	-134.40103512164566	26	52.335661605537496	100.37744611506206
2	-58.738370071956005	9.998752301499549	27	-15.850921287980054	-99.46581549836533
3	-15.23453086938994	-31.16339469228629	28	-27.423798828575276	96.19254159449736
4	32.50134835517449	135.11359469094953	29	-119.17842007986359	-68.1418192226286
5	-95.84245587672544	-100.62924409302478	30	-62.533498039505936	77.02723384018545
6	60.625867182704376	-10.890717790304366	31	-18.760084051516134	8.680933320114603
7	5.5603671990223935	-65.62644588290986	32	36.245395486849006	63.75636835608089
8	132.35536069337707	-42.35657695021956	33	53.41878712482017	-128.2905074326564
9	-55.846479341971914	-100.06191611833331	34	-98.7333769621281	9.366752416269424
10	-36.36493859507163	-65.12669649505298	35	10.268846465919694	-138.58777690501162
11	87.7482050838274	-107.76025992817152	36	117.97696273484603	73.44016189439523
12	15.697403893753947	28.995497664476666	37	-95.35263362022751	-30.490123708884273
13	-138.71929253968872	8.305359263833658	38	79.42309146446452	62.781878258333066
14	95.57655722026507	25.70335597416508	39	39.88978515805387	-45.09619837845917
15	114.80768856539429	-78.30208018620311	40	-55.21281688983165	-29.84557571090281
16	-111.51764290580806	82.63038844739233	41	-135.33854919778935	-31.551516861322636
17	-48.335950050750654	130.2906637020114	42	55.6969476152741	28.804442517943095
18	-7.498195366359793	135.3046498374579	43	-130.5914072611783	47.470873187427834
19	-3.754148234682075	63.947423502588094	44	100.50547678771078	-13.99180433407762
20	-76.36091512982578	-65.69402446975023	45	-83.44564926169468	111.12535594771875
21	19.22295707586755	-10.848830347909104	46	24.117238731561926	-101.06149236128749
22	12.336117884009687	100.56850126157023	47	134.008478127183	36.79332691203551
23	85.50614017212047	-51.073065114845406	48	-90.60549168361686	48.532266339860364
24	58.44665669056941	-80.53124485682662	49	-38.80777261175304	44.67974543721065
25	92.17106177980284	104.00248307420192	50	138.93739769463645	-2.9018333962010274

表 5 和 6 分别给出了算例 1.5 和 2.6 的最优格局的坐标, 在表 5 和 6 中, x 和 y 分别表示最优格局中相应圆形待布物的中心的坐标.

6 结论

对于带平衡约束的圆形装填问题, 本文提出了一个基于禁忌搜索的启发式 (TSH) 求解算法. 算法从任一初始格局出发, 应用基于自适应步长的梯度法进行极小化计算. 为了使计算能有效地逃离局部极小点的陷阱且避免迂回搜索, 算法采用了禁忌搜索的策略. 在禁忌搜索的过程中, 我们对传统的邻域解、禁忌对象以及当前解接受原则进行了有效的改进. 通过应用 TSH 算法对两组共 11 个有代表性的算例进行测试, 计算结果表明, 该算法是求解带平衡约束的圆形装填问题的有效算法, 刷新了其中 7 个算例的当今国际上的最好纪录, 对于其余 4 个算例, 该算法均达到问题的最优解. 从算法的实现过

程不难看出, 该算法稍作修改便可推广应用于其他各种装填布局问题. 我们希望在不久的将来能将此方法应用于矩形、三角形以及任意的多边形装填问题.

参考文献

- 1 Lodi A, Martello S, Monaci M. Two-dimensional packing problems: a survey. *Eur J Oper Res*, 2002, 141: 241–252
- 2 Birgin E G, Martinez J M, Ronconi D P. Optimizing the packing of cylinders into a rectangular container: a nonlinear approach. *Eur J Oper Res*, 2005, 160: 19–33
- 3 Wang Y S, Teng H F. Knowledge fusion design method: satellite module layout. *Chin J Aeronaut*, 2009, 22: 32–42
- 4 Leung J, Tam T, Wong C S, et al. Packing square into square. *J Parallel Distr Com*, 1990, 10: 271–275
- 5 Huang W Q, Liu J F. A deterministic heuristic algorithm based on Euclidian distance for solving the rectangles packing problem. *Chin J Comput*, 2006, 29: 734–739
- 6 Huang W Q, He K. A caving degree approach for the single container loading problem. *Eur J Oper Res*, 2009, 196: 93–101
- 7 Wei L J, Zhang D F, Chen Q S. A least wasted first heuristic algorithm for the rectangular packing problem. *Comput Oper Res*, 2009, 36: 1608–1614
- 8 Chen D B, Liu J F, Fu Y, et al. An efficient heuristic algorithm for arbitrary shaped rectilinear block packing problem. *Comput Oper Res*, 2010, 37: 1068–1074
- 9 Liu J F, Xue S J, Liu Z X, et al. An improved energy landscape paving algorithm for the problem of packing circles into a larger containing circle. *Comput Ind Eng*, 2009, 57: 1144–1149
- 10 Teng H F, Sun S L, Ge W H, et al. Layout optimization for the dishes installed on a rotating table—the packing problem with equilibrium behavioral constraints. *Sci China Ser A-Math*, 1994, 37: 1272–1280
- 11 Tang F, Teng H F. A modified genetic algorithm and its application to layout optimization. *J Softw*, 1999, 10: 1096–1102
- 12 Qian Z Q, Teng H F, Sun Z G. Human-computer interactive genetic algorithm and its application to constrained layout optimization. *Chin J Comput*, 2001, 24: 553–559
- 13 Yu Y, Cha J Z, Tang X J. Learning based GA and application in packing. *Chin J Comput*, 2001, 24: 1242–1249
- 14 Li N, Liu F, Sun D B. A study on the particle swarm optimization with mutation operator constrained layout optimization. *Chin J Comput*, 2004, 27: 897–903
- 15 Zhou C, Gao L, Gao H B. Particle swarm optimization based algorithm for constrained layout optimization. *Control Decision*, 2005, 20: 36–40
- 16 Lei K Y, Qiu Y H. A study of constrained layout optimization using adaptive particle swarm optimizer. *J Comput Res Develop*, 2006, 43: 1724–1731
- 17 Huang W Q, Chen M. Note on: An improved algorithm for the packing of unequal circles within a larger containing circle. *Comput Ind Eng*, 2006, 50: 338–344
- 18 Wang H Q, Huang W Q, Zhang Q A, et al. An improved algorithm for the packing of unequal circles within a larger containing circle. *Eur J Oper Res*, 2002, 141: 440–453
- 19 Wang Y S, Shi Y J, Teng H F. An improved scatter search for circles packing problem with the equilibrium constraint. *Chin J Comput*, 2009, 32: 1214–1221
- 20 Blum C, Roli A. Meta-heuristic in combinatorial optimization: overview and conceptual comparison. *ACM Comput Survey*, 2003, 35: 268–308
- 21 Liu J F, Li G. Basin filling algorithm for the circular packing problem with equilibrium behavioral constraints. *Sci China Inf Sci*, 2010, 53: 885–895
- 22 Hansmann U H E, Wille L T. Global optimization by energy landscape paving. *Phys Rev Lett*, 2002, 88: 068105
- 23 Glover F. Tabu search: Part I. *ORSA J Comput*, 1989, 1: 190–206
- 24 Glover F. Tabu search: Part II. *ORSA J Comput*, 1990, 2: 4–32
- 25 Glover F, Laguna M. *Tabu Search*. Boston: Kluwer Academic Publishers, 1997

- 26 Huang W Q, Kang Y. A short note on a simple search heuristic for the disk packing problem. *Ann Oper Res*, 2004, 131: 101–108
- 27 Lü Z P, Huang W Q. PERM for solving circle packing problem. *Comput Oper Res*, 2008, 35: 1742–1755
- 28 Zhang D F, Deng A S. An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Comput Oper Res*, 2005, 32: 1941–1951
- 29 Huang W Q, Xu R C. Two personification strategies for solving circles packing problem. *Sci China Ser E-Tech Sci*, 1999, 42: 595–602

Heuristic algorithm based on tabu search for the circular packing problem with equilibrium constraints

LI Gang¹ & LIU JingFa^{2,3*}

1 *School of Mathematics and Physics, Nanjing University of Information Science & Technology, Nanjing 210044, China;*

2 *Network Information Center, Nanjing University of Information Science & Technology, Nanjing 210044, China;*

3 *School of Computer and Software, Nanjing University of Information Science & Technology, Nanjing 210044, China*

*E-mail: jfliu@nuist.edu.cn

Abstract The circular packing problem with equilibrium constraints is an optimization problem about simplified satellite module layout design. A heuristic algorithm based on tabu search for solving this problem is put forward. The algorithm begins from a random initial configuration and applies the gradient method with an adaptive step length to search for the minimum energy configuration. To jump out of the local minima and avoid the search from doing repeated work, the algorithm adopts the strategy of tabu search. In the process of tabu search, we improve the traditional neighboring solutions, tabu objects and the acceptance criteria of the current solution effectively. We test two sets of benchmarks consisting of 11 representative instances from the current literatures. The numerical results show that the proposed algorithm breaks the records in 7 out of 11 instances, and obtains the optimal solutions for the other 4 instances.

Keywords equilibrium constraints, packing problem, heuristic algorithm, tabu search, layout optimization