中国科学: 信息科学 2015 年 第 45 卷 第 9 期: 1127–1140
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS

论 文

求解二维矩形 Packing 问题的一种优美度枚举算法

王磊[①②*], 尹爱华[③]

① 武汉科技大学计算机科学与技术学院, 武汉 430081
② 智能信息处理与实时工业系统湖北省重点实验室, 武汉 430081
③ 江西财经大学软件与通信工程学院, 南昌 330013
* 通信作者. E-mail: wanglei77@wust.edu.cn

收稿日期: 2014–10–09; 接受日期: 2015–01–08; 网络出版日期: 2015–05–22
国家自然科学基金 (批准号: 61262011, 61100055, 61472293)、湖北省自然科学基金 (批准号: 2014CFC1121) 和江西省自然科学基金 (批准号: 20142BAB207024) 资助项目

摘要 针对二维矩形 Packing 问题, 提出了基于角区的基本算法. 在此基础上提出了优美度枚举算法. 计算了两组有代表性的问题实例 c1∼c21 和 zdf1∼zdf16, 算法的表现优于当前文献中报道的表现领先的优秀算法. 针对矩形块方向固定的情形, 算法对 zdf6∼zdf9 得到了比此前国际上已报道记录更优的布局, 其中对 zdf8 和 zdf9 首次找到最优布局.

关键词 Packing 问题 NP 难度 组合优化 启发式算法 拟人

1 引言

二维矩形 Packing 问题是指: 在二维欧式空间中, 已知一个大矩形框和有穷个小矩形块. 大矩形框和小矩形块的宽度和长度为已知的正实数. 矩形框的宽度和长度记为 W 和 H. 第 i 个小矩形块的宽度和长度记为 w_i 和 h_i. 在满足约束条件的前提下, 要求将这些矩形块放入矩形框内, 目标是使得矩形框的面积利用率尽可能地大. 约束条件为: 第一, 放入的小矩形块须完全在框内. 第二, 矩形块的每条边均和矩形框的某条边平行或重合. 第三, 每两个小矩形块相互不重叠.

二维矩形 Packing 问题是一个典型的 NP 难度问题, 具有理论价值. 在实际生产的不同领域中, 有各种形态的二维矩形 Packing 问题. 例如, 在钢板、木板、服装和玻璃等的切割加工过程中, 都需要使得材料得到最大的利用以此来达到减少原料浪费和节约成本的目的. 二维矩形 Packing 问题还出现在报纸、网页等的排版中, 使得报纸、网页的版面得到充分利用. 在集装箱运输中如何布局以便装置最多的货物, 在集成电路的设计中如何布置以使得电子元件的数量达到最多, 这些问题都可以形式化为一个矩形 Packing 问题. 因此设计这类问题的高效算法, 具有实际价值.

本文讨论的是一种经典的二维矩形 Packing 问题, 其求解算法对于实际生产中的板材切割、版面布局、集装箱运输、集成电路设计有其借鉴作用.

求解二维矩形 Packing 问题的算法分为完整算法和近似算法两类. 完整算法 [1] 保证找到最优布局, 但是所花时间太长, 仅可用于计算较小规模的问题实例. 文献 [1] 中报道的 1CBP+GRASP 算法为

引用格式: 王磊, 尹爱华. 求解二维矩形 Packing 问题的一种优美度枚举算法. 中国科学: 信息科学, 2015, 45: 1127–1140, doi: 10.1360/N112014-00157

当前文献中表现领先的完整算法, 其测试最大规模的实例仅为 200 个矩形块. 因此本文的算法未与之比较.

近几十年来, 国内外学者提出了多种高效近似算法, 可分为非随机型和随机型近似算法.

非随机型近似算法着力于提出选择放置动作的优先序, 并在其基础上进行适当的枚举, 包括底部左齐择优匹配算法 [2]、分支限界 [3]、拟人算法 [4~8]. 文献 [8] 提出的两种非随机型近似算法 A_0 和 A_1 基于动作空间, 是已知文献中领先的非随机型近似算法. 动作空间定义的引入便于计算出合理的占角动作. 基本算法 A_0 在多个占角动作中优先选取与当前动作空间相贴边多、对剩余空间损害小、与其他已放入块关系更紧密的动作. 增强算法 A_1 在优先序中排名前 N 名的占角动作的每一轮考察中, 通过向前看并回溯的方法选择一个动作.

随机型近似算法使用各种放置矩形块的策略, 并做一定的随机化处理, 包括遗传算法 [9]、模拟退火 [10]、粒子群算法 [11]、随机局部搜索 [12]. 随机局部搜索算法 BSHA 是已知文献中领先的随机型近似算法. BSHA 算法也将矩形块放在角区. 但与 A_0 和 A_1 算法不同的是, BSHA 算法规定: 对每个已放入矩形块来说, 后放入矩形块的位置必须至少满足以下两条件之一. 第一, 后放入矩形块的下沿在已放入矩形块上沿之上或与其重合; 第二, 后放入矩形块的左沿在已放入矩形块右沿的右边或与其重合. 这样处理的优势是节省计算时间, 但其缺点是浪费量较大. BSHA 算法基于浪费量小优先策略和大矩形块优先策略. 首先将所有待放矩形块按照面积从大到小的顺序排序. 在多个占角动作中选取浪费量与矩形块面积的比值 (第 1 项指标) 最小的动作. 若多个占角动作的第 1 项指标并列最小, 则 BSHA 算法选择动作做完后剩余空间角区数 (第 2 项指标) 最少的动作, 使得剩余空间较平整. 若多个占角动作的第 2 项指标并列最小, 则 BSHA 算法选择矩形块左下角位置 (第 3 项指标) 在最左边的动作. 若多个占角动作的第 3 项指标并列最小, 则 BSHA 算法选择在矩形块序列中排序较前的矩形所对应的动作.

BSHA 算法所得终止格局与矩形块序列有关, 因此对初始给定的矩形块序列 (面积降序排列) 做一定的随机化处理后, 按照不同的矩形块序列, 算法可生成不同的终止格局. 算法取多次随机计算所得最好布局.

本文首先给出基本定义、基本算法, 然后提出基于优美度的枚举算法, 最后进行实验分析和讨论.

2 求解策略

定义1 (当前格局) 矩形框内已放入若干矩形块, 每个矩形块的位置、方向已知, 外面还有若干个块待放, 这称为一个格局. 初始格局中, 所有矩形块均在框外, 矩形框为空. 终止格局中, 或者所有矩形块均在框内, 或者虽有块在框外但已经无法再放入. 图 1 为一个格局.

定义2 (动作) 将一个矩形块按照指定的方向 (站或躺) 放在指定的位置 (用块左下角坐标表明), 称为一个动作.

定义3 (合理的动作) 矩形块不与矩形框内其他块重叠, 完全在框内, 矩形块每一边与框的某一条边平行或重合.

定义4 (角区) 矩形框中由块或者矩形框本身形成的直角形状的空白区域称为角区, 可分为 90° 角区和 270° 角区. 图 2 中有 6 个角区, 其中角区 1, 2, 4, 5, 6 为 90° 角区, 角区 3 为 270° 角区.

定义5 (占角动作) 我们做一个动作, 使得矩形块的一角正好与 90° 角区的角重合, 这称为占角动作. 图 3 中 A, B, C, D, E 所示均为占角动作. 与一般的合理动作相比, 占角动作在格局中腹留出大

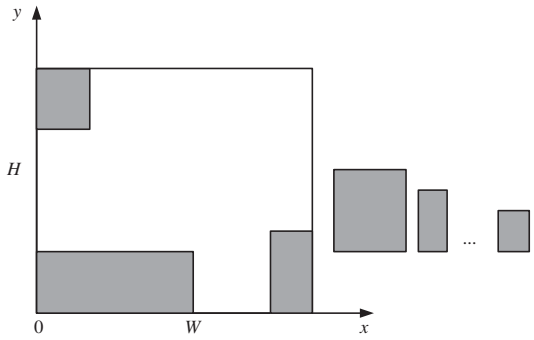

图 1 格局示例

Figure 1 An example of a configuration

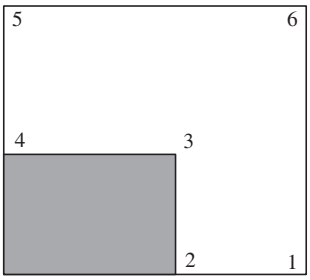

图 2 角区示例

Figure 2 An example of the corner areas

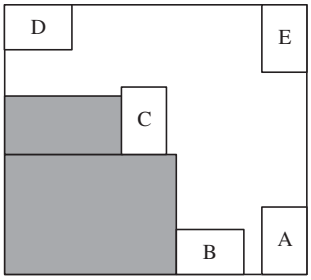

图 3 占角动作示例

Figure 3 An example of the corner-occupying actions

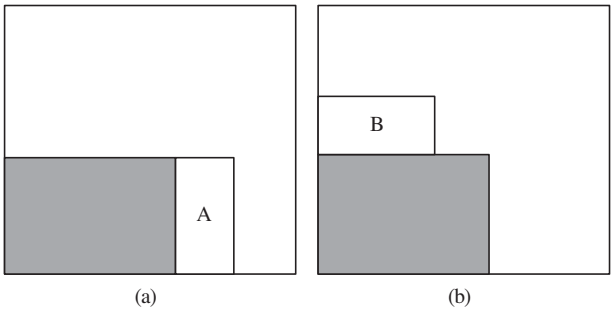

图 4 平整度示例

Figure 4 An example of the degree of planeness

片空白区域, 有利于放置后续矩形块. 角上的材料被利用的机会最少, 要最先被利用, 否则此后有可能无法被利用.

定义6 (剩余空间的平整度) 平整度为 e^{4-n}, 其中 n 为剩余空间的角区数. 剩余空间的角区数越少, 其平整度越高. 图 4(a) 中, 将矩形块放在位置 A, 剩余空间 n 值为 6, 其平整度为 e^{-2}. 图 4(b) 中, 将矩形块放在位置 B, 则剩余空间 n 值为 8, 其平整度为 e^{-4}. 图 4(a) 的剩余空间角区数少、更平整, 有利于放置后续矩形块.

定义7 (占穴动作) 占穴动作是一种特殊的占角动作. 当动作做完后, 如果剩余空间的角区数并

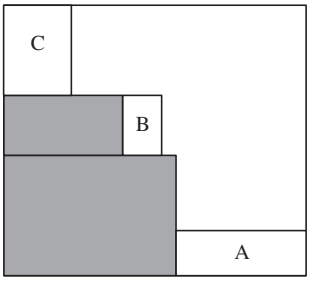

图 5　占穴动作示例

Figure 5　An example of the cave-occupying action

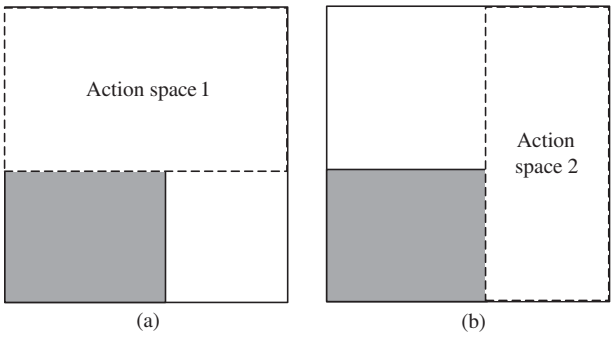

图 6　动作空间示例

Figure 6　An example of the action space

未增加, 则称为占穴动作. 图 5 中, A, B, C 所示均为占穴动作. 本文考虑的实例中, 矩形边长为整数, 因此可能出现占穴动作.

定义8 (动作空间)　在当前格局下, 若往框内合理地放入一个虚拟的矩形块, 使得该块的上下左右 4 条边均与其他已放入的块或框的边相贴 (重合的长度大于 0), 则该虚拟块所占的空间称为当前格局下的一个动作空间 [8].　初始格局下, 恰有一个动作空间, 当放入一个矩形块以后, 形成两个动作空间, 如图 6 所示.

定义9 (占角动作的重叠度)　一个占角动作所对应的矩形块与其所在动作空间的贴边数, 称为其重叠度 [8].　图 7(a) 中, 动作 A 的重叠度为 3. 图 7(b) 中, 动作 B 的重叠度为 2. 重叠度高的动作, 与所在动作空间贴合较好, 对剩余空间损害较小, 有利于放置后续矩形块.

定义10 (矩形块排序的 4 项指标)　(1) 矩形块的周长, 大优先; (2) 矩形块的长边长, 大优先; (3) 矩形块的短边长, 大优先; (4) 矩形块的序号, 小优先.

依字典序按照定义 10 中的 4 项指标将在矩形框外的矩形块排序. 首先考虑第 1 项指标 “周长”, 周长较大的矩形块优先. 若有两个矩形块周长相等, 则考虑第 2 项指标 “长边长”, 长边长较长的矩形块优先. 若长边长仍然相等, 则依此类推, 考虑第 3 项指标、第 4 项指标.

形状大小完全相同的矩形块在待放矩形块序列中相邻, 便于查找和计算.

定义11 (占角动作排序的 9 项指标)　(1) 动作做完后, 框内剩余空间的平整度, 大优先; (2) 动作的重叠度, 大优先; (3) 矩形块的周长, 大优先; (4) 矩形块的长边长, 大优先; (5) 矩形块的短边长, 大优先; (6) 矩形块的序号, 小优先; (7) 矩形块左下顶点的 y 坐标, 小优先; (8) 矩形块左下顶点的 x 坐标,

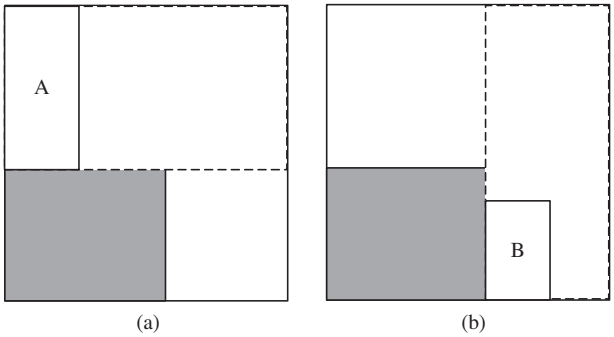

图 7 重叠度示例

Figure 7 An example of the degree of overlapping

小优先; (9) 矩形块的方向, 躺优先.

本文依字典序按照以上 9 项指标选取最优先的占角动作. 算法首先考虑第 1 项指标, 选取动作做完后剩余空间平整度最大的占角动作. 若这样的动作不止一个, 则考虑第 2 项指标, 选取其中重叠度最大的占角动作. 若这样的动作仍不止一个, 则依此类推, 考虑第 3 项指标. 直到选出唯一最优先的占角动作为止.

2.1 基本算法 B_0 的步骤

基本算法命名为 B_0 算法, 其步骤如下.

Step 0 初始格局. 所有矩形块均在矩形框外面. 矩形框是空的. 依字典序按照矩形块排序的 4 项指标 (见定义 10) 对所有矩形块排序.

Step 1 在当前格局下, 依字典序按 9 项指标 (见定义 11) 选择最优先的占角动作来做 (若有多个形状大小完全相同的待放矩形块, 则只需考虑其中一个矩形块的占角动作), 动作做完以后, 演化到新格局.

Step 2 对动作空间序列做如下 3 步操作. 第一, 更新占角动作所在的动作空间. 第二, 检查所放入的矩形块是否与动作空间序列中其他动作空间重叠. 若有重叠, 则更新与之重叠的动作空间. 第三, 删去被其他动作空间包含的动作空间.

依此类推, 循环做 Step 1∼Step 2, 直到终止格局为止.

2.2 优美度枚举算法 B_1 的步骤

基本算法 B_0 是一种非随机型近似算法, 在任何格局, 依 B_0 算法的指挥可走到终止格局. B_0 算法是一个评价工具, 可用于衡量一个动作的优美度.

定义12 (合理动作的优美度) 在当前格局, 定义合理动作的优美度. 动作做完以后, 依基本算法 B_0 的指挥, 走到终止格局, 终止格局中矩形块的总面积称为此动作的优美度 [7].

定义 11 中的 9 项选择占角动作的指标固然有一定道理, 但均有其局限性. 优美度则是一项硬指标, 符合问题的目标即放入矩形框内的矩形块面积和尽可能大.

优美度枚举算法命名为 B_1 算法, 其步骤如下.

Step 0 初始格局. 所有矩形块均在矩形框外面. 矩形框是空的. 我们按照矩形块排序的 4 项指标 (见定义 10) 对所有矩形块排序.

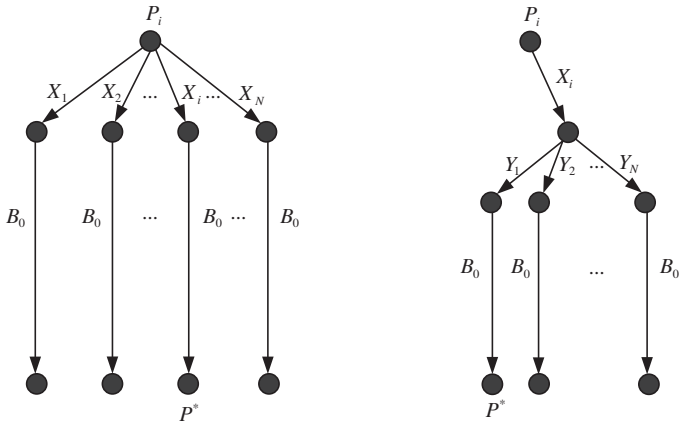

图 8　优度定理 1 示例

Figure 8　The illustration of the Theorem 1 of the goodness degree

Step 1 一轮计算的计算过程为: 在当前格局下, 我们首先依字典序按 9 项指标 (见定义 11) 对所有占角动作排序, 取排名为前 N 名的动作为候选动作. 计算这 N 个候选动作的优美度. 最后选择其中优美度最大的动作. 若优美度最大的动作不止一个, 则依字典序按照定义 11 中 9 项指标选择其中最优先的动作. 动作做完以后, 演化到新格局.

Step 2 更新动作空间序列.

依此类推, 循环做 Step 1∼Step 2, 直到算法结束.

优美度枚举算法在每一轮计算一个占角动作的优美度时, 调用基本算法计算出一个终止格局. 在多个终止格局中选最优的布局, 因此有可能找到更优的布局.

定义13 (布局的优度)　给定一个二维矩形 Packing 问题实例的布局, 其优度是指放入矩形框的矩形块的面积和.

定理1 (优度定理 1)　优美度枚举算法第 $i+1$ 轮计算所得最好布局的优度高于或等于第 i 轮计算所得最好布局的优度.

证明　第 i 轮计算前夕, 格局记为 P_i. 第 i 轮计算的计算过程为: 首先依字典序按 9 项指标对所有占角动作排序, 取排名为前 N 名的动作为候选动作, 记为 $X_1, X_2, \ldots, X_N$. 我们计算动作 X_1 的优美度: 首先试做动作 X_1, 然后调用 B_0 算法计算到终止格局, 算出终止格局中放在矩形框内的矩形块的面积和, 作为动作 X_1 的优美度. 回溯到格局 P_i, 计算下一个动作 X_2 的优美度. 依次类推, 计算这 N 个候选动作的优美度. 设其中优美度最高的动作记为 X_i, 选择此动作做, 演化到格局 P_{i+1}. 第 i 轮计算所得到的最好布局是: 从格局 P_i 开始, 首先做动作 X_i, 演化到格局 P_{i+1}, 然后用 B_0 算法计算到终止格局. 此格局记为 P^*, 参见图 8.

格局 P^* 与第 $i+1$ 轮计算中当我们计算排第 1 的候选动作 Y_1 的优美度时所得到的布局完全相同. 第 $i+1$ 轮计算从格局 P_{i+1} 开始. 计算排第 1 的候选动作 Y_1 的优美度时, 首先试做动作 Y_1, 然后调用 B_0 算法计算到终止格局. 此终止格局与第 i 轮所得到的最好布局 P^* 完全相同. 原因是其计算过程均为从格局 P_{i+1} 开始, 循环选择排第 1 的候选占角动作来做.

第 $i+1$ 轮计算中, 计算前 N 名候选占角动作的优美度, 得到 N 个布局, 包含第 i 轮计算所得最好布局 P^*, 取其中优度最高的布局. 因此优美度枚举算法第 $i+1$ 轮计算所得最好布局的优度高于或等于第 i 轮计算所得最好布局的优度.

基于定理 1, 对优美度枚举算法做如下改进: 第 $i+1$ 轮计算中, 对排第 1 名的候选动作不必计算其优美度, 其优美度等于第 i 轮计算所得布局的优度.

定理2 (优度定理 2) 优美度枚举算法所得布局的优度高于或等于基本算法所得布局的优度.

证明 B_1 算法在第 1 轮计算中依字典序按 9 项指标对所有占角动作排序, 取排名为前 N 名的动作为候选动作. 计算这 N 个候选动作的优美度. 其中排第 1 名的候选动作正是 B_0 算法所选的第 1 个动作. B_1 算法在计算第 1 名候选动作的优美度时, 首先做这个动作, 然后用 B_0 算法计算, 除第 1 个动作外, 每一步所选择的动作都用 B_0 算法选择. 因此其格局演化的过程与 B_0 算法完全相同, 所得终止格局与 B_0 算法所得终止格局完全相同.

在第 1 轮计算中每计算一个占角动作的优美度, B_1 算法都会得到一个终止格局, 其中包括 B_0 算法所得终止格局. B_1 算法选择所有这些终止格局中优度最高的, 因此第 1 轮计算后, B_1 算法所得布局优度高于或等于 B_0 算法所得终止格局. 再根据定理 1 可知, 当 B_1 算法结束时所得布局优度高于或等于 B_0 算法所得终止格局.

2.3 另一种基本算法 C_0 和优美度枚举算法 C_1 的步骤

将定义 11 中的指标 "矩形块的周长, 大优先" 删去. B_0 算法变为另一种基本算法 C_0. B_1 算法变为另一种优美度枚举算法 C_1. 基于人们在装箱时所使用的 "大矩形块优先" 和 "长矩形块优先" 策略, B_0 算法和 B_1 算法优先放置周长较大的矩形块, C_0 算法和 C_1 算法优先放置长边长较长的矩形块. 对于 C_0 和 C_1 算法, 定理 1 和 2 依然成立, 证明过程类似.

3 实验测试

本文的算法对两组 Strip Packing 问题实例 c 和 zdf 做了测试. Strip Packing 问题是指: 矩形框的宽度 W 固定, 要求用长度 H 尽可能小的矩形框将所有矩形块装入. 两组问题实例的具体数据来自网络[1)].

我们结合 B_1 算法和 C_1 算法 (记为 B_1C_1) 用于计算二维 Strip Packing 问题. 思路为: 首先调用 B_1 算法和 C_1 算法按照顺序查找的方式计算, 如果全部矩形块放入了矩形框, 则结束; 否则调用 B_1 算法和 C_1 算法按照跳跃式查找的方式计算, 直到找到一个布局将全部矩形块放入矩形框为止.

具体算法计算步骤用伪 C 语言表示如下.

Step 0 首先算出矩形框长度的下界 LB. $LB = \lceil \frac{\text{sum}}{W} \rceil$, 其中 sum 表示所有矩形块的面积和, $\lceil\ \rceil$ 表示向上取整.

Step 1 调用 B_1 算法和 C_1 算法作顺序查找:

```
for(l = LB; l ⩽ LB + k; l++)
{
    finish_flag = 0;
    将矩形框的长度设定为 l;
    for(N = 5; N ⩽ 205; N+ = 100)
    {
        调用 B1 算法计算 time_ub 秒;
        若矩形块已全部放入矩形框, 则 UB = l, finish_flag = 1, break;
        调用 C1 算法计算 time_ub 秒;
```

1) http://paginas.fe.up.pt/esicup/tiki-index.php.

若矩形块已全部放入矩形框, 则 $UB = l$, finish_flag = 1, break;
}
if(finish_flag == 1)
break;
}
当 Step 1 结束时, 如果 finish_flag 值为 1, 表示算法已经找到一个布局, 则输出 UB 值, 算法成功停机; 否则转 Step 2.
Step 2 调用 B_1 算法和 C_1 算法作跳跃式查找, d 为跳跃步长:
for($l = LB + k + d$; ; $l+ = d$)
{
finish_flag = 0;
将矩形框的长度设定为 l;
for($N = 5$; $N \leqslant 205$; $N+ = 100$)
{
调用 B_1 算法计算 time_ub 秒;
若矩形块已全部放入矩形框, 则 $UB = l$, finish_flag = 1, break;
调用 C_1 算法计算 time_ub 秒;
若矩形块已全部放入矩形框, 则 $UB = l$, finish_flag = 1, break;
}
if(finish_flag == 1)
break;
}
当 Step 2 结束时, 输出 UB 值, 算法结束.

为降低优美度枚举算法的时间复杂度, 在以上顺序查找和跳跃式查找中, 给定矩形框的长度和 N 值, 执行一次优美度枚举算法的上限定为 time_ub 秒. 当达到时间上限时, 即便计算尚未结束, 也会停止本次优美度枚举算法的执行.

实例规模越大, 优美度枚举算法调用基本算法计算所需时间越长, 因此上限 time_ub 设置得越高.

顺序查找的优点是所找到的 UB 值较小, 但所花时间较长. 跳跃式查找的优点和缺点正好与之相反. 本文结合两种查找方法, 首先执行 k 次顺序查找, 然后按照步长 d 做跳跃式查找.

实例规模越大, 将 k 值设置得越小以避免花过多计算时间用于顺序查找, 将步长 d 值设置得越大以避免花过多时间用于跳跃式查找.

具体分为以下 4 种情形:

第一, 对小规模实例 (矩形块数小于等于 200), time_ub = 600, $k = 100$, $d = 5$;

第二, 对中规模实例 (矩形块数大于 200 且小于等于 1000), time_ub = 1000, $k = 10$, $d = 10$;

第三, 对大规模实例 (矩形块数大于 1000 且小于等于 10000), time_ub = 2000, $k = 1$, $d = 50$;

第四, 对超大规模实例 (矩形块数大于 10000), time_ub = 25000, $k = 0$, $d = 100$.

N 值是 B_1 算法和 C_1 算法的重要参数. N 值较小时, 枚举的占角动作较少, 计算时间较少, 但可能漏掉好的占角动作. N 值较大时, 枚举的占角动作较多, 所花时间也较多. 为结合小 N 值和大 N 值的优点, 取长补短, 本文将 N 值从较小的 5 开始, 每次递增 100, 直到 205 为止.

我们用 C 语言编程, 在 CPU 为 1.8 GHz 的微机上进行了计算. 算法计算结果与当前文献中表现领先的非随机型近似算法 A_1 和随机型近似算法 BSHA 比较, 参见表 1~5. 表 1~5 中的符号含义说明如下: Instance 是问题实例, 本文列出其名字 (Name)、矩形块数 (n)、矩形框宽度 (W)、矩形框长度下界 (H^*)4 项数据. 对非随机型近似算法 A_1 和 B_1C_1 列出两项数据: H 表示算法算出的矩形框长

表 1 A_1, BSHA 和 B_1C_1 在 c1∼c21 上的计算结果比较 (矩形块可旋转 90°)

Table 1 Comparation of the computational results of A_1, BSHA, and B_1C_1 on c1∼c21 (the rectangles can rotate 90°)

Instance				A_1		BSHA (10 tests)			B_1C_1	
Name	n	W	H^*	H	Time (s)	Mean H	Best H	Mean time (s)	H	Time (s)
c1	16	20	20	**20***	0.02	**20***	**20***	0.03	**20***	0.03
c2	17	20	20	**20***	0.19	**20***	**20***	0.03	**20***	0.05
c3	16	20	20	**20***	0.00	**20***	**20***	0.01	**20***	0.00
c4	25	40	15	**15***	0.28	**15***	**15***	0.13	**15***	0.03
c5	25	40	15	**15***	0.05	**15***	**15***	0.03	**15***	0.00
c6	25	40	15	**15***	0.02	**15***	**15***	0.12	**15***	0.03
c7	28	60	30	**30***	0.50	**30***	**30***	0.25	**30***	0.47
c8	29	60	30	**30***	1.27	**30***	**30***	1.65	**30***	0.81
c9	28	60	30	**30***	2.61	**30***	**30***	1.59	**30***	0.26
c10	49	20	60	**60***	23.81	**60***	**60***	1.91	**60***	0.42
c11	49	20	60	**60***	0.17	**60***	**60***	3.05	**60***	0.36
c12	49	20	60	**60***	10.06	**60***	**60***	2.79	**60***	0.21
c13	73	60	90	**90***	4.13	**90***	**90***	8.93	**90***	0.03
c14	73	60	90	**90***	0.17	**90***	**90***	0.51	**90***	0.08
c15	73	60	90	**90***	11.66	**90***	**90***	3.03	**90***	0.86
c16	97	80	120	**120***	74.06	**120***	**120***	12.63	**120***	2.87
c17	97	80	120	**120***	2.92	**120***	**120***	1.28	**120***	0.97
c18	97	80	120	**120***	15.28	**120***	**120***	24.14	**120***	0.34
c19	196	160	240	**240***	2055.00	240.8	**240***	128.26	**240***	42.19
c20	197	160	240	**240***	1.56	**240***	**240***	42.1	**240***	0.55
c21	196	160	240	**240***	6381.91	240.8	**240***	134.48	**240***	0.29
Average					408.84			17.47		2.42

度, Time (s) 表示计算时间 (以秒为单位). 随机型近似算法 BSHA 对每一个问题实例计算 10 次, 本文列出其计算的统计结果: Mean H 表示 10 次计算所得到的矩形框长度的算术平均值, Best H 表示 10 次计算所得到的矩形框长度的最小值, Mean time (s) 表示 10 次计算时间的算术平均值 (以秒为单位). 如果算法刷新了当前记录, 则数据用黑体表示. 如果算法算出了最优解, 则数据用黑体表示并且加星号后缀 (*). 本文讨论的算法 A_0, A_1, BSHA, B_0, B_1 和 B_1C_1 均为高效近似算法, 不能保证找到最优布局, 但速度快且对于多数问题实例能找到最优布局或者接近最优布局.

3.1 矩形块方向可旋转 90°

(1) 第 1 组 21 个 benchmark 问题实例: 为 c1∼c21[13], 矩形块方向可旋转 90°.

B_1C_1 的计算结果与文献 [8] 中的非随机型近似算法 A_1 和文献 [12] 中的随机型算法 BSHA 作比较, 见表 1. B_1C_1 算法表现好于 A_1 和 BSHA.

首先, B_1C_1 与 A_1 比较. A_1 算法和 B_1C_1 算法均为非随机型近似算法. 非随机型近似算法的特点是对同一个实例每次计算所得布局相同. A_1 算法和 B_1C_1 算法计算本组实例的优度相同, 均找到 21 个实例的最优布局.

表 2　B_1C_1 在 zdf1∼zdf16 上的计算结果 (矩形块可旋转 90°)

Table 2　The computational results of B_1C_1 on zdf1∼zdf16 (the rectangles can rotate 90°)

Instance				B_1C_1	
Name	n	W	H^*	H	Time (s)
zdf1	580	100	330	**330***	106.80
zdf2	660	100	357	**357***	16.36
zdf3	740	100	384	**384***	0.29
zdf4	820	100	407	**407***	57.77
zdf5	900	100	434	**434***	50.10
zdf6	1532	3000	4872	4973	45158.47
zdf7	2432	3000	4852	5053	60204.60
zdf8	2532	3000	5172	**5172***	344.23
zdf9	5032	3000	5172	**5172***	579.78
zdf10	5064	6000	5172	**5172***	263.31
zdf11	7564	6000	5172	**5172***	358.58
zdf12	10064	6000	5172	**5172***	192.26
zdf13	15096	9000	5172	**5172***	4010.35
zdf14	25032	3000	5172	**5172***	7269.54
zdf15	50032	3000	5172	**5172***	9788.21
zdf16	75032	3000	5172	**5172***	15147.79

B_1C_1 算法计算本组实例的时间少于 A_1 算法. A_1 算法在 CPU 为 1.73 GHz 的微机上运行, B_1C_1 算法在 1.8 GHz 微机上运行. 1.8 GHz 微机的速度大约是 1.73 GHz 微机的 1.04 倍. 但表 1 中 B_1C_1 算法计算 21 个实例的平均时间仅为 2.42 s. A_1 算法计算的平均时间为 408.84 s.

然后 B_1C_1 与 BSHA 比较. B_1C_1 算法计算本组实例的优度略高于 BSHA 算法. BSHA 算法对 c19 和 c21 的各 10 次计算中, 虽然能找到最优布局, 但是由于 BSHA 算法的随机性, 不能保证每次计算均找到最优布局.

B_1C_1 算法计算本组实例的时间少于 BSHA 算法. BSHA 算法在 2.6 GHz 的微机上运行, 其速度大约为 1.8 GHz 微机的 1.44 倍. 表 1 中 B_1C_1 算法计算 21 个实例的平均时间仅为 2.42 s. BSHA 为 17.47 s.

(2) 第 2 组 16 个 benchmark 问题实例: 为 zdf1∼zdf16 [12], 矩形块方向可旋转 90°.

目前已知文献中尚未对 zdf1∼zdf16(矩形块方向可旋转 90°) 的计算结果作报道. 本文列举出 B_1C_1 算法的计算数据, 供读者参考.

B_1C_1 算法的具体计算数据见表 2. 对于 zdf1∼zdf5 和 zdf8∼zdf16 共 14 个实例, B_1C_1 算法算出了最优布局, 且计算时间在可接受范围内.

3.2　矩形块方向固定

在有些应用领域, 矩形块的方向固定. 例如木板切割, 由于木材有纹理, 切割方向不能改变. 我们不妨规定: 矩形框的边长等于其宽度 W 的一对边与 x 轴平行, 边长等于其长度 H 的一对边与 y 轴平行 (参见图 1) . 对小矩形块也这样规定.

表 3 BSHA 和 B_1C_1 在 c1∼c21 上的计算结果比较 (矩形块方向固定)

Table 3 Comparation of the computational results of BSHA and B_1C_1 on c1∼c21 (the orientation of the rectangles is fixed)

Instance				BSHA (10 tests)			B_1C_1	
Name	n	W	H^*	Mean H	Best H	Mean time (s)	H	Time (s)
c1	16	20	20	**20***	**20***	0.00	**20***	0.00
c2	17	20	20	**20***	**20***	0.31	21	0.66
c3	16	20	20	**20***	**20***	0.02	**20***	0.00
c4	25	40	15	**15***	**15***	0.08	**15***	0.00
c5	25	40	15	**15***	**15***	0.09	**15***	0.08
c6	25	40	15	**15***	**15***	0.01	**15***	0.00
c7	28	60	30	**30***	**30***	0.18	**30***	0.18
c8	29	60	30	31	31	15.54	**30***	6.65
c9	28	60	30	**30***	**30***	0.1	**30***	0.00
c10	49	20	60	61	61	40.44	**60***	57.27
c11	49	20	60	61	61	48.35	61	59.17
c12	49	20	60	60.8	**60***	37.71	**60***	43.92
c13	73	60	90	91	91	88.45	91	175.44
c14	73	60	90	91	91	75.13	**90***	19.47
c15	73	60	90	90.8	**90***	88.53	**90***	184.67
c16	97	80	120	121	121	221.67	121	479.96
c17	97	80	120	121.2	121	181.56	**120***	242.15
c18	97	80	120	121	121	170.78	121	437.54
c19	196	160	240	241	241	434.16	241	2464.77
c20	197	160	240	241.8	241	254.62	**240***	21.67
c21	196	160	240	241	241	468.82	241	2459.09
Average						101.26		316.80

第 2 节所介绍的算法略作修改, 可用于求解此类问题. 具体修改之处为我们将定义 10 中的指标 “矩形块的长边长, 大优先” 改为 “矩形块宽度, 大优先”, 指标 “矩形块的短边长, 大优先” 改为 “矩形块长度, 大优先”. 对定义 11 中的指标作相同修改, 并删去定义 11 中的指标 “矩形块的方向, 躺优先”.

对于矩形块方向固定的情形, 定理 1 和 2 依然成立. 证明过程类似.

文献 [8] 中未报道 A_1 算法对矩形块方向固定的实例的计算结果. 因此 B_1C_1 算法仅与 BSHA 算法比较.

(1) 第 1 组 21 个 benchmark 问题实例: 为 c1∼c21[13], 矩形块方向固定.

B_1C_1 算法计算本组实例的优度比 BSHA 算法高, 计算结果见表 3. 在 21 个实例中, B_1C_1 算法找到 14 个实例的最优解. 对其余 7 个实例, B_1C_1 算法所找到的框长度比最优解多 1 个长度单位. BSHA 算法对每个实例作 10 次计算, 只能找到 10 个实例的最优解. 而且由于 BSHA 算法的随机性, 不能保证每次计算对 c12 和 c15 均找到最优布局.

B_1C_1 算法计算本组实例的时间与 BSHA 算法相当. B_1C_1 算法计算 21 个实例的平均时间为 316.80 s. BSHA 为 101.26 s. BSHA 在 2.6 GHz 的微机上运行, 其速度大约为 1.8 GHz 微机的 1.44 倍.

表 4 BSHA 和 B_1C_1 在 zdf1∼zdf9 上的计算结果比较 (矩形块方向固定)

Table 4 Comparation of the computational results of BSHA and B_1C_1 on zdf1∼zdf9 (the orientation of the rectangles is fixed)

Instance				BSHA (10 tests)			B_1C_1	
Name	n	W	H^*	Mean H	Best H	Mean time (s)	H	Time (s)
zdf1	580	100	330	**330***	**330***	7.65	**330***	80.68
zdf2	660	100	357	**357***	**357***	9.72	**357***	46.29
zdf3	740	100	384	**384***	**384***	6.86	**384***	55.81
zdf4	820	100	407	**407***	**407***	9.04	**407***	206.24
zdf5	900	100	434	**434***	**434***	5.73	**434***	52.63
zdf6	1532	3000	4872	5175	5147	1302.98	**4973**	41068.64
zdf7	2432	3000	4852	5181.8	5172	3152.35	**5003**	52129.05
zdf8	2532	3000	5172	5522.8	5467	3000.56	**5172***	6855.35
zdf9	5032	3000	5172	5566	5566	4609.27	**5172***	305.90
Average						1344.91		11200.07

表 5 B_1C_1 算法在 zdf10∼zdf16 上的计算结果 (矩形块方向固定)

Table 5 The computational results of B_1C_1 on zdf10∼zdf16 (the orientation of the rectangles is fixed)

Instance				B_1C_1	
Name	n	W	H^*	H	Time (s)
zdf10	5064	6000	5172	**5172***	574.90
zdf11	7564	6000	5172	**5172***	942.61
zdf12	10064	6000	5172	**5172***	847.76
zdf13	15096	9000	5172	**5172***	451.84
zdf14	25032	3000	5172	**5172***	2741.80
zdf15	50032	3000	5172	**5172***	5130.89
zdf16	75032	3000	5172	**5172***	35581.02

为方便比较, 将 BSHA 的计算时间 101.26 s 乘以 1.44 倍, 为 145.81 s. 而且 BSHA 是随机型算法, 执行 10 次. B_1C_1 算法是非随机型近似算法, 只执行 1 次.

(2) 第 2 组 16 个 benchmark 问题实例: 为 zdf1∼zdf16 [12], 矩形块方向固定.

已知文献中仅有对 zdf1∼zdf9 的计算结果, 其中 zdf6∼zdf9 这 4 个问题实例的最优解在已知文献中尚未报道. B_1C_1 算法和 BSHA 算法对 zdf1∼zdf9 的具体计算数据见表 4, 对于 zdf1∼zdf5 和 zdf8∼zdf9 共 7 个实例, B_1C_1 算法算出了最优布局. 对于 zdf6∼zdf9, B_1C_1 算法给出了比已知文献报道记录更优的布局. 其中对 zdf8 和 zdf9, B_1C_1 算法历史上首次算出了最优布局.

B_1C_1 算法计算 zdf1∼zdf9 的优度明显高于 BSHA 算法, 且其计算时间在可接受的范围内. 对 zdf1∼zdf5, B_1C_1 算法与 BSHA 算法均找到最优布局. 面积利用率等于矩形框内矩形块的面积和除以矩形框面积. 对 zdf6∼zdf9, B_1C_1 算法所找到的布局平均面积利用率为 98.74%. BSHA 算法对每个实例计算 10 次, 取其最好布局, 对 zdf6∼zdf94 个实例的布局平均面积利用率为 94.00%. B_1C_1 算法对 zdf6∼zdf9 所找到的布局平均面积利用率比 BSHA 高 4.74 个百分点.

B_1C_1 算法计算本组实例的时间比 BSHA 略长, 但在可接受范围内. 对 zdf1∼zdf9, B_1C_1 算法的平

均计算时间为 11200.07 s, BSHA 算法为 1344.91 s. 但是表 4 中报道的是 BSHA 计算 10 次的平均时间. 并且其运行电脑 (2.6 GHz 微机) 速度大约为 B_1C_1 算法所运行电脑 (1.8 GHz 微机) 的 1.44 倍. B_1C_1 算法只计算 1 次.

zdf10∼zdf16 这 7 个问题实例的计算结果在已知文献中尚未报道. B_1C_1 算法对 zdf10∼zdf16 均找到最优布局, 且计算时间可接受, 参见表 5.

4 结论

本文从人们装箱、下围棋、砌砖的实际生活经验得到启发, 提出了矩形块占角并尽可能占穴的基本算法. 在此基础上提出了优美度枚举算法. 算法计算了两组 Strip Packing 问题实例 c1∼c21 和 zdf1∼zdf16, 算法的表现好于目前文献中所发表的优秀算法. 针对矩形块方向固定的情形, 算法对 zdf6∼zdf9 得到了比此前国际上最优布局更优的布局方案, 其中对 zdf8 和 zdf9 首次找到最优布局. 在今后工作中, 拟进一步提高算法效率.

致谢 谨以此文纪念黄文奇教授.

参考文献

1 Alvarez-Valdes R, Parreno F, Tamarit J M. A branch and bound algorithm for the strip packing problem. OR Spectrum, 2009, 31: 431–459

2 Jiang X B, Lü X Q, Liu C C. Lowest-level left align best-fit algorithm for the 2D rectangular strip packing problem. J Softw, 2009, 20: 1528–1538 [蒋兴波, 吕肖庆, 刘成城. 二维矩形条带装箱问题的底部左齐择优匹配算法. 软件学报, 2009, 20: 1528–1538]

3 Cui Y D, Yang Y L, Cheng X, et al. A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. Comput Oper Res, 2008, 35: 1281–1291

4 Huang W Q, Chen D B, Xu R C. A new heuristic algorithm for rectangle packing. Comput Oper Res, 2007, 34: 3270–3280

5 Huang W Q, Chen D B. An efficient heuristic algorithm for rectangle-packing problem. Simul Model Pract Theory, 2007, 15: 1356–1365

6 He K, Huang W Q. An efficient placement heuristic for three-dimensional rectanglar packing. Comput Oper Res, 2011, 38: 227–233

7 Huang W Q, He K. A pure quasi-human algorithm for solving the cuboid packing problem. Sci China Ser F-Inf Sci, 2009, 52: 52–58

8 He K, Huang W Q, Jin Y. Efficient algorithm based on action space for solving the 2D rectangular packing problem. J Softw, 2012, 23: 1037–1044 [何琨, 黄文奇, 金燕. 基于动作空间求解二维矩形 Packing 问题的高效算法. 软件学报, 2012, 23: 1037–1044]

9 Bortfeldt A. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. Eur J Oper Res, 2006, 172: 814–837

10 Leung T W, Chan C K, Troutt M D. Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. Eur J Oper Res, 2003, 145: 530–542

11 Jiang J Q, Liang Y C, Shi X H, et al. A hybrid algorithm based on PSO and SA and its application for two-dimensional non-guillotine cutting stock problem. Lect Notes Comput Sci, 2004, 3007: 666–669

12 Zhang D F, Wei L J, Leung S C H, et al. A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem. INFORMS J Comput, 2013, 25: 332–345

13 Hopper E, Turton B C H. An empirical investigation of meta-heuristic and heuristic algorithm for a 2D packing problem. Eur J Oper Res, 2001, 128: 34–57

A beauty degree enumeration algorithm for the 2D rectangular packing problem

WANG Lei[1,2*] & YIN AiHua[3]

1 *College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan* 430081*, China*;
2 *Hubei Province Key Laboratory of Intelligent Information Processing and Real-Time Industrial System, Wuhan* 430081*, China*;
3 *School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang* 330013*, China*
*E-mail: wanglei77@wust.edu.cn

Abstract To address the 2D rectangular packing problem, this paper presents a basic algorithm based on corner areas, and presents a beauty degree enumeration algorithm. For two sets of representative benchmark instances c1–c21 and zdf1–zdf16, the algorithm outperforms the best algorithms in the literature. When the orientation of the rectangles is fixed, the algorithm finds better packing configurations than the best reported results of four open benchmark instances zdf6–zdf9. The proposed algorithm finds the optimal solutions for zdf8 and zdf9; to the best of our knowledge, this is the first time this has been achieved.

Keywords packing, NP hard, combinatorial optimization, heuristic, quasi-human

WANG Lei was born in 1977. He received his Ph.D. degree in Computer Science from the Huazhong University of Science and Technology, Wuhan, in 2006. Currently, he is a lecturer at Wuhan University of Science and Technology. His research interests include job shop scheduling, packing, the traveling salesman problem, and the satisfiability problem.

YIN AiHua was born in 1970. He received his Ph.D. degree in Computer Science from the Huazhong University of Science and Technology, Wuhan, in 2003. Currently, he is a Senior Researcher at Jiangxi University of Finance and Economics. His research interests include the job shop scheduling problem (multiple or single machine), and the packing problem. Dr. Yin is a member of CCF.