OPTIMIZING THREE-DIMENSIONAL BIN PACKING THROUGH SIMULATION

通過仿真優化三維裝箱(2006)

Abstract

本文解決的問題是將給定的一組矩形物品直接打包到最小數量的三維矩形箱中。我們利用現代計算機的計算能力來解決這個 NP-Hard 問題，否則這個問題實際上會非常困難。我們開發的軟件工具使用啟發式和一些背包問題的方法，將解決方案作為解決空間的三維圖形表示。仿真模型的可視化和交互性提供了一個有紀律的途徑

Keyword

仿真，優化，裝箱，最佳擬合，首次擬合，旋轉

1. Introduction

本文描述了一種模擬方法來描述最優解的可行封裝和構造。尋找三維裝箱問題的最優解的複雜性，由於難以給出一個有用的問題公式而變得更加複雜。為了表述這個問題，我們將考慮有限集 S 中的每一項都有三個維度 wi, hi 和 di。每個相同的箱子 b 有尺寸 W, H 和 d。物品和箱子是矩形的盒子，三個尺寸對應於寬度，高度和深度值。為了使解與以前的工作更不同，這些項被允許進行正交旋轉。旋轉一個項目僅僅意味著按定義的有序方式交換它的寬度(wi)，高度(hi)和深度(di)值(表 1)。每個物品盒有 6 個矩形面，但只有 3 個不同的面，因為“相反的”面是相同的。這三個面中的每一個都可以進行正交旋轉以獲得盒子的新配置。因此，每個項目可以有 6 種不同的旋轉配置。

線性規劃方法已用於一維問題。在一些方法中，進化算法也被用來代替啟發式算法[1,3]。由於進化算法具有搜索大空間的能力，因此它們可能是尋找解的合適方法(在這種情況下，是裝箱問題的解空間)，但進化算法有一些缺點:解和問題之間幾乎沒有連續性。如果你稍微改變一下問題的參數，那麼解決方案就會發生相當大的變化。

2. Problem formulation

為了找到箱子 b 的解，我們在不損失一般性的前提下假設 $\sum_{i\in b} w_i \le W$, $\sum_{i\in b} h_i \le H$ $\sum_{i\in b} d_i \le D$ ，

因此 $\sum_{i\in b} w_i \bullet h_i \bullet d_i \le W \bullet H \bullet D$ ，

對於每個箱子 b，我們希望最小化冗餘空間 : $W \bullet H \bullet D - \sum_{i\in b} w_i \bullet h_i \bullet d_i$ 。

裝箱是一個 NP-Hard 問題，表明對最優解的詳盡搜索通常在計算上是難以處理的，因此對於該問題沒有已知的實際計算可行的最優解方法。因此必須找到其他方法來獲得解決方案。最受歡迎的是啟發式求解方法：

一次包裝一件物品，沒有回溯 (一旦物品被包裝，就不會重新包裝)。可以通過使用源自以下打包算法之一的形式邏輯來選擇要打包的項目。

First Fit

將未分配的項目打包到第一個有足夠空間的箱中。如果沒有這樣的 bin，則將 item 分配到新的 bin 中。

First Fit Decreasing

幾乎與 First Fit 相同，不同之處在於物品在打包之前首先按降序排序

Last Fit

將未分配的項目裝入具有足夠空間的最後一個垃圾箱。搜索與 First Fit 類似，但與 bin 的順序相反。如果沒有 such bin，則將 item 分配到 new bin。

Best Fit

最佳擬合算法將一個項目打包到一個 bin 中，這是該項目適合的那些 bin 中最滿的。更具體地說：項目一次打包一個 Item。要確定一個項目的 bin，首先確定該項目適合的容器集 B。如果 B 為空，則開始一個新的 bin 並將項目放入這個 new bin。否則，將物品裝入可用容量最小的 B 的箱中。

3. Solution Specification

開發的系統使用啟發式方法來執行裝箱的核心。在系統中使用這些啟發式近似算法來解決裝箱問題：

i. 保證問題的解決

ii．在合理的時間內獲得解決方案（即解決方案在計算上是可行的）

iii. 允許輸入通用數據

iv. 提供解決方案和問題之間的連續性

以上幾點為為什麼選擇啟發式方法提供了堅實的理由，因為任何不滿足上述任何條件的方法都不會完全滿足用戶要求，因此不會有任何用處

顯然，不滿足（i）和（ii）將是不令人滿意的。如果系統不滿足（iii），那麼它將失去其通用性和靈活性。條件 (iii) 也很重要，因為箱子（或集裝箱）需要裝滿不同尺寸的物品（或貨物）。不滿足（iv）也會使用戶在解決問題的過程中難以檢索數據和測試替代解決方案。因此，不滿足 (iv) 將意味著系統不支持此功能。

系統中使用的兩種主要啟發式裝箱算法是**首次擬合遞減**和**最佳擬合**。之所以選擇它們而不是其他啟發式算法，是因為它們具有更快的運行時間，並且產生的解決方案比大多數其他啟發式算法更接近最佳解決方案。

3.1 Analysis of the algorithms

令 M 為包裝一組 n 項所需的最佳箱數

令 Mf 成為當使用首次擬合遞減來包裝物品時所需的箱數

令 Mb 成為使用 Best Fit 包裝物品時所需的箱數

因此，與 First Fit Decreasing 相比，Best Fit 似乎更有可能產生更接近最佳解決方案的解決方案，但鑑於所需 bin 數量上限的這些值彼此相對接近，因此值得使用兩種算法。

Then it can be shown [2] that $m_f \le M + \left\lceil \frac{(M-1)}{3} \right\rceil$

and also that $m_b \le \left\lceil \frac{17M}{10} \right\rceil$.

Now:

$M + \frac{(M-1)}{3} \approx \frac{4M}{3}$ for M sufficiently large,

and thus $\frac{17M}{10} = 1\frac{7}{10}M > 1\frac{1}{3}M = \frac{4M}{3}$ for large M.

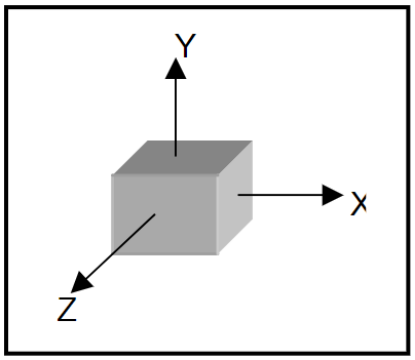
$$\left\lceil \left(\sum_{i \in S} s_i \right) / C \right\rceil \le M \le m_f \le M + \left\lfloor \frac{(M-1)}{3} \right\rfloor$$

我們還觀察到: $\left\lceil \left(\sum_{i \in S} s_i \right) / C \right\rceil \le M \le m_b \le \left\lceil \frac{17M}{10} \right\rceil$ 此處 C 是單個單位、面積或體積的箱子容量，具體取決於所考慮的當前箱子包裝情況。 Best Fit 的運行時間為 O (n log n)，而 First Fit Decreasing 的運行時間為 O (n log n)，不包括排序的運行時間。

4. Software Specification

我們有三個方向來包裝物品，寬度方向，高度方向和深度方向。如前所述，每個 Item 都有六種旋轉類型。考慮一個項目：通過繞 x 、 y 和/或 z 軸旋轉可以獲得六種旋轉類型，如表 1 所示：一次打包一個 bin，算法使用一系列樞軸點進行打包項目。

Rotation Type	First axis to rotate about	Second axis to rotate about
0	-	-
1	Z	-
2	Y	-
3	X	Y
4	X	-
5	X	Z



4.1 3D Best Fit Algorithm

確定包裝方向。每個 bin 具有三個方向來打包，一個寬度（或 x）方向，一個高度（或 y）方向，一個深度（或 z）方向。一次打包一個。

我們首先選擇一個樞軸點，樞軸是一個 (x, y, z) 坐標，它表示將嘗試打包項目的特定 3D 箱中的一個點。項目的後左下角將放置在樞軸處。如果項目不能在樞軸位置被打包，那麼它會被旋轉直到它可以在樞軸點被打包或直到我們嘗試了所有 6 種可能的旋轉類型。如果在旋轉它之後，該項目仍然無法在樞軸點打包，那麼我們繼續打包另一個項目，並將未打包的項目添加到將在嘗試打包剩餘項目後打包的項目列表。空箱中的第一個樞軸始終是 (0,0,0)。

4.1.1 The 3D Best Fit, with pivoting, algorithm is as follows

偽代碼(略)

4.1.2 Worst Case Running Time

在最壞的情況下，我們看到上面引用的 do-while 循環最多運行 (n) 次。循環將最多運行 (n-2) 次。cwill 運行 3 次。 dwill 最多運行 (n-1) 次，因為 bin 中的項目數可能是 (n-1)。整個算法執行的旋轉最多可以是 6 次，因此這不會顯著影響我們的運行時間。

所以我們有： **O (Best Fit)= n* (n-2) * 3 * (n-1) = O(n***) 因此，在最壞的情況下，算法會在多項式時間內產生一個解。**

4.1.3 Best Case Running Time

排除瑣碎情況的最佳情況是所有項目都適合一個 Bin ： a 將運行 1 次。 b 將運行 (n-2) 次。 c 將運行 3 次。 d 將運行 (n-1) 次。所以我們有： O (Best Fit) =1* (n-2) *3 * (n-1) = O(n**)

或者當每件物品被裝入自己的垃圾箱時，將運行 n 次。 b 將運行 (n-2) 次。 c 將運行 3 次。 d 將運

行 1 次。所以我們有： **O (Best Fit)=n* (n-2) *3 *1 = O(n**)，因此在最好的情況下，性能是 O(n**)**。

4.2 3D First Fit Decreasing Algorithm

要包裝一件物品，首先必須確定包裝方向。**箱的最長邊對應於包裝方向。然後旋轉每個項目，使該項目的最長邊是包裝方向的邊**，即如果我們按寬度包裝，那麼我們希望項目的最長邊是項目的寬度，例如，如果包裝方向按寬度和項目的當前高度大於其寬度，然後旋轉項目。如果在執行旋轉後，物品不能放入箱子中（即物品的一個或多個維度超過了箱子的對應維度），那麼我們旋轉該物品，直到該物品的第二長邊是對應的邊到包裝方向。如果在執行旋轉後，物品無法放入垃圾箱，那麼我們旋轉物品直到物品的第三長邊是與包裝方向對應的邊。接下來根據包裝方向按寬度、高度或深度的遞減順序對物品進行排序。

5. Simulation Model and Graphical display

略

6. Conclusion

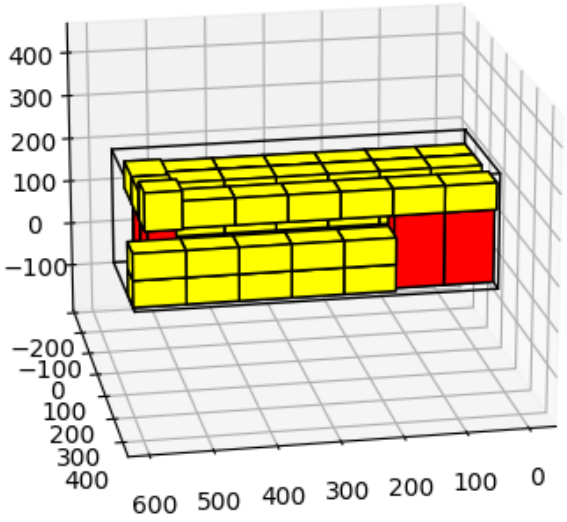
略

7. 理解 (By Jerry) :

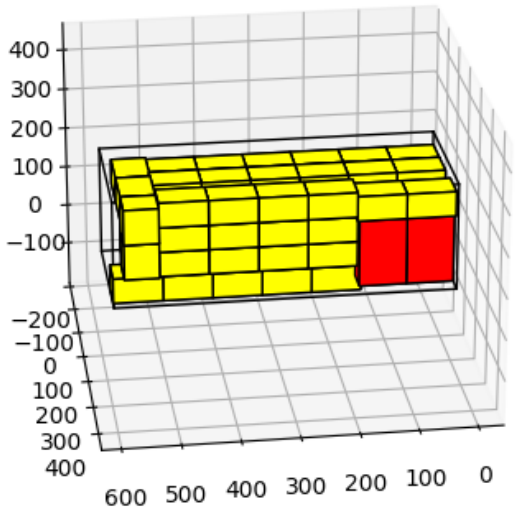
此篇論文應歸類於**在線式打包(Inline packing)**方法，並且依據一定規則(啟發式)打包，並**非最優化算法**，優點是可以在多項式時間內計算完成(最優化此問題屬於 NP hard)，缺點是無法確定此解屬於最佳解，但因其計算效率以及通用性較高，能應用在商業系統中。

8. 改進 (By Jerry) :

8.1 此打包方法並沒有考慮重力問題，所以時常發現打包完成之後有 item 浮在空中，大大降低裝箱的空間利用率，若能改進考慮重力問題，能提升裝箱率。(**已完成**)



原論文算法(可發現中間有一排空洞)



改進算法(不僅填補空洞，也增加空間利用)

8.2 此打包空間並沒有考慮**平衡性**(下方支點)，也就是說，若下方是較小物體，上方一樣可以擺放大物體，在實際應用場景中會發生傾倒的狀況，若能改進此問題，能提升實用性。

8.3 **最優化問題**，或許可以使用**貪婪算法**或是**暴力解法**解決優化問題(但無法達到最優化，因為這是 NP hard 問題)

8.4 真實應用需求的擴展性，此方法沒有考慮如下:

a. 物品承重(**部分完成**)

solution : 使用分級制(**無法判斷準確的承受重量，但承重力高的物品會優先放在下方**)

b. 必須一定得裝箱的物品(need to pack in)(**已完成**)

solution : 使用分級制 : 0~無限級(0 代表一定要安裝，依次類推)

c. 物品是否可以倒放(**已完成**)

solution : bool: 可選擇物品是否可以倒放(True/False)

d. **成套搭配 :** 1+1 搭配或是 1+2 搭配，指兩種或三種以上物品必須在 bin 中是同等數量(**已完成**)

8.5 輸出的圖是否可以轉換成 3D 物件格式，以及確認前端可支援的 3D 物件格式 (**已改成傳 position**)

STL(有難度，使用三角形繪圖) 、OBJ 似乎不支援顏色儲存，可能要使用 AMF、3MF

Example : three.js

8.6 輸出的結果必須包含放置順序，以利打包工人或是機器按照放置順序安裝(**頂開式還是側開式**)

8.7 集裝箱角件問題(**8 角 or 4 角 or 2 角 ?**)(**已完成目前預設 8 角**)

solution: 目前使用 8 角

8.8 堆疊數量問題(同品項商品堆疊數量限制)

8.9 托盤/棧板空間

8.10 緩衝材/包材空間計算以及預留

8.11 集裝箱重心

9.市面上的商業軟體

裝箱大師 : http://www.zhuangxiang.com/

CargoWiz : https://softtruck.com/

Searates : https://www.searates.com/cn/reference/stuffing/ (可直接網頁操作)