Применение алгоритма двумерной упаковки в промышленном производстве прямоугольных панелей

В. В. Осокин, Р. Ф. Алимов, Т. Р. Сытдыков

В данной статье рассматривается применение алгоритма двумерной упаковки при вырезании прямоугольных панелей произвольных размеров из стандартных прямоугольных панелей фиксированных размеров (сэндвич-панелей). Требуется определить минимальное количество сэндвич-панелей, необходимое для производства всех требуемых панелей. Будет рассмотрено решение этой задачи при помощи HBF-алгоритма с реализацией на языке программирования PHP.

Ключевые слова: PHP, двумерная упаковка, HBF-алгоритм.

Введение

Прямоугольные панели находят применение в самых различных сферах жизнедеятельности — строительстве, промышленной вентиляции [1], производстве упаковок, контейнеров и др. В общем случае панели могут состоять из различных материалов, иметь различную толщину и другие характеристики. Тем не менее, несмотря на все различия, в большинстве случаев прямоугольные панели изготовляются одним и тем же способом — вырезанием из стандартных прямоугольных панелей (так называемых сэндвич-панелей), обладающих фиксированной длиной

и шириной. Это объясняется тем, что при конвейерном изготовлении панелей из исходных материалов гораздо экономичнее производить стандартные панели фиксированных размеров, чем настраивать работу конвейера для каждого требуемого размера. Отсюда естественным образом возникает задача минимизации количества сэндвич-панелей, требуемых для изготовления всего заданного набора прямоугольных панелей.

Данная задача есть не что иное, как модификация классической задачи теории оптимизации, известной как «двумерная упаковка» («two-dimensional packing»), которая рассмотрена, например, в [2] и [3]. Известно, что данная задача является NP-трудной и эффективных точных алгоритмов ее решения на сегодняшний день не существует. Однако имеется ряд эвристических полиномиальных алгоритмов, позволяющих получить решения, достаточно близкие к оптимальному. Далее будет рассмотрено применение одного их наиболее известных алгоритмов — Hybrid Best-Fit (HBF).

Постановка задачи

Пусть имеется неограниченное количество сэндвич-панелей размера W x H. Требуется вырезать n прямоугольных панелей размера w_i x h_i, $i = 1, 2, \ldots, n$, затратив при этом минимальное количество сэндвич-панелей.

Нетрудно видеть, что в отдельных случаях необходимо проводить разрезы, непараллельные сторонам сэндвич-панелей (например, из сэндвич-панели 10x10 можно вырезать панель 1x11, только прибегая к такому приему). В дальнейшем мы не будем рассматривать такие случаи, поскольку, во-первых, рассмотрение этих вариантов существенно усложнит алгоритм, а во-вторых, при промышленном изготовлении панелей в целях упрощения производственного процесса нередко ограничиваются только разрезами, параллельными сторонам сэндвич-панели.

В общем случае при фиксированной ориентации сэндвич-панелей каждая панель может быть вырезана двумя способами, которые получаются друг из друга при повороте на 90^o. Соответ-

ственно, n панелей могут быть вырезаны 2^n способами. Очевидно, что непосредственное рассмотрение всех способов чрезмерно усложнит алгоритм решения задачи. Поэтому далее мы будем рассматривать вариант задачи с фиксированной ориентацией сторон вырезаемых прямоугольных панелей ($\forall\ i = 1, 2, \ldots, n$ сторона длиной w_i должна быть параллельна W, сторона длиной h_i — параллельна H). Будем считать, что $\forall\ i = 1, 2, \ldots, n$ $w_i \leq W$, $h_i \leq H$.

Модификация алгоритма для учета ориентации вырезаемых прямоугольников может быть выполнена с помощью имитации отжига, генетических алгоритмов и других эвристических подходов, ее рассмотрение выходит за рамки данной статьи.

HBF-алгоритм, описание и реализация на языке PHP

Будем использовать классический Hybrid Best-Fit (HBF) алгоритм. Вначале построим некоторое количество «слоев» из вырезаемых панелей, используя алгоритм Best-Fit Decreasing Height (BFDH). Затем распределим полученные «слои» по сэндвич-панелям, получив тем самым нужную схему разрезов.

Алгоритм BFDH работает следующим образом. Во время работы поддерживается множество частично заполненных слоев с прямоугольниками, которые будут заполнять эти слои слева направо. Каждый слой имеет длину W. Прямоугольники рассматриваются в порядке убывания высоты и добавляются в выбранный слой. Слой выбирается следующим образом: среди всех слоев выбираем такой, который сможет вместить текущий прямоугольник и имеет минимальную оставшуюся ширину свободного пространства. Если подходящего слоя нет, создается новый слой, и прямоугольник помещается в его левый нижний угол.

Алгоритм HBF работает аналогично алгоритму BFDH, только в другом направлении. Каждый из построенных алгоритмом BFDH слоев размещается в прямоугольнике W x H (это можно сделать, так как ширина каждого слоя не превосходит W). Слои размещаются один за другим способом, аналогичным алгорит-

му BFDH. Заметим, что сортировку слоев выполнять не нужно, так как по построению алгоритмом BFDH они упорядочены по убыванию высоты самого левого прямоугольника.

Общее время работы алгоритма, очевидно, полиномиально относительно числа панелей n. С использованием структуры OrderedSet, рассмотренной далее, для быстрого выполнения поиска и вставки, асимптотика алгоритма составит $O(n \log n)$.

Приступим к программной реализации описанного алгоритма на языке PHP. Создадим класс прямоугольника Packer2DRectangle. Помимо длины и ширины, добавим в него поля для сохранения информации о результирующей схеме разрезов — номер сэндвич-панели, координаты левого нижнего угла, номер слоя.

```
class Packer2DRectangle {
    public $Index, $W, $H, $X, $Y, $ContainerIndex,
        $LayerIndex;
    function __construct($_index, $_w, $_h){
        $this->Index = $_index;

        $this->W = $_w;
        $this->H = $_h;
    }
}
```

Реализуем вспомогательную функцию сравнения прямоугольников для сортировки нужным для алгоритма способом (по убыванию высоты).

```
class Packer2DRectangle {
    public static function compare($p1, $p2){
        if ($p1->H == $p2->H){
            return 0;
        }

        return ($p1->H < $p2->H) ? 1 : -1;
    }
}
```

Кроме того, создадим вспомогательный класс слоя прямоугольников Packer2DRectangleLayer.

```
class Packer2DRectangleLayer {
    public $H, $Y, $Rectangles, $ContainerIndex;
}
```

Далее потребуется структура данных, позволяющая эффективно размещать прямоугольники по слоям. Для алгоритма Best-Fit Decreasing Height это означает, что если имеется множество частично заполненных слоев и нам нужно поместить текущий прямоугольник в некоторый слой, то мы выберем слой с минимально необходимым оставшимся свободным пространством. Таким образом, требуется поддерживать множество слоев, упорядоченных по длине свободного пространства, и уметь быстро вставлять элемент в это множество, удалять, а также искать минимальный элемент, не меньший данного.

Реализуем соответствующий класс.

```
class OrderedSet {
    private $_elems;
    private $_comparator;

    public function __construct($elems, $comparator) {
        usort($elems, $comparator);
        $this->_comparator  = $comparator;

        $this->_elems = array();
        foreach($elems as $elem) {
            $this->Insert($elem);
        }
    }
}
```

Функция вставки будет искать вставляемый элемент в множестве, и, если его нет, добавлять.

```
public function Insert($elem) {
    $ind = $this->LowerBoundIndex($elem);
    if (!array_key_exists($ind, $this->_elems) || $this->
        _elems[$ind] != $elem) {
        array_splice($this->_elems, $ind, 0, array($elem)
            );
    }
}
```

Функция LowerBoundIndex ищет индекс минимального элемента, не меньшего заданного.

```
public function LowerBoundIndex($elem) {
    $l = 0;
    $r = count($this->_elems);

    if (count($this->_elems) == 0 || call_user_func($this
        ->_comparator, $elem, $this->_elems[0]) <= 0) {
        return 0;
    }

    while($l + 1 < $r){
        $mid = floor(($l + $r) / 2);
        $compare_result = call_user_func($this->
            _comparator, $elem, $this->_elems[$mid]);
        if ($compare_result <= 0) {
            $r = $mid;

        } else {
            $l = $mid;
        }
    }
    return (int)$r;
}
```

Аналогично вставке реализуется удаление элемента.

```
public function Erase($elem) {
    $ind = $this->LowerBoundIndex($elem);
    if (array_key_exists($ind, $this->_elems) &&  $this->
        _elems[$ind] == $elem) {
        array_splice($this->_elems, $ind, 1);
    }
}
```

Наконец, сделаем аналог функции LowerBoundIndex, возвращающий соответствующий элемент множества.

```
public function LowerBound($elem) {
    $ind = $this->LowerBoundIndex($elem);
    if (!array_key_exists($ind, $this->_elems)) {
        return null;
    }
    return $this->_elems[$ind];
}
```

Теперь необходимо создать класс для элемента множества и снабдить его соответствующим компаратором (функцией сравнения экземпляров).

```
class Packer2DSetItem {
    public $Index, $Len;
    function __construct($_index, $_len){
        $this->Index = $_index;
        $this->Len = $_len;
    }

    public static function compare($p1, $p2){
        if ($p1->Len == $p2->Len){
            if ($p1->Index == $p2->Index){
                return 0;
            }

            return ($p1->Index < $p2->Index) ? -1 : 1;
        }
        return ($p1->Len < $p2->Len) ? -1 : 1;
    }
}
```

Осталось создать собственно класс, решающий поставленную задачу. Он будет содержать размеры сэндвич-панелей, массив размещаемых прямоугольников, полученные на промежуточном этапе слои и ответ на задачу — требуемое количество сэндвич-панелей.

```
class Packer2D {
    public $W, $H, $Rectangles;
    public $Layers, $ContainersCount;
}
```

Реализуем алгоритм BFDH. Будем использовать созданный класс OrderedSet.

```
private function BFDH() {
    $orderedSet = new OrderedSet(array(), "
        Packer2DSetItem::compare");
    usort($this->Rectangles, "Packer2DRectangle::compare
        ");

    $this->Layers = array();
    $LayersCount = 0;
    $RectangleCounter = 0;
```

```
        foreach($this->Rectangles as $Rectangle) {
            $tmpSetItem = new Packer2DSetItem(-1, $Rectangle
                ->W);
            $tmpElem = $orderedSet->LowerBound($tmpSetItem);


            if (isset($tmpElem)) {
                $Rectangle->LayerIndex = $tmpElem->Index;
                $orderedSet->Erase($tmpElem);
                $tmpElem->Len -= $Rectangle->W;


                if ($tmpSetItem->Len > 0) {
                    $orderedSet->Insert($tmpElem);
                }
                $this->Layers[$Rectangle->LayerIndex]->
                    Rectangles[] = $RectangleCounter;


            } else {
                $Rectangle->LayerIndex = $LayersCount;
                $tmpElem = new Packer2DSetItem($Rectangle->
                    LayerIndex, $this->W - $Rectangle->W);


                if ($tmpSetItem->Len > 0) {
                    $orderedSet->Insert($tmpElem);
                }
                $LayersCount++;


                $this->Layers[$Rectangle->LayerIndex]->H =
                    $Rectangle->H;
                $this->Layers[$Rectangle->LayerIndex]->
                    Rectangles = array($RectangleCounter);
            }
            $RectangleCounter++;
        }
    }
```

Далее нужно распределить слои по сэндвич-панелям. Воспользуемся все тем же классом OrderedSet.

```
    private function BFD() {
        $orderedSet = new OrderedSet(array(), "
            Packer2DSetItem::compare");
        $this->ContainersCount = 0;
```

```
        foreach($this->Layers as $Layer) {
            $tmpSetItem = new Packer2DSetItem(-1, $Layer->H);
            $tmpElem = $orderedSet->LowerBound($tmpSetItem);

            if (isset($tmpElem)) {
                $Layer->ContainerIndex = $tmpElem->Index;
                $Layer->Y = $this->H - $tmpElem->Len;
                $orderedSet->Erase($tmpElem);

                $tmpElem->Len -= $Layer->H;
                if ($tmpSetItem->Len > 0) {
                    $orderedSet->Insert($tmpElem);
                }

            } else {
                $Layer->ContainerIndex = $this->
                    ContainersCount;
                $Layer->Y = 0;
                $tmpElem = new Packer2DSetItem($Layer->
                    ContainerIndex, $this->H - $Layer->H);

                if ($tmpSetItem->Len > 0) {
                    $orderedSet->Insert($tmpElem);
                }
                $this->ContainersCount++;
            }

            $OffestX = 0;
            foreach($Layer->Rectangles as $RectangleIndex){
                $Rectangle = $this->Rectangles[
                    $RectangleIndex];
                $Rectangle->ContainerIndex = $Layer->
                    ContainerIndex;

                $Rectangle->Y = $Layer->Y;
                $Rectangle->X = $OffestX;
                $OffestX += $Rectangle->W;
            }
        }
    }
```

В результате полученное размещение прямоугольников по сэндвич-панелям будет храниться в полях ContainerIndex, X и

Y элементов массива Rectangles. Общее количество сэндвич-панелей будет записано в переменную ContainersCount.

Заключение

Таким образом, получено решение задачи о минимизации числа сэндвич-панелей с помощью алгоритма HBF. Согласно оценкам из [2], полученное количество сэндвич-панелей достаточно близко к оптимальному. Дальнейшее улучшение полученного результата может быть получено применением более сложных эвристических алгоритмов (с асимптотикой медленнее, чем $O(n \log n)$), использованием эффективных методик подбора оптимальной ориентации вырезаемых панелей и комбинированием всех результатов.

Список литературы

[1] Кудрявцев В. Б., Осокин В. В. ВЕНТИЛЯЦИЯ [программирование расчетов промышленного оборудования]. — М.: Интеллектуальные системы, 2016. — 235 с.

[2] Berkey J. O., Wong P. Y/. Two dimensional finite bin packing algorithms. Journal of Operational Research Society, 2: 423–429, 1987.

[3] Lodi A., Martello S., Vigo D.. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing, 11: 345–357, 1999.