Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Кафедра системного программирования

Направление подготовки / специальность: 03.03.01 Прикладные математика и физика
Направленность (профиль) подготовки: Математическая физика, компьютерные технологии и математическое моделирование в экономике

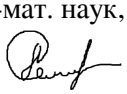
РАЗРАБОТКА И ИССЛЕДОВАНИЕ ЭФФЕКТИВНЫХ МЕТОДОВ УПАКОВКИ

(бакалаврская работа)

Студент:
Ахтямов Дамир Рамилевич

_____
(подпись студента)

Научный руководитель:
Семенов Виталий Адольфович,
д-р физ.-мат. наук, проф.



_____
(подпись научного руководителя)

Консультант *(при наличии)*:
_____
_____
(подпись консультанта)

Москва 2021

Аннотация.

В работе рассматривается задача двумерной упаковки прямоугольников в полубесконечную полосу. Проводится краткий обзор точных и приближенных методов, в частности, рассматриваются семейства популярных эвристических и метаэвристических алгоритмов. Программно реализованы два характерных представителя данных семейств, а именно: алгоритм Priority Best Fit и алгоритм имитации отжига. Проведенные вычислительные эксперименты показывают преимущества первого и необходимость более тонкой настройки параметров метаэвристического алгоритма.

ОГЛАВЛЕНИЕ

Введение.

В данной работе рассматривается задача ортогональной упаковки n прямоугольников в полубесконечную полосу шириной W с минимизацией используемой высоты полосы. Данная задача называется 2SPP – two-dimensional strip-packing problem. Ортогональная упаковка означает, что стороны прямоугольников и полосы параллельны или перпендикулярны. Согласно [1] задача классифицирована как two-dimensional open dimension problem (2-ODP).

Задача имеет множество применений в промышленности, например при раскрое материала необходимо в ленте вырезать куски определённых размер, используя наименьшую часть полосы, упаковке товаров. Также данная задача возникает приложениях к задачам проектного планирования.

Задача является NP-трудной. Данный факт был доказан сведением задачи к 1BP – одномерной упаковки контейнеры, решение этой задачи является нижней границей к исходной задаче. Сведение происходит разрезанием каждого прямоугольника на горизонтальные полосы высотой 1. Как показано в [3], задача 1BP является NP-полной в сильном смысле. Таким образом точные алгоритмы, как то полный перебор или метод ветвей и границ, неприменимы на практике, где используются задачи с количеством прямоугольников более 50.

Таким образом основным направлением в исследовании этой задачи является поиск полиномиальных приближённых алгоритмов, предоставляющих редукции задачи, нижние границы и приближённые решения. Алгоритмы, дающие приближённое решение, подразделяются на эвристические и метаэвристические. Их разница состоит в том, что эвристический алгоритм разрабатывается для решения конкретной задачи, на её основе; метаэвристика же – шаблон, по которому строится эвристический алгоритм для решения задачи. Метаэвристику можно применить к большому спектру задач, в то время как эвристика решает лишь одну задачу. Метаэвристики обладают некоторыми общими свойствами:

- В результате их работы последовательно строятся несколько решений;
- Построение каждого нового решения основывается на накопленных знаниях о качестве предыдущих полученных решений.

Перечислим основные эвристические подходы к решению данной задачи:

1) уровневые, такие как Next Fit (поставить на следующее место в уровне), First Fit (поставить на первый подходящий уровень), Best Fit (поставить в уровень с минимальным оставшимся пространством);

2) с горизонтом: Priority Best Fit (упаковка наиболее приоритетного прямоугольника в нишу);
3) остальные: Normal Pattern Shifting (упаковка в первый подходящий угол), блочный алгоритм (приближённое решение задачи 1CBP, затем построение решения 2SP на основе решения 1CBP), и т. д.

Перечислим основные метаэвристики:

1) Генетический алгоритм. Метаэвристика основана на естественном отборе в живой природе. Рассматривается популяция – множество решений. Каждая особь является некоторым решением. Каждую итерацию определяется лучшая особь (решение). Далее популяция изменяется посредством скрещивания, мутации и, возможно, введением в популяцию случайно сгенерированных особей. После пополнения популяции происходит отбор – удаление «неприспособленных» особей.
2) Муравьиный алгоритм. Метаэвристика основана на поведении муравьёв при поиске пищи, они оставляют феромоны, которые привлекают других муравьёв и помогают эффективно собирать пищу. Каждую итерацию запускается один или несколько муравьёв. Решением задачи является путь муравья. Муравей с большой вероятностью пойдёт по пути с большим значением феромона, но есть вероятность отклониться от маршрута, тем самым происходит поиск локального оптимума. Значение феромонов обновляется на основе путей с лучшими значениями целевой функции.
3) Алгоритм имитации отжига. Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Во время работы алгоритма имеется переменная температуры, отвечающая за вероятность перехода в точку со значением целевой функции похуже (если лучше – переход безусловный). Во время работы имеется лишь одно решение. Каждую итерацию вычисляется «близкое» к текущему решению решение. Далее выбирается следующая точка на основе температуры и значений целевой функций точек. Температура по ходу алгоритма убывает до нуля.

Целью данной работы является обзор существующих методов решения задачи 2SP, реализация и сравнение алгоритмов разных классов. В качестве эвристического алгоритма был выбран Priority Best Fit, подробно описанный в статье [2]. В качестве метаэвристики был выбран алгоритм имитации отжига, описание которого для задачи 2SP в подробностях описан в статье [6].

1. Постановка задачи и обозначения.

1.1. Постановка задачи.

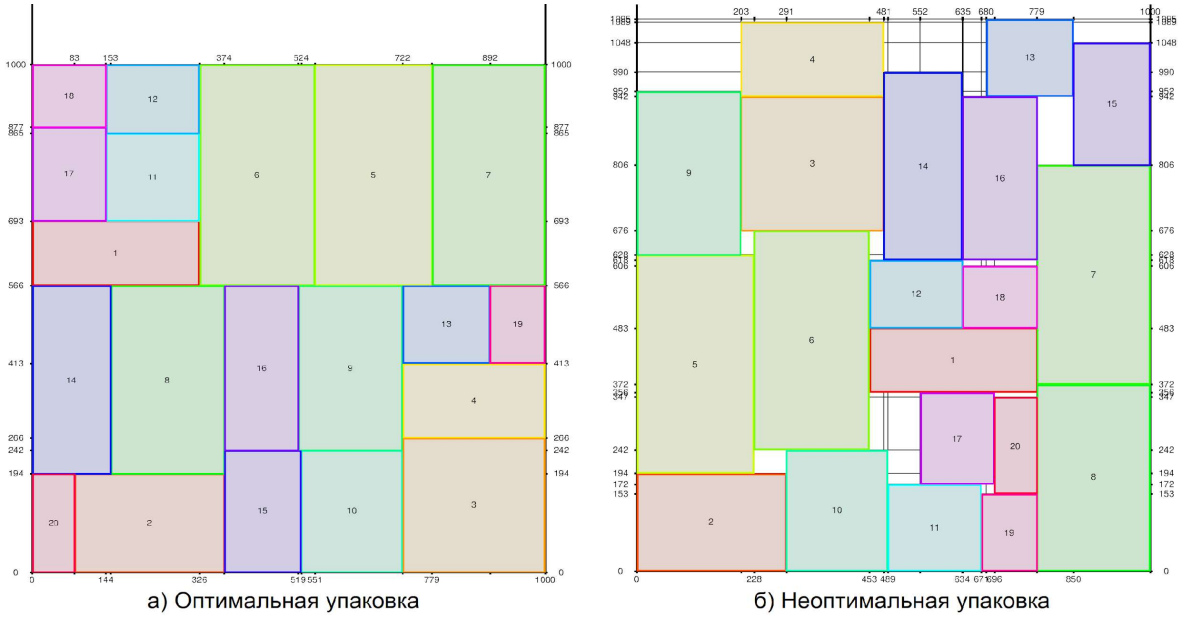
Двумерная задача упаковки в полубесконечную полосу (two-dimensional strip-packing problem, 2SP) состоит в упаковке n прямоугольников (множество P) в полосу ограниченной ширины W и неограниченной сверху высоты, стороны полосы пересекаются под прямыми углами. Левый нижний угол полосы имеет координаты $(0,0)$, правый нижний соответственно $(W,0)$. Каждый прямоугольник $i \in P$ имеет ширину w_i и высоту h_i. Упаковка прямоугольников происходит параллельно или перпендикулярно сторонам полосы, при упаковке вращать прямоугольники запрещено. При упаковке объекты не должны пересекаться друг с другом или выходить за пределы полосы. Установленный прямоугольник $i \in P$ имеет координаты левого нижнего угла (x_i, y_i). Необходимо минимизировать высоту занимаемой полосы H. Математически строго условие задачи выглядит следующим образом:

$$\min \mathrm{H}$$

$$\forall i \in P: \quad (0 \le x_i) \quad \wedge \quad (x_i + w_i \le W) \quad \wedge \quad (0 \le y_i) \quad \wedge \quad (y_i + h_i \le H)$$

$$\forall i,j \in P,\ i \ne j: \quad \left(x_i + w_i \le x_j\right) \vee \left(x_j + w_j \le x_i\right) \vee \left(y_i + h_i \le y_j\right) \vee \left(y_j + h_j \le y_j\right)$$

На следующем рисунке можно увидеть пример оптимальной и неоптимальной упаковки одной и той же задачи:



а) Оптимальная упаковка б) Неоптимальная упаковка

Рисунок 1

1.2. Принцип нормальных паттернов.

Далее во всех алгоритмах мы придерживаемся принципа нормальных паттернов (principle of normal patterns). Принцип строится на основе следующего наблюдения: каждый установленный прямоугольник в решении можно передвинуть насколько можно влево и вниз до тех пор, пока ни один объект нельзя будет передвинуть вниз или влево. Очевидно, что полученное решение не ухудшит исходное решение. Таким образом можно ввести для каждого $i \in P$ прямоугольника множества X_i и Y_i – множества соответственно x-координат и y-координат левого нижнего угла, которые этот угол может иметь, если придерживаться данного принципа. Математически эти множества определяются так:

$$X_i = \left\{ x = \sum_{k \in P \setminus \{i\}} w_k \, \xi_k : \quad 0 \le x \le W - w_i, \xi_k \in \{0,1\}, k \in P \setminus \{i\} \right\}$$

$$Y_i = \left\{ y = \sum_{k \in P \setminus \{i\}} h_k \, \xi_k : \quad 0 \le y \le H - h_i, \xi_k \in \{0,1\}, k \in P \setminus \{i\} \right\},$$

где H – некоторая верхняя граница.

Данные множества можно вычислить методом динамического программирования, как описано в статье [4].

2. Обзор литературы.

2.1. Статья «Точный алгоритм для двумерной задачи упаковки в полосу».

В данной статье [2] описаны алгоритмы редукции задачи, множество алгоритмов вычисления нижней границы, эвристические алгоритмы нахождения решения, постобработка решения для улучшения полученного решения.

2.1.1. Редукция задачи.

Прежде чем решать задачу, её можно упростить несколькими вычислительно дешёвыми методами для улучшения нижних границ и работы эвристического упаковщика.

Упростить проблему можно следующими способами:

1) Уменьшить ширину полосы W, не изменяя оптимальное решение.
2) Увеличить ширину каждого прямоугольника, не изменяя оптимальное решение.
3) Зафиксировать некоторые прямоугольники в решении.

2.1.1.1. Изменение ширины полосы.

В связи с тем, что мы устанавливаем объекты так, что каждый из них нельзя сдвинуть вниз или влево, мы можем методом динамического программирования найти наиболее правую точку, координата которой не превышает W, которую могут достать прямоугольники путём вышесказанной упаковки. То есть необходимо найти $W^* = max\{\sum w_i\xi_i : \sum w_i\,\xi_i \le W, \xi_i \in \{0,1\}, \mathrm{i} \in P\}$. Алгоритм нахождения следующий:

1. Инициализируем множество достигнутых x-координат: $R = \{0\}$.
2. Для каждого i-ого прямоугольника дополняем множество R: к каждому элементу x – достигнутой координате – прибавляем w_i. Получаемое число $x + w_i$ - координата, которую можно достигнуть, поставив i-ый прямоугольник. Если $x + w_i \le W$, то мы добавляем число в R.
3. Находим $W^* = max\{\,\mathrm{x}\colon \quad x \in R\}$.
4. Обновляем текущую ширину полосы $W \leftarrow W^*$.

Временная сложность алгоритма составляет $O(nW)$, где n – количество прямоугольников, W – ширина полосы. Алгоритм итерируется всем n

объектам, на каждой итерации обновляется множество достигнутых x-координат, мощность которого $|R| = W + 1$.

2.1.1.2. Изменение ширины объекта.

Аналогичным методом, что и с уменьшением ширины полосы, можно увеличить ширину любого объекта, не изменяя оптимального решения. Допустим мы рассматриваем j-ый прямоугольник. Необходимо найти самую правую точку, которую могут достичь прямоугольники из множества $P \setminus \{i\}$, координата которой не превышает $W - w_j$. То есть необходимо найти

$$W_j^* = max\left\{ \sum_{i \in P \setminus \{j\}} w_i \xi_i : \sum_{i \in P \setminus \{j\}} w_i \xi_i \le W - w_j, \quad \xi_i \in \{0,1\}, i \in P \setminus \{j\} \right\}$$

В таком случае обновлённой шириной будет $w_j^* = W - W_j^*$. Алгоритм нахождения W_j^* следующий:

1. Инициализируем множество достигнутых x-координат: $R = \{0\}$.
2. Для каждого i-ого прямоугольника из $P \setminus \{j\}$ дополняем множество R: к каждому элементу x – достигнутой координате – прибавляем w_i. Получаемое число $x + w_i$ - координата, которую можно достигнуть, поставив i-ый объект. Если $x + w_i \le W - w_j$, то мы добавляем число в R.
3. Находим $W_j^* = max\{\text{x}: \quad x \in R\}$.
4. Обновляем ширину j-го прямоугольника $w_j \leftarrow W - W_j^*$.

Временная сложность алгоритма также составляет $O(nW)$, где n – количество прямоугольников, W – ширина полосы. Оценка аналогична: итерация по $n - 1$ объектам, каждую итерацию обновляем R, мощность которого $|R| \le W$.

Из-за того, что изменение ширины какого-либо прямоугольника может повлиять на изменение ширины других прямоугольников, то для увеличения количества объектов с изменённой шириной применяется следующая эвристика: алгоритм применяется к прямоугольникам по порядку неувеличения ширины.

2.1.1.3. Фиксация некоторых объектов в решении.

Простейшую редукцию можно реализовать следующим образом: упаковать внизу полосы все прямоугольники шириной W. Данная редукция

малоэффективна, поэтому мы используем модифицированные её версии для большего числа зафиксированных объектов. Более простой вариант реализован следующим образом (см. Рисунок 2):

1. Определяем множество широких прямоугольников
$$R = \{j \in P: \quad w_j > W/2\}.$$
2. Сортируем объекты в R в порядке неувеличения ширины.
3. Устанавливаем следующий j-ый прямоугольник из R в самое левое нижнее положение. Левая сторона прямоугольника будет прилегать к левой стороне полосы, так как следующие действия алгоритма этому не противодействуют.
4. Справа от установленного объекта образуется ниша шириной $W - w_j$, высотой h_j. Образуем множество из неустановленных объектов, ширина которых не превышает $W - w_j$:
$$P_j = \{k \in P: \quad w_j + w_k \leq W\}$$
5. С помощью какого-либо эвристического алгоритма пытаемся упаковать прямоугольники из P_j в данную нишу. Если это удается, мы фиксируем установленные объекты, иначе – убираем из решения j-й прямоугольник и объекты из P_j.
6. Удаляем j-й прямоугольник из R.
7. Если R не пусто, возвращаемся в пункт 3.

Временная сложность очевидно будет зависеть от временной сложности эвристического упаковщика. Если упаковщик имеет сложность $O(n^k)$, то алгоритм будет иметь сложность $O\left(|R| \times \left(\max_{j \in R} |P_j|\right)^k\right)$, или, так как $|R|, |P_j| \leq n$, алгоритм имеет сложность $O(n^{k+1})$.

Эта идея может быть обобщена и далее, упаковывая не по одному широкому прямоугольнику, а сразу несколько, образуя свободное справа пространство в общем случае сложной структуры для заполнения узкими прямоугольниками (см. Рисунок 3 [2, стр. 1777]). Чтобы максимизировать количество зафиксированных объектов, мы будем пытаться упаковать все прямоугольники из $R = \{j \in P: \quad w_j > W/2\}$, в случае неудачи удаляя наиболее узкий объект и упаковывать заново до тех пор, пока не найдётся удачная упаковка. Формализовать вышеописанный подход можно таким образом:

1. Определяем множество $R = \{j \in P: \quad w_j > W/2\}$.
2. Ширина объекта с минимальной шириной $w_{min} = min\{w_j: \quad j \in R\}$.

3. Упаковываем элементы из R по порядку неувеличения ширины.
4. Высота упаковки прямоугольников из R: $H_R = \sum_{j \in R} h_j$.
5. Определяем множество узких прямоугольников
$$P^* = \{k: \quad w_k \leq W - w_{min}, k \in P\}$$
6. Упаковываем приближённым алгоритмом объекты из P^* в свободное пространство справа.
7. Высота упаковки объектов из P^*: $H_{P^*} = max\{ y_k + h_k: \quad k \in P^*\}$.
8. Если $H_{P^*} > H_R$, очищаем решение, удаляем из R все прямоугольники шириной w_{min}, и, если R не пусто, возвращаемся в пункт 2. Иначе, если $H_{P^*} \leq H_R$, фиксируем упакованные объекты и завершаем алгоритм.

Временная сложность также будет зависеть от сложности упаковщика. Если упаковщик имеет сложность $O(n^k)$, то алгоритм аналогично будет иметь сложность $O\left(|R| \times \left(\max_{j \in R} \left|P^{*j}\right|\right)^k\right)$, или, так как $|R|, \left|P_j\right| \leq n$, алгоритм имеет сложность $O(n^{k+1})$.

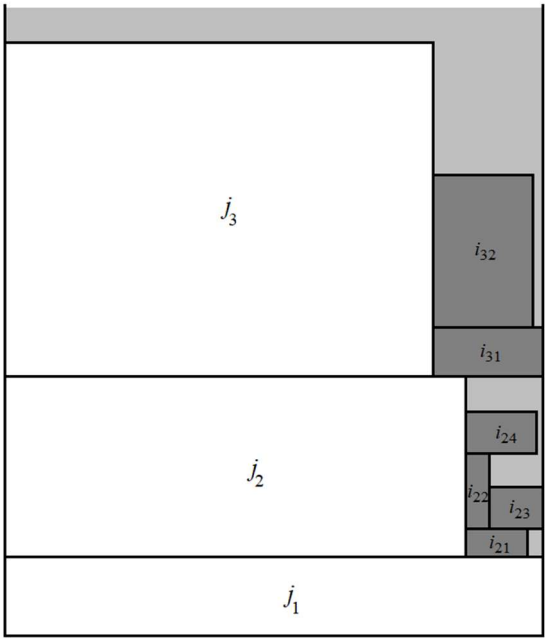


Рисунок 2

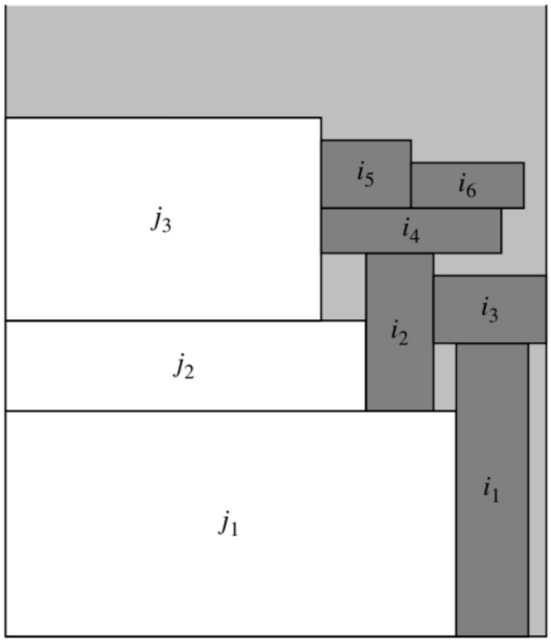


Рисунок 3 [2, стр. 1777]

2.1.2. Нижние границы.

Нижние границы необходимы для определения точности определённого алгоритмами значения целевой функции, также нередко текущие нижние

границы используются эвристическими упаковщиками и даже точным алгоритмом.

2.1.2.1. Простые нижние границы.

Наиболее тривиальными нижними границами являются: непрерывная нижняя граница $L_c = \lceil \sum_{i \in P} w_i h_i / W \rceil$ (continuous lower bound, основана на том, что площадь занимаемой полосы не меньше площади всех прямоугольников), и высота самого высокого объекта: $L_h = \max_{i \in P} h_i$.

Обе величины вычисляются за $O(n)$.

2.1.2.2. Нижние границы, основанные на двойственных выполнимых функциях.

Следующие нижние границы основаны на понятии двойственных выполнимых функций (dual feasible function). Основная идея этих функций – преобразовать ширины объектов таким образом, чтобы для любого множества объектов, плотно упакованных бок о бок по горизонтали, эти объекты также могут быть упакованы с новыми ширинами. Двойственная выполнимая функция имеет следующее определение:

Определение 1. Функция $f: [0, W] \to [0, W']$ называется двойственной выполнимой, если для любого конечного множества S неотрицательных вещественных чисел выполняется $\sum_{w \in S} w \leq W \Longrightarrow \sum_{w \in S} f(w) \leq f(W) = W'$.

В общем случае значения двойственной выполнимой функции не зависят от данных задачи. Тем не менее существуют зависимые от данных двойственные выполнимые функции (data-dependent dual feasible function). Такие функции могут быть определены следующим образом:

Определение 2. Функция $f: [0, W] \to [0, W']$ называется зависимой от двойственной выполнимой, если для любого подмножества $S \subseteq P$ объектов выполняется $\sum_{i \in S} w_i \leq W \Longrightarrow \sum_{i \in S} f(w_i) \leq f(W) = W'$.

С определённой двойственной выполнимой функцией можно получить действительную нижнюю границу вычислив непрерывную нижнюю границу: $L_f = \left\lceil \frac{\sum_{i \in P} f(w_i) h_i}{f(W)} \right\rceil$

Перечислим используемые в данной статье двойственные выполнимые функции:

a) Пусть $\alpha \in \Omega_1 \subseteq \mathbb{N}$. Функция $f^1(w)$ определена следующим образом:

$$f_\alpha^1(w) = \begin{cases} w, & \text{если } (\alpha + 1)\frac{w}{W} \in \mathbb{Z} \\ \left\lfloor (\alpha + 1)\frac{w}{W} \right\rfloor \frac{W}{\alpha}, & \text{иначе} \end{cases}$$

b) Пусть $\alpha \in \Omega_2 = [1, W/2]$, $\Omega_2 \subseteq \mathbb{N}$. Функция $f^2(w)$ определена следующим образом:

$$f_\alpha^2(w) = \begin{cases} W, & \text{если } w > W - \alpha \\ w, & \text{если } \alpha \le w \le W - \alpha \\ 0, & \text{если } w < \alpha \end{cases}$$

c) Пусть $\alpha \in \Omega_3 = [1, W/2]$, $\Omega_3 \subseteq \mathbb{N}$. Функция $f^3(w)$ определена следующим образом:

$$f_\alpha^3(w) = \begin{cases} 2\left(\left\lfloor \frac{W}{\alpha} \right\rfloor - \left\lfloor \frac{W-w}{\alpha} \right\rfloor\right), & \text{если } w > \frac{W}{2} \\ \left\lfloor \frac{W}{\alpha} \right\rfloor, & \text{если } w = \frac{W}{2} \\ \left\lfloor \frac{w}{\alpha} \right\rfloor, & \text{если } w < \frac{W}{2} \end{cases}$$

d) Пусть $\alpha \in \Omega_4 = [1, W/2]$, $\Omega_4 \subseteq \mathbb{N}$, $I_\alpha = \{i \in P: \quad w_i \ge \alpha\}$, $M(w, I) = max\{\sum_{k\in I} \xi_k : \sum_{k\in I} w_k \xi_k \le w, \xi_k \in \{0,1\}, k \in I\}$ (фактически $M(w, I_\alpha)$ – количество прямоугольников наименьшей ширины, но с шириной не менее α, сумма ширин которых не превышает w). Функция $f^4(w)$ определена следующим образом:

$$f_\alpha^4(w) = \begin{cases} M(W, I_\alpha) - M(W - w, I_\alpha), & \text{если } \frac{W}{2} < w \le W \\ 1, & \text{если } \alpha \le w \le \frac{W}{2} \\ 0, & \text{если } w < \alpha \end{cases}$$

Нижние границы также можно получить с помощью композиции двойственных выполнимых функций. Так функция $f_{\alpha,\beta}^{i,j}(w)$ определяется как $f_{\alpha,\beta}^{i,j}(w) = f_\alpha^i\left(f_\beta^j(w)\right)$. Автор статьи предлагает использовать следующую нижнюю границу на основе двойственных выполнимых функций:

$$L_{dff}^{BM} = \max_{k=1,2,3,4} \left\{ \max_{\alpha\in\Omega_k, \beta\in\Omega_2} \left\{ \left\lceil \frac{\sum_{i\in P} f_{\alpha,\beta}^{k,2}(w_i) h_i}{f_{\alpha,\beta}^{k,2}(W)} \right\rceil \right\} \right\},$$

где $\Omega_1 = \{1, \dots, W\}$, $\Omega_k = \{w_i: \quad w_i \le W/2, i \in P\} \cup$

$\cup \{W - w_i : w_i > W/2, i \in P\}$ при $k = 2,3,4$

Оценим временную сложность. Мощность множеств $\forall k = 1,2,3,4$: $|\Omega_k| = O(W)$. Время вычисления функций $\forall k = 1,2,3$ равна $O(1)$. Для $k = 4$ мы сначала сформируем $I = P$, состоящее из всех элементов, каждую итерацию мощность I уменьшается из-за изменения α. В целом все операции изменения I займут $O(n)$ времени. Для каждого нового α мы определяем для каждого $w \in [0, W]$ функцию $M(w, I_\alpha)$, что занимает $O(W)$. Таким образом само вычисление функции $f_\alpha^4(w)$ будет занимать $O(1)$.

При $k = 1,2,3$ тратится $O(W^2)$ времени (итерация по $\alpha \in \Omega_k$ и $\beta \in \Omega_2$). При $k = 4$: $O(n)$ – изменение I, $O(W)$ – итерация по $\alpha \in \Omega_4$, каждая итерация – вычисление $M(w, I_\alpha)$ за $O(W)$ и итерация по $\beta \in \Omega_2$ за $O(W)$. Таким образом общая временная сложность составляет $O(n + W^2)$.

2.1.2.3. Нижние границы, основанные на высотах объектов.

Далее в статье описываются нижние границы, основанные на высоте объектов. Одна из них уже была упомянута: $L_h = \max\limits_{i \in P} h_i$. Для следующих используется величина $Lay = \left\lceil \frac{\sum_{i \in P} w_i}{W} \right\rceil$, характеризующее количество слоёв, которые заняли бы объекты, будь они высотой 1, той же ширины, но упакованы бок о бок, а при выходе за W лента “отрезается” и переходит на следующий слой. Так как на каждом слое есть хотя бы один прямоугольник, высота слоя как минимум составляет высоту самого низкого объекта. А так как слоёв всего Lay, то мы берём первые Lay объектов с минимальной высотой:

$$L_h^1 = \sum_{k=1}^{Lay} h_k,$$

где объекты отсортированы по неуменьшению высоты

Нижняя граница L_h^1 может быть и далее улучшена по следующим соображениям. Мы определяем первые $Lay - 1$ слоёв $Lay - 1$ элементами наименьшей высоты, обозначим множество этих объектов как I_{Full}. Самый верхний слой имеет ширину как минимум $w_{top} = (\sum_{i \in P} w_i - 1) \,\%\, W + 1$, поэтому эту ширину мы заполним оставшимися объектами наименьшей высоты из $P \setminus I_{Full}$, это множество мы обозначим как I_{Top}. То есть для множества I_{Top} выполняются $\sum_{i \in I_{Top}} w_i \geq w_{top}$ и $\sum_{i \in I_{Top} \setminus \{j\}} w_i < w_{top}$, где j – самый высокий объект в I_{Top}. Нижняя граница таким образом имеет вид:

$$L_h^2 = \sum_{k \in I_{Full}} h_k + \max_{k \in I_{Top}} h_k$$

Аналогично можно определить L_h^3, но определив сначала I'_{Top}, а затем I'_{Full}. I'_{Top} - множество объектов наименьшей высоты из P такие, что $\sum_{i \in I'_{Top}} w_i \geq w_{top}$ и $\sum_{i \in I'_{Top} \setminus \{j\}} w_i < w_{top}$, где j – самый высокий объект в I'_{Top}. Далее мы берём $Lay - 1$ объектов наименьшей высоты из $P \setminus I'_{Top}$, это множество I'_{Full}. Нижняя граница имеет вид:

$$L_h^3 = \sum_{k \in I'_{Full}} h_k + \max_{k \in I'_{Top}} h_k$$

Между L_h, L_h^2 и L_h^3 нет никаких отношений доминирования, то есть любая из трёх нижних границ может быть лучше двух других, но L_h^2 и L_h^3 доминируют над L_h^1, поэтому общая нижняя граница по высотам объектов имеет вид:

$$L_h^{BM} = max\{L_h, L_h^2, L_h^3\}$$

Определим временную сложность. Как было сказано ранее L_h вычисляется за $O(n)$. Оценим величину Lay: так как $\forall i \in P: w_i \leq W$, тогда $Lay = \left\lceil \frac{\sum_{i \in P} w_i}{W} \right\rceil \leq \left\lceil \frac{\sum_{i \in P} W}{W} \right\rceil = n \Rightarrow Lay = O(n)$. Так как L_h^1 – сумма Lay элементов, то временная сложность вычисления L_h^1 равна $O(n)$. Для вычисления L_h^2 и L_h^3 необходимо перебрать некоторую часть объектов с малой высотой с помощью элементарных операция, так что временная сложность их вычисления также равна $O(n)$. Таким образом общая сложность вычисления L_h^{BM} равна $O(n)$.

2.1.2.4. Нижняя граница L_{F1}^{BM}.

Следующая нижняя граница основана на математических формулировках. Перед определением L_{F1}^{BM} введём необходимые для формулировки упрощённой задачи определения. X_i – множество x-координат, которые может иметь левый нижний угол прямоугольника $i \in P$ при соблюдении принципа нормальных паттернов (см. 0). Далее введём множество $X' = \bigcup_{i \in P} X_i$. Переменные задачи $t\_\{ix\}$ – двоичные переменные, принимающие значения 1, если прямоугольник $i \in P$ установлен так, что левый нижний угол имеет x-координату равную x. Множества $\alpha(i, x)$ определяются как $\alpha(i, x) = \{\overline{x} \in X_i: \quad max\{0, x - w_i + 1\} \leq \overline{x} \leq x\}$. Это множества x-координат, в которые объект $i \in P$ будучи установлен, будет пересекать координату x. Первая нижняя граница L_{F1}^{BM} является решением упрощённой задачи, формулируемой следующим образом:

$$min\, h$$

$$\sum_{x \in X_i} t_{ix} = 1, \quad \forall i \in P$$

$$\sum_{i \in P} h_i \sum_{\overline{x} \in \alpha(i,x)} t_{i\overline{x}} \leq h, \quad \forall x \in X'$$

$$t_{ix} \in \{0,1\}, \quad \forall i \in P \ \ \forall x \in X_i$$

Другими словами, релаксация проблемы состоит в том, что прямоугольники можно «разрезать» на вертикальные части, каждую из которых можно передвигать только вверх или вниз (см. Рисунок 4)

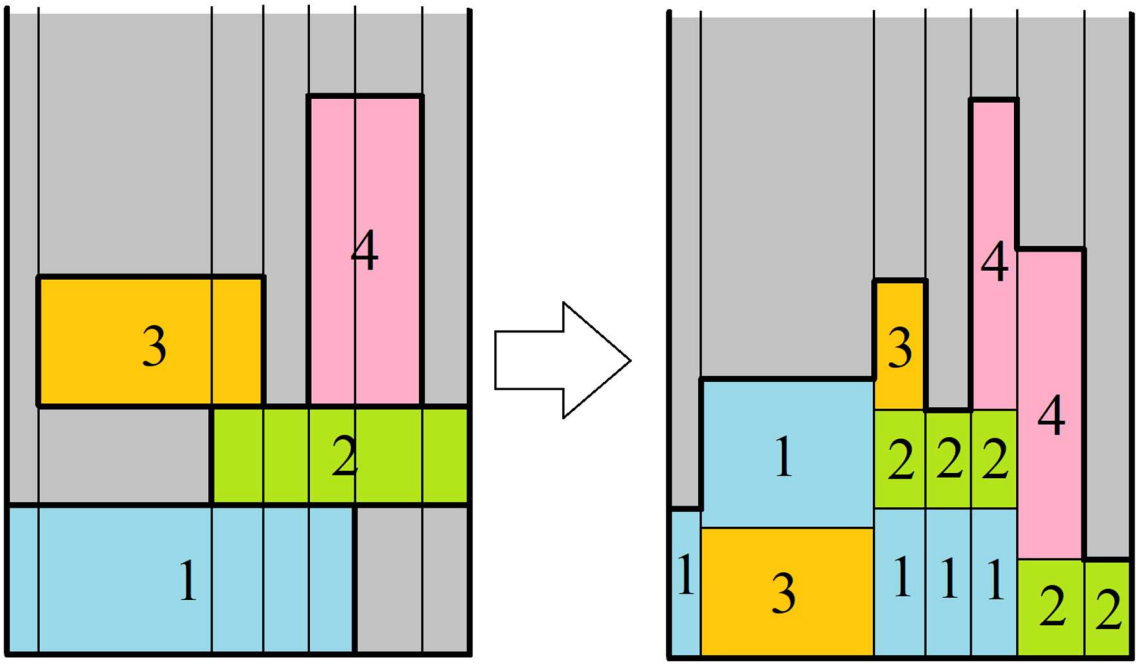


Рисунок 4

Автор статьи предлагает решать эту задачу с помощью программного обеспечения, решающего задачи линейного и целочисленного программирования, под названием CPLEX. Для лучшего исполнения предлагается добавить больше ограничений: $\sum_{j \in P(i,x)} t_{j,x-w_j} \geq t_{ix}, \quad \forall i \in P \ \ x \in X_i$, где $P(i,x) = \{j \in P \setminus \{i\}: \quad x - w_j \in X_j\}$.

Временная сложность вычисления этой нижней границы как минимум экспоненциальная, так как CPLEX применяет метод ветвей и границ для решения этой задачи.

2.1.2.5. Постобработка для улучшения нижней границы.

В завершение автор предлагает несколько способов постобработки для улучшения нижней границы.

Первый основан на принципе нормальных паттернов. Любая упаковка либо придерживается этого принципа, либо, если не придерживается, может быть приведена к упаковке, которая соответствует принципу, не ухудшая решение. Тогда улучшение получается из решения задачи динамического

программирования $\mathrm{LB}' = min\{\mathrm{y} = \sum_{k\in P} h_k\xi_k : \quad L\mathrm{B} \le y \le U\mathrm{B}, \xi_k \in \{0,1\}, \mathrm{k} \in P\}$. Временная сложность этого алгоритма равна $O(n \cdot UB)$, так как происходит итерация по n объектам, каждую итерацию обновляется множество достигнутых y-координат, мощность которого равна $UB + 1$.

Второй является вычислительно более дорогим. Мы выбираем $\hat{h} \in [LB, UB)$. Далее мы «поворачиваем» задачу: теперь ширина полосы - $\hat{h}$, каждый i-ый прямоугольник имеет ширину h_i и высоту w_i. Теперь мы вычисляем нижние границы новой задачи с помощью алгоритмов, описанных выше (автор не рекомендует использовать L_{F1}^{BM} из-за огромной вычислительной стоимости). Если полученная нижняя граница $LB_W \le W$, это значит, что объекты вероятно возможно упаковать в прямоугольник $(\hat{h}, W)$, и стоит уменьшить рассматриваемое $\hat{h}$. Иначе же рассматриваемого $\hat{h}$ недостаточно, и мы можем обновить текущую нижнюю границу оригинальной задачи $LB' = \hat{h} + 1$. Автор предлагает провести эту процедуру для $\hat{h}$ начиная с $UB - 1$ до LB, пока не найдётся улучшение нижней границы.

Временная сложность этого алгоритма следующая: задача «поворачивается» за $\mathrm{O(n)}$, далее выполняется максимум $UB - LB$ итераций, в которых вычисляются лёгкие нижние границы: L_c, L_h^{BM}, L_{dff}^{BM}, вычисление которых занимает $O(n + W^2)$. Общая временная сложность получается равной $O\big((UB - LB) \cdot (n + W^2)\big)$

2.1.3. Эвристические алгоритмы.

2.1.3.1. Normal pattern shifting.

Первый алгоритм, называемый сдвигом нормальных паттернов (normal pattern shifting, NPS), основан на принципе нормальных паттернов. Объекты отсортированы по какому-либо критерию (по неувеличению площади, высоты и т. д.), а также с помощью назначения «цены», как описано ниже.

Алгоритм оперирует seed-позициями – точками в полосе, определёнными уже установленными объектами. Всего три типа seed-позиций: стартовая, верхняя левая и нижняя правая. Стартовая позиция всего одна – это точка $(0,0)$. В начале полоса пуста, и список доступных seed-позиций содержит лишь стартовую позицию. Каждую итерацию в полосу устанавливается i-ый объект левым нижним углом в одну из доступных seed-позиций. Если эта позиция верхняя левая, то сдвигаем объект налево настолько, сколько возможно, если позиция нижняя правая – сдвигаем насколько возможно вниз. Далее в список добавляются две новые seed-

позиции: координаты левой верхней и нижней правой вершин. Количество доступных seed-позиций растёт со сложностью $O(n)$, где n – количество установленных прямоугольников.

Алгоритм выбирает доступную seed-позицию для i-ого объекта из списка доступных следующим образом. Производится попытка установить объект в каждую seed-позицию (если установленный объект перекрывается с другими объектами или краями полосы, то данная позиция отбрасывается из рассмотрения). Затем объект передвигается по вышеописанным правилам. Из всех полученных позиций выбирается позиция с наименьшей y-координатой, если таких несколько – с наименьшей x-координатой.

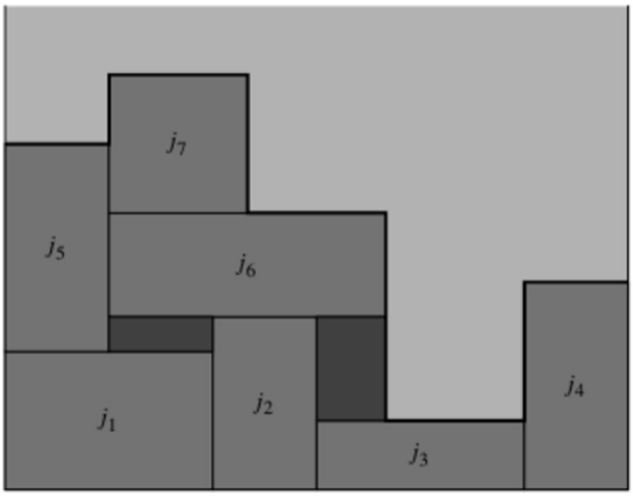
Процедура NPS может быть повторена несколько раз с разным порядком устанавливаемых объектов. Так вводится понятие цены объекта, по которому список сортируется по неувеличению цены. В начале каждый объект $i \in P$ имеет исходную цену p_i, это может быть площадь, ширина, высота, периметр объекта и т. п., либо же фиксированное значение, например $p_i = 100$. Мы запускаем NPS, по полученному решению, имеющему высоту H мы изменяем цены на основании seed-позиции (x_i, y_i), в которой находится i-ый объект (его левая нижняя вершина). Если объект находится в нижней половине, то есть $y_i \leq H/2$, цена объекта уменьшается: $p_i \leftarrow \alpha p_i$, где $\alpha < 1$. Иначе, если $H/2 < y_i \leq H$, цена объекта увеличивается: $p_i \leftarrow \beta p_i$, где $\beta > 1$. α и β – числа, сгенерированные следующим образом: $\alpha = 1 - r$, $\beta = 1 + r$, где r – случайное число в интервале $(0,1)$.

Оценим временную сложность NPS. Производится n итераций установки объектов. Каждую итерацию производится попытка поставить объект в $O(n)$ seed-позиций. В каждой попытке определяется неперекрываемость с $O(n)$ уже установленными объектами, передвижение объекта тратит также $O(n)$ времени на проверку неперекрываемости. Таким образом получаем оценку временной сложности NPS, равную $O(n^3)$. Изменение цен стоит $O(n)$ времени.

2.1.3.2. Priority Best-Fit.

Следующий алгоритм имеет название наилучший подходящий с приоритетом (Priority Best-Fit, PBF). В этом алгоритме вводится понятие горизонта (skyline) – линии, образованной верхними сторонами объектов, имеющих наибольшую y-координату. Горизонт состоит из множества платформ (gaps), одна из которых называется нишей (niche) – самая низкая левая платформа (см. Рисунок 5 [2, стр. 1782], Gap_4 на рисунке - ниша).

(a) For the given emerging solution the silhouette of the skyline is described by the bold line.



(b) Each gap, represented by a bold line, corresponds to an available position.

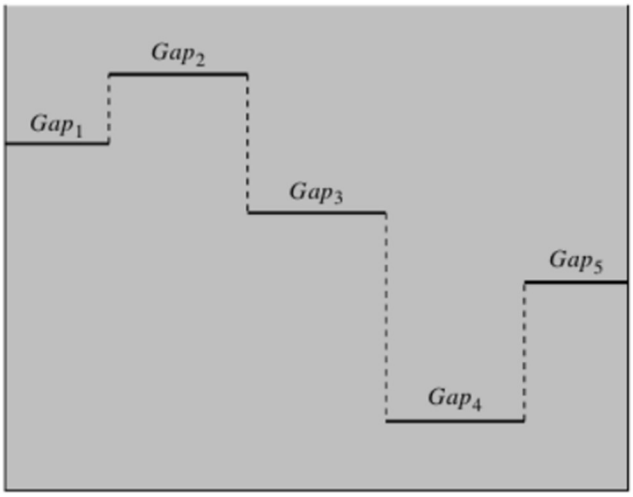


Рисунок 5 [2, стр. 1782]

Алгоритм ищет нишу, затем подбирает подходящий прямоугольник для установки в нишу с помощью определённого критерия, которые будут описаны ниже. Если ширина всех прямоугольников больше ширины ниши, то его высота поднимается до высоты соседней платформы с меньшей высотой, и поиск ниши и подходящего прямоугольника возобновляется. Если подходящий прямоугольник найден, он устанавливается по следующим правилам: если ширина прямоугольника совпадает с шириной ниши, то проблем нет, иначе, если ширина объекта меньше ширины ниши, обозначим высоту левой соседней платформы h_l, высоту правой соседней платформы h_r, y-координату установленного в нишу объекта h^*, тогда правила установки следующие:

a) Если ниша находится не у края полосы и $h_l \neq h_r$, то в случае $h = h_l$ объект выравнивают по левому краю ниши, в случае $h = h_r$ – по правому краю. В случае $h \neq h_l$ и $h \neq h_r$ объект выравнивают по тому краю ниши, со стороны которой платформа выше, то есть если $h_l > h_r$ – по левому краю, иначе – по правому.

b) Если ниша находится не у края полосы и $h_l = h_r$, то объект выравнивают в сторону ближайшего края полосы.

c) Если ниша находится у края полосы, то есть вместо левой или правой соседней платформы – край полосы, тогда если $\mathrm{h} = h_l$ ($h = h_r$), то объект выравнивают в сторону соседней левой (правой) платформы. Иначе – выравнивание в сторону правого (левого) края полосы.

Рассмотрим критерии выбора объекта для установки в нишу. Критерии могут быть классифицированы на сильные (hard selective) и слабые (weak selective). Перечислим сильные критерии:

1. $(\mathrm{h}.1)$ – объект имеет ту же ширину, что и ниша.
2. $(\mathrm{h}.2)$ – высота объекта с нишей равна высоте одной из соседних платформ.
3. $(h.3)$ – высота объекта с нишей равна высоте левой соседней платформе, если же слева край полосы – объект имеет наибольшую высоту среди объектов, вмещающихся в нишу.
4. $(h.4)$ – объект заполняет нишу вместе с другими объектами, даже разной высоты

Перечислим слабые критерии:

1. $(w.1)$ – объект имеет наибольшую высоту.
2. $(w.2)$ – объект имеет наибольшую ширину.
3. $(w.3)$ – объект имеет наибольшую площадь.
4. $(w.4)$ – установка объекта с другими объектами, имеющими ту же высоту, максимизирует ширину ниши.
5. $(w.5)$ – установка объекта с другими объектами, даже имеющими разную высоту, максимизирует ширину ниши.
6. $(w.6)$ – объект максимизирует плотность образующегося решения.

Критерии могут быть скомбинированы. Могут быть использованы не все комбинации, так как большинство из них не имеют смысла. Далее перечислены некоторые из них, которые предлагает автор статьи. Числа означают порядок применения критериев (см. Рисунок 6 [2, стр. 1783])

	Combinations																			
Criteria	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
h.1	2	1			1	1		1	1	1				2			1			2
h.2					2	2							1		1				1	
h.3	1	2	1	1			1				2			1		1				
h.4																2	2	2	2	
w.1	3	3		3				2				2							3	
w.2					3					3	3				3					
w.3						3	2		2				2			3	3	3		3
w.4			2	2						2	1	1			2			1		
w.5																				1
w.6			3											3						

Рисунок 6 [2, стр. 1783]

Далее описаны две эвристики для улучшения вышеописанного алгоритма.

Так как PBF действует жадно, рассматривая лишь локальную выгоду, он может оставить высокие объекты под конец, что сильно увеличит решение. Поэтому была предложена процедура опережающего просмотра (Look-ahead): пусть E – множество установленных объектов, i – объект выбранный критерием, j – наиболее высокий объект, вмещающий в нишу, A_E – оценка площади пустого места после упаковки j-ого объекта, A_M – суммарная площадь оставшихся объектов. Вычисляются A_E и A_M следующим образом: пусть $\overline{H}$ – высота решения после установки j-ого объекта, $\overline{y}$ – высота ниши, тогда $A_E = W(\overline{H} - \overline{y}) - \sum_{k \in E} w_k \Delta h'_k(\overline{y})$, где $\Delta h'_k(\overline{y}) = max\{\overline{y}, y_k + h_k\} - max\{\overline{y}, y_k\}$. Если $A_E > A_M$, устанавливается j-ый объект, иначе мы проводим процедуру оценки при установке i-ого объекта. Если на следующей итерации $A_E > A_M$, то мы устанавливаем j-ый объект, иначе - i-ый объект.

Ход алгоритма сильно зависит от прямоугольников, которые устанавливаются первыми. Поэтому была предложена процедура тёплого старта (Warm Start). Мы выбираем случайное подмножество объектов, сумма ширин которых максимизирована и не превышает W. Эти объекты устанавливаются в самое начало полосы. Авторы предлагают повторять PBF вместе с этой эвристикой $\lceil MS/n^2 \rceil$ раз, где n – количество объектов, $MS = 2 \times 10^6$.

Оценим временную сложность PBF, пока не учитывая Warm Start и Look-Ahead. Производим n итераций установки объектов. Определение с помощью критерия подходящего объекта занимает $O(n)$ времени. Установка с обновлением горизонта занимает $O(W)$. Таким образом получаем временную сложность $O(n^2 W)$. Учтём процедуры Warm Start и Look-Ahead. Первая из них выполняется лишь в самом начале один раз, но реализация выбора случайного заполнения первого слоя может быть различной. Тем не менее известно, что можно добиться временной сложности $O(nW)$

(подробности в разделе **3. Реализация.**). Процедура Look-ahead выполняется каждый раз после выбора критерием объекта для установки, то есть n раз. Для вычисления A_E и A_M выполняется элементарные операции с $O(n)$ объектами (уже установленными и свободными соответственно). Таким образом процедура Look-Ahead добавляет $O(n^2)$ времени в работу алгоритма. Общая временная сложность остаётся быть равной $O(n^2W)$.

2.1.3.3. Постобработка для улучшения решения.

Аналогично нижним границам автор предлагает улучшить решение поворотом задачи. Мы выбираем $\hat{h} \in [LB, UB)$, определяем задачу с шириной полосы $\hat{h}$, прямоугольниками для $i \in P$ с шириной h_i и высотой w_i. Далее вызываем эвристические алгоритмы. Если полученное решение $UB_W \leq W$, то можно упаковать объекты в прямоугольник $(\hat{h}, W)$ и обновить верхнюю границу оригинальной задачи $\mathrm{UB} = \hat{h}$, а также решение, «повернув» его. Автор предлагает выполнять эту процедуру для разных $\hat{h}$ начиная с LB до $\mathrm{UB} - 1$, пока не найдётся подходящее решение.

Временная сложность зависит от временной сложности эвристического алгоритма. Если его сложность $O(n^k)$, тогда общая сложность равна $O\left((UB - LB) \cdot n^k\right)$.

2.1.4. Точный алгоритм.

Точный алгоритм основан на методе ветвей и границ (Branch and Bound, BB). Метод представляет множество решений задачи в виде дерева, по которому происходит обход по каждой вершине, начиная с корневой, минимизируя целевую функцию на вершинах. При полном обходе дерева мы бы произвели обычный полный перебор, метод предполагает наличие легко вычисляемых функций нижней и верхней границ на каждом поддереве, которые могут позволить отсекать поддеревья с заведомо плохими решениями. Так как мы имеем задачу минимизации, то алгоритм будет запоминать лишь глобальную лучшую верхнюю границу, и нижнюю границу на каждом поддереве. Если на каком-либо поддереве нижняя граница выше верхней границы, то производить поиск в этом поддереве бессмысленно, и оно отбрасывается из поиска.

Автор предлагает следующее дерево поиска. Каждая вершина представляет собой горизонт и множество неупакованных прямоугольников. Корневая вершина имеет нулевой горизонт (ни один объект не установлен).

Дочерние узлы корневой вершины означают упаковку в $(0,0)$ одного прямоугольника, таким образом у корневой вершины имеется n дочерних.

Все вершины кроме корневой имеют другого рода множество дочерних узлов. Вследствие использования структуры данных в виде горизонта, автор определяет понятие угла (corner): каждая платформа k определяется триплетом $\left(x_k^l, x_k^r, y_k\right)$ – x-координаты левого и правого углов и y-координата платформы. Углами горизонта являются точки $\left(x_k^l, y_k\right)$.

Для каждой вершины кроме корневой определяется угол, в который будут устанавливаться объекты для определения дочерних узлов – это угол с наименьшей y-координатой, а среди углов с наименьшей y-координатой имеет наименьшую x-координату.

С определённым углом горизонта для установки объекта определяются дочерние узлы: все свободные объекты, которые могут быть установлены левым нижним углом в данный угол горизонта без перекрытия горизонта и выхода за полосу, определяют свой дочерний узел с обновлённым горизонтом. Также, если горизонт не ровный, есть дочерний узел, отвечающий за случай, когда в угол не установлен ни один объект. В данном случае дочерний узел обновляет горизонт удалением ниши с данным углом.

Алгоритм начинается с определения нижней и верхней границ для корневой вершины с использованием L_c, L_h^{BM}, L_{dff}^{BM}, L_{F1}^{BM} для нижней границы, PBF и NPS для верхней границы. Обход по дереву осуществляется поиском в глубину. При переходе в дочерний узел вычисляется его нижняя граница и сравнивается с глобальной верхней границей. Если нижняя граница больше или равна верхней границе, происходит возврат к родительской вершине. При переходе в листовой узел (дочерних узлов нет) мы имеем верхнюю границу с решением, тем самым обновляя глобальную верхнюю границу.

Метод ветвей и границ не может быть эффективным без эвристического порядка обхода дочерних вершин. Алгоритм должен посещать вершины с вероятно оптимальным решением первыми, для того, чтобы как можно раньше улучшить верхнюю границу для успешного отсечения поддеревьев без оптимального решения. Автор предлагает следующий способ сортировки дочерних вершин. Определяем $\overline{w} = (W - w_{min}/2)/2$. Если не менее 40% оставшихся объектов имеют ширину больше $\overline{w}$, то вершины сортируются по неувеличению высоты соответствующих объектов, иначе – по неувеличению ширины объектов.

2.2. Статья «Двухэтапный алгоритм умного поиска для двумерной задачи упаковки в полосу».

В данной статье [6] описывается применение метаэвристики имитации отжига для решения задачи 2SP с соответствующими дополнительными алгоритмами для улучшения выполнения основной процедуры имитации отжига.

2.2.1. Эвристический алгоритм.

Алгоритм имитации отжига очень сильно зависит от начального приближения. Решение представляется в виде последовательности объектов, означающей порядок упаковки. Упаковка аналогична PBF: объект упаковывается в нишу, при неудаче ниша удаляется, происходит поиск другой ниши.

Эвристический алгоритм ищет какую-либо хорошую последовательность упаковки. Реализация аналогична PBF из [2, стр. 1781]: наличествует горизонт, поиск нижней левой ниши. Различия наблюдаются лишь в наборе критериев.

Прямоугольник выбирается на основе score-функции. Её значение зависит от ширины $r[i].width$ и высоты $r[i].height$ объекта $r[i]$, ширины ниши w, разницы высоты до левой соседней платформы h_1 и до правой соседней платформы h_2 (см. Рисунок 7 [6, стр. 59]).

Scoring function score (i, h_1, h_2, w) for calculating fitness value *fitness* and the change of number of spaces m.

Conditions		*fitness*	m	Conditions		*fitness*	m
$h_1 \geqslant h_2$	$w = r[i].width$ and $h_1 = r[i].length$	4	−2 or −1	$h_1 < h_2$	$w = r[i].width$ and $h_2 = r[i].length$	4	−1
	$w = r[i].width$ and $h_1 < r[i].length$	3	0		$w = r[i].width$ and $h_2 < r[i].length$	3	0
	$w = r[i].width$ and $h_1 > r[i].length$	2	0		$w = r[i].width$ and $h_2 > r[i].length$	2	0
	$w > r[i].width$ and $h_1 = r[i].length$	1	0		$w > r[i].width$ and $h_2 = r[i].length$	1	0
	$w > r[i].width$ and $h_1 \neq r[i].length$	0	+1		$w > r[i].width$ and $h_1 \neq r[i].length$	0	+1

Рисунок 7 [6, стр. 59]

В полосу устанавливается объект с наивысшим score по тем же правилам, описанным в PBF.

Так как алгоритм аналогичен PBF, его временная сложность не отличается: $O(n^2W)$.

2.2.2. Локальный поиск.

Локальный поиск используется для дальнейшего улучшения решения, полученного эвристическим алгоритм. Алгоритм берёт на вход решение (в виде последовательности объектов) и ищет среди похожих решений лучшее решение. Похожие решения, получаемые при поиске, образуются обменом местами двух объектов в исходной последовательности.

Определяя временную сложность, наблюдаем $O(n^2)$ обменов, каждая полученная последовательность упаковывается за $O(nW)$ (n объектов, каждая установка вызывает обновление горизонта за $O(W)$). Таким образом получаем общую сложность $O(n^3W)$.

2.2.3. Алгоритм имитации отжига.

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Во время работы имеется лишь одно решение, каждую итерацию алгоритм вычисляет на основе текущего новое решение, после этого решает переходить к новому решению или нет. Если новое решение лучше текущего, происходит безусловный переход. Иначе решение о переходе определяется случайным образом на основе переменной температуры T и значений целевой функции нового и текущего решения.

Переменная температуры T определяет вероятность перехода в новое более плохое решение. Формула вероятности перехода заимствована из распределения Гиббса в статистической физике. Пусть x_0 – текущее решение (точка), x_1 – новая точка, $F(x)$ – целевая функция (в нашем случае – высота упаковки последовательности x). Тогда вероятность перехода равна

$$P\{x_0 \leftarrow x_1\} = min\left\{1, exp\left(\frac{F(x_0) - F(x_1)}{T}\right)\right\}$$

Во время работы алгоритма температура T уменьшается от T_0 до нуля. Скорость и закон убывания задаются автором с помощью гиперпараметров α и L. После L итераций переходов из решения в решение, температура уменьшается: $T \leftarrow \alpha T$. Алгоритм заканчивается после выполнения какого-либо условия (например истечение времени).

Автор задаёт следующие гиперпараметры на основе вычислительных тестов: $T_0 = 0.5,\ \alpha = 0.93$.

Новое решение формируется из текущего путём обмена местами двух случайных объектов в последовательности.

Помимо текущего решения также хранится лучшее решение, которое обновляется всякий раз, когда было найдено решение со значением целевой функцией меньше чем текущее лучшее.

2.2.4. Стратегия множественного старта.

Так как алгоритм имитации отжига принимает на вход последовательность, полученную от алгоритмов эвристического и локального

поиска, то для алгоритма имитации отжига тяжело выбраться из ловушки локального оптимума. Это можно исправить, производя поиск несколько раз с разных последовательностей прямоугольников. Автор предлагает две последовательности: по неувеличению ширины и по неувеличению периметра.

2.2.5. Двухэтапный алгоритм умного поиска.

Данный алгоритм является композицией эвристического и метаэвристического алгоритмов. На первом этапе берётся последовательность из процедуры множественного старта, последовательность улучшается с помощью локального поиска. Результат подаётся в алгоритм имитации отжига. Исследования автора показали, что такая комбинация эффективнее чистого алгоритма имитации отжига.

3. Реализация.

Проект реализован на языке C++. Реализована большая часть упомянутых алгоритмов. В процессе разработки использовались следующие вспомогательные классы и инструкции typedef:

1. Rectangle. Объектами являются прямоугольники. Содержит два поля: width и height.
2. Solution. Определяется инструкцией
   ```
   typedef std::vector<std::array<int, 4>> Solution
   ```
   Используется для описания решений, это список объектов из 4-х чисел: x-координата, y-координата, ширина, высота.
3. Problem. Объектами являются задачи 2SP. Определяется полями list – список прямоугольников (для эвристического алгоритма, локального поиска и алгоритма имитации отжига используется для хранения последовательности упаковки), listwidth – список объектов, отсортированных по невозрастанию ширины (значительное количество алгоритмов требуют такую сортировку), Width – ширина полосы, fixed – объект типа Solution, хранит предварительно установленные прямоугольники (необходима во время редукции с фиксацией объектов в решении)

Реализованы функции:

1. Чтение и запись задач и решений из/в файл;
2. Простейший генератор задач;
3. Проверка корректности задач и решений для отладки;
4. Редукция задачи (изменение ширины полосы, изменение ширины объекта, фиксация некоторых объектов в решении);
5. Нижние границы (непрерывная, основанные на высотах объектов, основанные на двойственных выполнимых функциях, L_{F1}^{BM});
6. Постобработка нижней границы (основанная на принципе нормальных паттернов, основанная на нижних границах повёрнутой задачи);
7. Эвристические алгоритмы (PBF, эвристический алгоритм из [6], локальный поиск, алгоритм имитации отжига, Intelligent Simulated Annealing);
8. Постобработка решения и верхней границы (основана на повороте задачи и упаковке с помощью PBF).

Далее отметим несколько пояснений, моментов и улучшений, которые были внесены в реализацию алгоритмов, и не были рассмотрены ранее.

3.1. Генератор задач.

Генератор должен генерировать задачу, для которой известно оптимальное решение. Реализованный генератор основан на непрерывной нижней границе L_c. Берётся прямоугольник с размерами (W, H) и последовательно разрезается т.н. гильотинной нарезкой на n частей. Так мы получим задачу с n прямоугольниками, с шириной полосы, равной W, оптимальным значением H. Для того, чтобы нарезка была сравнительно равномерной, то есть был малый разброс в размерах прямоугольников, выбор следующего прямоугольника для разреза выбирается случайного с весами на его площадь и отношение ширины к высоте.

3.2. Постобработка нижних и верхних границ.

Автор статьи [2] предлагает проводить $UB - LB$ итераций для нахождения соответствующего улучшения верхней и нижней границы. В проекте же был реализован бинарный поиск, производящий в среднем $\lceil log_2 UB - LB \rceil$ итераций. Бинарный поиск используется для нахождения искомого значения монотонной функции. Функции нижней границы и высоты решения от ширины полосы с большой долей уверенности можно назвать монотонной. Бинарный поиск реализован следующим образом на примере поиска улучшения верхней границы: искомое значение изначально находится в пределах $[LB, UB)$. Определим переменные $low = LB$ и $up = UB$ – динамически изменяющиеся границы поиска. Искомое значение в любой итерации находится в пределах $[low, up)$.

1. Инициализируем новую верхнюю границу $UB' = UB$.
2. Пока верно $up - low > 0$ выполнять инструкции:
    a. Находим среднее значение $\lfloor mid = (up + low)/2 \rfloor$, в которой будем искать значение функции.
    b. Упаковываем с помощью PBF в полосу шириной mid, получаем значение высоты решения повёрнутой задачи H_{rot}.
    c. Если $H_{rot} > W$, то mid слишком мал, чтобы PBF смог упаковать в неё повёрнутые объекты. В таком случае изменяется нижняя граница диапазона: $low = mid + 1$.
    d. Иначе, если удалось упаковать, мы сохраняем найденное значение в новую верхнюю границу: $UB' = mid$, далее мы продолжаем поиск в меньших значениях: $up = mid$.
3. Возвращаем UB'.

3.3. Процедура Warm Start.

Источник [2] не указывает в точности как реализовать данную процедуру, поэтому далее будет описана реализация в проекте. Алгоритм несколько повторяет алгоритм редукции с изменением ширины полосы, только в данном случае нам необходима определённая случайная упаковка дна полосы.

Для этого необходимо найти сначала случайную перестановку списка объектов. Это можно получить поэлементно определив случайным образом элемент на 1-м месте, на 2-м месте, и так далее до n-ого места. Допустим мы получили перестановку $(i_1, i_2, \dots, i_n)$

Далее запускается алгоритм определения достигнутых точек, но в отличие от реализации редукции с изменением ширины полосы, необходимо запомнить множество достигнутых точек после рассмотрения каждого i_k-ого объекта. Допустим R_k - множество достигнутых точек после установки i_k-ого объекта, а R_0 – начальное множество достигнутых точек. Как только будет достигнута точка W начинается стадия сборки решения. Если же точка W недостижима, стадия сборки начинается после рассмотрения всех объектов и с точки с максимальной x-координатой, меньшей W.

Определение объектов, заполняющих дно полосы определяется следующим образом:

1. Текущая достигнутая точка хранится в переменной z. Изначально она равна W или максимальной достигнутой точке.
2. Текущий рассматриваемый объект - i_k.
3. Если до установки i_k-ого объекта точка $z - w_{i_k}$ является достижимой (то есть $z - w_{i_k} \in R_k$), то объект i_k устанавливается на дно полосы, переменная z уменьшается: $\mathrm{z} \leftarrow \mathrm{z} - \mathrm{w}_{\mathrm{i_k}}$.
4. Иначе, если $z - w_{i_k}$ не является достижимой ($\mathrm{z} - w_{i_k} \notin R_k$), то поставить объект i_k нельзя.
5. $k \leftarrow k - 1$, если $k > 0$ – возвращаемся в пункт 3.

Полученная упаковка далее используется основным телом функции PBF.

4. Результаты вычислений.

4.1. Гиперпараметры алгоритма имитации отжига.

В источнике [6] нет определённых значений для гиперпараметра L, вследствие чего были проведены вычислительные тесты с разными значениями L на одних и тех же сгенерированных задачах. При определении порядка величины были обнаружены оптимумы в значениях $L = 10$, при которых алгоритм находил больше всего оптимальных решений.

При уточнении значения L в пределах $L \leq 100$ был обнаружен приблизительный оптимум в значении $L = 50$, который и был использован в следующих вычислительных тестах. Оптимум не является точным ввиду необходимости огромного количества вычислительных тестов.

4.2. Сравнение PBF и Intelligent Simulated Annealing.

В рамках сравнения эвристического и метаэвристического подходов были использованы сгенерированные тестовые задачи, образованные гильотинной и негильотинной нарезкой на 25, 50 и 100 объектов. Тестовые задачи разделены на 5 пакетов по 100 тестов. В таблице ниже указаны свойства задач каждого пакета. На каждую задачу алгоритм имел ограничение времени 2 секунды.

Пакет тестов	Количество прямоугольников, n	Ширина полосы, W	Оптимальное значение высоты, $\overline{H}$
guil_25	25	16	24
guil_50	50	20	30
guil_100	100	40	60
nonguil_50	50	20	30
nonguil_100	100	40	60

Таблица 1

Ниже выведены результаты вычислительных тестов: средняя доля отклонения от оптимума $\Delta H\% = \frac{H-\overline{H}}{\overline{H}} \cdot 100\%$ и средняя доля найденных оптимальных решений N_{OPT} в каждом пакете.

Пакет	PBF	ISA
guil_25	0.38%	4.8%
guil_50	0.53%	7.5%
guil_100	1.9%	12%
nonguil_50	0.43%	9.3%
nonguil_100	1.9%	13%

Таблица 2. Средняя доля отклонения от оптимума

Пакет	PBF	ISA
guil_25	91%	9%
guil_50	85%	0%
guil_100	32%	0%
nonguil_50	87%	0%
nonguil_100	27%	0%

Таблица 3. Средняя доля найденных оптимальных решений

Заключение.

Таким образом, в работе были рассмотрены, программно реализованы и исследованы эффективные алгоритмы для приближенного решения задач двумерной упаковки прямоугольников в полубесконечную полосу 2SP. Также приводятся алгоритмы оценки нижних границ точного решения.

Для отобранных алгоритмов Priority Best Fit и имитации отжига проводятся серии вычислительных экспериментов. На основе сравнения результатов делается вывод о преимуществах эвристического алгоритма и необходимость более тонкой настройки метаэвристических алгоритмов с учетом особенностей прикладных задач.

Список литературы

1. Gerhard Wäscher, Heike Haußner, Holger Schumann. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 1109-1130.
2. Marco Antonio Boschetti, Lorenza Montaletti. (2010). An Exact Algorithm for the Two-Dimensional Strip-Packing Problem. *Operations Research*, 1774-1791.
3. Michael R. Garey, David S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.
4. Nicos Christofides, Charles Whitlock. (1977). An Algorithm for Two-Dimensional Cutting Problems. *Operations Research*, 30-44.
5. Silvano Martello, Michele Monaci, Daniele Vigo. (2003). An Exact Approach to the Strip-Packing Problem. *INFORMS Journal on Computing*, 310-319.
6. Stephen C.H. Leung, Defu Zhang, Kwang Mong Sim. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research*, 57-69.