UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Reporte BinPackind 2D

Hecho por:

Edgar Antonio Arroyo Sánchez #1627262

Hora:

Martes V4-V6

Materia:

Temas selectos de Optimización

Impartida por:

Dra. María Angélica Salazar Aguilar

Fecha:

Martes 05/16/2017

Descripción del Proyecto.

En el problema de Bin Packing, objetos de diferentes volúmenes deben ser empaquetados. En el caso de Bin Packing 2D, hablaremos de áreas.

Objetos de diferentes áreas deben ser “empaquetados” dentro de un Bin con un area fija y una altura infinita.

Silvano Martello, lo define como

“Given a set of rectangular pieces to be cut from an unlimited number of standardized stock pieces (bins), the Two-Dimensional Finite Bin Packing Problem is to determine the minimum number of stock pieces that provide all the pieces”

Para nuestras infancias en particular lo podemos definir de la siguiente manera:

“Dada un set de piezas rectangulares para ser cortadas en un bin con un área finita y una altura infinita, encontrar el mejor orden para cortar las piezas que minimice la altura ocupada”

Pseudocódigo I Descripción.

Primero empezamos con el constructivo.
El constructivo es muy sencillo. Para cada pieza rectangular, la cual contiene dos lados, un lado X y un lado Y, rotaremos la pieza de tal manera que la parte más larga quede como ancho y la parte más corta como alto.

```
1
2
3 self.alto = min([X, Y])
4 self.ancho = max([X, Y])
```

Todas las piezas se encuentran en una lista. Ya que todas las piezas de la lista tienen su alto y su ancho, es necesario ordenar todas las piezas de mayor a menor según su atributo de alto, el cual llamaremos SELF.ALTO.

Teniendo la lista ordenada. Empezamos con el constructivo.

Básicamente el constructivo consiste en dividir la tela en telas más pequeñas, que tendrán de altura la misma altura que el primer objeto que se ha “insertado” o de otra manera, del primer objeto que será cortado de esa tira.

El Pedazo de tela, será llamado TELA, y contendrá los siguientes atributos:

```
class Tela:
    def __init__(self, altura, anchura):
        self.altura = altura
        self.anchura = anchura
        self.usado = 0
        self.cosido = []
```

Crearemos una TELA, cada vez que esta sea necesaria.
La haremos con la función constructora mostrada.

En sí, pasaremos cada una de las piezas a una función llamada constructor. Tal función checará si existe una Tela en donde esta pieza pueda entrar. Si no existe, creará una nueva, en donde la altura será igual a la altura de la pieza, para que después, en esa misma tela, entren otras piezas con altura menor.

```
def constructor(piezas_p):
    telas_a = []
    piezas_a = copy.deepcopy(piezas_p)
    i_pieza = 0
    i_tela = 0

    telas_a.append(Tela(int(piezas_a[0].getalto()), ancho_del_telon))
    while i_pieza < len(piezas_a):
        if piezas_a[i_pieza].getancho() <= telas_a[i_tela].getlibre() and piezas_a[i_pieza].getalto() <= telas_a[i_tela].getaltura() and piezas_a[i_pieza].getasignada() is False:
            piezas_a[i_pieza].setasignada()
            telas_a[i_tela].coser(piezas_a[i_pieza].getid())
            telas_a[i_tela].setusado(piezas_a[i_pieza].getancho())
            i_pieza += 1
        else:
            telas_a.append(Tela(int(piezas_a[i_pieza].getalto()), ancho_del_telon))
            i_tela += 1
            if i_tela == len(piezas_a):
                telas_a.append(
                    Tela(int(piezas_a[i_pieza].getancho()), ancho_del_telon))
    return telas_a
```

Primero, creamos una copia de las piezas. Después hacemos un ciclo while, que se encargará de que todas las piezas sean asignadas a un pedazo de tela y básicamente en

en el condicional if verificamos que la pieza a introducir tenga una altura menor o igual a la del pedazo de tela y que al introducirla no rebasemos el ancho especificado.

Ahora pasemos con el movimiento.

Consideremos como solución, el orden con el que las piezas son introducidos a través del constructivo, siendo la solución inicial las piezas ordenadas de mayor a menor según el atributo SELF.ALTO.

El movimiento consiste en ir cambiando el orden, supongamos que tenemos la siguiente lista, con los objetos representados como un número.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

Tomaremos 4 piezas y les aplicaremos reverse(), de tal manera que por ejemplo si tomamos el subconjunto [1, 2, 3 ,4] se devolverá el [4, 3, 2, 1] y hemos de insertarlo respetando el espacio, es decir, quedaría

[4, 3 ,2 ,1, 5, 6 ,7 ,8 ,9, 10,11, 12, 13, 14]

Después continuaríamos con el subconjunto [2, 3, 4, 5] que devolvería [5, 4, 3, 2]
y quedaría como:

[1, 5, 4 ,3, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14]

El funcionamiento general es tener un contador I, que representa la primer pieza que será tomada para formar parte del subconjunto, que formará desde I a I + 4, y continuará desde I = 0 a I = longitud(LISTA_PIEZAS) - 4; donde longitud devuelve el número de piezas en una lista.

Los movimientos son generados siempre y cuando sean pasados y den mejores soluciones. Hay una tolerancia, los movimientos serán generados hasta que se llegue a 10 movimientos sin mejora.

Si sí hay mejora, ese contador se reiniciará y la mejora será tomada como mejor solución, a su vez como solución inicial.

Pasemos al multiarranque.

Con multiarranque hacemos la tarea de generar soluciones alternas, en este caso 200. A las cuales se les realizará el procedimiento de la función constructor y la función de movimientos.

¿Cómo se construyen las soluciones?

Supongamos que se tienen una LISTA_PIEZAS, con objetos pieza donde cada uno tiene su respectiva altura.

Utilizando la fórmula *Cmin + alfa * (Cmax - Cmin)*

Encontramos el tope de altura para el subconjunto de candidatos, siendo alfa definido por nosotros (Valor entre 0 y 1), Cmax, la altura mayor de los candidatos a formar parte del subconjunto y siendo Cmin la altura menor.

Tomaremos todas las piezas donde su altura se encuentra entre Cmin y Cmax.

Después, ya formado el subconjunto, tomaremos una de las piezas al azar para que forme parte de la nueva solución.

A cada una de estas “soluciones” les pasaremos las funciones de constructor y movimiento.

Aquella solución que genere una menor altura será comparada con la solución inicial, si es mejor, se tomará como la mejor solución.

El alfa utilizado para obtener las soluciones, fue de 0.00001

Experimentación computacional.

Descripción de instancias

Las instancias son generosas, la menor es de 100 piezas y la mayor de 150. Contienen el número de pieza así como las medidas de sus lados.
Las medida más grande de un lado de una pieza es 30. Por lo que tampoco se realizan operaciones con números muy grandes.

Lenguaje

El código se ha realizado en Python.
Python tiene la ventaja de ser bastante sencillo, es poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos.

Tiene una sintaxis elegante, con un tipado dinámico, de naturaleza intepretada. Es un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

Especificaciones del equipo

Tabla de resultados

Instancía	Área necesaria	Sol. Inicial	Mejor Sol. Encontrada	GAP ÁreaNecesaria/ ÁreaOcupada	Gap Sol.Inicial/ Sol.Encontrada
M1A	2520	34	34	21.42	0
M1B	2597	35	35	17.55	0
M1C	2554	35	33	16.28	6.06
M1D	2846	36	36	13.84	0
M1E	2665	35	35	18.19	0
M2A	24753	326	326	18.53	0
M2B	25283	330	327	16.40	0.91
M2C	25200	331	331	18.21	0
M2D	25614	330	330	15.95	0
M2E	24811	321	321	16.44	0
M3A	25703	519	519	81.72	0
M3B	27414	526	526	72.68	0
M3C	27953	527	527	69.67	0
M3D	27439	509	509	66.95	0
M3E	29530	501	501	52.69	0

Tabla de Tiempos

Instancia	Tiempo Total	Tiempo Constructor Inicial	Tiempo Movimientos	Tiempo Multiarranque
M1A	95.27	0.003099	0.4198	94.8489
M1B	94.03	0.002944	0.4257	93.6056
M1C	96.24	0.003322	0.4226	95.8154
M1D	95.97	0.003189	0.4331	95.5396
M1E	95.87	0.002834	0.4220	95.4477
M2A	96.66	0.003222	0.4300	96.2346
M2B	97.51	0.003209	0.4314	96.0777
M2C	95.32	0.003346	0.4228	94.8954
M2D	98.38	0.003301	0.4232	97.9566
M2E	97.17	0.003202	0.42006	96.7543
M3A	271.75	0.005090	1.21459	271.5338
M3B	272.76	0.004701	1.20777	271.5514
M3C	265.60	0.004868	1.21040	264.3845
M3D	274.66	0.004802	1.22552	273.4309
M3E	273.55	0.004960	1.20165	272.3482

Conclusiones

Como se puede apreciar, el constructivo tiene ciertas desventajas, una de ella es el desperdicio de espacio. Es decir, se desperdicia espacio entre cada tope de tela y el tope de altura de la pieza que se va a “insertar” o más correctamente cortar”

Podrían hacerse modificaciones al algoritmo constructivo para que este aproveche el espacio de una mejor manera, pero esto sería más complicado.

También podría pensarse en un movimiento que traer de solucionar este problema.

Bibliografía

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management science*, *44*(3), 388-399.

INTRODUCCIÓN — TUTORIAL DE PYTHON 3.6.0 DOCUMENTATION
En el texto: (Docs.python.org.ar, 2017)
Bibliografía: Docs.python.org.ar. (2017). 1. Introducción — Tutorial de Python 3.6.0 documentation. [online] Available at: http://docs.python.org.ar/tutorial/3/real-index.html [Accessed 15 May 2017].