IX Congreso de Ingeniería de Organización
Gijón, 8 y 9 de septiembre de 2005

Un algoritmo simple de resolución del problema de strip-packing 2D con rotación

Jesús Lozano[1]

Palabras clave: *rectangular strip packing 2D, bin packing 2D.*

1. Introducción

El problema de *strip-packing* en 2 dimensiones es considerado como un problema NP-*hard* debido a que el número de soluciones posibles, entre las que se encuentra indistinguiblemente la óptima, esta relacionada con el número de permutaciones de objetos a "empaquetar." Se propone pues, un algoritmo heurístico sencillo que resuelva con un grado aceptable de eficacia cuál es el empaquetado de objetos de menor extensión espacial en un tiempo de computación razonable. Para una introducción al problema véase Lodi, Martello y Monaci (2002).

2. Definición del problema

En nuestra configuración, se dispone de un conjunto finito de objetos rectangulares de dos dimensiones a empaquetar en una tira de material de ancho conocido, y cuya longitud se debe de minimizar. El cálculo considerado se realizará *off-line*, después de conocido el número y tamaño de todos los objetos, y considerando la posibilidad de rotar 90 grados la posición del objeto en la tira de material o posición de almacenamiento, si ello ayuda a la minización de la longitud o el espacio ocupado. La importancia y relevancia práctica del problema en la organización industrial es evidente.

3. Otros algoritmos

Cabe distinguir dos tipos de modelos: los de resolución exacta y los metaheurísticos aproximados. Entre los primeros se encuentran los programas lineales enteros (Martello et al., 2000) y los de enumeración (p.ej. Fekete y Schepers, 1997), de los que cabe decir que son capaces de resolver instancias realistas del problema, si bien poseen un problema de escalado al degenerar sus prestaciones con el tamaño del problema. Otro inconveniente resulta de la imposibilidad de obtener resultados intermedios de acercamiento al óptimo.
Entre las múltiples metaheurísticas aplicadas al problema se encuentran: recocido simulado (Dowsland, 1993), los algoritmos genéticos (Jackobs, 1996) y la búsqueda tabú (Lodi et al., 1999). Mención especial merecen las heurísticas con ordenación y reglas de inserción. Así por ejemplo, para el caso en que no está permitida la rotación y por niveles tenemos las de *Next-Fit Decreasing Height*, *First-Fit Decreasing Height*, *Best-Fit Decreasing Height*, y sin niveles, *Bottom-Left*, con ordenación decreciente en ancho.

[1] Departamento de Administración de Empresas y Contabilidad. Universidad de Oviedo. lozano@epsig.uniovi.es

4. El modelo heurístico simple propuesto

Se parte de un estado inicial en el que se conoce el ancho de la tira y el número de objetos a empaquetar. En nuestro caso, generamos aleatoriamente las dimensiones rectangulares y el orden de tales objetos. El algoritmo en ningún momento usa procedimiento de ordenación alguno.
El modelo propuesto consiste básicamente en un bucle principal que va incrementando la altura conforme se intentan insertar los objetos aleatoriamente, rotándolos o no también aleatoriamente y que va eliminando los objetos que hayan sido previamente insertados debajo. Por supuesto, se almacena e imprime la solución que hasta el momento haya obtenido la mínima altura. Así pues, el algoritmo se comporta de una manera similar a una de las heurísticas más simples, la de intercambio de pares, que en este caso se plasma en un algoritmo *greedy* de inserción y destrucción simultánea (en el mismo bucle) Un algoritmo similar ha sido también probado con aceptables resultados para los problemas *bin packing* 2D.
Como inconvenientes al modelo, es frecuente que se estanque en óptimos locales, por lo que, como es frecuente en las heurísticas, las buenas soluciones procedan de reinicios aleatorios. Otro grave inconveniente es que hace un uso intensivo de la memoria, por lo que su utilización queda restringida a problemas medianos y pequeños en los ordenadores personales.

5. El código Perl

```
#!/usr/local/bin/perl -w

# strip packing 2D with rotation

$HORIZ =  2;        # altura inicial
$VERT  =  60;       # ancho
$GRID = 1;          # escalado (nulo)


for $i (1..$HORIZ*$GRID*10_000){   # máxima altura
    for $j (1..$VERT*$GRID){
        $s[$i][$j]=0;
    }
}


$N = 100;            # número de objetos
for $i (1 .. $N){   # generación aleatoria de dimensiones
    $l = 2+int (rand(10)*$GRID);
    $w = 2+int (rand(6)*$GRID);
    if ($w > $l) { ($w, $l) = ($l, $w); }   # swap
    $o[$i][1]=$l;
    $o[$i][2]=$w;
    $o[$i][3]=0;
    $o[$i][4]=$l*$w;
}


$MAXpercent=0;
for (;;){
    for $i (1 .. $HORIZ*$GRID){
        for $j (1 .. $VERT*$GRID){
            $s[$i][$j]=0;
        }
    }
    $HORIZ=2;
    for $i (1 .. $N){
        $o[$i][3]=0;
    }
```

```
    for (;;){       # bucle principal
        do{
            $k = 1+int(rand($N));
        }until($o[$k][3]==0);  # hasta encontrar un objeto libre
        $h = int (.5+rand());
        $o[$k][3]=1;         # el objeto está asignado
        if ($h==0) {         # rotación de 90 grados
            ($o[$k][1],$o[$k][2])=($o[$k][2],$o[$k][1]); }
        $ok=0;
        $HORIZ++;            # se sube la altura en una unidad
        $cc=0;
        $kontador=0;
        for $i (1..$HORIZ*$GRID){
            $a = $i+$o[$k][1];
            for $j (1..$VERT*$GRID){
                $b = $j+$o[$k][2];
                if (defined($s[$a][$b]) || $ok==1){
                    if ($ok==0){
                        if (($s[$i][$j]==0 && $s[$a][$b]==0
                            && $s[$a][$j]==0
                            && $s[$i][$b]==0)){
                                $ok=1;
                                $origi=$i;
                                $origj=$j;
                                $limith=$a-1;
                                $limitv=$b-1;
                            }
                    }
                    if ($ok==1 && $i>=$origi && $j>=$origj
                        && $i<=$limith && $j<=$limitv) {
                            if ($s[$i][$j]!=0){
                                destroy($s[$i][$j]);
                            }
                            $s[$i][$j]=$k;            # se asigna el objeto al elemento
                            $kontador++;
#                           last if ($kontador==$o[$k][4]);
                        }
                }
                if ($i==$HORIZ*$GRID && $s[$i][$j]==0) { $cc++; }
            }
        }
        if ($cc==$VERT*$GRID){ $HORIZ--; }
        if ($kontador != $o[$k][4]){
#       warn "Error de objeto - altura baja";
#       print "contador $kontador - tamaño $o[$k][4]\n";
            destroy($k);
        }
        if ($limitv-$origj+1 != $o[$k][2] &&
            $limitv-$origj+1 != $o[$k][1]){ warn "Error de w o l"; }
        if ($limith-$origi+1 != $o[$k][1] &&
            $limith-$origi+1 != $o[$k][2]){ warn "Error de l o w"; }
        $ok=0;
        for $i (1..$N){
            if ($o[$i][3]==0) { $ok=1; }
        }
        last if ($ok==0);  # si todos los objetos están asignados, acabar
    }

    $count=0;
    for $i (1 .. $HORIZ*$GRID){
        for $j (1 .. $VERT*$GRID){
            if ($s[$i][$j]!=0) { $count++; }
        }
    }
    $percent = 100*$count/($HORIZ*$GRID*$VERT*$GRID);
    if ($percent >= $MAXpercent){
        $MAXpercent = $percent;
        for $i (1..$HORIZ*$GRID){ # dibujar el resultado
            for $j (1..$VERT*$GRID){
                print chr(33+$s[$i][$j] % 90);  # ! = espacio libre
            }
            print "\n";
        }
        print "$HORIZ\n";    # decir la altura
        print "La ocupación es del $percent%\n";
    }
}
```

```
sub destroy {
    my $kk = @_;
    $o[$kk][3]=0;   # libera al objeto de debajo
    for my $ii (1..$HORIZ*$GRID){
       for my $jj (1..$VERT*$GRID){
           if ($s[$ii][$jj]==$kk) {
              $s[$ii][$jj]=0;  # desocupa el espacio
           }
       }
    }
}

__END__
```

6. Un ejemplo con 100 objetos

```
mmmmbbbbbbbppppNNNNNNxxxxxxxxxxx??????iiiiiiiEEEKKKKKK""""'!
mmmmbbbbbbbppppNNNNNNxxxxxxxxxxx??????iiiiiiiEEEKKKKKK""""'!
mmmmbbbbbbbppppNNNNNNxxxxxxxxxxx####!!iiiiiiiEEEKKKKKK""""'!
``!!bbbbbbbppppNNNNNNxxxxxxxxxxx####!!iiiiiii!!ooooooooo''!
``,,,,LLLL!ppppNNNNNNxxxxxxxxxxx####!!iiiiiii33ooooooooo''!
``,,,,LLLL))))!NNNNNNxxxxxxxxxxx####11111(((!33ooooooooo''!
``,,,,LLLL))))7777!!!xxxxxxxxxx####11111(((PP!ooooooooo''!
``,,,,LLLL))))7777ddddd>>>>>>>!!####11111(((PPkkkkkkkkkjj!!
``...!LLLL))))7777ddddd>>>>>>>!!####11111(((PPkkkkkkkkkjj!!
``...!LLLL))))7777ddddd>>>>>>>@@@@!!11111(((PPkkkkkkkkkjj!!
CC...C///!))))7777ddddd>>>>>>>@@@@!!11111(((PPkkkkkkkkk%%%!
CC...C///!))))7777ddddd>>>>>>>@@@@''''''!(((PPkkkkkkkkk%%%!
CC...C///aaTTTTTT:::::!!>>>>>>>@@@@''''''!(((PPkkkkkkkkk%%%!
CC...C///aaTTTTTT::::FFFJJJJ!!@@@@''''''hhhhh!kkkkkkkkkk44!!
CC...C///aaTTTTTT::::FFFJJJJ!!@@@@''''''hhhhhZZ88888888!!!!!
CC...CAAUU!TTTTTT::::FFFJJJJ!!@@@@''''''hhhhhZZ88888888!!!!!
&&&&&&AAUUuuuuuuulllllll999999YYYYYYYY$$!hhhhhZZ88888888!!!!!
&&&&&&AAUUuuuuuuulllllll999999YYYYYYYY$$!hhhhhZZ8888888886666!
&&&&&&AAUUuuuuuuulllllll999999YYYYYYYY$$qqqq!!----------6666!
&&&&&&AAUUggggg!!llllll222222eeeeeOOO$$OOOOO!----------6666!
+++++!AAUUggggg]]]]]]!!222222eeeeeOOO$$OOOOO!----------6666!
+++++!*********]]]]]]!!222222eeeeeOOO$$OOOOOwww!++++++++++!
+++++!*********]]]]]]!!222222eeeeeOOO$$OOOOOwww!++++++++++!
+++++!*********]]]]]]00000000000!!OOO$$OOOOOwww!++++++++++!
+++++!*********]]]]]]00000000000<<<<<vvvvvvvvvv!++++++++++!
+++++[[[[zzDDD!]]]]]]00000000000<<<<<vvvvvvvvvv!++++++++++!
+++++[[[[zzDDD55555!!00000000000<<<<<vvvvvvvvvv)))))))QQQQ!!
rrrr![[[[zzDDD55555!!00000000000###!!vvvvvvvvvv)))))))QQQQ!!
rrrr![[[[zzDDD55555======MMMMMMM###M!vvvvvvvvvv)))))))ccc$$!
rrrr![[[[zzDDD55555======MMMMMMM###M!!!HHHHHHHHfffff(((((($$!
rrrr;;;!!zzDDD55555======MMMMMMM###M__________!fffff(((((($$!
rrrr;;;ttttttttttt!======MMMMMMM###M__________sssssssssss&&!
rrrr;;;ttttttttttttRRRRR!!MMMMMMM###M__________sssssssssss&&!
rrrr!!!ttttttttttttRRRRR!!MMMMMMM###M^^^^^^^!!!sssssssssss!!!
WWWWW!!ttttttttttttRRRRR!!MMMMMMM###M^^^^^^^!%%%%%%%%%%%"""!!
WWWWW!!ttttttttttttRRRRRSSSSSSS!!###!^^^^^^^!%%%%%%%%%%%"""!!
WWWWW!!ttttttttttttRRRRRSSSSSSS!!###!^^^^^^^!%%%%%%%%%%%"""!!
WWWWW*******XXXXX!RRRRRSSSSSSS!!GGGGGGGGGGGBBBBBByy!!!!!!!!!
WWWWW*******XXXXX!!!!!!SSSSSSS!!GGGGGGGGGGGBBBBBByy!!!!!!!!!
!!!!!*******XXXXX!!!!!!SSSSSSS!!GGGGGGGGGGGBBBBBB!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!BBBBBB!!!!!!!!!!
41
La ocupación es del 89.024390243902439%
```

7. Conclusiones

La eficacia del algoritmo parece depender de la relación entre el numero de objetos, sus dimensiones medias y las del ancho de la tira de almacenaje. Así pues, la eficacia de ocupación puede variar de un 75% en problemas pequeños (número de objetos, 40) al 89% si el número de objetos es grande (100).
El código propuesto es multiplataforma y fácil de implementar (alrededor de 100 líneas de código), por lo que puede aportar fácilmente soluciones a bajo coste o tener un propósito didáctico.

Agradecimientos: Dr. Belarmino Adenso Díaz y Dr. David de la Fuente.

Referencias

Dowsland, K. (1993), Some experiments with simulated annealing techniques for packing problems, European Journal of Operational Research, vol. 68, pp. 389-99.

Fekete, S.P, Schepers, J. (1997),*On more dimensional packing III: Exact algorithms*, Technical paper ZPR97-290, Mathematisches Institut, Universität zu Köln.

Jackobs, S. (1996), *On genetic algorithms for the packing of polygons*, European Journal of Operational Research, vol. 88, pp. 165-81.

Lodi, A., Martello, S. y Monaci, M. (2002), *Two-dimensional packing problems: A survey*, European Journal of Operational Research, vol. 141, pp. 241-52.

Lodi, A., Martello, S. y Vigo, D. (1999), *Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems*, INFORMS Journal of Computing, vol. 11, pp. 345-57.

Martello, S., Monaci, M. y Vigo, D. (2000), *An exact approach to the strip packing problem*, Technical paper OR/00/18, Dipartamento di Elettronica, Informatica e Sistemica, Università di Bologna.