131069105 Heurísticas 12 de Enero 2009

RECTANGLE PACKING PROBLEM

ULL
Universidad de La Laguna

Moisés Domínguez García
Antonio José Medina Rivero
Beatriz A. Izquierdo Marrero
Johannes Matiasch

Índice

Bibliografía

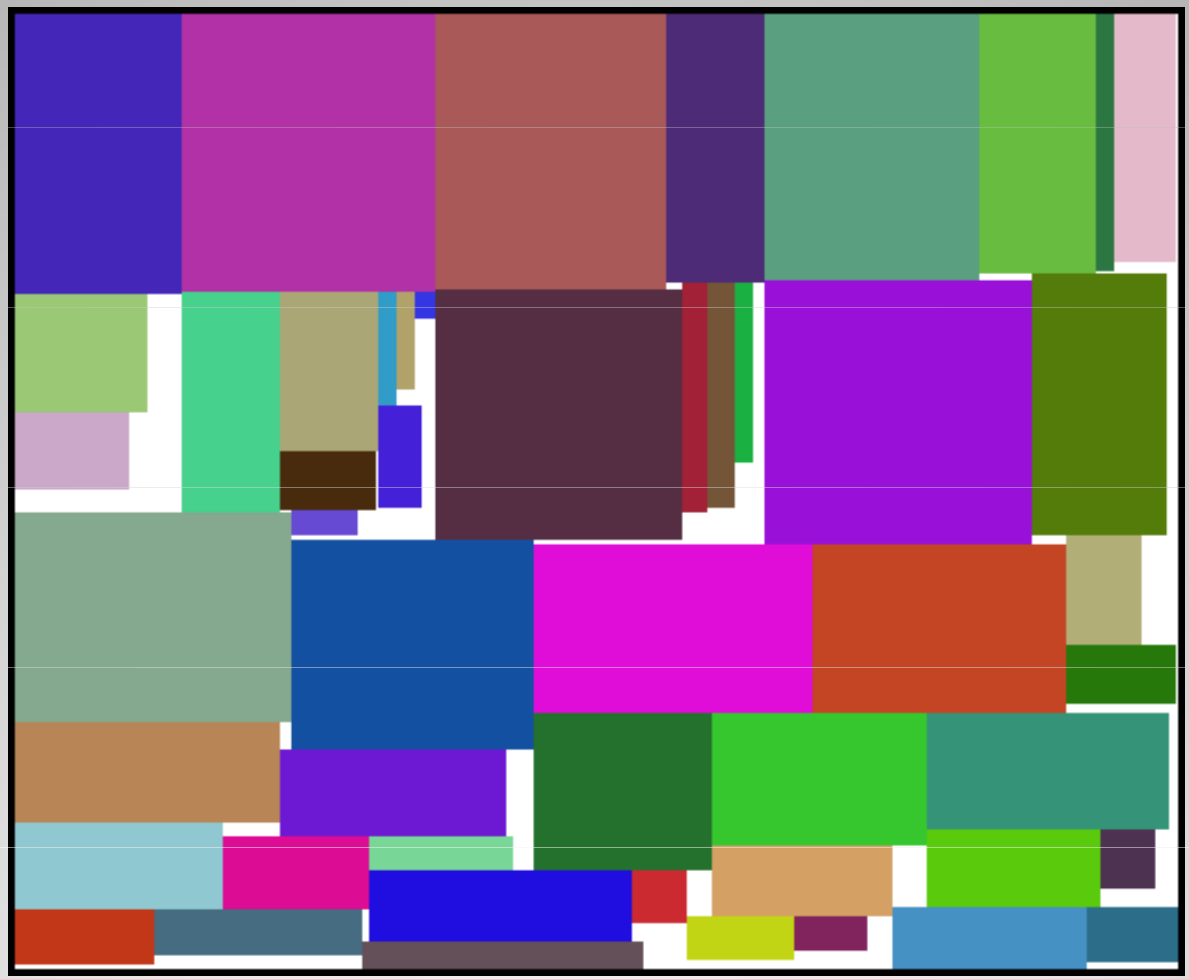
- Packing Problem (Wikipedia)
- Packing-based VLSI module placement using genetic algorithm with sequence-pair representation.
  - A. Drakidis, R.J. Mack, R.E. Massara.
  - IEE Proc-Circuits Devices Syst., Vol. 153 No. 6 (2006) pp.545-551
- Efficient Algorithms for 2-D Rectangle packing
  - http://www.csc.liv.ac.uk/~epa/index.html
- Cutting and Packing
  - Jakob Puchinger, Andreas M. Chwatal
  - Vienna University of Technology
  - http://www.ads.tuwien.ac.at/teaching/ss08/FortgeschritteneAD/folien/2dbp_slides.pdf

Rectangle Packing Problem

- **Descripción del problema:**

Dado un conjunto de n items bidimensionales (rectángulos),$\{i_1 , i_2 , . . . , i_n \}$, de dimensiones respectivas $(h(i_1), w(i_1))$, $(h(i_2), w(i_2))$, . . . , $(h(i_n), w(i_n))$, encontrar el rectángulo de menor área que los contiene. No se permiten rotaciones ni solapamientos de los items.

Ejemplo para la colocación de 50 rectángulos



Criterios de evaluación

-) Duración de la ejecución (en ms)

-) Porcentaje de Basura :

$$\textbf{basura (\%)} = ((\textbf{area}_{ut} - \textbf{area}_{min}) * 100) / \textbf{area}_{ut}$$

-) Perímetro :

p = 2a* 2b

width: 549px, height: 451px; circumference: 2000px
minimal area: 236358px, used area: 247599px
calculated cut/waste: 4.5400023%
duration of execution: 355ms

Si el porcentaje de basura es igual en 2 soluciones distintas, la solución más optima es la que se obtiene del menor perímetro.

Resolviendo el problema

Paso 1: Crear la entrada

-) Leer los rectángulos de una lista en el código.

-) Crear rectángulos con tamaños diferentes totalmente al azar.

-) Cortar un rectángulo de tamaño selecionado en piezas → sabemos la solución óptima!

Paso 2: Preparar la entrada (Ordenación de los rectángulos)

- Criterios de ordenación:

  -) Primero la **altura** de forma descendiente y luego la **anchura** de forma descendiente.

  -) Primero la **anchura** de forma descendiente y luego la **altura** de forma descendiente.

  -) Primero la **altura** de forma ascendente y luego la **anchura** de forma ascendente.

  -) Primero la **anchura** de forma ascendente y luego la **altura** de forma ascendente.

  -) Seleccionamos aleatoriamente según el **orden de creación**.

Paso 3: Fijar el ancho

-) Heurística de Mediana:

#mediana = $\lfloor n / Z^2 \rfloor$

Z ... valor heurístico
n ... #rectángulos

-) Heurística de Máximos:

#máximos = $\lfloor \log_5(n^Z) \rfloor$

El ancho fijado (**$ancho_f$**) es la suma de las #mediana / los #máximos.

-) Dependiendo de la mínima área posible:

$ancho_f = \lfloor \sqrt{(area_{min} * (1 + 1/\sqrt{n})} \rfloor$

Paso 4: Seleccionar la representación de soluciones

-) *BottomLeft* en combinación con el orden de la entrada

-) *BottomLeftImproved* en combinación con el orden de la entrada

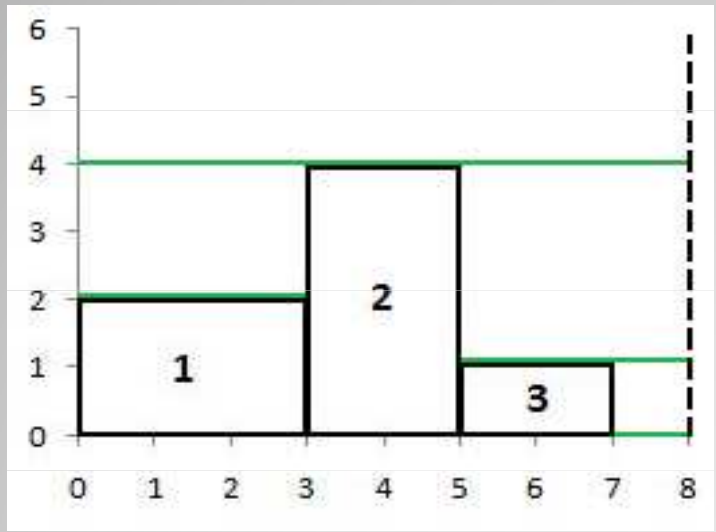
-) Idea y prueba (pero ya no funciona):
Bidirectional-Coding (valdría tambien como GRASP)

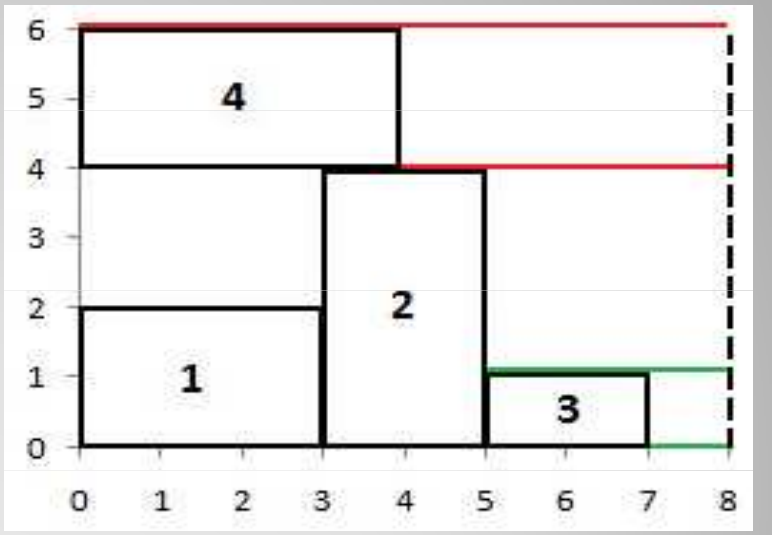
Paso 5: Conseguir la solución inicial mediante GRASP

- Tipos de GRASP:
  - *BottomLeft*
  - *BottomLeftImproved*
  - *BestFit*
  - *MultiGRASP* (!)

Implementación *BottomLeft*

- *freeSpace*: contiene todo el espacio libre

1) 

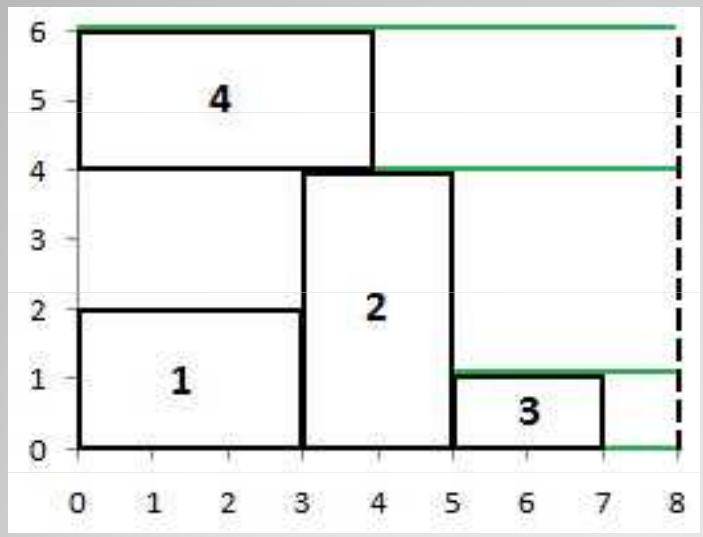
2) 

1) {<(7,0),1>, <(5,1),3>, <(0,2),3>, <(0,4),8>}
2) {<(7,0),1>, <(5,1),3>, <(4,4),4>, <(0,6),8>}

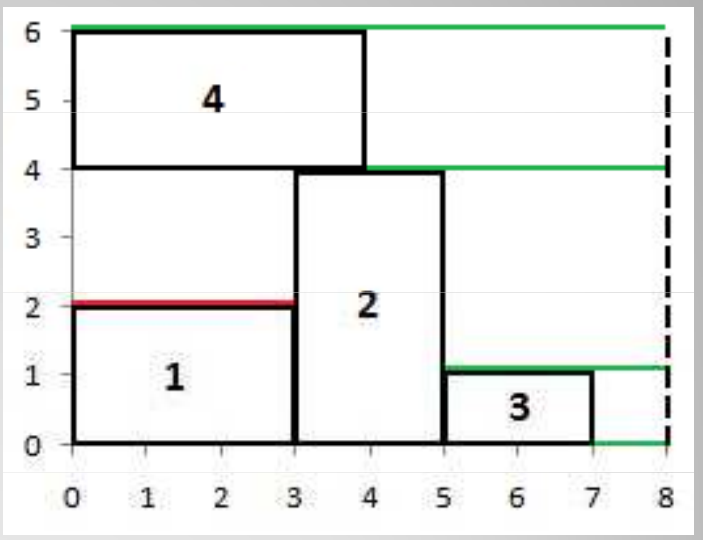
BottomLeft vs. *BottomLeftImproved*

- *freeSpace*: contiene todo el espacio libre

BL:



BLI:

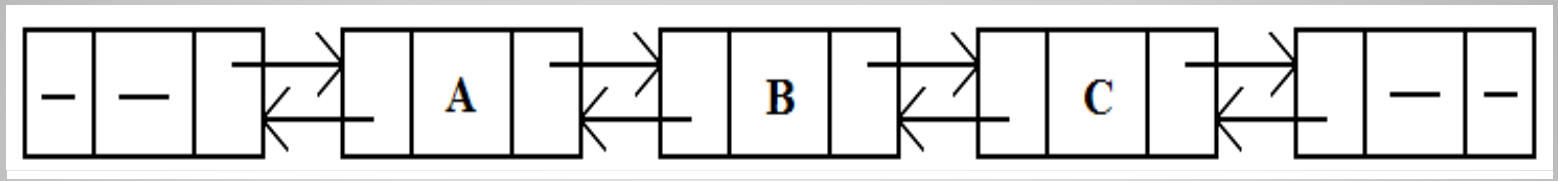


BL: {<(7,0),1>, <(5,1),3>, <(4,4),4>, <(0,6),8>}

BLI: {<(7,0),1,∞>, <(5,1),3,∞>, <(0,2),3,2>, <(4,4),4,∞>, <(0,6),8,∞>}

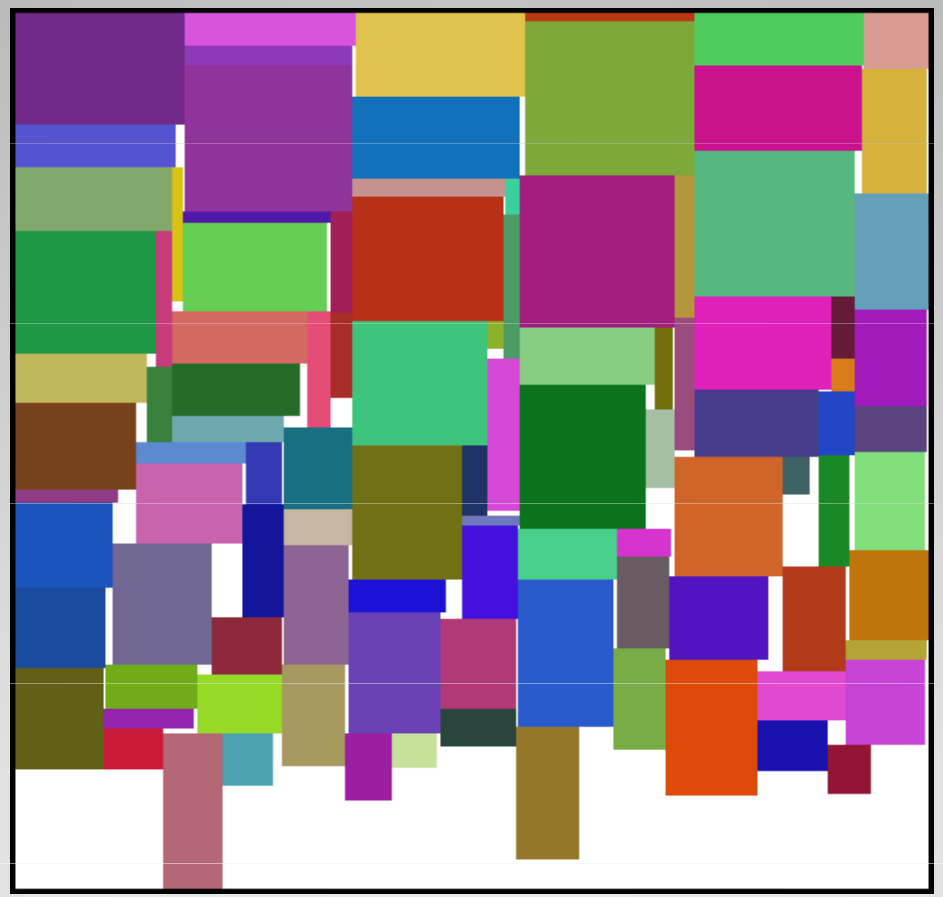
SortedDoubleLinkedList

- Estructura de datos para realizar el *freeSpace*
- DoubleLinkedList:



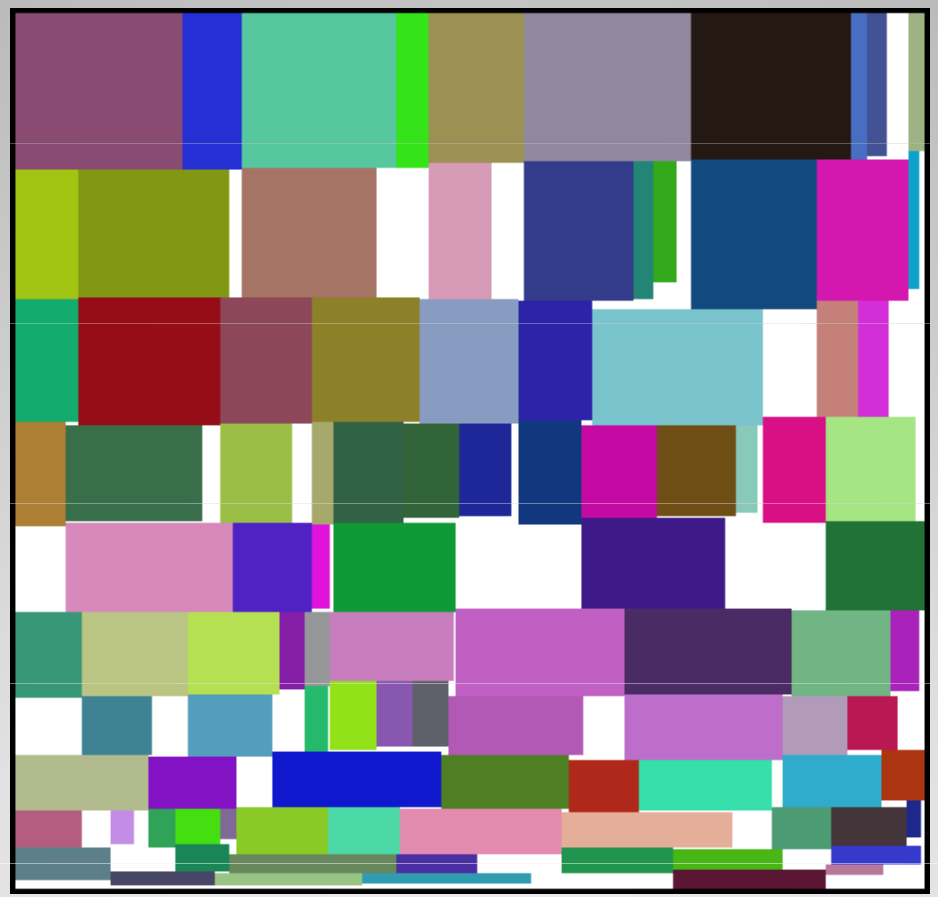
- Clave: punto (X,Y)
- Criterias de orden
  - "Y" : Y -> X (para *BottomLeft* y *BottomLeftImproved*)
  - "X" : X -> Y (para *BestFit*)

Ejemplo *BestFit* (n = 100)



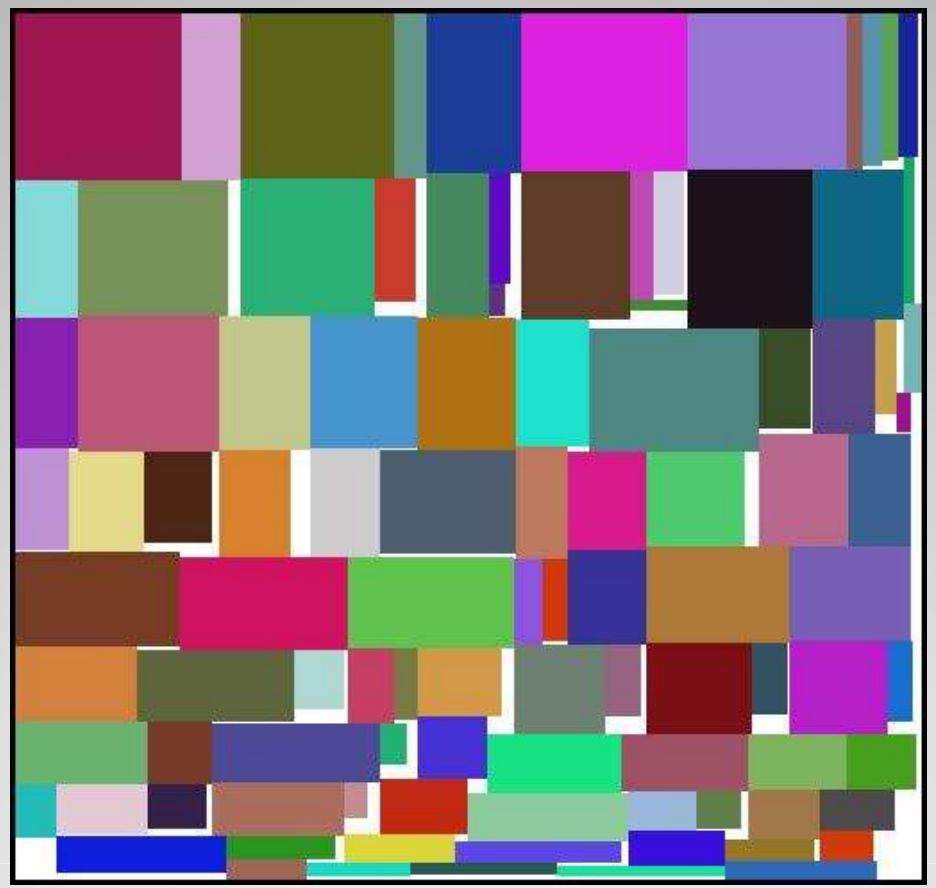
% basura: 14.382109%

Ejemplo *BottomLeft* (n = 100)



% basura: 11.593794%

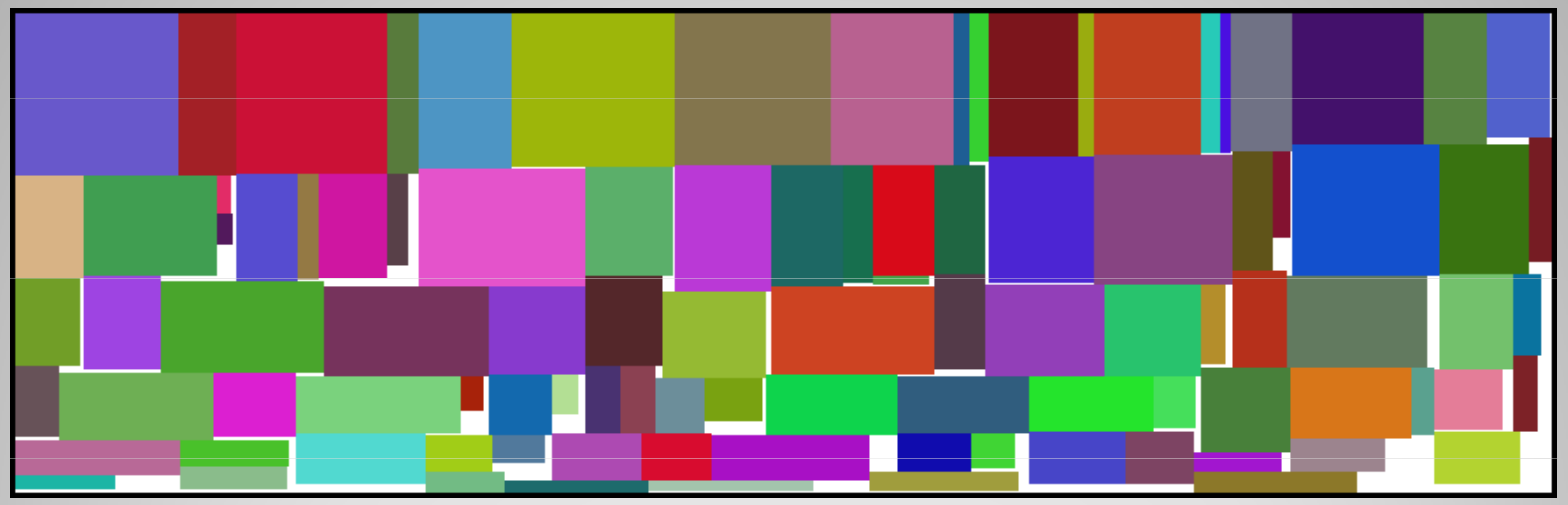
Ejemplo *BottomLeftImproved* (n = 100)



% basura: 4.8690095%

Ejemplo *MultiGRASP* / *BLImproved* (n = 100)

Idea: ejecutar un GRASP "normal" con anchos fijados diferentes
→ utiliza *BestFit*, *BottomLeft* o *BottomLeftImproved* como GRASP anidado



% basura: 4.3545365

Límite Inferior: el ancho mayor de todos los rectángulos
Límite Superior: Mínimo {$ancho_f * 3$; Σ ancho rectángulos}

Paso 6: Mejorar la solución inicial

- Búsquedas locales (todos con/sin entorno dínamico)
  - *AleatoricNeighborhood*
  - *HybridNeighborhood_All*
  - *HybridNeighborhood_End*
  - *DeterministicNeighborhood*
  - *VariableWidth*
- Búsqueda multiarranque (sólo después de MultiGRASP)
- Variable Neighborhood Search
- Recocido Simulado
- Lista de tabu para no probar el mismo intento otra vez

Búsqueda local: *AleatoricNeighborhood*

- Idea: intercambiar 2 (o más) rectángulos distintos en el orden; selecionarlos sólo aleatoriamente

- Ejemplo:

  [1, 2, 3, 4, 5, 6]
  [1, 2, 5, 4, 3, 6]
  [4, 2, 5, 1, 3, 6]
  [5, 2, 4, 1, 3, 6]
  ...

- Funciona bien con muy pocos rectángulos (se puede encontrar casi todos las permutaciones diferentes) o con muchos rectángulos.
- Funciona bien con *BottomLeft* y *BottomLeftImproved*.

Búsqueda local: *HybridNeighborhood_All*

- Idea: intercambiar 2 rectángulos distintos de orden; en la 1ª iteración coger la última posición (2ª iteración: penúltima, ...) e intercambiarlo con otro rectángulo en cualquiera posición aleatoriamente

- Ejemplo:

  [R3, R2, R0, R1]
  [R1, R2, R0, R3]
  [R0, R2, R1, R3]
  [R0, R3, R1, R2]
  [R3, R0, R1, R2]
  [R3, R2, R1, R0]
  ...

- Funciona bien con muy pocos rectángulos (se pueden encontrar casi todas las permutaciones diferentes) o con muchos rectángulos.
- Funciona bien con *BottomLeft* y *BottomLeftImproved*.

Búsqueda local: *HybridNeighborhood_End*

- Idea: Intercambiar 2 rectángulos distintos de orden; empieza al final del orden con (kStart+1)-rectángulos, para la próxima iteración intercambia increment-rectángulos más. Siempre intercambia estos rectángulos con otros rectángulos en cualquier posición aleatoriamente.

- Ejemplo (kStart=1, increment=1):

[R3, R1, R2, R0]	[R0, R2, R1, R3]
[R3, R0, R2, R1]	[R1, R2, R0, R3]
[R2, R0, R3, R1]	[R1, R3, R0, R2]
[R2, R0, R1, R3]	[R1, R3, R2, R0]
[R2, R0, R3, R1]	[R1, R2, R3, R0]
[R2, R3, R0, R1]	[R0, R2, R3, R1]
[R2, R1, R0, R3]	[R0, R1, R3, R2]
[R0, R1, R2, R3]	...

- Funciona bien con muchos rectángulos.
- Funciona bien con *BestFit*.

Búsqueda local: *DeterministicNeighborhood*

- Idea: intercambia 2 rectángulos distintos de orden; siempre utiliza el mismo esquema; la posición más a la izquierda se calcula con la fórmula siguiente:

  $$pos = \lfloor 1/2 + \sqrt{(1/4 + 2 * \#iterations)} \rfloor + 2$$

- Ejemplo (#iterations = 8):

  [R4, R3, R2, R1, R0, R5]
  [R4, R3, R2, R1, R5, R0]
  [R4, R3, R2, R0, R5, R1]
  [R4, R3, R1, R0, R5, R2]
  [R4, R2, R1, R0, R5, R3]
  [R4, R2, R1, R5, R0, R3]
  [R4, R2, R0, R5, R1, R3]
  [R4, R1, R0, R5, R2, R3]

- Funciona bien con muchos rectángulos.
- Funciona bien con *BestFit*.

Búsqueda local: *VariableWidth*

- Idea: probar diferentes anchos fijados alrededor del original simétricamente; por eso utiliza la función para calcular el ancho fijado para la iteración i:

  $\mathbf{ancho_f(i) = ancho_f(i-1) + i * -1^i}$

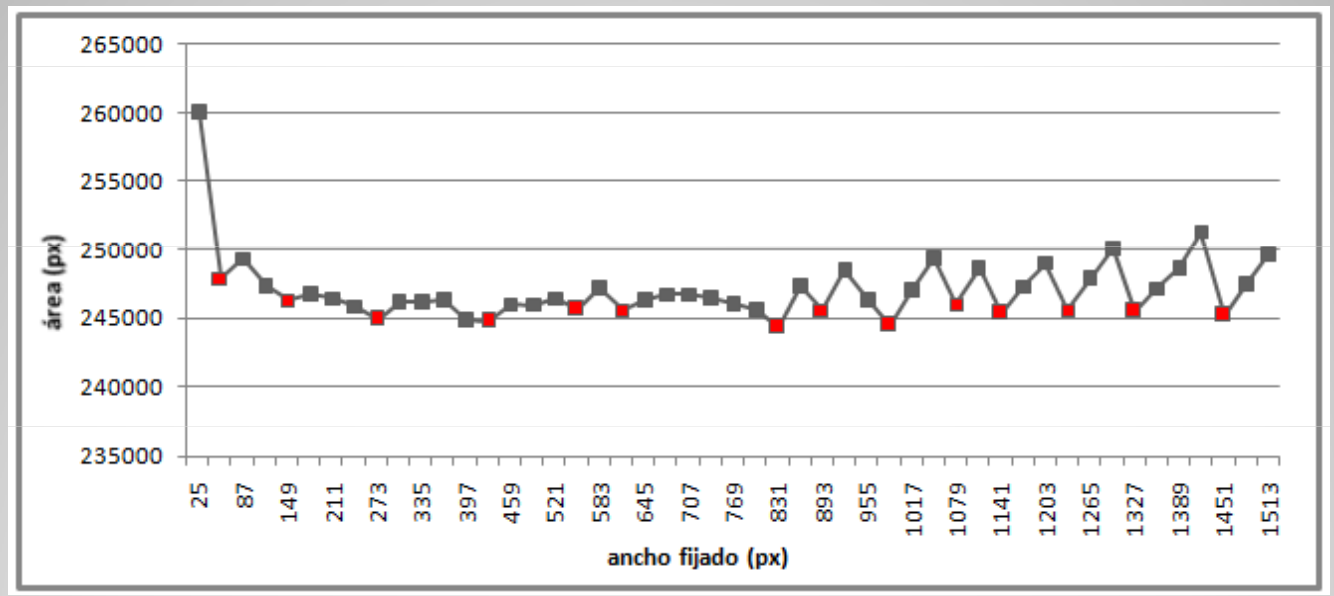
- Ejemplo ($ancho_f(0) = 10$):

  $ancho_f(1) = 9$
  $ancho_f(2) = 11$
  $ancho_f(3) = 8$
  $ancho_f(4) = 12$
  ...

- Funciona MUY bien con muchos rectángulos.
- Funciona bien con todos los GRASPs!

Búsqueda multiarranque

- Idea: después de ejecutar *MultiGRASP* ya tenemos un conjunto de soluciones donde podemos buscar mínimos (y máximos) con poco gasto → buscamos mínimos y probamos como mejorarlos con una búsqueda local



- Funciona MUY bien con muchos rectángulos.
- Funciona MUY bien con búsqueda local *VariableWidth*.

Variable Neighborhood Search

- Idea: Generar aleatoriamente un vecino x' del k-ésimo entorno de x ($x' \in N_k(x)$). Aplicar una búsqueda local desde x'; sea x'' el óptimo local obtenido. Si x'' es mejor que x, mover (x := x'') y continuar la búsqueda con N_1 (k := 1). En otro caso, hacer k := k + 1.

- Implementación: Intercambio de k rectángulos aleatoriamente (como *AleatoricNeighborhood*). Para la búsqueda local "anidada" (al menos teóricamente) valen todas las búsquedas locales anteriores.

- Funciona bien con muchos rectángulos.
- Funciona mejor con la búsqueda local *VariableWidth*.

Recocido Simulado

- Idea: Crear una solución $x' \in N_k(x)$ aleatoriamente. Si x' es mejor que $x \rightarrow x := x'$, la temperatura T no está modificada. Si x' es peor que x, tambien puede ser aceptado con probabilidad p_{accept} y T está reducido. Repetir hasta el criterio de parada (p.ej.: $T < T_{min}$).

- Implementación: Variar el ancho fijado en un entorno definido aleatoriamente. Utilizar *geometric cooling* con alfa elegible para reducir T.

  $$T_0 = area_{ut} - area_{min}$$

  $$p_{accept} = e^{-|area_{ut}(x') - area_{ut}(x)|/T}$$

- Funciona bien con muchos rectángulos y con poco gasto al mismo tiempo.

En total

BottomLeft	x 5 (ordenes diferentes)
BottomLeftImproved	x 5 (ordenes diferentes)
BestFit	x 1
	11
MultiGRASP	x 2
	22 GRASPs
Búsquedas Locales	x 5 (teóricamente también con/sin entorno dinámico!)
Búsqueda Multiarranque	x 5 (búsquedas locales "anidadas")
Variable Neighborhood Search	x 5 (búsquedas locales "anidadas")
Recocido Simulado	x 1
	16 heurísticas de mejora
Representacion de soluciones	x 2
= 22 x 16 x 2 =	**712 heurísticas combinadas!!!**

Evaluación

- Cada heurística está evaluada con cada número de rectángulos siempre al menos 3 veces → valor medio
- Cada heurística está evaluada con 3 números diferentes de rectángulos, es decir con 10, 100 y 2000 rectángulos.
- Siempre con los mismos ajustes.
- Equipo para la evaluación:
  - Intel Pentium Cual Core 1.73 GHz
  - 2 GB RAM
  - Windows Vista 32-Bit

GRASPs

GRASP	BottomLeft		BestFit		BottomLeftImproved		MultiGRASPVariableWidth (100)	
orden de input	sin orden	altura descendente	(en general) igual!		altura descendente	altura descendente	altura descendente	altura descendente
10 rectángulos	45,1368410	14,6212059	24,8290345	24,8290345	7,5601373	7,5601373	2,5601573	2,5601573
10 rectángulos	6	10	10	13	9	8	43	48
100 rectángulos	19,0082920	12,7520885	11,1969708	11,1969708	6,6650930	6,6650930	4,5487424	4,5487424
100 rectángulos	15	13	46	60	26	28	159	209
2000 rectángulos	7,8397121	1,6728481	4,9749134	4,9749134	0,7787905	0,7787905	0,2690285	0,2690285
2000 rectángulos	137	63	2648	2693	98	105	1554	1460
médio (basura)	**23,9949484**	**9,6820475**	**13,6669729**	**13,6669729**	**5,0013403**	**5,0013403**	**2,4593094**	**2,4593094**
médio (duración)	**52**	**29**	**901**	**922**	**44**	**47**	**585**	**572**
búsqueda local	Hybrid_All	Hybrid_All	Hybrid_End	Hybrid_End	Aleatoric	Deterministic	Deterministic	VariableWidth
# iter. (inicial)	100	100	100	100	100	100	50	50
representación	BottomLeft	BottomLeft	BottomLeft	BottomLeftImproved	BottomLeftImproved	BottomLeftImproved	BottomLeftImproved	BottomLeftImproved
10 rectángulos	21,4289732	10,7778346	20,2336598	20,2336598	6,8081340	6,8081340	2,4731045	2,5601573
10 rectángulos	42	37	41	31	28	21	55	67
100 rectángulos	13,3968367	10,9594223	11,1969708	9,9941283	6,5069407	6,6650930	4,5487424	4,2146393
100 rectángulos	120	81	121	372	245	237	244	321
2000 rectángulos	7,4491951	1,6096342	4,9749134	4,9749134	0,7138801	0,7787905	0,2690285	0,1959408
2000 rectángulos	3042	1414	4688	29072	7517	1506	2146	2398
médio (basura)	**14,0916683**	**7,7822970**	**12,1351814**	**11,7342339**	**4,6763183**	**4,7506725**	**2,4302918**	**2,3235791**
médio (duración)	**1068**	**511**	**1617**	**9825**	**2597**	**588**	**815**	**929**

# rectángulos	3x10, 3x100, 3x2000
heur. ancho inicial	área min. + offset
entorno variable	si
lista de tabu	si

BottomLeft vs. *BottomLeftDecreasingHeight*

	BottomLeft	
orden de input	sin orden	altura descendente
10 rectángulos	45,1368410	14,6212059
10 rectángulos	6	10
100 rectángulos	19,0082920	12,7520885
100 rectángulos	15	13
2000 rectángulos	7,8397121	1,6728481
2000 rectángulos	137	63
médio (basura)	**23,9949484**	**9,6820475**
médio (duración)	**52**	**29**
búsqueda local	Hybrid_All	Hybrid_All
# iter. (inicial)	100	100
representación	BottomLeft	BottomLeft
10 rectángulos	21,4289732	10,7778346
10 rectángulos	42	37
100 rectángulos	13,3968367	10,9594223
100 rectángulos	120	81
2000 rectángulos	7,4491951	1,6096342
2000 rectángulos	3042	1414
médio (basura)	**14,0916683**	**7,7822970**
médio (duración)	**1068**	**511**

Representación *BottomLeft* vs. *BottomLImproved*

	BestFit	
orden de input	(en general) igual!	
10 rectángulos	24,8290345	24,8290345
10 rectángulos	10	13
100 rectángulos	11,1969708	11,1969708
100 rectángulos	46	60
2000 rectángulos	4,9749134	4,9749134
2000 rectángulos	2648	2693
médio (basura)	**13,6669729**	**13,6669729**
médio (duración)	**901**	**922**
búsqueda local	Hybrid_End	Hybrid_End
# iter. (inicial)	100	100
representación	BottomLeft	BottomLeftImproved
10 rectángulos	20,2336598	20,2336598
10 rectángulos	41	31
100 rectángulos	11,1969708	9,9941283
100 rectángulos	121	372
2000 rectángulos	4,9749134	4,9749134
2000 rectángulos	4688	29072
médio (basura)	**12,1351814**	**11,7342339**
médio (duración)	**1617**	**9825**

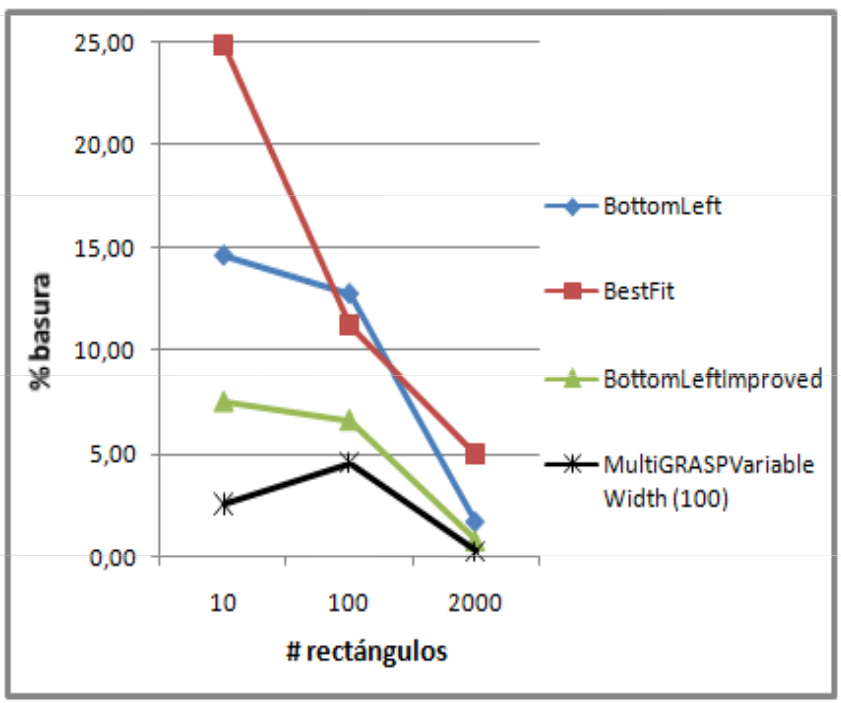
AleatoricNeighborh. vs. DeterministicNeighborh.

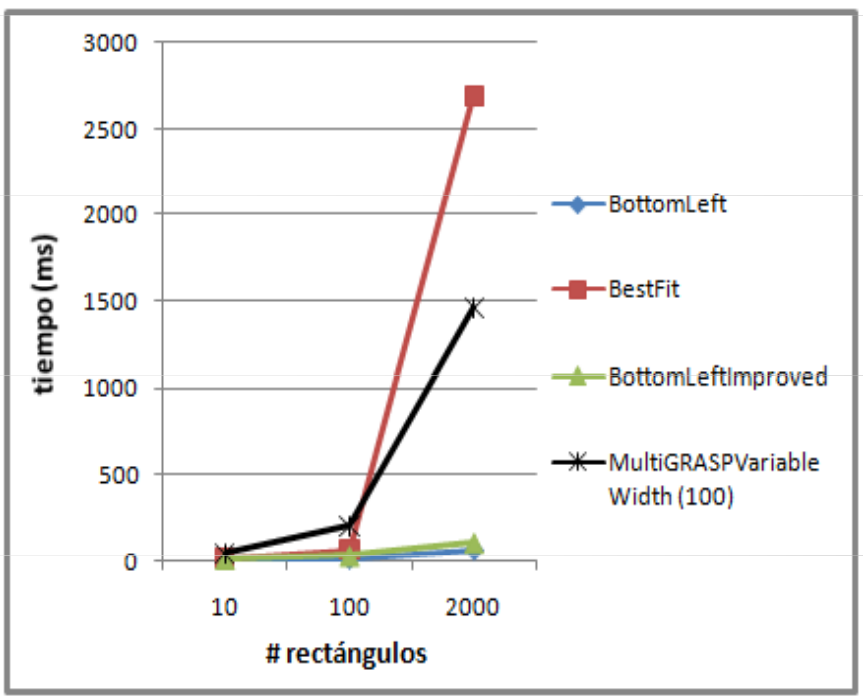
	BottomLeftImproved	
orden de input	altura descendente	altura descendente
10 rectángulos	7,5601373	7,5601373
10 rectángulos	9	8
100 rectángulos	6,6650930	6,6650930
100 rectángulos	26	28
2000 rectángulos	0,7787905	0,7787905
2000 rectángulos	98	105
médio (basura)	**5,0013403**	**5,0013403**
médio (duración)	**44**	**47**
búsqueda local	Aleatoric	Deterministic
# iter. (inicial)	100	100
representación	BottomLeftImproved	BottomLeftImproved
10 rectángulos	6,8081340	6,8081340
10 rectángulos	28	21
100 rectángulos	6,5069407	6,6650930
100 rectángulos	245	237
2000 rectángulos	0,7138801	0,7787905
2000 rectángulos	7517	1506
médio (basura)	**4,6763183**	**4,7506725**
médio (duración)	**2597**	**588**

DeterministicNeighborhood vs. VariableWidth

	MultiGRASPVariableWidth (100)	
orden de input	altura descendente	altura descendente
10 rectángulos	2,5601573	2,5601573
10 rectángulos	43	48
100 rectángulos	4,5487424	4,5487424
100 rectángulos	159	209
2000 rectángulos	0,2690285	0,2690285
2000 rectángulos	1554	1460
médio (basura)	**2,4593094**	**2,4593094**
médio (duración)	**585**	**572**
búsqueda local	Deterministic	VariableWidth
# iter. (inicial)	50	50
representación	BottomLeftImproved	BottomLeftImproved
10 rectángulos	2,4731045	2,5601573
10 rectángulos	55	67
100 rectángulos	4,5487424	4,2146393
100 rectángulos	244	321
2000 rectángulos	0,2690285	0,1959408
2000 rectángulos	2146	2398
médio (basura)	**2,4302918**	**2,3235791**
médio (duración)	**815**	**929**

GRASPs

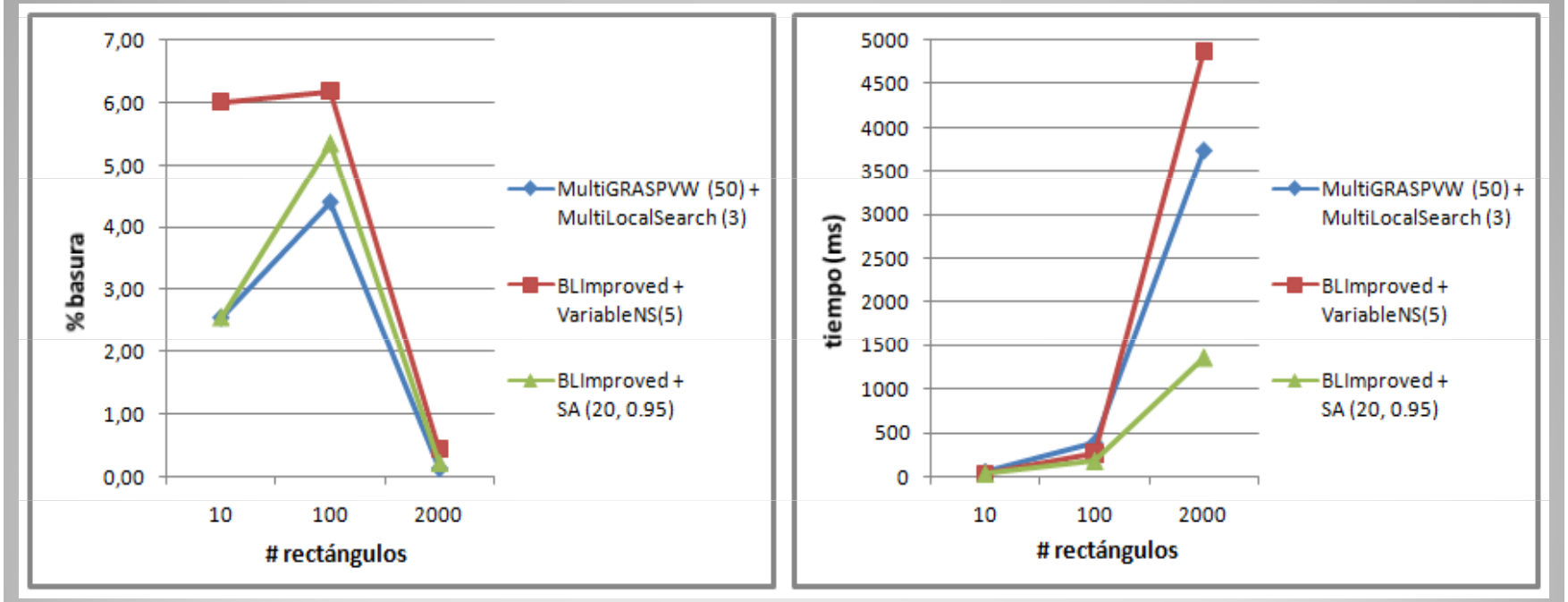

25,00
20,00
15,00
10,00
5,00
0,00
% basura
10
100
2000
rectángulos
BottomLeft
BestFit
BottomLeftImproved
MultiGRASPVariable Width (100)


3000
2500
2000
1500
1000
500
0
tiempo (ms)
10
100
2000
rectángulos
BottomLeft
BestFit
BottomLeftImproved
MultiGRASPVariable Width (100)

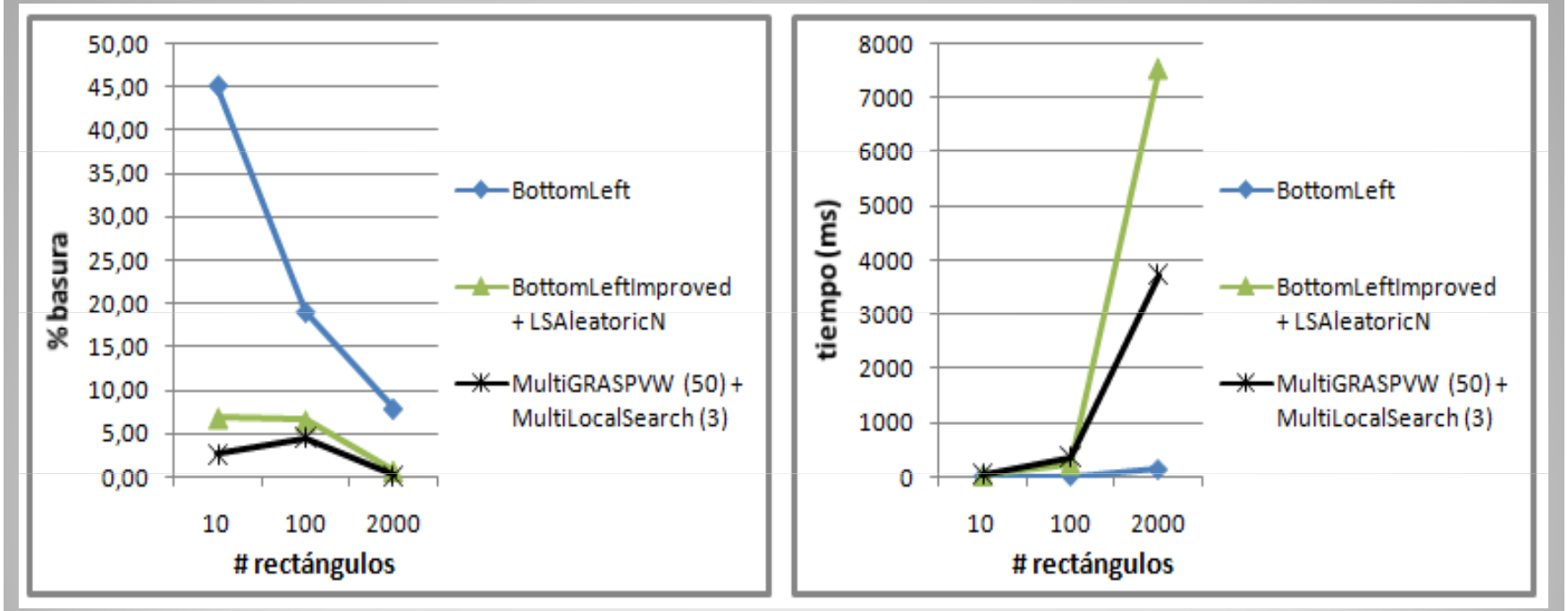
Heurísticas combinadas

	MultiGRASPVW (50) + MultiLocalSearch (3)		BLImproved + VariableNS(5)	BLImproved + SA (20, 0.95)
orden de input	altura ascendente	altura descendente	altura descendente	altura descendente
10 rectángulos	14,2753463	2,5601573	7,5601373	7,5601373
10 rectángulos	30	34	6	6
100 rectángulos	9,6735973	4,9920417	6,6650930	6,6650930
100 rectángulos	167	108	20	16
2000 rectángulos	3,0373732	0,2878534	0,7787905	0,7787905
2000 rectángulos	2910	912	98	100
médio (basura)	**8,9954389**	**2,6133508**	**5,0013403**	**5,0013403**
médio (duración)	**1036**	**351**	**41**	**40**
búsqueda local	VariableWidth	VariableWidth	VariableWidth	-
# iter. (inicial)	50	50	30	-
representación	BottomLeftImproved	BottomLeftImproved	BottomLeftImproved	BottomLeftImproved
10 rectángulos	14,2753463	2,5601573	6,0169000	2,5601573
10 rectángulos	64	55	37	31
100 rectángulos	8,2105318	4,4023247	6,1759024	5,3445806
100 rectángulos	535	386	265	177
2000 rectángulos	2,6482668	0,1432552	0,4339067	0,2237661
2000 rectángulos	14131	3732	4862	1352
médio (basura)	**8,3780483**	**2,3685791**	**4,2089030**	**2,7095014**
médio (duración)	**4910**	**1391**	**1721**	**520**

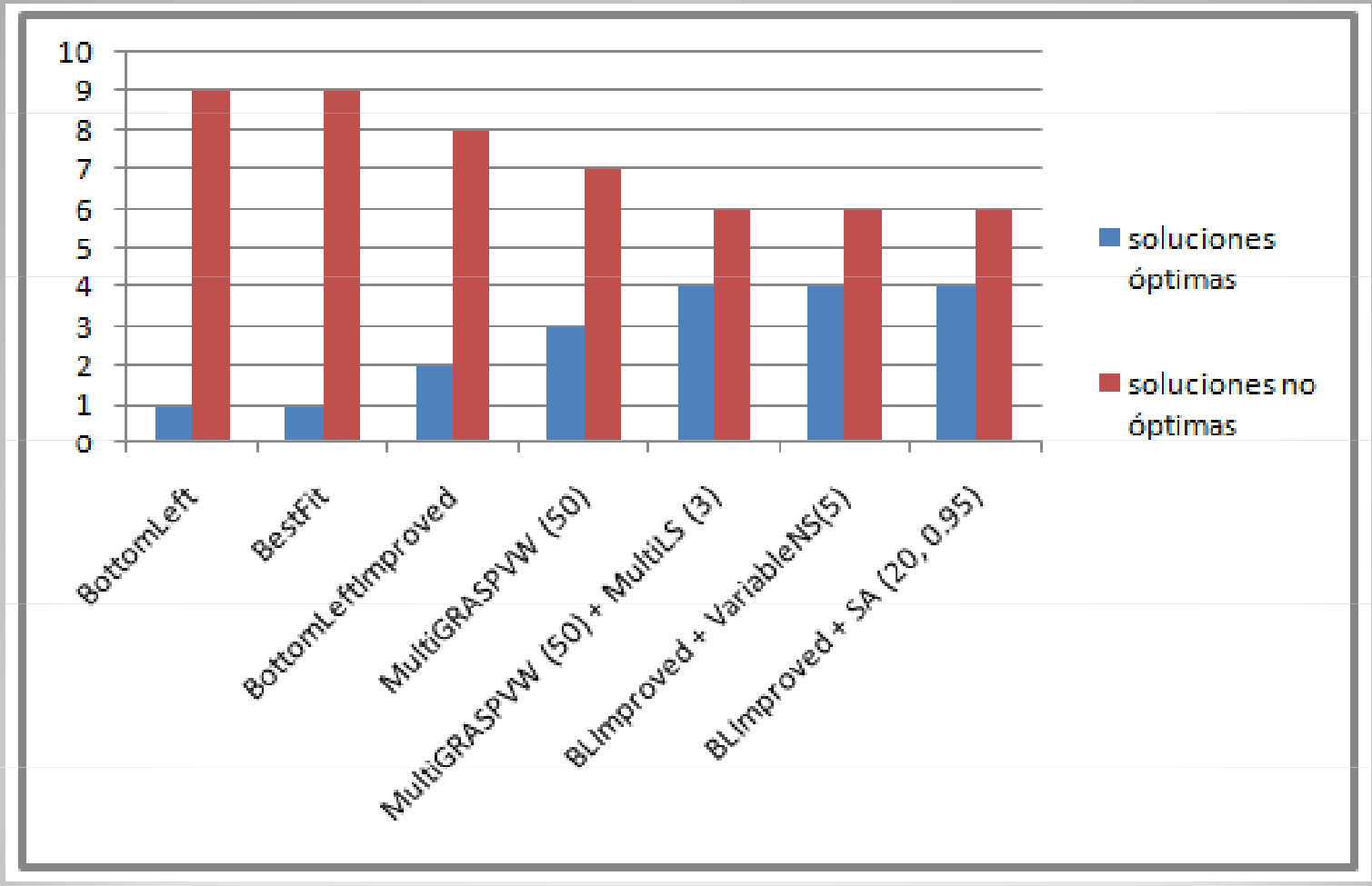
Heurísticas combinadas



Heurísticas con *BottomLeft(Improved)*



Solución optima encontrada (n=15)



Trabajo futuro

-) Implementar "threads" → paralelismo (para *MultiGRASP*, búsqueda multiarranque)

-) Probar/implementar nueva representación (p.ej. *Component-Graph*, *Bidireccional-Coding*, ...)

-) Identificar grupos de rectángulos "óptimos"

-) GRASP *LinePacking* (GRASP de dos fases) → solución guillotina!

-) Adaptar el GUI/nuevo GUI para strip-packing