Bin Packing Heuristics

G14

December 24, 2011

Outline

Back Ground Introduction

- computational complexity theory, the bin packing problem is a combinatorial NP-hard problem. In it, objects of different volumes must be packed into a finite number of bins of capacity V in a way that minimizes the number of bins used.
- There are many variations of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. They have many applications, such as filling up containers, loading trucks with weight capacity, creating file backup in removable media and technology mapping in Field-programmable gate array semiconductor chip design.

Back Ground Introduction

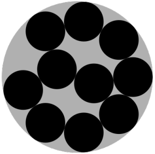
- The bin packing problem can also be seen as a special case of the cutting stock problem. When the number of bins is restricted to 1 and each item is characterised by both a volume and a value, the problem of maximising the value of items that can fit in the bin is known as the knapsack problem.
- Despite the fact that it is NP-hard, optimal solutions to very large instances can be produced with sophisticated algorithms. In addition, many heuristics have been developed.

One-dimensional algorithms

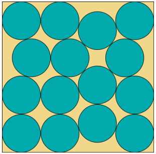
- Next Fit (NF)
- First Fit (FF)
- Best Fit (BF)
- Harmonic (HK) algorithm
- Next Fit-K (NFK)
- AFBK algorithm
- K-Bounded Best Fit (BBFK)
- First Fit Decreasing (FFD)
- Best Fit Decreasing (BFD)

Other packing problems

- Packing in 2-dimensional containers:
  - Circles in circle

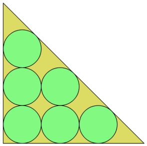


  - Circles in square

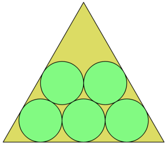


Other packing problems

- Packing in 2-dimensional containers:
  - Circles in isoscele right triangle（等腰直角三角形）

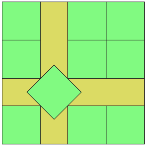


  - Circles in equilateral triangle（等边三角形）



Other packing problems

- Packing in 2-dimensional containers:
  - Squares in square



  - Squares in circle
    Pack n squares in the smallest possible circle.

Other packing problems

- Packing in 3-dimensional containers:
  - Spheres into a Euclidean ball
  - Spheres in a cuboid
- Packing of irregular objects

Main idea

- NF:
  - Time complexity: $O(n)$
- FF:
  - We use the most naive way, check the bins from the very beginning until find one bin fits.
  - Time complexity: $O(n^2)$
- BF:
  - We use a binary search tree to store the free capacity of bins, and find the bin to store new element from the tree. The full bin are not in the search tree.
  - Time complexity: $O(nlogn)$
- FFD:
  - FFD is the only off-line algorithm in these four. First sort the inputs, then apply FF to them.
  - Time complexity: $O(n^2)$

Test of different input size

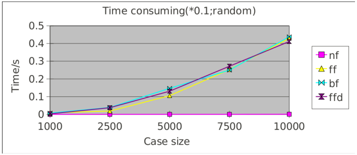
random part1: ***0.1**

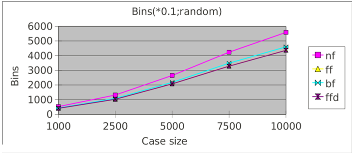
time consuming table

case size	1000	2500	5000	7500	10000
nf	0.0001128	0.0002254	0.0003594	0.0005255	0.0006819
ff	0.0094	0.0188	0.106	0.253	0.4295
bf	0.0075	0.039	0.147	0.25	0.438
ffd	0.003	0.0378	0.131	0.2718	0.4113

ans table

case size	1000	2500	5000	7500	10000
nf	525	1316	2647	4234	5589
ff	409	1023	2074	3286	4365
bf	430	1084	2178	3487	4598
ffd	409	1023	2074	3284	4364



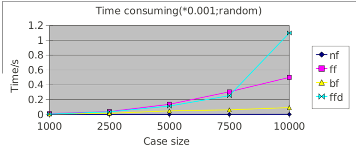


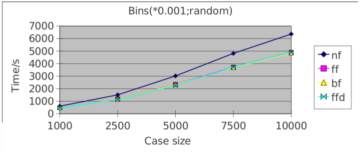
Test of different input size

random part2: ***0.001**

time consuming table					
case size	1000	2500	5000	7500	10000
nf	0.0001034	0.0002095	0.0003627	0.00051	0.0006643
ff	0.012	0.039	0.1374	0.306	0.5
bf	0.006	0.015	0.049	0.062	0.093
ffd	0.0062	0.035	0.1133	0.2535	1.099

ans table					
case size	1000	2500	5000	7500	10000
nf	603	1506	3013	4815	6362
ff	477	1185	2343	3717	4900
bf	471	1170	2319	3678	4840
ffd	476	1127	2261	3713	4899





Test of different ordered input

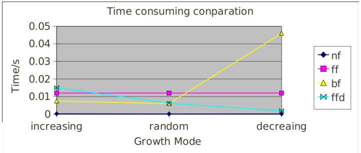
comparing part for different growth mode: *0.001;case size=1000

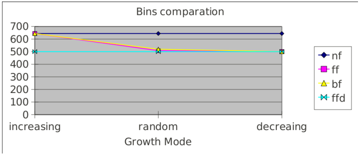
time consuming table

Growth Mo	increasing	random	decreaing
nf	0.0002031	0.0001034	0.000188
ff	0.012	0.012	0.012
bf	0.0075	0.006	0.046
ffd	0.015	0.0062	0.002

ans table

Growth Mo	increasing	random	decreaing
nf	644	644	644
ff	644	508	500
bf	644	519	500
ffd	500	500	500





First Fit in Cutting stock

In the first fit algorithm, the allocator keeps a list of free blocks (known as the free list) and, on receiving a request for memory, scans along the list for the first block that is large enough to satisfy the request. If the chosen block is significantly larger than that requested, then it is usually split, and the remainder added to the list as another free block.

The first fit algorithm performs reasonably well, as it ensures that allocations are quick. When recycling free blocks, there is a choice as to where to add the blocks to the free list – effectively in what order the free list is kept:

Best Fit in Cutting stock

Increasing size
This is equivalent to the best fit algorithm, in that the free block with the "tightest fit" is always chosen. The fit is usually sufficiently tight that the remainder of the block is unusably small.

Decreasing size

This is equivalent to the worst fit algorithm. The first block on the free list will always be large enough, if a large enough block is available. This approach encourages external fragmentation, but allocation is very fast.

- test the new plugin
- I think is very useful for me to use LaTeX
- The curent version is LaTeX 2_{ε}.