Strip Packing Based on Local Search and a Randomized Best-Fit

Bertrand Neveu and Gilles Trombettoni

CERTIS, University of Nice-Sophia, COPRIN Project, INRIA, 2004 route des Lucioles, 06902 Sophia.Antipolis cedex, B.P. 93, France
neveu@sophia.inria.fr, trombe@sophia.inria.fr

Abstract. We present an incomplete algorithm with no user-defined parameter for handling the strip-packing problem, a variant of the famous 2D bin-packing. The performance of our approach is due to several devices. We propose a move, based on the geometry of the layout, which is made incremental by maintaining the set of *maximal holes*. For escaping from local minima, the *Intensification Diversification Walk* (`ID Walk`) metaheuristic is used. `ID Walk` manages only one parameter that is automatically tuned by our tool.
We focus here on the greedy heuristics used to perform the moves and to compute the first layout before running the metaheuristic. In particular, we propose a variant of the well-known Best-fit (decreasing) (BF), called `RBF`, in which the criterion (i.e., height, width, perimeter, surface) changes every time a hole is selected. This simple way to randomize the most efficient greedy strategy is a key for obtaining good bounds while diversifying the layouts.
This paper provides an experimental evidence that a local search approach can be competitive with the best known incomplete algorithms.

1 The problem

Packing problems have numerous practical applications and the most interesting ones are all NP-hard, leading to the design of complete combinatorial algorithms, incomplete greedy heuristics, metaheuristics or genetic algorithms.

Packing problems consist in placing pieces in containers, such that the pieces do not intersect. Specific variants differ in the considered dimension (1D, 2D or 3D), in the type of pieces, or in additional constraints (e.g., for cutting applications, whether the (2D) container is guillotinable or not). The strip-packing is a variant of the 2D bin packing problem. A set of rectangles must be positioned in *one* container, called *strip*, which is a rectangular area. The strip has a fixed width dimension and a variable height. The goal is to place all the rectangles on the strip with no overlapping, using a minimum height of the container. We also study the variant that allows the rotation of rectangles with an angle of 90 degrees. The reader will refer to [15] and [9] for a description of incomplete methods handling strip-packing.

2 Our approach

We have designed an incomplete algorithm divided into two main phases:

1. a greedy heuristic produces a first layout of the rectangles on the strip,
2. a local search procedure, called `ID Walk`, driven by an automatic tuning procedure, repairs the first solution.

The efficiency of our method reaches that of state-of-the-art incomplete algorithms. To our knowledge, no other existing local search method has proven such a performance.

Incremental operations and maximal holes

To limit the combinatorial explosion, most algorithms maintain the *Bottom-Left (BL) property, that is, a layout where the bottom and left segments of every rectangle touch the container or another rectangle.* (Some algorithms maintain only a *bottom property* where only the bottom segment touchs. We refer by B[L] property both properties.) First, the B[L] property lowers the number of possible locations for rectangles. Second, it can be proven that any solution of a 2D packing problem can be transformed into a solution respecting the BL property with a simple repacking procedure. However, maintaining the B[L] property after a rectangle removal is not local to the removed rectangle and to its neighbors, but modifies the whole layout in the worst case (e.g., when the rectangle placed on the bottom-left corner of the container is removed).

We have proposed in [16] and [15] an alternative approach that does not respect the B[L] property but maintains instead the set of *maximal holes*.

Definition 1 *(Maximal hole) Let us consider a container C partially filled with a set S of rectangles. A maximal hole H (w.r.t. C and S) is an empty rectangular surface in C such that:*

- *H does not intersect any rectangle in S (i.e., H is a "hole" in the container),*
- *H is maximal, i.e., there is no hole H' such that H is included inside H'*[1].

Fig. 1 shows three examples with resp. 2, 2 and 4 maximal holes (from left to right). The maximal hole in grey corresponds to the most bottom-left one.

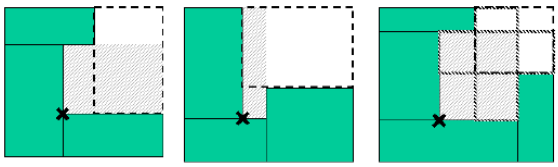

Fig. 1. Examples of maximal holes

It appears that the notion of maximal holes, designed by *maximal areas*, has been introduced independently by El Hayek et al. in [8] for 2D bin packing problems. They have proven that the maximum number of maximal areas managed during a packing procedure is $O(n^2)$, where n is the number of rectangles to be placed. We have detailed in [15] atomic operations that can be performed

[1] This property implies that the bottom and left segments of H touch the container or rectangles in S.

in constant time on rectangles and maximal holes. **Minus**(H, R) adds/modifies at most 4 maximal holes when one rectangle R is added in a container. A second operation **Plus**(H_1, H_2) holds between two rectangular maximal holes. If H_1 and H_2 intersect or are contiguous, `Plus` returns at most two new maximal holes.

Based on these operations, we have designed two procedures `AddRectangle` and `RemoveRectangle` for adding/removing one rectangle in/from a container [15]. They can be used in most algorithms handling 2D packing problems.

Incremental move for strip packing

To handle strip packing, our metaheuristic uses a move based on the "geometry" of the rectangles on the strip. This move makes an intensive use of the incremental `AddRectangle` and `RemoveRectangle` procedures. It removes one rectangle R on the top of the layout and places it inside the strip. More precisely, the new location for R is a maximal hole or a placed rectangle. The rectangles of the layout that intersect R in its new location are placed again on the strip with a greedy heuristic such as Best-Fit Decreasing (`BF`).

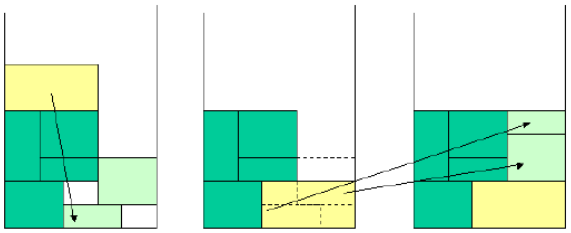

Fig. 2. One complete move

We report in Table 1 of Ref. [15] statistics that show that, in practice, during the local search, the number of possible locations for a rectangle grows in a linear way w.r.t the number of rectangles, and that the number of displaced rectangles in one move remains always very small on average.

Selected local search procedure

We have selected a fine two-dimensional objective function equal to $W \times H + n$, where W is the width of the strip, H is the ordinate of the top side of a highest rectangle on the strip *minus one*, and n is the number of units filled by rectangles on the highest line of the strip.

We have implemented our incomplete algorithm in the `INCOP C++` library developed by the first author (Ref. [14]). This algorithm could be specialized with number of greedy heuristics and metaheuristics. We have tried the main metaheuristics available in `INCOP` and have selected `ID Walk` (i.e., the `ID(best)` variant) for its efficiency and its simplicity [17]. In particular, the automatic tuning procedure provided by `INCOP` is more robust when it tunes only one parameter. A description of the automatic tuning procedure can be found in Refs. [17] and [15]. Note that the tuning time represents about 30% of the global time. `ID(best)` is a *candidate list strategy* that uses one parameter `MaxNeighbors` to perform one move from a configuration x to a configuration x', as follows:

1. `ID(best)` picks randomly neighbor candidates one by one and evaluates them. The *first* neighbor x' with a cost better than or equal to the cost of x is accepted.
2. If `MaxNeighbors` neighbors have been rejected, then the *best* neighbor among them (with a cost strictly worse than the cost of x) is selected.

`ID(best)` has a common behavior with a variant of tabu search once all the candidates have been rejected (item 2 above). However, the differences are the absence of tabu list and another policy for selecting a neighbor x' (step 1 above).

Finally, during the local search, a *repacking procedure* is launched occasionally to recover the BL property. It is run only when the latest move leaves the current solution at a significantly worse cost, i.e., an increase of 1 of the upper line of the layout. The repacking procedure is a variant of the famous Bottom-Left-Fill greedy heuristic [7]. It performs a simple loop on the rectangles in a bottom-left order and repacks them on the current layout. The idea behind our metaheuristic was to *not* recover the BL property every time a rectangle is removed during the moves performed by local search. Instead, the set of maximal holes is incrementally recomputed at every move and the repacking procedure is called so occasionally that the time spent is dominated by the local search. The impact of the repacking procedure on performance is positive but slight.

Overall, used with `ID Walk`, its automatically tuned parameter, and with the repacking procedure presented above, our metaheuristic has simply *no* user-defined parameter.

3 Greedy heuristics

Our latest advances concern the choice of greedy heuristics. It is used to compute a first layout. Also, this allows our move to put again displaced rectangles into the strip. The impact of the greedy heuristics on the efficiency is significant.

Our algorithm works with a simple version of the Best-Fit Decreasing greedy heuristic (BF) [6]:

The rectangles are initially sorted in decreasing order, according to one of the following criteria: largest Width first (w), largest Height first (h), largest Perimeter first (p), largest Surface first (s). A more original criterion, called largest Max first (m), selects the rectangle one dimension of which (width *or* height) is the largest among all the dimensions of the others. The criterion m stastistically implies to place first the rectangles that are more difficult to place. In other words, like the criterion p, this strategy differs the placement of the rectangles that are small in both dimensions and are thus easy to place.

Then, following a bottom-left order, `BF` tries to fill every hole. (A significant property is that only the holes on the top of the current layout - having an infinite height - must be considered.) At each iteration, the lists are traversed to place one of the rectangles on the strip, as follows:

- The list is traversed a first time to select the first rectangle (if any) that *fits* exactly the width of the hole or one of the two neighbor heights.
- If no such rectangle is selected, the list is traversed a second time to select the first rectangle (if any) with a width less than that of the hole.

If no rectangle can fit inside the hole, this hole will never been filled and the next hole is considered.

3.1 A simple randomized Best Fit

We introduce in this paper a Randomized variant of `BF` denoted by `RBF`. `RBF` accepts as parameter a list of sorting criteria. For instance, `RBF(w,p)` admits the two criteria *largest Width first* and *largest Perimeter first*. `RBF` follows the scheme of `BF` except that the chosen criterion changes from an iteration (filling a hole) to another, the criterion being picked randomly in the list. Although more studies must be performed to validate it, `RBF` is simple and intensifies well the search. Also, successive calls to `RBF` produce different layouts.

3.2 Initial layout

The first layout is simply obtained by successive calls to `BF` or `RBF`, following two possible strategies. The first one simply calls `RBF` iteratively until the time limit is reached.

The second strategy, designed here by `RR`, accepts as parameter a list G of greedy heuristics. The method applies iteratively one greedy heuristic picked in G (in a round robin - RR - manner) until the time limit is reached. The list G can include `BF` and `RBF` greedy heuristics together. In the experiments reported hereafter, the greedy heuristic designed by `RR10` is a round-robin with 5 different RBFs and 5 BFs.

3.3 Greedy heuristics used during the local search

Experiments (not reported here) have shown that several choices of greedy heuristics give similar results when they put again displaced rectangles into the strip. The reason is probably that only a small number of rectangles are displaced at every move. In our latest version, we use `RBF(p,s,w,h)`.

4 Experiments

We have performed experiments on five series enclosing 552 benchmarks. The 21 zero-waste instances by Hopper and Turton [10], the 13 *gcut* instances by Beasley [3], the 3 *cgcut* instances, the 10 *beng* instances by Bengtsson [4], and the 500 instances proposed by Martello and Vigo [13], Berkey and Wang [5].

The hybrid tabu/genetic algorithm is run by Iori during 300 seconds on a `Pentium III 800 Mhz` (Ref. [11]). The BLD* algorithm is run by Lesh et al on a `Pentium 2 GHz` (Ref. [12]). The two presented results correspond to time limits of respectively 60 seconds and 3600 seconds. The results presented for Burke et al. correspond to their BF heuristic enhanced with simulated annealing. They run their heuristic 10 times with a time limit of 60 seconds per run on a `Pentium IV 2 GHz`. Bortfeldt's genetic algorithm is run 10 times on every instance with an average time per run of 160 seconds on a `Pentium 2 GHz`. The GRASP algorithm (Ref. [1]) is the best known approach to handle strip packing. It is also run 10 times on every instance with a time limit of 60 seconds on `Pentium IV Mobile 2 GHz`. The hyperheuristic by Araya et al. [2] runs 10 times with a time limit of 100 seconds per run on a `Pentium IV 3 GHz`.

Our metaheuristic (`IDW`) is also run 10 times with a time limit of 100 seconds per trial on a `Pentium IV 3 GHz`: the first 10 seconds are used by the greedy heuristic `RR10` for computing the first layout while 90 seconds are spent in the local search. When greedy heuristics (`RR10`, `RR4`, `RBF`) are run alone, they perform 10 trials on a `Pentium IV 3 GHz`. Every trial runs the greedy heuristic iteratively until the time limit of 100 seconds is reached.

The cells in tables 1, 2, 4 report the average percentage deviation from optimum.

4.1 Synthesis

Iori's, Lesh's, Burke's and Bortfeld's algorithms are not competitive with the others. `HH` gives very good results but is generally slightly outperformed by GRASP. Our metaheuristic is competitive with the state-of-the-art GRASP. However, our approach is simpler and can handle the variant allowing the rotation of rectangles with an angle of 90 degrees. On this variant, it is difficult to evaluate our approach due to the lack of competitors. `IDW` seems to behave well while remaining behind Bortfeld's algorithm.

As a conclusion, most of the underlying concepts can be applied to several 2D packing problems.

Acknowledgments

Special thanks to Ignacio Araya and Maria-Cristina Riff for the collaboration on previous works about strip packing.

References

1. R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. Reactive grasp for the strip-packing problem. *Computers and Operations Research*, 35:1065–1083, 2008.
2. I. Araya, B. Neveu, and M-C. Riff. An efficient hyperheuristic for strip packing problems. In C. Cotta, M. Sevaux, and K. Sarensen, editors, *Adaptive and Multilevel Metaheuristics*, Studies on Computational Intelligence. Springer, to appear.
3. J.E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *J. of the operational research society*, 33:49–64, 1985.
4. B.E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The computer journal*, 25:353–357, 1982.
5. J.O. Berkey and P.Y. Wang. Two-dimensional finite bin packing algorithms. *J. of the oper. resear. society*, 38:423–429, 1987.
6. E. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock cutting problem. *Operations Research*, 52:697–707, 2004.
7. B. Chazelle. The bottom left bin packing heuristic: an efficient implementation. *IEEE Transactions on Computers*, 32:697–707, 1983.
8. J. El Hayek, A. Moukrim, and S. Negre. New Resolution Algorithm and Pretreatments for the Two-dimensional Bin-packing Problem. *Computers & Operations Research*, 35(10):3184–3201, 2008.
9. E. Hopper. *Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods*. PhD. Thesis Cardiff University, 2000.
10. E. Hopper and B.C.H. Turton. An empirical investigation on metaheuristic and heuristic algorithms for a 2d packing problem. *European J. of Operational Research*, 128:34–57, 2001.

11. M. Iori, S. Martello, and M. Monaci. *Metaheuristic algorithms for the strip packing problem*, pages 159–179. Kluwer Academic Publishers, 2003.
12. N. Lesh, J. Marks, A. Mc. Mahon, and M. Mitzenmacher. New heuristic and interactive approaches to 2D strip packing. *ACM J. of Experimental Algorithmics*, 10:1–18, 2005.
13. S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management science*, 15:310–319, 1998.
14. B. Neveu and G. Trombettoni. INCOP: An Open Library for INcomplete Combinatorial OPtimization. In *Proc. Constraint Programming, LNCS 2833*, pages 909–913, 2003.
15. B. Neveu, G. Trombettoni, and I. Araya. Incremental Move for Strip-Packing. In *Proc. ICTAI'07, int. conference on tools with articifcial intelligence, IEEE*, 2007.
16. B. Neveu, G. Trombettoni, and I. Araya. Recherche locale pour la découpe de rectangles. In *Actes du congrès ROADEF 2007*, pages 43–44, Grenoble, France, february 2007.
17. B. Neveu, G. Trombettoni, and F. Glover. ID Walk: A Candidate List Strategy with a Simple Diversification Device. In *Proc. Constraint Programming CP'04, LNCS 3258*, pages 423–437, 2004.

Table 1. Comparison on the zero-waste instances by Hopper and Turton. N is the number of rectangles and Opt. is the optimum.

Class	Width	N	Opt.	IDW	RR10	Iori	Lesh 3600	Burke	Bortfeldt	HH	GRASP
C1	20	1617	20	**0.00**	1.59	1.59	-	**0.00**	1.59	1.59	**0.00**
C2	40	25	15	**0.00**	**0.00**	2.08	-	6.25	2.08	**0.00**	**0.00**
C3	60	2829	30	2.15	2.15	2.15	-	3.23	3.23	2.15	**1.08**
C4	60	49	60	**1.09**	1.64	4.75	-	1.64	2.70	2.70	1.64
C5	60	73	90	**0.73**	1.10	3.92	2.17	1.46	1.46	1.10	1.10
C6	80	97	120	**0.83**	**0.83**	4.00	1.64	1.37	1.64	1.64	**0.83**
C7	160	196.3	240	**0.41**	0.69	-	-	1.77	1.23	1.10	1.23
# optimal solutions				**9/21**	6/21	5/18	0/6	3/21	4/21	6/21	8/21

Table 2. Comparison on Hopper and Turton's instances with non-fixed orientation of rectangles. The column HT report the results obtained by the authors themselves. GRASP cannot handle this variant of strip-packing.

Class	N	Opt.	IDW		RR10		HH		HT	Bortfeld
			best	aver.	best	aver.	best	aver.	best	best
C1	1617	20	**0.00**	**0.00**	**0.00**	**0.00**	1.67	2.50	4	1.70
C2	25	15	0.00	0.00	0.00	0.00	0.00	0.00	6	0.00
C3	2829	30	**2.22**	3.11	3.33	3.33	**2.22**	3.00	5	**2.22**
C4	49	60	1.11	1.61	1.67	1.67	1.67	2.50	3	**0.00**
C5	73	90	0.74	1.07	1.11	1.11	1.11	1.22	3	**0.00**
C6	97	120	0.83	0.83	0.83	0.83	0.83	1.56	3	**0.28**
C7	196-197	240	0.42	0.57	0.42	0.75	0.69	1.33	4	**0.28**
Overall			0.76		1.05		1.17		4	**0.64**

Table 3. Comparison on *beng*, *cgcut* and *gcut* instances. The benchmarks `gcut09'`...`gcut13'` (Ref. [1]) are variants of the benchmarks `gcut09`...`gcut13` in which the rectangles have been rotated with an angle of 90 degrees (i.e., the width and the height of every rectangle have been exchanged). The column LB yields a Lower Bound of the optimum (which is not necessarily reached). The following columns report the bound of the best solution computed by the corresponding algorithm.

Instance	Width	N	LB	IDW	RR10	Iori	Lesh 60	Lesh 3600	HH	GRASP
beng01	25	20	30	30	31	31	–	–	30	30
beng02	25	40	57	57	57	58	–	–	58	57
beng03	25	60	84	84	84	86	–	–	85	84
beng04	25	80	107	107	107	110	–	–	108	107
beng05	25	100	134	134	134	136	–	–	134	134
beng06	40	40	36	36	36	37	–	–	36	36
beng07	40	80	67	67	67	69	–	–	67	67
beng08	40	120	101	101	101	–	–	–	101	101
beng09	40	160	126	126	126	–	–	–	126	126
beng10	40	200	156	156	156	–	–	–	156	156
cgcut01	10	16	23	23	65	23	–	–	23	23
cgcut02	70	23	63	65	65	65	–	–	65	65
cgcut03	70	62	636	**658**	663	676	–	–	662	661
gcut01	250	10	1016	1016	1016	1016	1016	1016	1016	1016
gcut02	250	20	1133	**1187**	1204	1207	1211	1195	1196	1191
gcut03	250	30	1803	1803	1803	1803	1803	1803	1803	1803
gcut04	250	50	2934	3026	3025	3130	3072	3054	3019	**3002**
gcut05	500	10	1172	1273	1273	1273	1273	1273	1273	1273
gcut06	500	20	2514	2639	2637	2675	2682	2656	2644	**2627**
gcut07	500	30	4641	4701	4701	4758	4795	4754	4694	**4693**
gcut08	500	50	5703	5913	5935	6240	6181	6081	5922	**5908**
gcut09'	1000	10	2022	**2241**	2303	–	–	–	2290	2256
gcut10'	1000	20	5356	6399	6413	–	–	–	6402	**6393**
gcut11'	1000	30	6537	7736	7736	–	–	–	7736	7736
gcut12'	1000	50	12522	13184	13174	–	–	–	**13172**	**13172**
gcut13'	3000	32	4772	**5007**	5041	–	–	–	5028	5009

Table 4. Comparison on the 500 instances proposed by Martello, Vigo, Berkey, Wang. The cells include the percentage deviation between the (not necessarily reached) lower bound and a value v: v is an average value over the 50 best solution costs obtained for the 50 instances in a given class.

Class	Width	IDW	RR10	Iori	Lesh 60	Lesh 3600	Bortfeldt	HH	GRASP
01	10	**0.64**	0.77	**0.64**	0.81	0.68	0.75	0.72	**0.63**
02	30	**0.10**	0.35	1.78	1.12	0.42	0.88	0.34	**0.10**
03	40	1.82	2.02	3.05	2.71	2.23	2.52	**1.72**	**1.73**
04	100	**1.80**	2.03	5.08	4.41	3.54	3.19	2.60	2.02
05	100	2.15	2.24	3.15	2.85	2.43	2.59	**2.05**	**2.05**
06	300	3.41	3.72	5.99	6.45	5.13	4.96	3.80	**3.08**
07	100	1.19	1.27	1.16	1.17	1.12	1.19	1.13	**1.10**
08	100	3.77	4.03	6.16	5.99	4.93	3.85	3.74	**3.57**
09	100	0.07	0.13	0.07	0.07	0.07	0.07	0.07	0.07
10	100	2.91	3.13	4.67	4.11	3.48	3.05	2.93	**2.80**
Overall		1.79%	1.97%	3.17%	2.97%	2.40%	2.31%	1.90%	**1.73%**

Table 5. Comparison between our BF-based greedy heuristics on a sample of 62 benchmarks. All the results correspond to 10 trials of 100 seconds each, on a same computer `Pentium IV 3 GHz`. The instances named C201 and the following ones correspond to first subclasses of the 500 instances presented above. `RR4` is our round-robin greedy heuristics with the four standard BFs: `BF(h)`, `BF(w)`, `BF(s)`, `BF(p)`. `RBF` generally outperforms `RR4`, justifying the interest of our simple randomization. `RR10` is slightly better than `RBF`.

Instance	RR10		RR4		RBF		Instance	RR10		RR4		RBF	
	best	average	best	average	best	average	best	best	average	best	average	best	average
HT C1-P1	20	20	21	21	20	20	beng1	31	31	31	31	31	31
C1-P2	21	21	21	21	21	21	beng2	57	57.3	57	57.9	57	57.3
C1-P3	20	20	21	21	20	20	beng6	36	36.2	37	37	36	36.1
C2-P1	15	15	16	16	15	15	beng7	67	67	67	67	67	67.3
C2-P2	15	15.3	16	16	15	15.3	C201	22	22	23	23	22	22
C2-P3	15	15	15	15	15	15	C311	233	233.9	234	234	233	233.8
C3-P1	31	31	31	31	31	31	C341	727	728.9	727	729.4	726	729.9
C3-P2	31	31	31	31	31	31	C401	72	72.6	73	73	72	72
C3-P3	30	30	31	31	30	30	C411	92	92.5	93	93	92	92.2
C4-P1	61	61	61	61.4	61	61	C421	220	220.4	220	220.5	219	220.5
C4-P2	61	61	61	61.5	61	61	C431	246	246.7	246	246.5	246	246.7
C4-P3	61	61	61	61	61	61	C441	290	290.3	290	290.9	290	290.8
C5-P1	91	91	91	91	91	91	C511	744	746.1	754	754	744	745.8
C5-P2	91	91	91	91	91	91	C521	1864	1865	1864	1864.4	1874	1879.1
C5-P3	91	91	91	91	91	91	C541	2320	2330	2320	2328.8	2367	2374.4
C6-P1	121	121	121	121	121	121	C601	188	188	192	192	188	188
C6-P2	121	121	121	121	121	121	C611	240	241.7	242	242.2	240	241
C6-P3	121	121	121	121	121	121	C621	590	591.9	590	592.2	592	594.6
C7-P1	242	242	242	242.2	242	242.3	C631	654	658.7	657	658.4	657	660
C7-P2	241	241.7	241	242	241	241.9	C641	772	775.4	774	776.2	775	776.8
C7-P3	241	241.9	242	242	242	242							
cgcut03	663	663	667	670.8			C731	1997	1997	1997	1997	2064	2064
gcut2	1204	1204	1204	1204	1204	1204	C741	2664	2664	2664	2664	2708	2708
gcut4	3025	3039	3068	3068	3032	3052.9	C801	500	501.5	511	511	500	501.4
gcut5	1273	1273	1295	1295	1273	1273	C811	1014	1018.1	1018	1018.6	1008	1013.3
gcut6	2637	2643.9	2695	2695	2644	2644	C821	1517	1523.2	1510	1523.3	1505	1525.1
gcut7	4701	4712.1	4745	4745	4701	4701.8	C831	1898	1907.9	1883	1904	1903	1912.9
gcut8	5935	5950.9	5940	5942.2	6010	6042.4	C841	2333	2343.5	2326	2338.3	2352	2361.6
gcut9'	2303	2303	2303	2303	2303	2303	C911	1915	1915	1915	1915	1940	1940
gcut10	6413	6413	6434	6434	6393	6393	C1011	742	742.6	749	749	743	743.9
gcut12'	13174	13197.9	13312	13323.4	13404	13495.8	C1021	1064	1072.2	1066	1071.5	1071	1074
gcut13'	5041	5058.9	5077	5084.4	4996	5031.1							