A Parallel Algorithm for the Two-Dimensional Cutting Stock Problem

Luis García, Coromoto León, Gara Miranda, Casiano Rodríguez

Dpto. Estadística, I. O. y Computación
Universidad de La Laguna
E-38271 La Laguna, Tenerife, Spain
{lgforte|cleon|gmiranda|casiano}@ull.es

Abstract. Cutting Stock Problems (CSP) arise in many production industries where large stock sheets must be cut into smaller pieces. We present a parallel algorithm - based on Viswanathan and Bagchi algorithm (VB) - solving the Two-Dimensional Cutting Stock Problem (2DCSP). The algorithm guarantees the processing of best nodes first and does not introduce any redundant combinations - others than the already present in the sequential version. The improvement is orthogonal to any other sequential improvements. Computational results of an OpenMP implementation confirm the optimality of the algorithm. We also produce a new syntactic based reformulation of the 2DCSP problem which leads to a concise representation of the solutions. A highly efficient data structure to store subproblems is introduced.

1 Introduction

Cutting Stock Problems (CSP) arise in many production industries where large stock sheets (glass, textiles, pulp and paper, steel, etc.) must be cut into smaller pieces. CSP can be classified [1, 2] attending to several characteristics: the number of dimensions (1D, 2D, 3D), the number of available surfaces and patterns, the shape of the patterns (regular or irregular), the orientation, etc.

The Constrained 2 Dimensional Cutting Stock Problem (2DCSP) is one of the most interesting variants of CSP and targets the cutting of a large rectangle S of dimensions $L \times W$ in a set of smaller rectangles using orthogonal guillotine cuts. That means that any cut must run from one side of the rectangle to the other end and be parallel to the other two edges. The produced rectangles must belong to one of a given set of rectangle types $\mathcal{D} = \{T_1 \dots T_n\}$ where the i-th type T_i has dimensions $l_i \times w_i$. Associated with each type T_i there is a profit p_i and a demand constraint b_i. The goal is to find a feasible cutting pattern with x_i pieces of type T_i maximizing the total profit:

$$\max \sum_{i=1}^{n} x_i p_i \text{ subject to } x_i \leq b_i \text{ and } x_i \in \mathbb{N}$$

Though a large number of heuristics have been proposed [3–6], the number of exact algorithms is not so extensive. The optimal algorithms fall in two

categories: depth-first searches [7] and best-first search methods [8–10]. To our knowledge, not many parallel exact algorithms have been devised [11, 12].

Wang [13] was the first to make the observation that all guillotine cutting patterns can be obtained by means of horizontal and vertical builds of pieces (Figure 1). Her idea was exploited by Viswanathan and Bagchi [8] to propose a brilliant best-first search algorithm (VB) which uses Gilmore and Gomory [14] dynamic programming solution - for the unbounded version of the problem - to build an upper bound. The algorithm resembles A* algorithms and uses two lists OPEN and CLIST to yield the set of feasible solutions. At each step, the best build of pieces (or meta-rectangle) from OPEN is chosen and combined with the already found best meta-rectangles (elements in CLIST) to produce horizontal and vertical builds. Later, Hifi [9] and Cung, Hifi and Le-Cun [10] proposed a modified version of VB algorithm (called MVB) introducing an initial lower bound and rules to find in constant/time duplicated/dominated patterns. The efficiency of MVB is also a consequence of other two novelties: CLIST is represented by a bidimensional data structure and VB upper bound is reduced combining it with the solution of a One-Dimensional Knapsack Problem. The Knapsack Problem results from mapping the 2DCSP bidimensional constraints onto one dimensional area constraints (similar proposals were made by Tschöeke and Holthöfer in [11]).

Niklas et al. in [12] proposed a parallel version of Wang's approximation algorithm [13]. Unfortunately, Wang's method does not always yield optimal solutions in a single invocation and is slower than VB algorithm [8]. Tschöeke and Holthöfer parallel version [11] starts from the original VB algorithm and uses the Paderborn Parallel Branch-and-Bound Library (PPBB-LIB [15]). Due to the asynchronous nature provided by the PPBB-LIB skeleton, the algorithm does not guarantee the processing of best subproblems first. Another consequence is the generation of unwanted duplicates which aren't produced by the sequential version. In the worst case an exponential growth of elements may result. The authors proposed a stamp-based mechanism to hinder the generation of duplicates.

Though the next section is devoted to introduce VB algorithm, it contains some contributions. Namely, it emphasizes a new syntactic based reformulation of the problem and proposes a concise representation of the solutions. More important, a highly efficient data structure to store subproblems is introduced. In this section we also present an improvement to avoid unwanted repetitions of computations which are common to the two parallel loops. The parallel algorithm is presented in section 3. On each step the best subproblem from OPEN is all-to-all reduced and combined with each of the elements in CLIST. The design of the parallel algorithm was suggested by the bidimensional data structure proposed by Cung and others [10, 11] to hold VB CLIST. The bidimensional structure leads to two traversing loops which can be parallelized. The performance gain is compatible with any other sequential improvements. Some computational results are shown in section 4. Finally, the conclusions and future works are given in section 5.

2 A Sequential Algorithm

Reformulating the 2DCSP Given two meta-rectangles α and β of dimensions (α^l, α^w) and (β^l, β^w) the vertical build $\alpha|\beta$ is a meta-rectangle of dimensions $(\max\{\alpha^l, \beta^l\}, \alpha^w + \beta^w)$. The horizontal build $\alpha - \beta$ is a meta-rectangle of dimensions $(\alpha^l + \beta^l, \max\{\alpha^w, \beta^w\})$. Using this idea, a feasible solution can be represented by a formula like $(T_2|T_3)|(T_1 - T_2)$. Even better, we may use postfix expressions avoiding the need for parenthesis, i.e. $T_2T_3|T_1T_2 - |$, and leading to a compact representation of the syntax tree (see Figure 1).

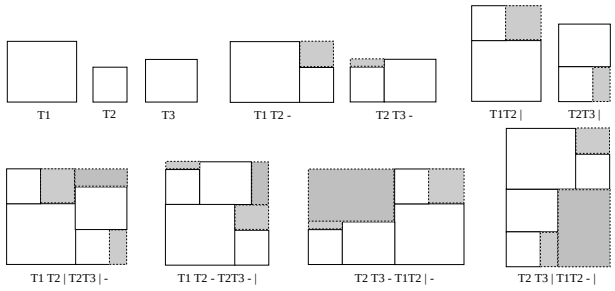

Fig. 1. Examples of vertical and horizontal builds, shaded areas represent waste

Figure 2 presents a context free grammar G and the associated semantic rules defining the attributes value (g), length (l), width (w) and number of used patterns (i) associated with meta-rectangles $\alpha \in L(G)$. Given the aforementioned syntax directed definition the 2DCSP can be reformulated as:
$\max\{\alpha^g$ such that $\alpha \in L(G)$, $\alpha^l \leq L$, $\alpha^w \leq W$ and $\alpha^i \leq b_i$ for any pattern $i\}$
Being $L(G)$ the language generated by the grammar G. Observe how semantic geometrical properties can be embeded into the syntactic structure easing the expression of constant-time dominance rules as described in [10]. Moreover, this notation makes possible to easily build new patterns compositions and to manage and represent the problem (partial) solutions.

Syntax	Semantic Rules
$S \rightarrow S_1 S_2 \|$	$S^g = S_1^g + S_2^g$ $S^l = \max\{S_1^l, S_2^l\}$; $S^w = S_1^w + S_2^w$ $S^i = S_1^i + S_2^i$
$S \rightarrow S_1 S_2 -$	$S^g = S_1^g + S_2^g$ $S^l = S_1^l + S_2^l$; $S^w = \max\{S_1^w, S_2^w\}$ $S^i = S_1^i + S_2^i$
$S \rightarrow T_i$ for each $T_i \in \mathcal{D}$	$S^g = c_i$; $S^l = l_i$; $S^w = w_i$; $S^i = S^i + 1$

Fig. 2. Syntax Directed Definition for the 2DCSTP. S^i is initialized to 0

```
1  OPEN := {T1, T2, ..., Tn}; CLIST := ∅; f' := UpperBound();
2  BestSol := Heuristic(); B: = BestSol^g;
3  repeat
4     choose α meta-rectangle from OPEN with higher f' value;
5     return(Bestsol) if B = f'(α);
6     insert α in CLIST at entry (α^l, α^w);
7     for x := 0 to L - α^l do {
8        forall β ∈ CLIST_x such that β^i ≤ b_i − α^i do {
9          γ = αβ−; γ^l = α^l + β^l; γ^w = max(α^w, β^w); /* horizontal build */
10         γ^g = α^g + β^g; γ^i = α^i + β^i ∀i;
11         if (γ^g > B) then { free OPEN from B to γ^g; B = γ^g; BestSol = γ; }
12         if (f'(γ) > B) then {insert γ in OPEN at entry f'(γ); }
13       }
14    }
15    for y := 0 to W - α^w do {
16       forall β ∈ CLIST_y such that β^i ≤ b_i − α^i do {
17         γ = αβ|; γ^l = max(α^l, β^l); γ^w = α^w + β^w; /* vertical build */
18         γ^g = α^g + β^g; γ^i = α^i + β^i ∀i;
19         if (γ^g > B) then { free OPEN from B to γ^g); B = γ^g; BestSol = γ; }
20         if (f'(γ) > B) then { insert γ in OPEN at entry f'(γ); }
21       }
22    }
23    return(Bestsol) if OPEN = ∅;
24 forever;
```

Fig. 3. Modified Version of Viswanathan and Bagchi Algorithm (MVB)

The Modified Viswanathan and Bagchi Algorithm In VB original version the combination is achieved traversing the whole CLIST, discarding non feasible solutions. To alleviate this, Cun and others [10] introduced the skillful data structure depicted in Figure 4. This way using two loops (see Figure 3), one for the horizontal combinations (lines 7-14) and another for the vertical combinations (lines 15-22) only problems holding the geometry constraints are visited. There is one loss however. Observe that $(\alpha\beta-)^i = (\alpha\beta|)^i$ and $(\alpha\beta-)^g = (\alpha\beta|)^g$ for any α and β and any pattern i. When using the original list data structure these common values are computed only once. The decoupling on two loops implies the repetition of such calculus (lines 10 and 18). We can reduce this overhead as follows: During the horizontal loop (lines 7-14) we save a pointer to α inside the data structure representing the meta-rectangle β (for that we use an extra field, let us call it *current*). On a second field (call it *horizontal*) we store a pointer to $\alpha\beta-$. Now if during the vertical loop (lines 15-22) the meta-rectangle β (line 16) has its *current* field pointing to α we can recover the values $\alpha^i + \beta^i$ stored in $\alpha\beta-$ using the *horizontal* field of β.

Data Structure to Represent the Upper Bounds On any best-first search algorithm, subproblems are sorted by the value of their upper bounds. Maintaining this usually very large set is often cause for performance degradation.

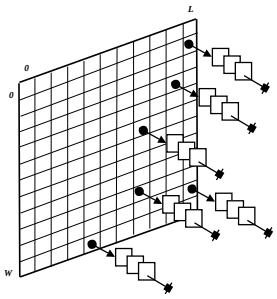

Fig. 4. Data Structure to store CLIST

Since along the execution of any branch-and-bound the lower bounds keep ascending and the upper bounds descending we can state that during the search all the upper bounds fall in the interval $[best_0, upper_0]$. We denote by $best_0$ the initial heuristic value and by $upper_0$ the upper bound of the initial problem. That suggest a natural solution: to have an array $[best_0 \dots upper_0]$ of pointers to linked lists of subproblems. Subproblems with the same upper bound go to the same linked list. Insertion then can be done in constant time. Notice that insertion using the classical list approach [8, 13] leads - for the VB algorithm but it is also the general case for branch-and-bound - to $\mathcal{O}(2^n)$ time since in the worst case the list grows exponentially with the number of patterns. The other main operation involved, choosing/extracting the subproblem with the largest upper bound consists now in descending the interval searching for a non void pointer. Full segments of memory can be freed any time the lower bound improves (line 19). When memory is an issue and there is no space to afford storing the whole interval $[best_0, upper_0]$ the data structure becomes a tree-of-intervals (Figure 5). The root node is now a smaller interval $[0, C]$ where $C = \frac{upper_0 - best_0}{d}$. Each item in $[0, C]$ is a pointer to an interval of size $\frac{C}{d} = \frac{upper_0 - best_0}{d^2}$ and so on.

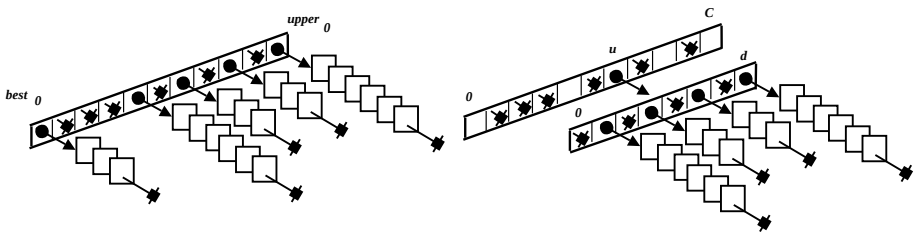

Fig. 5. Data Structure to store OPEN

The idea is extremely simple - but as far as we know it is the first time that it is proposed - and it can be applied to any best-first search branch-and-bound algorithm and therefore to algorithmic skeletons [15–17] supporting this

technique. Taking advantage of this structure we have sorted the subproblems with same uppers by their lower values. This can change the search order and cause the exploration of more nodes but the best solution value will increase more quickly and we will be able to discard more subproblems.

3 The Parallel Algorithm

In order to improve the sequential scheme, we propose a parallel algorithm that introduces a parallel generation of subproblems (meta-rectangles or builds) from a certain best subproblem. The general operation of this parallel scheme follows the same structure than the sequential scheme presented before (Figure 3). The main difference appears in the subproblem generation loops. Each processor involved in the resolution of the problem works on a section of the bidimensional CLIST. It combines the current best subproblem with the subproblems contained in its matrix section. The work distribution will depend on the processor characteristics. It can be a dynamic or static (cyclic, block) distribution.

Each processor keeps a replicated copy of CLIST. Meanwhile, OPEN will be distributed and only contains the subproblems generated by its owner processor. These structures allow the processors to work independently in the generation of new subproblems. After every combination of the current subproblem with the previous best ones, each processor has its own best current subproblem. To determine which is the global best current subproblem, it is necessary to add an all-to-all reduction point. Once all processors have the new best subproblem, they can begin with the generation work. The same reduction point is used to update the best solution current value.

A brief description of the OpenMP implementation is given below:

1. Each thread initializes its OPEN and CLIST variables. The master thread creates the initial subproblems and inserts them into its OPEN list.
2. At every iteration of the search loop, the global best subproblem must be identified. Each thread makes public its best subproblem by updating the corresponding entry at a shared static structure. The master thread determines the thread identifier having the best current subproblem and writes it to a shared variable. Then the slave threads are able to access the best global subproblem and copy it to a private variable. The owner of the best global subproblem must remove it from its OPEN list.
3. If the subproblem is not the solution and is not dominated/duplicated, it is inserted in each local CLIST (notice that operations for the detection of dominated/duplicated builds are not included in the pseudocode of Figure 3). If the subproblem is discarded, go to the previous step.
4. The horizontal new builds are generated in a first loop and in the second loop are generated the vertical ones. These loops have a *parallel-for* pragma, so each thread will do combinations of the subproblem with certain sections of the CLIST matrix. The new subproblems are inserted into the corresponding thread OPEN list.

5. Once all the new subproblems have been created and inserted into the lists, each thread must find its current best subproblem and copy it to the static shared array. The same is done with the best solution.
6. These steps must be followed until the solution is found.

The main problem of the implementation done on shared memory deals with the use of dynamic linked lists. These lists have to be modified by all threads and there is no mechanism available to ensure the integrity of data. OpenMP compilers usually ensure the integrity of the static structures, that is, arrays or structs stored in the static segment or in the execution stack. But this is not the case when dealing with data structures allocated in the heap. By this reason, some additional operations are necessary to update the shared dynamic data structures used by our implementation.

The exposed parallel algorithm can be easily implemented on a distributed memory scheme. As in this case, each processor would have its own OPEN and CLIST variables. A barrier point would be necessary to do the reduction of the best subproblem and send it to every processor in the team.

4 Computational Results

For the computational study, we have selected some instances from the ones available at [18]. From the instances proposed at [19] we have selected problem 1 from category 1 (cat1_1) and problem 2 from category 3 (cat3_2). The algorithms have been also tested with the problem instances exposed in [9]. Tests have been run over La Laguna University cluster (tarja). The cluster provides a Bull NovaScale 6320 SMP server that consists of 32 Intel Itanium 2 processors at 1.5GHz. The compilers used are: *gcc* 3.3.3 and *Intel* $C/C++$ 8.1.20.

Table 1 presents the results for the sequential algorithms. Columns labelled "Time" show execution times in seconds and the labelled "Gen.", "Comp." and "Ins." show the number of average generated, computed and inserted nodes respectively. Notice that the generated nodes are the nodes that represent any build created during the search process. Nodes removed from OPEN to be combined with all the previous best subproblems are the computed nodes. Inserted nodes represent the non duplicated generated nodes that can be inserted into OPEN. The results grouped under the name "Initial Version" are for an initial implementation based on VB algorithm. Sequential times for the modified version described in section 2 are also shown in the table under the label "Improved Version". As we can see, the modified sequential implementation introduces great improvements over the original version. The differences are due to the new data structures. They make possible to easily sort elements in OPEN and find duplicated/dominated nodes. But, when the number of generated nodes increases, the insertion of subproblems into OPEN turns too heavy for the first implementation. By this reason, large problem instances are not approachable by this version.

Table 2 shows the results for the parallel implementation of the improved algorithm. The columns labelled "Ins." and "Gen." show the number of nodes,

Problem Instance	Original Version				Improved Version			
	Gen.	Comp.	Ins.	Time	Gen.	Comp.	Ins.	Time
cat1_1	71631356	80575	124683	329,054	71631356	42842	122307	74,031
cat3_2	5073790	10968	146468	437,434	5067565	10966	145802	29,698
CL_10_24_01	1142805	4533	24654	2,714	1136429	3116	25335	1,042
CL_10_24_03	3547161	9551	32554	6,146	3544239	6625	33535	2,908
CL_10_24_09	1699757	7051	31819	4,873	1685099	4908	32049	1,580
CL_10_51_01	825620	2359	30913	4,582	848216	1732	31703	1,068
1	652	48	198	0,001	825	42	266	0,001
1_	27543	974	1979	0,017	35694	578	2555	0,079
2	11014	240	5804	0,171	11142	161	6011	0,017
2_	4414	136	2571	0,021	4586	91	2745	0,008
3	29935	1120	1868	0,046	31467	628	2141	0,038
3_	282	44	182	0,002	290	32	194	0,002
A1	21757	688	4031	0,061	25026	440	4057	0,104
A2	184186	5813	11749	0,694	172684	2511	9642	0,238
A3	24331	379	4188	0,069	24538	262	4384	0,036
A4	64866	558	27373	3,303	68394	374	29002	0,174

Table 1. Sequential Results - Original and Improved Versions

	Thread 1		Thread 2		Thread 3		Thread 4		Thread 5		Thread 6		Thread 7		Thread 8			
	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Ins.	Gen.	Comp.	Time
cat1_1																		
Th 1	122	71631															42	112.09
Th 2	72	35551	52	36079													50	73.49
Th 4	39	18896	28	19874	35	18093	26	17515									56	73.96
Th 8	20	11443	16	13898	14	8216	15	9427	18	7068	10	5657	20	9503	10	7894	57	68.99
cat3_2																		
Th 1	145	5067															10	32.08
Th 2	43	2621	44	2919													11	5.59
Th 4	32	175	31	179	23	148	23	110									3	0.77
Th 8	30	200	19	126	21	184	17	128	25	172	28	228	12	112	22	117	5	1.01

Table 2. Improved Version - Parallel Results

in thousands, inserted and generated by every thread. The number of computed nodes during the process is shown (also in thousands) in column "Comp.". Computational time, in seconds, invested in the search process is presented in column "Time". Parallel speedups strongly depend on the particular problem. That is a result of changing the search space exploration order when more than one thread is collaborating in the resolution. Even in worse cases (cat1_1) we can improve sequential times. A fair work load distribution between threads is difficult to obtain since it is not only needed to fairly distribute the subproblems to generate but also the ones to be inserted. Before doing a certain combination we are not able to know if a build will be valid or not (to be inserted).

5 Conclusions

An exact algorithm for the resolution of the Two-Dimensional Cutting Stock Problem has been presented. The implementation is based on VB and MVB algorithms. First of all, we have presented a new reformulation of the problem. A new syntax is introduced for the representation of the solutions. This notation helps in the detection of similar properties between different subproblems, making possible to efficiently detect duplicated combinations. By these representations we also are able to easily build the solution found. New data structures have been designed in order to efficiently manage insertions, combinations and dominance/duplication detections. The new data structure to manage subproblems in OPEN allows to do insertions in constant time independently of the number of nodes in the lists. The idea can be easily extend to any best-first search, branch-and-bound or algorithmic skeletons giving support to these techniques. In order to avoid the unnecessary recomputation of some subproblems, we have added a mechanism to store subproblems related to a particular element in CLIST. All these new features introduce an important improvement in the sequential exact algorithm. For being able to afford larger problem instances, we have presented a general parallel algorithm. The algorithm proposes a parallelization of the new build generation loop. On most space search algorithms, the slightest changes in the search order may cause dramatic consequences on the execution time. Super and sublinear speedups may occur since the parallel algorithm alters the sequential order. A first parallel implementation has been developed over shared memory. Porting an existing C application to OpenMP even if the algorithm is straightforwardly parallel can be sometimes a nightmare due to the lack of support to qualify dynamic memory variables as shared or private.

Future work targets improvement of both, the upper bound and the initial heuristic lower bound. This improvement in the bounds will allow to highly reduce the search space. In relation to the parallel algorithm we would like to develop a message passing implementation in order to compare with the one presented. A deep study of the work load distribution is also required.

6 Acknowledgements

This work has been supported by the EC (FEDER) and by the Spanish Ministry of Education inside the 'Plan Nacional de I+D+I' with contract number TIN2005-08818-C04-04. The work of G. Miranda has been developed under the grant FPU-AP2004-2290.

References

1. Sweeney, P.E., Paternoster, E.R.: Cutting and Packing Problems: A categorized, application-orientated research bibliography. Journal of the Operational Research Society **43**(7) (1992) 691–706

2. Dyckhoff, H.: A Typology of Cutting and Packing Problems. European Journal of Operational Research **44**(2) (1990) 145–159
3. Dowsland, K.A., Dowsland, W.B.: Packing Problems. European Journal of Operational Research **56**(1) (1992) 2–14
4. Burke, E., Kendall, G.: Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem. In Arabnia, H.R., ed.: Proceedings of the International Conference on Artificial Intelligence (IC-AI'99). Volume 1., CSREA Press (1999) 51–57 ekb@cs.nott.ac.uk, gxk@cs.nott.ac.uk.
5. Maouche, S., Bounsaythip, C.: Optimizing Textile Shape Placement by Tree Genetic Annealing. In: Proceedings of the Society for Computer Simulation Conference (SCSC'96). (1996) Salah.Maouche@univ-lille1.fr.
6. Roussel, G., Maouche, S.: Automatic Lay Planning for Irregular Shapes on Plain Fabric. Search in Direct Graph and ε-Admissible Resolution. In: Proceedings of the XVI IFIP-TC7 Conference, Compiègne, France (1993)
7. Christofides, N., Whitlock, C.: An Algorithm for Two-Dimensional Cutting Problems. Operations Research **25**(1) (1977) 30–44
8. Viswanathan, K.V., Bagchi, A.: Best-First Search Methods for Constrained Two-Dimensional Cutting Stock Problems. Operations Research **41**(4) (1993) 768–776
9. Hifi, M.: An Improvement of Viswanathan and Bagchi's Exact Algorithm for Constrained Two-Dimensional Cutting Stock. Computer Operations Research **24**(8) (1997) 727–736
10. Cung, V.D., Hifi, M., Le-Cun, B.: Constrained Two-Dimensional Cutting Stock Problems: A Best-First Branch-and-Bound Algorithm. Technical Report 97/020, Laboratoire PRiSM - CNRS URA 1525. Université de Versailles, Saint Quentin en Yvelines. 78035 Versailles Cedex, FRANCE (1997)
11. Tschöke, S., Holthöfer, N.: A New Parallel Approach to the Constrained Two-Dimensional Cutting Stock Problem. In Ferreira, A., Rolim, J., eds.: Parallel Algorithms for Irregularly Structured Problems, Berlin, Germany, Springer-Verlag (1995) 285–300 sts@uni-paderborn.de, http://www.uni-paderborn.de/fachbereich/AG/monien/index.html.
12. Nicklas, L.D., Atkins, R.W., Setia, S.K., Wang, P.Y.: The Design and Implementation of a Parallel Solution to the Cutting Stock Problem. Concurrency - Practice and Experience **10**(10) (1998) 783–805
13. Wang, P.Y.: Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems. Operations Research **31**(3) (1983) 573–586
14. Gilmore, P.C., Gomory, R.E.: The Theory and Computation of Knapsack Functions. Operations Research **14** (1966) 1045–1074
15. Tschöke, S., Polzer, T.: Portable parallel branch-and-bound library - PPBB-lib (1996)
16. Alba, E., et al: MaLLBa: A Library of Skeletons for Combinatorial Optimization. In: Proceedings of Euro-Par. Volume 2400 of Lecture Notes in Computer Science., Paderborn (GE), Springer-Verlag (2002) 927–932
17. Le-Cun, B., Roucairol, C.: BOB : a unified platform for implementing branch-and-bound like algorithms (1995)
18. Group, D.O.R.: Library of Instances (Two-Constraint Bin Packing Problem) http://www.or.deis.unibo.it/research_pages/ORinstances/2CBP.html.
19. Hopper, E., Turton, C.H.: An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem (1999) http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/strip1.txt.