LINEAR ONE-DIMENSIONAL CUTTING-PACKING PROBLEMS: NUMERICAL EXPERIMENTS WITH THE SEQUENTIAL VALUE CORRECTION METHOD (SVC) AND A MODIFIED BRANCH-AND-BOUND METHOD (MBB)

E.A. Mukhacheva
G.N. Belov
V.M. Kartack
A.S. Mukhacheva
Ufa State Aviation Technical University
Ufa, Russia

Abstract

Two algorithms for the one-dimensional cutting problem, namely, a modified branch-and-bound method (exact method) and a heuristic sequential value correction method are suggested. In order to obtain a reliable assessment of the efficiency of the algorithms, hard instances of the problem were considered and from the computational experiment it seems that the efficiency of the heuristic method appears to be superior to that of the exact one, taking into account the computing time of the latter. A detailed description of the two methods is given along with suggestions for their improvements.

Keywords: packing, cutting, pseudo-values, branch-and-bound method.

ISSN 0101-7438

1. Introduction

The classical one-dimensional cutting stock problem (**CSP**) has the following input data: length (bin capacity) L of the material to be cut, length (weight) l_i and demand b_i for each required item type $i = \overline{1,m}$. The goal is to determine a *cutting plan* $(A,x)=((a_{ij}),(x_j))$ $i = \overline{1,m};\ j = \overline{1,n}$ with minimal material consumption (the number of used bins). The columns of A are called *cutting patterns*, a_{ij} being the number of items of type i obtained from the j-th pattern. x_j indicates how many times the j-th pattern has to be cut. A possible mathematical model for the **CSP** is

$$\min\left\{ N = \sum_{j=1}^{n} x_j \left| \sum_{j=1}^{n} a_{ij} x_j = b_i, i = \overline{1,m}; x_j \in Z_+, j = \overline{1,n} \right. \right\}, \tag{1}$$

where n gives the number of different cutting patterns, N is the number of material units consumed, Z_+ is the set of non-negative integer numbers.

According to Dyckhoff's typology of cutting and packing this problem is classified as 1/V/I/M, Dyckhoff (1990). The one-dimensional bin-packing problem (BPP) (L,m,l) with $l = (l_1, l_2, ..., l_m)$ is a specific case of the **CSP** (L,m,l,b),when $b_i = 1$, $i = \overline{1,m}$. Both problems are very well known NP-hard problems and can be formulated as integer linear programming problems.

Heuristic methods e.g. the "*first fit decreasing*" (FFD) and "*greedy*" algorithm (GA) often result in optimal solution. The results of experiment by Schwerin P. and Wäscher G. (1998) prove this statement.

The authors though have not singled out the *hard*-problem classes of **BPP**. The experiment results are presented for only those classes with 100% positive solution.

In this paper we will:

- introduce a new heuristic algorithm, the Sequential Value Correction (**SVC**) and demonstrate its performance for the **BPP**;
- describe a new exact procedure for the **BPP** – a modified branch-and-bound method;
- show how the algorithm can be applied for *grouping* the initial data in order to increase the number m of different items and its *integration* into the **SVC** Mukhacheva & Zalgaller (1993) and a modified exact the branch-and-bound procedure (MBB) Katzev (1997) Kartack (1997) in order to decrease the computation time for solving a problem instance to a proven optimum;
- carry out numerical experiments according to Schwerin & Wäscher (1997) and try to single out *svc-hard* and *mbb-hard* problem classes.

Schwerin & Wäscher (1997) singled out *ffd-hard* and *extremely ffd-hard* **BPP** problem classes for **FFD**-algorithm. They also examined a *branch and bound* optimization procedure, **MTP**, from Martello & Toth (1990). Our experiment does not go beyond examining the variants of *ffd-hard* and *extremely ffd-hard* problem instances.

Additionally, a pre-processing technique for item grouping (**GROUP**) is implemented for the **SVC** (**MBB),** and the integration of the **SVC** with **MBB** allows us to apply these methods for problem classes with larger dimensions.

We do not compare, although the effectiveness of **SVC** and **MBB** against other well-known algorithms. However, some preliminary conclusions concerning **MTP** and **MBB** have been made. We believe that their algorithms could eventually be improved. A comparison of the results generated by the heuristic **SVC** and **MBB** can be viewed as a practical importance of **SVC**. Additionally, **SVC** and **MBB** are supposed to be used for creating hybrid algorithms of one- and two-dimensional packing.

The paper consists of Introduction, three Sections and Conclusion. The first section describes the **SVC** and **MBB** methods, their general ideas and the corresponding algorithms. The second section considers the grouping technique and its integration into **SVC** and **MBB** for solving problems with larger dimensions. The third section presents the test sample selection and the results of the computing experiments. It embraces all problem classes described in the paper by Schwerin & Wäscher (1997) and compares the above results with those of **MTPCS**, Schwerin & Wäscher (1998), and those of **BISON**, Scholl *et al* (1997). An outlook on further possible developments is presented in the Conclusion.

2. The SVC Packing

The sequential value correction method is carried out by a modified FFD with priority and repetition procedures.

The first phase of SVC is the generation of an initial feasible solution (or plan) and this is carried out by the FFD, where the corresponding value of the goal function N_f is taken as an upper bound. The lower bound is the natural bound $N_0 = \left\lceil \sum_{i=1}^{m} l_i b_i / L \right\rceil$ or N_0: $=N_\mu = \lceil Z_C^* \rceil$, where Z_C^* is the optimal value of the continuous relaxation of (1) obtained by the revised simplex method, Scheithauer & Terno (1995).

The paper by Mukhacheva & Zalgaller (1993) was the first work to present formulae for calculating initial y_i° and current y_i^v *pseudo-values* on the basis of the material-per-item consumption calculation rules (the idea of the simplex multipliers is used). Based on the initial cutting plan by **FFD**, trim losses h_j of every pattern $j = \overline{1, N}$ are distributed pro rata into the lengths of the items in the cutting patterns, taking an average within a single item type:

$$y_i^0 = \frac{l_i L}{b_i} \sum_{j=1}^{N} \frac{a_{ij}}{L - h_j}, \; i = \overline{1, m} \; . \tag{2}$$

In any iteration a new cutting plan is built pattern by pattern, each one under maximization of the total pseudo-value sum of the items, and these pseudo-values are corrected thereafter. The new value is a weighted average of the old one and the material consumption of the new pattern:

$$y_i^j = \frac{1}{b_i}(a_{i,j-1} l_i \frac{L}{L - h_{j-1}} + y_i^{j-1}(b_i - a_{i,j-1})) \; , \; \forall i : a_{i,j-1} > 0 \; , \tag{3}$$

here $\frac{b_i - a_{i,j-1}}{b_i}$ and $\frac{a_{i,j-1}}{b_i}$ are the corresponding weights after pattern $j-1$ is generated.

Notice that formula (3) is *deterministic*, moreover, when applied to some classes of *ffd-hard* problems the **SVC** algorithm does not lead these classes out of difficult domains boundaries. A further modification of the method is proposed, the *random value correction method,* that allows us to vary weights:

$$y_i^j = \frac{1}{\Omega \times (b_i + b_i^j) + a_{i,j-1}} (a_{i,j-1} l_i \frac{L}{L - h_{j-1}} + y_i^{j-1} \times \Omega \times (b_i + b_i^j)), \quad (4)$$

where Ω – is a parameter that should be changed randomly within the limits from one up to several units. This introduces a chance mechanism into the search process and prevents cycling. The weight of the old pseudo-value decreases when the current order demand b_i^j also decreases. This is one of many possible formulae which has proved its effectiveness. Each problem instance has several values of Ω that bring best solutions. Since we are unable to find them, we propose the use of Ω, varying in the interval [1,3] with a linear trend in a period of 30 pattern generations with a random distortion. The algorithm stops at reaching a lower bound or in a given number of iterations. We come to this conclusion having carried out a great number of calculations. This technique was used by Scheithauer, Terno and Belov.

Algorithm: Sequentional Value Correction Method

procedure GeneratePattern: The Branch&BoundMethod
procedure CorrectValues according to (4).

Step 0: ConstructSimpleSolution; CalculateInitialValues (2); *N*=0 (Iteration number);

Step 1: (Generate all patterns of a new cutting plan): *N=N+1; K=0;* If $N \bmod 100 = 0$ then $v = 2 + 4\, rnd(0,1)$;

Step 2: (Setting the random correction parameter): *K=K*+1 (Pattern Index);

Step 3: $\Omega = 1 + abs((K + N) \bmod 30 - 15) / v + 0.5 \cdot rnd(0,1)$;
If $\Omega < 1$ then $\Omega = 3$ else if $\Omega > 3$ then $\Omega = 1.12$;

Step 4: GeneratePattern; CorrectValues;

Step 5: If not all items used then go to Step 2;

Step 6: If not stop iterations then go to Step 1;

Step 7: STOP.

A peculiarity of the method was detected: the main improvement of the solution happens within the initial iterations which are carried out in fractions of the total running time. However, a large number of iterations may be necessary for the goal function to improve its value for the last unit.

The branch-and-bound method used for the pattern generation has an advantage over dynamic programming in speed and memory capacity even for large problems. It is so because the priority list of items according to their specific weights makes it possible to find a good solution very quickly. The pseudo-values do have the convenient property of significantly different quotients y_i / l_i that makes item dominance considerations effective as well.

3. The MBB Packing

For the context of this article an *optimization* algorithm is understood as an *exact* one. The searching scheme, suggested by Katzev (1979), is taken as the algorithm basis. The main idea is to develop an admissible solution based on the *first fit decreasing* algorithm (**FFD**). The number of bins involved is the upper bound. The lower bound is the solution of the continuous relaxation rounded up. In order to find the optimal solution one should first solve a corresponding **LP**. In most cases it brings one to the most exact lower bound and helps to escape using more "simple" lower bounds L_1, L_2, L_3, Marthello & Todd (1990). If the upper bound coincides with the lower one, then an optimal solution is found, otherwise the upper bound is reduced by 1. For this purpose, the searching procedure is fulfilled with the branch and bound method, using efficient bounding, based on the comparison of summary trim losses and possible reserves. If the bound cannot be lowered, then the solution obtained is considered to be an optimal one.

The above scheme cuts off in advance the solutions that are not optimal, but it does not take into account the properties of the generated cutting plans and it results in the excessive amount of information. Among the properties inherent to cutting plans the following ones should be pointed out:

Symmetry

Definition. A plan is considered to be *symmetrical* to another one if both of the plans consist of the same cutting patterns and differ only in which the pieces are allocated.

A plan consisting of **K** patterns has a symmetrical set of **K!** plans and they can be considered as equals since only the patterns are of interest and not the relative order in which they appear in a given plan.

In order to construct only this plan it is necessary to use the procedure of pattern rearrangement according to some priority. The highest priority is assumed to belong to that of some two patterns having a larger number of larger items (lexicographic ordering).

Dominance

Let two problems be given: $P_1 = (L, l_{1i}, m_1)$ and $P_2 = (L, l_{2j}, m_2)$, $i = \overline{1, m_1}$, $j = \overline{1, m_2}$, where L – bin capacity; m_1, m_2 – the number of items in problems P_1, P_2; l_{1i}, l_{2j} – item weights in problems P_1, P_2.

In problems P the number of items of each type is equal to 1. That is why any problem where every l_k item has some number of $b_k>1$ can be easily reorganized into problem P by using a l_k item b_k times.

Definition. Problem P_1 *dominates* P_2, if every item from P_1 can be compared to an item from P_2 in such a way that in every resultant pair $l_{1i} \le l_{2j}$, $i = \overline{1, m_1}$, $j = \overline{1, m_2}$ is observed.

Let us assume that $Z^*(P)$ is an optimal solution for problem P. The following statement is obvious:

Lemma 1. If problem $P_1 = (L, l_{1i}, m_1)$ dominates $P_2 = (L, l_{2j}, m_2)$, then $Z^*(P_1) \le Z^*(P_2)$.

A packing plan is built up from bin to bin so that each step may be presented as a solution of some sub-problem $P_{(k)}$, the latter consisting of so far unused items. Let us store

all sub-problems solved at each step. If the current sub-problem appears to be dominated by at least a single previously formed one, then there is no need to examine it, since the result obtained cannot exceed the previous one.

In practice it is impossible to store all sub-problems due to the exponential amount of memory required. However, a partial realization of the above condition is possible. Thus, at each step, before generating a packing procedure of the next in turn bin ($\mathbf{K_H}$), we shall memorize only the last sub-problem from those under consideration. Prior to generating a new ($\mathbf{K_H}$) bin, we just check if a new sub-problem is not dominated by the previous one. If this is not the case, then we make a step back, otherwise this new sub-problem is stored in the memory.

Algorithm of MBB

Step 1. Initialization.

- $S = \sum_{i=1}^{m} l_i b_i$
- Set lower bound $N_0 = \lceil Z_C(P) \rceil$ – simplex solution.
- Construct initial solution with **FFD** (resulting matrix **A** – cutting plan), $\mathbf{A}_{best}=\mathbf{A}_{\mathbf{FFD}}$. The number of patterns used is the upper bound N_u.
- Set reserve (i.e. trim loss when a stock is cut according to pattern j)
$$\Delta_j = L - \sum_{i=1}^{m} l_i a_{ij} \ .$$
- Set partial reserve $R_{(j)} = \sum_{i=1}^{j} \Delta_i$.

Step 2. If $N_0=N_u$ then go to Step 10. (The received solution is optimal).

Step 3. Set $R_o=L*(N_u-1)-S$ – admissible reserves. (A new attempt to build a cutting plan using less number of stocks, one at least, will be undertaken. Total remainder can't be more than R_0, that is why all solutions with the total remainder more than R_o may be cut off.)

Set K where $R_{(K-1)} \le R_o < R_{(K)}$.(The dimension of subproblem that satisfies to the above condition is determined. For the sake of saving the lexicographic order of cutting patterns the last stocks are removed.)

Step 4. Generation of new patterns $\mathbf{A}(K)$ that satisfy to $R(K) \le R_o$, symmetry and dominance. If it is successfully done then go to Step 7, else go to Step 5. (Sorting out all admissible variants with properties of *dominance* and *symmetry* we try to construct a new cutting plan with the total remainder no more than R_0, choosing stock by stock. If it is impossible then we make a step back and go on sorting.)

Step 5. $K=K$-1 (Back step).

Step 6. If $K=0$ then go to Step 10, else go to Step 4. (The cutting plan is not improved.)

Step 7. If all items are used then go to Step 9. (An improved cutting plan is constructed.)

Step 8. $K=K+1$ (Forward Step), go to Step 4.

Step 9. $\mathbf{A}_{best}=\mathbf{A}$, N_u – a number of stocks in $\mathbf{A}_{best}$, go to Step 2. (A new upper bound is constructed and the process is repeated with this bound.)

Step 10. Optimal solution $\mathbf{A}_{best}$.

Step 11. STOP.

4. Methods of Expansion of SVC and MBB Application Area

The **SVC** and **MBB** algorithms described in Sections 2 and 3 as it will be presented in the computational experiments have a good performance for BPP up to 100 items. Some methods to attack problems with larger number of items are suggested here. The first method is the *grouping* algorithm (**GROUP**) that allows us to decrease the initial problem dimension *m* and this can help problems with instances *m>100*. The second method is the **SVC** and **MBB** integration (**IMBB&V**) performed consequently. The suggested techniques have a positive effect in diminishing the experimental running time necessary to solve the problem.

4.1. Grouping

The **SVC** and **MBB** procedures have some drawbacks when a problem with *m>100* has to be solved. It is also a characteristic of problems with average-sized items: the item lengths are allocated in a small interval and their specific values y_i/l_i are quite close. Therefore, priority lists become of no use, branching ineffective and the overall process stalls. Item length bounds must be drawn apart.

To overcome the above situations a *grouping* technique is used. It reduces dimension *m* of the initial **BPP,** substituting it by a **CSP**. The problem of linear integer cutting *E* is known to have **IRUP** property if $Z^*(E)=\lceil Z_C(E)\rceil$

Thus a problem *P* is reorganized into P_r that includes a fewer number of item types but in a larger than $b_i=1$ quantity. Problem P_r should satisfy the following conditions:

1^0. *P* dominates P_r.

2^0. $\lceil Z_c(P)\rceil=\lceil Z_c(P_r)\rceil$.

3^0. $\lceil Z_c(P_r)\rceil= Z*(P_r)$.

Lemma 2. If a problem P_r satisfies the above conditions 1^0, 2^0, 3^0, then optimal integer solutions of *P* and P_r coincide, i.e. $Z^*(P_r) = Z^*(P)$.

It is obvious that $Z^*(P_r)$ and a cutting plan corresponding to it would represent a plan for problem *P* and $Z^*(P_r) = Z^*(P)$. It results from the condition that *P* dominates P_r.

The final solution is generated by substituting items from cutting plan of P_r by the domination of matched pairs from *P*. There are several different ways to construct the P_r. An example follows.

Let a problem $P=\left(L,\ m^P,\ l_i^P,\ b_i^P\right)$ be given. Then a corresponding problem $P_r=\left(L,\ m^{\Pr},\ l_i^{\Pr},\ b_i^{\Pr}\right)$ may be obtained by dividing all items from P into groups in such a

way that the maximal difference between the item lengths in a group does not exceed a certain threshold K_r. Here $m^{\Pr}$ is the number of groups; $l_i^{\Pr}$ is the maximal item length in the i-th group; $b_i^{\Pr}$ is the total sum of order demands in the i-th group; $i = \overline{1, m^{\Pr}}$.

If problem P_r satisfies the above conditions 2 and 3, then the procedure stops. Otherwise K_r is diminished and the procedure reiterates again.

Algorithm **SVC** with grouping is executed successively: **GROUP; SVC**. During the **GROUP** procedure, dimension m of the problem P is reduced. After **SVC** is done, the initial item lengths are reconstructed.

Grouping **MBB**, after items have been grouped, solves problem P_r in two steps. **LP** algorithm (continuous relaxation with *suitable rounding*) is applied to P_r at first. The solution vector x, generally not integral, is rounded down producing unpacked items that constitute the *residual problem* $\overline{P_r}$. The latter is processed as a trim loss problem (branch and bound method) in the second step. If the residual problem has the IRUP property (the gap between the integer and continuous solution less than 1), then we have an optimum for the main problem. Otherwise the exact algorithm should be applied to the latter that may have IRUP, see Scheithauer & Terno (1995). For the **BPP** we have order demands of one unit making suitable rounding ineffective (residual problem is often equal to the whole one) which makes such a technique as grouping necessary to produce larger order demands.

4.2. Integration of SVC and MBB

A great number of numerical experiments has shown that the solutions obtained by **SVC** performing are often optimal (the number of bins in a cutting plan equals to the lower bound). To quickly solve a problem to the optimum, one should use **SVC** first and then, if it failed, try **MBB**. The aggregate scheme of **SVC** and **MBB** performance is as follows.

Notation:

$P = \left(L, m^P, l_i^P, b_i^P\right)$ is the initial problem; $P_r = \left(L, m^{P_r}, l_i^{P_r}, b_i^{P_r}\right)$ is the problem obtained as a result of the grouping algorithm performance applied to P; $Z_C(P)$ is the optimal continuous solution of P, obtained by the simplex algorithm performance; $\overline{P}$ is the residual problem to P which has been obtained by intercepting of an integer part from the simplex solution and by reducing the number of items in the initial problem; $Z_{MBB}(P)$ is the integer optimal solution of P obtained by **MBB** performance; $Z_{SVC}(P)$ is the solution of P obtained by **SVC**.

Algorithm: Integration of SVC and MBB

Step 1: $Z_C(P)$ is calculated by the simplex algorithm;

Step 2: The residual problem $\overline{P}$ is constructed for P;

Step 3: The grouping problem P_r is constructed for P;

Step 4: The residual problem $\overline{P_r}$ is constructed for $\overline{P}$;

Step 5: $\overline{P_r}$ is solved by **SVC**;

Step 6: If optimal solution is found, that is $Z_{SVC}(\overline{P_r})=\lceil Z_c(\overline{P_r})\rceil$ (P_r has the **IRUP** property), then the end;

Step 7: $\overline{P_r}$ is solved by **MBB**;

Step 8: If optimal solution is found, that is $Z_{MBB}(\overline{P_r})=\lceil Z_c(\overline{P_r})\rceil$ (P_r has the **IRUP** property), then the end;

Step 9: If optimal solution is found, that is $Z_{SVC}(\overline{P})=\lceil Z_c(\overline{P})\rceil$ (P has the **IRUP** property), then the end;

Step 10: $\overline{P}$ is solved by **MBB**;

Step 11: If optimal solution is found, that is $Z_{MBB}(\overline{P})=\lceil Z_c(\overline{P})\rceil$ (P has the **IRUP** property), then the end;

Step 12: The initial problem P has not the IRUP property that is why it is necessary to solve the whole problem by **MBB**;

Step 13: STOP.

The performance of both algorithm procedures according to the above scheme can show one of the following results:

1. Both algorithms supplement each other (almost all problems can be solved with the help of **SVC** and **MBB**). There are no difficult classes.
2. The algorithms have their own areas where it is difficult to find solutions; their combinatorial efficiency level is made up of individual levels.
3. The algorithms have the same difficult areas in each class; their combination does not improve results.

5. Experiment with SVC and MBB Packing

5.1. Selection of test problem quantity

It is impossible to consider all the class problems within an acceptable time, hence a subset of problems uniformly distributed in the class should be chosen for an experiment.

The experiment is aimed at estimating a quota for problems of each class, solved to an optimum in a reasonable time. The value built on the basis of particular problem solutions, chosen randomly and uniformly from the class, is *unbiased*.

No more than 100 problems are usually solved to build up a value. Below we test this case reliability. The quota for the class problems solved optimally on the whole, denoted as *p,* is equal to optimal solution probability for a problem taken at random. Let us denote the quota of problems optimally solved in the test process as $\tilde{p}$; $\tilde{p}$ being a realized value of the corresponding random variable. Let us build up a 95% confidence interval for $\tilde{p}$ when p=0.5, 0.9, 0.99.

As far as sampling problems are generated independently, the solution optimality of each problem is independent too. Hence, $\tilde{p}$ is distributed according to the binomial law, so the following is true:

Laplace integral theorem. In a single test the probability of event A is equal to p, $p \in [0;1]$. The probability α that event A will appear from k_1 to k_2 times in a series of n independent tests is approximated by the definite integral

$$\alpha = \frac{1}{\sqrt{2\pi}} \int_{(k_1-np)/\sqrt{npq}}^{(k_2-np)/\sqrt{npq}} e^{-t^2/2} dt \text{ , where } q=1-p.$$

The following relation is a **corollary**

$$P\left(\left|\tilde{p} - p\right| \le \varepsilon\right) \approx 2\Phi\left(\varepsilon \times \sqrt{\frac{n}{p(1-p)}}\right) = \alpha ,$$

where ε is the half length of the confidence interval, $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-t^2/2} dt$ is the Laplace function, n is the sampling scope and α the confidence probability. For α=0.95 we find: $\varepsilon \sqrt{\frac{n}{p(1-p)}} \approx 1.96$. If n=100, then $\varepsilon \approx 0.196 \sqrt{p(1-p)}$; substituting p-values, we obtain:

$$p = 0.5: \quad \varepsilon \approx 0.098 ,$$

$$p = 0.9: \quad \varepsilon \approx 0.0588 ,$$

$$p = 0.99: \varepsilon \approx 0.0195 ,$$

and this is practically enough to estimate the algorithm efficiency. For n=10 the length of the confidence interval multiplies by $\sqrt{10}$ which is too large.

5.2. Test problems for the SVC and MBB

In order to carry out the experiments, the problem generator **BPPGEN** is used, in which every quadruple (L,m,v_1,v_2) describes a special homogeneous class of problems **BPP**. m is the problem dimension (the number of items), L is the bin capacity, v_1L and v_2L are the lower and upper bounds of item weight.

We conduct the test with **FFD** packing, Schwerin & Wäscher (1997), (Table 1) as initial results. According to them we define *ffd*-problem classes as:

ffd-easy problems, when $100 \ge \tilde{p} \ge 80$,

ffd-hard problems, when $80 > \tilde{p} \ge 20$,

extremely ffd- hard problems, when $20 > \tilde{p} \ge 0$,

here $\tilde{p}$ – an number of problems solved optimally out of 100 problems.

In the test with **SVC** and **MBB** only *ffd-hard* and *extremely ffd-hard* problem classes were considered. In Tables 1-3 gray and dark gray filling marks them. The calculation test has been done for the following parameters:

bin capacity:	$L = 1.000;$
problem dimension:	$m = 20,40,..., 180,200;$
lower limit of item weights:	$v_1 = 0.001; 0.05; 0.15; 0.25;$
upper limit of item weights:	$v_2 = 0.1; 0.2;...; 0.8; 0.9; 1.0.$

Thus the experiment covers all problem classes of Schwerin & Wäscher (1998). At first **FFD** and **LP** were used to solve the problem instances. The lower bound N_0 was found with the help of **LP**. When N_f, received by **FFD**, coincided with the lower bound N_0, the *ffd*-packings were excluded out of the experiment and were interchanged with new ones.

5.3. Results of the MBB and SVC methods

The test results with **MTPCS** by Schwerin & Wäscher (1998) for the problem classes, picked out by them, are presented in Table 1. Table 2 and 3 show the test results of *mbb*- and *svc*-algorithms performance on the same problem classes. Table 4 shows the test results of the integrated algorithm (**IMBB&V**) performance. The test results are compatible, since they have been carried out on the same type of computer (Pentium-200) and with the same computation time limit of 1,000 seconds for the optimization (exact) algorithm (not taking into account the time of linear programming algorithm performance). Time for solving a problem with the **SVC** algorithm has not been limited but it does not exceed 100 seconds. The maximum number of iterations was set to 10m. It was reached in particular classes, e.g. where m=60; v_1=0.15 and v_2=0.6 and 0.7. The average number of iterations varies from 0.05m to 0.5m.

All classes of problems, where m>100, have been solved by **MBB** with the help of grouping technique. **SVC** with grouping has been applied to problem classes with v_1=0.15; 0.25 and $m \geq 120$. **IMBB&V** has been used to solve the *mbb-hard* problems in cases where they have been *svc-easy*. For problem classes with v_1=0.25 **MBB** and **SCV** have been used autonomously.

The following conclusions may be drawn after analyzing the **MBB** results and those of **MTPCS**:

- there are no *extremely mbb- hard* problems;
- there are some *extremely mtpcs- hard* problems, namely $v_1 = 0.15; v_2 = 0.3; 0.5$ with the item quantity $m=180, 200;$
- all classes are *mbb-easy* when $m<100;$
- for all v_1 there are 32 classes of *mbb-hard* problems with $0.5 \leq v_2 \leq 0.8$ when $m \geq 100;$
- there are 43 classes of *mtpcs- hard* problems for all v_1 when $m \geq 100;$
- **MBB** is preferable to **MTPCS** when $v_1 = 0.001, 0.05, 0.15$ with $v_2 \leq 0.6$;
- **MTPCS** is preferable to **MBB** when $v_1 = 0.25$ with $v_2 = 0.5, 0.6$ and $m \geq 120$;
- when $m \leq 140$ all classes are *svc-easy*;
- when $m \geq 160$, $v_1 = 0.15; 0.25$ there are 4 classes of *svc-hard* problems;
- the heuristic **SVC**, the exact **MBB** or their integration should be chosen depending on the aim and a problem class, taking into account the computing time.

Table 1 – Number of problem instances per class for which **MTPCS** provides optimal solutions (*ffd-hard* instances only)

a) $v_1 = 0.001$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4										
0.5								93		
0.6				87		79	75	83	41	
0.7					78	70	74	71	51	53
0.8							50		58	49
0.9										

b) $v_1 = 0.05$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1				100		99		99		99
0.2									100	
0.3									99	96
0.4									93	88
0.5		97			71	77	81	73	44	31
0.6		99	91	86	82	75	80	70	75	61
0.7			97		84	67	70	54	31	59
0.8						85	68	65	52	55
0.9										

c) $v_1 = 0.15$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2		100	100	100	100	100	100	100	100	100
0.3		100	100	97	94	87	82	63	27	15
0.4		100	99	100	98	97	96	95	90	90
0.5	100	100	98	91	90	74	57	33	9	1
0.6		100	100	98	93	83	79	78	72	74
0.7		100	100	98	96	90	75	82	69	68
0.8								98	96	98
0.9										

d) $v_1 = 0.25$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4	100	100	100	100	100	100	100	99	99	97
0.5	100	100	100	100	100	100	99	98	89	59
0.6		100	100	100	100	100	100	99	100	100
0.7			100			100	74	100	100	100
0.8										
0.9										

Table 2 – Number of problem instances per class for which **MBB** provides optimal solutions (*ffd-hard* instances only)

a) $v_1 = 0.001$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4										
0.5								96		
0.6				88		85	87	80	71	
0.7					70	68	65	57	40	32
0.8							45		43	47
0.9										

b) $v_1 = 0.05$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1				100		100		98		100
0.2									100	
0.3									100	100
0.4									100	100
0.5		100			100	100	99	100	100	100
0.6		100	99	89	84	83	81	79	76	76
0.7			95		70	67	60	41	33	22
0.8						70	65	60	67	60
0.9										

c) $v_1 = 0.15$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2		98	97	100	100	100	100	100	100	100
0.3		100	100	100	100	98	98	98	94	90
0.4		100	100	100	100	100	96	99	96	96
0.5	100	100	99	100	90	94	86	82	90	86
0.6		100	100	100	98	83	76	78	84	60
0.7		100	99	97	95	80	80	72	70	60
0.8								94	83	81
0.9										

d) $v_1 = 0.25$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4	100	100	100	100	100	98	100	98	100	100
0.5	100	100	100	100	98	86	82	82	62	70
0.6		100	100	100	100	98	84	94	90	80
0.7			100			100	100	98	100	100
0.8										
0.9										

Table 3 – Number of problem instances per class for which **SVC** provides optimal solutions (*ffd-hard* instances only)

a) $v_1 = 0.001$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4										
0.5								100		
0.6				100		100	100	100	100	
0.7					100	100	100	100	99	82
0.8							98		95	98
0.9										

b) $v_1 = 0.05$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1				100		100		100		100
0.2									100	
0.3									100	100
0.4									100	100
0.5		100			100	100	100	100	100	100
0.6		100	100	100	100	100	100	100	98	98
0.7			100		100	98	93	98	95	90
0.8						92	92	97	88	92
0.9										

c) $v_1 = 0.15$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2		100	99	100	99	100	99	96	98	96
0.3		100	100	100	100	100	100	100	100	100
0.4		100	100	100	100	100	100	100	100	100
0.5	100	99	98	99	100	100	100	100	98	94
0.6		100	99	99	98	91	95	94	88	85
0.7		100	99	99	93	94	94	84	80	77
0.8								91	90	90
0.9										

d) $v_1 = 0.25$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4	100	100	100	100	100	100	98	94	99	96
0.5	100	99	94	84	81	91	80	76	68	70
0.6		99	99	99	95	100	92	91	83	81
0.7			100			92	90	81	84	80
0.8										
0.9										

Table 4 – Number of problem instances per class for which **IMBB&V** provides optimal solutions (*ffd-hard* instances only)

a) $v_1 = 0.001$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4										
0.5								100		
0.6				100		100	100	100	100	
0.7					100	100	100	100	99	82
0.8							98		95	98
0.9										

b) $v_1 = 0.05$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1				100		100		100		100
0.2									100	
0.3									100	100
0.4									100	100
0.5		100			100	100	100	100	100	100
0.6		100	100	100	100	100	100	100	98	98
0.7			100		100	98	93	98	95	90
0.8						92	92	97	88	92
0.9										

c) $v_1 = 0.15$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2		100	99	100	99	100	99	96	98	96
0.3		100	100	100	100	100	100	100	100	100
0.4		100	100	100	100	100	100	100	100	100
0.5	100	99	98	99	100	100	100	100	98	94
0.6		100	99	99	98	91	95	94	88	85
0.7		100	99	99	93	94	94	84	80	77
0.8								91	90	90
0.9										

d) $v_1 = 0.25$

v_2	m									
	20	40	60	80	100	120	140	160	180	200
0.1										
0.2										
0.3										
0.4	100	100	100	100	100	98	100	98	100	100
0.5	100	100	100	100	98	86	82	82	70	70
0.6		100	100	100	100	100	93	94	90	83
0.7			100			100	100	98	100	100
0.8										
0.9										

5.4. Comparison with BISON packing

The hybrid algorithm **BISON** by Scholl *et al* (1997) appeared to be highly efficient for solving **BPP** with different sets. One of the sets, data set 2, coincides with that considered in our paper. However, **BISON** has been tested by the authors according to a scheme different from the classification by Schwerin & Wäscher (1997, 1998). Scholl *et al* considers the classes with the stock length $L=1000$ and with the items of average length $l_j=L/3, L/5, L/7, L/9$ in the range with deviations $\delta = 20\%, 50\%, 90\%$ and with different item quantities $m=50, 100, 200, 500$. Thus, they select 12 different classes for each m, while Schwerin & Wäscher select 40 classes for the same purpose. We did not go beyond comparing the results for $m=50, 100, 200$. Thus 36 classes (Scholl *et al*) are selected out of 400 classes (Schwerin & Wäscher). Taking into account the fact that the test procedure did not involve *ffd-easy* problems, our experiment covered 145 classes. Besides, more than half of the classes by Scholl *et al* may also be classified as *ffd-easy* problems and should be eliminated from the experiment. Note that a quantity of problems solved by Scholl *et al* is equal to 10 per class. We think that it is not quite enough as a sample size (see 4.1). Nevertheless we show some results in Table 5, built according to the type of Table A3, Scholl *et al* (1997). The quality index is the number of problems solved to optimum out of 10 (#opt) with the help of algorithms **BISON, MBB** and **SVC**.

Table 5 – Detailed results of **BISON, MBBG** and **SVC** for data set 2.

m	l_i	BISON			MBB			SVC		
		δ=20%	δ=50%	δ=90%	δ=20%	δ=50%	δ=90%	δ=20%	δ=50%	δ=90%
50	*L*/3	10	10	9	10	10	10	10	10	10
	L/5	10	10	10	10	10	10	10	10	10
	L/7	10	10	10	10	10	10	10	10	10
	L/9	10	10	10	10	10	10	10	10	10
100	*L*/3	10	8	10	10	9	9	10	10	10
	L/5	10	10	10	10	10	10	10	10	10
	L/7	10	10	10	10	10	10	10	10	10
	L/9	10	10	10	10	10	10	10	10	10
200	*L*/3	10	10	9	10	9	8	10	9	10
	L/5	10	10	10	10	10	10	10	10	10
	L/7	10	10	10	10	10	10	10	10	10
	L/9	10	10	10	10	10	10	10	10	10

6. Conclusions and Outlook

In this paper, we have presented two algorithms for the Cutting Stock Problem and the Bin-Packing Problem (**CSP** and **BPP**), namely: a sequential value correction method (**SVC**) and a modification of the branch-and-bound method (**MBB**). Both heuristic and exact algorithms were tested for **BPP**. They appeared to be highly efficient for solving problems with $m \le 100$. As for large calculations of m we suggested a grouping procedure reducing the

problem dimension and an integrated algorithm **IMBB&V**. Grouping algorithms had been applied for problems with $m > 100$. Then we compared the efficiency of **IMBB&V** algorithm performing with that of **MTPCS**. The results appeared to be quite compatible with those received by **BISON**.

Encouraged by the results of our experiments we take the liberty of suggesting that further research should be as follows.

The **GROUP** algorithm is to be developed and corrected. It does not work well as yet for problem classes with $v_1=0.001$ and $v_1=0.05$. It is necessary to specify and analyze different grouping techniques for hard problems and to elaborate a suitable strategy for **GROUP** procedure combined with **MBB** and **SVC**.

To increase the **SVC** efficiency we are planning to introduce a procedure of "dominance" testing. It is proposed to wipe out all unpromising items when a new pattern is generated.

The hybrid algorithm, operating in **MBB** with a smaller upper bound obtained by the **SVC** fast variant, seems to be very promising.

The hybrid algorithm is now being worked out for solving the two-dimensional problem. The one-dimensional **SVC** with additional restrictions is used for this case to find a lower bound and an initial upper solution. It is important because exact algorithms are usually used only for $m \leq 20$.

The hybridization of **SVC** with exact algorithms for solving two-dimensional packing problems deserves a special investigation.

Acknowledgements

Thanks to the referees for detailed comments and suggestions that improved the presentation. Special thanks to Dr. Horacio Hideki Yanasse for his patience and goodwill.

This work was supported by the Russian Foundation for Basic Research, grand 99-01-00937.

References

(1) Belov, G. (1997). A modified sequential values correction method for the one-dimensional cutting stock problem. Decision Making under Conditions of Uncertainty (Cutting-Packing Problems). *The International Scientific Collection, Ufa, Russia*, 41-47.

(2) Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, **44**, 145-159.

(3) Kartack, V.M. (1997). Combinatorial algorithms for solving linear cutting. Decision Making under Conditions of Uncertainty (Cutting-Packing Problems). *The International Scientific Collection, Ufa, Russia*, 33-40.

(4) Katzev, S.V. (1979). Solution of mini-maximum problems of set with determined equivalent relations. *Cybernetics*, **3**, *Kiev*, 113-118.

(5) Martello, S. & Toth, P. (1990). *Knapsack problems: Algorithms and Computer Implementations.* John Wiley and Sons, Chichester, 1990.

(6) Mukhacheva, E.A. & Zalgaller, V.A. (1993). Linear programming cutting problems. *International Journal of Software Engineering and Knowledge Engineering*, **3**, 463-476

(7) Scheithauer, G. & Terno, J. (1995). The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, **84**, 563-571.

(8) Scheithauer, G.; Terno, J.; Muller, A. & Belov, G. (1999). *Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm*. Technische Universitet Drezden Math-Nm-06-1999, 24p.

(9) Scholl, A.; Klein, R. & Juergens, C. (1997). BISON: A fast hybrid procedure for exactly solving the one-dimensional Bin-Packing Problem. *Computers and Operational Research*, **24**(7), 627-645.

(10) Schwerin, P. & Wascher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, **4**, No 5/6, 337-389.

(11) Schwerin, P. & Wäscher, G. (1998). A new lower bound for the Bin-Packing Problem and its integration into MTP. *Betriebswirtschaftliche Diskussionsbeiträge, Beitrag Nr. 98/26.* Martin-Luther Universität Halle-Wittenberg. 23p.