Seminar of I. Vekua Institute of Applied Mathematics REPORTS, Vol. 34, 2008

ONE-DIMENSIONAL BIN PACKING CLASS: FAST ALGORITHMS OF FINDING THE BOUNDS OF OBJECTIVE FUNCTIONS

Fedulov G.

Institute of Analytical Technique

Abstract. We research a class of 16 combinatorial models, that are semantically near to a known One-Dimensional Bin Packing task. All models have a large number of practical applications in the different areas: an one-dimensional stock cutting, a placing of files to CDs, a schedule theory, a placing of loads to the containers and so on. A general description of class is to divide an initial set of weights into a some number of disjoint subsets with the given properties, which are defined by using the model restrictions. Primary attention of paper has been given to the estimation of quality of approximation solutions as a measure of closeness to the optimal solutions. With that purpose, we build the bounds of objective function which the approximation solutions are compared with. To find the bounds, we use two blocks: a block to reduce the initial size of tasks and a block to build an estimation corridor of reasonable solutions. A first block removes the dominate groups of weights (the dominate pairs, triplets, quarters and so on) from the initial data. A second block estimates the existence of reasonable solutions for a fixed number of subsets. Our algorithms for finding the bounds can be used in practice for large-sizes tasks (the number of different weights may be 50000 and more) as an alternative to other approaches when the time factor is important.

Keywords and phrases: Bin packing, approximation solution, lower bound, upper bound, initial reduction, estimation corridor.

AMS subject classification (2000): 90C10.

1. Introduction

We research a class of 16 combinatorial models that are semantically near to a known One-Dimensional Bin Packing (1DBP) problem [4]. All models have a large practical applications in the different areas: One-Dimensional Stock Cutting, placing of files on CDs, Scheduler Theory, a Container Loading and so on. A general description of class is following. Given a set of items $A = \{a_1, a_2, \ldots, a_n\}$, to each item a_k corresponds a weight $s(a_k)$ and a profit(cost) $p(a_k)$, $s(a_k) \geq s(a_{k+1})$. We need to divide the initial set A into M disjoint subsets $A_1, A_2, \ldots, A_M$, $\bigcup_{i=1}^{M} A_i = A$, $A_i \cap A_j = \emptyset$, $i \neq j$, $i, j \in [1, M]$ with the given properties. All subsets are independence ones and a sequence of weights within each subset is any. We denote $S(A) = \sum_{k=1}^{n} s(a_k)$ as a sum of weights A, $C_i = \sum_{a_k \in A_i} s(a_k)$ as a sum size of items (a bin content) of ith bin and $P_i = \sum_{a_k \in A_i} p(a_k)$ as a sum profit(cost) of items of ith bin, $i \in [1, M]$. One can represent an initial set of weights $\{s(a_1), s(a_2), s(a_n)\}$ in a compact form: $W = \{w_1 \circ k_1, w_2 \circ k_2, \cdots, w_m \circ k_m\}$, where $w_1 > w_2 > \cdots > w_m$, $w_i \circ k_i$ is a group of equal weights w_i , k_i is a multiplacity, $\sum_{i=1}^{m} k_i = n$, $\sum_{i=1}^{m} k_i w_i = S(A)$. Thus, a parameter m is a **number of different weights**. Below we give a description of models of **1DBP** class.

Model 0. Base Model. Given a fixed list of bins $L = \{B_1, B_2, \ldots, B_M\}$, $B_i \geq B_{i+1}$, the B_i is a capacity of ith bin, $S(L) \geq S(A)$, where $S(L) = \sum_{i=1}^{M} B_i$. We need to pack A into L: $C_i \leq B_i$, $i \in [1, M]$. An answer is **YES** if we can pack A into L and **NO** otherwise.

Model 1. Classical Bin Packing. To divide A into a **minimal number** M of disjoint subsets: $C_i \leq B$, $i \in [1, M]$, B is a bin capacity.

Model 2. Bin Covering. To divide A into a **maximal number** M of disjoint subsets: $C_i \geq B$, $i \in [1, M]$, B is a bin quota.

Model 3. Bin Packing & Bin Covering 1. To divide A into **a minimal number** M of disjoint subsets: $B_{\min} \leq C_i \leq B_{\max}$, $i \in [1, M]$, where the parameters $B_{\min}$ and $B_{\max}$ are the lower and upper thresholds respectively.

Model 4. Bin Packing & Bin Covering 2. Model 4 is similar to **Model 3** but it is need to find **a maximal number** M.

Model 5. Schedule Theory. M is fixed. To find a **minimal bin size** B in order to divide A into M of disjoint subsets: $C_i \leq B$, $i \in [1, M]$.

Model 6. Schedule Theory (General Model 5). Given a list $\tau_1, \tau_2, \ldots \tau_M$ of positive real numbers. It is need to find a **minimal positive integral number** T in order to pack A into a list of bins $L = \{B_1, B_2, \ldots B_M\}$: $C_i \leq B_i$, $i \in [1, M]$, $B_i = T\tau_i$.

Model 7. Bin Packing with a range of B. Given a range $[B_{\min}, B_{\max}]$ of bin capacities. It is need to find an **optimal bin capacity** B in order to a product $MB \rightarrow min$, where M is a solution of **Model 1**.

Model 8. Bin Packing with the decreasing bin capacities. Given a decreasing sequence of bins $B_1 \geq B_2 \geq \cdots \geq B_q$. It is need to find a minimal number $M \leq q$ in order to pack A into a list of bins $\{B_1, B_2 \ldots B_M\}$: $C_i \leq B_i$, $i \in [1, M]$.

Model 9. Maximal loading of weights. Given a fixed list of bins $L = \{B_1, B_2 \ldots B_M\}$, where $S(A) \geq S(B)$, where $S(B) = \sum_{i=1}^{n} B_i$. It is need to find a subset $A' \subseteq A$ in order to pack A' into L: $C'_i \leq B_i$, $i \in [1, M]$ and a sum weight $S(A') \rightarrow max$.

Model 10. Maximal loading of profits (General model 9). Model 10 is similar to **Model 9** but it is need to find a subset A': sum profit $S(P') \rightarrow max$.

Model 11. Minimal loading of weights Model 11 is similar to **Model 9** but it is need to find a subset A': $C'_i \geq B_i$, $i \in [1, M]$ and a sum weight $S(A') \rightarrow min$.

Model 12. Minimal loading of costs (General Model 11). Model 12 is similar to **Model 11** but it is need to find a subset A': a sum cost $S(P') \rightarrow min$.

Model 13. Minimal sum capacity of subset of bins. Given a list of bins $L = \{B_1, B_2, \ldots B_M\}$, where $S(A) \leq S(L)$. It is need to find a subset $L' \subseteq L$ in order to pack A into L': a sum bin capacity $S(L') \rightarrow min$.

Model 14. Minimal sum cost of subset of bins (General Model 13). Given a list of bins $L = \{B_1, B_2, \ldots B_M\}$. Each bin B_i has a cost $\mathcal{P}_i$. **Model 14** is similar to **Model 13** but it is need to find a subset L': a sum bin cost $S(\mathcal{P}') \rightarrow min$.

Model 15. Bin Packing with a range of multiplicities of weight. We consider such W, where $k_i \in [k_i^{min}, k_i^{max}]$. We fix k_i and for a given bin capacity B and solve **Model 1**. We need to find such k_i in order a sum waste $MB - S(W) \rightarrow min$, where $S(W) = \sum_{i=1}^{m} w_i k_i$.

All initial data are the positive integer numbers. Here we have presented the known models from [2,3,6] (and the other sources) and new models as 3,4,8,14. The **Models 1-15** are the optimization ones that is led to **Model 0** in process of solving. All models are the **NP**-hard problems to find the optimal solutions $OPT(D)$ for an arbitrary initial data D and are solved in practice as rule using the approximation algorithms . Let we have some approximation algorithm $\mathcal{A}$ that produces an approximation solution $\mathcal{A}(\mathcal{D})$ that it is necessary to evaluate somehow. A measure of closeness $\mathcal{A}(D)$ to $OPT(D)$ is $q = \frac{\mathcal{A}(\mathcal{D})-\mathcal{OPT}(\mathcal{D})}{OPT(D)}100\%$. But finding of $OPT(D)$ is NP-hard problem and as rule $OPT(D)$ is not known. In this case we find the bounds of objective function: a lower bound $LB(D)$ to $OPT(D)$ for the tasks "to minimum" and an upper bound $UB(D)$ to $OPT(D)$ for the tasks "to maximum". One can write $UB(D) = \mathcal{A}(\mathcal{D})$ for the tasks "to minimum" and $LB(D) = \mathcal{A}(\mathcal{D})$ for the tasks "to maximum". Thus we get $LB(D) \leq OPT(D) \leq UB(D)$. Since $OPT(D)$ is not known, we consider other measure $p = \frac{UB(D)-LB(D)}{LB(D)}100\%$, where $p \geq q$. In case $p = 0$ we claim $\mathcal{A}(\mathcal{D})$=$OPT(D)$. Thus, finding the best bounds has the large practice importance.

At the moment is most known **Model 1** to that is devoted many publications. This problem lets to develop a large number of the different heuristic approximation algorithms $\mathcal{A}$. Among these algorithms there are a large interest to the such ones where one can evaluate a behavior of algorithm in worst case for all initial data by formula $\mathcal{A}(\mathcal{D}) \leq \alpha\mathcal{OPT}(\mathcal{D}) + \beta$. where α and β are the real constants, $\alpha \geq 1$. As example, for a known fast algorithm FFD (First Fit by Decreasing) in [5] is proved a result $FFD(A) \leq \frac{11}{9}OPT(A) + 4$. A complexity of FFD is $O(m^2)$. But a number of like algorithms is not very many. Most of effective algorithms no have the theoretical results of worst case. Here we ask: how to evaluate a worse case (to find the asymptotic constant α) of algorithm $\mathcal{A}$ by experimental way? We suppose a rough decision would be following. For a fixed n we present a limited number of N ranges $\Delta_i(n) = [\rho_i, \sigma_i], 0 < \rho_i < \sigma_i < 1, \rho_i < \frac{1}{2}, i \in [1, N]$. For each range $\Delta_i(n)$ we generate the different random distribution types (e.g. a uniform distribution) very many times. For each kth distribution $s(a_j) \in [\rho_i B, \sigma_i B], j \in [1, n]$, we find $\mathcal{A}(\mathcal{D})$, $LB(D)$ and set $p_i^k(n) := p$. Then $p_i^{\max}(n) = \max_k p_i^k(n)$ will be a worst case of all $p_i^k(n)$ for a range $\Delta_i(n)$. We define $p(n) = \max_i p_i^{\max}(n)$ and state an experimental result as $\alpha'(n) = 1 + p(n)/100$ that shows a primary opinion about the real α, where $\alpha'(n) \leq \alpha$ or $\alpha'(n) > \alpha$. At present a best lower bound for **Model 1** is an $LP(A)$ that is found using a Linear Programming technique. In [7] are spoken a hypothesis that $OPT(A) - LP(A) \leq 1$ for all initial data D since no one instance has been discovered with $OPT(D) - LP(D) = 2$ yet. It follows the LP-bound is a near-optimal one. But a complexity of finding LP-bound increases dramatically with m. In [1] there are the tables of experiments for the different groups of initial data where in particular for a range of weights $(B/4, B/2]$ and $m = 3200$, $n = 1,000,000$ the total LP-runtime is about 100 hours . Thus, using LP-method for the largest parameters m is impossible in practice. But we must observe that a structure of LP-approach doesn't let to break a process to a given moment since LP-bound may be incorrect. That is, to get a correct lower bound by LP-method it is necessary to wait a finish of LP-program. In order words, the LP-approach lets to find the near-optimal lower bounds for the large m by expensive price. To reduce the total runtime is used a grouping method. An idea of method is we change an initial data W to a data W' with a less number $m' < m$ (we can represent W and W' as the break-lines $\{s(a_k)\}$ and $\{s'(a_k)\}$ respectively, $s'(a_k) \leq s(a_k), k \in [1, n]$). But a difference $d = LP(W) - LP(W')$ can be essential: the less m' the more d. Thus is actual the approaches to find the both fast and quality lower bounds for the large m. In our paper we propose one of such approaches to solve this problem.

Our estimation technique is of the two blocks: the initial reduction and estimate corridor. The first block removes the dominate groups of weights from the initial data D and produces the initial reduction of two types. The first type (**A-type**) is used only for **Model 1** by a formula: $OPT(A) = M_0 + OPT(A'), M_0 = M_1 + M_2 + M_3 + M_4 + \cdots + M_H$, where M_1, M_2, M_3, M_4, $\ldots M_H$ are the numbers of the dominate singletons, pairs, triplets, quarters, $\ldots$ respectively, M_0 is a number of bins reduced, $A' = A \setminus A^0, A^0 = \bigcup_{i=1}^{H} A^i$, $A^i = \bigcup_{j=1}^{M_i} A^i_j$, $H := H(B)$ is a maximal number of weights to put into a bin of capacity B. A singleton is a bin of one weight, a pair is a bin of the two weights, a triplet is a bin of the three weights and so on. Each subset A^i_j is a dominate group of i weights. Here $A^1, A^2, A^3, A^4, \ldots A^H$ are the lists of the dominate singletons, pairs, triplets, quarters, $\ldots$ respectively. The second type (**B-type**) is the general one for all models and is used to solve **Model 0** by a formula: $(A, L) \rightarrow (A', L')$. Here we lead an initial data (A, L) to a data (A', L'). The second block estimates an existence of reasonable solutions for a fixed number (M) of subsets. This block solves a problem: does exist a packing A into M bins: $0 < B_i^{\min} \leq C_i \leq B_i^{\max} \leq B_i, i \in [1, M]$? We define a predicate $P(A, B^{\min}, B^{\max})$ =**NO**, if we claim "packing A into L doesn't exist" and $P(A, B^{\min}, B^{\max})$ =**YES** otherwise. A result of solving it problem is an estimate corridor $[C_i^{\min}, C_i^{\max}] : B_i^{\min} \leq C_i^{\min} \leq C_i \leq C_i^{\max} \leq B_i^{\max}$, $i \in [1, M]$. These blocks are interlinked closely. The results of the first block are used in the second block and vice versa.

In Section 2 we describe the initial reduction algorithms. In Section 3 we describe the algorithms of building the estimation corridor. In Section 4 we give the procedures of building the bounds for our models. In Section 5 we give the experimental results.

2. Initial reduction

2.1 A-type initial reduction

Definition 1. We call a group $G = \{a_{N_k(i)}\}$, $N_k(i) = N_{k-1}(i) + 1$, $k \in [1, i]$ as a dominate one, if a number $p := N_1(i)$ has a property: $\sum_{k=p}^{p-k+1} s(a_k) \leq B$, $\sum_{k=p-1}^{p+i-2} s(a_k) > B$, where $N_0(i) := N_1(i) - 1$ is a number of items before a_p.

Here $N_1(1)$ defines a number for the dominate singletons, $N_1(2)$ for the dominate pairs, $N_1(3)$ for the dominate triplets, $N_1(4)$ for the dominate quarters and so on. If an optimum solution has at least one group $G' = \{a_{N'_k(i)}\}, k \in [1, i]$, where $N'_1(i) \geq N_1(i)$, then we can remove G from A and put G into A^0 since $s(a_{N_k(i)}) \geq s(a_{N'_k(i)})$ because of $N'_k(i) > N_k(i)$, $k \in [1, i]$.

Algorithm A to build A^0.
1. $A^0 := \emptyset$, $A' := A$, $M :=$**P**(A').
2. $i := 0$, $\mu(0) := 0$.
3. $i := i + 1$.
4. $M' := M - \mu(i-1)$. **If** ($M' = 0$) **STOP**.
5. Algorithms **A2** and **A3** to find G.
6. **If** $G \neq \emptyset$ { $A^0 := A^0 \bigcup G$, $A' := A' \setminus G$, $M :=$**P**(A') }.
7. Algorithm **A1** to find $\mu(i)$.
8. **Go to 2**.

Here **P**(A') is an algorithm that produces a bound $M : \min_M P(A', B^{\min}, B^{\max})$=YES, $B_i^{\min} = w_m, B_i^{\max} = B, i \in [1, M]$, $\mu(i-1)$ is a maximal number of bins that can be used by weights of range $[1, N_0(i)]$, $\mu(i-1) \leq N_0(i)$. The algorithms **A1** and **A2** try to find G.

Algorithm A1 to find $\mu(i)$ We will find $\mu(i)$ by using a formula $\mu(1) = N_0(2)$, $\mu(i) = \mu(i-1) + x(i)$, $i \geq 2$, where $x(i)$ is a maximal number of bins that can use $x(i)$ weights from a range $\Delta(i) = [N_1(i), N_0(i+1)]$, $x(i) \leq k_0 = N_0(i+1) - N_1(i) + 1$. We ask: can we put each weight of $\Delta(i)$ into a personal bin? Suppose we have put k weights of $\Delta(i)$ into k bins. We consider a sum of the first k weights of $\Delta(i)$ as $S_1(k) = \sum_{j=N_1(i)}^{N_1(i)+k-1} s(a_j)$ and a sum of ik easiest weights as $S_2(k) = \sum_{j=n-ik+1}^{n} s(a_j)$. If $S_1(k) + S_2(k) > kB$ then at least one of k bins will be have not more i weights. As any group $\{s(a_{J_1}), s(a_{J_2}), \ldots s(a_{J_i})\}$ is dominated by G, $J_1, J_2, \ldots, J_i \in \Delta(i)$, we can not use k bins by k weights of $\Delta(i)$. Because of we have the two cases: to put the kth weight of $\Delta(i)$ into a bin of $\mu(i-1)$ bins where we have put the weights $s(a_j)$, $j \in [1, N_0(i)]$ or to join the kth weight with one of previous $k-1$ weights $s(a_j)$, $j \in [\mu(i-1)+1, \mu(i-1)+k-1]$. The other details we put to an algorithm of building $x(i)$.

We denote $S_1(i, q) = \sum_{j=n-iq+1}^{n} s(a_j)$ and $S_2(q) = qB$.

Algorithm to find $x(i)$
1. $x(i) := 0$, $k_0 := N_0(i)$, $k := k_0$, $p := 0$.
2. $k := k + 1$, $q = k - k_0$. **If** ($k > N_0(i+1)$) **STOP**
3. If $\left(\sum_{j=k_0+1}^{k} s(a_j) + S_1(i, q) \leq S_2(q) \right)$ $\{x(i) := x(i) + 1\}$
else { $p := p + 1$, $k_0 := k_0 + 1$, **If** ($p = i$) $\{x(i) := x(i) + 1, p := 0\}$ }.
4. **Go to 2**.

Algorithm A2 to build A^0. We consider a number M' of bins of range $[1, \mu(i-1)]$. Let an algorithm packs a maximum number K of the weights $s(a_j)$ into M' bins, $K \geq M'$, $j \in [1, K]$. It follows we can put not more $n - K$ weights into $M - M'$ bins since a set of weights $\{s(a_1), \ldots, s(a_K)\}$ dominates any set of K weights $\{s(a_{J_1}), \ldots, s(a_{J_k})\}$ since $s(a_k) \geq s(a_{J_k})$, $k \in [1, K]$. If $n - K < (i+1)(M - M')$ it follows we find at least one group of i weights to put into a bin from $M - M'$ bins. If we get a result $n - K < (i+1)(M - M')$ for all $M' \in [1, \mu(i-1)]$ then we can remove the dominate group G from A and put G into A^0.

Algorithm A3 to build A^0. Let a difference $M'' = M - \mu(i-1) > 0$. It follows: each bin of range $\Delta = [N_1(i), N_1(i) + M'' - 1]$ has the weights with the numbers $j \geq N_1(i)$. We consider any $k \in \Delta$ and ask: can we put k weights $\{s(a_j)\}, j \in [N_1(i), N_1(i)+k-1]$ into k bins? In other words: can we put only one weight into each bin? In this case each bin have to get not less $i+1$ weights. We denote $S_1(k) = \sum_{j=N_1(i)}^{N_1(i)+k-1} s(a_j)$ as sum of k weights and $S_2(k) = \sum_{j=n-ik+1}^{n} s(a_j)$ as sum of ik easiest weights. If $S_1(k)+S_2(k) > kB$ then at least one of k bins must get a group of i weights of range Δ. As any i-group $\{s(a_{j_1}), s(a_{j_2}), \ldots s(a_{j_i})\}$ is dominated by $\{s(a_{N_1(i)}, s(a_{N_1(i)+1}, \ldots s(a_{N_1(i)+i-1})\}$, $j_1, j_2, \ldots j_i \in \Delta$, we claim: we can't put k weights into k bins. Now we want to know: can we put k weights $s(a_j)$ into $k' \in [1,k]$ bins? Again, each bin have to get not less $i+1$ weights. We denote $S_2(k') = \sum_{j=n-k'(i+1)+k+1}^{n} s(a_j)$. If $S_1(k) + S_2(k') > k'B$ then at least one of k' bins gets not more i weights of range Δ. If we get a result $S_1(k) + S_2(k') > k'B$ for all $k' \in [1,k]$ for a fixed $k \in [1, M'']$, then we can remove the dominate group G from A and put G into A^0.

2.2 B-type initial reduction

Let we given by the constraints $B_i \geq B_i^{min}, i \in [1,M]$. We will use a parameter **par** as **1** in case $B^{min} \neq \emptyset$ and as **0** otherwise. Now we consider a group G and a range of bins B_i, $i \in [q,Q]$. Let $P(q)$ is a minimal number: $s(a_{P(q)}) \leq B_q$, $s(a_{P(q)-1}) > B_q$. Let $\sum_{j=p}^{p+i-1} s(a_j) \leq B_Q$, $p := N_1(i)$, here $N_1(i)$ we form for $B := B_Q$, $i = 1, 2, \ldots H(B)$. Let a difference $M'' = Q - q + 1 - \mu(i-1) > 0$.

Algorithm B2. We consider a number M' of bins of range $[1, \mu(i-1)]$. We build B' as a set of bins as following: $B'_i = B_i$, $i \in [1, q-1]$, $B'_i = 0$, $i \in [q, q-1+M']$, $B'_i = B_i$, $i \in [q+M', M]$. Let an algorithm packs a dominate set of weights $D(K) = \{s(a_{I_1}), s(a_{I_2}), \ldots, s(a_{I_K})\}$ into B' bins and a number K is maximal. It follows any set $D'(K) = \{s(a_{J_1}), s(a_{J_2}), \ldots, s(a_{J_K})\}$ of K weights that we can put into B' bins will be dominated by $D(K)$: $s(a_{I_k}) \geq s(a_{J_k}), k \in [1,K]$. Then $n-K$ will be a maximal number of weights that we can put into the bins B_i, $i \in [q, Q-M']$. If we get a result $n - K < (i+1)(Q-q+1-M')$ for all $M' \in [1, \mu(i-1)]$ then we can remove the dominate group G from A and put G into B_Q, after we remove G and B_Q from the initial A and L and set $A' := A \setminus G$, $L' := L \setminus B_Q$.

Algorithm B3. We define a set of numbers $J = \{1, 2, \ldots n\} \setminus \{I_1, I_2, \ldots, I_K\}$ that we can use as the numbers for the easiest weights. We denote $S_1(k) = \sum_{j=N_1(i)}^{N_1(i)+k-1} s(a_j)$ as a sum of k heaviest weights from a range $[N_1(i), N_1(i) + M'' - 1]$, $S_2(k') = \sum_{j=n-k'(i+1)+k+1}^{n} s(a_{I_j})$ as a sum of ik' easiest weights and $S_3(k') = \sum_{i=q}^{q+k'-1} B_i$ as a sum of k' heaviest bins of range $[q,Q]$, $k' \in [1,k]$. If we get a result $S_1(k) + S_2(k') > S_3(k')$ for all $k' \in [1,k]$ for a fixed $k \in [1, M'']$, then we can remove G from A and put into B_Q, after we set $A' := A \setminus G$ and $L' := L \setminus B_Q$.

Algorithm B(par).
1. $A' := A$, $L' := L$.
2. $q := 0$, $G := \emptyset$.

3. $q := q + 1$. **If** ($q > M$) **return 1**.
To build $N_1(i)$ for the $B := B_Q$, $i = 1, 2, \dots H(B)$.
If ($P(q) = P(q-1)$) **Go to 3**.
4. $Q := q - 1$.
5. $Q := Q + 1$. **If** ($Q > M$) **Go to 3**.
If ($B_Q = B_{Q+1}$) **Go to 5**.
6. $i := 0$, $\mu(0) := 0$.
7. $i := i + 1$.
8. $M' := Q - q + 1 - \mu(i-1)$. **If** ($M' = 0$) **Go to 5**.
9. Algorithms **B2** and **B3** to find G.
10. **If** ($G \neq \emptyset$) {
If (par = 1) { **If** ($sum(G) < \min_{q \le j \le Q} B_j^{min}$) **return 0** else **Go to 11** }.
Build $A' := A' \setminus G$ and $L' := L' \setminus B_Q$.
If ($P(A', L')$=NO) **return 0** else **Go to 2**. }
11. Algorithm **A1(**$B := B_q$**)** to find $\mu(i)$.
12. **Go to 7**.

3. Estimation corridor

We denote $\lambda(h, H)$ as a maximal number of disjoint subsets that one can get from the initial A in order to a sum of weights in each subset would belong to a range $[h, B]$. As a problem of finding of $\lambda(h, H)$ is NP-hard in the strong sense, we will find an upper bound $\nu(h, H) \ge \lambda(h, H)$. Below we give a recursive algorithm **A4** to build $\nu(h, H)$.

Algorithm A4
$A' := A, A^+ := \emptyset, s := 0, z_0 := 0, \nu(h, H) := 0$.
For $x = h$ **To** H
    y:=0
    For $k = 1$ **To** n
        If ($\exists A'' \subseteq A'$: $h \le \sum_{a_j \in A''} s(a_j) + s(a_k) \le x$)
        { $y := y + s(a_k)$, $s := s + s(a_k)$, $A' := A' \setminus a_k$, $A^+ := A^+ \bigcup a_k$ }.
    End
    $\lambda := \lfloor y/x \rfloor$.
    While ($\lambda > 0$)
        If ($P(H, x, s, \lambda) = 0$) { $\lambda := \lambda - 1$ } else **Break** While.
    End While
    $\nu(h, H) := \nu(h, H) + \lambda, z_x := \lambda$.
End
STOP

Algorithm $P(H, x, s, \lambda)$
1. $K := \nu(h, H) + \lambda$, $A := A^+, M := K + 1$.
2. $B_i^{max} := x$, $B_i^{min} := h$, $i = 1, 2, \dots K$,
$B_{K+1}^{max} := s - \lambda x - \sum_{i=1}^{x-1} z_i i$, $B_{K+1}^{min} := max\{s - Kx, 0\}$.
3. **If** (**Algorithm B(0)** $= 0$) **return** 0.

4. **If** (**Algorithm B(1)** $= 0$) **return** 0.
5. **return** 1.

Now we define:

An operator $P^+(h,H,x) = W^+ = \{w_i^+ \circ k_i^+\}$, where $w_i^+ = H - i + 1$, $k_i^+ = \nu(H-i+1,H) - \nu(H-i+2,H),\ i \in [1,p],\ k_p^+ = x - \nu(H-p+2,H)$, $k_p^+ < \nu(H-p+1,H) - \nu(H-p+2,H),\ \sum_{i=1}^{p} k_i^+ = x,\ \nu(H+1,H) := 0$, a sum $S^+(h,H,x) = \sum_{i=1}^{p} k_i^+ w_i^+,\ w_p^+ < h \Rightarrow S^+(h,H,x) := 0,\ \ C^+ = \{C_j^+\}$ as

$C_j^+ = H, \qquad j \in \left[1, k_1^+\right],$
$C_j^+ = H - 1, \qquad j \in \left[k_1^+ + 1, k_1^+ + k_2^+\right], \ldots$
$C_j^+ = H - p + 1,\ j \in \left[\sum_{i=1}^{p-1} k_i^+ + 1, \sum_{i=1}^{p} k_i^+\right].$

An operator $P^-(h,H,x) = W^- = \{w_i^- \circ k_i^-\}$, where $w_i^- = h + i - 1$, $k_i^+ = \nu(h+i-1,h) - \nu(h+i-2,h),\ i \in [1,p],\ k_p^- := x - \nu(h+p-2,h)$, $k_p^- < \nu(h+p-1,h) - \nu(h+p-2,h),\ \sum_{i=1}^{p} k_i^- = x,\ \nu(h-1,h) := 0$, a sum $S^-(h,H,x) = \sum_{i=1}^{p} k_i^- w_i^-,\ w_p^- > H \Rightarrow S^-(h,H,x) := \infty,\ \ C^- = \{C_j^-\}$ as

$C_j^- = h + p - 1,\ \ j \in \left[1, k_1^-\right],$
$C_j^- = h + p - 2,\ \ j \in \left[k_1^- + 1, k_1^- + k_2^-\right], \ldots$
$C_j^- = h, \qquad j \in \left[\sum_{i=1}^{p-1} k_i^- + 1, \sum_{i=1}^{p} k_i^-\right].$

Algorithm A5 to build the corridor $[C^{min}, C^{max}]$.
1. $C_i^{min} := B_i^{min},\ C_i^{max} := B_i^{max},\ i = 1, 2, \ldots M$
2. $i := 0,\ REP := 0.$
3. $i := i + 1$. **If** ($i > M$) **Go to** 6. $\ g := C_i^{max}$.
4. $C_i^{max} := \max_h \{\ C_i^{min} \le h \le min(g, C_{i-1}^{max}) : C_j^- \le C_j^{max},\ j = 1, 2, \ldots i-1,$ $\sum_{j=1}^{i} C_j^- + \sum_{j=i+1}^{M} C_j^{min} \le S(A)\ \}$, where $C^- = \{C_1^-, C_2^-, \ldots C_i^-\} = P^-(C_i^{max}, C_1^{max}, i)$.
5. **If** ($C_i^{max} < g$) $REP := 1$. **Go to** 3.
6. $i := M + 1$.
7. $i := i - 1$. **If** ($i < 1$) { **If** ($REP = 1$) **Go to** 2 else **STOP** }. $g := C_i^{min}$.
8. $C_i^{min} := \min_h \{\ C_i^{min} \le h \le C_i^{max}\} : C_{i+j}^+ \ge C_{i+j}^{min},\ j = 1, 2, \ldots M - i,$ $\sum_{j=1}^{i-1} C_i^{max} + \sum_{j=1}^{M-i+1} C_i^+ \ge S(A)\ \}$, where $C^+ = \{C_1^+, C_2^+, \ldots C_{M-i+1}^+\} = P^+(C_M^{min}, C_i^{min}, M - i + 1)$.
9. **If** ($C_i^{min} > g$) $REP := 1$. **Go to** 7.

4. Building the bounds for the 1DBP class

Here we give building the bounds only for the first 7 models from the 1DBP class.

Model 0.
a. **Lead** (A, L) to (A', L') using the algorithm **B(0)**. **Set** $A := A'$ and $L := L'$.
b. **Answer** is a result of **B(0)**, where $B_i^{max} := B_i,\ i = 1, 2, \ldots M,\ \ M = |L'|$.

Model 1.
a. **Lead** A to A^0 using the algorithm **A.Set** $A' := A \setminus A^0$.
b. **Answer** is $LB(A) := P(A')$.

Model 2.

For $M = \lfloor S(A)/B \rfloor$ **To** 1 **By** -1

a. **Set** $B_i^{min} := B$, $B_i^{max} := S(A)$, $i = 1, 2, \ldots M$.

b. **If (B(0)=0)** continue For. **If (B(1)=0)** continue For.

c. **Answer** is $UB(A) := M$. **STOP**.

End

Model 3.

For $M = \lfloor S(A)/B \rfloor$ **To** $S(A)$

a. **Set** $B_i^{min} := B_{min}$, $B_i^{max} := B_{max}$, $i = 1, 2, \ldots M$.

b. **If (B(0)=0)** continue For. **If (B(1)=0)** continue For.

c. **Answer** is $LB(A) := M$. **STOP**.

End

Model 4.

For $M = \lfloor S(A)/B \rfloor$ **To** 1 **By** -1

a. **Set** $B_i^{min} := B_{min}$, $B_i^{max} := B_{max}$, $i = 1, 2, \ldots M$.

b. **If (B(0)=0)** continue For. **If (B(1)=0)** continue For.

c. **Answer** is $UB(A) := M$. **STOP**.

End

Model 5.

For $B = \lceil S(A)/M \rceil$ **To** $S(A)$

a. **Set** $B_i^{min} := w_m$, $B_i^{max} := B$, $i = 1, 2, \ldots M$.

b. **If (B(0)=0)** continue For.

c. **Answer** is $LB(A) := B$. **STOP**.

End

Model 6.

For $T = 1$ **To** $S(A)$

a. **Set** $B_i^{min} := w_m$, $B_i^{max} := \tau_i T$, $i = 1, 2, \ldots M$.

b. **If (B(0)=0)** continue For.

c. **Answer** is $LB(A) := T$. **STOP**.

End

5. Experimental results

Our program is written in Microsoft Visual C++. We performed our experiments on computer Intel/Core2 Duo/E6400 2,13GHz,2Gb RAM. Here we present our results only for the Model 1. We used a fastest mode to get A^0 and a lower bound $LB(A) = M_0 + \lceil S(A')/B \rceil$, $A' = A \setminus A^0$, using the algorithm **A**. Our purpose was to evaluate a quality of $LB(A)$. We developed new fast approximation algorithm **FG** to get the upper bound $FG(A)$ to $OPT(A)$ and measure of quality of $LB(A)$ (and $FG(A)$ too) as $p = \frac{FG(A)-LB(A)}{LB(A)} 100\%$.

Algorithm FG

1. Find A^0 and M_0 by algorithm **A**. Set $A'' := A \setminus A^0$.
2. Set the initial $a := \lceil S(A'')/B \rceil$ and $b := FFD(A'')$.
3. **While** ($b > a$)
4. Set $A := A''$, $M := a + (b-a)/2$, $L := \{B_i\}$, $B_i = B, i = 1, 2, \ldots M$.
5. **If** ($\mathbf{B(0)} = 0$) { Set $a := M + 1$; **Continue** While }
6. Lead $(A, L) \to (A', L')$ during **B**, set $A := A'$, $L := L'$, $n := |A'|$, $M' := |L'|$.
7. Find $k_0 = \max\limits_i \{ s(a_i) + s(a_{i+1}) + s(a_n) > B \}$, set $B_{M'-i+1} := B - s(a_i)$, $i = 1, 2, \ldots k_0$.
8. **For** $k = k_0 + 1$ **To** n
9. **For** $i = M'$ **To** 1 **By** -1
10. Set $gap := B_i - s(a_k)$. **If** ($gap < s(a_n)$) **continue** For i
11. **If** ($B_i < B$) { Find a maximal $s(a_p) \leq gap$,
13. Set $A := A \setminus \{a_k \bigcup a_p\}, L := L \setminus B_i$ }.
14. **else** { Set $B_i := B - s(a_k)$, $A := A \setminus a_k$ }.
15. Sort L by decreasing: $B_i \geq B_{i+1}$, **Break** For i (continue For k)
16. **End** For i
17. **If** ($B_i - s(a_k) < s(a_n)$) for all $i \in [1, M']$ { Set $a := M + 1$, **continue** While}.
18. **End** For k
19. Set $b := M$, **continue** While.
20. **End** While
21. Set $FG(A) := M_0 + b$. **STOP**.

The test instances we created using a random distribution generator $BS(\rho B, B/2, B, m)$ from [1]. The first series of experiments was for the ranges $(\rho, B/2]$, $\rho = 0.25, 0.24, \ldots 0.16$. Our parameters were $m = n = 10000$ and $B = 10^9$. We denote $M^0 = \lceil S(A)/B \rceil$.

ρ	$\frac{\lvert A^0 \rvert}{n} 100\%$	$\frac{LB(A)-M^0}{M^0} 100\%$	$\frac{FG(A)-LB(A)}{LB(A)} 100\%$	$\frac{FFD(A)-LB(A)}{LB(A)} 100\%$	$FG(A)$ time sec
0.25	34.66	3.933	0.102	6.903	176
0.24	26.32	2.500	0.185	7.264	176
0.23	24.34	2.186	0.160	6.873	188
0.22	19.90	1.439	0.109	6.848	435
0.21	14.80	0.901	0.391	6.788	1037
0.20	10.48	9.458	0.485	6.500	1180
0.19	7.12	0.232	0.695	6.000	843
0.18	4.12	0.059	0.823	5.530	1565
0.17	1.48	0.000	1.040	4.070	1506
0.16	0.00	0.000	1.240	4.540	4075

The second series of experiments was for a range $(B/4, B/2]$, $B = 10^9$ and $m = n \in [10000, 50000]$.

$m = n$	10000	15000	20000	25000	30000	35000	50000
$LB(A)$ time sec	0	1	1	3	3	4	8
$FG(A)$ time sec	176	757	1850	2387	7690	7894	24655

The third series of experiments was for a range $(B/4, B/2]$, $B = 10^6$, $B = 10^9$ and $m = n = 1000, 1500, 2000$. For $m = n = 1000 \wedge B = 10^6$ we generated the 18 test instances, for $m = n = 1500 \wedge B = 10^9$ the 6 test instances and for $m = n = 1500 \wedge B = 10^9$ the 6 test instances. We wish to know how often can appear the optimal solutions. We saw the optimal solutions arrived about in the 50% cases for $m = n = 1000$ and less 50% for the others $m = n$.

The forth series of experiments was for a range $(B/4, B/2]$ and $m = n = 700$. Here we wished to know about $k = LP(A)/LP(A')$, where we recall $A' = A \setminus A^0$. We sent a query to both G.Belov and G.Scheithauer from Dresden Technology University (DTU) to solve the 6 test instances. The first instance $A(1)$ was for $B = 10000$. The second instance was $A(2) := A'(1) = A(1) \setminus A^0(1)$ with $|A(1)^0| = 338$ (the 119 dominate pairs). The third instance $A(3)$ was for $B = 50000$. The forth was as $A(4) := A'(3) = A(3) \setminus A^0(3)$ with $|A^0(3)| = 208$ (the 104 dominate pairs). Fifth instance was for $B = 50000$. The sixth instance was as $A(6) := A'(5) = A(5) \setminus A^0(5)$ with $|A^0(5)| = 226$ (the 113 dominate pairs). We got the following results from DTU to our query:

$LP(A(1)) = 1774$ sec, $LP(A(2)) = 925$ sec, $k = 1774/\ 925 = 1.9178.$
$LP(A(3)) = 1935$ sec, $LP(A(4)) = 1086$ sec, $k = 1935/1086 = 1.7818.$
$LP(A(5)) = 1952$ sec, $LP(A(6)) = 1013$ sec, $k = 1952/1013 = 1.9269.$

Thus, for these 6 instances we have a result: an LP-time(A') of the reduced instances faster about 2 times than an LP-time(A) of the original instances.

REFERENCES

1. Applegate D., Buriol L., Dillard B., Johnson D. and Shor P. The Cutting-Stock Approach to Bin Packing: Theory and Experiments, In Proceedings of the Fifth Workshop on Algorithm Engineering and Experimentation, R.E. Ladner (Editor), SIAM, 2003, pp. 1-15.

2. Fedulov G. One-dimensional bin packing class: fast algorithms to find the bounds of objective functions, Georgian Engineering News, **2**, (2008), 42-47.

3. Fukunaga A. Bin Completion Algorithms for Multicontauner Packibg, Knapsack, and Covering Problems, Journal of Artifical Intelligence **28**, (2007), 393-429.

4. Garey M. and Johnson D. Computers and intractability: A Guide to the theory of NP-Completeness, W.H. Freeman and Company, 1979.

5. Johnson D. Fast Algorithm for Bin Packing, Journal of Computer and System Sciences, **8**, (1974), 272-314.

6. Martello S. and Toth P. Knapsack Problems, John Wiley and Sons, 1990.

7. Scheithauer G. and Terno J. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. Oper. Res. Lett., **20**, (1997), 93-100.

Received: 25.06.2008; revised: 27.10.2008; accepted: 26.11.2008.