Pergamon

PII: S0305-0548(96)00082-2

Computers Ops Res. Vol. 24, No. 7, pp. 627–645, 1997
© 1997 Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0305-0548/97 $17.00+0.00

BISON: A FAST HYBRID PROCEDURE FOR EXACTLY SOLVING THE ONE-DIMENSIONAL BIN PACKING PROBLEM

Armin Scholl†‡, Robert Klein§ and Christian Jürgens¶

Institut für Betriebswirtschaftslehre, Technische Hochschule Darmstadt, Hochschulstrasse 1, D-64289 Darmstadt, Germany

(Received May 1996; in revised form October 1996)

Scope and Purpose—Packing of items into boxes or bins is a recurring task in distribution and production. Concerning the size and shape of items as well as the form and capacity of bins, a large variety of different packing problems can be distinguished {see for example Dyckhoff and Finke, *Cutting and Packing in Production and Distribution* (1992) [1]}. Similar problems concern the cutting of pieces into particular smaller ones so as to minimize the wastage of material and scheduling of identical parallel processors so as to minimize the total completion time. We consider a basic packing problem which is known as the one-dimensional bin packing problem {BPP-1, see Martello and Toth, *Knapsack Problems* (1990) [2]}. It is to pack a given set of items of different sizes into a minimum number of equal-sized bins. Even though this problem seems to be simple it is NP-hard, i.e. no procedure is able to solve each problem instance in polynomial time. In order to solve more general (and more complex) realistic problems within a reasonable computation time it is important to have available effective procedures for the basic problem. We propose an exact solution procedure for BPP-1 (called BISON) which is composed of different known and new bound arguments and reduction procedures, several heuristics, and a branch and bound procedure. Among the heuristics, an application of the successful meta-strategy tabu search is most effective. The branch and bound procedure contains a new branching scheme (local lower bound method) which is of general interest for capacity constrained problems. Computational results indicate that BISON is very effective and clearly outperforms the well-known branch and bound procedure MTP (from Martello and Toth's 1990 work [2]). Even if MTP runs for a multiple of the computation time given to BISON it cannot achieve competitive results.

Abstract—In this paper, we consider the well-known one-dimensional bin packing problem (BPP-1), which is to pack a given set of items having different sizes into a minimum number of equal-sized bins. For solving BPP-1, an exact hybrid solution procedure, called BISON, is proposed. It favourably combines the well-known meta-strategy tabu search and a branch and bound procedure based on known and new bound arguments and a new branching scheme. Computational results indicate that BISON is very effective and outperforms existing approaches. © 1997 Elsevier Science Ltd

1. PROBLEM DESCRIPTION

The one-dimensional bin packing problem is defined as follows. A set $J=\{1,..., n\}$ of n indivisible items, each of which has a specific size or weight w_j ($j=1,..., n$), is given. Without relevant loss of generality, we assume these weights to be integral. Each item has to be packed into one of m bins all having the same capacity c. The total weight of items contained in any bin must not exceed the capacity. Therefore, a necessary condition for feasibility is $w_j \leq c$ for all j. Concerning the objective, two versions of the problem can be distinguished.

- BPP-1: Minimize the number m of bins for a given bin capacity c.
- BPP-2: Minimize the capacity c for a given number m of bins.

Both problem versions are NP-hard optimization problems [3]. BPP-2 is equivalent to the problem of scheduling n independent jobs having operation times w_j on m identical parallel processors with the objective of minimizing the makespan c [4,5]. Between BPP-1 and BPP-2 there is a strong relationship which may be interpreted as some type of duality [6]. Both problems are based on the decision problem of finding out whether or not a feasible solution exists for a given combination of c and m. This dual relationship is favourably utilized in our new hybrid procedure for BPP-1. Another problem which can

† To whom correspondence should be addressed (email: scholl@bwl.bwl.th-darmstadt.de).

‡ Armin Scholl is an Assistant Professor at the Technische Hochschule Darmstadt (THD), Germany. He received a doctoral degree in Business Administration at the THD with a thesis on assembly line balancing. His research interests include production planning, routing and educational software.

§ Robert Klein is a research assistant at the THD where he studied and received a graduate degree in Computer Science and Business Administration. His research interests lie in the fields scheduling, routing and efficient algorithms.

¶ Christian Jürgens studied at the THD and received a graduate degree in Computer Science and Business Administration.

be seen as being dual to BPP-1 is proposed in ref. [7]. An extension of BPP-1 with an additional capacity restriction is called two-dimensional vector packing problem [8].

The remainder of the paper is organized as follows. In Section 2, we give an outline of exact and heuristic procedures proposed for solving BPP-1 so far. The new hybrid procedure BISON is presented in Section 3. Results of comprehensive computational experiments are reported in Section 4. The paper is closed by a summary and some remarks in Section 5.

2. SURVEY OF SOLUTION PROCEDURES FOR BPP-1

Most literature references on BPP-1 are concerned with approximation algorithms and their average or worst-case performance. These algorithms are usually based on very simple planning strategies and often provide satisfactory results. An excellent survey is given in ref. [3]. Up to now, only a few exact procedures are known [2]. In the following, we give brief outlines of both types of algorithms.

2.1. Approximation algorithms

Two basic classes may be distinguished with respect to the order in which items are packed [3]. *On-line algorithms* consider the items in the given order and pack each of them into a bin chosen due to a particular selection strategy. *Off-line algorithms* first build an order of the items according to some priority criterion. Usually, these procedures sort the items in non-increasing order of their weights and in this order assign them to the bins selected. Off-line algorithms require that the problem data are static, while the first ones may be applied when items dynamically arrive at a packing station.

In either class of procedures, the following main *selection strategies* are used [2,3]:

- *Next-fit.* The first item is packed into bin 1. Each further item is packed into the same bin as the preceding item, if the residual capacity of the bin is sufficient. Otherwise, it is packed into a new bin. The on-line version (NF) has time complexity $O(n)$ and the off-line version (next-fit-decreasing, NFD) takes time $O(n \log n)$ due to the sorting of items.
- *First-fit.* In each step, the current item is assigned to the (partially filled) bin with the smallest index which has sufficient residual capacity. If no such bin is available, the item is packed into a new bin. The on-line version FF takes time $O(n \log n)$ if a special data structure is used for storing the residual capacities of the bins. The off-line version FFD (first-fit-decreasing) has the same time complexity. Obviously, FF and FFD always find solutions at least as good as those found by NF and NFD, respectively.
- *Best-fit.* In each step, the current item is assigned to that partially filled bin which has the smallest sufficient residual capacity. Ties are broken in favour of the bin with the smallest index. In absence of such a bin, a new one is initialized. The time complexity of the on-line version BF and the off-line version BFD (best-fit-decreasing) is $O(n \log n)$.
- *Worst-fit.* In opposite to best-fit, a partially filled bin with the largest sufficient residual capacity is selected. If none exists, a new bin is initialized. Obviously, the time complexities of the on-line version WF and the off-line version WFD (worst-fit-decreasing) are identical with that of BF and BFD, respectively.

Many further planning strategies most of which are extensions of the above ones have been developed [3]. For example, the procedure best-two-fit (B2F) works as FFD until a bin is completed, i.e. the residual capacity is lower than the weight of the lightest free (unassigned) item. In this case, B2F tries to exchange the lightest item contained in this bin for two lighter free items such that the wasted residual capacity gets as small as possible. B2F takes time $O(n^2)$ due to the additional improvement step.

These heuristics have comprehensively been examined with respect to their average- and worst-case performance [9–12]. Among the procedures described above, FFD and BFD show the best worse-case behaviour. They have an *asymptotic worst-case performance ratio* of 1.222. That is, the objective function value obtained by either procedure does not deviate from the optimal value by more them 22.2% provided that the number of bins is sufficiently large. This performance ratio gets even better if the maximal weight of the items declines relative to the bin capacity. Some modified procedures show slightly improved worst-case performance ratios. Even though the worst-case errors are rather small, such possible deviations are not acceptable when additional bins are expensive. Therefore, for evaluation purposes, it is desirable to have exact solution procedures which quickly compute optimal solutions.

2.2. *Exact procedures*

In contrast to heuristics, only a few exact algorithms have been presented for BPP-1. The first one has been proposed in ref. [13]. It is a simple branch and bound procedure with a depth-first search and a branching strategy, which selects items in non-increasing order of their weights. In each node of the enumeration tree, alternative subproblems are generated by assigning the selected item to all initialized bins in increasing order of their residual capacities if it fits. Furthermore, a new bin is initialized by assigning the current item to it. That is, the branching scheme follows the logic of the BFD procedure (see Section 2.1). For fathoming nodes, only the following simple lower bound is used. It is obtained by assuming that each item can be split up between two or more bins ($\lceil x \rceil$ denotes the smallest integer being larger than or equal to x):

$$LB_1 = \lceil \sum_{j=1}^{n} w_j / c \rceil$$

The branch and bound procedure of ref. [14], originally described for a more general problem with different bin capacities, performs a depth-first search and uses a branching strategy similar to that in the procedure of ref. [13]. Additionally, it ensures that within every set of equivalent solutions only one is explicitly enumerated. In this context, two solutions are *equivalent* if for each bin in one solution there is another bin in the other solution which contains the same set of weights. That is, the procedure utilizes the fact that often several items with different indices may have the same weight. The lower bound LB_1 is computed in the root node and locally in nodes of the enumeration tree whenever a bin is filled *maximally*, i.e. no further item fits into this bin.

The most effective procedure currently available is the branch and bound algorithm MTP of ref. [2]. We give an outline of its different components since some of them are also used in our procedure BISON, and MTP is used as a reference procedure in our numerical experiments.

2.2.1. The procedure MTP. In the following description of the procedure MTP, we assume that the items are indexed in non-increasing order of their weights ($w_1 \geq w_2 \geq, \ldots, \geq w_n$). MTP applies a reduction procedure and lower bounds LB_2 and LB_3 which are explained before we address the branching scheme.

Reduction procedure MTRP

In order to reduce the problem data the following *dominance criterion* is defined:

> A set I of items is called a *packing* if $\Sigma_{j \in I} w_j \leq c$. A packing I_1 is said to *dominate* another packing I_2 if there exist a partition of I_2 into subsets $S_1, \ldots, S_q$ and a subset $\{j_1, \ldots, j_q\}$ of I_1 such that $w_{j_h} \geq \Sigma_{k \in S_h} w_k$ is true for $h = 1, \ldots, q$. Different packings are *equivalent* to each other if they contain the same weights. Among such packings one (e.g. the first one considered) can arbitrarily be defined to dominate the others.

The definition of the dominance ensures that any solution including a (dominated) packing I_2 can easily be transformed into a solution with the same number of bins containing the (dominating) packing I_1 by exchanging each subset S_h for the corresponding item j_h. For example, consider six items with weights 4, 3, 3, 2, 2, 1 and let $c = 10$. The packing $I_2 = \{1, 4, 5, 6\}$ is dominated by $I_1 = \{1, 2, 3\}$, since items 4 and 6 can be replaced by item 2 and item 5 by item 3.

The reduction procedure MTRP utilizes the dominance relationship as follows: if a packing I which contains some item j dominates *all* other packings containing j too, the problem data is reduced by *fixing* the packing I, i.e. assigning all items of I to a (new) bin.

Since it is computationally too expensive to examine all possible dominance relationships, MTRP (arbitrarily) considers only packings that contain no more than three items. Furthermore, the procedure is restricted to some simple cases of dominance which can be tested efficiently. The items are considered in their index order. If one of the simple tests can identify a packing containing j to dominate all the others with j, it is fixed reducing the problem data. A more detailed explanation of the reduction procedure and its efficient implementation is given in ref. [2]. Its time complexity is of order $O(n^2)$.

Remark 1. Obviously, MTRP is poor whenever the item weights are much less than the bin capacity. However, to any problem instance with $w_n \geq c/3$, i.e. each item fills at least one-third of a bin, MTRP finds an optimal solution. Since at most two items can share a bin (except the simple case where three items with weight $c/3$ are combined), it is easy to show that the reduction rules of MTRP are sufficient. That is, such problem instances are special cases of BPP-1 which can be solved in polynomial time.

The application of the reduction procedure is two-fold. First, it can be utilized for reducing the size of the problem instance before an exact procedure is applied. Sometimes it is even possible to find an optimal solution by the reduction procedure only. Second, it is used for computing a lower bound LB_3 as described below.

Lower bound LB_2

It is based on partitioning the set of items into three subsets according to an integral parameter $\alpha \in [0, c/2]$:

$$J_1 = \{j \in J | w_j > c - \alpha\},\ J_2 = \{j \in J | c - \alpha \geq w_j > c/2\},\ J_3 = \{j \in J | c/2 \geq w_j \geq \alpha\}$$

A bound $L(\alpha)$ on the number of bins required for packing the items contained in those sets can be computed as follows ($|J|$ denotes the cardinality of a set J):

$$L(\alpha) = |J_1| + |J_2| + \max\left\{0, \left\lceil \frac{\sum_{j \in J_3} w_j - \left(|J_2|c - \sum_{j \in J_2} w_j\right)}{c} \right\rceil\right\}$$

The items in J_1 and J_2 cannot be combined with any other item in those sets, because they fill more than half a bin each. Therefore, the number of items in both sets is a lower bound on the number of bins required. Items in J_3 can only be combined with items in J_2. In case that the sum of weights of items in J_3 is larger than the sum of residual capacities in bins occupied by items in J_2, the bound can be increased by the far-right term which applies the logic of LB_1 to this weight difference.

The lower bound LB_2 is obtained by computing the maximal $L(\alpha)$ with $\alpha \in [0, c/2]$:

$$LB_2 = \max\left\{ L(\alpha) \,\middle|\, 0 \leq \alpha \leq \frac{c}{2} \right\}$$

Martello and Toth [2] prove that this bound can be computed efficiently in time $O(n)$ by successively considering only distinct w_j and 0 as values of the parameter α. Furthermore, it is shown that the inequality $LB_2 \geq LB_1$ holds due to $L(0) = \max\{|J_2|, LB_1\}$. Obviously, the application of LB_2 only makes sense when at least some weights exceed the value $c/2$. Therefore, MTP contains an additional simple bound argument LB'_2 which also takes time $O(n)$. For computing LB'_2, the number of items with their weights exceeding c/i is counted and divided by $i-1$, because at most $i-1$ of these items can share a bin (with $i = 2, \ldots, \lfloor c/w_n \rfloor + 1$). Each value is rounded up to the next integer and serves as a lower bound on the number of bins. The maximal value is used as bound LB'_2.

Lower bound LB_3

Let P be the data of the original problem instance. Being applied to P, MTRP fixes p_1 packings reducing P to become a residual instance P_1. Applying LB_2 to P_1 gives a lower bound $L'_1 = p_1 + LB_2(P_1)$ on the total number of bins, where $LB_2(P_1)$ denotes the result of applying the bound argument LB_2 to an instance P_1. A further bound is obtained as follows: P_1 is relaxed by deleting the lightest item. By applying MTRP to the relaxed instance, p_2 further packings are fixed, leaving a residual instance P_2. Then a lower bound for the original problem is given by $L'_2 = p_1 + p_2 + LB_2(P_2)$. This process is repeated while producing bounds $L'_h = p_1 + p_2 + \cdots + p_h + LB_2(P_h)$ until the residual instance is empty. Thus, a lower bound LB_3 for the original problem is given as follows, where H denotes the number of iterations performed: $LB_3 = \max\{L'_1, \ldots, L'_H\}$.

Note that deleting the lightest item from the residual instance may lead to one or more additional packings being fixed by MTRP. Whenever the idle capacity caused by these packings is larger than the weights of the items deleted, a stronger bound may be obtained. Since in each iteration at least one item is removed from the residual instance and MTRP takes time $O(n^2)$, the bound LB_3 can be computed in time $O(n^3)$. It is easy to verify that $LB_3 \geq LB_2$.

Branch and bound procedure

MTP applies a depth-first search with its branching scheme following the FFD strategy (see Section 2.1). In each node of the enumeration tree, alternative subproblems are successively obtained at each (re)visit of the node by assigning the heaviest free (unassigned) item to the bins having sufficient residual capacity in increasing order of their indices and to a new bin. When visiting a node for the first time, MTRP and the lower bound arguments LB_2 and LB_3 are applied. For the computation of the bounds the current subproblem of this node is relaxed as follows: any packing I already built by MTRP or branching defines a fictitious item with its weight being equal to the total weight of the items contained in I. Those extra items are considered together with the items currently being free.

A node is fathomed when one of its lower bounds is larger than or equal to the upper bound UB, which is the objective function value of the current incumbent solution. If no fathoming is possible, the heuristics FFD, BFD and WFD are applied to complete the partial solution in order to improve UB and possibly fathom the node. Any unfathomed node is being branched as outlined above. A simple dominance criterion is additionally used in order to restrict the number of subproblems to be generated. This criterion prevents building packings in which the lightest item is replaced by a single lighter one, because those packings are dominated by the original one.

3. THE PROCEDURE BISON

In this section. we describe a new exact procedure for BPP-1 which is called BISON (**bi**n packing **solution** procedure). This hybrid procedure contains some of the components already successfully utilized by MTP and some interesting new components.

3.1. *Additional lower bounds*

We introduce some further bounds for BPP-1 which partly utilize the relationship of BPP-1 to other combinatorial optimization problems. Related problems appear in capacitated vehicle routing, assembly line balancing, set partitioning as well as scheduling of parallel processors [15]. Furthermore, BPP-1 is related to any problem of the knapsack type [2]. Therefore, several bounds for such problems are either directly applicable to BPP-1 or need minor modifications. In the following, it is assumed again that items are indexed in non-increasing order of their weights ($w_1 \geq w_2 \geq \ldots \geq w_n$).

Lower bound LB_4. This bound which is directly applicable to BPP-1 has been described for a capacitated vehicle routing problem [16] and for a simple assembly line balancing problem [17]. A restricted version of the bound has been presented for BPP-1 [18]. LB_4 is based on ideas similar to those within LB_2 and considers three sets of items:

$$J_1 = \{j \in J | w_j > c/2\},\ J_2 = \{j \in J | c/2 \geq w_j > c/3\},\ J_3 = \{j \in J | w_j \leq c/3\}$$

The items in J_1 cannot be combined with each other. Therefore, $L_1 = |J_1|$ bins are necessary to pack those items.

Any item in J_1 can be combined with at most one item in J_2. In order to determine the maximal number of items out of J_2 that can be combined in this way, the following procedure is applied. The items in J_1 and J_2 are assumed to be indexed in non-increasing and non-decreasing order of their weights, respectively. The items in J_2 are considered according to this order. Each item j is combined with the lowest-indexed item in J_1 which leaves sufficient residual capacity. This process is repeated until no item of J_2 is left or the current item j cannot be combined with one of the remaining items in J_1. In the latter case, a set J'_2 of items remains. Since at most two of these items can share a bin at least $L_2 = \lceil |J'_2|/2 \rceil$ additional bins are required.

Generally, the items in J_3 can be combined with items in J_1 and J_2. In order to examine the possible combinations, different subsets of J_1 and J_3 are defined depending on parameter values $\alpha \in \{w_j | j \in J_3\}$: $J_1(\alpha) = \{j \in J_1 | w_j \leq 1 - \alpha\}$ and $J_3(\alpha) = \{j \in J_3 | w_j \geq \alpha\}$.

Lower bounds $L_3(\alpha)$ on the number of bins additionally required by the items in $J_3(\alpha)$ are computed as follows:

$$L_3(\alpha) := \frac{\sum_{j \in J_1(\alpha) \cup J_2 \cup J_3(\alpha)} w_j}{c} - |J_1(\alpha)| - L_2$$

Using these values, a lower bound LB_4 is given by

$$LB_4 := \lceil L_1 + L_2 + \max\{0, \max\{L_3(\alpha) | \alpha \in \{w_j | j \in J_3\}\}\} \rceil$$

Remark 2. Considering the values of α in decreasing order, each item has to be added to set $J_1(\alpha)$ or $J_3(\alpha)$, respectively, at most once. Therefore, the bound LB_4 can be computed in time $O(n)$ [15]. In case of $w_n > c/3$, at most two items can form a packing and LB_4 provides an optimal number of bins (see

Remark 1).

The bound arguments LB_1 to LB_4 try to find a lower bound by analysing relaxed versions of a problem. Another way is to improve on a given lower bound value LB by restricting the problem according to the objective function value LB and then relaxing it. This technique, which we call *destructive bound computation*, because it tries to disprove the feasibility of a possible objective function value, is utilized by the following bound arguments.

Lower bound LB_5. It utilizes the close relationship between BPP-1 and BPP-2 [15]. Let $L_C(m)$ be an arbitrary lower bound on the capacity for a BPP-2 instance with given number m of bins. A lower bound LB_5 for BPP-1 is determined by the minimal number of bins for which the BPP-2 bound does not exceed the given capacity c:

$$LB_5 = \min\{m \mid L_C(m) \le c\}$$

LB_5 is computed starting with a valid lower bound, e.g., the maximal value of LB_1 to LB_4. This value is increased by one until the capacity bound obtained for the respective BPP-2 instance is lower than or equal to c. A simple example of a bound argument for BPP-2 is as follows:

$$L_C(m) := \max\left\{ \sum_{i=0}^{k} w_{k\cdot m+1-i} \mid k=1,\ldots,\lfloor (n-1)/m \rfloor \right\}$$

If we assume that the first (heaviest) $m+1$ items are to be packed, at least one bin gets two or more items. Therefore, $w_m + w_{m+1}$ is a lower bound on the bin capacity for this reduced problem. If the first $2\cdot m+1$ items are considered, then into at least one bin three or more items must be packed with a minimal capacity requirement of $w_{2m-1}+w_{2m}+w_{2m+1}$. In general, at least one packing must contain $k+1$ or more of the first $k\cdot m+1$ items providing a lower bound $\sum_{i=0}^{k} w_{k\cdot m+1-i}$.

Lower bound LB_6. We consider the special case $w_n > c/4$ for which we may partition the set of items into the following disjoint subsets:

$$J_1 = \{j \in J \mid w_j \ge 3c/4\},\ J_2 = \{j \in J \mid 3c/4 > w_j > c/2\},\ J_3 = \{j \in J \mid c/2 \ge w_j > c/3\},\ J_4 = \{j \in J \mid c/3 \ge w_j > c/4\}$$

The items in J_1 cannot be combined with any other item and require a bin of their own. Each item in J_2 can be combined with at most one item out of $J_3 \cup J_4$. Items in J_3 may form a packing alone, together with at most one item out of $J_2 \cup J_3$ or at most two items out of J_4.

Starting with a lower bound LB, which may be determined as the maximal value of LB_1 to LB_5, we try to improve this value by disproving that LB bins are sufficient. We propose a simple procedure which utilizes the special problem structure for reducing the problem data and possibly enlarging the value of LB. As stated earlier, the items are indexed in non-increasing order of their weights. The procedure contains the following steps:

(1) Let k be the number of items in J_1. If $k>0$, then assign each of the items $1,\ldots,k$ to a separate bin, because it cannot share a bin with any other item.

(2) If J_2 is empty, set $p=k$ and go to step (3). Otherwise, let p be the highest index of all items in J_2, i.e. $J_2=\{k+1,\ldots,p\}$. Assign each of the items $k+1,\ldots,p$ to a separate bin. Consider these bins in reverse order. In each step, the heaviest item out of $J_3 \cup J_4$ which fits into the current bin (if any exists) is assigned to the bin and removed from J_3 or J_4, respectively. If no items remain in $J_3 \cup J_4$ an optimal solution to the original problem is found.

(3) Now p bins are filled maximally i.e. no free item can be added, and $r \le n-p$ remaining items with $w_j \le c/2$ have to be packed into $LB-p$ bins. Due to $w_n > c/4$, at most three items can share a bin. In order not to use more than LB bins, at most $s = 3\cdot(LB-p)-r$ bins can contain less than three items. It is always possible to combine any two of the remaining items into a bin. Since all packings with only one item are dominated by other packings with two items, exactly s bins can be filled with two items and $LB-p-s$ bins with three items. In order to fill the first s bins as well as possible, the heaviest $2\cdot s$ items are arbitrarily packed into these bins and removed from J_3 or J_4, respectively. Note that only two items will be assigned to each of these bins even if an additional item fits. If $s=LB-p$, an optimal solution to the original problem is achieved. The same is true for $s=LB-p+1$ (i.e. $r=2\cdot(LB-p)-1$), where one of the $LB-p$ bins contains only one item.

(4) After step (3), $t=r-2\cdot s$ items and $b=LB-p-s$ bins remain. Due to $t=3\cdot b$, all bins must be filled

with exactly three items or *LB* bins are not sufficient. Assume that the remaining items are reindexed from 1 to t in non-increasing order of their weights and that the first q items are contained in J_3. In the following simple cases, it is possible to find an assignment of the remaining items to b bins, resulting in an optimal solution to the original problem:

(a) $q=0$. Each bin is filled with an arbitrary combination of three items since J_3 is empty.

(b) If one of the following sufficient conditions is true, it is easy to find an assignment according to the logic of the respective condition (further conditions are easy to find): $\max\{w_i+w_{2b+1-i} \mid i=1,\ldots, b\}+w_{2b+1}\leq c$, or $w_1+\max\{w_{b+i}+w_{t+1-i} \mid i=1,\ldots, b\}\leq c$, or $\max\{w_i+w_{t+1-i} \mid i=1,\ldots, b\}+w_{b+1}\leq c$.

(5) In the following cases, no solution with *LB* bins is possible and *LB* is increased by 1:

(a) $q>2b$ or $w_1+w_{t-1}+w_t>c$. At least one bin cannot contain three items.

(b) $q\in[b+1, 2b]$. At least $q-b$ bins must contain two items out of J_3 and one out of J_4. This is not possible if one of the following sufficient conditions is true: $\max\{w_{b-i}+w_{b+1+i} \mid i=0,\ldots, q-b-1\}+w_t>c$, or $w_{q-1}+w_q+w_{t+1-q+b}>c$.

(6) If neither steps (4) nor (5) have been successful in finding a solution or increasing *LB*, an exact procedure may be applied to this reduced problem in order to find out whether b bins are sufficient or not. In the first case, an optimal solution of the original problem is found. In the latter case, *LB* can be increased by 1. If the reduced problem is not sufficiently small, a time limit may be defined for the trial application of the exact procedure. Note that in some cases it may be easy to solve this restricted problem due to the necessity of finding packings with exactly three items each. If this is not possible within the given time limit, then the bound must not be increased.

(7) If *LB* has been increased in steps (5) or (6), the reductions performed in step (3) are cancelled and the steps (3)–(7) are repeated. Otherwise, the bound computation stops.

Whenever an optimal solution to the original problem is found in any step, the overall algorithm (BISON) stops.The bound argument LB_6 can also be applied to problem instances with $w_n\leq c/4$. Items j with $w_j\leq c/4$ are ignored in all steps but step (2), where some of them may be assigned to a bin whenever no item out of $J_3\cup J_4$ fits. However, LB_6 cannot provide an optimal solution to the original problem if not all of those items are packed into a bin during step (2).

Remark 3. Note that in case of $w_n>c/4$ the simple reduction possibilities of steps (1) and (2) are also recognized by the reduction procedure MTRP. Furthermore, step (2) is also contained in the computation of LB_4. The maximum expressions within the conditions of steps (4) and (5) are based on an optimal solution of a special version of BPP-2 with $2\cdot b$ items and b bins, each of which must contain exactly two items. The objective is to minimize the capacity of the bins. Provided that the items are indexed in non-increasing order of their weights, an optimal assignment is given by the packings

$$I_1=\{1,2\cdot b\},\ I_2=\{2,2\cdot b-1\},\ldots,I_b=\{b,b+1\}$$

with minimal capacity $\max\{w_i+w_{2b+1-i} \mid i=1,\ldots, b\}$. This result follows directly from the inequality below, because any other solution can be obtained by successively exchanging two items between different bins and no exchange reduces the capacity required ($i\leq p<q\leq j$):

$$\max\{w_i+w_j,\ w_p+w_q\}\leq\max\{w_i+w_q,\ w_p+w_j\}\leq\max\{w_i+w_p, w_q+w_j\}$$

3.2. Dominance rules

We describe some dominance rules used for fathoming nodes in the branch and bound procedure which are originally proposed for the related simple assembly line balancing problem [19]. Those rules are based on more simple dominance relationships between packings than used in MTRP. However, they are not restricted to packings with three items only. Furthermore, dominance relationships between partial solutions within an enumeration tree are exploited.

Dominance rule 1. As already mentioned, a packing is said to be filled *maximally* if no free item can be added without violating the capacity constraint. Non-maximal packings are dominated by maximal ones. Therefore, any node of an enumeration tree representing a partial solution which contains a non-maximal (complete) packing can be fathomed.

Dominance rule 2. A packing I dominates another packing I' if I contains at least one item i not being member of I' such that $w_i \geq \sum_{j \in I' - I} w_j$. This criterion, which is a special case of that in Section 2.2.1 with $q=1$, can be applied as follows. Whenever a maximal packing has been generated, it is checked if one or more items can be replaced by a single free item without reducing the weight of the packing. If such a replacement is possible, the examined packing or any partial solution containing this packing need not be considered further on. In the special case $I-I'=\{i\}$ and $I'-I=\{j\}$ with $w_i=w_j$, either packing dominates the other one. Hence, we define that I dominates I' only if $i<j$.

The following dominance rule 3 is based on the enumeration scheme of BISON (see Section 3.5) which forms a complete packing of a bin in each branching step. Therefore, all bins already contained in a partial solution of a node in the enumeration tree are filled maximally.

Dominance rule 3. A partial solution x dominates another partial solution x' which contains the same or a larger number of filled bins if all items already assigned to a bin within x' are also contained in some bin of x.

The applicability of dominance rule 3 is restricted by the necessity of storing partial solutions already considered during the enumeration. Therefore, this rule can only be applied in a very restricted manner.

3.3. *Combining FFD and B2F*

We propose an off-line approximation algorithm based on FFD and B2F. In contrast to the heuristics described in Section 2.1, it works bin-by-bin instead of considering the items in a prespecified order. It starts with the first empty bin and successively assigns the heaviest item which can be added feasibly. When the bin is filled maximally, the second one is opened. This process is repeated until all items are packed into some bin. Since items are considered in non-increasing order of their weights for each bin, we call this procedure a *bin-oriented FFD*. Note that the bin-oriented FFD constructs the same solutions as FFD, but may consider the items in a different order.

The procedure is extended as follows. After filling a bin maximally, the current partial solution is completed by applying B2F to the residual problem defined through the free items. In the first solution, the first bin is filled by the bin-oriented FFD while the others are packed by B2F. In the second solution, the first two bins are filled by the bin-oriented FFD and the remaining ones by B2F, and so on. The process stops whenever the remaining items fit into a single bin. The complete procedure (FFD-B2F for short) takes time $O(n^2 \cdot m)$, where m is the smallest number of bins found, because B2F is applied for $m-2$ times.

3.4. *Dual strategy with tabu search*

In order to obtain a strong upper bound before the branch and bound procedure is started, an improvement heuristic based on tabu search is developed. *Tabu search* is a meta-strategy which guides a local search procedure to overcome the barrier of local optimality [20–22]. A basic version of tabu search may be outlined as follows: it attempts to improve on a given (feasible) solution by iteratively transforming it into other (feasible) solutions while allowing for intermediate deteriorations of the objective function value. Transformations are referred to as *moves* and may be described by a set of one or more *attributes*. In order to prevent the procedure cyclically visiting the same set of solutions, some moves must temporarily be forbidden by setting attributes *tabu*. Hence, the most important component of tabu search is the *tabu list management*, i.e. deciding on how many and which attributes have to be set tabu within any iteration of the search. In the basic approach a static tabu list management is used where attributes are set tabu for a fixed number of future iterations.

Extended versions of tabu search contain more sophisticated components such as dynamic tabu list management and intensification or diversification based on long-term memory functions which frequently improve the basic approach. However, there is a trade-off between the computational effort of those extensions and the gain in solution quality. This trade-off has to be taken into account when designing a tabu search procedure for some problem, in particular, when it is to be applied as one of several components in an overall solution concept.

Trying to construct an effective tabu search procedure for BPP-1, one has to face with the problem that the capacity constraints are very restrictive and the usual definition of moves (shifting and swapping of items) merely has any effect on the objective function value [23]. Note that it is necessary to remove all items from a bin to obtain an improved solution. As a consequence, all moves possible in a certain iteration are often equivalent with respect to the objective function value, such that it is not clear which move has to be chosen. In order to overcome these difficulties, our approach utilizes the (dual) relationship between BPP-1 and BPP-2. This is achieved by the so-called *dual strategy*:

Starting with the best-known lower bound *LB*, it tries to find a feasible solution for the BPP-2 instance with $m=LB$ bins such that the weight of no bin exceeds the value of c. If such a solution (which is not necessarily optimal for BPP-2) is found, this solution is optimal for the original BPP-1 instance to be solved. Otherwise, the trial number m of bins is increased by 1 and the process is repeated until a feasible solution for BPP-1 is found or m reaches the value of the best-known upper bound UB. In each iteration of the dual strategy, any procedure for BPP-2 may be applied. Within BISON, we compute start solutions to the BPP-2 instance by a variant of WFD and improve on this solution by a tabu search procedure.

To be more specific, our application of the dual strategy (hereafter called *DualTabu*) works as follows. Given a *trial* number m of bins, the following variant of WFD is applied to this restricted BPP-2 instance. The items are considered in the order $j=1,..., n$ and are successively assigned to a bin which currently has the smallest weight. If the maximal weight of a bin does not exceed the capacity c, the procedure stops. Otherwise, the tabu search procedure is applied to the BPP-2 instance with fixed m.

In the following, we give a description of the components of our tabu search procedure for BPP-2. These components are chosen according to the following reflections. The procedure should be very fast, because it has to be applied for several trial numbers of bins within the dual strategy. In this context it is not necessary to determine optimal BPP-2 solutions, but to find feasible ones for a given combination of m and c. Furthermore, it is not applied as a stand-alone procedure, but within a hybrid approach to determine strong initial upper bounds. Bearing these arguments in mind it seems appropriate to restrict the tabu search procedure to a simple static approach. This decision is verified by results of some preliminary computational tests, which include more involved concepts such as long-term memory and dynamic tabu list management. These findings are supported by a comprehensive study on applying such concepts to the related simple assembly line balancing problem [23].

- *Definition of the neighbourhood.* From a solution x, neighbouring solutions x' are obtained by two simple moves [24]. Either one item j is shifted from its current bin h to a bin i (*shift move*) or items j and k are exchanged between their current bins h and i (*swap move*). Shift moves are characterized by the *attributes* $(j, \bar{h})$ (item j leaves bin h) and (j, i) (item j enters bin i). Since swap moves are composed of two shifts, they are characterized by four attributes $(j, \bar{h})$, $(k, \bar{i})$, (j, i) and (k, h). In order to obtain a really differing solution, w_j and w_k must not be equal. In each iteration of the procedure, one move is chosen and performed.
- *Basic tabu strategy.* In order to prevent the procedure cyclically revisiting solutions which have been generated in earlier iterations of the procedure, certain moves must be forbidden in each iteration. Obviously, it is necessary to forbid undoing the latest move, because the search could immediately return to the previous solution. However, further moves may close a cycle. Therefore, a *tabu list* is used to temporarily store attributes which must not be contained in a current move. We say that those attributes are set tabu and call them *tabu-attributes*. All moves which contain a tabu-attribute are forbidden in a current iteration of the search so that no cycling can occur at least as long as the attributes remain tabu. Whenever a shift move is performed, the attribute (j, h) is set tabu in order to prevent item j returning to bin h. After applying a swap move, the attributes (j, h) and (k, i) are set tabu. Note that this simple mechanism of defining tabu-attributes usually unnecessarily forbids a large number of moves which would not close a cycle.
- *Tabu list management.* We use a *static* tabu list with a fixed *tabu duration TD*, i.e. attributes which have been set tabu are released from the tabu list after a fixed number of iterations. The parameter *TD* is very important for the performance of tabu search. If *TD* is small, the risk of cycling is high. If *TD* is large, many moves may unnecessarily be forbidden (additionally to those unnecessarily forbidden by the definition of tabu-attributes) such that good solutions near to a current one could be missed. Therefore, *TD* is chosen depending on the size of the problem (in our tests we set $TD=\lfloor n/20 \rfloor$, see Section 4.3).
- *Move selection strategy.* In each iteration of the procedure, a non-forbidden (shift or swap) move is performed. Among several possible moves, one which results in a solution with the smallest weight of the heaviest bin (*best fit*) is preferred. That is, the most improving or least deteriorating move (*steepest descent/mildest ascent*) which does not contain a tabu-attribute is chosen. In order to restrict the number of moves to be examined, only moves which remove an item of the heaviest bin (say bin h) are considered. This is no restriction in case that an improving move exists, because then the weight of h must be reduced anyway. So computation time is saved only at expense of possibly missing the best non-improving move.

- *Evaluation of the neighbourhood.* In order to find a move to be performed in a current iteration, the objective function value of the respective neighbouring solutions must be computed. This value is obtained by computing the maximum of the new weights of the affected bins h and i and the weight of the heaviest bin p among the remaining ones. However, if the new objective function value is equal to the weight of bin p (in case of an improvement), it makes sense to consider only the weights of the bins h and i to distinguish between several moves giving the same objective function value (weight of p). Therefore and for speeding up the procedure, the maximum of these weights is used for evaluating the moves. The correct objective function value is computed only for the chosen move.
- *Stopping criteria.* The search is stopped when a solution feasible for BPP-1 is found or when a prespecified number of iterations has been performed. In the first case, the solution is stored as incumbent solution and the value of UB is updated.

Alternatively, we may use a simplified tabu strategy. It serves as a very simple diversification mechanism and simultaneously saves computation time. Instead of setting tabu an attribute (j, h) when an item j has been removed from a bin h, the attribute $(j, \bar{\imath})$ is set tabu so that it is forbidden to remove item j from its new bin i during the next TD iterations. That is, all moves concerning item j need not be considered during these iterations, such that the procedure is accelerated and the search is directed into different regions of the solution space within a short time.

3.5. *The local lower bound method*

We present a depth-first search (DFS) branch and bound procedure which includes a new enumeration strategy called the *local lower bound method.* This method attempts to overcome a typical drawback of DFS algorithms [25, 26]: whenever the first branches at the early levels of the enumeration tree are inappropriate with respect to finding good solutions, these procedures usually waste significant amounts of computation time for examining inferior parts of the tree. Hence, the local lower bound method tries to follow 'good' branches in every node of the tree at first. This is achieved by considering local lower bounds of nodes.

The procedure starts with a global lower bound LB in the root node (level 0 of the tree). Alternative subproblems are built by generating all *maximal* packings for the first bin. In order to prevent identical solutions with different orders of their packings being computed, only such packings containing the heaviest (free) item are considered in each node. These packings are found by a systematic enumeration based on the given index order of the items such that only really differing packings are built. Therefore, in case of items $j, j+1, ..., j+q$ having identical weights, an item $j+i$ $(1 \leq i \leq q)$ must not be added to a packing as long as one of the items $j, ..., j+i-1$ is free.

That is, the local lower bound method performs a *bin-oriented* branching, whereas MTP branches in an *item-oriented* manner by assigning one item to alternative bins in each step. The packings considered for the first bin can be subdivided into two classes according to the idle (unused) capacity (c – the weight of the packing). The *first class* contains those packings whose idle capacity does not exceed the total idle capacity $IC(LB) = LB \cdot c - \Sigma_{j \in J} w_j$ which is available provided that LB bins are sufficient. All packings with larger idle capacity constitute the *second class*. In order to find a solution with LB bins (if one exists) as early as possible, the packings of the first class are chosen for branching prior to those of the second class. After considering all branches (and the corresponding subtrees) due to the first class without success (no solution with LB bins exists for this subproblem), the lower bound LB is increased by 1 and the second class branches are examined. Note that $IC(LB+1) = IC(LB) + c$.

The same method is applied *locally* in each node at level k of the enumeration tree starting with the local lower bound LLB_k whose value is initially inherited from the local lower bound of the father node. This node represents a partial solution which contains k bins built by former branching steps and a residual problem which consists of the free items. The two classes of packings are defined according to the *remaining* total idle capacity $RIC(LLB_k) = IC(LLB_k)$ – the sum of idle capacities caused by the k bins already filled maximally. At first the different first-class branches are followed at each revisit of the node. If in the corresponding subtrees a solution with LLB_k bins is found, the node is fathomed and a new upper bound $UB = LLB_k$ is obtained. Otherwise, LLB_k is increased by 1 and $RIC(LLB_k)$ is increased by c. In the case of $LLB_k = UB$, the node is fathomed. Otherwise, the second-class branches and the corresponding subtrees are examined until a solution with LLB_k bins is found or all nodes in the subtrees are fathomed.

In each node of the tree, the reduction procedure MTRP (see Section 2.2.1) is applied in order to reduce the data of the (sub)problem. Furthermore, dominance rules 2 and 3 are utilized in order to exclude dominated packings from being used for branching.

Example. We consider a problem instance with $n=10$ items having the weights 50, 40, 35, 26, 20, 17, 17, 15, 14 and 5. The bin capacity is $c=80$ and the procedure starts with the (global) lower bound $LB=LB_1=\lceil 239/80 \rceil=3$ and the total idle capacity $IC(3)=3\cdot 80-239=1$. In the root node, the following maximal packings are possible:

$$I_1=\{1,4\},\ I_2=\{1,5,10\},\ I_3=\{1,6,10\},\ I_4=\{1,7,10\},\ I_5=\{1,8,9\},\ I_6=\{1,8,10\},\ I_7=\{1,9,10\}$$

The packings I_2–I_4 (I_6 and I_7) are dominated by I_1 (I_5) and eliminated by dominance rule 2. Note that I_4 would not have been built, because it contains identical weights as I_3. The complete enumeration tree with all non-dominated packings is given in Fig. 1. The nodes are numbered consecutively and the packings used for branching are denoted on the edges and arranged in lexicographic order. The hatched area shows the subtree where the local lower bounds of the nodes are equal to 3, whereas all other nodes show a value of 4.

In the root node, the packings I_1 and I_5 have to be considered. Since their idle capacities are 4 and 1, respectively, I_5 belongs to the first class and I_1 to the second class of packings. Hence, the packing I_5 is assigned to the first bin and node 11 is obtained. There, only one maximal non-dominated packing exists and is assigned to bin 2, leading to node 12. Since the remaining items build a packing for bin 3, a feasible (=optimal) solution with three bins is obtained. The procedure stops without checking the rest of the tree due to $LB=3$.

The example indicates that a rather small tree may be constructed by the local lower bound method whenever a solution with LB bins exists. In contrast to other procedures (including MTP) which may examine a large subtree following an inferior first assignment, the local lower bound method considers partial solutions with small idle capacities first. Even if no solution with LB bins exists, a good feasible solution is found in the first complete branch of the tree, because it attempts to find packings with or without small idle capacities. However, the enumeration of packings must be done twice in order to avoid storing the packings of the second class. This results in a small additional effort which is definitely justified whenever the enumeration of large subtrees can be avoided.

3.6. The complete procedure BISON

The hybrid procedure BISON contains the following six steps:

(1) Apply the reduction procedure MTRP.
(2) Compute the lower bounds LB_2, LB_4 and LB_5. Set $LB:=\max\{LB_2, LB_4, LB_5\}$.
(3) Apply FFD, B2F, BFD and WFD as well as FFD-B2F (Section 3.3), resulting in an incumbent solution with upper bound UB. If $UB=LB$ then stop.

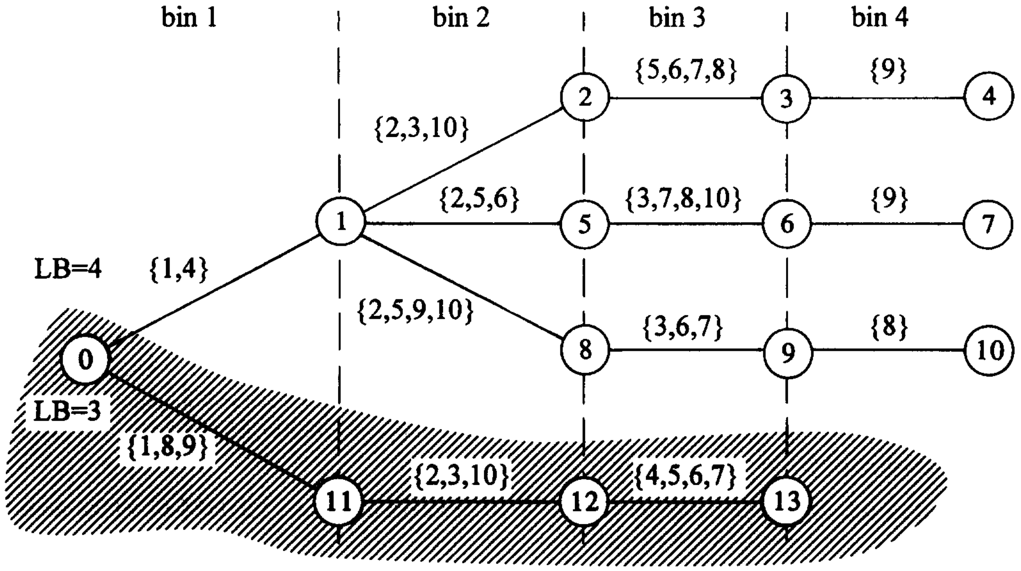

Fig. 1. Enumeration tree.

(4) Compute the lower bounds LB_3 and LB_6. Set $LB := \max\{LB, LB_3, LB_6\}$. Stop if $UB = LB$.
(5) Apply the dual strategy with tabu search (DualTabu, Section 3.4) in order to improve UB. If $UB = LB$ then stop.
(6) Apply the local lower bound method (Section 3.5).

The given order of components is chosen for the following reasons. At first, only simple bound arguments and simple heuristics are applied. This enables the procedure to find optimal solutions to easy instances within a very short computation time. Harder instances are tackled by the more expensive bound arguments, LB_3 and LB_6. When these bound arguments succeed in improving the bound values, a heuristic solution of step (2) may be recognized as being optimal. Otherwise, it attempts to find an improved solution with DualTabu which is more expensive than the simple heuristics. DualTabu is applied after all bound arguments, because its effort depends on the value of LB. If it is not possible to find a solution with LB bins, the local lower bound method has to be applied.

4. COMPUTATIONAL EXPERIMENTS

We report on results of computational experiments, which have been performed in order to examine and evaluate the performance of our new procedure BISON. As a reference procedure, MTP of Martello and Toth [2] is considered.

4.1. Data sets and computational environment

Three data sets have been generated for testing the procedures. These sets are available via the internet (URL: http://www.bwl.th-darmstadt.de/bwl3/forsch/projekte/binpp/index.htm).

Data set 1. It is constructed similar to the data set proposed by Martello and Toth [2]. The following parameter settings are considered:

- $n = 50, 100, 200, 500$;
- $c = 100, 120, 150$;
- $w_j \in [1, 100], [20, 100], [30, 100]$ for $j = 1, \dots, n$.

The weights are chosen as integer values from the given intervals using uniformly distributed random numbers. For each of the 36 instance classes defined by the settings above, 20 instances are generated resulting in a total of 720 instances. Currently, 704 instances are solved to optimality. In contrast to Martello and Toth, we omit instances with $w_j \in [50, 100]$, because these instances are very simple and are solved by applying MTRP (see Remark 1). In order to avoid such simple cases for each value of the bin capacity c, we instead use the third interval given above.

Data set 2. In data set 1, all parameters are chosen such that the expected average number of items per bin is not larger than three. In order to generate instances with an average of three, five, seven or nine items per bin, the following settings are considered:

- $n = 50, 100, 200, 500$;
- $c = 1000$;
- $\bar{w}_j = c/3, c/5, c/7, c/9$;
- $\delta = 20\%, 50\%, 90\%$.

The parameter $\bar{w}_j$ represents the desired average weight of the items, while δ specifies the maximal deviation of the single values from $\bar{w}_j$. For example, the weights are randomly chosen from the interval [160, 240] in case of $\bar{w}_j = c/5$ and $\delta = 20\%$. For each of the 48 classes, 10 instances are generated resulting in a total of 480 instances. In 477 cases, an optimal solution is known.

Data set 3. With instances of the first two data sets, the number of possible different values for the weights is rather small. In such cases, it is likely that several items have the same weight (reducing the number of really differing packings) and that many packings which do not cause idle capacity are present. Therefore, 10 (difficult) instances with $n = 200$, $c = 100\,000$ and $w_j \in [20\,000, 35\,000]$ are generated. Only three instances are solved to optimality up to now.

Combined data set. The three data sets contain 1210 problem instances, for 1184 of which an optimal solution is known.

Computational environment. All tests have been performed on an IBM-compatible personal computer

Table 1. Results of MTRP

Data set	Data set 1	Data set 2	Data set 3	Combined
Average percentage of reduction	74%	1%	0%	45%
Number of optimal solutions	280	0	0	280

with 80486 DX2-66 central processing unit. The algorithms MTP and BISON have been coded by the authors using Borland's Pascal 7.0. In order to utilize the efficient way of implementing MTP used in the original Fortran code [2], the Pascal code has been programmed as similar as possible. The recoding of MTP seemed to be necessary for a fair comparison. Note that the original Fortran code gives the same results, but requires longer computation times.

For evaluating and comparing procedures we use the following definitions of the *relative deviation* and the *absolute deviation* of a value x from a value y, where x and y are values of lower or upper bounds:

$$\Delta_{\mathrm{rel}}(x,y)=\frac{|x-y|}{y}\cdot 100\%,\ \Delta_{\mathrm{abs}}(x,y)=|x-y|$$

In particular, we investigate the deviations from the optimal number z^* of bins. Whenever z^* is unknown, the best-known lower bound value is used for calculating the deviations.

4.2. *Analysis of MTRP and the lower bounds*

We examine the efficacy of the reduction procedure MTRP and the different bound arguments LB_2–LB_6 contained in BISON.

As shown in Table 1, MTRP is very successful in reducing the instances of data set 1, because 74% of the bins can be fixed by MTRP on average. For 280 of the 720 instances contained in data set 1, all items are packed into some bin, thereby finding an optimal solution to the problem. Since the instances of data sets 2 and 3, respectively, do not contain large items and in most cases require that more than three items are packed into some bins, MTRP is not useful for these data sets. These results show that many instances contained in data set 1 are rather simple while the other data sets seem to be more challenging.

The different bound arguments are compared to the simple bound LB_1. Since it is possible to obtain improved bound values only for problem instances where it is known that the optimal number of bins exceeds the value of LB_1, we restrict our comparison to the corresponding subset of the combined data set which contains 133 instances. Before computing the values of the lower bounds, the reduction procedure MTRP is applied to every instance. The destructive bound arguments LB_5 and LB_6 start their search processes with the value of LB_1, even if sharper values are available. When computing LB_6, the local lower bound method is applied for at most 10 s whenever step (6) is reached (cf. Section 3.1).

The bounds LB_i ($i=2,\ldots,6$) are evaluated with respect to the following measures:

- # inc. LB_1—number of instances for which LB_i is larger than LB_1;
- av. rel. LB_1—average relative deviation of LB_i from LB_1 (%);
- # opt. LB—number of instances for which LB_i is equal to the optimal number of bins;
- av. rel.—average relative deviation of LB_i from the optimal number of bins (%);
- av. cpu—average computation time in seconds (for all instances of the data sets).

Table 2 gives a summary of the results obtained for those 133 instances selected from the combined data set. This subset contains 113 instances out of data set 1 and 20 instances out of data set 2. In the latter case, all respective instances stem from the class characterized by $\bar{w}_j=c/3$ and $\delta=20\%$. In order to find differences between the data sets used, the results are given for the instances of the combined data set and those of the data sets 1 and 2 separately. The last column specifies results for the best bound value $LB=\max\{LB_i \mid i=2,\ldots,6\}$.

The results show that LB_3 and LB_6 are most effective. They are both able to exceed the value of LB_1 for about 70% of the 133 instances considered. However, they differ depending on the two single data sets. While LB_3 is more successful for data set 1 due to the effectiveness of MTRP (cf. Table 1), LB_6 always finds the optimal value for data set 2 due to the special structure of instances contained. LB_3 gives better values for nine instances, whereas LB_6 is superior for 11 instances. That is, both bound arguments complement one another in a good manner. Since the instances of data set 2 can be reduced by MTRP only in a few cases, the computation times of both bound arguments are much longer than for data set

Table 2. Comparison of bounds

Measure	Data set	LB_2	LB_3	LB_4	LB_5	LB_6	LB
# inc. LB_1	Combined	72	94	64	41	92	103
	Data set 1	59	81	64	28	72	83
	Data set 2	13	13	0	13	20	20
av.rel. LB_1	Combined	1.78	2.24	1.77	1.01	2.31	2.50
	Data set 1	1.88	2.42	2.08	0.98	2.23	2.45
	Data set 2	1.21	1.21	0.00	1.21	2.77	2.77
# opt. LB	Combined	60	91	55	25	90	102
	Data set 1	47	78	55	12	70	82
	Data set 2	13	13	0	13	20	20
av. rel.	Combined	1.26	0.83	1.28	1.99	0.76	0.58
	Data set 1	1.23	0.71	1.04	2.09	0.90	0.68
	Data set 2	1.48	1.48	2.66	1.48	0.00	0.00
av. cpu	Combined	0.00	0.22	0.00	0.00	0.26	0.48
	Data set 1	0.00	0.04	0.00	0.00	0.01	0.05
	Data set 2	0.00	0.49	0.00	0.00	0.65	1.14

1. However, it is always worthwhile to invest some seconds of computation time whenever a bound improvement is possible, because examining large parts of an enumeration tree may be avoided.

The other bound arguments, taking negligible computation times, give worse results. Even though LB_2 is contained in the computation of LB_3, it may be advantageous to compute its value. Whenever a heuristic solution with LB_2 bins is identified in step (3) of BISON (cf. Section 3.6), the expensive computation of LB_3 and LB_6 (step 4) is avoided. LB_4 is not useful for the 20 instances considered within data set 2, because it can find better values than LB_1 only if at least some tasks require more than half a bin each. LB_5 gives the worst results, because the bound for BPP-2 applied within this destructive bound argument for BPP-1 is too weak. Since it is nevertheless successful for almost one-third of all instances, it seems promising to search for better BPP-2 bounds.

4.3. Analysis of the heuristics

In the following, we analyse the performance of the different heuristics which are used as components of BISON. Besides comparing the solution quality of the different off-line algorithms and B2F, it seems to be worth examining whether the newly introduced algorithms FFD-B2F and DualTabu justify their additional computational effort.

In order to reduce the computational complexity of FFD-B2F to $O(n^2)$, B2F is only applied 10 times irrespective of the value of LB. That is, B2F is always applied after filling $\lfloor m/10 \rfloor$ (further) bins by FFD.

In case of DualTabu, the following settings found by parameter adjusting tests are used. For each trial number of bins examined by DualTabu, a maximal number of $5 \cdot n + 1000$ iterations are performed and the tabu duration TD is set to $\lfloor n/20 \rfloor$. Furthermore, the simplified tabu strategy is used, because further tests have shown that the results are better than in case of using the basic strategy when only a small amount of computation time is available (cf. Section 3.4 for descriptions of these strategies).

For the comparison, each heuristic has been applied to all instances of the combined data set after reducing them by MTRP. The average computation time per instance required by the constructive algorithms FFD, BFD, WFD and B2F is about 0.01 s, while FFD-B2F and DualTabu take about 0.1 and 0.4 s of computation time on average. Of course, it is problematic to compare constructive procedures which generate one or a few (FFD-B2F) solutions to an improvement procedure which has to evaluate a considerable number of solutions. However, DualTabu is allowed to perform only a relatively small number of iterations per trial number of bins.

The results have been evaluated with help of the following measures (based on all instances of a data set):

- # found—number of instances for which an optimal solution is found (but not necessarily proven);
- av. rel.—average relative deviation from optimality (%);
- max. rel.—maximal relative deviation from optimality (%);
- av. abs.—average absolute deviation from optimality;
- max. abs.—maximal absolute deviation from optimality.

In Table 3, we give a summary of the results for the combined data set and the single sets. In the last column, the results of a compound procedure which applies all heuristics are given. That is, the last column refers to the best solutions provided by any heuristic.

Table 3. Results of heuristics

Measure	Data set	FFD	BFD	WFD	B2F	FFD-B2F	DualTabu	Compound
# found	Combined	783	784	601	837	936	1135	1155
	Data set 1	547	548	409	545	617	666	685
	Data set 2	236	236	192	292	319	466	467
	Data set 3	0	0	0	0	0	3	3
av. rel.	Combined	1.55	1.55	2.43	0.89	0.63	0.12	0.08
	Data set 1	0.49	0.48	1.30	0.50	0.20	0.13	0.07
	Data set 2	3.02	3.02	3.86	1.43	1.23	0.07	0.07
	Data set 3	7.39	7.39	15.33	3.07	2.70	1.26	1.26
max. rel.	Combined	16.67	16.67	20.00	15.00	14.29	6.25	6.25
	Data set 1	6.25	6.25	10.53	15.00	6.25	5.88	5.88
	Data set 2	16.67	16.67	20.00	14.29	14.29	6.25	6.25
	Data set 3	9.26	9.26	18.52	3.70	3.70	1.85	1.85
av. abs.	Combined	0.90	0.90	1.44	0.52	0.38	0.08	0.05
	Data set 1	0.41	0.41	0.93	0.35	0.18	0.11	0.06
	Data set 2	1.56	1.56	2.06	0.76	0.67	0.03	0.03
	Data set 3	4.10	4.10	8.50	1.70	1.50	0.70	0.70
max. abs.	Combined	21	21	25	7	7	6	2
	Data set 1	7	7	9	5	3	6	2
	Data set 2	21	21	25	7	7	1	1
	Data set 3	5	5	10	2	2	1	1

The results show that DualTabu clearly performs best. In particular, it is able to obtain much better solutions than all constructive algorithms for data sets 2 and 3 with respect to all measures of solution quality. For 218 instances (for 75, 135 and 8 in data set 1, 2, and 3, respectively) it finds a better solution than all other procedures. Therefore, in any case it seems worthwhile to invest the small amount of computation time necessary for DualTabu in order to speed up the overall algorithm BISON.

The other procedures, especially FFD-B2F, obtain competitive results only for data set 1, where large portions of the instances are already reduced by MTRP. The off-line algorithms FFD, BFD and WFD perform clearly worse for the other data sets, because respective instances do not have very small items which can be used to fill small idle capacities which have been caused by combining larger items in a greedy manner. This behaviour results in large absolute deviation values with up to 25 additional bins. In particular, these strategies are poor whenever the maximal number of items per bin is small. B2F and FFD-B2F give better average- and worst-case results, because they try to reoptimize a packing by exchanging one item for two smaller ones. While WFD performs much worse than FFD and BFD for data sets 1 and 3, it gives results only slightly worse for data set 2, where it sometimes may be advantageous to leave large idle capacities for items considered later so that two items may be added to a current packing instead of one.

Furthermore, Table 3 shows that FFD and BFD behave almost identically. At first glance this is very surprising. However, when analysing their proceeding, one recognizes that differences are really rather seldom (in our test BFD finds a better solution only in one instance). Both procedures consider items in non-increasing order of weights. As long as FFD is able to fill the bins according to the order of their initialization, i.e. no currently used bin contains more items and has a smaller idle capacity than the preceding bin, respectively, FFD and BFD perform the same assignments. The following example with eight items shows a case where both procedures may lead to different solutions. With item weights 45, 42, 40, 30, 18, 11, 8, 5 and $c=100$, FFD and BFD determine solutions with three and two bins, respectively.

The compound procedure with an average computation time of about 0.4 s per instance provides very nice results. In particular, it is remarkable that the maximal relative and absolute deviations from optimality are very small. In the worst case, only two additional bins are required whereas all single procedures have larger deviations. This small empirical worst-case error shows that the different heuristics complement one another in a fine manner.

4.4. Comparison of BISON and MTP

The complete procedures BISON *and* MTP have been applied to every instance of the combined data set with a particular time limit. Whenever the time limit is reached ('time out' for short) without proving the current incumbent solution to be optimal, only a heuristic solution with an upper bound UB on the number of bins is available.

The procedures are compared with respect to the measures used above. Instead of the number of optimal solutions found we specify the following:

Table 4. Results for data set 1

Measure	*TL*=50		*TL*=100		*TL*=250		*TL*=500		*TL*=1000	
	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON
#opt.	632	695	632	697	634	697	635	697	637	697
av. rel.	0.19	0.04	0.19	0.04	0.19	0.04	0.19	0.04	0.18	0.04
max. rel.	4.55	2.38	4.55	2.38	4.55	2.38	4.55	2.38	4.55	2.38
av. abs.	0.26	0.04	0.26	0.04	0.26	0.04	0.26	0.04	0.26	0.04
max. abs.	7	2	7	2	7	2	7	2	7	2
av. cpu	6.5	2.0	12.6	3.6	30.5	8.4	60.2	16.5	118.3	32.4

- # opt.—number of instances for which an optimal solution is found *and* proven.

The procedure BISON is applied as described in Section 3.6 with following specifications. The reduction procedure MTRP and the dominance rules are applied in every node, whereas the bound arguments LB_2 to LB_6 are only computed in the root node. In step (2), the heuristics are applied in the order FFD, B2F, BFD, WFD, FFD-B2F due to the results of Section 4.3. Within DualTabu the simplified tabu strategy is used, because it requires much less computation time than the basic strategy or more involved concepts for equivalent solution qualities.

Due to their computational complexity, the dominance rules 2 and 3 are applied in restricted manners. Concerning rule 2, only dominances with $|I' - I| \leq 2$ are identified (cf. Section 3.3). In case of rule 3, only a restricted number of partial solutions is stored using a binary representation of sets. In each level k of the tree (k bins are already built), the first LB-k partial solutions which have been enumerated are stored and compared to current nodes. This simple strategy is based on the assumption that it is most effective to fathom nodes at the first levels of the tree, because large subtrees may be avoided.

Both MTP and BISON are applied with time limits of TL=50, 100, 250, 500 and 1000 s. Tables 4–6 show summaries of the results for the data sets 1, 2 and 3, respectively. While MTP obtains good results for data set 1, it performs clearly worse for data set 2 and very poor for data set 3. BISON produces clearly better results in all cases and outperforms MTP even if it is applied only for 50 s and MTP runs for 1000 s. BISON takes shorter computation times with 973 instances of the combined data set, while MTP is faster with 49 instances (36 instances are solved by neither procedure within 1000 s, 152 instances take almost the same time). Applying a simple non-parametric statistical sign test gives a confidence limit very close to 1 for the hypothesis that BISON is superior concerning computational speed.

In particular, BISON is clearly shown to be dominating in data set 2 (cf. Table 5). It solves about 30% more instances to optimality and reduces the average relative deviation by an order of magnitude. The poor performance of MTP for data set 2 is probably due to the structure of instances contained in this set. In particular, these instances neither contain very heavy items which can favourably be utilized by MTRP and the bound LB_3, nor very light items which can easily be combined with other items. Furthermore, more than three items are to be packed into the bins on average in all cases but $\bar{w}_j = c/3$. Though a larger

Table 5. Results for data set 2

Measure	*TL*=50		*TL*=100		*TL*=250		*TL*=500		*TL*=1000	
	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON
# opt.	352	472	357	472	359	472	363	473	364	473
av. rel.	1.02	0.03	1.00	0.03	0.99	0.03	0.95	0.02	0.94	0.02
max. rel.	14.29	5.56	14.29	5.56	14.29	5.56	14.29	2.94	14.29	2.94
av. abs.	0.66	0.02	0.64	0.02	0.62	0.02	0.60	0.01	0.60	0.01
max. abs.	9	1	9	1	9	1	9	1	9	1
av. cpu	14.0	1.5	27.0	2.4	65.2	4.9	126.8	9.0	247.7	16.3

Table 6. Results for data set 3

Measure	*TL*=50		*TL*=100		*TL*=250		*TL*=500		*TL*=1000	
	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON	MTP	BISON
#opt.	0	3	0	3	0	3	0	3	0	3
av. rel.	3.07	1.26	3.07	1.26	3.07	1.26	2.71	1.26	2.71	1.26
max. rel.	7.27	1.85	7.27	1.85	7.27	1.85	7.27	1.85	7.27	1.85
av. abs.	1.70	0.70	1.70	0.70	1.70	0.70	1.50	0.70	1.50	0.70
max. abs.	4	1	4	1	4	1	4	1	4	1
av. cpu	50.0	35.2	100.0	70.2	250.0	175.2	500.0	350.2	1000.0	700.2

number of items per bin produces low idle capacities of packings and therefore is expected to simplify the problem, the relatively small intervals of weights complicate finding good item combinations.

In case of data set 3, MTP does not solve any of the 10 instances, whereas BISON finds optimal solutions for three instances by DualTabu. The results indicate that these instances are really very challenging due to the large interval of weight values. Note that neither procedure is able to solve further instances even if they are applied for several hours of computation time. Also, more sophisticated tabu search procedures fail to solve those instances.

The maximal absolute deviation of BISON never exceeds two bins, whereas MTP requires up to nine unnecessary additional bins. This very good empirical worst-case error of BISON is already obtained by the heuristics. Furthermore, the local lower bound method is very successful in finding very good solutions within short computation times. Therefore, BISON requires significantly shorter computation times on average.

It is worth noting that neither procedure is able to significantly improve its results when longer computation times are available. That is, if it is not possible to direct a DFS branch and bound procedure to a 'promising' part of the enumeration tree at the early levels, the procedure has to spent a lot of time with examining inferior subtrees without success. Therefore, it is very important to have available very good heuristics and intelligent search mechanisms such as the local lower bound method.

Detailed results of MTP and BISON with respect to different problem classes are given in the Appendix.

5. SUMMARY AND CONCLUSIONS

In this paper, we present BISON, a new hybrid exact procedure for the one-dimensional bin packing problem (BPP-1). This NP-hard problem is to pack a given set of items having individual weights into a minimum number of bins which have equal capacity.

BISON is composed of different known and new bound arguments and reduction procedures, several heuristics, and a branch and bound procedure. Among the heuristics, an application of a tabu search procedure for the related problem BPP-2 (given the number of bins the capacity is to be minimized) within a search procedure, called dual strategy, is most effective. The branch and bound procedure contains a new branching scheme, which is called local lower bound method, and some dominance rules. The local lower bound method differs from traditional depth-first search approaches in that among several subproblems descending from a current node, it prefers those which do not require to increase the current local lower bound of the node. By this partial ordering of descending nodes, it is possible for the procedure to avoid examining large subtrees which cannot contain good solutions.

Computational experiments on randomly generated data sets show that BISON clearly outperforms the well-known branch and bound procedure MTP of ref. [2]. In particular for hard problem instances, BISON is much more effective. Even if MTP runs for a multiple of the computation time given to BISON, it cannot achieve competitive results. The effectiveness of BISON is mainly due to the combination of different solution concepts: several bounds are utilized to compute sharp initial lower bounds and heuristics are applied to find good initial upper bounds which are often equal to the optimal value. In both cases it is accepted that for some hard instances a remarkable amount of computation time is consumed. Whenever a gap between the lower and the upper bound remains, the local lower bound method simultaneously tries to find an improved upper bound and systematically increases the lower bounds (of the root and all other nodes).

Encouraged by our research, we believe that further developments for bin packing and other combinatorial optimization problems should concern improved concepts of integrating heuristic strategies such as tabu search within exact solution procedures. Both approaches have different strengths which may favourably be combined while shortcomings may be avoided. Exact procedures are enforced to examine a solution space in a systematic manner, thereby considering many inferior solutions and consuming a lot of computation time, but are able to prove the optimality of a solution found. By way of contrast, heuristic search procedures focus on examining promising parts of the solution space intensively, but are unable to prove solutions to be optimal if lower bounds are too weak.

REFERENCES

1. Dyckhoff, H. and Finke, U., *Cutting and Packing in Production and Distribution.* Physica, Heidelberg, 1992.
2. Martello, S. and Toth, P., *Knapsack Problems.* Wiley, Chichester, 1990.

3. Coffman, E. G., Garey, M. R. and Johnson, D. S., Approximation algorithms for bin-packing—an updated survey. In *Algorithm Design for Computer System Design*, eds G. Ausiello, M. Lucertini and P. Serafini, pp. 49–106. Springer, Berlin, 1984.
4. Cheng, T. C. E. and Sin, C. C. S., A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 1990, **47**, 271–292.
5. Domschke, W., Scholl, A. and Voß, S., *Produktionsplanung—Ablauforganisatorische Aspekte*. Springer, Berlin, 1993.
6. Hochbaum, D. S. and Shmoys, D. B., Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the Association for Computing Machinery*, 1987, **34**, 144–162.
7. Labbé, M., Laporte, G. and Martello, S., An exact algorithm for the dual bin packing problem. *Operations Research Letters*, 1995, **17**, 9–18.
8. Spieksma, F. C. R., A branch-and-bound algorithm for the two-dimensional vector packing problem. *Computers and Operations Research*, 1994, **21**, 19–25.
9. Johnson, D. S., Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 1974, **9**, 256–278.
10. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R. and Graham, R. L., Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 1974, **3**, 299–325.
11. Baker, B. S. and Coffman, E. G., A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM Journal on Algebraic and Discrete Methods*, 1981, **2**, 147–152.
12. Simchi-Levi, D., New worst-case results for the bin-packing problem. *Naval Research Logistics*, 1994, **41**, 579–585.
13. Eilon, S. and Christofides, N., The loading problem. *Management Science*, 1971, **17**, 259–267.
14. Hung, M. S. and Brown, J. R., An algorithm for a class of loading problems. *Naval Research Logistics Quarterly*, 1978, **25**, 289–297.
15. Scholl, A., *Balancing and Sequencing of Assembly Lines*. Physica, Heidelberg, 1995.
16. Labbé, M., Laporte, G. and Mercure, H., Capacitated vehicle routing on trees. *Operations Research*, 1991, **39**, 616–622.
17. Berger, I., Bourjolly, J.-M. and Laporte, G., Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European Journal of Operational Research*, 1992, **58**, 215–222.
18. Martello, S. and Toth, P., Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 1990, **28**, 59–70.
19. Jackson, J. R., A computing procedure for a line balancing problem. *Management Science*, 1956, **2**, 261–271.
20. Glover, F., Tabu search: part I. *ORSA Journal on Computing*, 1989, **1**, 190–206.
21. Glover, F., Tabu search: part II. *ORSA Journal on Computing*, 1990, **2**, 4–32.
22. Glover, F. and Laguna, M., Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*, ed. C. R. Reeves, pp. 70–150. Blackwell, Oxford, 1993.
23. Scholl, A. and Voß, S., Simple assembly line balancing—heuristic approaches. *Journal of Heuristics*, 1996, **2**, 217–244.
24. Hübscher, R. and Glover, F., Applying tabu search with influential diversification to multiprocessor scheduling. *Computers and Operations Research*, 1994, **21**, 877–884.
25. Scholl, A. and Klein, R., SALOME: a bidirectional branch and bound procedure for assembly line balancing. *INFORMS Journal on Computing* (to appear).
26. Klein, R. and Scholl, A., Maximizing the production rate in simple assembly line balancing—a branch and bound procedure. *European Journal of Operational Research*, 1996, **91**, 367–385.

APPENDIX

Tables A1 and A2 show detailed results of MTP and BISON, respectively, for data set 1 ($TL=50$) subdivided with respect to the different problem classes each of which consists of 20 instances.

For both procedures, the classes of instances with $c=150$ and $w_j \in [20, 100]$ or $w_j \in [30, 100]$ are most complex. However, BISON clearly outperforms MTP just for those hard instances. While MTP in some of these classes with $n=200$ or 500 cannot solve any instance, BISON is able to find most optimal solutions except in the case of $c=150$ and $w_j \in [30, 100]$, where it solves only eight instances. MTP gives a better solution only for two instances with $c=120$ and $w_j \in [1, 100]$.

Tables A3 and A4 show aggregate results ($TL=50$) for the different problem classes of data set 2, each of which contains 10 instances. It becomes obvious that MTP performs worst if $\delta=20\%$ or if $\delta \geq 50\%$ and packings contain three or five items ($\bar{w}_j \geq c/5$). In the first case, all weights have rather similar values and all bins are expected to have almost the same number of items in an optimal solution. For example, in case of $\delta=20\%$ and $\bar{w}_j=c/5$, at least four items and at most six items can share a bin. Since the branching scheme of MTP follows an FFD strategy, it packs as many of the heaviest items into the first bins as possible. This may result in wasted idle capacities being a little smaller than the weight of the lightest item. Due to the item-oriented branching this is not recognized before a feasible solution is found or the respective subproblem is fathomed by bounding. That is, an huge subtree

Table A1. Detailed results of MTP for data set 1

		MTP								
Data set 1			$w_j \in [1, 1000]$			$w_j \in [20, 100]$			$w_j \in [30, 100]$	
n	c	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu
50	100	20	0.00	0.03	20	0.00	0.03	20	0.00	0.03
	120	20	0.00	0.08	20	0.00	0.03	20	0.00	0.03
	150	20	0.00	0.27	20	0.00	1.86	20	0.00	0.30
100	100	20	0.00	0.07	20	0.00	0.06	20	0.00	0.05
	120	19	0.12	2.59	20	0.00	0.09	20	0.00	0.06
	150	20	0.00	0.06	15	0.48	12.60	16	0.35	13.25
200	100	19	0.00	2.66	20	0.00	0.27	20	0.00	0.11
	120	18	0.12	5.16	20	0.00	0.10	20	0.00	0.11
	150	19	0.07	2.66	12	0.56	21.28	0	2.18	50.10
500	100	16	0.08	10.26	19	0.00	2.87	20	0.00	0.25
	120	20	0.00	0.77	20	0.00	0.76	20	0.00	0.39
	150	19	0.03	3.18	0	0.92	50.00	0	2.02	50.00

Table A2. Detailed results of BISON for data set 1

						BISON				
Data set 1			$w_j \in [1, 100]$			$w_j \in [20, 100]$			$w_j \in [30, 100]$	
n	c	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu
50	100	20	0.00	0.05	20	0.00	0.04	20	0.00	0.01
	120	19	0.00	2.57	20	0.00	0.01	20	0.00	0.00
	150	20	0.00	0.35	20	0.00	0.37	20	0.00	0.11
100	100	20	0.00	0.12	20	0.00	0.00	20	0.00	0.01
	120	19	0.12	2.61	20	0.00	0.12	20	0.00	0.01
	150	20	0.00	0.01	17	0.24	9.28	18	0.24	5.89
200	100	20	0.00	0.69	20	0.00	0.01	20	0.00	0.01
	120	19	0.06	2.67	20	0.00	0.01	20	0.00	0.01
	150	20	0.00	0.06	19	0.06	2.77	17	0.23	8.31
500	100	19	0.02	3.01	20	0.00	0.17	20	0.00	0.06
	120	20	0.00	0.50	20	0.00	0.17	20	0.00	0.16
	150	20	0.00	0.15	20	0.00	0.99	8	0.42	30.40

may unnecessarily be examined by MTP, while the local lower bound method of BISON avoids constructing such inferior packings as long as possible.

BISON fails only for a few single instances. Some classes which are very complex for MTP are easily solved by BISON. In particular, DualTabu is very successful in finding solutions for such instances.

Table A3. Detailed results of MTP for data set 2

						MTP				
Data set 2			δ=20%			δ=50%			δ=90%	
n	$\bar{w}_j$	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu
50	$c/3$	8	0.00	15.05	9	0.63	5.97	10	0.00	0.33
	$c/5$	9	0.91	8.82	10	0.00	0.03	10	0.00	0.03
	$c/7$	7	4.29	15.03	10	0.00	0.05	10	0.00	0.04
	$c/9$	10	0.00	0.03	10	0.00	0.03	10	0.00	0.02
100	$c/3$	10	0.00	0.53	4	2.33	34.76	6	1.28	20.08
	$c/5$	4	2.86	30.08	10	0.00	0.10	9	0.48	5.07
	$c/7$	8	1.43	10.08	10	0.00	0.12	10	0.00	0.05
	$c/9$	8	1.82	10.07	10	0.00	0.07	10	0.00	0.06
200	$c/3$	9	0.00	13.40	0	4.39	50.02	6	0.60	20.12
	$c/5$	2	5.14	40.13	8	0.51	11.30	8	0.50	10.12
	$c/7$	8	0.71	10.15	10	0.00	0.24	10	0.00	0.42
	$c/9$	7	1.30	15.13	9	0.45	5.18	10	0.00	0.11
500	$c/3$	1	1.73	45.61	0	4.88	50.05	7	0.18	15.45
	$c/5$	0	7.23	50.02	0	1.19	50.03	10	0.00	1.62
	$c/7$	0	1.41	50.04	3	0.99	35.56	10	0.00	0.32
	$c/9$	5	1.07	25.37	7	0.55	15.27	10	0.00	0.26

Table A4. Detailed results of BISON for data set 2

						BISON				
Data set 2			δ=20%			δ=50%			δ=90%	
n	$\bar{w}_j$	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu	# opt.	av. rel.	av. cpu
50	$c/3$	10	0.00	0.10	10	0.00	0.02	9	0.56	5.15
	$c/5$	10	0.00	0.00	10	0.00	0.00	10	0.00	0.00
	$c/7$	10	0.00	0.02	10	0.00	0.00	10	0.00	0.00
	$c/9$	10	0.00	0.00	10	0.00	0.00	10	0.00	0.00
100	$c/3$	10	0.00	2.16	8	0.58	10.13	10	0.00	0.23
	$c/5$	10	0.00	0.06	10	0.00	0.02	10	0.00	0.04
	$c/7$	10	0.00	0.03	10	0.00	0.00	10	0.00	0.00
	$c/9$	10	0.00	0.03	10	0.00	0.00	10	0.00	0.00
200	$c/3$	10	0.00	9.39	10	0.00	0.41	9	0.15	5.93
	$c/5$	10	0.00	0.32	10	0.00	0.12	10	0.00	0.07
	$c/7$	10	0.00	0.09	10	0.00	0.03	10	0.00	0.03
	$c/9$	10	0.00	0.01	10	0.00	0.00	10	0.00	0.01
500	$c/3$	10	0.00	10.50	9	0.06	6.83	9	0.06	6.36
	$c/5$	8	0.20	11.11	10	0.00	1.12	10	0.00	0.39
	$c/7$	10	0.00	1.05	10	0.00	0.35	10	0.00	0.08
	$c/9$	10	0.00	0.59	10	0.00	0.15	10	0.00	0.05