Parallel greedy algorithms
for packing unequal circles
into a strip or a rectangle

Timo Kubach, Andreas Bortfeldt und Hermann Gehring

Parallel greedy algorithms for packing unequal circles into a strip or a rectangle

Timo Kubach, Andreas Bortfeldt and Hermann Gehring

Abstract:
Given a finite set of circles of different sizes we study the Strip Packing Problem (SPP) as well as the Knapsack Problem (KP). The SPP asks for a placement of all circles (without overlap) within a rectangular strip of fixed width so that the variable length of the strip is minimized. The KP requires packing of a subset of the circles in a rectangle of fixed dimensions so that the wasted area is minimized. To solve these problems some greedy algorithms were developed that enhance the algorithms proposed by Huang et al. [15] Furthermore, these greedy algorithms were parallelized using a master slave approach and following the subtree-distribution model. The resulting parallel methods were run on a dualcore 64 bit PC under Linux. For the six instances introduced by Stoyan and Yaskov [18] competitive results in terms of solution quality as well as runtime effort were achieved. In order to stimulate more detailed comparisons of different methods dealing with the problems studied here two sets of 128 instances each for the SPP and for the KP were generated. For this several parameters of the instances such as total number of circles, number of different circle types, radius of smallest and of biggest circle, respectively, were varied in a systematic manner. Results for these new benchmark instances are also reported and analysed.

Key words:
Packing, circles, strip packing problem, knapsack problem, greedy algorithm, parallelization.

Fakultät für Wirtschaftswissenschaft, FernUniversität in Hagen
Profilstr. 8, D-58084 Hagen, BRD

Tel.:   02331/987–4433
Fax: 02331/987–4447
E-Mail: andreas.bortfeldt@fernuni-hagen.de

Parallel greedy algorithms for packing unequal circles into a strip or a rectangle

Timo Kubach, Andreas Bortfeldt and Hermann Gehring

1   Introduction

This paper deals with the two-dimensional (2D) Strip Packing Problem (SPP) and the constrained 2D Knapsack Problem (KP) where unequal circles are the small objects to be packed. Given a finite set of circles, the SPP asks for a non-overlapping placement of all circles within a rectangular strip of fixed width so that the variable length of the strip is minimized. The KP requires packing a subset of a given set of circles in a rectangle of fixed dimensions without overlap so that the wasted area is minimized. The KP is called constrained since each given circle may be used only once. The problems described are known to be NP-hard [16].

According to the new typology of Cutting and Packing (C&P) problems proposed by Wäscher et al. [20], the Knapsack Problem is called a Circular Single Large Object Placement Problem (C-SLOPP) if the set of circles is weakly heterogeneous and a Circular Single Knapsack Problem (C-SKP) if the circle set is strongly heterogeneous. The Strip Packing Problem is referred to as a Circular Open Dimension Problem (C-ODP) in the new typology. KP and SPP or related problems with differently sized circles occur in several industries (cable, glass, paper, textile, wood, etc). For example in the pulp industry, packing of cylinders of pulp with different diameters and equal lengths into a shipping container is a common problem [7].

In this paper, two greedy algorithms [12] for the KP and for the SPP are presented. Both methods are based on the algorithms of Huang et al. [15]. These authors adopt the idea of placing the next circle according to the maximum hole degree (MHD) rule, which is inspired from human activity in packing, and they apply a forward-looking search strategy [13,14]. Here, a continuous threshold parameter is introduced that serves to control the trade-off between solution quality and runtime. To take advantage of the latest advances in processor design marking the beginning of the multicore era, the two algorithms proposed here are parallelized using a master slave approach and applying the subtree-distribution model.

The performance of the algorithms is compared to the published methods using the six benchmark instances from Stoyan and Yaskov [18]. In order to stimulate more detailed

comparisons of different methods dealing with the problems studied here two sets of 128 instances each for the SPP and for the KP are additionally generated. To this end several parameters of the instances, such as total number of circles and number of different circle types, are varied in a systematic manner.

The paper is organized as follows. In Section 2, formal definitions of the SPP and the KP are given, followed by a literature overview in Section 3. The algorithms developed are presented in their sequential versions in Section 4. The parallelization of the algorithms is discussed in Section 5, while the new benchmark instances for both problems are introduced in Section 6. In Section 7, the experimental results are presented, analysed and compared to related work in the literature. Finally, the paper is summarized in Section 8.

2   Problem definitions

The constrained 2D Knapsack Problem with differently sized circles can be formally stated as follows. Suppose a rectangular container of given width w and length l, and a finite set $C = \{1, 2, \ldots, n\}$ of n circles of not necessarily equal radii $r_1, \ldots, r_n$. The container is embedded in the 2D Euclidian plane as shown in figure 1. The placement of a circle i is denoted by a triple $p = (i, x_i, y_i)$ where x_i and y_i are the coordinates of the centre of i. Hence, a packing plan (in short plan) P including all circles of a subset $C' \subseteq C$ is given by a set of triples $P = \{(i, x_i, y_i) \mid i \in C' \subseteq C\}$. The density d_P of the plan P measures the fraction of the container area covered by circles and is defined as

$$d_P = \sum_{i \in C'} \frac{\pi \cdot r_i^2}{l \cdot w} = \frac{\pi}{l \cdot w} \sum_{i \in C'} r_i^2. \tag{1}$$

The problem is to determine an optimal packing plan P for a certain subset $C' \subseteq C$, i.e.,

maximize d_P (2)

such that the following constraints are met

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - r_i - r_j \geq 0,\ i \neq j,\ i, j \in C', \tag{3}$$

$$d_{i,s1} = x_i - r_i \geq 0, \forall i \in C', \tag{4}$$

$$d_{i,s2} = w - y_i - r_i \geq 0, \forall i \in C', \tag{5}$$

$$d_{i,s3} = y_i - r_i \geq 0, \forall i \in C', \tag{6}$$

$$d_{i,s4} = l - x_i - r_i \geq 0, \forall i \in C'. \tag{7}$$

Constraint (3) requires that the circles placed in the rectangle do not overlap. Constraints (4) to (7) assure that all packed circles lie completely within the container. The constraints are illustrated by figure 1. If a placement p satisfies constraints (3) to (7) it is said to be a feasible placement. The feasibility of a packing plan P is analogously defined. A plan that accommodates all given circles in the container is obviously a global optimal solution and it is called, hereafter, a successful plan.
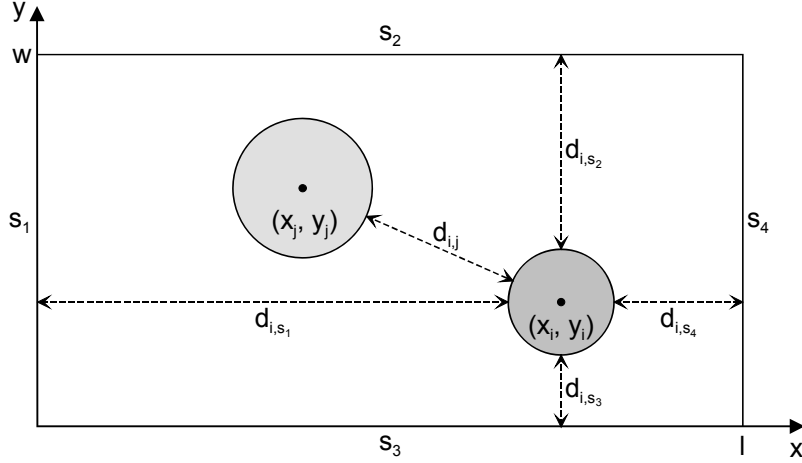

Figure 1: Constraints of the KP with differently sized circles.

The Strip Packing Problem with differently sized circles can be formulated in a similar way to that of the KP. Again, a set C of n circles and a rectangular container embedded in the Euclidian plane are given, as shown in figure 1. The container – also called strip – now has a fixed width w but a variable length l. The SPP requires finding a packing plan that includes all circles, i.e. $C' = C$, in such a way that the density d_P is maximized and the constraints (3) to (6) are met. For each compete plan P, the density is calculated (cf. equation (1)) using the length l of the rectangle enveloping P, i.e.,

$$l = \max_{i \in C}(x_i + r_i). \tag{8}$$

Thus, constraint (7) no longer applies and a placement (or a plan) is feasible with regard to the SPP if only the constraints (3) to (6) are met. Of course, a packing plan with maximum density guarantees a strip of minimum length at the same time.

3   Literature overview

In the literature most research on circle packing problems focuses on packing equal circles [3,6,9]. The approaches discussed in those publications heavily depend on the congruence of the circles and hence are not applicable to problems dealing with unequal circles. Up to the present, only a few solution methods are known that address C&P problems with differently sized circles. George et al. [7] were faced with a 2D Knapsack Problem with special stability requirements from industrial practice. They propose a set of packing rules that form the core of several heuristic procedures. Among them a random search procedure and a genetic algorithm (GA) [5] yielded the best results. Stoyan and Yaskov [18] dealt with the SPP as defined in this paper. A mathematical model of the SPP is formulated and solved by a sophisticated analytical solution method combining the idea of increasing the problem dimension and a reduced gradient method [8], as well as the concept of active inequalities and the Newton method. Hifi and M'Hallah [10] presented a construction heuristic and a genetic algorithm for solving the dual cutting problem of the KP defined in this paper. As often practised in GAs for C&P problems, a chromosome stipulates a certain order in which the circles are packed into the container. Consequently, common operators are applied to generate offspring. For the same cutting problem a simulated annealing algorithm [4] was published by Hifi et al. [11]. Neighbours of a solution are generated by means of elementary geometrical transformations of different types and – as a further feature – infeasible solutions may occur during the search. As the algorithms developed in this work are based on the two greedy algorithms proposed by Huang et al. [15], these algorithms are explained in greater detail. First of all, it should be noted that Huang et al. do not solve any of the optimization problems defined above. Instead, they consider a rectangular container of fixed dimensions, a set C of (generally unequal) circles and solve the following decision problem: Is there a feasible packing plan P placing all circles of C (cf. Section 2)? Although an algorithm to this decision problem can be easily extended to cope with the SPP, Huang et al. did not consider this possibility.

To benefit from human experience in packing, the authors introduce three important concepts: corner placement, hole degree and the maximum hole degree (MHD) rule.

A placement $p = (i, x_i, y_i)$ is called a *corner placement* if it is feasible (i.e., it satisfies the constraints (3) to (7)) and if circle i touches (at least) two items. An item may be another circle or one of the four sides of the rectangle.

Let P be a feasible packing plan and $p = (i, x_i, y_i)$ a corner placement belonging to P. Moreover, let u and v be the two items (circle or side) touching circle i. The *hole degree* λ of the corner placement p is defined as

$$\lambda(p) = \lambda((i, x_i, y_i)) = 1 - \frac{d_{\min}}{r_i}. \tag{9}$$

In equation (9) r_i is the radius of circle i while $d_{\min}$ is the minimal distance from circle i to other circles in P and to the sides of the rectangle with the exception of items u and v, formally:

$$d_{\min} = \min_{j \in \{k \in C \setminus \{i\} | \exists\, p \in P: p = (k,x,y)\} \cup \{s1,s2,s3,s4\} \setminus \{u,v\}} d_{i,j}. \tag{10}$$

The hole degree of a corner placement indicates how close to other circles of a plan (and to the rectangle sides) a given circle is accommodated. The higher the mean hole degree of the placements of a packing plan the higher the density of the plan. Therefore, the hole degree is used to evaluate the benefit of a circle placement in the greedy algorithms. The *maximum hole degree rule* says that given a set of possible (additional) corner placements, the placement with the maximum hole degree should be selected as the next one.

The first greedy algorithm – called B1.0 – consists of a core procedure and a frame procedure:

- The core procedure (B1.0C) takes a packing plan P' as input and provides a complete (i.e. not extendable) packing plan P''. To generate P'' the following step is repeatedly done: all possible corner placements are determined and the MHD rule is applied to implement the next placement. A complete plan is reached if either all n circles are placed (success) or no further corner placements are available for the remaining circles (failure).

- The frame procedure generates step by step so-called initial configurations. An initial configuration consists of two circle placements. The first circle is placed at the bottom left corner (cf. figure 1), and the second circle touches the first one and a side of the rectangle or it does not touch the first circle but two sides of the container. For each possible initial configuration (that plays the role of plan P'), the core procedure B1.0C is called once. If

B1.0C returns with success, the frame procedure (i.e. the algorithm) stops immediately. Otherwise, the algorithm ends if there is no further untried initial configuration.

The second greedy algorithm, denoted by B1.5, has a similar structure to B1.0 and the frame procedure of B1.0 is transferred. Again, the core procedure (B1.5C) transforms a packing plan P' into a complete packing plan P'' by corner placements carried out one after another. The next corner placement $p*$ for an interim plan P_1 is now determined as follows:

- Each corner placement p that is possible with regard to P_1 is tentatively implemented and the extended plan $P_1 \cup \{p\}$ is then completed by means of the core procedure B1.0C. This results in a complete packing plan $P_2(p)$ with a density $d_P(P_2)$.

- At the end, the corner placement $p*$ is selected and finally implemented for which the density $d_P(P_2)$ of the associated complete plan $P_2(p*)$ is maximised.

Obviously, the core procedure B1.5C employs a forward-looking strategy to make the selection of corner placements more sophisticated. While in algorithm B1.0 the decision about the next corner placement only depends on placements implemented earlier in algorithm B1.5, the circles not yet packed are also taken into account.

Huang et al. [15] have shown that the B1.0's time complexity is $O(n^6)$ and B1.5's is $O(n^{10})$ (n stands for the number of circles).

4   Developed algorithms

4.1   Algorithm B1.6_KP for the Knapsack Problem

As the algorithm developed for the KP is an advancement of algorithm B1.5, it is called B1.6_KP. It has a frame procedure as shown in figure 2.

```
Procedure B1.6_KP(in: instance data I, parameter τ , out: packing plan P_best)
P_best := ø;
for (every initial configuration P_init) do
        P_res := B1.6_KP_C(I, τ , P_init);
        if (d_p(P_res) > d_p(P_best)) then
                P_best := P_res;
                // stop immediately if a global optimal solution is reached
                if (P_best includes all given circles) then return P_best; endif;
        endif;
endfor;
return P_best;
end.
```

Figure 2: Frame procedure of algorithm B1.6_KP.

The frame procedure generates step by step different initial configurations as defined in Section 3. Every initial configuration is then extended to a complete packing plan by means of the core procedure B1.6_KP_C and the best packing plan in terms of density is updated whenever necessary and returned at last.

To limit the search effort, only a subset of possible initial configurations is investigated. Let n_{dif} be the number of circle types of a given instance (where a circle type is given by a radius) and let the circle types be sorted by the radius in descending order. For each pair (i, j) of circle types $(1 \leq i \leq j \leq n_{dif})$ at least the initial configuration of type A (circle i in the bottom lefthand corner, circle j in the top righthand corner, cf. figure 3) is probed. Note that a circle type can only be paired with itself if at least two circles of the type are available. If there are no more than 1000 initial configurations of type A, then all possible initial configurations of the types B and C (see figure 3) are also tried.
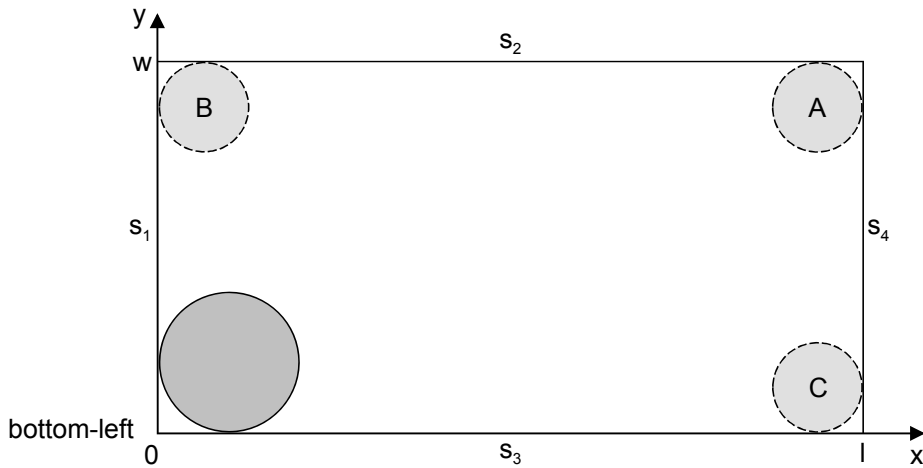

Figure 3: Types of initial configurations for the Knapsack Problem.

Although the frame procedure just described is similar to those of the algorithms B1.0 and B1.5, there are some important differences: Having been adapted to the Knapsack Problem, algorithm B1.6_KP searches for packing plans of maximum density instead of focusing only on plans accommodating all given circles. B1.6_KP is able to deal efficiently with strictly heterogeneous instances (where any two circles are unequal) as well as with non-strictly heterogeneous instances. In the latter case, redundant calculations are consistently avoided only by B1.6_KP and this applies, in particular, to the generation of initial configurations. Finally, the generation of initial configurations is governed by different rules compared to B1.0 and B1.5. In figure 4 the core procedure B1.6_KP_C of algorithm B1.6_KP is listed.

A main feature of the core procedure is the control mechanism introduced with the continuous threshold parameter τ. Let p^* be a possible corner placement with maximum hole degree λ_{max} at a given time, i.e., for a certain cycle of the while-loop. If λ_{max} exceeds the value of the threshold parameter, the circle corresponding to p^* is packed as in procedure B1.0C. Otherwise the forward-looking strategy of procedure B1.5C is used to decide which circle will be placed next. Hence, the core procedure B1.6_KP_C combines the core procedures of the algorithms B1.0 and B1.5 and this combination shows two aspects: From a formal point of view two algorithms are replaced and generalized by one. If threshold τ is set to 1, procedure B1.6_KP_C behaves exactly like B1.5C; if τ is set to a sufficiently small value (e.g., to $1 - (w+l)/\min r_i$) B1.6_KP_C proceeds as B1.0C. More importantly, the threshold parameter allows the trade-off between solution quality and runtime effort to be controlled. The higher the value of τ the higher the packing density that may be expected and the lower τ the faster the search will be finished. However, the amount of computing time saved by a reduction of the threshold value may only be estimated with the help of experimental experience.

```
Procedure B1.6_KP_C(in: instance data I, parameter τ , inout: packing plan P)
determine list L of possible corner placements p = (i, xi, yi) w.r.t. (incomplete) packing plan P and
        calculate the hole degrees λ(p);
while (there are corner placements in L) do
        select placement p* with the maximum hole degree λmax from L;
        if (λmax > τ ) then
                P := P ∪ {p*}; // implement placement p*
                update list L;
        else
                best_density := 0;
                for (every corner placement p in L) do
                        let P' be a copy of P and L' be a copy of L;
                        P' := P' ∪ {p}; // implement placement p tentatively
                        update list L';
                        P'' := B1.6_KP_C2(I, L', P'); // complete plan P'
                        if (P'' includes all given circles) then P := P''; return P; endif;
                        if (dp(P'') > best_density or
                           dp(P'') = best_density and λ(p) > λ(p*)) then
                                p* = p; best_density := dp(P'');
                        endif;
                endfor;
                P := P ∪ {p*}; // implement placement p* finally
                update list L;
        endif;
endwhile;
return P;
end.
```
Figure 4: Core procedure B1.6_KP_C.

The forward-looking strategy adopted from procedure B1.5C was enhanced by a second evaluation criterion for placements: The hole degree is taken as a tie breaker if two completed packing plans have the same density. Procedure B1.6_KP_C2 is called repetitively by B1.6_KP_C and it coincides with procedure B1.0C already explained in Section 3. For the sake of completeness, B1.6_KP_C2 is displayed in figure 5.

```
Procedure B1.6_KP_C2(in: instance data I, list L of corner placements, inout: packing plan P)
while (there are corner placements in L) do
        select placement p* with the maximum hole degree λmax from L;
        P := P u {p*}; // implement placement p*
        update list L;
endwhile;
return P;
end.
```

Figure 5: Core procedure B1.6_KP_C2.

Two details of the implementation should be mentioned. Each time a new (current) best plan is identified it is stored immediately on the hard disk. This feature, not shown in figure 2, serves to strengthen the reliability of the system. The second detail concerns the numerical calculations. If a circle is one of the touched items of a corner placement $p = (i, x, y)$, its computation requires extracting a square root. This may lead to difficulties due to the limited floating point accuracy of computer systems. To tackle this problem, we use a slightly larger radius $r_i' = r_i \cdot (1 + 10^{-13})$ in the computation of the circle's position (x, y) where the factor $(1 + 10^{-13})$ was determined by experiments. Each plan generated is checked to ensure that the constraints (3) to (7) are met and only in this case is a plan accepted as a new best plan. Thus, B1.6_KP only provides solutions that are consistent with the computer's floating point model.

To determine the time complexity of algorithm B1.6_KP we adopt the results of Huang et al. [15]. The core procedure B1.6_KP_C proceeds either as procedure B1.0C having time complexity $O(n^4)$ or as procedure B1.5C that runs in $O(n^8)$. Further actions in B1.6_KP_C, namely generating list L and computing of hole degree values as well as determining the MHD placement, are of lower complexity. Thus B1.6_KP_C runs in $O(n^8)$ too. As the number of initial configurations is bounded by $O(n^2)$ and B1.6_KP_C is called once per configuration, the time complexity of B1.6_KP is $O(n^{10})$.

4.2   Algorithm B1.6_SPP for the Strip Packing Problem

The algorithm proposed for the SPP is termed B1.6_SPP. It consists of a frame procedure and two core procedures that are similar to those of algorithm B1.6_KP. Therefore, only the main differences between the corresponding procedures are outlined in what follows.

Again, the frame procedure of B1.6_SPP generates initial configurations and a core procedure called B1.6_SPP_C tries to extend each configuration to a complete packing plan. Now a complete plan should include all given circles and the search is for a complete plan of minimum length. Hence, initialization and update of the best plan are changed accordingly and the (current) best plan is stored together with its length l_{best}. Checking a best plan for global optimality is omitted, since there is no easy way to identify global optimal solutions to the SPP. As in the KP method, only a subset of initial configurations is investigated and an analogous rule is applied to select initial configurations. However, for a given pair of circle types (i, j) ($r_i \geq r_j$) generally up to 15 types of initial configurations are distinguished:

- The types A and B are shown in figure 6. Type C is analogously defined as type B but has the smaller circle in the bottom lefthand corner. Given a pair of circle types, initial configurations of types B and C are generated only if no configuration of type A exists.

- Further configuration types are only applied if the number of all valid configurations (cf. (3) to (7)) of types A to C does not reach 1000. The configuration types D to I are also illustrated in figure 6. The remaining six configuration types result if the circles of each of the configuration types D to I change their positions.
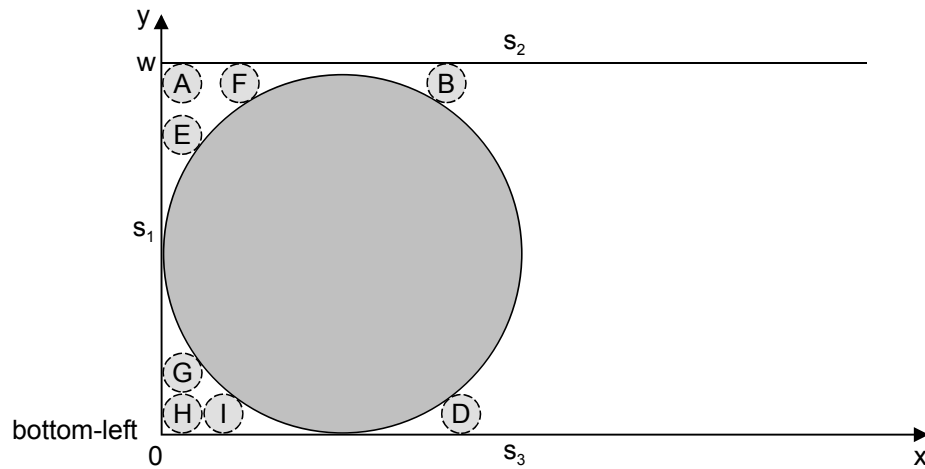

Figure 6: Types of initial configurations for the Strip Packing Problem.

In the core procedure B1.6_SPP_C, a list L of possible corner placements is supplied first, as in B1.6_KP_C, but as there is no length given for the SPP, no side s4 (cf. figure 1) can be considered either as a possibly touched item of a corner placement or in the computation of hole degree values. Moreover, no corner placement touching or overlapping the line $x = l_{best}$ is accepted for list L, since after such a placement the current best solution cannot be improved anymore. The features explained above not only affect the generation of list L but also its later updates. Starting with an initial configuration, B1.6_SPP_C tries to pack all n given circles. Using threshold parameter τ, the next placement taken from the non empty list L is either the one with maximum hole degree or it is stipulated by a modified version of procedure B1.5C (cf. figure 4). However, this process is finished (at the latest) after $n - 2$ circles have been packed. As the computational effort is negligible, the last two circles are placed afterwards choosing the best existing pair of placements in terms of total used strip length. As mentioned above, list L is only filled by corner placements that still allow a new best solution to be obtained. Thus, it may be that L becomes empty before $n - 2$ circles are placed. In this case, a new best solution is out of reach and, for the given initial configuration, B1.6_SPP_C terminates with an incomplete plan that is assigned a sufficiently large pseudo length.

A forward-looking strategy is implemented again by means of a second core procedure termed B1.6_SPP_C2. Different to the corresponding procedure B1.6_KP_C2, this core procedure tries to pack all given circles (i.e., packing all circles is not crossed by a fixed container length) and the special handling of the last two circles is also adopted from B1.6_SPP_C. In B1.6_SPP_C2 the calculation of placements is prematurely aborted if an improvement to the current best plan among all complete plans derived from a fixed partial plan P (cf. figures 4 and 5) is no longer possible.

It is easy to see that B1.6_SPP doesn't differ from B1.6_KP in terms of worst case time complexity. So B1.6_SPP's time complexity is also $O(n^{10})$.

5   Parallelization of the algorithms

B1.6_KP and B1.6_SPP sequentially check a (in most cases large) number of initial configurations. Initial configurations are checked independently of each other. Hence, it is possible to check them in parallel, leading to a runtime reduction while the solution quality is not affected. The parallelization applied here can be described in detail as follows:

- A shared memory master-slave approach was chosen where the communication between processes is kept to a minimum and only a little control effort accounts for the master process. Therefore, the master is also involved in checking initial configurations.

- The shared variables allocated by the master process are primarily used to store the input data (filename of instance, threshold parameter) and the (current) best density (KP) or best length (SPP). The communication between the processes is done asynchronously, as each process decides on its own at which moment shared variables are read or modified.

- The master process as well as the n_{SP} slave processes check initial configurations; in the tests performed here on dualcore PCs only one slave process exists besides the master process ($n_{SP} = 1$). To avoid multiple examinations of initial configurations, the processes mark checked placements by increasing a shared counter variable that is initialized by the master and indicates the next initial configuration to be checked. A new best plan is immediately stored on hard disk, overwriting the old best plan and the shared variable, and the best objective function value is also updated.

- When all initial configurations have been checked, the master process simultaneously terminates all processes. As it is possible to find a successful plan when solving the KP, each process of B1.6_KP has the possibility of preparing all other processes for termination in case of success, while the termination signal is always sent by the master process. This feature is not implemented in B1.6_SPP, as there is no comparable stopping criterion.

The algorithms proposed are construction methods by their very nature instead of local search algorithms. Hence, it is not surprising that the parallelism applied here does not fall into one of the three categories introduced by Crainic and Toulouse [2] for parallel meta-heuristics. However, the algorithms perform a degenerated tree search (with more than one successor only

at the first two tree levels). Correspondingly, the parallelization follows the subtree-distribution model, a well-known approach for parallelizing branch-and-bound methods (cf. [2]).

6   New benchmark instances

New benchmark instances for the SPP and KP with unequal circles are introduced in a similar fashion as proposed in [1] for the 2D SPP with rectangular items. To generate new instances for the SPP (referred to as KBG_SPP), in a first step, four factors (or instance parameters) were identified that probably affect the accessible solution quality and runtime of corresponding solution methods. These factors are: total number of circles n, number of different circle types n_{dif}, radius of smallest circle $r_{\min}$, and radius of biggest circle $r_{\max}$. Here, an equal distribution for the radii is applied, so the average radius is automatically varied by the variation of $r_{\min}$ and $r_{\max}$ as well as the average ratio of circles' radii and strip width. In step 2, multiple values for each of the factors were fixed. To cover a broad spectrum of instances, different values of the factors were determined as shown in table 1. The strip's width was kept constant for all instances ($w = 10$).

n	n_{dif}	$r_{\min}$	$r_{\max}$
25	n	$w/8$	$w/4$
50	$n/2$	$w/16$	$w/6$
75	$n/5$	$w/30$	$w/10$
100	$n/10$		

Table 1: Values for the instances parameters.

Finally, in step 3 just one problem instance was generated at random for each admissible combination of the factor values. This procedure resulted in a total number of $4*4*(3*3-1) = 128$ instances. Note that the combination $r_{\min} = w/8$ and $r_{\max} = w/10$ is invalid ($r_{\min} > r_{\max}$).

To obtain benchmark instances for the KP (referred to as KBG_KP) the KBG_SPP instances were modified only by adding a certain container length. The lengths were defined in such a way that the area of the resulting rectangle is a certain multiple (or fraction) $f_{KP} \in \{\frac{3}{4}, 1, \frac{5}{4}\}$ of the sum of all circles' areas belonging to the instance:

$$A_{rect} = w \cdot l = f_{KP} \cdot \pi \sum_{i=1}^{n} r_i^2 = f_{KP} \cdot A_{circles}. \tag{11}$$

The intention was not to generate more instances for the KP than for the SPP. Therefore, the three selected values for f_{KP} were distributed alternately to the KBG_SPP instances. Note that only KP instances generated with factor $f_{KP} = 5/4$ allow for successful plans.

The new benchmark instances serve to enable more meaningful and more reliable comparisons of solution methods for the SPP and KP with unequal circles. Moreover, these instances can be used for exploring the influence of instance features, such as the heterogeneity of the circle stock, on the solution quality achieved by heuristics. Both sets of benchmark instances are available from *http://www.fernuni-hagen.de/WINF*.

7 Experimental results and analysis

Stoyan and Yaskov [18] introduced six benchmark instances for the SPP (instances SY_SPP). These instances were later modified by adding the strip lengths achieved by Stoyan and Yaskov as container lengths in order to supply six KP instances (instances SY_KP). Further benchmark instances for the KP (called SYH_KP instances hereafter) were introduced by Huang et al. [15]: the SY_SPP instances were handled as decision problems and the minimum lengths for which successful plans were achieved using algorithm B1.5 were added to these instances. The algorithm B1.6_KP was tested on the SY_KP, SYH_KP as well as on the KBG_KP instances. Algorithm B1.6_SPP was tested using the SY_SPP and the KBG_SPP instances. Both of the algorithms were coded in C. The tests were run on identical PCs using SUSE Linux 10.0 operating system (Linux kernel 2.6.13) with an AMD Athlon64 X2 3800+ (dual core) processor running at 2200MHz (overclocked) and 512MB RAM each.

For both algorithms, B1.6_KP and B1.6_SPP, 14 different threshold values were explored: –1000, –1, –0.5, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0. For value $\tau = -1000$ both of our algorithms correspond to method B1.0, and for value $\tau = 1,$ they correspond to B1.5 (cf. Section 4.1). B1.6_SPP yields, on average, almost the same solution quality for $\tau = 0.8$ as for $\tau = 1$ in about one third (37%) of the runtime. $\tau = 0.8$ seems to be a good choice for B1.6_KP, too, as once again a slightly worse solution quality is yielded in much less time (42%) compared to $\tau = 1$. As a consequence of these observations, the results presented in the following subsections were primarily calculated for $\tau = 0.8$.

The following report on the numerical experiments is arranged in two parts. First the results for the new benchmark instances are presented and analyzed. Next, the proposed algorithms are compared to other methods from the literature. For uniformity reasons, the solution quality achieved for SPP instances is also measured in terms of densities. Run times are given in seconds throughout.

7.1 Results and analysis for the new benchmark instances

For an analysis of the influence of different factors on runtime and solution quality, the results obtained with B1.6_KP($\tau = 0.8$) and B1.6_SPP($\tau = 0.8$) were averaged over groups of instances with equal values of instance parameters (see http://www.fernuni-hagen.de/WINF). Table 2 shows the results for groups of KBG_KP instances with equal values for n and n_{dif}.

n	n_{dif}	$d_{avg,fail}$ (%)	$t_{avg,fail}$ (s)	$t_{avg,suc}$ (s)	q_{suc}	d_{avg} (%)	t_{avg} (s)
25	25	81.264	52	0.11	2/2	80.949	39
	12	79.999	3.8	1.1	3/3	80.002	2.78
	5	79.190	0.39	–	0/3	79.190	0.39
	2	72.117	0.02	–	0/2	72.117	0.02
50	50	84.108	1664	14.3	3/3	82.565	1046
	25	83.660	632	0.31	2/3	82.747	474
	10	83.016	26.7	<0.01	2/2	82.263	19.5
	5	79.475	2.5	<0.01	2/3	79.606	1.9
75	75	85.913	17894	9294	3/3	83.696	14669
	37	84.671	10346	<0.01	2/2	83.504	7760
	15	83.433	342	0.1	3/3	82.145	214
	7	82.854	18.9	0.01	3/3	81.785	11.8
100	100	85.092	287372	0.02	2/2	83.820	215529
	50	84.891	16697	2125	3/3	83.056	11232
	20	83.824	1888	0.25	2/3	82.870	1416
	10	83.464	197	<0.01	2/2	82.598	148

Table 2: Averaged results for groups of KBG_KP instances with equal n and n_{dif} values.

$d_{avg,fail}$ and $t_{avg,fail}$ specify the average density and runtime for instances for which no successful plan was achieved, while $t_{avg,suc}$ indicates the average runtime for instances for which a successful plan was found. q_{suc} gives the ratio of the number of instances per group for which a successful plan was found and the number of instances per group for which a successful plan could exist (i.e. $f_{KP} = 1.25$). The average density d_{avg} and runtime t_{avg} consider all instances per group. The following tendencies can be identified:

- The average density d_{avg} increases generally with the number of circle types n_{dif} for constant n and with the total number of circles n for constant n_{dif}. The first trend seems plausible, as a larger heterogeneity of circles allows for a better interlocking of the circles and, furthermore, yields a larger number of possible corner placements during the whole process of creating a plan. Moreover, this trend follows a theoretical prediction stating that the minimal possible waste of a plan decreases with increasing number of circle types (cf. [19], pp. 71 – 73). For instances with a larger total number of circles, the fraction of circles touching one of the sides is lower. As the losses in covered area are generally larger for circles touching the container than for circles only touching circles, the second trend seems plausible, too. Note that the area of the container increases with increasing n (cf. Section 6).

- The average runtime t_{avg} increases when n_{dif} is increased for constant n values, as more initial configurations have to be checked and the list of possible placements has more elements leading also to larger numbers of possible corner placements. Hence, one would expect an increase of t_{avg} with increasing n and for constant n_{dif}, which was indeed observed.

- Due to the way the benchmark instances were generated, there are instances for which successful plans can be found (cf. Section 6). If a successful plan is found, the algorithm stops immediately and no further initial configurations are checked. When solving the SPP, all possible initial configurations are checked. For a better comparison between KP and SPP results, the density and runtime are reported for KBG_KP instances for which the algorithm didn't find a successful plan (and hence checked all possible initial configurations). The average times for finding a successful plan $t_{avg,suc}$ are of less significance and primarily shown for the purpose of future comparisons.

- A comparison of the average runtimes for $n_{dif} = n$ and $n_{dif} = n/10$ for equal n values indicates that the speedup achieved with the avoidance of redundancy in corner placements (initial configurations, list elements) is probably in the range of 1000.

For the KBG_SPP instances no successful plans exist and so only the averaged density and the averaged runtime are displayed. Table 3 shows the results for groups of KBG_SPP instances with equal values for n and n_{dif}.

The average density d_{avg} increases with increasing n_{dif} for constant n (except for $n_{dif} \in \{10,20\}$, $n = 100$) and with increasing n for constant n_{dif} in a similar way and for the same reasons as for B1.6_KP. The average runtime increases monotonically with increasing n_{dif} at constant n, which can be explained in the same way as for B1.6_KP. t_{avg} also increases with increasing n at constant n_{dif}, which can simply be explained by the larger number of placements to be performed for each initial configuration (recall all circles have to be placed when solving the SPP). The speedup achieved with the avoidance of redundancy in corner placements is in the same range as for the KBG_KP instances.

n	n_{dif}	d_{avg} (%)	t_{avg} (s)
25	25	81.763	36
	12	81.599	3.89
	5	78.941	0.25
	2	76.678	0.01
50	50	83.747	1387
	25	83.254	367
	10	82.691	17.7
	5	80.390	1.9
75	75	84.420	19913
	37	84.212	4483
	15	82.989	224
	7	82.148	13.2
100	100	84.689	132786
	50	84.242	12680
	20	83.281	1201
	10	83.577	103

Table 3: Averaged results for groups of KBG_SPP instances with equal n and n_{dif} values.

The runtime effort of the parallel versions of both of the algorithms is reduced to 52% compared to the sequential versions (the corresponding speedup factor is 93%).

7.2 Comparisons to other algorithms

Table 4 shows the comparison of the closely related algorithms B1.6_KP and B1.5 for the SYH_KP instances. The main parameters n, n_{dif}, l_{Huang} of the instances and the density of a successful plan d are also included in the table.

Since B1.5 is only able to solve decision problems, the comparison of the runtimes for finding successful plans is the only way to compare the algorithms. Although B1.6_KP is more general than B1.5 (cf. 4.1) and the lengths given in table 4 are the best ones found for B1.5, B1.6_KP is capable of finding successful plans for each SYH_KP instance. $t_{B1.5}$ and $t_{B1.6\_KP,\tau=0.8}$ indicate the runtimes B1.5 and B1.6_KP($\tau = 0.8$) need to find a successful plan, respectively. Due to the close relation between B1.5 and B1.6_KP($\tau = 1$) caused by the invariable use of the forward-looking strategy (cf. 4.1), $t_{B1.6\_KP,\tau=1}$ is also displayed. The speedup factor $t_{B1.5}/t_{B1.6\_KP,\tau=0.8}$ shows that B1.6_KP($\tau = 0.8$) is at least 130 times and up to about 1000 times faster than B1.5 for the SYH_KP instances. When the difference in performance between Huang et al.'s computer (AMD AthlonXP 2000+, 256MB RAM) and the PCs used here is taken into account (approximately a factor of three according to Dhrystone benchmarks [17]), B1.6_KP($\tau = 1$) is between 25 and 62 times faster than B1.5. This effect can only be explained by a better implementation, as both algorithms do almost the same things and neither the threshold concept nor the avoidance of redundancy according to rather homogeneous instances leads to an advantage for B1.6_KP. For $\tau = 0.8$ the speedup is between 43 and 337 when the difference in machine performance is taken into account.

I	SYH_KP1	SYH_KP2	SYH_KP3	SYH_KP4	SYH_KP5	SYH_KP6
$n\,;n_{dif}$	30;30	20;20	25;25	35;35	100;98	100;99
l_{Huang}	17.291	14.535	14.470	23.555	36.327	36.857
$d(\%)$	84.148	83.660	84.545	84.514	86.113	86.237
$t_{B1.5}(s)$	1628	396	1385	6654	81199	47085
$t_{B1.6\_KP,\tau=1}(s)$	21.17	2.12	13.55	121.39	641.88	489.94
$t_{B1.5}/t_{B1.6\_KP,\tau=1}$	77	187	102	55	127	96
$t_{B1.6\_KP,\tau=0.8}(s)$	7.21	2.91	10.64	20.39	80.34	144.33
$t_{B1.5}/t_{B1.6\_KP,\tau=0.8}$	226	136	130	326	1011	326

Table 4: Comparison of runtimes of B1.5 and B1.6_KP for finding successful plans for the SYH_KP instances.

In table 5 B1.6_KP($\tau = 0.8$) is compared to algorithms solving the Knapsack Problem for the SY_KP instances whose lengths (which differ from the ones shown in table 4) are also shown. d_x and t_x represent the achieved density and the required runtime for the different algorithms

x. Densities written in bold letters indicate that a successful plan was found and $i_x = (d_{B1.6\_KP,\tau=0.8} - d_x)/d_x$ is the gain obtained by B1.6_KP($\tau = 0.8$) over algorithm x.

Besides B1.6_KP($\tau = 0.8$), results are shown for the following methods:

- the simulated annealing algorithm (SA) by Hifi et al. [11],

- the construction heuristic (CH) and the genetic algorithm (GA-BH) by Hifi and M'Hallah [10], who used an Intel Pentium III 733MHz PC for their tests,

- the greedy algorithms B1.0 and B1.5 by Huang et al. [15]; note that results are only available for these algorithms if they were able to solve the related decision problem.

I	SY_KP1	SY_KP2	SY_KP3	SY_KP4	SY_KP5	SY_KP6
l_{SY}	17.491	14.895	14.930	24.355	38.047	38.647
$d_{SA}(\%)$	74.357	69.908	65.385	71.796	80.208	79.453
$i_{SA}(\%)$	11.87	16.78	25.32	13.85	2.51	3.51
$d_{CH}(\%)$	79.582	77.535	79.756	80.307	**82.220**	82.042
$i_{CH}(\%)$	4.53	5.29	2.74	1.78	–	0.24
$d_{GA-BH}(\%)$	80.960	79.846	81.898	80.549	**82.220**	**82.243**
$i_{GA-BH}(\%)$	2.75	2.24	0.05	1.48	–	–
$d_{B1.0}(\%)$	–	**81.638**	**81.940**	**81.738**	**82.220**	**82.243**
$t_{B1.0}(s)$	–	<1	<1	3	2	4
$d_{B1.5}(\%)$	**83.186**	**81.638**	**81.940**	**81.738**	**82.220**	**82.243**
$t_{B1.5}(s)$	186	7	1	2	2	2
$d_{B1.6\_KP,\tau=0.8}(\%)$	**83.186**	**81.638**	**81.940**	**81.738**	**82.220**	**82.243**
$t_{B1.6\_KP,\tau=0.8}(s)$	1.99	0.04	0.01	0.22	0.01	0.01

Table 5: Comparison of methods SA, CH, GA-BH, B1.0, B1.5 and B1.6_KP for the SY_KP instances.

According to the results obtained for B1.5 and B1.6_KP, successful plans can be found for all the SY_KP instances derived from Stoyan and Yaskov's SPP results [18]. B1.0 calculates successful plans for all instances except for SY_KP1. GA-BH proves to be the best of the algorithms (besides B1.6_KP) dealing with the KP as defined in this paper. The solution quality achieved by the other methods is considerably worse. Looking at the runtime effort, B1.6_KP is once again significantly faster than B1.5 and B1.0. If the difference in machine

performance is considered, B1.6_KP is about two orders of magnitude faster than CH and GA-BH. The comparison of algorithm B1.6_SPP($\tau = 0.8$) with Stoyan and Yaskov's algorithm SY [18] by means of the instances SY_SPP is shown in table 6. For both methods and each instance the achieved length (*l*) and the corresponding density (*d*) are displayed, while the runtimes (*t*) are only available for B1.6_SPP. The gain (i_{SY}) of B1.6_SPP($\tau = 0.8$) over SY is defined as before in table 5.

I	SY_SPP1	SY_SPP2	SY_SPP3	SY_SPP4	SY_SPP5	SY_SPP6
l_{SY}	17.491	14.895	14.930	24.355	38.047	38.647
$d_{SY}(\%)$	83.186	81.638	81.940	81.738	82.220	82.243
$i_{SY}(\%)$	1.42	2.47	3.20	2.69	6.10	6.02
$l_{B1.6\_SPP,\tau=0.8}$	17.247	14.536	14.467	23.717	35.859	36.452
$d_{B1.6\_SPP,\tau=0.8}(\%)$	84.365	83.654	84.565	83.938	87.236	87.196
$t_{B1.6\_SPP,\tau=0.8}(s)$	72	3.4	15.2	193	86679	116273

Table 6: Comparison of the solution quality of methods SY and B1.6_SPP for the SY_SPP instances.

The gain values show that B1.6_SPP delivers better solutions than SY for all six instances. Especially for the instances with 100 circles, B1.6_SPP produces significantly better solutions ($i_{SY} > 6\%$). The best lengths achieved by algorithm B1.5 are indicated in table 4 (cf. row l). Comparing them to the lengths given in table 6 for B1.6_SPP($\tau = 0.8$) demonstrates that the latter method generates better SPP solutions except for SY_SPP2 and SY_SPP4. In addition, for the first five SY_SPP instances, B1.6_SPP solves the optimization problem in less or comparable time than B1.5 needs to solve the dedicated decision problem. Moreover, for the threshold value $\tau = 1$, B1.6_SPP produces better solutions than B1.5 for all SY_SPP instances (see http://www.fernuni-hagen.de/WINF). Figure 7 shows the best plan obtained by B1.6_SPP($\tau = 0.8$) for SY_SPP5.
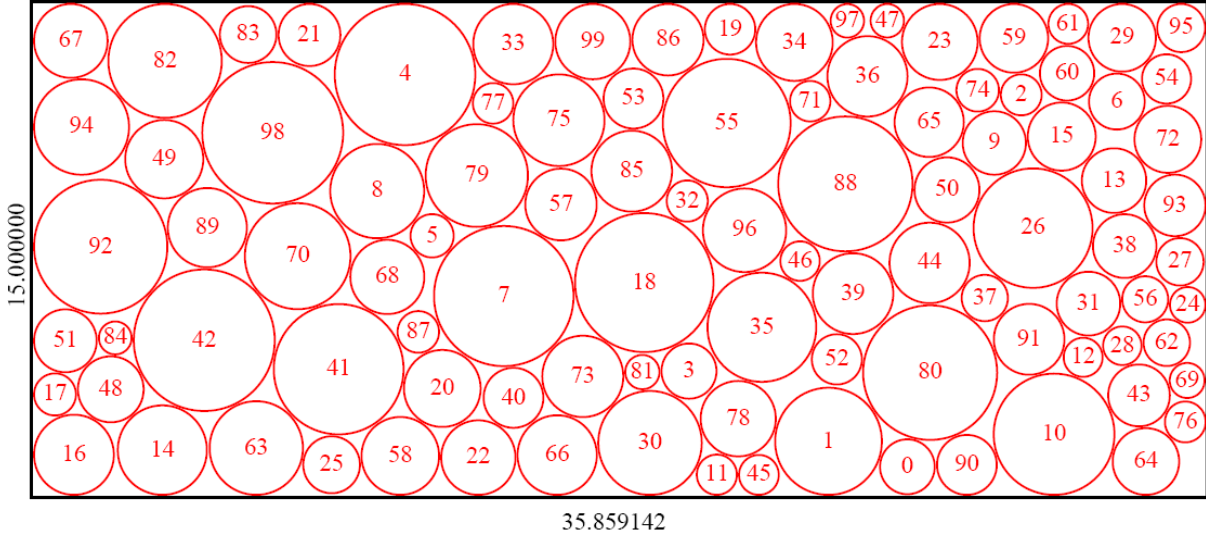
Figure 7: Solution for the instance SY_SPP5, B1.6_SPP($\tau = 0.8$), $d = 87.2364\%$.

8 Summary

This paper presents two greedy algorithms for the Knapsack Problem and the Strip Packing Problem, each with unequal circles. Both algorithms are derived from the methods B1.0 und B1.5 put forward by Huang et al. [15] which, however, only address the corresponding decision problem. Important enhancements to the new algorithms, called B1.6_KP and B1.6_SPP, are dedicated to controlling the trade-off between solution quality and runtime effort, to select suitable sets of initial configurations for constructing complete solutions, and to avoid redundancy in handling of circle placements. Finally, the greedy algorithms are parallelized using a shared memory master-slave approach to investigate initial configurations simultaneously according to the parallelization model of subtree-distribution. A comparison test was carried out that considers practically all existing methods for the circular KP and SPP, respectively, and that is based on the SPP benchmark instances from Stoyan and Yaskov [18] and the KP instances derived from them. Both the KP algorithm and the SPP algorithm achieved the best results in terms of solution quality as well as runtime effort. In particular, the findings reached by the methods B1.0 and B1.5 were considerably improved, e.g., in the calculation of the SYH_KP instances speedup factors between 43 and 337 could be achieved. Furthermore, 128 new benchmark instances each for the circular KP and for the circular SPP were introduced to enable more meaningful and reliable comparisons of solution methods. First results for the new instances generated by B1.6_KP and B1.6_SPP reveal the usefulness of the enhancements mentioned above including the parallelization. In addition, the results evince

some interesting correlations between instance parameter values and the achieved solution quality. It is intended to address the three-dimensional KP and SPP with unequal spheres in the near future.

References

[1] Bortfeldt A and Gehring H: *New Large Benchmark Instances for the Two-Dimensional Strip Packing Problem with Rectangular Pieces*. Proceedings of the 39th Annual Hawaii International Conference of System Sciences (HICCS'06) Track 2, p. 30b, 2006.

[2] Crainic TG and Toulouse M: *Parallel strategies for meta-heuristics*. In Glover F and Kochenberger G (Eds.): *State-of-the-Art handbook in metaheuristics*, 475–513, Kluwer Academic Publishers, Norwell, 2003.

[3] Dowsland KA: *Palletisation of cylinders in cases*. OR Spektrum 13: 171–172, 1991.

[4] Dowsland KA: *Simulated Annealing*. In: Reeves, CR (Ed.): *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, London, 20–69, 1995.

[5] Falkenauer E: *Genetic Algorithms and Grouping Problems*, John Wiley & Sons, Chichester, 1998.

[6] Fraser HJ and George JA: *Integrated container loading software for pulp and paper industry*. European Journal of Operational Research 77: 466–474, 1994.

[7] George JA, George JM and Lamer BW: *Packing different-sized circles into a rectangular container*. European Journal of Operational Research 84: 693–712, 1995.

[8] Gill PE, Murray W and Wright MH: *Practical Optimization*. Academic Press, 1981.

[9] Graham RL and Lubachevsky BD: *Repeated patterns of dense packings of equal disks in a square*. Electronic Journal of Combinatorics 3, Report No. 16, 1996.

[10] Hifi M and M'Hallah R: *Approximate algorithms for constrained circular cutting problems*. Computers and Operations Research 31: 675–694, 2004.

[11] Hifi M, Paschos VTh and Zissimopoulos VA: *Simulated annealing approach for the circular cutting problem*. European Journal of Operational Research 159: 430–448, 2004.

[12] Horowitz E and Sahni S: *Fundamentals of Computer Algorithms*. Pitman, London, 1978.

[13] Huang WQ, Li Y, Gérard S, Li CM and Xu RC: *A 'Learning From Human' heuristic for solving unequal circle packing problem*. In: Hao JK and Liu BD (eds) *Proceedings of the First International Workshop on Heuristics*, Beijing, China. Tsinghua University, Beijing, China, 39–45, 2002.

[14] Huang WQ, Li Y, Jurkowiak B and Li CM : *A two-level search strategy for packing unequal circles into a circle container*. In: Francesca R (ed) *Proceedings of Principles and Practice of Constraint Programming CP2003*, Kinsale, Ireland. Lecture Notes in Computer Science, Vol 2833. Springer, Berlin, 868–872, 2003.

[15] Huang WQ, Li Y, Akeb H and Li CM: *Greedy algorithms for packing unequal circles into a rectangular container,* Journal of the Operational Research Society 56: 539–548, 2005.

[16] Lenstra JK and Rinnooy Kan AHG: *Complexity of packing, covering, and partitioning problems*. In: Schrijver A (ed): *Packing and Covering in Combinatorics*. Mathematisch Centrum, Amsterdam, 275–291, 1979.

[17] SiS Sandra 2005 benchmark suite, SiSoftware, http://www.sisoftware.net, 2005.

[18] Stoyan Yu G and Yaskov G: *A mathematical model and a solution method for the problem of placing various-sized circles into a strip*. European Journal of Operational Research 156: 590–600, 2004.

[19] Toth FJ: *Lagerungen in der Ebene, auf der Kugel und im Raum*. Springer, Berlin, 1953.

[20] Wäscher G, Haußner H and Schumann H: *An Improved Typology of Cutting and Packing Problems*, Working Paper No. 24, Faculty of Economics and Management, Otto von Guericke University, Magdeburg, 2005.