CEJOR (2014) 22:157–186
DOI 10.1007/s10100-012-0282-3

ORIGINAL PAPER

A hybrid algorithm for constrained order packing

Nikolaus Furian · Siegfried Vössner

Published online: 16 March 2013

Abstract Constraint order packing, which is an extension to the classical two-dimensional bin packing, adds an additional layer of complexity to known bin packing problems by new additional placement and order constraints. While existing meta heuristics usually produce good results for common bin packing problems in any dimension, they are not able to take advantage of special structures resulting from these constraints in this particular two-dimensional prolbem type. We introduce a hybrid algorithm that is based on greedy search and is nested within a network search algorithm with dynamic node expansion and meta logic, inspired by human intuition, to overrule decisions implied by the greedy search. Due to the design of this algorithm we can control the performance characteristics to lie anywhere between classical network search algorithms and local greedy search. We will present the algorithm, discuss bounds and show that their performance outperforms common approaches on a variety of data sets based on industrial applications. Furthermore, we discuss time complexity and show some ideas to speed up calculations and improve the quality of results.

Keywords Bin packing · Constraints · Heuristics · Shortest path

1 Introduction

Two-dimensional bin packing problems consist, informally speaking, of allocating a set of small rectangular parts, without overlapping, to a minimum number of rectangular bins. Many variants and different constraints have been discussed in the past,

N. Furian (✉) · S. Vössner
Institut für Maschinenbau und Betriebsinformatik,
TU Graz, Kopernikusgasse 24/III, 8010 Graz, Austria
e-mail: nikolaus.furian@tugraz.at

as for example conflicts, rotation constraints and guillotine cuts. Two-dimensional bin packing has a wide range of real world applications, as for example wood, glass and textile industry, the paging of newspapers and many more. We consider an extension to the single bin-size bin packing problem (SBSBPP) from Wäscher et al. (2007), constrained order packing (COP), that consists of an additional set of order, placement and conflict constraints, that has been introduced by Furian and Vössner (2013).

The problem is motivated by the production of precast concrete parts. Precast-concrete parts are produced on identical rectangular iron bins, and thereby traverse sequently several work stations. At the end of the production line parts are removed from bins and subsumed to so called stacks. Doing so, they must be stapled at so called staple stations in the inverse order they are needed at building sites, where only a limited number of staple stations exist. In particular that means that parts have to be placed in bins according to that order, meaning that parts lower in the stack must be placed in previous bins than parts higher in the stack, where bins are ordered sequently. Furthermore, stacks themselves also underlie an ordering based on due dates.

Bin packing with conflicts describes a problem class where certain small items are in conflict with other items and hence must not be packed in the same large item. In general the existence of conflicts is independent of other restrictions on the problem as dimension or rotation. However, much work has been done on two-dimensional types, as for example by Epstein et al. (2008). Most solution procedures for problems with conflicts consist of the construction of a conflict graph and the usage of graph colouring methods. In the concrete industry, and so also for COP instances, parts are made of different materials with different qualities. Due to production conditions only parts of the same material qualities can be packed in a single bin, which can be seen as classical conflict constraints. Furthermore, parts can be also be of two different types and parts of a certain type must be placed on the left border of a bin.

The problem is defined as follows: We are given an infinite set of ordered identical rectangular bins $B = (b_k), k \in \mathbb{N}$ with length L and width W, a set of n stacks, $S = \{s_1 \dots s_n\}$, where each stack $s_i, i = 1, \dots, n$ consists of $m_i \geq 1$ ordered rectangular parts, $p_{i,1}, \dots, p_{i,m_i}$ of length $l_{i,j} \leq L$ and width $w_{i,j} \leq W$. Furthermore, part $p_{i,j}$ can be of two different types $t_{i,j} \in T = \{(t_1, t_2)\}$ and h different material qualities, $q_{i,j} \in Q = \{q_1, \dots q_h\}$. Further, let $P = \{p_{i,j} | 1 \leq i \leq n, 1 \leq j \leq m_i\}$ be the set of all parts, P_{t_1} the set of all parts of type t_1 and analog P_{t_2} the set of all parts of type t_2 for a given set S. Without loss of generality we can assume that all input data are integers. Some further definitions are required to state the problem and its constraints.

Definition 1 *(Allocation)* For given sets B, S and resulting P an *allocation* A denotes a function $A : P \to \mathbb{N}$ that allocates every part $p \in P$ to an integer k. An allocation A on P is *geometrically feasible* if $\forall k \in \mathbb{N}$ all parts $A^{-1}(k)$ can be fully placed in bin b_k, without overlapping, further, all parts $p \in P_{t_1} \cap A^{-1}(k)$ are placed on the left border of bin b_k, which means that their left-most point is translated to a point $p = (0, y)$ in the coordinate system generated by a bin. The origins of these systems are then given by the left-most points of bins and the orientation follows the bins boundaries. Let

$$A_{k_{max}} = \max\{k \in \mathbb{N} : |A^{-1}(k)| \geq 1\} \tag{1}$$

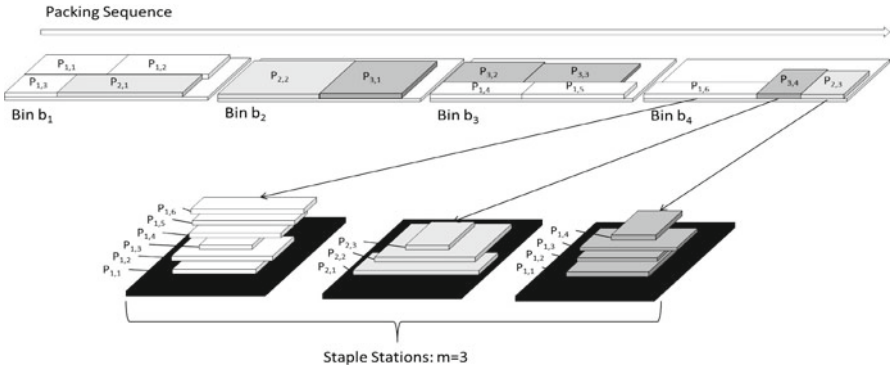

Fig. 1 Structure of COP

be the number of active bins. A stack s_i with $1 \leq i \leq n$ is called open at bin b_k if $A(p_{i,1}) \leq k$ and $A(p_{i,m_i}) \geq k+1$, it is called closed at bin b_k if $A(p_{i,m_i}) \leq k$. Further, let $O_k(C_k)$ be the set of all open(closed) stacks at bin b_k.

An instance of COP is then given by $(B \times S \times Q \times T \times m \times ow)$, where m denotes the maximum number of open stacks and ow the opening window for stacks with $ow \geq m$, which will be described in the following:

Given an instance I of COP the problem is to find the smallest k_0 for which a *geometrically feasible* allocation A with $A_{k_{max}} = k_0$ exists that satisfies the following constraints:

$$A(p_{i,j}) = k \Rightarrow \forall l < j, A(p_{i,l}) \leq k \quad \text{for} \quad i \in \{1, \ldots, n\}, j \in \{1, \ldots m_i\} \tag{2}$$

$$s_i \in O_k \Rightarrow \forall l \leq i - ow, s_l \in C_k \quad \text{for} \quad i \in \{ow + 1, \ldots, n\}, k \in \mathbb{N} \tag{3}$$

$$|O_k| \leq m \quad \text{for} \quad k \in \mathbb{N} \tag{4}$$

$$\left|\left\{q_{i,j} | p_{i,j} \in A^{-1}(k)\right\}\right| = 1 \quad \text{for} \quad k \in \mathbb{N}\ . \tag{5}$$

Constraint (2) ensures the ordering of parts within one stack, (3) simulates the production sequence of stacks themselves. For optimality reasons this sequence might be violated to a certain extent, in particular stacks can be chosen from an interval in the production sequence. This interval, or window, starts with the first unfinished stack and has size ow, where ow is called the opening window. As soon as the first unfinished stack in the production sequence is closed the interval is shifted. Condition (5) ensures that only parts of the same material quality are produced in one bin. The basic principle of COP is also illustrated in Fig. 1.

SBSBPP is a special case of COP where $m_i = 1$ for $i = 1 \ldots n$, $m = n$, $ow = n$ and $|Q| = 1$. Therefore, COP is NP hard in the strong sense, since SBSBPP is such.

Several heuristics for bin packing have been introduced over the last decades. Lodi et al. (1999a,b, 2004) stated a Tabu Search Framework for different forms of bin packing, Babu and Babu (1999, 2001) and Iima and Yakawa (2003) introduced a Genetic algorithm, Wu et al. (2003) improved the nesting heuristic of the approach from Babu and Babu (1999) and stated a Simulated Annealing algorithm,

Caprara and Pferschy (2004, 2005) solved the one-dimensional bin packing problem using a SubSetSum method. Mack and Bortfeldt (2012) construct a heuristic especially for large packing problems consisting of more thant 1,500 small items, just to mention a few. Furian and Vössner (2013) classified different approaches into sequence and set based approaches and adapted them to solve COP. Although Sequence based heuristics showed to be more competitive in terms of results, they were not able to benefit from the special structure of COP. Whereas the SubSetSum algorithm based on ideas from Caprara and Pferschy (2004) produced comparative good results requiring small computational effort. Caprara and Pferschy (2004) solve the classical one dimensional bin packing problem. Bins are filled one bin at a time, putting in the bin a subset of (still unpacked) items of maximum weight not exceeding the bin capacity. Packing items in each bin requires to solve a version of the SubSetSum problem, which can formally be described as follwos: a set of integers and an integer s, does any non-empty subset sum to s? Subset sum can also be thought of as a special case of the knapsack problem.

The adapted version by Furian and Vössner (2013) fills bins one at a time by computing all subsets of unplaced parts and allocating the best set, that can be *geometrically feasible* translated to a single bin, to an additional active bin. The advantage of this approach is that (2), (3), (4) and (5) can be used to reduce the number of possible subsets of unplaced parts. In particular Furian and Vössner (2013) propose that only combinations of ranges for each stack instead of subsets have to be computed when considering (2). Furthermore, (5) can be used to tighten this ranges notably and the number of ranges bigger than zero can be limited by (3) and (4).

Considering the constraints from above separately more or less literature can be found on problems with the same or similar restriction. The restriction on different material qualities is a classical conflict constraint which has been discussed by Epstein and Levin (2007) or Epstein et al. (2008). Lower bounds for problems with conflicts were more recently introduced by Khanafer et al. (2012), but in general existing literature on conflict constraints, especially for two-dimensional problems, is quite little. The constraint 3 which imposes that stacks are produced in a certain order is similar to so called due-date-constraints where certain commissions have to be packed or produced with regard to given order or time limits. They have received very little attention so far, some rare examples are Reinertsen and Vossen (2010) or Johnston and Sadinlija (2004). Rinaldi and Franz (2007) introduce a problem where small items are partitioned into different types and at most two types can be produced at the same segment within a strip packing problem. The proposed constraints are quite similar to the limited number of open stacks 4. They refer to these restrictions as sequencing constraints, what is a bit confusing since this term is often used for constraints that restrict the position of small items in bins or strips. Iori and Martello (2010) review work that has been done on routing problems that consist also of a vehicle loading problem. There small items must be packed in a certain order into the vehicle to avoid reordering of items while unloading. Augustine et al. (2006) transform scheduling constraints into two-dimensional packing problems and introduce precedence constraints and release times. Those constraints work similar to the unloading constraints and ensure that tasks are done in the right order. However, the sequencing of small items within COP is not imposed on positions of them but on the allocation to bins, what results in combinatorial challenges rather than geometric issues. Further, COP

combines all these different restrictions on combinations and sequencing what leads to a whole new complexity and difficulty of the problem.

In Sect. 2 we use these ideas to construct a directed network graph where nodes represent possible allocation status of stacks during calculations. Section 3 provides a search algorithm on that graph that is based on standard network search algorithms, in Sect. 4 we discuss lower bounds and worst case scenarios for the proposed algorithm. In Sect. 5 we introduce some techniques to speed up calculations and improve results. The performance of our algorithm compared to approaches proposed by Furian and Vössner (2013) is evaluated in Sect. 7.

2 Network graph

During each iteration of the SubSetSum algorithm all possible subsets of unplaced parts are computed. The best that can be *geometrically feasible* allocated to a single bin is then translated to a new active bin. Note that the progress of the algorithm could be measured by the set of unplaced part or respectively the set of already allocated parts. More precisely, at each iteration, the set of placed parts can be described by a progress vectors which is defined as follows. For the following discussion let $I = (B \times S \times Q \times m \times ow)$ be an instance of COP.

Definition 2 *(Progress Level)* Let $\tilde{S}$ be a subset of S where each $s_i \in \tilde{S}$ consists of parts $p_{i,1} \ldots p_{i,\tilde{m}_i}$, with $0 \leq \tilde{m}_i \leq m_i$. If there exists a feasible allocation $\tilde{A}$ on the sub problem $(B \times \tilde{S} \times Q \times m \times ow)$ subset $\tilde{S}$ is called a progress level of I, the vector

$$v_{\tilde{S}} = (\tilde{m}_1, \ldots, \tilde{m}_2) \tag{6}$$

its progress vector and

$$AP_{\tilde{S}} = \{p_{i,j} : 1 \leq i \leq n, 1 \leq j \leq \tilde{m}_i\} \tag{7}$$

the set of allocated parts. Further, let N be the set of all progress Levels and $\tilde{S}_1, \tilde{S}_2 \in N$. We say that $\tilde{S}_2$ dominates $\tilde{S}_1$, $\tilde{S}_1 \leq \tilde{S}_2$, if

$$v_{\tilde{S}_1,i} \leq v_{\tilde{S}_2,i} \quad \text{for i} = 1, \ldots, \text{n}. \tag{8}$$

We now consider the directed graph $G = (N, E)$ where each node $\tilde{S} \in N$ represents one progress level of I and the set of edges, E, is given by

$$E = \left\{ \left(\tilde{S}_1, \tilde{S}_2 \right) | \tilde{S}_2 \geq \tilde{S}_1 \quad \text{and all parts in} \quad AP_{\tilde{S}_2} \backslash AP_{\tilde{S}_1} \quad \text{can be placed in a single bin} \right\} \tag{9}$$

Note that obviously the placement of parts must satisfy all required constraints on the layout, namely that parts are placed fully within the bin, that parts do not overlap and parts of type t_1 are placed at the left border if the bin. Informally speaking an edge exists if and only if all parts that have been allocated at $\tilde{S}_2$ but not at $\tilde{S}_1$ can be placed in a single bin. The length of an edge e is chosen as

$$|e| = 1 - \frac{\sum_{p_{i,j} \in AP_{\tilde{S}_2} \setminus AP_{\tilde{S}_1}} l_{i,j} w_{i,j}}{LW}, \tag{10}$$

1 minus the fraction of the unused area when placing all parts to get from $\tilde{S}_1$ to $\tilde{S}_2$ in a single bin. Obviously, there is no edge $e \in E$ with $|e| < 0$ if. We note that the size of G strongly depends on m, ow and most of all on the dimension of bins and parts. Further, one can see that $\tilde{S}_0 = \{\}$, the initial allocation, is the source node and $\tilde{S} = S$, the final allocation, is the sink node of G. Every feasible allocation A for I can be represented by a path from the initial allocation to the final allocation, by translating each edge to a single bin, and the other way round. The problem of finding an allocation with smallest A_{max} can then be formulated as the problem to find the shortest path from S_0 to S in G. In the next section we will discuss a standard algorithm for shortest path problems and apply adapted versions on the graph from above.

3 A relaxation of *A**

The idea to translate bin packing-, or more precisely stock cutting problems to a network search problem can already be found in Albano and Sapuppo (1980). Albano and Sapuppo (1980) consider the irregular Strip Packing problem where irregular shaped parts are placed on a strip with fixed width and infinite length. The goal is to find an allocation that minimizes the used length of the strip. Nodes in their network represent placed parts and their orientations. The shortest path from the initial calculation to the final allocation is obtained by the A^* algorithm. However, they conclude that the network is too large for finding optimal solutions with A^* and propose some techniques to speed up calculations. Thereby it cannot be granted any more that an optimal solution is found, but computational results proof that the competitive ability of the approach. We will summarize the principles of the A^* algorithm, discuss heuristic variants and the arising difficulties for COP. Afterwards we improve some of the strategies to speed up calculations proposed by Albano and Sapuppo (1980), to fit our problem.

3.1 Dijkstra's algorithm and A^*

Dijkstra's algorithm was first introduced by Dijkstra (1959) and computes the shortest path from a starting node v_0 to either all other nodes $v \in V$ with $v \neq v_0$, or to a destination node v_d for a directed graph $G = (V, E)$ with nonnegative length of edges $e \in E$. The two versions only differ in their termination criteria; due to the needs of COP we consider only the algorithm to find the shortest path between two nodes. Informally speaking it consists of a set of labeled nodes LA and a set of expanded nodes O. At each iteration the node $v \in LA$ with the smallest label is selected, labels of all adjacent nodes are updated and v is put in O. The algorithm terminates as soon as v_d is selected from LA. Let λ_v denote the label of node v, ρ_v its predecessor and $|(v, \tilde{v})|$ the length of edge $(v, \tilde{v})$. Algorithm 1 shows the pseudo code of Dijkstra's algorithm.

Algorithm 1 Dijkstra's algorithm

```
1: v = v_0; λ_ṽ = ∞ ∀ṽ ≠ v ∈ V; ρ_v = NULL; LA = {v}, O = {}
2: while v_d ∉ O do
3:   select and remove v ∈ LA with smallest λ_v and put it in O.
4:   for all (v, ṽ) ∈ E do
5:     if ṽ ∉ O and λ_v + |(v, ṽ)| < λ_ṽ then
6:       λ_ṽ = λ_v + |(v, ṽ)|; ρ_ṽ = v.
7:       if ṽ ∉ LA then
8:         insert node ṽ in LA.
9:       end if
10:    end if
11:  end for
12: end while
```

A drawback of this algorithm is that it expands nodes regardless of the probability that they lie on the optimal path from v_0 to v_d. Consider the example where one wants to compute the shortest path between two cities in a road network and the destination is located south of the origin. Obviously nodes north of the start node are less likely on the shortest path and therefore, may not need to be expanded. Furthermore, determining the node with smallest label in L has high computational costs. Several techniques to reduce computational effort have been introduced in the past, for an general overview see for example Fu et al. (2006). Basically, they can be classified whether they try to reduce the search costs for finding the node with the smallest label, by using special data structures, or if the search space of expanded nodes is tightened. As we will see, the use of special data structures, as for example buckets or threshold list, is not essential for the algorithm we propose for COP. However, since the network described in Sect. 2 is vast for industrial applications, methods to cut down the number of expanded nodes are from great interest. Hart et al. (1968) proposed the use of heuristic estimators $est(v, v_d)$ for the path from v to v_d. The label of a node v in Algorithm 1 is substituted by $f_v = \lambda_v + est(v, v_d)$, the sum of the length of the current path from v_0 to v, the label from the Algorithm 1 and the estimated value $est(v, v_d)$. This sum represents the probability of node v to be on the desired shortest path. The search algorithm based on these ideas, called A^*, selects node v with the smallest value f_v from set LA and updates its predecessors. Algorithm 2 shows the pseudo code of A^*.

Algorithm 2 A^*

```
1: v = v_0; λ_ṽ = ∞, f_ṽ = ∞ ∀ṽ ≠ v ∈ V; ρ_v = NULL; LA = {v}, O = {}
2: while v_d ∉ O do
3:   select and remove v ∈ LA with smallest f_v and put in O.
4:   for all (v, ṽ) ∈ E do
5:     if ṽ ∉ O and λ_v + |(v, ṽ)| + est(ṽ, v_d) < f_ṽ then
6:       λ_ṽ = λ_v + |(v, ṽ)|; f_ṽ = λ_v + |(v, ṽ)| + est(ṽ, v_d); ρ_ṽ = v.
7:       if ṽ ∉ L then
8:         insert node ṽ in LA
9:       end if
10:    end if
11:  end for
12: end while
```

Despite to so called pruning algorithms where whole areas of the network are neglected during search, Hart et al. (1968) proofed that A^* is still admissible, meaning that it always finds optimal paths, if *est* always underestimates the real lengths. In that case *est* is called conservative. For other methods to limit the search area as bi-directional search, sub goal methods or hierarchical search methods the reader is referred to Fu et al. (2006).

3.2 A^* for COP

Some major problems arise when attempting to compute the shortest path through networks representing an instance of COP. Real world data sets often consist of more than 20 stacks and 200 parts. Obviously the corresponding networks consist of a vast number of nodes. Moreover, it would be too time consuming to check all possible edges on their existence, meaning to check if parts can actually be placed in a single bin. A advantage of A^* is that one does not necessarily need to know the whole network in advance. Whenever a node S_c is selected from LA, we calculate all possibly adjacent nodes. Meaning that we calculate all progress steps that could be reached from S_c without violating (2), (3), (4) or (5) and the resulting edge would have length smaller than one. Let S_a one of these possible adjacent nodes of S_s. Still there might be no feasible allocation for parts $AP_{S_a} \backslash AP_{S_c}$ in a single bin. However, checking the existence of each edge obtained in the described way has immense computational costs and therefore, is done only if required in the next steps. Furthermore, we mark node as checked and unchecked to avoid allocating the same set of parts more than once. For each adjacent progress level S_a the following label-update procedure is performed. If S_a is already in O we move on to the next node. Otherwise if S_a has not been added to LA yet, we label S_a, mark it as unchecked and put it in L. For $S_a \in L$ we distinguish the following three cases:

1. $\lambda_{S_a} > \lambda_{S_c} + |(S_c, S_a)|$
2. $\lambda_{S_a} \leq \lambda_{S_c} + |(S_c, S_a)|$ with S_a unchecked
3. $\lambda_{S_a} \leq \lambda_{S_c} + |(S_c, S_a)|$ with S_a checked

In the first case we try to allocate parts $AP_{S_a} \backslash AP_{S_c}$ to a single bin , if possible the label and predecessor of S_a are updated and latter is marked as checked. In the second case we try to allocate parts $AP_{S_a} \backslash AP_{\rho_{S_a}}$ where ρ_{S_a} denotes the current predecessor. If possible the node is marked as checked and we move on, else label and predecessor are updated and S_a is marked unchecked. The third case requires no action since we already have found a feasible and better path to S_a. Note that it is still an admissible search algorithm when using conservative estimators. The pseudo code of the procedure described above is provided by Algorithm 3. Where ALLOC is an exact subroutine to allocate parts in a bin that is based on the heuristic method of Furian and Vössner (2013). It places parts in a given sequence on an infinite strip of Width W and checks whether the right most point of all parts exceeds the bin length L. Instead of using heuristic methods, as Tabu search, to control the search over different sequences a enumeration scheme is used. To ensure that an exact 'yes or no' solution to the decision problem whether a set of parts can be allocated in single bin it is important that the

method is able to branch not only over the position of a part in the sequence but also over the position in the bin. Therefore, only positions that result in touching at least two other parts or one other part and the bin are considered. As the calls for ALLOC usually contain relatively small sets of parts this approach is still practicable. However, the attempt to find optimal solutions for COP is only of theoretical interest and for practical heuristic relaxations of the RA^* algorithm, as described in Sect. 3.3, heuristic methods can be used.

Algorithm 3 A^* for COP

$S_c = S_0;\ \lambda_{\tilde{S}} = \infty,\ f_{\tilde{S}} = \infty\ \forall \tilde{S} \neq S_c \in N;\ \rho_{S_c} = \text{NULL};\ LA = \{S_c\}, O = \{\}$
while $S \notin O$ **do**
  set $\tilde{E} = \{\}$.
  select $S_c \in N$ with smallest f_{S_c} and put in O.
  calculate all S_a with $\sum_{p \in AP_{S_a} \setminus AP_{S_c}} \frac{\lambda_p w_p}{LW} \leq 1$ and put (S_c, S_a) in $\tilde{E}$.
  for all $(S_c, S_a) \in \tilde{E}$ **do**
    if $S_a \notin O$ **then**
      if $S_a \in LA$ **then**
        if $S_a > \lambda_{S_c} + |(S_c, S_a)| \ \wedge\ \text{ALLOC}\left(\text{AP}_{\text{Sa}} \setminus \text{AP}_{\text{Sc}}\right)$ **then**
          mark S_a as checked; $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$.
        else if $S_a \leq \lambda_{S_c} + |(S_c, S_a)| \ \wedge\ S_a$ marked unchecked **then**
          if ALLOC $\left(\text{AP}_{\text{Sa}} \setminus \text{AP}_{\rho_{\text{Sa}}}\right)$ **then**
            mark S_a as checked.
          else
            $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$.
          end if
        end if
      else
        mark S_a as unchecked; $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$; insert S_a in LA.
      end if
    end if
  end for
end while

The choice of a heuristic estimator *est* is crucial in the proposed approach. In Sect. 4 we provide an idea how to use lower bounds for standard bin packing problems for COP and revisit some results from literature. However, none of the bounds guarantee acceptable run times for real world applications. Therefore, we introduce a heuristic relaxation of A^* where run times are more controllable.

3.3 RA^*: a heuristic relaxation of A^* for COP

As mentioned above computing optimal solutions for real world sized data sets of COP is often not possible in reasonable time. Although the use of data structures designed especially for COP reduce the computational effort significantly, the exponential complexity of the problem extinguishes these reductions for larger problems. Therefore, we introduce a heuristic relaxation of A^* for COP that is based on some ideas from

Albano and Sapuppo (1980). The main feature is that run times of the algorithm are controllable. The algorithm can be scaled between the exact version from last section and the SubSetSum algorithm from Furian and Vössner (2013), where obviously better solutions require more computational effort. Albano and Sapuppo (1980) propose to limit LA to a fixed size, if exceeding this size nodes with worst labels are erased. Further, when the search is at the kth level only nodes with level higher than $k - t$ are expanded, where t is a given threshold and $k - t$ is called Expansion band. We define a data structure for LA that combines and improves these ideas. In addition to $\lambda_{\tilde{S}}$, $f_{\tilde{S}}$ and $\rho_{\tilde{S}}$ we also store the depth in the search tree, $d_{\tilde{V}}$, for each $\tilde{V}$ that is expanded during the search. Instead of a simple list to store nodes in LA we use t sorted lists, where nodes are divided in lists by their depth in the search tree and sorted by increasing label within a single list. Let d_{max} denote the largest depth of all nodes expanded so far, then a node $\tilde{S}$ with depth $d_{\tilde{S}} >= d_{max} - t + 1$ is stored in list $t - d_{max} + d$. Further, after all adjacent nodes of S_c have been inserted to the data-structure, lists 1 to $t - 1$ are resized. Thereby the progress levels with highest labels are removed until no more vertices than a given maximum number, α_i with $i = 1, \ldots, t - 1$, are stored in list i. The maximum number of nodes is obtained by

$$\alpha_i = \frac{n_{max}}{(t - i)}, \tag{11}$$

where t and n_{max} are parameters free to chose. With these parameters one is able to control the total number of expanded nodes and thereby the runtime of the algorithm. Obviously small t, n_{max} lead to small run times and the other way around. Furthermore, the quality of obtained solutions is negative correlated to the computational effort put in. However, as already mentioned by Albano and Sapuppo (1980), small changes of t have often no affect on the number of used bins. Note that since the list of nodes with the highest depth is not restricted the algorithm always terminates finding a feasible solution.

Let I be an Instance of COP. Further, let $SSS(I)$, $A^*(I)$ and $RA^*(I, t, n_{max})$ denote the number of active bins obtained by the SubSetSum-, A^*- and RA^* algorithm. Then

$$SSS(I) = RA^*(I, 1, n_{max}), \tag{12}$$

and

$$\lim_{t, n_{max} \to \infty} RA^*(I, t, n_{max}) = A^*(I). \tag{13}$$

This basically means that choosing $t = 1$ results in the SubSetSum algorithm and that for n_{max} and t chosen large enough RA^* produces optimal results. However, the non existence of approximation bounds is shown in Sect. 4.2.

Since in general no optimal solution is found there is no need to use an exact method to allocate a set of parts to a single bin. Therefore, the sub-routine ALLOC from the last section can be replaced by a heuristic method to save computational effort. We use the same heuristic method as in Furian and Vössner (2013) that is based on Tabu search and 2-exchange operators, see for example Gomes and Oliveira (2002).

4 Lower bounds and worst case scenarios

As the number of expanded nodes is also dependent on the used heuristic estimator est and its accuracy we will revisit some lower bounds for standard bin packing problems and adapt them for COP. Further, we show the non-existence of approximation bounds for both, the SubSetSum- and RA^* algorithm.

4.1 Lower bounds

Over the last years a couple of lower bounds for oriented and non oriented two-dimensioal bin packing problems have been proposed. Non oriented means that parts may be rotated by 90 °, whereas for oriented problems the rotation is fixed. In general all of them could be used for COP right away, as long as they are designed for the two-dimensional version of the problem. However, they do not deal with any type of conflict or ordering constraints and use only geometric information. Therefore, we introduce a simple framework to tighten any bound for the non-oriented two-dimensional bin packing problem by considering different material qualities of parts and the resulting conflict constraints. Let

$$P = P_{q_1} \cup \ldots P_{q_h} \quad \text{with} \quad \mathrm{P_{q_i}} \cap \mathrm{P_{q_j}} = \{\} \quad \text{for i} \neq \text{j} \quad \text{and} \quad \text{i, j} \in \{1, \ldots, \text{h}\} \tag{14}$$

be a partition of parts of an Instance I of COP, where all parts of a subset P_{q_i} have the same material quality,

$$\forall\, w \in \{1, \ldots, h\}\,,\ \forall\, p_{i,j}, p_{\tilde{i},\tilde{j}} \in P_{q_w} \quad q_{i,j} = q_{\tilde{i},\tilde{j}}. \tag{15}$$

Further, if $LB(P)$ is a lower bound for non-oriented two-dimensional bin packing then

$$LB_{\mathrm{COP}}(P) = \sum_{i=1}^{h} LB(P_{q_i}) \tag{16}$$

is a valid lower bound for I.

g problem. A trivial lower bound for all types of bin packing problems is the continuous bound that is given by

$$L_{\mathrm{C}}(P) = \left\lceil \frac{\sum_{p_{i,j} \in P} l_{i,j} w_{i,j}}{LW} \right\rceil. \tag{17}$$

All lower bounds that are discussed in the following are constructed especially for two-dimensional variants of the bin packing. Martello and Vigo (1998) showed that the absolute worst-case performance of LB_{cont} is $\frac{1}{4}$ for oriented- and non-oriented standard two-dimensional bin packing problems, what in general does not hold for COP due to the additional constraints.

Martello and Vigo (1998) and Fekete and Schepers (2000) were the first that provided lower bounds for the oriented problem except the continuous bound. For the non-oriented problem only a few results exist in literature, as for example Dell'Amico et al. (2002), Boschetti and Mingozzi (2003b) and more recently Clautiaux et al. (2007). Since dominance results of Clautiaux et al. (2007) hold only for square bins we summarize results for non oriented problems and use (16) to apply them on instances of COP. Some results consist of a transformation of the original instance to an instance for the oriented problem. Therefore, we first illustrate a lower bound for oriented problems introduced by Carlier et al. (2007). Therefore, let $D^R = (P \times B)$ be an instance of a two-dimensional bin packing problem, where $P = 1, \ldots, n$ is the set of parts with length l_i and width w_i and $B = L \times W$ a bin.

4.1.1 The lower bound L_{CCM} of Carlier et al. (2007) for oriented problems

A lot of bounds for the oriented problem are obtained by applying so called dual-feasible functions (DFF) Johnson (1973) on both dimensions of the instance and calculating the continuous lower bound for the resulting instance, see also Fekete and Schepers (2004).

Definition 3 Let f be a discrete application from $[0, X]$ to $[0, X']$ where X and X' are integers. Then f is called a discrete DFF if

$$x_1 + \cdots + x_k \leq X \Rightarrow f(x_1) + \ldots f(x_k) \leq X'. \tag{18}$$

Furthermore, only DFF that are increasing and super additive are considered, i.e. if f is a given DFF, $x + y < z \Rightarrow f(x) + f(y) \leq f(z)$. Moreover, Carlier et al. (2007) introduce *data-dependent* DFF which have the properties of a DFF but only for a specific instance. We now summarize definitions the DFF (f_0^k ,f_2^k) and DDFF (f_1^k) used by Carlier et al. (2007). Note that f_0^k is a classical family of DFF and f_2^k an improved version of latter introduced by Boschetti and Mingozzi (2003b). Let C be an integer value and $k = 1, \ldots, \frac{C}{2}$ then

$$f_0^k(x) = \begin{cases} 0, & \text{if } x < k, \\ x, & \text{if } k \leq x \leq C - k, \\ C, & \text{if } C - k < x \leq C, \end{cases} \tag{19}$$

and

$$f_1^k(x) = \begin{cases} 0, & \text{if } x < k, \\ 1, & \text{if } k \leq x \leq \frac{C}{2}, \\ M_C(C, J) - M_C(C - x, J), & \text{if } \frac{C}{2} < x, \end{cases} \tag{20}$$

where f_1^k is defined for given integer values C and $c_1, \ldots, c_n$ ($I = \{1, \ldots, n\}$). Further, the set J is given by $J = \{i \in I : \frac{1}{2}C \geq c_i \geq k\}$ and $M_C(X, J)$ denotes the optimal value for the one-dimensional knapsack problem over the instance given by J and X, meaning the maximum number of items $c_i \in J$ that can be packed in a bin of size X. Further, f_2^k is given by

$$f_2^k(x) = \begin{cases} 2\left\lfloor \frac{x}{k} \right\rfloor, & \text{if } x < \frac{C}{2}, \\ \left\lfloor \frac{C}{k} \right\rfloor, & \text{if } x = \frac{C}{2}, \\ 2\left(\left\lfloor \frac{C}{k} \right\rfloor - \left\lfloor \frac{C-x}{k} \right\rfloor\right), & \text{if } \frac{C}{2} < x, \end{cases} \tag{21}$$

then

$$L_{CCM}^{DDFF} = \max_{u\in\{0,1,2\},v\in\{0,1,2\}} \max_{1\le k\le W/2, 1\le l\le L/2} \left\{ \left\lceil \sum_{i\in P} \frac{f_u^k(w_i) f_v^l(l_i)}{f_u^k(W) f_v^l(L)} \right\rceil \right\}, \tag{22}$$

is a valid lower bound for $2PB|O|F$. L_{CCM}^{DDFF} is tightened by using the framework of Boschetti and Mingozzi (2003a) where four different bounds ($L_{BM}^{F,2a}$, $L_{BM}^{F,2b}$, $L_{BM}^{F,3}$ and $L_{BM}^{F,4}$) are considered. The first transforms the instance into a one-dimensional problem by multiplying lengths and widths of parts, the second considers *large*, *tall* and *wide* items separately. The third and fourth bound result from applying f_1^k and a weaker version of f_2^k on both dimensions of the problem, for more details the reader is referred to Boschetti and Mingozzi (2003a), Carlier et al. (2007).

4.1.2 The lower bound LB_{DA}^R of Dell'Amico et al. (2002)

The basic idea is to transform the original instance to an instance consisting of only square parts and calculating a lower bound for latter. The square parts are obtained by a pseudo-polynomial algorithm called CUTSQ by Boschetti and Mingozzi (2003b) that works as follows. Initially parts are considered in the orientation r where $l_i^r \ge w_i^r$. At each iteration the maximum number of squares with length w_j are cut from the current rectangle with dimensions (l_j, w_j). Afterwards the remaining rectangle is rotated by 90°. These steps are iterated as long as $w_j > 1$ of the remaining rectangle. Let M denote the set of square obtained in the way described above and λ_j the length of square f$j \in M$. For a given integer $0 \le q \le \frac{1}{2}W$ M is then partitioned in the following subsets

$$\begin{aligned} S_1 &= \left\{ j \in M : \lambda_j > L - q \right\} \\ S_2 &= \left\{ j \in M : L - q \ge \lambda_j > \frac{1}{2}L \right\} \\ S_3 &= \left\{ j \in M : \frac{1}{2}L \ge \lambda_j > \frac{1}{2}W \right\} \\ S_4 &= \left\{ j \in M : \frac{1}{2}W \ge \lambda_j \ge q \right\}. \end{aligned}$$

Further, let $S_{23} = \left\{ j \in S_2 \cup S_3 : \lambda_j > W - q \right\}$. Then

$$LB_{DA}^R = \max_{0\le q\le \frac{1}{2}W} \left\{ |S_1| + \tilde{L} + \max\left\{ 0, \left\lceil \frac{\sum_{j\in S_2\cup S_3\cup S_4} \lambda_j{}^2 - LW\tilde{L} + \sum_{j\in S_{23}} \lambda_j (W-q)}{LW} \right\rceil \right\} \right\}$$

is a valid lower bound for the non-oriented two-dimensional bin packing. $\tilde{L}$ is given by

$$\tilde{L} = |S_2| + \max\left\{\left\lceil \frac{\sum_{j\in S_3\setminus\overline{S}_3}\lambda_j}{L}\right\rceil, \left\lceil \frac{|S_3\setminus\overline{S}_3|}{\left\lfloor\frac{W}{2}+1\right\rfloor}\right\rceil\right\},$$

and denotes a valid lower bound for squares in $S_2 \cup S_3$, where $\overline{S}_3$ is the set of large parts in S_3 that fit in bins where parts of S_2 have been placed.

4.1.3 The lower bound L^R_{BM} of Boschetti and Mingozzi (2003b)

L^R_{BM} is the combination of two other lower bounds $L^{R,1}_{BM}$ and $L^{R,2}_{BM}$ and is given by the maximum of these. The first, $L^{R,1}_{BM}$, reduces the problem to a oriented problem by transforming parts that might be rotated to the largest square contained. Therefore, dimensions of parts are resized as follows:

$$\bar{l} = \begin{cases} \min\{l_j, w_j\}, & \text{if } l_j \le W \quad \text{and} \quad w_j \le L \\ l_j, & \text{else} \end{cases} \tag{23}$$

and

$$\bar{w} = \begin{cases} \min\{l_j, w_j\}, & \text{if } l_j \le W \quad \text{and} \quad w_j \le L \\ w_j, & \text{else} \end{cases} \tag{24}$$

Since, as mentioned above, orientation is not a issue for the new obtained problem, lower bounds for the oriented two-dimensional bin packing can be used to compute bounds for the non-oriented variant. Boschetti and Mingozzi (2003b) propose the use of L^F_{BM} introduced by Boschetti and Mingozzi (2003a) that basically is the maximum of four different bounds. For more details the reader is referred to Boschetti and Mingozzi (2003a,b). Since L^F_{CCM} dominates L^2_{BM} we apply the modified version of $L^{R,1}_{BM}$ where latter is used.

The second bound, $L^{R,2}_{BM}$, considers both dimension of parts and also possible rotations by 90 ° by them. Therefore, the set of parts is divided into $P' = \{i \in P : l_i \neq \bar{l}_i\}$ and $P'' = P\setminus P'$. Further, let $1 \le k \le \frac{1}{2}W$ and $1 \le l \le L$, then $L^{R,2}_{BM}$ is given by

$$LB^{R,2}_{BM} = \max_{k,l}\left\{\left\lceil \frac{\sum_{i=1}^{n}\mu^{k,l}(i)}{\left\lfloor\frac{W}{k}\right\rfloor\left\lfloor\frac{L}{l}\right\rfloor}, \right\rceil\right\} \tag{25}$$

where

$$\mu^{k,l}(i) = \begin{cases} \min\left\{\eta^{l,L}(l_i) \times \eta^{k,W}(w_i), \eta^{l,L}(w_i) \times \eta^{k,W}(l_i)\right\}, & \text{if } i \in P' \\ \eta^{l,L}(l_i) \times \eta^{k,W}(w_i), & \text{else,} \end{cases} \tag{26}$$

and

$$\eta^{k,X}(x) = \begin{cases} \left\lfloor \frac{X}{k} \right\rfloor - \left\lfloor \frac{X-x}{k} \right\rfloor, & \text{if } x > \frac{X}{2} \\ \left\lfloor \frac{x}{k} \right\rfloor, & \text{else.} \end{cases} \tag{27}$$

4.1.4 The lower bound L^R_{CJE} of Clautiaux et al. (2007)

The idea of Clautiaux et al. (2007) is to transform a given instance $I = (P \times B)$ with n parts and $B = (L \times L)$ to a new oriented problem with $2n$ parts as follows: $\hat{D}^F = (\hat{P}^F \times B)$ with $\hat{D}^F = \{1, \ldots, 2n\}$ such that

- $l_i = l_i$ and $w_i = w_i$ for $i \in \{1, \ldots, n\}$
- $l_i = w_{i-n}$ and $w_i = l_{i-n}$ for $i \in \{n+1, \ldots, 2n\}$

The main result of Clautiaux et al. (2007) is that $\hat{D}^F$ needs at most twice the number of bins needed for D^R. Therefore, any valid lower bound LB^F for oriented two-dimensional bin packing can be used to obtain a valid lower bound for non-oriented problems by calculating $\left\lceil LB^F(\frac{\hat{D}^F}{2}) \right\rceil$. The bound resulting from applying this scheme on L^F_{CCM} of Carlier et al. (2007) is called L^R_{CJE}, see Clautiaux et al. (2007). However, if bins are no squares one dummy item per bin has to be added to the new instance to fill the area exceeding the largest square contained in the bin. Therefore, Clautiaux et al. (2007) state a lifting procedure to compute the right number of dummy items that has to be added. However, dominance results hold only for the case when bins are squares. For rectangular bins and also in the case where some parts must not be rotated L^R_{CJE} shows some drawbacks.

Since in general RA^* does not compute optimal solutions non-conservative bounds could also be used for calculations. To avoid time consuming calculations we suggest to use an estimator of the form

$$est = \alpha(I) L_C, \tag{28}$$

where α is a function over instances of COP depending on n, m, ow and $\mathrm{Var}(|P_{q_1}|, \ldots, |P_{q_h}|)$.

4.2 Worst-case scenarios

Caprara and Pferschy (2004) showed some absolute approximation bound of the SubSetSum algorithm for the one-dimensional bin packing problem, for more details see Caprara and Pferschy (2004). Unfortunately no such bounds exist neither for SubSetSum nor for RA^* for COP as will be shown in the following.

Definition 4 Let $OPT(I)$ denote the optimal solution value for instance I of COP and $z^H(I)$ the solution value obtained by a heuristic algorithm H. The (asymptotic) *worst-case ratio* of H is then given by the smallest constant β^H such that

$$z^H(I) \leq \beta^H \cdot OPT(I) + \alpha. \tag{29}$$

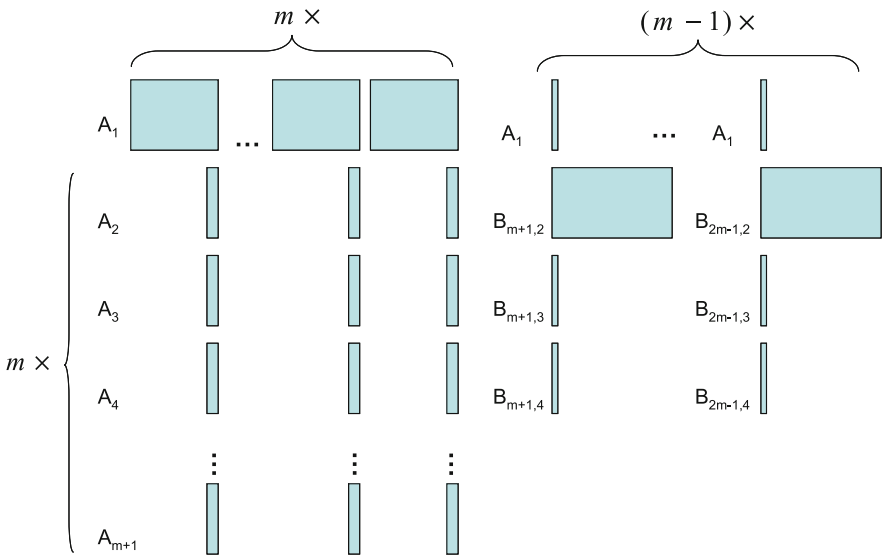

Fig. 2 Dimensions and material qualities of parts

Let's assume that there exists a constant $\tilde{\beta}^{RA^*}$ satisfying (29) for fixed t, $n_{\max}$ and $est \equiv 0$. Consider the following instance AB of COP: $B = L \times W$, the set of stacks, $S = \{s_1, \ldots, s_{2m-1}\}$, the set of different material qualities, all parts are of type t_2, and the stack window, $ow = 2m - 1$. Let the stack sizes be given by

$$m_i = \begin{cases} m+1, & \text{if } i \leq m, \\ 4, & \text{else.} \end{cases}$$

Further, let

$$l_{i,j} = \begin{cases} m+1, & \text{for } i \leq m \text{ and } j = 1, \\ 2, & \text{for } i \leq m \text{ and } j \geq 2, \\ 2m, & \text{for } i \geq m+1 \text{ and } j = 2, \\ 1, & \text{for } i \geq m+1 \text{ and } j \neq 2 \end{cases}$$

and

$$q_{i,j} = \begin{cases} A_j & \text{for } i \leq m, \\ A_j, & \text{for } i \geq m+1 \text{ and } j = 1, \\ B_{i,j}, & \text{else,} \end{cases}$$

and $w_{i,j} = W$ for $i \in \{1, \ldots n\}$, $j \in \{1, \ldots m_i\}$. Figure 2 illustrates dimensions and material quality of parts.

RA^* starts with placing the first part of a stack from $\{s_1, \ldots, s_m\}$ and parts $p_{m+1,1}, \ldots, p_{2m-1,1}$ together in the first bin. Afterwards it places parts $p_{m+1,2}, \ldots, p_{2m-1,2}$ in separate bins and thereby loses the ability to go back to the first node. If we

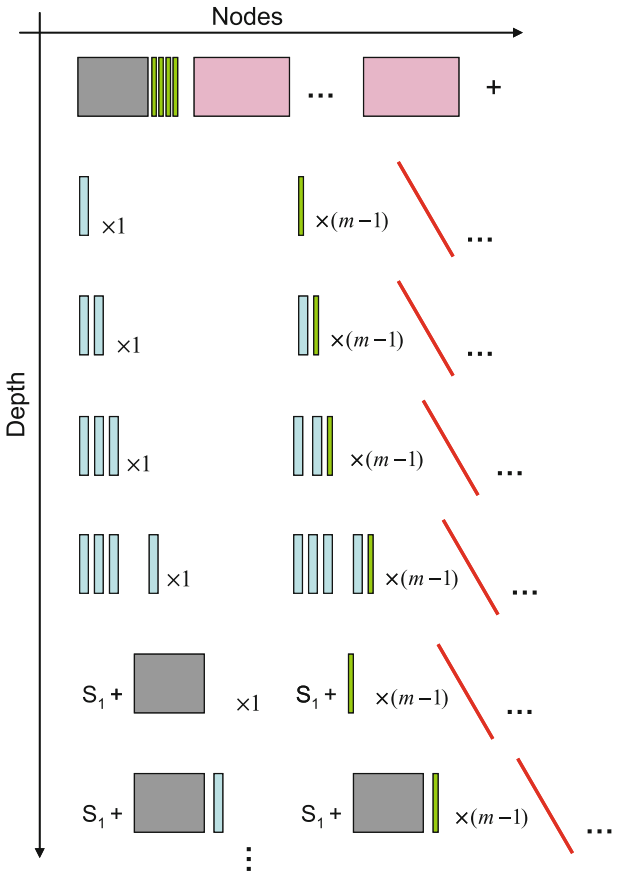

Fig. 3 Search behavior of RA^*

chose $m \geq \max\{n_{max}, t+1\}$ all nodes S_a with $v_{C_a,i} = 4$ and depth d_{S_a} for $i > m$ are deleted from LA when a node with $d_{S_a} + 1$ is labeled. This follows from the fact that there are always $m-1$ nodes S_a with $v_{C_a,i} = 3$ explored and a new depth is always obtained by placing a part from stacks s_1 to s_m. Figure 3 illustrates this behavior. Therefore, RA^* places all parts $p_{i,j}$ with $i \in 1, \ldots, m$ and $j \geq 2$ in separate bins. The overall numbers of bins is given by

$$RA^*(AB) = m + m^2 + 3(m-1). \tag{30}$$

The optimal solution finishes stacks s_{m+1} to s_{2m-1} after the first bins an uses only $m+m-1$ bins for the remaining parts. The overall number bins used is then given by

$$OPT(AB) = m + 3(m-1) + m - 1. \tag{31}$$

Therefore, no approximation bound can exist since $RA^*(AB)$ is increasing quadratically with respect to m where $OPT(AB)$ shows linear behavior.

5 Technical improvements

Intensive testing showed that the calls of the sub-routine ALLOC and the tests if a node is in LA or O need the most computational effort within RA^*. The problem to find suitable data structures for LA and O is also a wide spread issue in the literature for shortest paths. For a more information see for example Fu et al. (2006). We first introduce a technique to reduce the number of calls of ALLOC and afterwards state data-structures for LA and O that take advantage of the special structure of COP.

5.1 Pool of infeasable packings

Obviously, checks if a set of parts can be feasibly placed in a single bin are very time consuming. During calculations it often occurs that a set of parts that has to be allocated has been tested before, or more general that a set of part consisting all of these parts has been allocated before. Furthermore, our experience showed that especially calls of ALLOC that return a negative result, meaning that the considered parts cannot be placed in a single bin, are very time consuming. Therefore, we store a list of sets of parts that have been previously checked and could not be placed in a single bin. ALLOC first checks whether any set stored in this list is a subset of the considered set. Although the search in this list is not cheap in terms of computational effort the overall performance in terms of run times becomes considerable better.

5.2 Special data structure for O

To reduce the effort for searching O for a node S_a we replace the list by an array of lists of the dimension $(m \times n)$. Each entry of the array stores nodes with the corresponding number of open and closed stacks. Thereby, the search for S_a can be reduced to a fraction of the overall number of nodes in O. Since in most practical applications the sizes of stacks themselves do not exceed 20 parts the array is mostly growing with respect to the number of parts to be allocated.

5.3 Special data structure for LA

When expanding a possible edge (S_c, S_a) at line 8 in Algorithm 3 LA has to be searched for containing node S_a. In general it cannot be granted that the existing node has the same depth in the search tree as the node with same coordinates but with updated label. Therefore, nodes may be separated to different list in LA and all list have to be searched. Especially for large instances and higher values of n_{max} and t this search procedure is very time consuming. Since the proposed relaxation of RA^* requires the structure implied by the depths of nodes a similar construct as described in the last section for O is not possible. Keeping an additional array, similar as for O, and storing positions of nodes in both structures yields in a reduction of search costs. However, deleting nodes from LA would require to change position entries of nodes and the savings of computational effort are lost. Therefore, we use chains of nodes, each consisting of nodes with the same number of open and closed stacks and store

the first nodes of these chains in an array. Deleting a node from LA becomes then an issue of deleting it in the sorted lists and changing predecessor and successor of the node in the corresponding chain. The search for a node with particular coordinates is first done in the correct chain. If found, the search in LA can be reduced to a single list since the depth is now known. Furthermore, also the label of the node is known and lists in LA are sorted by increasing labels, so more sophisticated search methods can be used.

6 Standard heuristics from Furian and Vössner (2011)

To compare the performance of RA^* to standard heuristics the methods introduced by Furian and Vössner (2013) are briefly revisited.

6.1 Sequence based algorithms

Sequence based algorithms are hybrid algorithms consisting of two heuristics. A global heuristic modifies the sequence by which parts are placed in bins. For the actual placement of parts an additional, fast and simple sub-heuristic, a so called nesting heuristic, is used. One may notice that the latter could also be used as a stand-alone algorithm.

The main idea of sequence based algorithms for COP is that the ordering of parts implied by (2), (3) and (4) is already satisfied by the sequence in which parts are placed. Therefore, Furian and Vössner (2013) state an extended model for sequences of parts, where sequences themselves already satisfy (2), (3) and (4), meaning that if we place one part per bin in the order given by the sequence, the resulting allocation satisfies these constraints. Such a sequence is called *strongly ordered* and the set of all strongly ordered sequences for an instance of COP is referred to as SWSO. Further, an adapted nesting heuristic that is based on Best Fit nesting heuristics (BFNH) is used. The main difference to the standard nesting heuristic is that the adopted version must not overrule the ordering of parts within stacks and the order in which stacks are opened and closed implied by the sequence. Furthermore, constraint (5) has to be checked before placing a part in a bin. Therefore, for each part $p_{i,j}$, the first bin to consider, $\mathrm{FBC}_{i,j}$ of all bins activated so far is calculated and it is attempted to allocate the part to bin b_k, with $k \geq \mathrm{FBC}_{i,j}$, in which it fits best. $\mathrm{FBC}_{i,j}$ is given by

$$FBC_{i,j} = \begin{cases} \max\{A^{-1}(p_{\tilde{i},m_{\tilde{i}}}) : 1 \leq \tilde{i} \leq n, \tilde{i} \neq i\} & \text{for } \mathrm{j}=1, \\ A^{-1}(p_{i,j-1}) & \text{for } 2 \leq \mathrm{j} \leq \mathrm{m_i} - 1, \\ \max\{\{A^{-1}(p_{\tilde{i},1}) : 1 \leq \tilde{i} \leq n, \tilde{i} \neq i\} \cup \{A^{-1}(p_{i,j-1})\}\} & \text{for } \mathrm{j}=\mathrm{m_i}, \end{cases}$$

where A^{-1} is set to one for unplaced parts. Due to the definition of $\mathrm{FBC}_{i,j}$ placing a part $p_{i,j}$ in a bin b_k with $k \geq \mathrm{FBC}_{i,j}$ (2) is always satisfied and furthermore the order in which stacks are opened and closed in the sequence is retained in the actual packing. If no bin was found a new one is activated after the last active bin and the part is placed in latter.

The search through the space of different sequences is controlled by standard simulated annealing (SA) and genetic algorithms (GA). However, new neighbourhood and

mutation operators were defined which ensure that neighbour and offspring sequences still satisfy the required constraints. For more details the reader is referred to Furian and Vössner (2013).

6.2 Set based algorithms

A major drawback of sequence based approaches is that similar sequences often result in the same allocation. Set based approaches try to overcome these redundancies. They are also hybrid algorithms, consisting of a global heuristic, which selects subsets of parts, and a nesting algorithm, which allocates these parts to one or more bins.

The SubSetSum (SSS) algorithms proposed by Caprara and Pferschy (2004, 2005) fill bins one by one. For eachbin the subset of unplaced parts with biggest weight, e.g. area, is computed, allocated to a new bin and the set of unplaced part is updated. The problem to find the subset with biggest weight is of course NP-hard in the strong sense, but, as mentioned in Caprara and Pferschy (2004), still can be solved with quite low computational effort. For two-dimensional problem, computing the subset with the biggest weight or area that does not exceed the area of a bin is not enough. Certain combinations of unplaced parts might not fit geometrically in the bin or are forbidden due to the additional constraints. Therefore all subsets of unplaced parts whose aggregate area does not exceed the area of a bin are computed. One may note that the number of these subsets can be dramatically reduced for COP by using the constraints checking (2), (3), (4) and (5). For the two-dimensional COP a nesting algorithm, that checks whether a set of parts can be placed in a single bin, is needed. In this paper a sequence based heuristic is used, where sequences are modified by a 2-exchange operator as in Gomes and Oliveira (2002). The local search is controlled by a Tabu search method for more than 3 parts and an exact method for equal or less than 3 parts.

The main idea of the Tabu Search introduced by Furian and Vössner (2013) is to determine the weakest bin of a current solution and the attempt to empty this it. This is done in two stages: First, a part from the current weakest bin is selected and reallocated to any other bin, if possible not tabu, using the same nesting heuristic as in Sect. 6.1, BFNH. If no further move can be performed and there are still parts in the weakest bin the algorithm moves on to the second stage. There, all possible k surrounding bins of the weakest bin are considered. All parts in the surrounding bins and in the weakest bin are reallocated using the SSS algorithm described above. If the obtained solution uses less than $k + 1$ bins the move is performed immediately and k is decreased. If it uses $k + 1$ or more bins the move is only performed if it is not tabu, where k is only decreased in the first case. If no valid move could be performed and $k < k_{max}$ k is increased, if $k = k_{max}$ a diversification step is performed. After each move the weakest bin is recalculated.

7 Experimental results

For evaluation of RA^* we tested the algorithm on the problems proposed by Furian and Vössner (2013), available at http://mbi.tugraz.at/projekte/data/TestInstancesAnd

Table 1 Stacks and their sizes

Stack sizes	Number of stacks class II	Number of stacks class III
15	3	5
10	5	7
8	3	5
7	3	5
5	2	6
Total	16	28

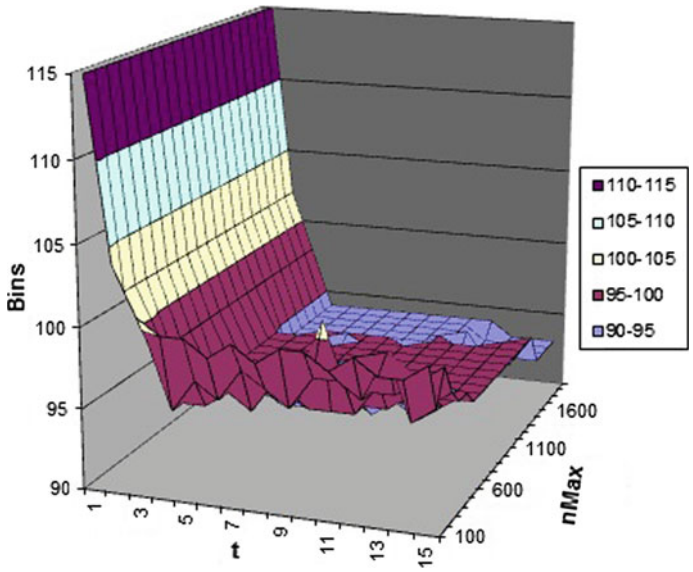

Fig. 4 Parameter tests for III × 2 instance

Results.zip, and compared the performance with the results obtained by the standard heuristics described in Furian and Vössner (2013). Test instances are divided in three classes with different problem sizes. Problems of class I, small problems, consists of 6 stacks and 50 parts, in particular one stack with 15 parts, 2 stacks with 10 parts and 3 stacks with 5 parts. Medium sized problems, class II problems, consists of 16 stacks and 150 parts and large problems, class III, of 28 stacks and 250 parts. Distribution of parts over stacks for class II and III problems can be seen in Table 1. Furthermore, $m = 3, ow = 4$ for class I instance and $m = 4, ow = 8$ for class II and III and $Q = \{A, B, C, D\}$. Table 1 describes the stack sizes and number of stacks for problems of class I and II. Bins have dimension $L = 20$ and $W = 10$. For more details on dimensions of parts distributions of different material qualities and types the reader is referred to Furian and Vössner (2013).

Albano and Sapuppo (1980) already stated small changes of t mostly do not lead to changes in results. The size of LA, given by n_{max}, has a much bigger influence on the

Table 2 Mean least square error parameter test results

t^*, n^*	100	200	300	400	500	600	700	800	900	1,000
1	1,258.77	1,220.60	1,212.90	1,228.53	1,239.47	1,246.48	1,256.94	1,255.91	1,266.29	1,280.37
2	606.12	549.15	534.64	542.89	548.52	553.35	562.99	563.95	573.43	586.14
3	781.24	713.72	694.61	697.39	698.53	700.67	707.65	707.17	714.07	724.44
4	1,114.75	1,048.15	1,029.87	1,033.27	1,034.67	1,036.85	1,043.33	1,042.34	1,048.31	1,057.85
5	1,652.20	1,586.74	1,569.36	1,573.36	1,575.16	1,577.61	1,584.18	1,583.38	1,589.02	1,598.22
6	2,216.97	2,152.64	2,136.60	2,141.53	2,144.12	2,147.20	2,154.13	2,153.89	2,159.50	2,168.54
7	2,735.97	2,672.50	2,657.63	2,663.47	2,666.72	2,670.31	2,677.59	2,677.98	2,683.83	2,692.67
8	3,190.42	3,127.68	3,113.93	3,120.64	3,124.56	3,128.66	3,136.41	3,137.55	3,143.81	3,152.61
9	3,579.70	3,518.38	3,505.93	3,513.60	3,518.40	3,523.12	3,531.45	3,533.51	3,540.25	3,549.21
10	3,901.16	3,842.19	3,830.47	3,839.20	3,844.95	3,850.30	3,859.26	3,861.78	3,868.76	3,877.92
11	4,203.78	4,146.35	4,135.77	4,145.64	4,152.16	4,158.10	4,167.53	4,170.80	4,178.14	4,187.42
12	4,476.92	4,420.90	4,410.66	4,420.72	4,427.42	4,433.30	4,442.81	4,446.82	4,454.52	4,463.64
13	4,691.12	4,638.41	4,629.42	4,639.71	4,646.35	4,652.41	4,662.45	4,667.28	4,675.38	4,684.67
14	4,881.63	4,833.18	4,824.99	4,838.54	4,845.95	4,852.06	4,863.31	4,869.53	4,878.06	4,887.36
15	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69
t^*, n^*	1,100	1,200	1,300	1,400	1,500	1,600	1,700	1,800	1,900	2,000
1	1,305.06	1,328.15	1,343.91	1,352.55	1,358.40	1,369.46	1,380.40	1,385.51	1,396.75	1,686,617.07
2	605.76	623.23	634.62	640.80	644.75	652.22	659.89	664.24	671.87	1,482,866.77
3	739.73	754.20	762.71	767.47	770.52	776.75	783.49	786.34	793.83	1,303,052.59
4	1,071.35	1,084.60	1,092.27	1,096.28	1,098.64	1,103.93	1,110.37	1,112.88	1,120.94	1,142,527.98
5	1,610.90	1,622.94	1,629.96	1,633.44	1,634.67	1,639.00	1,644.97	1,647.48	1,655.02	991,783.32
6	2,180.06	2,190.89	2,197.59	2,200.14	2,201.59	2,204.36	2,209.40	2,212.23	2,219.01	850,518.69
7	2,703.25	2,713.25	2,719.14	2,721.03	2,722.52	2,724.74	2,728.31	2,730.26	2,735.48	718,724.55

Table 2 continued

t^*, n^*	1,100	1,200	1,300	1,400	1,500	1,600	1,700	1,800	1,900	2,000
8	3,162.64	3,171.96	3,177.61	3,179.39	3,180.91	3,183.08	3,185.50	3,187.99	3,193.08	596,403.02
9	3,559.18	3,568.15	3,573.74	3,575.44	3,576.95	3,579.04	3,581.12	3,583.25	3,589.21	483,629.81
10	3,888.27	3,897.34	3,902.93	3,904.27	3,905.43	3,907.17	3,908.60	3,910.62	3,914.88	380,603.60
11	4,197.92	4,206.83	4,211.96	4,213.22	4,213.95	4,215.40	4,216.28	4,217.19	4,220.02	286,635.06
12	4,473.94	4,482.73	4,487.26	4,488.26	4,488.86	4,489.78	4,490.79	4,491.52	4,494.89	201,967.69
13	4,695.54	4,704.58	4,709.30	4,710.25	4,710.56	4,711.37	4,712.11	4,712.68	4,716.20	127,029.00
14	4,898.71	4,907.04	4,911.84	4,912.96	4,913.05	4,913.40	4,913.51	4,914.36	4,915.91	61,313.51
15	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69	5,083.69

Table 3 Experimental results for RA^*

	Bounds					$RA*$					
						n^1_{max}			n^2_{max}		
	LB_C	LB^R_{DA}	L^R_{BM}	L^R_{CJE}	Opt.	Bins	Nodes	Time	Bins	Nodes	Time
I × 1	15	15	18	18	27	27	8,119	1.9	27	8,119	1.9
I × 2	12	11	14	15	21	21	7,379	1.6	21	7,379	1.6
I × 3	15	16	20	18	28	28	7,064	1.3	28	7,064	1.3
I × 4	16	16	19	18	25	25	8,436	1.5	25	8,436	1.5
I × 5	14	14	15	17	21	21	8,680	2.5	21	8,680	2.5
I × 6	14	14	15	16	25	25	13,607	4.3	25	13,607	4.3
I × 7	13	13	16	17	24	24	9,957	2.6	24	9,957	2.6
I × 8	13	13	14	16	21	21	9,554	4.5	21	9,954	4.5
I × 9	15	14	17	15	26	26	12,234	3.1	26	12,234	3.1
I × 10	15	14	16	16	25	25	6,850	1.0	25	6,850	1.0
II × 1	37	37	42	40	*	57	41,680	48.2	56	85,245	202.9
II × 2	40	40	45	43	*	60	42,727	43.9	61	97,768	202.0
II × 3	43	43	48	44	*	59	51,408	66.3	59	97,865	244.0
II × 4	39	39	42	42	*	60	46,882	51.8	60	92,765	196.8
II × 5	39	39	44	42	*	56	39,529	44.0	56	76,600	160.6
II × 6	42	41	46	42	*	61	62,045	73.9	61	1,11,920	241.0
II × 7	42	43	45	44	*	58	47,541	63.5	58	94,729	229.5
II × 8	42	42	46	43	*	59	44,977	46.5	59	96,833	197.1
II × 9	44	46	49	47	*	64	53,598	65.8	63	1,01,273	252.1
II × 10	38	37	40	38	*	59	59,964	70.1	58	1,12,178	263.9
III × 1	68	68	74	70	*	98	51,055	50.8	98	1,01,473	180.6
III × 2	64	65	69	67	*	96	55,554	55.5	95	1,04,153	189.8
III × 3	67	66	75	68	*	104	61,789	63.3	101	1,13,452	198.1
III × 4	64	64	71	68	*	93	51,063	56.6	92	92,262	167.7
III × 5	69	68	74	70	*	99	52,907	60.3	101	1,13,817	230.5
III × 6	68	66	75	71	*	99	57,745	67.1	97	1,07,223	239.8
III × 7	67	65	73	68	*	98	51,721	56.4	101	1,17,827	231.1
III × 8	67	67	72	70	*	101	59,877	61.6	97	1,04,818	186.7
III × 9	70	70	76	71	*	96	50,151	52.1	97	1,09,878	241.9
III × 10	67	67	73	69	*	99	53,128	58.7	96	1,78,301	178.3

quality of results. To verify this for COP we tested all possible parameter combinations of $t \in \{1, \ldots, 15\}$ and $n_{max} \in \{100, 200, \ldots, 2,000\}$. Figure 4 shows the typical behaviour of the solution quality as a function of (t, n_{max}). For small values of both parameters significant improvement can be achieved by increasing them. However, for larger values of t or n_{max} no correlation between parameter size and the number of used bins can be observed. To proof this in a more statistical way we fitted the data

Table 4 Standard heuristics

Class × instance	SA		GA		SSS		Tabu		RA^*	
	Bins	Time	Bins	Time	Bins	Time	Bins	Time	Bins	Time
I × 1	27 (27.2)	7.5	27 (27.3)	8.3	30	<0.1	30	12.2	27	1.9
I × 2	21 (21.9)	6.7	22 (22.9)	7.2	28	<0.1	24	23.9	21	1.6
I × 3	28 (28.6)	6.8	29 (30.1)	7.5	34	<0.1	32	13.4	28	1.3
I × 4	26 (26.1)	6.6	26 (26)	7.5	30	<0.1	28	13.9	25	1.5
I × 5	23 (23)	7.1	22 (23.5)	8.0	27	<0.1	26	10.3	21	2.5
I × 6	25 (25.9)	6.6	26 (26.9)	7.5	29	<0.1	27	11.4	25	4.3
I × 7	24 (24.1)	6.6	24 (25.2)	7.7	26	<0.1	26	19.9	24	2.6
I × 8	21 (21.8)	7.3	22 (23)	8.1	28	<0.1	33	7.2	21	4.5
I × 9	26 (26.1)	7.1	26 (26.1)	8.1	33	<0.1	28	34.0	26	3.1
I × 10	25 (25.2)	7.4	26 (26.2)	7.7	30	<0.1	29	6.9	25	1.0
II × 1	61 (62.6)	24.0	62 (65.4)	139.3	65	0.3	67	102.2	57	48.2
II × 2	66 (67.2)	25.2	67 (69.6)	139.9	72	0.2	90	85.7	60	43.9
II × 3	64 (65.7)	24.1	66 (67.2)	138.3	71	0.2	72	69.7	59	66.3
II × 4	64 (66.5)	23.2	66 (68.4)	138.5	77	0.2	81	180.0	60	51.8
II × 5	63 (64.3)	24.1	62 (64.9)	153.2	72	0.3	64	180.0	56	44.0
II × 6	63 (67.6)	23.9	65 (69.6)	140.5	70	0.1	77	43.7	61	73.9
II × 7	63 (64.7)	24.5	62 (65.7)	145.1	69	0.2	78	43.8	58	63.5
II × 8	64 (66.9)	24.7	67 (68)	135.8	76	0.1	82	47.7	59	46.5
II × 9	66 (67.6)	23.8	69 (69.7)	137.4	78	0.2	107	46.9	64	65.8
II × 10	62 (63.5)	24.7	64 (67.2)	141.6	66	0.1	71	40.8	59	70.1
III × 1	109 (112.1)	38.4	114 (120.4)	180.0	123	0.4	143	180.0	98	50.8
III × 2	105 (106.4)	38.3	108 (112.3)	180.0	112	0.3	117	180.0	96	55.5
III × 3	111 (114.5)	40.0	119 (122.9)	180.0	120	0.2	134	180.0	104	63.3
III × 4	103 (106.8)	39.4	108 (111.4)	180.0	112	0.5	118	180.0	93	56.6
III × 5	108 (111.6)	37.8	115 (117.5)	180.02	113	0.3	185	180.0	99	60.3
III × 6	110 (112.7)	38.1	116 (118.7)	180.0	116	0.3	151	180.0	99	67.1
III × 7	109 (113)	38.5	116 (119.8)	180.0	123	0.6	146	180.0	98	56.4
III × 8	108 (110.9)	37.9	112 (116)	180.0	117	0.4	171	180.0	101	61.6
III × 9	104 (117.3)	39.4	111 (114.4)	180.0	123	0.4	136	180.0	96	52.1
III × 10	105 (108.7)	39.8	112 (114)	180.0	112	0.4	127	180.0	99	58.7

to the following log-linear-model where $b(t, n_{max})$ denotes the number used bins, t^* and n^* the critical values for t and n_{max}:

$$\log\left(b\left(t, n_{max}\right)\right) = \begin{cases} \beta_0 + \beta_1 \cdot t + \beta_2 \cdot n & \text{for } t \le t^* \wedge \frac{n_{max}}{100} \ge t, \\ \beta_3 + \beta_4 \cdot t + \beta_5 \cdot n & \text{for } n_{max} < n^* \wedge \frac{n_{max}}{100} < t, \\ \alpha \end{cases}$$

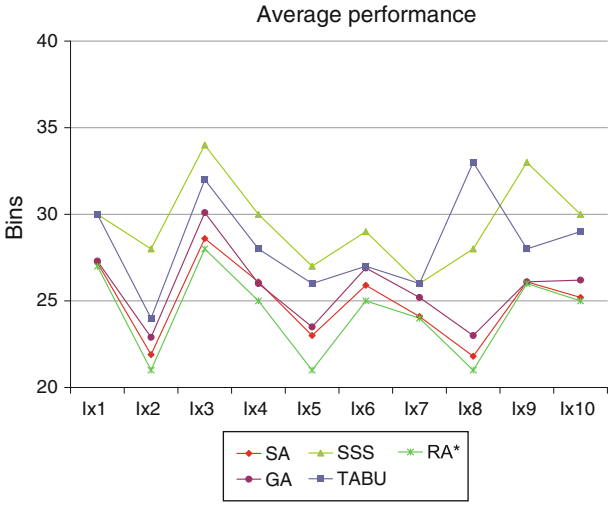

Fig. 5 Comparisons of average performances for class I instances

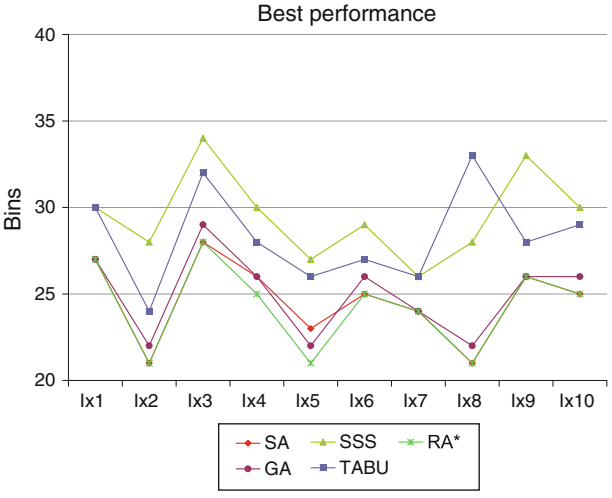

Fig. 6 Comparisons of best performances for class I instances

For parameters smaller or equal than the critical values the results are fitted to two different log-linear models and so an exponential improvement of solution quality is assumed. For parameter settings greater than the critical values the results are fitted to a constant. For all combinations of $t^* \in \{1, \ldots, 15\}$ and $n^* \in \{100, 200, \ldots, 2,000\}$ the mean of all least-square errors over class III instances are reported in Table 2. The smallest least-square-error, found for $t = 2$ and $n_{max} = 300$, represents the model that fits the data best and shows that for greater parameter values no statistical significant change of solution quality can be observed. All results of the parameter

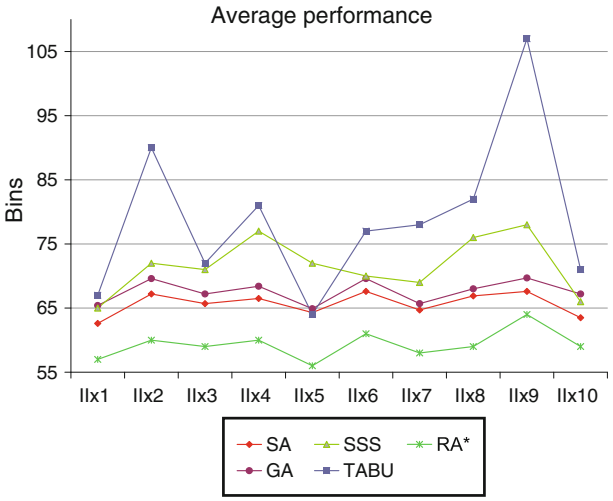

Fig. 7 Comparisons of average performances for class II instances

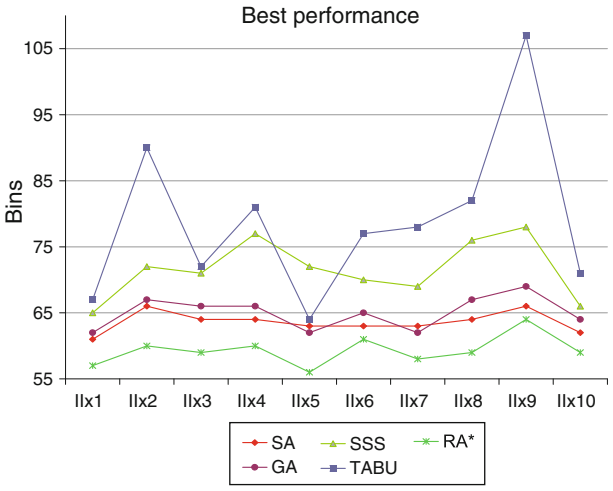

Fig. 8 Comparisons of best performances for class II instances

testing can be found at http://mbi.tugraz.at/projekte/data/TestInstancesAndResults.zip.

To compare results to standard heuristics we fully report results for $t = 15$ and two different scenarios for n_{max}, both depending on the size of an instance:

$$n^1_{max} = \begin{cases} 10{,}000 & \text{for } n = 50, \\ 800, & \text{for } n = 150, \\ 500, & \text{for } n = 250 \end{cases}$$

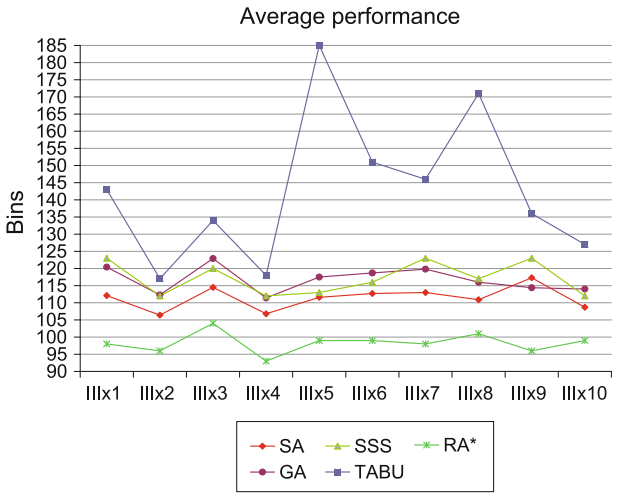

Fig. 9 Comparisons of average performances for class III instances

and

$$n^2_{max} = \begin{cases} 20{,}000 & \text{for } n = 50, \\ 1{,}600, & \text{for } n = 150, \\ 1{,}000, & \text{for } n = 250 \end{cases}$$

Furthermore, results for RA^* were obtained using the following estimator est:

$$est = (1.2 - 10\text{Var}\left(\frac{|\text{P}_{\text{Q}_\text{A}}|}{|\text{P}|}, \ldots, \frac{|\text{P}_{\text{Q}_\text{D}}|}{|\text{P}|}\right)\text{L}_\text{C}. \tag{32}$$

This estimator is based on the observation that lower bounds tend to be weaker for instances where the cardinalities of subsets of parts with same material qualities do not vary that much.

Table 3 shows lower bounds and results of all tested instances. Bounds were calculated using L_{COP} and different bounds for bin packing problems. Furthermore, Opt. denotes the optimal solution if it could be found within a time limit of 1,200 s.

One can see that all lower bounds are quite weak due to the fact that they do not consider order- and stack constraints. This is also a reason why optimal solutions could not found for problems of class II and III. Comparing the results obtained under n^1_{max} with results under n^2_{max} one may note that larger n_{max} does not necessarily lead to better solutions. However, runtimes are more than three times higher than for n^1_{max}.

Table 4 shows the performance of standard heuristics proposed by Furian and Vössner (2013) on the same instances. Although a time limit of 180 s was applied to these algorithms we can see that using n^1_{max} all runs of RA^* terminated within this limit and therefore, we are able to compare results. To investigate their struc-

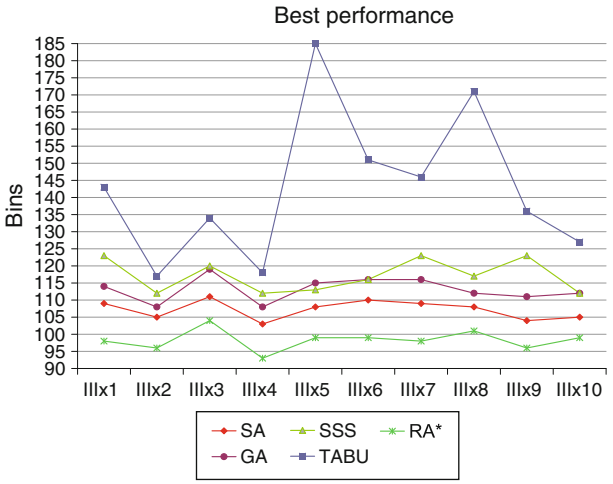

Fig. 10 Comparisons of best performances for class III instances

ture, solutions for class III instances are reproted at http://mbi.tugraz.at/projekte/data/TestInstancesAndResults.zip to allow future researcher.

Figures 5, 6, 7, 8, 9 and 10 compare the results of our algorithm with the results from Furian and Vössner (2013).

One can see that RA^* outperforms all other heuristics significantly, as all other algorithms are not able to profit from the structures of COP as RA^* does.

References

Albano A, Sapuppo G (1980) Optimal allocation of two-dimensional irregualr shapes using heuristic search methods. IEEE Trans Syst Man Cybern 10(5):242–248

Augustine J, Banerjee S, Irani S (2006) Strip packing with precedence constraints and strip packing with release times. In: Proceedings of the eighteenth annual ACM symposium on parallelism in algorithms and architectures, SPAA '06, ACM, New York, pp 180–189

Babu AR, Babu NR (1999) Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. Int J Prod Res 37(7):1625–1643

Babu AR, Babu NR (2001) A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. Comput Aided Des 33:879–891

Boschetti A, Mingozzi A (2003a) The two-dimensional finite bin packing problem. Part i: new lower bounds for the oriented case. 4OR 1:27–42

Boschetti A, Mingozzi A (2003b) The two-dimensional finite bin packing problem. Part ii: new lower and upper bounds. 4OR 1:135–147

Caprara A, Pferschy U (2004) Worst-case analysis of the subset sum algorithm for bin packing. Oper Res Lett 32:159–166

Caprara A, Pferschy U (2005) Modified subset sum heuristics for bin packing. Inf Process Lett 96:18–23

Carlier J, Clautiaux F, Moukrin A (2007) New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. Comput Oper Res 33:2223–2250

Clautiaux F, Jouglet A (2007) A new lower bound for the non-oriented two-dimensional bin packing problem. Oper Res Lett 35:365–373

Dell'Amico M, Martello S, Vigo D (2002) A lower bound for the non-oriented two dimensional bin packing problem. Discret Appl Math 118:13–24

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1:269–271

Epstein L, Levin A (2007) Approximation and online algorithms, vol 4368, chapter on bin packing with conflicts. Springer, Heidelberg
Epstein L, Levin A, van Stee R (2008) Two-dimensional packing with conflicts. Acta Inf 45(3):155–175
Fekete SP, Schepers J (2000) On more-dimensional packing ii: bounds. Technical report ZPR97-289, Mathematisches Institut, Universitt zu Kln
Fekete SP, Schepers J (2004) A general framework for bounds for higher-dimensional orthogonal packing problems. Math Methods Oper Res 60:311–329
Fu L, Sun D, Rilett LR (2006) Heuristic stortest path algorithms for transportation applications: state of the art. Comput Oper Res 33:3324–3343
Furian N, Vössner S (2013) Constrained order packing–comparison of heuristic approaches for a new bin packing problem. Cent Eur J Oper Res 21:237–265
Gomes AM, Oliveira JF (2002) A 2-exchange heuristic for nesting problems. Eur J Oper Res 141(2): 359–370
Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heurisitic determination of minimum cost paths. IEEE Trans Sys Sci Cybern 4(2):100–107
Iima H, Yakawa T (2003) A new design of genetic algorithm for bin packing. Congr Evolut Comput 2:1044–1049
Iori M, Martello S (2010) Routing problems with loading constraints. TOP 18:4–27
Johnson DS (1973) Near optimal bin packing algorithms. Dissertation, Massachussetts Institute of Technology, Cambridge
Johnston RE, Sadinlija E (2004) A new model for complete solutions to one-dimensional cutting stock problems. Eur J Oper Res 153(1):176–183
Khanafer A, Clautiaux F, Talbi E-G (2012) Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. Comput, Oper Res 39(1):54–63
Lodi A, Martello S, Vigo D (1999a) Approximation algorithms for the oriented two-dimensional bin packing problem. Eur J Oper Res 112:158–166
Lodi A, Martello S, Vigo D (1999b) Heuristic and metaheuristic approaches for a class of two-dinemsional bin packing problems. INFORMS J Comput 11:345–357
Lodi A, Martello S, Vigo D (2004) Tspack: a unified tabu search code for multi-dimensional bin packing problems. Ann Oper Res 131:203–213
Mack D, Bortfeldt A (2012) A heuristic for solving large bin packing problems in two and three dimensions. Cent Eur J Oper Res 20(2):337–354
Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. Manag Sci 44(3)
Reinertsen H, Vossen TWM (2010) The one-dimensional cutting stock problem with due dates. Eur J Oper Res 201:701–711
Rinaldi F, Franz A (2007) A two-dimensional strip cutting problem with sequencing constraint. Eur J Oper Res 183:1371–1384
Wäscher G, Haußner H, Schumann H (2007) An improved typology of cutting and packing problems. Eur J Oper Res 183:1109–1130
Wu TH, Chen JF, Low C, Tang PT (2003) Nesting of two-dimensional parts in multiple plates using hybrid algorithm. Int J Prod Res 41(16):3883–3900