Guided local search for the three-dimensional bin packing problem*

Oluf Faroe, David Pisinger, Martin Zachariasen†

December 1999

Abstract

The three-dimensional bin packing problem is the problem of orthogonally packing a set of boxes into a minimum number of three-dimensional bins. In this paper we present a heuristic algorithm based on Guided Local Search (GLS). Starting with an upper bound on the number of bins obtained by a greedy heuristic, the presented algorithm iteratively decreases the number of bins, each time searching for a feasible packing of the boxes using GLS. The process terminates when a given time limit has been reached or the upper bound matches a precomputed lower bound. The algorithm can also be applied to two-dimensional bin packing problems by having a constant depth for all boxes and bins. Computational experiments are reported for two- and three-dimensional instances with up to 200 boxes, and the results are compared with those obtained by heuristics and exact methods from the literature.

1 Introduction

The *three-dimensional bin packing problem* asks for an orthogonal packing of a set of boxes into a minimum number of three-dimensional bins. The only restriction imposed on the solution is that the boxes have fixed orientation. The problem has several industrial applications in cutting and loading contexts, and since rotation of the boxes is not allowed, the model can be used for solving several scheduling problems.

In this paper we present a new local search heuristic based on the *Guided Local Search (GLS)* method [20, 23] which has its origin in constraint satisfaction applications, but also has proven to be a very powerful metaheuristic for solving hard combinatorial problems. GLS uses memory to guide the search to promising regions of the solution space; this is done by augmenting the cost function with a penalty term that penalizes "bad" features of previously visited solutions.

Starting with an upper bound on the number of bins obtained by a greedy heuristic, the GLS algorithm iteratively tightens the upper bound by removing one bin from a feasible solution. That is, for a given number of bins, GLS constructs a feasible packing of the boxes. When such a packing has been found, the number of bins is decreased by one, and the procedure iterates until a given time limit has been reached or the upper bound matches the lower bound — in our case the L_2 bound from Martello, Pisinger and Vigo [16].

*Tech. Rep. 99/13, DIKU, University of Copenhagen, Denmark

†Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark. E-mail: {oluf,pisinger,martinz}@diku.dk

The GLS algorithm assigns coordinates (including bin numbers) to the boxes. Translation of boxes along coordinate axes within one bin or moving boxes from one bin to another defines the neighbourhood of the local search algorithm. The objective value of a given solution is the total volume of the pairwise overlap between boxes. Searching for a feasible solution is therefore equivalent to minimizing the objective function since an objective value of zero indicates a feasible packing.

To speed up the local search an implementation of *Fast Local Search (FLS)* has been used [20, 23]. Since the neighbourhood is fairly large in the current setting, FLS drastically speeds up the search for a local minimum. This is done by shadowing less promising parts of the neighbourhood, that is, temporarily fixing box positions.

The GLS algorithm has been evaluated on instances with up to 200 boxes of varying types showing very promising results compared to previous approaches: Within a fixed time limit, it generally finds better solutions than the exact algorithm by Martello, Pisinger and Vigo [16], and this also applies when comparing to other heuristic algorithms as Lodi, Martello and Vigo [14, 15]. Furthermore, it is easily generalized to problems in which rotations are allowed and/or when the items to be packed have irregular shape.

The present paper is organized as follows: In Section 2 we define the considered problem more formally, and describe some lower bounds. Section 3 gives an overview of previous heuristics based on local search. The concept of Guided Local Search and our new algorithm is presented in Section 4. The paper is concluded with extensive empirical results in Section 5.

2 The three-dimensional bin-packing problem

Cutting and packing problems have numerous applications, spanning from the direct use of the models (loading cargo into ships, vehicles, containers) to a more abstract use of the models (scheduling problems, budgeting, generation of valid inequalities). Due to the large number of applications, different variants of the problems have been developed based on the additional constraints present in the concrete application. Nearly all the problems considered in the literature are $\mathcal{NP}$-hard, and thus difficult to solve to optimality. This makes it attractive to consider heuristic approaches, as heuristics often are able to return a "sufficiently" good solution in reasonable time, and generally are flexible to handle additional side constraints.

In the present paper we consider the *three-dimensional bin-packing problem* (3D-BPP) in which we have a set J of n rectangular-shaped *boxes*, each having a width w_j, height h_j and depth d_j (all integers), and an unlimited number of identical three-dimensional *bins* of width W, height H and depth D. The objective is to pack all the boxes into the minimum number of bins, such that the original orientation is respected. The *two-dimensional bin-packing problem* (2D-BPP) is the obvious restriction of 3D-BPP to two dimensions (width and height).

Other variants of packing and loading include the *knapsack container loading problem* where the boxes have an associated profit, and the objective is to pack a subset of the boxes into a single bin of fixed dimensions such that the profit of the included boxes is as large as possible. Other applications consider the *minimum depth container loading problem* (also known as the *strip packing problem*) where the objective is to pack all the boxes into a single container with fixed width W and height H but variable depth D, which has to be minimized. There may be additional constraints to these models with respect to *rotation* of the boxes, and to the packing pattern. In a *guillotine packing* the boxes should be organized such that one can separate them through a number of guillotine cuts (i.e. cuts through the

whole subject). For recent surveys on cutting and packing problems see Dyckhoff [11] and Dyckhoff, Scheithauer and Terno [12]. Using the typology of Dyckhoff [11], our problem may be classified as `3/V/I/M` with the additional information that the boxes have fixed orientation and there is no restriction on the packing pattern.

Martello, Pisinger and Vigo [16] proposed the first exact algorithm for solving the oriented three-dimensional bin packing problem. The algorithm is based on branch-and-bound and thus relies on tight lower bounds on the objective value. An obvious lower bound for 3D-BPP comes from continuous relaxation, where it is assumed that any fraction of a box may be packed into the bin. Thus the *continuous lower bound* L_0 is given by

$$L_0 = \left\lceil \frac{\sum_{j=1}^{n} w_j h_j d_j}{WHD} \right\rceil \tag{1}$$

The bound L_0 can be computed in $O(n)$ time, and its worst-case performance ratio is $\frac{1}{8}$ [16]. In practice, L_0 is not a very tight bound.

Another lower bound can be derived by reduction to the one-dimensional case. Thus we consider those boxes which do not fit besides or above each other

$$A = \{j \in J : w_j > W/2 \text{ and } h_j > H/2\} \tag{2}$$

For a given integer p with $0 < p \leq D/2$ we define the following three sets to contain those boxes which have a specific depth d_j:

$$\begin{array}{lcl} J_d(p) & = & \{j \in A : d_j > D/2\} \\ J_\ell(p) & = & \{j \in A : D - p \geq d_j > D/2\} \\ J_s(p) & = & \{j \in A : D/2 \geq d_j \geq p\} \end{array} \tag{3}$$

Let

$$L_1'(p) = \max\left\{ \left\lceil \frac{\sum_{j \in J_s(p)} d_j - |J_\ell(p)|D + \sum_{j \in J_\ell(p)} d_j}{D} \right\rceil, \left\lceil \frac{|J_s(p)| - \sum_{j \in J_\ell(p)} \lfloor (D - d_j)/p \rfloor}{\lfloor D/p \rfloor} \right\rceil \right\}$$

then a lower bound on 3D-BPP is given by

$$L_1' = |J_d(p)| + \max_{1 \leq p \leq D/2} L_1'(p) \tag{4}$$

Let L_1'' be the bound (4) obtained by first rotating all boxes and bins 90° around the horizontal axis and let L_1''' be the same bound obtained by first rotating all boxes and bins 90° around the vertical axis. In this way, we get the tighter lower bound [16]

$$L_1 = \max\{L_1', L_1'', L_1'''\} \tag{5}$$

which can be computed in $O(n^2)$ time [17, 9].

A bound which explicitly takes all three dimensions into account is defined as follows. For any pair of integers (p, q), with $0 < p \leq W/2$ and $0 < q \leq H/2$ we split the boxes into two classes

$$\begin{array}{lcl} K_\ell(p,q) & = & \{j \in J : w_j > W - p \text{ and } h_j > H - q\} \\ K_s(p,q) & = & \{j \in J \setminus K_\ell(p,q) : w_j \geq p \text{ and } h_j \geq q\} \end{array} \tag{6}$$

where the very small boxes with $w_j < p$ and $h_j < q$ are left out of the problem. Let

$$L_2'(p,q) = L_1' + \max\left\{0, \left\lceil \frac{\sum_{j\in K_s(p,q)} w_j h_j d_j - WHDL_1' + WH\sum_{j\in K_\ell(p,q)} d_j}{WHD} \right\rceil \right\}$$

which leads to the valid lower bound:

$$L_2' = \max_{1\le p\le W/2;\ 1\le q\le H/2} L_2'(p,q) \tag{7}$$

If we again let L_2'' be the bound (7) obtained by first rotating all boxes and bins 90° around the horizontal axis, and L_2''' be the same bound obtained by first rotating all boxes and bins 90° around the vertical axis, we get the tighter lower bound [16]

$$L_2 = \max\{L_2', L_2'', L_2'''\} \tag{8}$$

which can be computed in $O(n^2)$ time. In [16] it is shown that the bounds L_0 and L_1 do not dominate each other, but L_2 dominates both L_0 and L_1. We will thus use L_2 as lower bound in all our experiments.

3 Local search heuristics

Heuristics for 3D-BPP may be divided into *construction* and *local search* heuristics. Simple construction heuristics add one box at a time to an existing partial packing until all boxes are packed. The boxes are often pre-sorted by one of their dimensions and added using a particular strategy, e.g., variants of *first* fit or *best* fit strategies.

Local search heuristics *iteratively* try to find a better packing of the boxes. New solutions are generated by defining a neighbourhood function $\mathcal{N}$ over the set of solutions $\mathcal{X}$. In the current context, the set of solutions is all possible packings of the boxes into bins; these packings need not be feasible, as will be shown later. The neighbourhood function assigns to every solution $\mathbf{x} \in \mathcal{X}$ a set $\mathcal{N}(\mathbf{x}) \subseteq \mathcal{X}$ of solutions that are in the "vicinity" of $\mathbf{x}$, e.g., solutions that can be obtained by moving a box to another location within its bin or to a new bin.

Given an initial solution $\mathbf{x}_0$, local search visits a sequence of solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$ such that $\mathbf{x}_i \in \mathcal{N}(\mathbf{x}_{i-1})$ for every $i = 1, 2, \ldots, k$. Solutions are compared using the objective function f, which assigns a value $f(\mathbf{x})$ to every solution $\mathbf{x} \in \mathcal{X}$. We assume that we would like to minimize f, since our goal is to minimize the number of bins needed to pack the boxes. However, it should be pointed out that f need not be directly related to the number of bins used; other quality measures may be incorporated.

When the series of solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k$ fulfils $f(\mathbf{x}_0) > f(\mathbf{x}_1) > \ldots > f(\mathbf{x}_k)$ the local search algorithm is denoted *local optimization* (also called iterative improvement or hill-climbing). Local optimization stops when the current solution $\mathbf{x}_k$ a local minimum, that is, when $\mathcal{N}(\mathbf{x}_k)$ contains no solution better than $\mathbf{x}_k$. Applying local optimization to a solution using the objective function f will be denoted by the operator $\textsc{LocalOpt}_f$. In the above case we have $\mathbf{x}_k = \textsc{LocalOpt}_f(\mathbf{x}_0)$.

All the classical metaheuristics simulated annealing, tabu search and genetic algorithms have been applied to 2D- and 3D-BPP (see [1] for a general description of these methods). In the following we describe some of the most successful local search algorithms from the

literature. It should be noted that we have also included results on the related strip packing problem. Furthermore, some of the algorithms *only* allow guillotine packings and/or *do* allow 90° rotations of the boxes.

The first method that may be characterized as a local search is the 2D-BPP heuristic given by Bengtson [4]. A construction heuristic for the strip packing problem which uses (partial) backtracking forms the basis for the overall algorithm. Starting with a packing of a subset of the pieces (∼boxes in 3D), the remaining pieces are iteratively packed into the bin having maximum unused space. The method uses an approach in which bins are either "active" or "inactive"; the latter are bins that appear to be difficult to pack any further. When all bins become inactive, the search stops. Experiments showed that the algorithm was fast and produced packings with a high utilization.

Dowsland [10] presented a simulated annealing algorithm for the strip packing problem in 2D. The algorithm tries to pack the pieces into one containing rectangle. When a feasible packing has been found, the height of the containing rectangle is reduced and a new feasible packing is sought for. The problem of finding a feasible solution for a given height is solved by minimizing the overlap between the pieces (the objective function is the pairwise sum of the overlap). Neighbouring solutions are generated by moving any single piece to any other position. Only few experiments on small instances (10-14 pieces) were made.

A tabu search algorithm for 2D-BPP was given by Lodi, Martello and Vigo [14]. This algorithm uses two simple construction heuristics for doing the actual packing of pieces into bins. The tabu search algorithm only controls the movement of pieces between bins. More precisely, two neighbourhood functions were considered. Both try to relocate a piece from the *weakest* bin to another bin (the weakest bin is essentially the bin that appears to be easiest to empty). The first neighbourhood function simply tries to pack the piece directly into another bin, while the second tries to recombine two bins such that one of them can hold the piece being moved. Since the heuristics used for packing the pieces produce guillotine packings, so does the overall algorithm. In [15] this tabu search approach was generalized to other variants of 2D-BPP, including the one considered in this paper, that is, without the guillotine constraint. The experimental results obtained by these tabu search algorithms will be discussed in Section 5.

An implicit solution representation was also used by Corcoran and Wainwright [8] who presented a genetic algorithm for strip packing in 3D. Solutions were represented by a permutation of boxes. Given such a permutation, a packing was constructed using a first or next fit packing heuristic which processed the boxes in the order given by the permutation. The genetic algorithm maintained a population of permutations on which standard crossover and mutation operators were applied. Fairly large instances were considered, but the algorithm was not compared to other algorithms from the literature.

Another genetic algorithm, also based on an implicit representation, was presented by Kröger [13]. He considered the strip packing problem in 2D, constrained to guillotine packings but allowing 90° rotations of boxes. The approach used the fact that any guillotine packing can encoded using a slicing tree structure from which a packing can be constructed in linear time. Fairly complex crossover and mutation operators were devised for the slicing tree structure. Also, all solutions were locally optimized using a variant of the mutation neighbourhood (corresponds to using the LocalOpt$_f$ operator on each generated offspring). Experiments on medium sized problems showed that the genetic algorithm produced better solutions than simpler alternatives such as simulated annealing.

4 A new heuristic for 3D-BPP

Based on the metaheuristic *Guided Local Search (GLS)* we present a new algorithm for producing good solutions to 3D-BPP. The section is organized as follows: First we give a short description of GLS, and then an overview of the general approach taken to solve the problem. Finally there is a detailed description of the application of GLS.

4.1 GLS

GLS is a new metaheuristic that has proven to be effective on a wide range of hard combinatorial optimization problems. The heuristic was developed by Voudouris and Tsang [20, 23] — originally for solving constraint satisfaction problems. The heuristic may be classified as a tabu search heuristic; it uses memory to control the search in a manner similar to tabu search. However, the definition is simpler and more compact.

GLS is based on the concept of *features*, i.e., a set of attributes which characterizes a solution to the problem in a natural way. We assume that any solution can be described using a set of M features: A solution $\mathbf{x} \in \mathcal{X}$ either has or does not have a particular feature $i \in \{1, \ldots, M\}$; the indicator function $I_i(\mathbf{x})$ is 1 if $\mathbf{x}$ has feature i and 0 otherwise.

The features should be defined such that the presence of a feature in a solution has a more or less direct contribution to the value of the solution. This (direct or indirect) contribution is the *cost* c_i of the feature. A feature with a high cost is not attractive and may be *penalized*. The number of times a feature has been penalized is denoted by p_i (initially zero).

Penalties are incorporated into the search by constructing an augmented objective function

$$h(\mathbf{x}) \;=\; f(\mathbf{x}) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(\mathbf{x})$$

and performing local search on this function instead of the original objective function. The parameter λ should balance the original objective function to the contribution from the penalty term. This is the only parameter in GLS that must be determined experimentally.

The main GLS algorithm performs a number of local optimization steps, each transforming a solution $\mathbf{x}$ into a local minimum $\mathbf{x}_* = \textsc{LocalOpt}_h(\mathbf{x})$; note that since all penalties initially are zero, the first local optimization actually finds a local optimum with respect to f. Every time a local minimum has been reached, one or more features are penalized by incrementing their p_i value by one. Those features that have maximum *utility*

$$\mu_i(\mathbf{x}_*) \;=\; \frac{c_i}{1 + p_i} \cdot I_i(\mathbf{x}_*)$$

are penalized. Loosely speaking, these are the features with maximum cost in $\mathbf{x}_*$ that have not been penalized too often in the past. After having penalized these features, the local optimization continues from $\mathbf{x}_*$ (now with respect to the modified h function).

4.2 General approach

One of the key obstacles in applying metaheuristics like GLS to 3D-BPP is the representation of a solution space and a corresponding neighbourhood function which permits a natural traversal between all feasible solutions. The reason is that even the construction of feasible solutions which are better that the solutions obtained by polynomial heuristics is a difficult

task. We note that the crucial constraint that makes it hard to construct good feasible solutions is that there must be no overlap between boxes in the same bin. Therefore, we have chosen to relax this constraint such that we get a solution space for which the only constraint imposed is that the boxes must be placed within the walls of a bin.

Empirical results from [16] show that the typical span we can expect between upper bounds (ub) obtained by heuristics and lower bounds (lb) like the L_2-bound is likely to be small compared to the optimal solution. Since the 3D-BPP asks for a packing of the boxes into a minimum number of bins, the function we want to minimize will only map to a few discrete values within the set $\{lb, \ldots, ub\}$.

We therefore apply an algorithm that can roughly be split into two separate phases. The initial phase starts by computing an upper bound and a lower bound while the second phase tightens the upper bound solution using a GLS-based heuristic. If it in the first phase turns out that the lower bound equals the upper bound we have found an optimal solution and are done. Otherwise, we move to the second phase and use a heuristic to improve the upper bound by iteratively removing one bin from a feasible solution. The improving upper bound then reflects the improving current solution for the 3D-BPP. A similar approach — but based on simulated annealing — was used by Dowsland [10] in the context of two-dimensional strip packing of rectangles into a larger containing rectangle; the initial height of the containing rectangle was set to an upper bound value and reduced each time a feasible packing had been found.

4.3 Application of GLS

The problem handed to the GLS heuristic is therefore to construct a feasible packing of the boxes into a fixed number of bins. When such a packing has been found the number of bins is decreased by one and GLS is restarted with the remaining bins. This process continues until a given time limit is exceeded or the upper bound matches the lower bound.

Since the core of GLS is a local search algorithm, we will first develop a local search framework for the 3D-BPP. Based on this description a GLS heuristic will be implemented by adding solution features and penalties. The presentation is concluded with a description of the Fast Local Search technique which drastically speeds up the local search.

Local search for 3D-BPP

Let $J = \{1, \ldots, n\}$ denote the set of boxes, m the number of bins, and let x_j, y_j, z_j denote the coordinates of the left-bottom-back corner of box $j \in J$ with respect to the left-bottom-back corner of bin s_j. A solution space $\mathcal{X}$ can now be defined as the set of all possible positions of boxes $j \in J$ such that

$$\begin{array}{rcl} x_j & \in & \{0, \ldots, W - w_j\} \\ y_j & \in & \{0, \ldots, H - h_j\} \\ z_j & \in & \{0, \ldots, D - d_j\} \\ s_j & \in & \{1, \ldots, m\} \end{array} \tag{9}$$

Given a solution $\mathbf{x} \in \mathcal{X}$ we define the neighbourhood $\mathcal{N}(\mathbf{x})$ as all solutions that can be obtained by translating any single box along one of the coordinate axes or to the same position in another bin. The set $\mathcal{N}(\mathbf{x})$ is therefore constructed by assigning a new value to one of the variables x_j, y_j, z_j or s_j for any single box $j \in J$. It is clear that this definition of a solution

space includes all feasible packings and that there is a path of moves between every pair of solutions in $\mathcal{X}$.

The initial solution with m bins is generated by moving the boxes in bin $m+1$ to random positions in the bins 1 to m. If it is the first call to GLS for a given problem instance, i.e. $m = ub - 1$, the solution with $m + 1$ bins has been constructed by the upper bound heuristic, otherwise the previous call to GLS found a feasible packing with $m + 1$ bins. By not generating the initial solution purely on a random basis some of the information from previous solutions is preserved. Of course, a drawback to this approach is that the structure of a previous solution can confine GLS to an area of the solution space that can be difficult to escape.

The objective value of a given solution is the total volume of the pairwise overlap between boxes. Searching for a feasible solution is therefore equivalent to minimizing the objective function since an objective value of zero indicates a feasible packing. For a given solution $\mathbf{x} \in \mathcal{X}$ let $\text{overlap}_{ij}(\mathbf{x})$ be the volume of overlap between boxes i and j, where $i, j \in J$. If $s_i \neq s_j$ we set the overlap equal to zero. The objective function can now be formulated as

$$f(\mathbf{x}) = \sum_{i,j \in J \ i<j} \text{overlap}_{ij}(\mathbf{x}) \tag{10}$$

A similar solution space and objective function was used by Dowsland [10], but with a definition of a larger set of neighbouring solutions obtained by moving any piece to *any* other position. Dowsland notes that the size of this neighbourhood gives a too slow convergence towards a local minimum. Our set of neighbouring solutions is still fairly large but, as will be describe later, with the application of Fast Local Search it is possible to achieve a significant speedup of local search.

Features and augmented objective function

A feature is defined for each pair of boxes $i, j \in J$ where $i < j$ and we let a particular solution $\mathbf{x} \in \mathcal{X}$ exhibit the feature if there is an overlap between box i and j. For boxes $i, j \in J$ the presence of the feature is given by the indicator function

$$I_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{if } \text{overlap}_{ij}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

A similar definition of features is used by Voudouris and Tsang [21] where GLS is used to solve *Partial Constraint Satisfaction Problems* (PCSP). In [21] each of the constraints in a PCSP is relaxed and incorporated into the formulation as a feature with the feature cost given by the violation cost of a constraint. With a weight assigned to each constraint (high for hard constraints) the objective is given as the weighted sum of violated constraints. Each time the local search settles in a local minimum the penalties for one or more of the most *expensive* violated constraints is increased.

In the context of 3D-BPP finding a feasible packing can be viewed as a constraint satisfaction problem with a constraint defined for each pair of boxes stating that the boxes may not overlap. With the weight of a constraint dynamically set to the amount of overlap, it gives an objective defined as the sum of pairwise overlap. The features capture the properties of the solution which account to the value of the objective function. For each feature p_{ij} denotes the

corresponding penalty parameter which initially is zero. The augmented objective function can now be defined as

$$h(\mathbf{x}) = f(\mathbf{x}) + \lambda \cdot \sum_{i,j \in J \; i<j} p_{ij} \cdot I_{ij}(\mathbf{x}) \tag{12}$$

Feature costs

The purpose of the features is to introduce or strengthen constraints on the solution space on the basis of information collected during the search. The source of information that determines which features are penalized in a local minimum $\mathbf{x}_*$ is the feature cost and the amount of previous penalties assigned to the features in $\mathbf{x}_*$. To escape the local minimum we want to penalize features in $\mathbf{x}_*$ with maximum utility

$$\mu_{ij}(\mathbf{x}_*) \;=\; \frac{c_{ij}}{1 + p_{ij}} \cdot I_{ij}(\mathbf{x}_*)$$

Note that a feasible packing is equal to a solution with no exhibited features, so in the long run all features must be eliminated. We have chosen to identify *bad* overlaps on the general principle that an overlap between large boxes is worse than an overlap between small boxes. The feature cost c_{ij} on the one hand depends on the overlap between the boxes i and j and on the other hand on the volume of the boxes. The results presented later are based on the following feature cost:

$$c_{ij}(\mathbf{x}) = \begin{cases} \text{overlap}_{ij}(\mathbf{x}) + \text{volume}(i) + \text{volume}(j) & \text{if overlap}_{ij}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where volume(i) denotes the volume of box $i \in J$. Good results were also obtained with other cost functions, e.g. the product between overlap and volume which very aggressively avoids overlap between relatively large boxes. A cost function which only incorporated the overlap also showed good results.

The λ parameter

The value of λ determines to what degree an increased penalty will modify the augmented objective value and push the local search out of a local minimum. A large value will make the search more aggressive to avoid solutions with penalized features and provoke the search to make large jumps in the solution space without paying much attention to the overlap term in the augmented objective function. A small λ on the other hand may require more penalties to escape a local minimum, but will result in a more cautious exploration of the solution landscape which is more sensitive to the gradient of $f(\mathbf{x})$. However, a disadvantage with a too small λ value might be a too restricted exploration of the solution space.

In Figure 1 two plots are shown that illustrate how the behaviour of the terms in the objective function typically are affected by the choice of λ. The plots show the overlap $f(\mathbf{x})$ drawn with a solid line and the penalty term $\lambda \cdot \sum_{i,j \in J \; i<j} p_{ij}$ drawn with a dashed line as the heuristic finds a feasible packing for the problem.

On the upper figure a small λ-value of 0.4% of the maximum box volume is used. Note the plateaus of $f(\mathbf{x})$ and how the penalty is increased until GLS escapes the local minimum. On the lower figure a λ-value of 4% is used. Most of the plateaus have disappeared and the

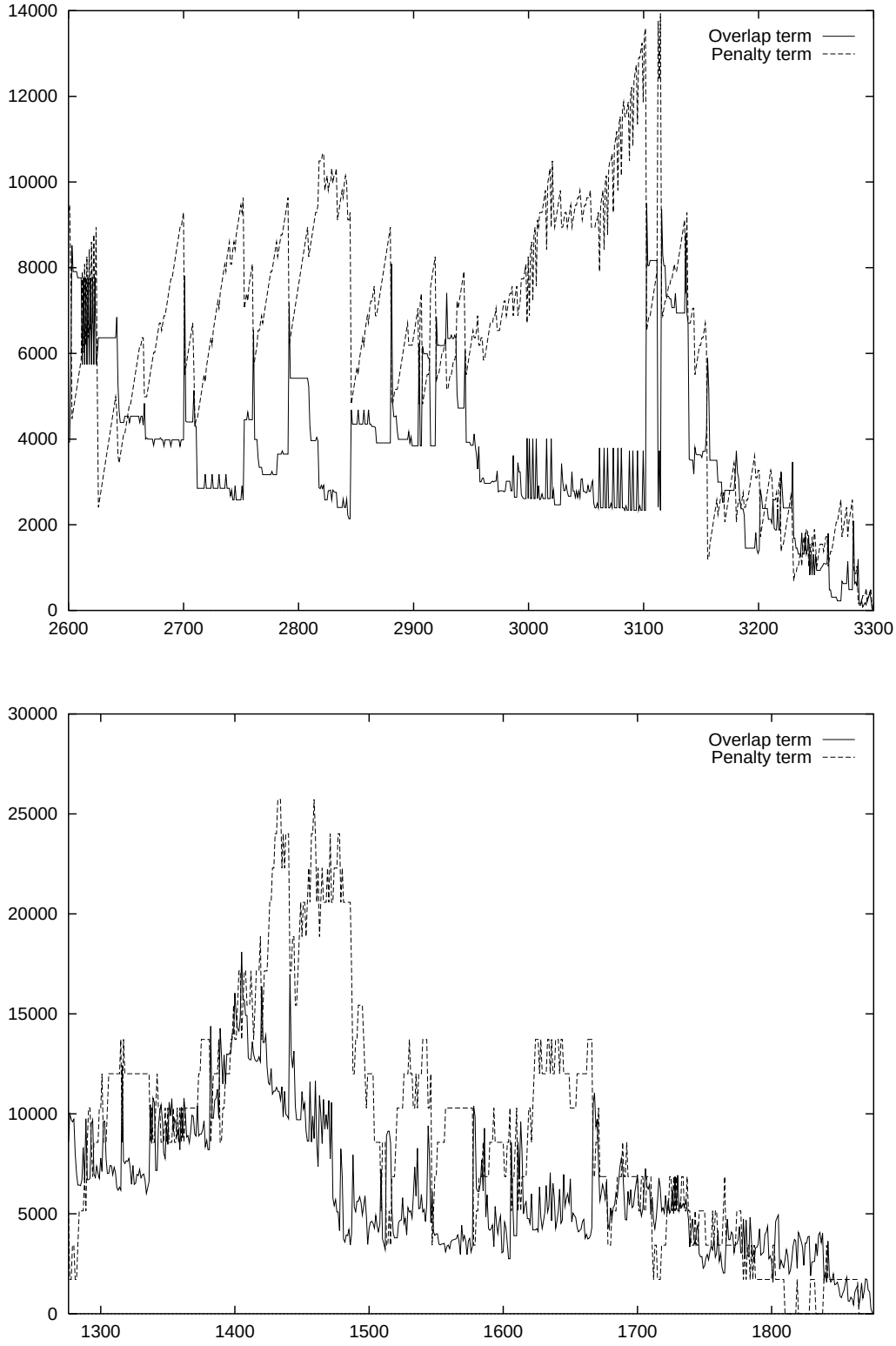


Figure 1: Typical behaviour of the terms in objective function for two different λ values as GLS finds a feasible solution. In the upper figure λ is set to 0.4% of maximum box volume and in the lower figure to 4% of maximum box volume.

overlap function shows a much larger variation, that is, GLS visits a more diverse part of the solution space. Note however that the penalty term reaches much higher values than in the previous figure.

With reference to the present formulation of 3D-BPP the λ expresses the amount of volume that is added to an overlap between the boxes that are associated with the penalized feature. Through empirical tests a value of a few percent of the maximum pairwise overlap showed good results. During these experiments it also seemed as if λ should be chosen dependent on the expected average volume of overlap between the boxes in a solution. As illustrated in Figure 1 a λ-value of 0.4% of the maximum box volume lead to a too slow convergence since it takes very long time before the penalty term grows large enough to escape a local minima. On the other hand a λ-value of 4% implied a too scattered search, since the penalty term grows so quickly that a local minima is not investigated thoroughly. Thus in the computational experiments in Section 5 a λ-value of 1% of the maximum box volume was used, since this value lead to good results for most of the test instances.

We also performed some experiments where the value of λ was dynamically adapted to the specific instance. This was done by looking at the number of plateaus (in the original objective function) during the local search and adjusting λ so that they had an "appropriate" structure. However, we were not able to obtain better results by the self-adjusting framework than those obtained by using a fixed λ-value of 1% of the maximum box volume.

Fast Local Search

Already during the preliminary experiments it was clear that due to the very large neighbourhood adopted, conventional local search methods were to slow to converge to a local minimum. To speed up the local search an implementation of *Fast Local Search (FLS)* [20] has been used. Although FLS alone does not provide very good solutions it has proved to be a powerful tool when combined with GLS (see [19, 20, 21, 22, 23]).

In FLS the neighbourhood is divided into a number of smaller sub-neighbourhoods that can be either *active* or *inactive*. Initially all sub-neighbourhoods are active. FLS now continuously visits the active sub-neighbourhoods in some order. If a sub-neighbourhood is examined and does not contain any improving move it becomes inactive. Otherwise it remains active and the improving move is performed; a reactivation of a number of other sub-neighbourhoods may be performed if we expect these to contain improving moves as a result of the move just performed.

The order in which the sub-neighbourhoods are visited may be either static or dynamic. As the solution value improves, more and more sub-neighbourhoods become inactive, and when all sub-neighbourhoods have become inactive the best solution found is returned by FLS as a local minimum.

The key to decide how to split the neighbourhood into sub-neighbourhoods is to make an association between features and sub-neighbourhoods. The association should enable us to know exactly which sub-neighbourhoods have a direct effect upon the state of a certain feature. This association is used each time GLS settles in a local minimum. As penalties are assigned to one or more features, the sub-neighbourhoods associated with the penalized features are activated and FLS is restarted. Due to the limited reactivation each local optimization using FLS will be aimed at removing the penalized features from the solution instead of exploring all possible moves.

To apply FLS in the present 3D-BPP heuristic we let each box $j \in J$ correspond to a

sub-neighbourhood. As a result each sub-neighbourhood holds moves that correspond to

- all translations of the box along the coordinate axes in the same bin, i.e., the assignment of a new value to either x_j, y_j or z_j, or
- moving the box to the same coordinates in another bin, i.e., the assignment of a new value to s_j.

During a local optimization sub-neighbourhoods are visited in a static round-robin fashion (no reactivation is made during the optimization). When a local minimum is reached the following reactivation scheme is used. First the boxes corresponding to the penalized feature are reactivated. Secondly, we reactivate all those boxes which overlap with the penalized boxes. The latter reactivation is added to permit FLS to pay attention not just to the two overlapping boxes but to the whole area around the penalized feature. Another reactivation scheme which showed good results was to penalize more than one feature and activate the corresponding boxes. For example, assigning penalties to the 5 features having maximum utility and reactivating the corresponding boxes showed good results.

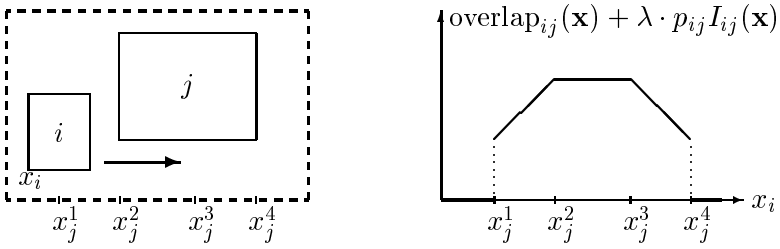
Polynomial-time evaluation of the neighbourhood

As mentioned above each sub-neighbourhood corresponds to all translations of a box along the coordinate axes in the same bin, or the movement of the box to another bin. If we consider translations of box i along the x-axis we wish to minimize the following objective:

$$\min_{x_i=0,\ldots,W-w_i} \quad \sum_{j\in J}(\text{overlap}_{ij}(\mathbf{x}) + \lambda \cdot p_{ij} I_{ij}(\mathbf{x})) \tag{14}$$

Evaluating this function will demand $O(nW)$ time, which is exponential in the input size. In practice this means that for large bin-dimensions it becomes too expensive to search the neighbourhood.

We may, however, decrease this time complexity by showing that only certain coordinates of the neighbourhood need to be investigated. Translating box i along the x-axis will lead to the following overlap function with respect to another box j:



As seen, the overlap function is a piecewise linear function with breakpoints in $x_j^1, x_j^2, x_j^3, x_j^4$. The function (14) is the sum of such piecewise linear functions, and thus it will also be a piecewise linear function with breakpoints $\{x_j^1, x_j^2, x_j^3, x_j^4\}_{j\in J}$. Obviously such a function will attain its minimum in one of the breakpoints or for $x_i = 0$ resp. $x_i = W - w_i$. There are at most $4n$ breakpoints meaning that we only need to consider these candidate positions for placing box i in (14). This leads to an overall $O(n^2)$ time complexity for minimizing (14).

5 Computational experiments

In this section we will present the results from the computational experiments. The presented algorithm was coded in C++ and compiled using the GNU C++ compiler. Experiments were performed on problem instances in both two and three dimensions. For the instances in three dimensions the algorithm was compared with the exact algorithm for 3D-BPP by Martello, Pisinger and Vigo [16] and for the two-dimensional case with the tabu search heuristic for 2D-BPP by Lodi, Martello and Vigo [14, 15].

To construct the initial upper bound the GLS algorithm uses the following simple heuristic: The boxes are sorted according to non-increasing depths, and a subset of boxes with total volume larger than the bin volume are selected. The selected boxes are packed in two dimensions using the shelf-approach [7, 5]. This process is repeated until a number of "bin-slices" have been generated. These "bin-slices" are then combined to whole bins by using a first-fit decreasing algorithm on the depths of the "bin-slices". The heuristic runs in $O(n^2)$ time.

In the following section we will first present the results obtained on the three-dimensional problem, and then the results for the two-dimensional problem.

5.1 3D instances

In order to evaluate the GLS algorithm we compared it with the exact algorithm (MPV) for 3D-BPP by Martello, Pisinger and Vigo [16]. The following instances from [16] were considered for problems with 50 to 200 boxes:

- **Class 1:** The majority of the boxes are very high and deep.
- **Class 4:** The majority of the boxes have large dimensions.
- **Class 5:** The majority of the boxes have small dimensions.
- **Class 6:** Berkey-Wang [5] instances with dimensions randomly generated in a small interval.
- **Class 7:** Berkey-Wang instances with dimensions randomly generated in a medium interval.
- **Class 8:** Berkey-Wang instances with dimensions randomly generated in a large interval.

For each class (i.e. 1, 4, 5, 6, 7 and 8) and number of boxes (i.e. 50, 100, 150 and 200) there were generated 10 different problem instances based on different random seeds. We did not consider classes 2 and 3 from [16], since these have similar properties as class 1. Also class 9 was not considered, since these problems merely have the character of puzzles than of packing problems. Both algorithms were run on a Digital 500au workstation with a 500 MHz 21164 CPU (SPECint95 value of 15.7) with a time limit of 1000 seconds for each instance.

While GLS uses a simple $O(n^2)$ heuristic to construct the first initial solution, the exact algorithm uses two advanced heuristics [16]. The first is based on "bin-slices" which are combined to whole bins by using an exact one-dimensional bin-packing algorithm, and the second based on a branch-and-bound algorithm with limited width of the search tree.

Table 1 compares the average solutions from the GLS algorithm and the MPV algorithm over 10 different problem instances for each class and box count. The first three columns

Table 1: Results for the three-dimensional instances. Time limit is 1000 seconds. The solutions are compared with results obtained with the exact algorithm (MPV) for 3D-BPP by Martello, Pisinger and Vigo [16]. Columns indicate GLS solutions after 60 (z_{60sec}), 150 (z_{150sec}) and 1000 ($z_{1000sec}$) seconds, z_{MPV} shows the best solution of the MPV algorithm, and L_2 the corresponding lower bound. Column opt_{GLS} indicates the number of instances solved to known optimality and "$z_{1000sec} \leq z_{MPV}$" the instances where GLS obtained equal or better solutions than the exact algorithm. GLS solutions are in italics when they are greater than z_{MPV}.

Class	*Bins*	n	*GLS*			*MPV*	L_2	opt_{GLS}	$z_{1000sec} \leq z_{MPV}$
			z_{60sec}	z_{150sec}	$z_{1000sec}$	z_{MPV}			
1	100×100	50	13.4	13.4	13.4	13.6	12.5	3	•
		100	26.9	26.7	26.7	27.3	25.1	0	•
		150	37.5	37.2	37.0	38.2	34.7	0	•
		200	*52.8*	52.1	51.2	52.3	48.4	0	•
4	100×100	50	29.4	29.4	29.4	29.4	28.7	4	•
		100	59.0	59.0	59.0	59.1	57.6	1	•
		150	87.1	86.9	86.8	87.2	85.2	1	•
		200	*119.9*	*119.7*	119.0	119.5	116.3	1	•
5	100×100	50	8.3	8.3	8.3	9.2	7.3	2	•
		100	15.1	15.1	15.1	17.5	12.9	0	•
		150	20.7	20.3	20.2	24.0	17.4	0	•
		200	27.8	27.5	27.2	31.8	24.4	0	•
6	10×10	50	9.8	9.8	9.8	9.8	8.7	1	•
		100	19.3	19.1	19.1	19.4	17.5	0	•
		150	29.5	29.4	29.4	29.6	26.9	0	•
		200	*38.5*	38.0	37.7	38.2	35.0	0	•
7	40×40	50	7.4	7.4	7.4	8.2	6.3	0	•
		100	12.5	12.3	12.3	15.3	10.9	1	•
		150	16.1	15.8	15.8	19.7	13.7	0	•
		200	24.4	24.1	23.5	28.1	21.0	0	•
8	100×100	50	9.2	9.2	9.2	10.1	8.0	0	•
		100	18.9	18.9	18.9	20.2	17.5	1	•
		150	24.5	24.1	23.9	27.3	21.3	1	•
		200	30.6	30.1	29.9	34.9	26.7	0	•
Total			738.6	733.8	730.2	769.9	684.0	16	24
Average			30.78	30.58	30.43	32.08	28.50	0.67	

show the current GLS solution values after 60, 150 and 1000 seconds, respectively. The next column gives the solution value found by the MPV algorithm after 1000 seconds. The column L_2 is the common lower bound used by both the GLS heuristic and the MPV algorithm. The column opt_{GLS} gives the number of times the algorithm finds a solution equivalent to the lower bound. The last column indicates when the GLS algorithm finds an equal or better solution value compared to the MPV algorithm (within 1000 seconds).

The GLS algorithm finds similar or better solutions in all the cases, and even after 60 seconds the solutions are on average considerably better than those obtained by the MPV algorithm. Note especially the class 4 instances where the MPV algorithm performs very well since it does not use much time on the exact filling of a single bin — as only one or two boxes fit into each bin. Anyhow, the GLS algorithm is capable of finding equal or even better solutions for these instances.

On average, the GLS algorithm used 30.43 bins while the MPV algorithm used 32.08 bins. Since the lower bound is 28.50 the relative gap is 6.8% and 12.6%, respectively, meaning that the GLS algorithm decreased the gap to the lower bound to about half its value.

In the appendix figures that plot the development of the GLS algorithm and the exact algorithm as a function of the solution time are shown.

5.2 2D instances

The presented GLS algorithm is also used to solve 2D bin-packing problems by setting the depth of all boxes and bins to a constant value of one. For the two-dimensional instances we compare the GLS algorithms with the tabu search (TS) algorithm by Lodi, Martello and Vigo [14, 15]. This algorithm is based on construction algorithms for packing a single bin, where the tabu search algorithm is used to control the movement of pieces between the bins. It should be emphasized that the GLS algorithm does not take advantage of the fact that the instances are two-dimensional, and thus obviously is somewhat slower than the TS algorithm.

The GLS algorithm was run on a Digital 500au workstation with a 500 MHz 21164 CPU (SPECint95 value of 15.7), while the results for the tabu search algorithm were taken directly from [14] and [15] (these experiments were run on a Silicon Graphics INDY R4000sc with a 100 MHz CPU and Silicon Graphics INDY 10000sc with a 195 MHZ CPU, respectively). The GLS algorithms was assigned a time limit of 100 seconds for each instance, approximately matching the computing effort of the tabu search algorithms.

In the following tables we report the GLS solution values after 5, 30 and 100 seconds in the columns z_{5sec}, z_{30sec} and z_{100sec}, and the solution value found by TS is shown in column z_{TS}; the TS solution is taken from [14], since no results were reported for these instances in [15]. The column L_2 is the lower bound used by GLS while L_{LMV} shows the lower bound reported in [14].

In Table 2 results on problem instances from the literature are reported. The considered instances are `cgcut1-cgcut3` [6] and `gcut1-gcut13/ngcut1-ngcut12` [2, 3] (all these instances are obtained from the OR-Library, see `http://www.ms.ic.ac.uk/info.html`). These instances are two-dimensional cutting problems which were transformed to 2D-BPP in the following way: First the value of the pieces is ignored and secondly for the `cgcut` and `ngcut` instances the maximum number of pieces is generated for each type.

The GLS algorithm always finds a solution which is at least as good as the TS solution. On average, GLS used 5.82 bins while TS used 6.11 bins. With a lower bound of 5.50 the relative gap has been decreased from 11.1% to 5.8%. It should be emphasized that since we use a very

Table 2: Results for two-dimensional instances from the literature. Time limit is 100 CPU seconds. The columns z_{5sec}, z_{30sec} and z_{100sec} indicate the solutions obtained by GLS after 5, 30 and 100 seconds. The solutions are compared with results (z_{TS}) obtained with the tabu search algorithm by Lodi, Martello and Vigo [14]. In the following columns L_2 is the lower bound, and L_{LMV} the lower bound from [14]. The column "$z_{100sec} \leq z_{TS}$" indicates the instances where GLS obtained equal or better solutions than TS, "$z_{100sec} < ub$" the instances where GLS was able to improve the initial upper bound and opt_{GLS} the instances solved to known optimality. GLS solutions written in italics indicate the GLS solutions that are greater than z_{TS}.

Instance	*GLS*			TS	L_2	L_{LMV}	$z_{100sec} \leq z_{TS}$	$z_{100sec} < ub$	opt_{GLS}
	z_{5sec}	z_{30sec}	z_{100sec}	z_{TS}					
cgcut1	2			2	2	2	•		•
cgcut2	2			2	2	2	•		•
cgcut3	23			23	23	23	•		•
gcut1	5	5	5	5	4	4	•		•
gcut2	6	6	6	6	5	6	•	•	•
gcut3	8	8		8	8	8	•	•	•
gcut4	14	14	14	14	13	13	•		•
gcut5	3			4	3	3	•	•	•
gcut6	7	7	7	7	6	6	•		•
gcut7	11	11	11	12	10	10	•	•	
gcut8	14	13	13	14	12	12	•		
gcut9	3			3	3	3	•	•	•
gcut10	7	7	7	8	6	7	•		•
gcut11	9	9	9	9	8	8	•	•	•
gcut12	16			16	16	16	•		•
gcut13	2			2	2	2	•	•	•
ngcut1	3	3	3	3	2	2	•		•
ngcut2	4	4	4	4	3	3	•		•
ngcut3	3			4	3	3	•		•
ngcut4	2			2	2	2	•		•
ngcut5	3			3	3	3	•	•	•
ngcut6	3	3	3	3	2	2	•	•	•
ngcut7	1			1	1	1	•	•	•
ngcut8	2			2	2	2	•	•	•
ngcut9	3			4	3	3	•	•	•
ngcut10	3			3	3	3	•		•
ngcut11	2			3	2	2	•	•	•
ngcut12	3			4	3	3	•	•	•
Total	164	163	163	171	152	154	28	14	26
Average	5.86	5.82	5.82	6.11	5.43	5.50			

Table 3: Problem classes proposed by Berkey-Wang [5]

Class	*Items*	*Bins*
1	$[1, 10] \times [1, 10]$	10×10
2	$[1, 10] \times [1, 10]$	30×30
3	$[1, 35] \times [1, 35]$	40×40
4	$[1, 35] \times [1, 35]$	100×100
5	$[1, 100] \times [1, 100]$	100×100
6	$[1, 100] \times [1, 100]$	300×300

simple heuristic for the initial solution, it is in most cases the GLS algorithm which actually finds the improved solutions. Moreover the GLS algorithm finds the returned solutions already after 30 seconds, which is comparable to the solution times of the TS algorithm. Only two instances are not solved to known optimality.

Table 4 compares the GLS and TS heuristics for the Berkey-Wang instances [5] (with TS results taken from [15]). The classes are described in Table 3 and were considered for problems with 20, 40, 60, 80 and 100 items with 10 different instances for each class item number (we used the same generated problem instances as in [14]; these instances can be obtained from `http://www.or.deis.unibo.it/ORinstances/`).

In all the instances the GLS algorithm finds an equally good or better solutions than the TS algorithm. Actually, after 5 seconds, it finds better solutions on average than the TS algorithm. After 100 seconds, the GLS algorithm finds the optimal solution for more than 60% of the instances. Looking at the average values, the GLS algorithm finds solutions using 9.90 bins, while the TS algorithm finds solutions using 10.11 bins. The average lower bound is 9.48 bins, and thus the relative deviation is 4.4% and 6.6%, respectively.

Table 5 considers the problem instances proposed by Martello and Vigo [18]. All bins have dimensions 100×100 while the items have the following properties:

- **Class 7:** The majority of the items are wide.
- **Class 8:** The majority of the items are high.
- **Class 9:** The majority of the items are large in both dimensions.
- **Class 10:** The majority of the items are small in both dimensions.

The GLS algorithm finds equivalent or better solutions than the TS algorithm (from [15]) for almost all of the problems. The GLS algorithm finds the optimal solution for more than 50% of these instances. The average solution values are 21.58 for GLS, and 21.65 for TS. The average lower bound is 21.10, and thus the relative deviation is 2.3% and 2.6%, respectively.

6 Conclusion

Our experiments have shown that Guided Local Search can be applied to bin packing problems in two- and three-dimensions with success. Since the concept of GLS is still relatively new, it is important to determine the classes of problems for which it is suitable.

Table 4: Results for the two-dimensional Berkey-Wang instances [5]. Column opt_{GLS} indicates the number of instances solved to known optimality. The solutions are compared with results (z_{TS}) obtained with the tabu search algorithm by Lodi, Martello and Vigo [15]. For a description of the other columns we refer to Table 2.

Class	*Bins*	n	*GLS*			*TS*	L_2	L_{LMV}	opt_{GLS}	$z_{100sec} \leq z_{TS}$
			z_{5sec}	z_{30sec}	z_{100sec}	z_{TS}				
1	10×10	20	7.1	7.1	7.1	7.1	6.6	6.7	6	•
		40	13.4	13.4	13.4	13.6	12.8	12.8	5	•
		60	*20.2*	20.1	20.1	20.1	19.0	19.3	2	•
		80	27.7	27.5	27.5	28.2	26.2	26.9	5	•
		100	32.4	32.1	32.1	32.7	30.8	31.4	3	•
2	30×30	20	1.0	1.0	1.0	1.0	1.0	1.0	10	•
		40	1.9	1.9	1.9	2.1	1.9	1.9	10	•
		60	2.5	2.5	2.5	2.8	2.5	2.5	10	•
		80	3.2	3.2	3.1	3.3	3.1	3.1	10	•
		100	4.0	3.9	3.9	4.0	3.9	3.9	10	•
3	40×40	20	5.1	5.1	5.1	5.5	4.6	4.6	5	•
		40	9.6	9.5	9.4	9.8	8.9	8.8	5	•
		60	14.0	14.0	14.0	14.0	13.2	13.3	3	•
		80	19.6	19.3	19.1	19.9	17.9	18.4	6	•
		100	23.1	22.9	22.6	23.7	21.4	21.7	1	•
4	100×100	20	1.0	1.0	1.0	1.0	1.0	1.0	10	•
		40	1.9	1.9	1.9	1.9	1.9	1.9	10	•
		60	2.5	2.5	2.5	2.6	2.3	2.3	8	•
		80	3.3	3.3	3.3	3.3	3.0	3.0	7	•
		100	*3.9*	*3.9*	3.8	3.8	3.7	3.7	9	•
5	100×100	20	6.5	6.5	6.5	6.7	6.0	6.0	5	•
		40	11.9	11.9	11.9	11.9	11.3	11.4	6	•
		60	18.2	18.1	18.1	18.2	17.0	17.2	2	•
		80	25.0	25.0	24.9	25.0	23.4	23.6	1	•
		100	29.3	28.8	28.8	29.5	27.2	27.3	1	•
6	300×300	20	1.0	1.0	1.0	1.0	1.0	1.0	10	•
		40	1.9	1.8	1.8	2.1	1.5	1.5	7	•
		60	2.2	2.2	2.2	2.2	2.1	2.1	9	•
		80	3.0	3.0	3.0	3.0	3.0	3.0	10	•
		100	*3.5*	3.4	3.4	3.4	3.2	3.2	8	•
Total			299.9	297.8	296.9	303.4	281.4	284.5	194	30
Average			10.00	9.93	9.90	10.11	9.38	9.48	6.47	

Table 5: Results for the two-dimensional Martello-Vigo instances [18]. Column opt_{GLS} indicates the number of instances solved to known optimality. The solutions are compared with results (z_{TS}) obtained with the tabu search algorithm by Lodi, Martello and Vigo [15]. For a description of the other columns we refer to Table 2.

Class	*Bins*	n	*GLS*			*TS*	L_2	L_{LMV}	opt_{GLS}	$z_{100sec} \leq z_{TS}$
			z_{5sec}	z_{30sec}	z_{100sec}	z_{TS}				
7	100×100	20	5.5	5.5	5.5	5.5	5.2	5.3	8	•
		40	11.3	11.3	11.3	11.4	10.4	10.8	5	•
		60	16.1	16.1	15.9	16.3	14.9	15.5	6	•
		80	*23.5*	*23.3*	23.2	23.2	21.7	22.3	1	•
		100	*27.8*	27.6	27.5	27.6	25.7	26.8	3	•
8	100×100	20	5.8	5.8	5.8	5.8	5.3	5.5	7	•
		40	*11.5*	11.4	11.4	11.4	10.4	11.1	7	•
		60	*16.5*	*16.3*	*16.3*	16.2	15.3	15.9	6	
		80	*22.8*	*22.8*	22.5	22.6	21.4	22.2	7	•
		100	28.3	28.2	28.1	28.4	26.5	27.3	2	•
9	100×100	20	14.3	14.3	14.3	14.3	14.3	14.3	10	•
		40	*27.8*	*27.8*	*27.8*	27.7	27.5	27.4	7	
		60	43.7	43.7	43.7	43.7	43.5	43.3	8	•
		80	*57.7*	*57.7*	*57.7*	57.5	57.3	56.9	6	
		100	69.5	69.5	69.5	69.6	69.2	68.9	7	•
10	100×100	20	4.2	4.2	4.2	4.4	3.9	4.0	8	•
		40	7.4	7.4	7.4	7.5	7.0	7.1	7	•
		60	10.3	10.2	10.2	10.4	9.5	9.7	5	•
		80	*13.2*	13.0	13.0	13.0	12.2	12.3	3	•
		100	16.3	16.3	16.2	16.5	15.3	15.3	1	•
Total			433.5	432.4	431.5	433.0	416.5	421.9	114	17
Average			21.68	21.62	21.58	21.65	20.83	21.10	5.70	

However, a more important achievement is that the presented algorithm applies local search directly on the packing problems. Most other successful local search heuristics for cutting and packing somehow make use of a construction algorithm inside the search. Thus the local search is restricted to a higher level of the search, like assigning items to bins, or determining an appropriate packing order. But the construction algorithm may become a bottleneck for the search, since one will never find better solutions than the construction algorithm is able to produce. Another benefit of working directly on the packing problem is, that we are able to handle items and bins of a general form as long as the overlap between items can be determined efficiently.

Dowsland [10] used a similar objective function without significant success. Using simulated annealing the investigated neighbourhood became too large, and thus the convergence towards good solutions was slow. It is thus interesting how GLS and FLS are able to focus the search on interesting parts of the solution space.

Acknowledgement

The authors wish to thank Kenneth Jönsson for having taken part in the development of the first version of this algorithm.

References

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[2] J.E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society*, 36:297–306, 1985.

[3] J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33:49–64, 1985.

[4] B.E. Bengtsson. Packing rectangular pieces – a heuristic approach. *The Computer Journal*, 25:353–357, 1982.

[5] J.O. Berkey and P.Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[6] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25:30–44, 1977.

[7] F.K.R. Chung, M.R. Garey, and D.S. Johnson. On packing two-dimensional bins. *SIAM Journal of Algebraic and Discrete Methods*, 3(1):66–76, 1982.

[8] A.L Corcoran III and R.L. Wainwright. A genetic algorithm for packing in three dimensions. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 1021–1030, 1992.

[9] M. Dell'Amico and S.Martello. Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7(2):191–200, 1995.

[10] K. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

[11] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[12] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and Packing (C&P). In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[13] B. Kröger. Guillotinable bin packing: A genetic approach. *European Journal of Operational Research*, 84:645–661, 1995.

[14] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research*, 112:158–166, 1999.

[15] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, to appear.

[16] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, march–april, 2000.

[17] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[18] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[19] C. Voudouris and E. Tsang. Function optimization using guided local search. Technical Report CSM-249, Dept. of Computer Science, University of Essex, England, 1995.

[20] C. Voudouris and E. Tsang. Guided local search. Technical Report CSM-247, Dept. of Computer Science, University of Essex, England, 1995.

[21] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of Practical Application of Constraint Technology (PACT'96)*, pages 337–356, 1996.

[22] C. Voudouris and E. Tsang. Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters*, 20(3):119–127, 1997.

[23] C. Voudouris and E. Tsang. Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

Appendix

On the following pages figures are shown for each problem class in three dimensions. The figures show plots of the development of the GLS algorithm and the exact algorithm as a function of the solution time.

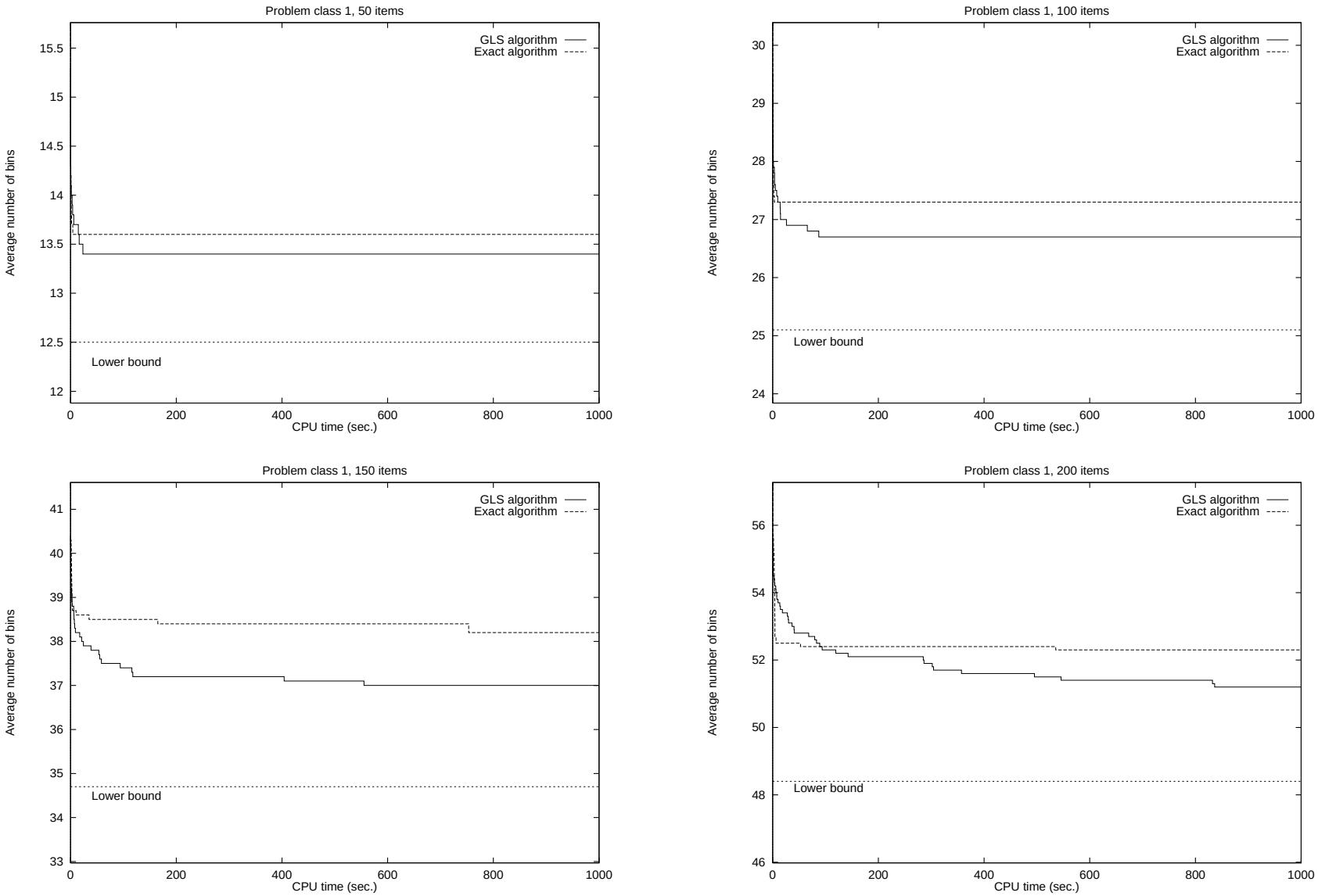


Figure 2: Plots showing the average solutions on 10 different test instances of class 1.

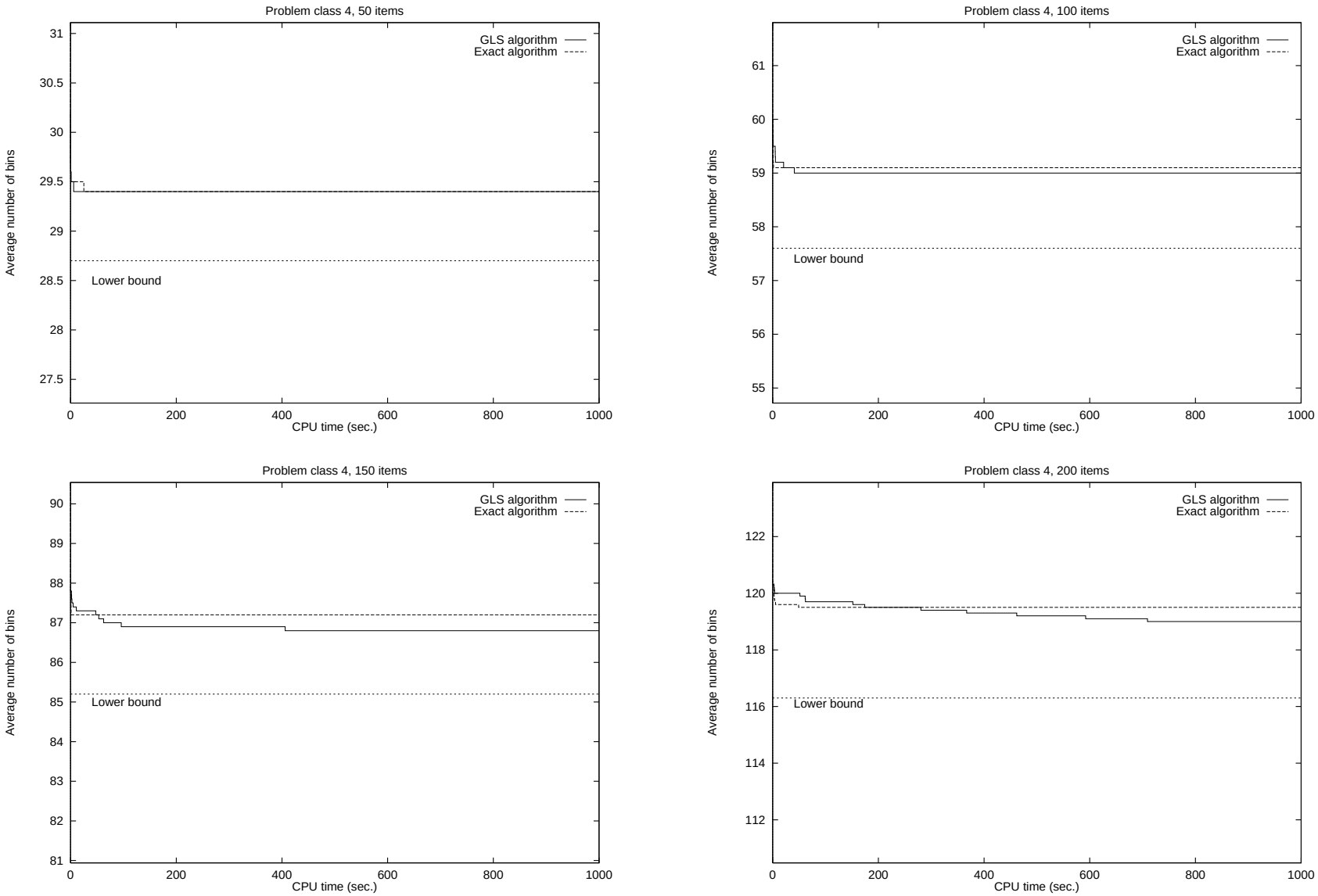


Figure 3: Plots showing the average solutions on 10 different test instances of class 4.

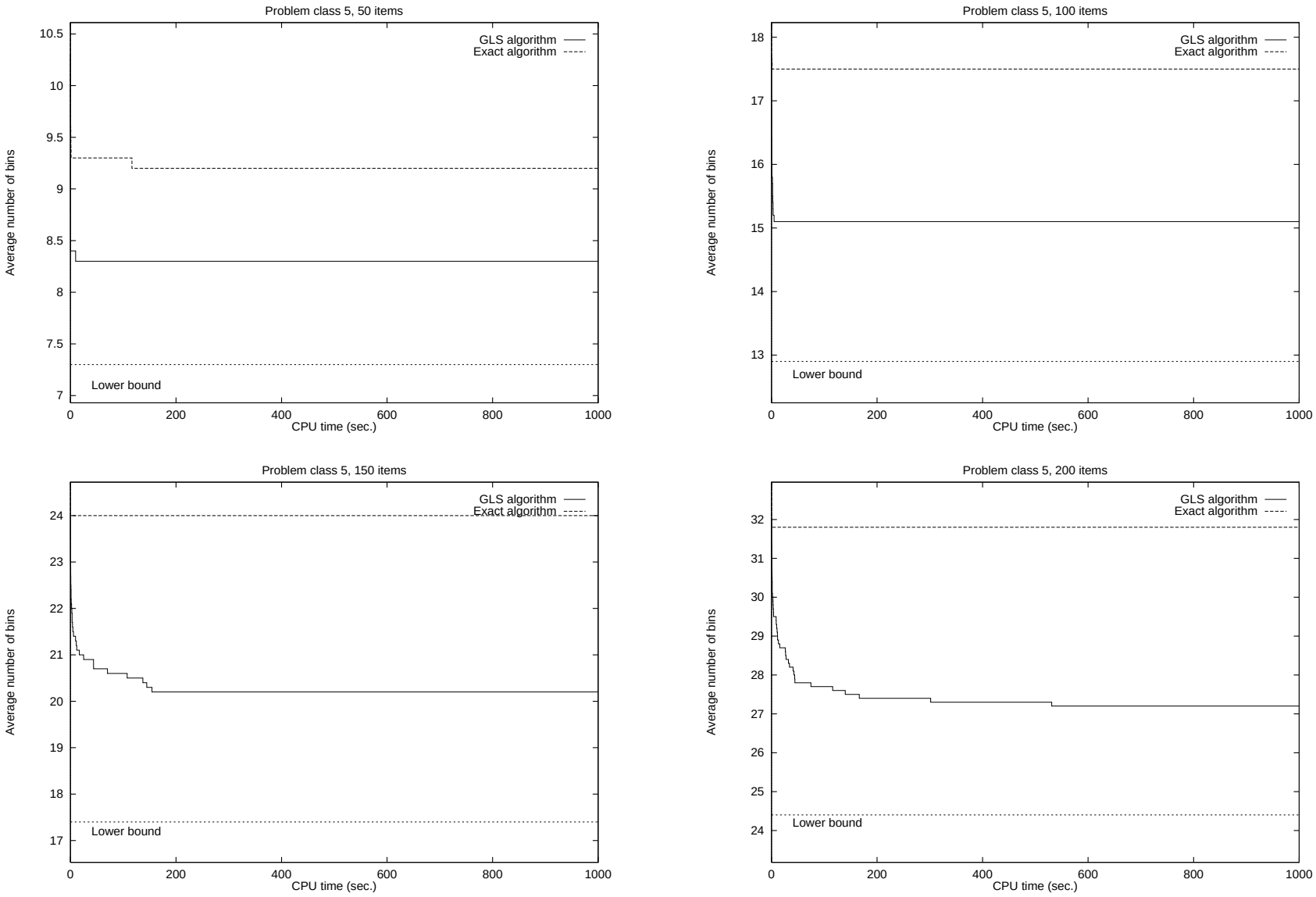


Figure 4: Plots showing the average solutions on 10 different test instances of class 5.

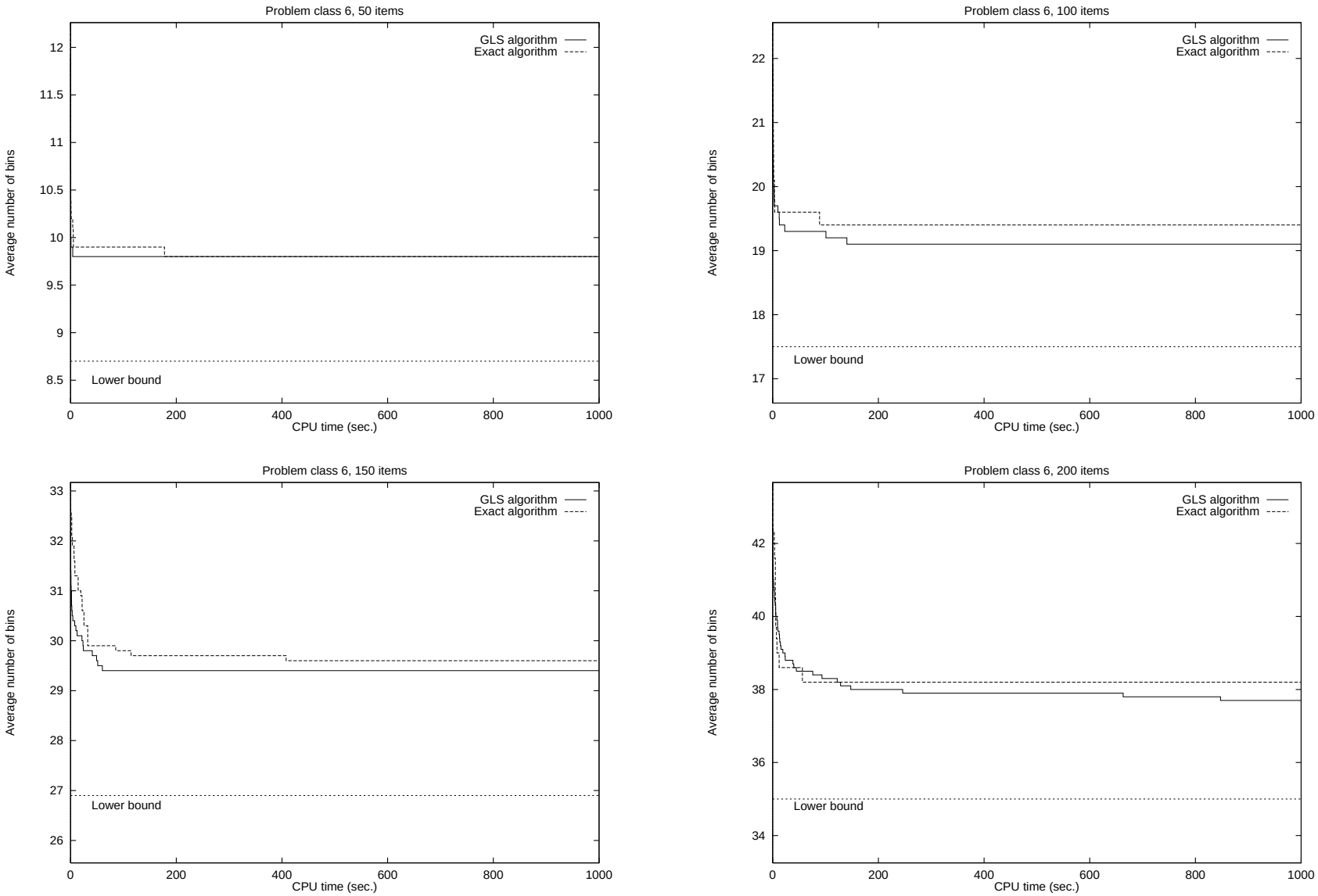


Figure 5: Plots showing the average solutions on 10 different test instances of class 6.

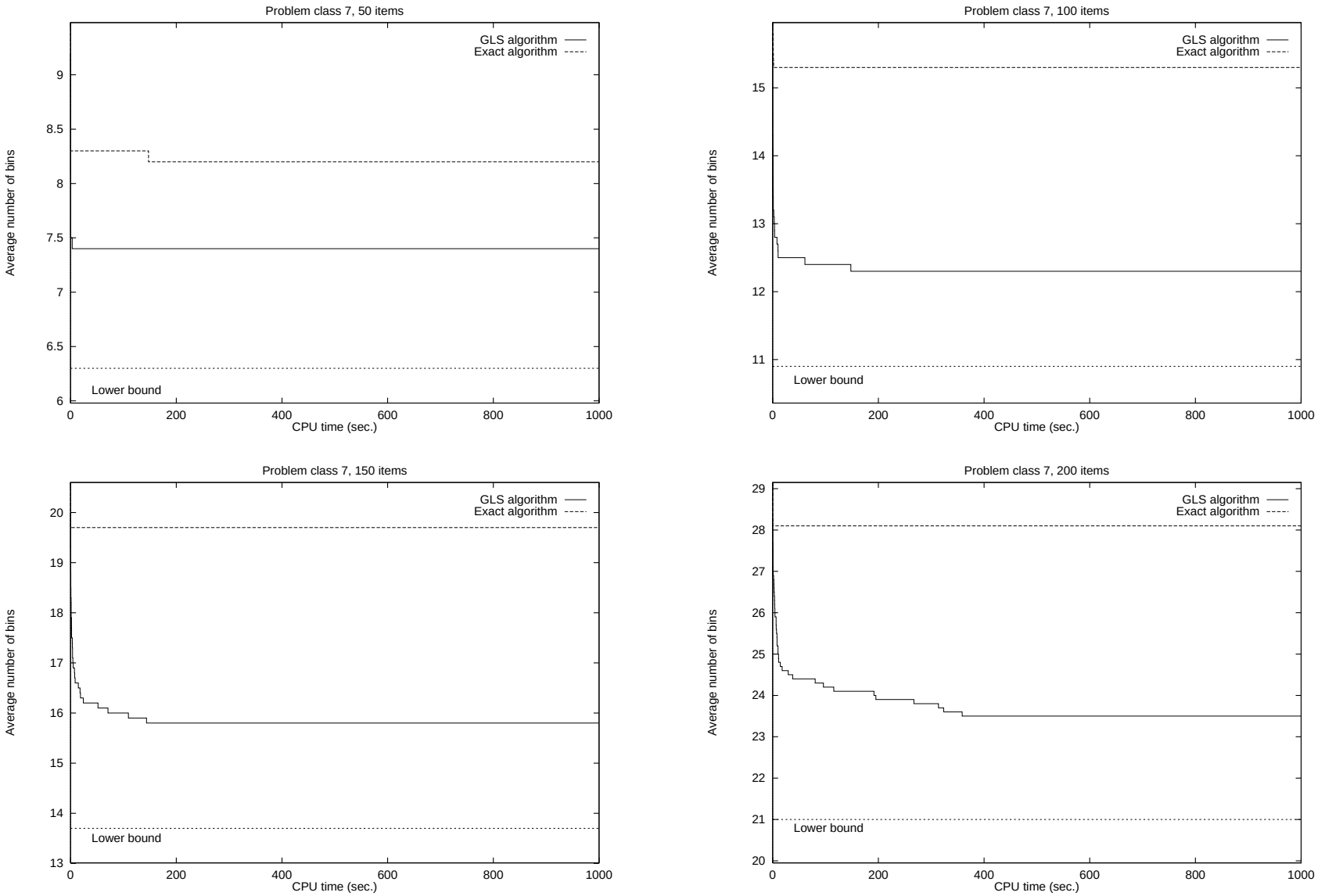


Figure 6: Plots showing the average solutions on 10 different test instances of class 7.

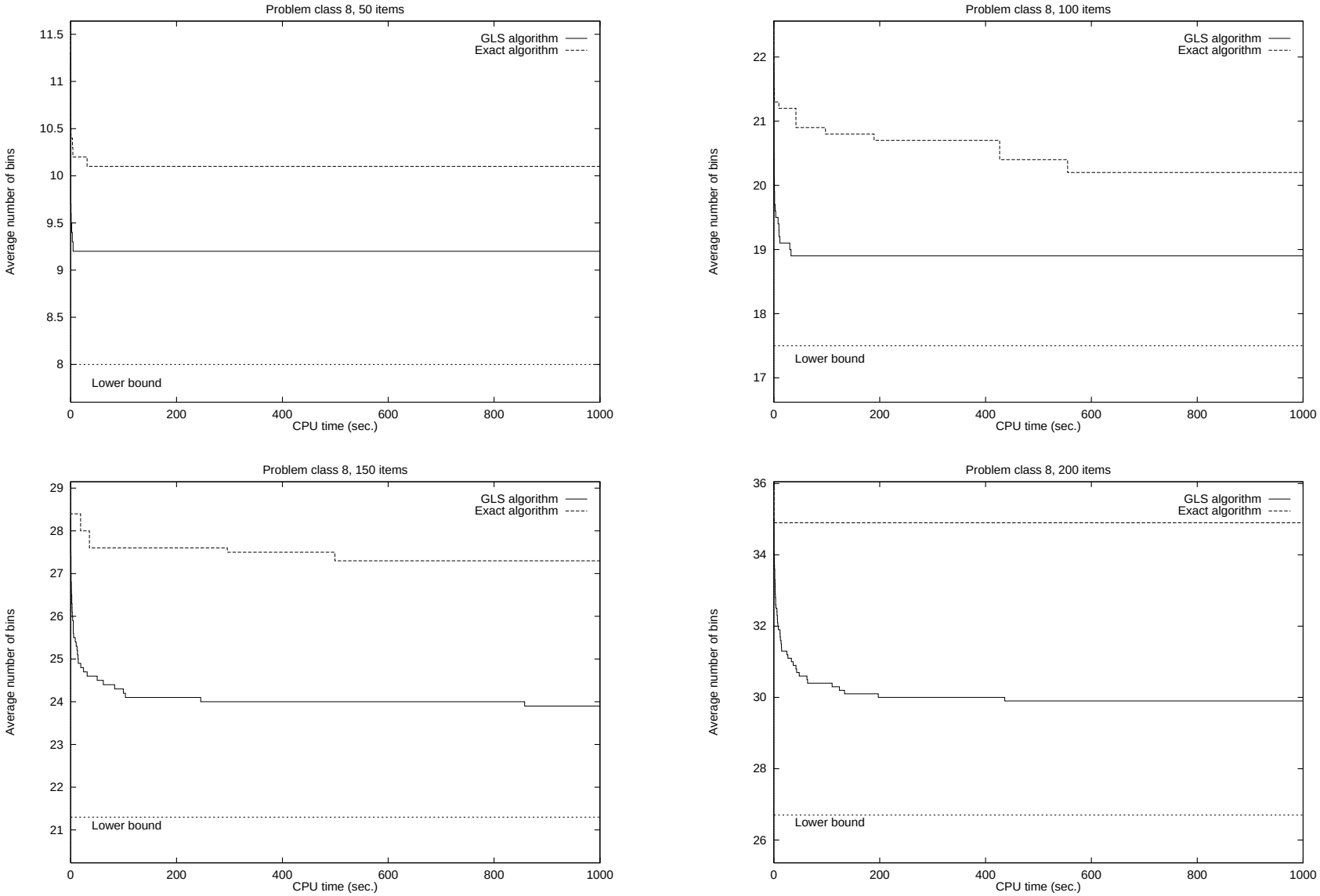


Figure 7: Plots showing the average solutions on 10 different test instances of class 8.