A linear approximation algorithm for bin packing with absolute approximation factor $\frac{3}{2}$

Rudolf Berghammer* , Florian Reuter

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Olshausenstraße 40, D-24098 Kiel, Germany

Received 16 May 2002; received in revised form 31 October 2002; accepted 4 November 2002

Abstract

We present a new approximation algorithm for the bin packing problem which has a linear running time and an absolute approximation factor of $\frac{3}{2}$. It is known that this approximation factor is the best factor achievable, unless $P = NP$.

Keywords: Bin packing; Approximation algorithm; Formal program development

1. Introduction

Given a set of objects, each supplied with a certain weight, the (offline-version of) the well-known bin packing problem asks to pack them into bins of equal capacity in such a way that a minimum number of bins is used. The problem has many real-world applications (for example in stock-cutting or when loading trucks or railway carriages subject to weight limitations) but, unfortunately, is $NP$-hard. Therefore, in the last decades a lot of approximation algorithms have been developed to compute near-optimal solutions since in practice near-optimality is often good enough. See [2] for an overview.

In the case of the bin packing problem there are two standard metrics for the worst-case estimation of an approximation. First, $r$ is said to be an absolute approximation factor of an approximation algorithm if for all inputs the size of the computed solution $P$ is guaranteed to be no greater than $r$-times the size of an optimal solution $P^*$. If,

* Corresponding author. Fax: +49-431-5661-43.
E-mail addresses: rub@informatik.uni-kiel.de (R. Berghammer), flr@informatik.uni-kiel.de (F. Reuter).
however, $|P| \leq r|P^*|$ does not hold for all inputs but only for those for which $n \leq |P^*|$ for some natural number $n$, then $r$ is called an asymptotic approximation factor.

Both approximation factors have their merits, but in this paper we only deal with the absolute approximation factor. It is of interest especially when relatively small sets of objects are considered. As shown in [8], there is no approximation algorithm for the bin packing problem with an absolute approximation factor smaller than $\frac{3}{2}$, unless $P=NP$. Using the simple Next Fit approach (cf. [2] for details), one obtains a linear running time and the smallest absolute approximation factor $r=2$. Even the value $r=\frac{3}{2}$ is possible, for example, if the objects are sorted in order of non-increasing weight in a preparatory step and after that the so-called Best Fit rule (cf. again [2]) is applied. But all approximation algorithms with this—in all probability—optimal absolute approximation factor which can be found in the literature are non-linear; the best running time is $O(n \log n)$, where $n$ is the number of input objects. See again the overview paper [2].

In this paper we present a new approximation algorithm for the bin packing problem. It is optimal wrt the running time, i.e., has linear running time. Unless $P=NP$, it is also optimal wrt the absolute approximation factor since we are able to prove that it has an absolute approximation factor of $\frac{3}{2}$. To show the correctness of our algorithm we start with a formal specification of the problem and use then techniques of formal program specification, verification, and development. The reason for this proceeding is that we want to obtain an algorithm which is not only correct “in principle” but in all details and which immediately can be formulated as a structured program in a conventional programming language (like Pascal, Modula, C, or Java). Furthermore, by developing the algorithm from a formal specification (instead of presenting only its final form together with its justification as usually preferred by the “algorithms community”) we want to show how algorithm design can benefit from ideas and techniques of formal program development.

Our approximation algorithm follows the Best Fit idea. In contrast with the original approach, however, it works with two partial solutions $P_1$ and $P_2$ instead of one and two auxiliary bins $B_1$ and $B_2$—one for each partial solution. Roughly speaking, our algorithm proceeds as follows: First, the objects are packed one by one into $B_1$ until its capacity would be exceeded by the insertion of some object $u$. In this situation the contents of $B_1$ is inserted into $P_1$, the bin $B_1$ is cleared, $u$ is packed into $B_2$, and the process starts again with the remaining objects. If, however, the insertion of $u$ would lead to an overfilling of $B_1$ as well as $B_2$, then additionally the contents of $B_2$ is inserted into $P_2$ and $u$ is packed into the cleared $B_2$. This “book-keeping” in combination with a suitable selection of the next object (based on a partition of the objects into small and large ones at the beginning of the algorithm) allows us to avoid the costly search of a bin the next object will fit in optimally.

The remainder of the paper is organized as follows: First, we recall the formal definition of the bin packing problem in Section 2. This leads to the pre- and postcondition of our algorithm in the sense of partial or total correctness, i.e., the problem specification. In Section 3 we then present a simple algorithm, called BinPACKING, and use the invariant technique (see for example [4,5]) to verify that it computes a solution of an instance of the bin packing problem, i.e., satisfies the “feasibility-part” of the problem.
2. The problem specification

An instance of the bin packing problem consists of a finite, non-empty set $U$ of objects and a positive bin capacity $c \in \mathbb{N}$. Each object $u \in U$ has a weight $w(u) \in \mathbb{R}$ such that $0 < w(u) \leq c$. An object $u \in U$ is called small if $w(u) \leq c/2$; otherwise $u$ is called a large object. The weight of a set $B \subseteq 2^U$ of objects is defined as $w(B) = \sum_{u \in B} w(u)$.

Given an instance $(U, c)$ of the bin packing problem, a set $P \subseteq 2^U$ is called a (feasible) solution or a bin packing if $P$ is a partition of $U$ and $w(B) \leq c$ for all $B \in P$. In this case each member of $P$ is said to be a bin. A solution $P$ is said to be optimal if $|P| \leq |Q|$ for all solutions $Q$ of $(U, c)$. By $\text{opt}(U, c)$ we denote the number of bins of the optimal solutions of $(U, c)$. As mentioned in the introduction, an approximation algorithm for the bin packing problem has absolute approximation factor $r \in \mathbb{R}$ if it computes for all inputs $(U, c)$ only solutions $P$ for which $|P| \leq r \cdot \text{opt}(U, c)$.

In the next three sections we develop a linear approximation algorithm for the bin packing problem and formally prove its correctness using the invariant technique. Its absolute approximation factor is $\frac{3}{2}$, which is optimal unless $P = \mathsf{NP}$. To make the technical presentation easier we first assume besides the set $U$ of objects, the weight function $w$, and the bin capacity $c$ also the sets $S$ and $L$ of small, respectively, large objects of $U$ as input. Later on in Section 5 we will skip this restriction. Furthermore, we use the variable $P$ to store the result of the algorithm. Hence, the formal problem specification is given by the conjunction of the three conditions

$$\forall u \in U : 0 < w(u) \leq c, \quad S = \left\{ u \in U : w(u) \leq \frac{c}{2} \right\}, \quad L = U \setminus S \quad (1)$$

as precondition $\text{Pre}(U, w, c, S, L)$ and the conjunction of the three conditions

$$P \text{ partition of } U, \quad \forall B \in P : w(B) \leq c, \quad |P| \leq \frac{3}{2} \cdot \text{opt}(U, c) \quad (2)$$

as postcondition $\text{Post}(P)$. To enhance readability, in the case of postconditions and loop invariants we only mention the non-input variables.

3. An algorithm for feasible solutions

In this section we present an algorithm, called $\text{BinPacking}$, which is correct wrt the precondition $\text{Pre}(U, w, c, S, L)$ and the conjunction of the first two conditions of $\text{Post}(P)$
as postcondition. This means that it computes a solution of the instance \((U,c)\) of the bin packing problem.

To formulate the algorithm we will use an unary operation \([\cdot]\) from sets of objects to sets of sets of objects, which is defined by

\[
\begin{align*}
[\emptyset] = \emptyset, & \quad [B] = \{B\} \text{ if } B \neq \emptyset.
\end{align*}
\]

From this definition we get \(P \cup [\emptyset] = P\) and \(P \cup [B] = P \cup \{B\}\) for all \(P \subseteq 2^U\) and non-empty \(B \in 2^U\). These equations show that the operation \([\cdot]\) can be used to prevent the insertion of an empty set into a partially computed solution. This is necessary to maintain the first property of the postcondition.

The algorithm \textsc{BinPacking} uses two variables \(P_1\) and \(P_2\) of type \(2^2^U\) (i.e., for sets of sets of objects), three variables \(B_1\), \(B_2\), and \(V\) of type \(2^U\) (i.e., for sets of objects), and one variable \(u\) of type \(U\) (i.e., for an object) and looks in a Modula-like syntax as follows:

\[
\begin{align*}
\textsc{BinPacking}(U,w,c,S,L) \\
P_1 &:= \emptyset; P_2 := \emptyset; B_1 := \emptyset; B_2 := \emptyset; V := U; \\
\text{while } V \cap S \neq \emptyset \text{ do} \\
\text{u} &\in V; V := V \setminus \{u\}; \\
\text{if } w(B_1) + w(u) \leq c &\text{ then } B_1 := B_1 \cup \{u\} \\
\text{else if } w(B_2) + w(u) \leq c &\text{ then } B_2 := B_2 \cup \{u\} \\
\text{else } P_2 := P_2 \cup [B_2]; B_2 := \{u\} &\text{ fi;} \\
P_1 &:= P_1 \cup [B_1]; B_1 := \emptyset \text{ fi;} \\
P &:= P_1 \cup [B_1] \cup P_2 \cup [B_2] \cup \{\{v\} : v \in V\}; \\
\text{return } P
\end{align*}
\]

In this program—which is a (first) formalization and completion of the sketch given in the introduction—the statement \(u \in V\) (known from the refinement calculus; see for example [6]) non-deterministically assigns some member of the value of \(V\) to the variable \(u\).

Obviously, the program \textsc{BinPacking} terminates. To show the (partial) correctness of \textsc{BinPacking} wrt. the precondition \(\text{Pre}(U,w,c,S,L)\) and the first two properties of the postcondition \(\text{Post}(P)\) we use

\[
P_1 \cup [B_1] \cup P_2 \cup [B_2] \cup \{\{v\} : v \in V\} \text{ is a solution of } (U,c)
\]

as loop invariant \(\text{Inv}_1(P_1,P_2,B_1,B_2,V)\). Due to the equation

\[
\emptyset \cup [\emptyset] \cup \emptyset \cup [\emptyset] \cup \{\{v\} : v \in U\} = \{\{v\} : v \in U\}
\]

and since the set \(\{\{v\} : v \in U\}\) of all singleton subsets of \(U\) constitutes a partition of \(U\) with \(w(\{v\}) = w(v) \leq c\) (first condition of (1)), we obtain \(\text{Inv}_1(\emptyset,\emptyset,\emptyset,\emptyset,U)\) to be true. Hence, we have:
Theorem 3.1. The loop invariant \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \) is established by the initialization of the variables of \( \text{BinPacking} \).

This theorem is the first proof obligation of the formal verification that the program \( \text{BinPacking} \) is correct. In the next theorem the decisive second proof obligation of our correctness proof is shown.

Theorem 3.2. The loop invariant \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \) is maintained by the body of the while-loop of \( \text{BinPacking} \).

Proof. Assume \( V \cap S \neq \emptyset \) and \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \) to be true. Furthermore, let \( u \) be an arbitrary element of \( V \). We have to prove that the loop invariant also holds for the new values of the variables \( P_1, P_2, B_1, B_2, \) and \( V \), provided the respective guards of the nested conditional are true.

The first case is \( w(B_1) + w(u) \leq c \). If \( P_1 \cup \{B_1\} \cup P_2 \cup \{B_2\} \cup \{v : v \in V\} \) is a partition of \( U \), then the same holds if \( \{u\} \) is replaced by \( B_1 \cup \{u\} \). Combining this fact with the equation

\[
P_1 \cup \{B_1 \cup \{u\}\} \cup P_2 \cup \{B_2\} \cup \{v : v \in V \setminus \{u\}\}
= P_1 \cup \{B_1 \cup \{u\}\} \cup P_2 \cup \{B_2\} \cup \{v : v \in V \setminus \{u\}\}
\]

yields the partition property of \( P_1 \cup \{B_1 \cup \{u\}\} \cup P_2 \cup \{B_2\} \cup \{v : v \in V \setminus \{u\}\} \). But this set is even a solution of \((U, c)\): If \( B \) is from \( P_1, P_2, B_2 \), or a singleton set \( \{v\} \), where \( v \in V \setminus \{u\} \), then \( w(B) \leq c \) follows from (4). The remaining estimation \( w(B_1 \cup \{u\}) \leq w(B_1) + w(u) \leq c \) is an immediate consequence of the assumption of the case.

In the same manner the validity of \( \text{Inv}_1(P_1 \cup \{B_1\}, P_2, \emptyset, B_2 \cup \{u\}, V \setminus \{u\}) \) can be shown if \( w(B_1) + w(u) > c \) and \( w(B_2) + w(u) \leq c \).

The final case is given by \( w(B_1) + w(u) > c \) and \( w(B_2) + w(u) > c \). Here we start with

\[
P_1 \cup \{B_1\} \cup \emptyset \cup P_2 \cup \{B_2\} \cup \{v : v \in V \setminus \{u\}\}
= P_1 \cup \{B_1\} \cup \emptyset \cup P_2 \cup \{B_2\} \cup \{v : v \in V \setminus \{u\}\}
= P_1 \cup \{B_1\} \cup P_2 \cup \{B_2\} \cup \{v : v \in V\}
\]

Because of the validity of the loop invariant \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \), the last set of this derivation is a solution of the instance \((U, c)\) of the bin packing problem. Hence, the same holds for the first set of the derivation, too, which concludes the proof. 

Since the loop invariant \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \) is established by the initialization part of \( \text{BinPacking} \) and maintained by the body of its while-loop, it is true after the while-loop. Now, \( \text{Inv}_1(P_1, P_2, B_1, B_2, V) \) and the final assignment to \( P \) show that \( P \) is a solution of \((U, c)\).
4. Refinement to a $\frac{3}{2}$-approximation algorithm

Without changing the running time, in the following we present a refinement of the above program BinPacking for which—after termination—also the third condition $|P| \leq \frac{3}{2} \text{opt}(U,c)$ of the original postcondition (2) holds. As already mentioned in the introduction, for a proof of this absolute approximation factor of $\frac{3}{2}$ good lower bounds of $\text{opt}(U,c)$ play a decisive role.

A trivial lower bound of $\text{opt}(U,c)$ is the number of large objects of $U$, since a bin of a solution never contains two large objects. Hence, if we strengthen the hitherto loop invariant $\text{Inv}_1(P_1,P_2,B_1,B_2,V)$ by adding the conjunct

$$V \cap L \neq \emptyset \implies \forall B \in P_1 \cup [B_1] : B \cap L \neq \emptyset,$$

then we get the implication

$$V \cap L \neq \emptyset \implies |P_1 \cup [B_1] \cup \{v \in V \cap L\}| \leq \text{opt}(U,c).$$

Another lower bound follows from the estimation $\sum_{u \in U} w(u) \leq c \text{opt}(U,c)$ which is a consequence of $\sum_{u \in U} w(u) = \sum_{B \in P} w(B) \leq \sum_{B \in P} c = c|P|$ for all solutions $P$ of $(U,c)$. To improve this estimation we add a conjunct to the hitherto loop invariant, viz.

$$\exists f \in O(P_2,B_2)^f : f \text{ bijective} \land \forall B \in P_1 : w(B) + w(f(B)) > c.$$  (7)

In this property $O(P_2,B_2)$ denotes the set $\bigcup_{B \in P_2 \cup [B_2]} B$, i.e., the set of objects contained in the bins of $P_2$ or the set $B_2$. The intention behind property (7) is that a call $f(B)$ yields the unique object whose insertion into $B$ during the execution of BinPackaging would lead to an overfilling of bin $B$. With the help of the refined loop invariant we obtain the estimation

$$|P_1 \cup [B_1]| \leq \text{opt}(U,c)$$

because of the calculation

$$c \text{opt}(U,c) \geq \sum_{u \in U} w(u)$$

$$\geq \sum_{B \in P_1} \sum_{u \in B} w(u) + \sum_{B \in P_2 \cup [B_1]} \sum_{u \in B} w(u)$$

$$= \sum_{B \in P_1} w(B) + \sum_{B \in P_2 \cup [B_2]} w(B)$$

$$= \sum_{B \in P_1} w(B) + w(f(B))$$

$$> \sum_{B \in P_1} c$$

$$= c|P_1|$$

and the estimation $|P_1 \cup [B_1]| \leq |P_1| + 1$, which follows from $|[B_1]| \leq 1$. 

Estimation (8) describes a lower bound of $\text{opt}(U, c)$ in terms of $P_1$ and $B_1$. To obtain a similar result for $P_2$ and $B_2$ we need a further strengthening of the hitherto loop invariant. It consists of the conjunction of

\[ 2|P_2| \leq \left| \bigcup_{B \in P_2} B \right|. \]  

(9)

In words, this estimation says that $P_2$ contains at most half as many bins as objects. Starting with property (9) we can calculate

\[ 2|P_2| \leq \left| \bigcup_{B \in P_2} B \right| \]

\[ \implies 2|P_2| + 2|B_2| \leq \left| \bigcup_{B \in P_2} B \right| + |B_2| + 1 \quad |B_2| \leq 1 \]

\[ \implies 2|P_2| + 2|B_2| \leq \left| \bigcup_{B \in P_2} B \right| + |B_2| + 1 \quad |B_2| \leq |B_2| \]

\[ \implies 2|P_2 \cup B_2| \leq \left| \left( \bigcup_{B \in P_2} B \right) \cup B_2 \right| + 1 \]  

(4)

\[ \iff 2|P_2 \cup B_2| \leq \left| \bigcup_{B \in P_2 \cup B_2} B \right| + 1 \]  

(3)

\[ \implies 2|P_2 \cup B_2| \leq |P_1| + 1 \]  

(7).

From the last estimation of this derivation and the already shown estimation $|P_1| < \text{opt}(U, c)$, finally, we immediately obtain the desired lower bound

\[ |P_2 \cup B_2| \leq \frac{\text{opt}(U, c)}{2}. \]  

(10)

Now, let $\text{Inv}_2(P_1, P_2, B_1, B_2, V)$ denote the refined loop invariant, i.e., the conjunction of the four properties (4), (5), (7), and (9). Then, we are able to show the following result:

**Theorem 4.1.** If $V \cap S = \emptyset$ and the loop invariant $\text{Inv}_2(P_1, P_2, B_1, B_2, V)$ is true, then we have

\[ |P_1 \cup [B_1] \cup P_2 \cup [B_2] \cup \{\{v\} : v \in V\}| \leq \frac{3}{2} \text{opt}(U, c). \]
Proof. We consider two cases. First, we assume \( V = \emptyset \). Then we can prove

\[
|P_1 \cup [B_1] \cup P_2 \cup [B_2] \cup \{v\} : v \in V | \\
= |P_1 \cup [B_1] \cup P_2 \cup [B_2]| \\
= |P_1 \cup [B_1]| + |P_2 \cup [B_2]| \\n\leq \text{opt}(U, c) + \frac{\text{opt}(U, c)}{2} \\
= \frac{3}{2} \text{opt}(U, c).
\]

The remaining case is \( V \neq \emptyset \). Since \( V \cap S = \emptyset \), we obtain \( S = \emptyset \) which in turn implies \( V \cap L \neq \emptyset \). Now, we get the desired estimation by

\[
|P_1 \cup [B_1] \cup P_2 \cup [B_2] \cup \{v\} : v \in V | \\
= |P_1 \cup [B_1] \cup \{v\} : v \in V | + |P_2 \cup [B_2]| \\
\leq \text{opt}(U, c) + \frac{\text{opt}(U, c)}{2} \\
= \frac{3}{2} \text{opt}(U, c). \quad \square
\]

Hence, if we are able to show that the initialization part of BinPacking establishes \( \text{Inv}_2(P_1, P_2, B_1, B_2, V) \) and the body of its while-loop maintains it, then from Theorem 4.1 and the final assignment to \( P \) we get the desired result, viz. that BinPacking is an approximation algorithm for the bin packing problem with an absolute approximation factor of \( \frac{3}{2} \).

The verification of \( \text{Inv}_2(\emptyset, \emptyset, \emptyset, U) \) is trivial. To show the maintenance of the loop invariance \( \text{Inv}_2(P_1, P_2, B_1, B_2, V) \), however, we have to refine the selection \( u : \in V \) in program BinPacking as follows:

\[
\text{if } B_1 \neq \emptyset \text{ then } u : \in V \cap S \\
\text{else if } V \cap L \neq \emptyset \text{ then } u : \in V \cap L \\
\text{else } u : \in V \cap S \text{ fi fi;} \\
\]

This conditional statement means that bin \( B_1 \) is opened with a large object, but, of course, only if this is possible. Otherwise bin \( B_1 \) is opened with a small object. After having opened bin \( B_1 \) only small objects are inserted. In addition to the refinement of the choice of \( u \), we have to strengthen the hitherto loop invariant a last time. We refine \( \text{Inv}_2(P_1, P_2, B_1, B_2, V) \) by adding the conjunct

\[
B_2 \subseteq S. \\
\]
This property is motivated by the refined selection (11) of the element \( u \) and the fact that in \( B_2 \) those objects are collected whose insertion would lead to an overflowing of \( B_1 \).

Let \( \text{Inv}_3(P_1, P_2, B_1, B_2, V) \) abbreviate the final loop invariant, i.e., the conjunction of (4), (5), (7), (9), and (12). Then the verification of \( \text{Inv}_3(\emptyset, \emptyset, \emptyset, \emptyset, U) \) being true is trivial. I.e., we have:

**Theorem 4.2.** The loop invariant \( \text{Inv}_3(P_1, P_2, B_1, B_2, V) \) is established by the initialization of the variables of \textsc{BinPacking}.

Wrt maintenance of the final loop invariant, the remaining proof obligation, we have the following result:

**Theorem 4.3.** The loop invariant \( \text{Inv}_3(P_1, P_2, B_1, B_2, V) \) is maintained by the body of the while-loop of \textsc{BinPacking} if the selection \( u : \in V \) is replaced by the nested conditional (11).

**Proof.** Assume \( V \cap S \neq \emptyset \) and \( \text{Inv}_3(P_1, P_2, B_1, B_2, V) \) to be true. Furthermore, let \( u \) be selected by (11). Since this nested conditional is defined in each case, it refines the original selection \( u : \in V \) and, therefore, property (4) is maintained due to Theorem 3.2.

It remains to show the validity of (5), (7), (9), and (12) for the new values of the variables \( P_1, P_2, B_1, B_2, \) and \( V \), provided the respective guards of the nested conditional are true.

We start with the maintenance proof of property (5) and have to consider two cases. If \( w(B_1) + w(u) \leq c \), then we get the desired implication by

\[
(V\setminus\{u\}) \cap L \neq \emptyset \implies V \cap L \neq \emptyset
\]

\[
\implies \forall B \in P_1 \cup B_1 : B \cap L \neq \emptyset \quad (5)
\]

\[
\implies \forall B \in P_1 \cup [B_1 \cup \{u\}] : B \cap L \neq \emptyset.
\]

The second case is \( w(B_1) + w(u) > c \). Here we obtain \( B_1 \neq \emptyset \) since \( w(u) \leq c \) holds because of the first formula of precondition (1). Hence, (11) has selected the element \( u \) from \( V \cap S \). From this fact we get

\[
(V\setminus\{u\}) \cap L \neq \emptyset \iff V \cap L \neq \emptyset \quad u \in V \cap S
\]

\[
\implies \forall B \in P_1 \cup B_1 : B \cap L \neq \emptyset \quad (5)
\]

\[
\iff \forall B \in P_1 \cup [B_1 \cup \emptyset] : B \cap L \neq \emptyset
\]

\[
\iff \forall B \in P_1 \cup [B_1 \cup \emptyset] : B \cap L \neq \emptyset \quad (3).
\]

Putting both cases together we obtain the desired maintenance of (5).

Next, we deal with property (7). By assumption, there exists a bijective function \( f : P_1 \rightarrow \bigcup_{B \in P_1 \cup B_2} B \) such that the estimation \( w(B) + w(f(B)) > c \) holds for all \( B \in P_1 \).
Since the values of \( P_1, P_2, \) and \( B_2 \) only change if \( w(B_1) + w(u) > c \), we are allowed to suppose this estimation for the following proof. Having the intention of the function \( f \) in mind and looking to the algorithm we update \( f \) by the function

\[
F(B) = \begin{cases} 
  f(B), & B \neq B_1, \\
  u, & B = B_1.
\end{cases}
\tag{13}
\]

Due to (4) we obtain \( B_1 \notin P_1 \). Property (4) also shows that the object \( u \) from \( V \) is not contained in a bin of \( P_2 \) nor in \( B_2 \). Hence, (13) defines a bijective function \( F \) from \( P_1 \cup \{B_1\} \) to the objects of \( P_2 \cup \{B_2 \cup \{u\}\} \) as well as from \( P_1 \cup \{B_1\} \) to the objects of \( P_2 \cup \{B_2 \cup \{u\}\} \). We still have to show the estimation demanded in (7). Therefore, let \( B \in P_1 \cup \{B_1\} \). If \( B \neq B_1 \), then we can show

\[
w(B) + w(F(B)) = w(B) + w(f(B)) > c;
\]

in the remaining case \( B = B_1 \) we get

\[
w(B_1) + w(F(B_1)) = w(B_1) + w(u) > c
\]

by the supposed estimation and we are done.

To show the maintenance of property (9) we are allowed to restrict ourselves to the case that \( w(B_1) + w(u) > c \) and \( w(B_2) + w(u) > c \) hold since otherwise the value of \( P_2 \) does not change. As shown above, then statement (11) has selected \( u \) from \( V \cap S \). Now, we combine this fact with \( w(B_2) + w(u) > c \) and (12) and obtain that \( B_2 \) contains at least two elements. Using this property we can prove (9) for the new value of \( P_2 \) by

\[
2|P_2 \cup B_2| \leq 2|P_2| + 2 \quad \|B_2\| = 1
\]

\[
\leq \left| \bigcup_{B \in P_2} B \right| + 2 \quad (9)
\]

\[
= \left| \bigcup_{B \in P_2} B \right| + |B_2| \quad 2 \leq |B_2|
\]

\[
= \left| \bigcup_{B \in P_2 \cup \{B_2\}} B \right| \quad (3), (4).
\]

The last property we have to treat is (12). Since the value of \( B_2 \) only changes if \( w(B_1) + w(u) > c \), we are allowed to assume this estimation for proving the maintenance of (12). As a consequence, the object \( u \) has again been selected from \( V \cap S \). This shows \( \{u\} \subseteq S \) and, in combination with (12), also \( B_2 \cup \{u\} \subseteq S \). Hence both statements of the inner conditional maintain property (12).

This ends the correctness proof of the refined approximation algorithm wrt the precondition \( \text{Pre}(U, w, c, S, L) \) and the entire postcondition \( \text{Post}(P) \).
5. The final linear time version of the algorithm

In this section we apply some further program development steps to the hitherto approximation algorithm to obtain a version which has besides the absolute approximation factor of $\frac{3}{2}$ also a linear running time and which, furthermore, immediately can be transferred into a conventional programming language like Pascal, Modula, C, or Java.

The first step consists in the removal of the parameters $S$ and $L$. Instead the sets of small, respectively, large elements of $U$ are computed via three variables $S$, $L$, and $W$ and the following piece of code:

$$
S := \emptyset; L := \emptyset; W := U; \\
\textbf{while } W \neq \emptyset \textbf{ do } \\
\hspace{1em} u := W; \\
\hspace{2em} \textbf{if } w(u) \leq \frac{c}{2} \textbf{ then } S := S \cup \{u\} \\
\hspace{3em} \textbf{else } L := L \cup \{u\} \; \text{fi}; \\
W := W \setminus \{u\} \; \text{od;} \\
$$

The correctness of this computation, which, after the removal of the parameters $S$ and $L$, has to be inserted in front of the initialization of $P_1, P_2, B_1, B_2$, and $V$, easily can be shown using the conjunction of the equations

$$
S = \left\{ v \in U \setminus W : w(v) \leq \frac{c}{2} \right\}, \quad L = \left\{ v \in U \setminus W : w(v) > \frac{c}{2} \right\}
$$

as loop invariant.

The introduction of the local variables $S$ and $L$ and the computation of the small and large elements via (14) allows a further simplification using a technique similar to formal differentiation. After the computation of $S$ and $L$ we change their values in the following while-loop (i.e., the while-loop of the program we have obtained in Section 4) according to the selection of $u$. This means that we refine the nested conditional (11) as follows:

$$
\textbf{if } B_1 \neq \emptyset \textbf{ then } u : V \cap S; S := S \setminus \{u\} \\
\hspace{1em} \textbf{else if } V \cap L \neq \emptyset \textbf{ then } u : V \cap L; L := L \setminus \{u\} \\
\hspace{2em} \textbf{else } u : V \cap S; S := S \setminus \{u\} \; \text{fi fi;}
$$

If we replace (11) by (16), then, obviously, the equations

$$
S = \left\{ u \in V : w(u) \leq \frac{c}{2} \right\}, \quad L = \left\{ u \in V : w(u) > \frac{c}{2} \right\}
$$

which are established by the execution of (14) and the subsequent assignment $V := U$, are maintained by the body of the new while-loop. Due to this invariance property of (17), hence, each occurrence of $V \cap S$ can be replaced by $S$, each occurrence of $V \cap L$
can be replaced by \( L \), and \( \{ v : v \in V \} \) can be replaced by \( \{ v : v \in L \} \). In doing so, \( V \) becomes superfluous and may be removed.

The last step of our program development consists in the computation of the union of \( P_1 \cup [B_1] \cup P_2 \cup [B_2] \) and \( \{ v : v \in L \} \) via a simple while-loop. This, finally, leads to the following program BinPackingApprox which is correct wrt the first formula of (1) as precondition and the original postcondition Post\( (P) \), i.e., is an approximation algorithm for the bin packing problem with \( \frac{3}{2} \) as absolute approximation factor.

**BinPackingApprox\( (U,w,c) \)**

\[
\begin{align*}
S &:= \emptyset; L := \emptyset; W := U; \\
\text{while } W \neq \emptyset \text{ do} \\
& \quad \begin{cases} 
    \text{if } w(u) \leq \frac{c}{2} & \text{then } S := S \cup \{u\} \\
    \text{else } L := L \cup \{u\} & \text{fi;}
\end{cases} \\
& \quad W := W \setminus \{u\} \\
P_1 := \emptyset; P_2 := \emptyset; B_1 := \emptyset; B_2 := \emptyset; \\
\text{while } S \neq \emptyset \text{ do} \\
& \quad \begin{cases} 
    \text{if } B_1 \neq \emptyset & \text{then } u \in S; S := S \setminus \{u\} \\
    \text{else if } L \neq \emptyset & \text{then } u \in L; L := L \setminus \{u\} \\
    \text{else } u \in S; S := S \setminus \{u\} & \text{fi;}
\end{cases} \\
& \quad \begin{cases} 
    \text{if } w(B_1) + w(u) \leq c & \text{then } B_1 := B_1 \cup \{u\} \\
    \text{else if } w(B_2) + w(u) \leq c & \text{then } B_2 := B_2 \cup \{u\} \\
    \text{else } P_2 := P_2 \cup [B_2]; B_2 := \{u\} & \text{fi;}
\end{cases} \\
& \quad P_1 := P_1 \cup [B_1]; B_1 := \emptyset \\
P := P_1 \cup [B_1] \cup P_2 \cup [B_2]; W := L; \\
\text{while } W \neq \emptyset \text{ do} \\
& \quad u \in W; P := P \cup \{u\}; W := W \setminus \{u\} \\
\text{return } P
\end{align*}
\]

Obviously, the running time of this program is linear if a datatype for sets is used which allows to perform the insertion, removal, and selection of an element, the union of two sets, and the emptiness test in constant time.

### 6. Concluding remarks

In this paper we have presented a new approximation algorithm for the bin packing problem which has optimal running time and, unless \( P = NP \), also optimal absolute approximation factor. To show its correctness we have used techniques of formal program specification, verification, and development.
It should be mentioned that in [9] a polynomial approximation schema for bin packing is presented which follows the asymptotic approach mentioned in the introduction. This schema computes a bin packing which uses at most \((1 + \varepsilon)|P^*| + O(1/\varepsilon)\) bins, where \(P^*\) is again an optimal solution and \(\varepsilon\) is a fixed positive number. The running time of the algorithm of [9] is linear in the number of objects, but grows exponentially in \(1/\varepsilon\). Ignoring the additive constant, for a fixed \(\varepsilon = 1/2\), hence, one obtains a linear algorithm with asymptotic approximation factor \(\frac{3}{2}\). If we compare it with our algorithm, then two facts speak well for the latter. First, it does not have an additive constant in the approximation bound. This means that it performs reasonable well even when the number of bins is small. Many experiments with implementations of BINPACKINGAPPROX in the programming languages ML and Java have confirmed this behaviour. And, second, it is much more simple than the algorithm of [9]. Hence, the constant factor in its linear running time is much smaller than the constant factor in the running time of the algorithm of [9].

Let us close with a few remarks on a specific point of our formal approach. When verifying an approximation algorithm one has to show two things, viz. the feasibility of the computed result and after that an estimation for its maximal deviation from an optimal solution. Proofs of worst-case bounds frequently refer to entities that are not present in the program code. In our case we have used a bijective function which maps each bin \(B\) of a partial solution to the object whose insertion into \(B\) would lead to an overfilling. To give another example, in the proof of the approximation factor of the well-known approximation algorithm of Gavril and Yannakakis for minimum vertex cover (see e.g. [3]) the matching formed by the edges that were selected during the execution of the algorithm plays a decisive role. Using the invariant technique, a just mentioned entity \(e\) leads to an existential quantification \(\exists e \ldots\) in loop invariants of a program \(P\). But quantifications are sometimes hard to manage, especially if the verification is supported by computer systems. Due to the experiences gained so far, e.g., in the course of [7], we believe that formal proofs of approximation factors become easier and more direct if an augmentation \(P'\) of \(P\) is considered which additionally contains \(e\) in form of a variable and some assignments that realize its variation. In particular this makes the design of loop invariants more reliant on “eurekas”. The newly introduced variable \(e\) is used at most in assignments to itself but neither in assignments to other variables nor in conditions. Hence, it is an auxiliary variable in the sense of [1]. In [1] it is also shown that all assignments to auxiliary variables may be deleted without changing correctness. As a consequence, if \(P'\) guarantees a certain maximal deviation from an optimal solution, the same holds for \(P\), the program we are actually interested in.

References