Computers Ops Res. Vol. 21, No. 1, pp. 19–25, 1994
Printed in Great Britain.

0305-0548/94 $6.00+0.00

A BRANCH-AND-BOUND ALGORITHM FOR THE TWO-DIMENSIONAL VECTOR PACKING PROBLEM

FRITS C. R. SPIEKSMA†
Department of Mathematics, University of Limburg, P.O. Box 616, 6200 MD, Maastricht, The Netherlands

(Received March 1992; in revised form December 1992)

Scope and Purpose—Bin-packing problems arise when a number of non-overlapping objects have to be packed into a number of so-called bins, where a bin usually represents a fixed amount of space or time. Applications of bin-packing problems are quite diverse, and can be found in, for example, scheduling theory, VLSI-design, computer network design and in the field of cutting-stock problems. In this paper, a bin-packing problem is studied where each object has two requirements, and the objective is to find the minimum number of unit-capacity bins. In geometric terms: given a set of rectangles (where the length and width of each rectangle represent the two requirements (or dimensions) of an object), pack them in as few unit squares (the bins) as possible, such that the rectangles are placed corner to corner, in a diagonal fashion.

The main contributions of this paper are the description of a new heuristic for this problem, and the presentation of two types of lower bounds. These bounds are used by a branch-and-bound algorithm, whose performance is tested on randomly generated instances.

Abstract—The two-dimensional vector packing (2DVP) problem can be stated as follows. Given are N objects, each of which has two requirements. The problem is to find the minimum number of bins needed to pack all objects, where the capacity of each bin equals 1 in both requirements. A heuristic adapted from the first fit decreasing rule is proposed, and lower bounds for optimal solutions to the 2DVP problem are investigated. Computing one of these lower bounds is shown to be equivalent to computing the largest number of vertices of a clique of a 2-threshold graph (which can be done in polynomial time). These lower bounds are incorporated into a branch-and-bound algorithm, for which some limited computational experiments are reported.

1. INTRODUCTION

The well-known bin-packing problem may be described as follows: given N objects, each with a value v_i, $i=1, \ldots, N$, pack these objects in as few bins as possible such that the sum of the v_is of objects packed in the same bin does not exceed the bin's capacity. There exists a wealth of literature on this subject; see for instance Martello and Toth [1] for further references.

In this paper a natural extension of the bin-packing problem is studied: again, N objects are to be distributed over a number of bins, however, now two values (or requirements), denoted by v_i^1 and v_i^2, $i=1, \ldots, N$, are associated with each object. Both the sum of the v_i^1s *and* the sum of the v_i^2s of objects assigned to the same bin must be less than 1 (the standardized bin's capacity). The problem is how to assign the objects to a minimum number of bins, while satisfying these constraints. In the terminology of Coffman *et al.* [2] this problem is called the two-dimensional vector-packing (2DVP) problem.

An interesting application of the 2DVP problem is mentioned by Vercruyssen and Muller [3]. The application arises in a factory where coils of steel plates (the objects), each having a certain physical weight (v_i^1) and a certain height (v_i^2), have to be distributed over identical furnaces (the bins) with a limited capacity for height and weight. Another application of the problem is described by Sarin and Wilhelm [4], in the context of layout design. Here, a number of machines (the objects) have to be assigned to a number of robots (the bins), with each robot having a limited capacity for space, as well as a limited capacity for serving a machine.

Clearly, the 2DVP problem is $\mathcal{NP}$-hard, since the classical one-dimensional bin-packing problem is a special case of the 2DVP problem: by letting $v_i^2=0$ for $i=1, \ldots, N$, the bin-packing problem

† F. C. R. Spieksma is an Assistant Professor in the Department of Mathematics of the University of Limburg, The Netherlands. He received a Ph.D. in Operations Research from the University of Limburg. His research interests include combinatorial optimization problems in automated manufacturing.

arises (for an introduction to complexity theory, see for instance Garey and Johnson [5]).

Much research effort in bin-packing has concentrated on approximation algorithms, which, although not guaranteeing an optimal solution, deliver near-optimal solutions (for an excellent survey on this subject the reader is referred to Coffman *et al.* [2]). However, except for an algorithm given by Martello and Toth [1], little can be found in the literature on exact algorithms for the bin-packing problem. Here, a branch-and-bound method for the 2DVP problem is described. Upper bounds are derived from a heuristic, adapted from the first fit decreasing (*FFD*)-rule for the bin-packing problem (see Section 2). To find good lower bounds, properties of pairs of objects are investigated. A detailed description is given in Section 3. In Section 4 the branch-and-bound algorithm is described, Section 5 discusses results from conducted experiments and Section 6 contains the conclusion.

2. HEURISTICS

The first-fit (*FF*) rule (see [6]) for the 2DVP problem may be described as follows: given a list L of objects, start at the top and assign the objects consecutively to the bin with minimal index while satisfying both capacity restrictions. The resulting number of bins needed is denoted by $FF(L)$. By constructing the list L in a clever manner, it is possible to improve the performance of the heuristic. Here, the following way to construct a list is proposed (see also [7]). Every object i is given a priority based on its two requirements:

$$\text{priority}_i = \lambda \cdot v_i^1 + v_i^2 \quad \text{for } i = 1, \ldots, N,$$

where λ is a parameter indicating a relative weight for the first requirement. Now, a list $L(\lambda)$ is constructed by arranging the objects in decreasing order of their priority. When applying *FF* to this list, it turns out that some bins are well-filled, that is in both requirements the remaining capacity is close to 0, whereas other bins have relatively more free capacity. Moreover, different choices for λ lead to different well-filled bins. So, the following new heuristic, denoted by *FFD2* is proposed.

Step 0. $k = 1$, welfilbin $= 0$, $sol = N$

Step 1. specify λ_k, construct list $L(\lambda_k)$ of remaining objects and apply *FF* to $L(\lambda_k)$. If welfilbin $+ FF(L(\lambda_k)) < sol$ then $sol :=$ welfilbin $+ FF(L(\lambda_k))$

Step 2. check for every bin: if it is filled well, delete from the problem instance all the objects contained in the bin and welfilbin: = welfilbin + 1, if no bin is filled well then stop (with solution: *sol*), else $k := k+1$ and go to Step 1

where welfilbin denotes the number of well-filled bins and *sol* denotes the number of bins needed by *FFD2*.

This heuristic turns out to perform well, however, two factors are not specified yet:

(1) What is the criterion for a bin to be considered as a well-filled bin?
(2) How to determine the value of λ in every iteration?

Regarding the first question, we consider a bin r to be filled well when

$$\sum_{i \in B_r} v_i^1 \geqslant \sum_{i \in ND} v_i^1 \Big/ \left[\sum_{i \in ND} v_i^1 \right] \quad \text{and} \quad \sum_{i \in B_r} v_i^2 \geqslant \sum_{i \in ND} v_i^2 \Big/ \left[\sum_{i \in ND} v_i^2 \right]$$

where

$B_r = \{i$: object i is in bin $r\}$
$ND = \{i$: object i is not deleted from the problem$\}$ and
$[x]$ denotes the smallest integer greater than or equal to x.

This criterion is intuitively speaking, based on the idea that a bin is filled well when it is filled more than average in both requirements. (Note that this criterion may change in every iteration.)

As to the second question, an idea suggested in Maruyama *et al.* [7] is used. They propose to compute a weight for the first requirement in the following way. Solve the 2DVP problem by the next-fit rule, i.e. given a list of objects, start at the top and assign them consecutively to a bin.

When the current object does not fit in the bin, start a new bin. Now, let p_1 (p_2) denote the number of times a new bin has to be started due to excess in the first (second) requirement. Since p_1/p_2 can be considered as a measure for the importance of the first requirement, λ_k is set equal to p_1/p_2 (if $p_2=0$, λ_k is set equal to a large number). This completes the description of *FFD2*.

A first-fit heuristic for the 2DVP problem proposed in Garey *et al.* [8] was also implemented: construct the list L in such a way that the priorities of the objects are in decreasing order, where

$$\text{priority}_i = \max(v_i^1, v_i^2).$$

The running time of this heuristic, in the sequel denoted by *FFGAR*, is $O(N \log N)$, whereas the complexity of *FFD2* is $O(N^2 \log N)$.

3. LOWER BOUNDS

In this section two types of lower bounds are investigated. It turns out that each of these bounds is suited for a different class of problem instances. Of course, an obvious lower bound is the following one (see [1] for a similar bound for the one-dimensional bin-packing problem):

$$lb1 = \max\left(\left\lceil \sum_{i=1}^{N} v_i^1 \right\rceil, \left\lceil \sum_{i=1}^{N} v_i^2 \right\rceil\right).$$

When the values of the objects are relatively small, this lower bound seems to be appropriate.

Another lower bound is based on considering pairs of objects. An important characteristic of such a pair is whether they can be assigned to the same bin or not. Now, the idea is to find the maximum number of objects for which it is known that no two of these objects can be assigned to the same bin. (Obviously, this number is a lower bound for the optimal solution.) This idea can be interpreted in graph-theoretic terms: construct a graph $F=(V, E)$ where each vertex $i \in V$ represents an object and $(i, j) \in E$ if and only if $v_i^1 + v_j^1 > 1$ or $v_i^2 + v_j^2 > 1$ (or both). Computing the described lowerbound is equivalent to finding a clique in F with the largest number of vertices. Let this number be denoted by $\omega(F)$. Notice that computing $\omega(F)$ for arbitrary graphs is $\mathcal{NP}$-hard. However, the special structure of F (F is the edge-union of two threshold graphs, also called a 2-threshold graph) implies that $\omega(F)$ can be found in polynomial time by the following algorithm [9].

Define $N(i)=\{j \in V\backslash\{i\}: v_i^1 + v_j^1 > 1 \text{ or } v_i^2 + v_j^2 > 1\}$.

Procedure maxclique

Begin

```
l:=0
while V≠∅ do
begin
    l:=l+1;
    pick i ∈ V with v_i^1 = max{v_k^1: k ∈ V};
    S_l:=V\N(i);
    pick j ∈ S_l\{i} with v_j^2 = max{v_k^2: k ∈ S_l\{i}};
    V:=N(i)∪N(j);
end
lb2:=l;
```

End.

In each iteration of the procedure maxclique a set of vertices $V' \subset V$, is identified. The vertices of V' are connected to either a vertex i (with $v_i^1 = \max_{k \in V} v_k^1$) or to a vertex j (with $v_j^2 = \max_{k \in S} v_k^2$, with $S = V\backslash N(i) \cup \{i\}$). Hence, in the terminology of the 2DVP problem, there exists at least one object (assuming $V' \neq \emptyset$) which does not fit with object i or object j in one bin. This implies that at least one other bin is needed, so the lowerbound is increased by one, and a new iteration starts. For a formal proof of correctness, see Hammer and Mahadev [9]. Since each iteration of the procedure reduces V with at least one element the procedure has complexity $O(N^2)$.

This lowerbound is suited for problem instances where several objects have high values for v_i^1 and/or v_i^2, $1 \leqslant i \leqslant N$. Evidently, the overall lowerbound lb is equal to $lb := \max(lb1, lb2)$.

4. A BRANCH-AND-BOUND ALGORITHM

In this section a branch-and-bound algorithm is presented that finds an optimal solution for the 2DVP problem. The main idea of the algorithm is to find a so-called sequential maximal solution (*sms*) (see also [10]). A sequential maximal solution is a sequence $B_1, B_2, \ldots$ of bins (or more precisely of subsets of objects, each subset fitting in a bin) for which every bin B_r, $r = 1, 2, \ldots$ is maximal. A bin B_r is called maximal if no object l, $l \in V \setminus \bigcup_{i=1}^{r} B_i$ can be added to B_r, where V denotes the set of all objects.

The following proposition implies that for every optimal solution there exists a corresponding *sms* (which is also optimal).

Proposition 1. If there exists an optimal solution using k bins, then there exists an *sms* using k bins.

Proof. Let $B_1, \ldots, B_k$ be an optimal solution. Let i be the smallest index such that B_i is not maximal. Then there exists an object $j \in B_l$ $(l > i)$ which can be added to B_i, without increasing the number of used bins. Repeating this process until all bins are maximal results in an *sms* consisting of k bins. □

Hence, restricting the search space to *sms*s does not exclude all optimal solutions.

The branch-and-bound algorithm can be described as follows. It checks whether there exists a solution using k bins, where k denotes the value of the current solution minus one. If there is not, the current best solution is optimal, else the value of the current solution is decreased by one and the algorithm starts again. Now, at an arbitrary moment in the execution of the algorithm the following situation occurs: there is a list L of not yet assigned objects (let $i(L)$ denote the first object of L), and there are, say r, earlier generated maximal bins. To proceed, let us define the set of all maximal bins with $i(L)$ and solely objects from L in it as $MB(L)$. (So, any bin from $MB(L)$ has object $i(L)$ in it). From this set a bin is generated. Then two tests are performed:

(1) a dominance test (which will be discussed later), and
(2) a lowerbound test. Here, it is tested whether the lower bound (as described in Section 3) for the remaining objects in L is smaller or equal to $k-(r+1)$ (i.e. whether these objects could fit in the remaining $k-(r+1)$ bins).

If both tests succeed then list L is adapted (more precisely, the elements in the new bin are deleted from L), r is increased by one and the process starts again with the list of remaining objects. If one of the tests fails, another maximal bin from the set $MB(L)$ is generated for which both tests are performed, and so on.

Of course, an important feature of the algorithm is how it generates new maximal bins. This is done as follows. Suppose a maximal bin from $MB(L)$ does not pass one of the tests. Then another maximal bin from $MB(L)$ is found by first eliminating that object from the maximal bin that is the last one in the list L (say object j_1). Next, the list L is scanned, starting with the object following j_1. If an object can be added to the current bin, this is done. When indeed one or more objects are added, a new maximal bin is found. Otherwise, again the current object in the bin which appears last in the list L is eliminated from the bin and the same procedure is repeated. This process continues until object $i(L)$ is eliminated from the bin. Then all maximal bins from the set $MB(L)$ are investigated.

It remains to specify how the dominance test is applied. This test relies on the following observation: if there does not exist a solution using k bins with a maximal bin Q_1, from set $MB(L)$ for a certain L, then there does not exist a solution using k bins with maximal bin Q_2 from the set $MB(L)$ if for each object $j \in Q_2$ there exists a unique $j' \in Q_1$ with $v_j^1 \leqslant v_{j'}^1$ and $v_j^2 \leqslant v_{j'}^2$. In other words Q_1 dominates Q_2.

Now, to use this dominance test, it is necessary to keep track of discarded bins. Each time a bin is discarded at level $r+1$ (i.e. there are r already generated maximal bins present) this bin is stored in the set called DO_{r+1} (except for those bins who are discarded because of the dominance test, of

course!). When a new bin at this level is generated, it is checked whether a bin from DO_{r+1} dominates the current bin. Of course, when a bin at level r is generated all sets DO_s with $s>r$ are empty.

For a formal description, define

sol : value of current best solution
lb : lowerbound (see Section 3)
DO_r : set of discarbed bins at level r
L : list of objects, ordered according to the solution found by *FFD2*
$i(L)$: first element in the list L
$MB(L)$: the set of all maximal bins with $i(L)$ and solely objects from L in it
$lb(L)$: lower bound for the objects in the list L (see Section 3).

Step 0. Initialization : $DO_s=\varnothing\ \forall s=1,\ldots,sol-1$,
$r=1$, $k=sol-1$.

Step 1. Branching : pick a $B_r \in MB(L)$ (See preceding text)
$MB(L):=MB(L)\setminus\{B_r\}$. $L:=L\setminus B_r$
if $L=\varnothing$ go to Step 4.

Step 2. Bounding test : if $(lb(L)+r\leqslant k)$ and
(no bin from DO_r dominates B_r) then
begin
$DO_r:=DO_r\cup\{B_r\}$
$r:=r+1$
go to Step 1
end.

Step 3. Bounding : $L:=L\cup B_r$.
if $(MB(L)=\varnothing)$ and $(r=1)$ then
stop, sol is optimal
if $(MB(L)=\varnothing)$ and $(r>1)$ then
$DO_r:=\varnothing$, $r:=r-1$, go to Step 3
if $MB(L)\neq\varnothing$ then
go to Step 1.

Step 4. Termination : if $r=lb$, stop: r is optimal
else $sol:=r$ and go to Step 0.

5. COMPUTATIONAL RESULTS

Three types of problems were generated: for one type of problems v_i^1 and v_i^2, $i=1, 2, \ldots, N$, were drawn uniformly and independently from the interval [0.1, 0.4], for the second type of problems the interval [0, 1] was used, and for the third type of problems the interval [0.2, 0.8] was used. All numbers were generated such that they had three digits behind the decimal point. For each of these types four problems with 25, four problems with 35, four problems with 50 and four problems with 100 objects were generated. All problems were solved on a personal computer with a 486 processor.

The results can be seen in Tables 1–3.

Computation times of the heuristics *FFD2* and *FFGAR* were all within 0.5 s (with *FFGAR* being a bit faster). As can be seen from the tables, their solutions differ nine times, six times in favour of *FFD2*, three times in favour of *FFGAR*.

The results from the tables also show that problems with up to 35 objects are solved by the branch-and-bound algorithm with limited computation times. From the problems with 50 or 100 objects, four Type I problems and one Type II problem caused difficulties. The problems of Type I seem to be somewhat more difficult than those of Types II or III, which is probably due to the larger amount of possible maximal bins, caused by the on average smaller values of v_i^1 and v_i^2.

Table 1. $v_i^1, v_i^2 \sim U$ [0.1, 0.4] for $i = 1, \ldots, N$

N	*lb*1	*lb*2	*FFD*2	*FFGAR*	*b* and *b*	Time
25	7	1	7	8	NN	
25	7	1	7	7	NN	
25	7	1	7	7	NN	
25	7	1	7	7	NN	
35	10	1	10	10	NN	
35	9	1	9	10	NN	
35	10	1	10	10	NN	
35	10	1	11	11	10	1 s
50	13	1	14	14	14	6 h, 31 min, 36 s
50	13	1	14	14	13	1 s
50	14	1	14	14	NN	
50	14	1	16	15	14	5 min, 5 s
100	25	1	26	27	25	4 h, 34 min, 57 s
100	24	1	27	27	24	26:1 s, 25:1 s, 24 > 10 h
100	26	1	27	28	26	5 s
100	26	1	27	28	27	7 h, 8 min, 18 s

NN means not necessary.

Table 2. $v_i^1, v_i^2 \sim U$ [0, 1] for $i = 1, \ldots, N$

N	*lb*1	*lb*2	*FFD*2	*FFGAR*	*b* and *b*	Time
25	14	16	16	16	NN	
25	13	16	16	16	NN	
25	14	15	15	15	NN	
25	14	15	15	15	NN	
35	20	22	22	22	NN	
35	19	20	21	21	21	20 s
35	17	17	18	18	18	1 s
35	18	20	21	21	21	18 s
50	28	30	31	31	31	5 s
50	24	24	26	27	26	> 10 h
50	33	40	40	40	NN	
50	31	33	33	33	NN	
100	60	66	67	66	NN	
100	49	57	57	57	NN	
100	51	57	57	57	NN	
100	58	67	67	67	NN	

NN means not necessary.

Table 3. $v_i^1, v_i^2 \sim U$ [0.2, 0.8] for $i = 1, \ldots, N$

N	*lb*1	*lb*2	*FFD*2	*FFGAR*	*b* and *b*	Time
25	15	15	15	15	NN	
25	15	18	18	18	NN	
25	14	16	16	16	NN	
25	14	19	19	19	NN	
35	24	29	29	29	NN	
35	20	24	24	24	NN	
35	21	23	23	23	NN	
35	21	24	24	24	NN	
50	25	29	30	30	30	6 min, 9 s
50	28	34	35	35	35	15 s
50	27	29	29	29	NN	
50	29	32	33	33	33	6 s
100	53	62	62	62	NN	
100	50	54	54	54	NN	
100	54	59	60	59	NN	
100	52	65	65	65	NN	

NN means not necessary.

Another possible explanation for this phenomenon is the good performance of *lb*2 for problems of Types II and III.

As a final remark, the real-world problem with 100 objects described in Vercruyssen and Muller [3] (a Type I problem) was solved to optimality by *FFD*2.

6. CONCLUSIONS

This paper deals with the two-dimensional vector packing (2DVP) problem. An iterative first-fit algorithm is presented where the objects are ordered based on a weighted sum of their two requirements. Also, a lowerbound (*lb*2) is presented which is based on identifying a set of objects no two of which fit together in one bin. It is shown that finding the largest set of objects with this property is equivalent to computing the largest clique in a graph whose structure allows a polynomial algorithm to solve this problem. Finally, a branch-and-bound algorithm is developed which incorporates these lower bounding procedures. Computational experiments indicate that the algorithm solves problems with up to 35 objects in reasonable computing times. Also, the experiments show that the performance of *lb*2 for problem instances where objects may have high values for v_i^1 and/or v_i^2, $i=1, \ldots, N$ is quite satisfactory.

Acknowledgements—The author wishes to thank Yves Crama for pointing out the algorithm for *lb*2 and Koos Vrieze and Yves Crama for helpful discussions on the subject. Also, the comments of the referees are greatly appreciated.

REFERENCES

1. S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester (1990).
2. E. G. Coffman Jr, M. R. Garey and D. S. Johnson, Approximation algorithms for bin-packing—an updated survey. In *Algorithm Design for Computer System Design* (Edited by G. Ausiello, M. Lucertini and P. Serafini), pp. 49–106. Springer, Vienna (1984).
3. D. Vercruyssen and H. Muller, Simulation in production. A report of the University of Gent, Belgium (1987).
4. S. C. Sarin and W. E. Wilhelm, Prototype models for two-dimensional layout design of robot systems. *IIE Trans.* **16**, 206–215 (1984).
5. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York (1979).
6. D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey and R. L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.* **33**, 299–325 (1974).
7. K. Maruyama, S. K. Chang and D. T. Tang, A general packing algorithm for multidimensional resource requirements. *Int. J. comput. inform. Sci.* **6**, 131–149 (1977)
8. M. R. Garey, R. L. Graham and D. S. Johnson, Resource constrained scheduling as generalized bin packing. *J. combinat. Theory* (*A*) **21**, 257–298 (1976).
9. P. L. Hammer and N. V. R. Mahadev, Bithreshold graphs. *SIAM J. algebr. disc. Meth.* **6**, 497–506 (1985).
10. C. S. Tang and E. V. Denardo, Models arising from a flexible manufacturing machine, part II: minimization of the number of switching instants. *Ops Res.* **36**, 778–784 (1988).