NEW ALGORITHMS FOR THE BIN PACKING PROBLEM

by

ALİ TAMER ÜNAL

B.S. in I.E. Boğaziçi University, 1986

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering

Boğaziçi University

1988

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Doç.Dr. Gündüz Ulusoy for his invaluable guidance as the supervisor of this thesis and especially for his support, encouragement and understanding throughout all phases of this study.

I also sincerely wish to thank Doç.Dr Ilhan Or and Yard.Doç.Dr. Vahan Kalenderoğlu for their comments and suggestions .

I also would like to thank my friends Oktay Günlük and Yavuz Sakallı for their merciless counter examples in the early phases of this thesis.

ABSTRACT

Bin Packing is a well-known NP-complete problem which has many real-life applications. In this study, an extensive literature survey is followed by a number of new heuristic and optimal algorithms developed using a new general procedure called " Similar Tree Search Algorithm " to solve zero-one integer programming problems.

Besides, a new special case of the bin packing problem, smooth packing, is defined and algorithms to solve this new problem are generated and tested.

ÖZET

Tek Boyutlu Yerleştirme problemi geniş bir uygulama alanı olan tanınmış bir NP-kapsar problemdir. Bu çalışmada, bir literatür taraması yapılmış ve problemin NP-kapsar olmasından dolayı öncelikle bazı sezgisel algoritmalar üzerinde durulmuş ve bunun yanı sıra sıfır-bir tam sayı problemleri çözebilen bir prosedür kullanılarak, iyi çözülebilir ve en iyi çözümü bulan bir algoritma geliştirilmeye çalışılmıştır.

Ayrıca, Tek Boyutlu Yerleştirme probleminin yeni bir özel durumu olarak yerleştirmenin dengeli yapılması problemi tanımlanmış ve gene bu problemi çözmek için bazi algoritmalar üzerinde çalışılmış ve bu algoritmalar denenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

In general, the bin packing problem can be defined as the problem of packing a number of "pieces" into a number of "bins" so as to attain some objective(s) subject to the constraints on the size of the bins and/or number of the bins and/or the number of the pieces.

Some special cases of the problem can be listed as

(1) the classical bin packing problem,

(2) the variable sized bin packing problem,

(3) the dual bin packing problem.

These problems will be defined and discussed in Chapter 2.

Bin packing problem is a special case of the cutting stock problem and the assembly line balancing problem, and it can be used to model a number of real world applications such as [1, 2, 3] :

- Cutting standard size stock (cable, steel bar, etc.) into usable size,
- placing files of varying sizes on as few tracks of a disk as possible,
- prepaging,
- packing of variable length strings into fixed length words,
- minimizing number of machines necessary for completing all tasks by a given deadline.

Further application areas of bin packing problems can be found in the above references .

Bin packing problem is NP-hard in the strong sense. NP-hard problems leave very little hope for finding polynomial time algorithms for exact solutions. The reason is that they are proven to be " just as hard as " a large number of other problems that are widely recognized as being difficult and that have been confounding the experts for years [4]. Therefore, it is wise to search for "good" solutions using heuristic algorithms instead of searching for the optimal solution to these kinds of problems.

The heuristic algorithms do not guarantee to find an optimal solution, however, they use some simple heuristics by which it is possible to produce near optimal or sufficiently good solutions. Here, the question is to be able to estimate and evaluate the behavior of an algorithm given a string of input. To do this, three analytical methods are available : Worst-case analysis, probabilistic analysis and statistical analysis.

The worst-case analysis establishes the maximum deviation from optimality that can occur when a specified heuristic is applied within a given problem class [5]. The worst-case performance measure of a bin packing algorithm S gives an upperbound to the ratio of the number of bin used by that heuristic bin packing algorithm executed on a list of elements L, say S(L), to the optimum number of bins L^* . Let $R_S(K)$ be the maximum ratio $S(L)/L^*$ for any list L with $L^* = K$. The 'performance ratio', r(S), is defined as $\lim_{K\to\infty} R_S(K)$.

In the probabilistic approach, one assumes a density function for the problem data and establishes probabilistic properties of a heuristic such as the expected performance of the heuristic or a bound on the probability that the heuristic finds a solution within a prespecified percentage of optimality.

In the statistical approach, one usually applies the heuristic on a large number of randomly generated problems performing a deterministic simulation to draw some statistical inferences on the algorithm [5].

Although, in general, the algorithms with exponential time complexity functions do not yield satisfactory results in regard of computation time when input string is large, some exponential algorithms, on the average, may be able to solve sufficiently large problems. Being an algorithm which has an exponential time complexity function, simplex algorithm which is used to solve linear optimization problems constitutes a good example to exponential algorithms that work well on the average. So it may worth trying to generate optimal algorithms to solve NP-hard problems.

In this study, an optimal algorithm for the "bin packing" problem will be developed. Besides, a new bin packing problem will be defined as packing a number of pieces into bins keeping the level of bins as smooth as possible, " Smooth Packing Problem ", and the previous optimal algorithm will be modified to solve this newly defined bin packing problem . In addition to these optimal algorithms , some heuristics to solve both of the problems will be developed and compared against existing ones.

2. LITERATURE SURVEY

Depending on the type of application they are meant for, various definitions and formulations of the bin packing problem have appeared in the literature. Classical bin packing, dual bin packing and variable sized bin packing problems are the main topics of the following survey. In this chapter, these different definitions will be discussed using examples taken from the literature.

2.1. Classical Bin Packing

The bin packing problem can be defined as the problem of placing the elements of a given list L of real numbers between 0 and 1 into a minimum number, L^*, of "bins" so that no bin contains numbers whose sum exceeds one [1].

In analyzing bin packing problems, because they are NP-hard, main interest is concentrated on finding efficient heuristics to solve the problem and analyzing the performance of these heuristics compared to the optimal solution.

Mainly, there are two types of bin packing algorithms. If the numbers in list L are available one at a time and the algorithm has to assign each number to a bin before the next one becomes available, this kind of algorithms are called the ' on-line ' algorithms [6]. However, if there is no such requirement, then the corresponding algorithms are called ' off-line ' algorithms.

Worst-case performance bounds for simple on-line heuristics First-Fit (FF) and Best-Fit (BF) ; and off-line heuristics First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) are discussed in [1] . The definitions are given as follows :

First-Fit : Let the bins be indexed as $B_1, B_2, \ldots$, with each initially filled to level zero. The numbers $a_1, a_2, \ldots, a_n$ will be placed in that order. To place a_i, find the least j such that B_j is filled to level $\beta \leq 1-a_i$ and place a_i in B_j. B_j is now filled to level $\beta+a_i$.

Best-Fit : Let the bins be indexed as $B_1, B_2, \ldots$, with each initially filled to level zero. The numbers $a_1, a_2, \ldots, a_n$ will be placed in that order. To place a_i, find the least j such that B_j is filled to level $\beta \leq 1-a_i$ and β is as large as possible, and place a_i in B_j. B_j is now filled to level $\beta+a_i$.

First-Fit Decreasing : Arrange $L = (a_1, a_2, \ldots, a_n)$ into non-decreasing order and apply First-fit algorithm to the derived list.

Best-Fit Decreasing : Arrange $L = (a_1, a_2, \ldots, a_n)$ into non-decreasing order and apply Best-fit algorithm to the derived list.

The main results concerning the performance of the above heuristics can be summarized as follows [1]:

(1) $r(FF) = 17/10$,

$r(BF) = 17/10$,

$r(FFD) = 11/9$,

$r(BFD) = 11/9$,

(2) for L $[1/6, 1]$, $BFD(L) \leq FFD(L)$,

(3) for L $[1/5, 1]$, $BFD(L) = FFD(L)$,

(4) for L $(0, 1/2]$, $FFD(L) \leq 71/60\ L^* + 5$.

The algorithms dicussed in [1] are simple list processing algorithms. A possible 'one step further' improvement in these algorithms is to split the list L into some sublists and treat them accordingly. Two such algorithms are presented in [6] : The on-line algorithm Refined First-Fit (RFF) and the off-line algorithm Refined First-Fit Decreasing (RFFD). In that paper it is shown that

(1) $r(RFF) \leq 5/3$ and

(2) $r(RFFD) \leq 11/9 - \epsilon$ for $\epsilon = 10^{-7}$.

Furthermore, a lower bound for on-line bin packing algorithms are derived and it's proved that for any on-line bin packing algorithm S, $r(S) \geq 3/2$. It is also shown that for $\epsilon = 10^{-5}$ there is an $O(n \log n)$-time algorithm S for bin packing such that if a list L has all numbers in $(0, 1/2]$ then $S(L) \leq (71/60 - \epsilon)\ L^* + 5$. Besides, the question ' how well can an $O(n \log n)$-time algorithm perform' is discussed. It is shown that for an algorithm S for the generalized d-dimensional bin packing, S must have $r(S) \geq d$.

An off-line algorithm offBP is discussed in [6] and it's shown that for any list L, $offBP(L) \leq 4/3\ L^* + 2$ and offBP should be faster then $O(n \log n)$ algorithms .

Another way of evaluating the bin packing algorithms is to analyze the probabilistic performances of the algorithms. The recent studies are mostly done in this field.

A probabiliy model of the bin packing problem is given in [5]. It concentrates on the so called Next-Fit (NF) algorithm and develops expected values for the comparative performance of this rule and an optimization rule. In this paper, a markov model is used to represent the behaviour of the algorithm and after deriving general formulas for the expected performance, some numerical results are obtained for uniformly and exponentially distributed piece sizes.

Instead of representing the algorithms by a markov model in which the expected performance is bounded by the unknown expected optimal number of bins as in [5], it is possible to estimate the expected number of bins packed by a heuristic as a function of the number of pieces to be packed . Such an analysis for the NF algorithm is done in [7] and numerical results are obtained for the uniformly and exponentially distributed piece sizes. Besides, the marginal performance and average performance ratios of a number of bin packing heuristics (NF, BF, FF, FFD) are compared statistically and coefficients of a linear regression model relating the number of pieces to the expected number of bins packed are presented.

Another important work is reference [8] which addresses the asymptotic probabilistic behaviour of OPT(I)/n as $n \to \infty$ where instance I is a vector of n independent random variables, with common distribution

function (CDF), $F(x)$, $0 \leq x \leq 1$. analysis are done for symetric, convex and concave CDF's seperately and some general formulas to show convergence of $OPT(I_F)/n$ when F is an observed distribution are given. Besides, in this paper , the author represents the calculation of covergence rate as a problem to be solved.

A general problem with the probabilistic analysis of the bin packing algorithms is that the analytical results are very hard to obtain, even for the simplest algorithms and the simplest piece size distributions, because of the complexity of the calculations.

2.2 Variable Sized Bin Packing

The variable sized bin packing problem is that of packing a list of pieces into bins so as to minimize the total space used in the packing where bin sizes need not to be equal in size [9]. In [9] a model representing the problem and three algorithms are discussed extensively.

The algorithms presented are Next-Fit using Largest bins only (NFL), First Fit Decreasing using Largest bins at end Repack the smallest possible bins (FFDLR), First Fit Decreasing using Largest bins but Shifting as necessary (FFDLS). By analyzing the worst-case performance bounds for these algorihms it is shown that

$$NFL(L) < 2\,L^* + 1\,,$$

$$FFDLR(L) < 3/2\,L^* + 1\,,$$

$$FFDLS(L) < 4/3\,L^* + 3$$

for any list L. It is also observed that the time complexity functions of NFL, FFDLR, and FFDLS are O(n), O(n log n + h log k), O(n log n + n log k), respectively, where h denotes the number of bins packed and k denotes the number of distinct bin sizes.

2.3 Dual Bin Packing

The dual bin packing problem is defined in [10] as follows : "Suppose you are given a set $I = (a_1, a_2, \ldots, a_n)$ of items, a size $s(a) > 0$ for each item a and a threshold, $T > 0$. What is the maximum number m such that I can be partitioned into sets $X_1, \ldots, X_m$, where each set X has total size $s(X) = \sum_{a \in X} s(a) \geq T$, and hence can fill a one dimensional bin to at least this threshold ?" . Algorithms NF, FFD and Lowest-Fit Decreasing (LFD) are discussed with respect to dual binpacking and a statistical analysis is performed to analyze the average-case behaviour of dual bin packing algorithms [10].

The First-Fit Increasing heuristic for dual bin packing is studied under the assumption that piece sizes are choosen uniformly over (0, 1] and it is shown that

$$P(L^*/FFI(L) < 1 + (1/n)) \geq 1 - \epsilon$$

given a desired degree of confidence $1-\epsilon$ [11].

III. FORMULATIONS OF BIN PACKING PROBLEMS

The classical bin packing problem as defined in Chapter 2 can be formulated as follows:

$$(1) \qquad \min \sum_{j=1}^{m} y_j$$

$$\sum_{i=1}^{n} c_i x_{ij} \leq y_j \quad , \quad j = 1 \ldots m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad , \quad i = 1 \ldots n$$

where x_{ij} and y_j can be defined as

$$x_{ij} = \begin{cases} 1 & \text{if } i^{th} \text{ piece is packed into the } j^{th} \text{ bin} \\ & i = 1 \ldots n \, , \; j = 1 \ldots m \\ 0 & \text{otherwise} \end{cases}$$

$$y_j \in (0, 1) \text{ and integer} \, , \; j = 1..m \; .$$

Because the main concern of the problem is to minimize the number of bins packed, the same problem can be formulated by assigning some weights to the bins and eliminating the variable y_j, $j = 1..m$, from the formulation as follows:

$$(2) \qquad \min \quad \sum_{j=1}^{m} w_j \sum_{i=1}^{n} c_i x_{ij}$$

$$\sum_{i=1}^{n} c_i x_{ij} \leq 1 \quad , \; j = 1 \ldots m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad , \; i = 1 \ldots n$$

where x_{ij} , $i = 1 \ldots n$, $j = 1 \ldots m$, is defined as in the above formulation and w_j, $j = 1 \ldots m$, being the weight assigned to bin j. In this formulation, assigning the weights, w_j, is very important. They should be choosen in such a way that packing a piece to a bin with a lower index value should absolutely be profitable compared to packing it to a bin with a higher index value. In general, $w_j << w_{j+1}$ is a valid choice .

A further improvement in all the formulations can be attained by considering an obvious fact.

OBSERVATION 3.1. If the capacity of each bin is taken to be the unity , then pieces with sizes larger than 1/2 should be put into seperate bins, that is they can not be put into the same bin.

Taking Observation 3.1 into consideration, (2) can be formulated as

$$(3) \qquad \min \sum_{j=1}^{m} w_j \sum_{i=1}^{n-|A|} c_i x_{ij}$$

$$\sum_{i=1}^{n} c_i x_{ij} \leq 1 \quad , \; j = |A|+1 \ldots m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad , \; i = 1 \ldots n-|A|$$

$$\sum_{i=1}^{n} c_i x_{ij} \leq 1 - l_j \quad , \; j = |A|+1 \ldots m$$

If x_{ij} is defined as in the previous formulations, the pieces having sizes in $(0, 1/2]$ are put into set A and $|A|$ represents the size of this set; and l_j is defined as the level of bin j after pieces in set A are packed into seperate bins. This modification decreases the number of decision variables by $|A| * m$ and transforms the classical bin packing problem into a special case of the variable sized binpacking problem where bins with capacities less then the unity are counted as packed bins.

IV. SIMILAR TREE SEARCH ALGORITHM (STSA)

In this chapter, a new procedure will be introduced to solve zero-one integer programming problems stated in the form

$$\max \quad f(x) = cx$$

s.t.

$$Ax \leq b$$

$$x \in (0, 1) \text{ and integer.}$$

STSA utilizes a forest representation of the solution space of zero-one integer programming models . Figure 1 graphically shows this way of representation. Every node in the forest represents the solution which is obtained by setting the variable associated with this node and the variables in the upper level nodes but on the same path equal to one and setting all the other variables to zero. For example, in Figure 1, Node A represents the solution obtained by setting x_4 and x_3 equal to one and all the other variables to zero.

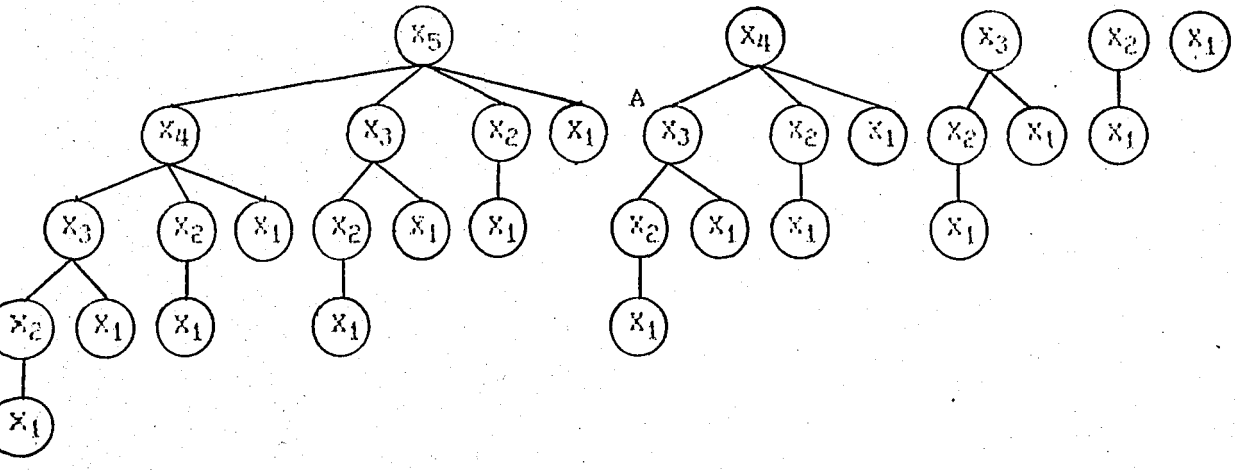

Figure 1. Representation of the Solution Space

4.1. The Algorithm

STSA uses a search mechanism on the forest shown in Figure 1 making use of the similarity between some trees in the forest.

Let us define t_i as the tree having the node which is related to variable i as the root node.A solution S can be represented by a set of indices whose related variables have the value of one in that particular solution, and $V(t_j)$ as the optimal value attained in this solution. Besides, the set notation $W(t_i) = \{ i \setminus K \}$ will be used to indicate that the first element of set $W(t_i)$ is i and $K = W(t_i) - \{i\}$, and Val(i, j) will denote the value of the solution $Sol(i, j) = \{i \setminus S(t_j) \}$. Using these definitions we reconfigure the forest shown in Figure 1 as it is in Figure 2.

Figure 2 inspires us to use a backsearch algorithm to implicitely enumerate all the possible solutions. Therefore, the following algorithm can be used to calculate the optimal values of all the trees and thus the overall optimal solution.

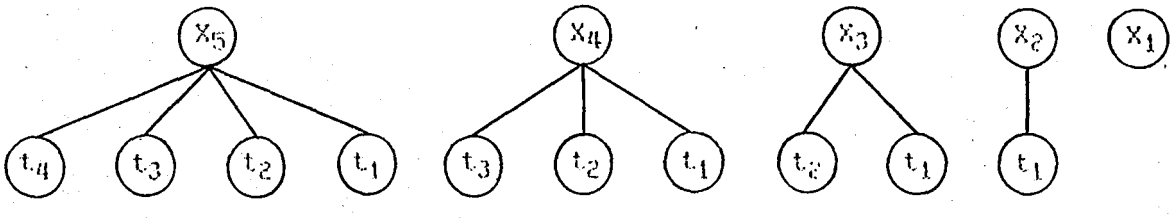

Figure 2. Reconfiguration of the Solution Space

Algorithm STSA

(input : Termination Node, Pre-Assigned

output : Optimal Value , Optimal Solution)

STEP 1 : Set $i = 1$,

$$V(t_1) = c_1 + \sum_{j \in \text{Pre-Assigned}} c_j,$$

$S(t_1) = \{ 1 \setminus \text{Pre-Assigned} \}$.

IF $S(t_1)$ is infeasible then Optimal Value = 0

Optimal Solution = {}

STEP 2 : Set $i = i + 1$.

Calculate $Val(i, j) = V(t_j) + c_i$ and

$Sol(i, j) = \{ i \setminus S(t_j) \}$ for all j, $j<i$.

IF $Sol(i, h)$ is infeasible for any $h < i$, then

run STSA(k, NewPre-Assigned, $NewV(t_k)$, $NewS(t_k)$)

where NewPre-Assigned = Pre-Assigned + {i} and

$k = \max \{ h \mid Sol(i, h) \text{ is infeasible} \}$.

Set $Val(i, h) = NewV(h)$,

$Sol(i, h) = NewS(h)$ for all h .

STEP 3 : Set $V(t_i) = \max \{ c_i, \max_{j<i} Val(i, j) \}$.

STEP 4 : IF $V(t_i) = c_i$, then set $S(t_i) = \{i\}$.

IF $V(t_i) = Val(i, k)$, then set $S(t_i) = Sol(i, k)$.

STEP 5 : IF $i <$ Termination Node, then go to STEP 2.

STEP 6 : Optimal Value = $\max_{j= 1..n} \{ V(t_j) \}$.

Optimal Solution = $S(t_k)$, if $V(t_k)$ = Optimal Value.

To run STSA the initial values for the Termination Node and Pre-Assigned should be n and null set , respectively.

4.2. An Example on How to Use STSA

In this section , a special case of the well-known " Knapsack Problem " will be solved to demonstrate the use of Similar Tree Search Algorithm (STSA). The formulation of the general zero-one Knapsack problem is as follows :

$$\max \quad \sum_{i=1}^{n} c_i x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} b_i x_i \leq \text{GOAL}$$

$$x_i \in (0, 1) \text{ and integer.}$$

In this formulation c_i is the cost and b_i is the weight parameters . Available optimal solution procedures to this well-known problem is presented in reference [12].

In this study, the cost and the weight parameters will be assumed to be equal, and this problem will be called as the Selection Routine, in the sense that this is the formulation of the question " how to select a number of pieces among a given set { x_i, i=1...n} which will sum up

to a value as close to a prespecified GOAL as possible," and in Section 5.1 this routine will be used to generate a heuristic algorithm to solve the bin packing problem.

STSA requires a feasibility check and a value function to be able to compare the alternative solutions. In this example the feasiblity check and the value function are stated as follows:

The value of a solution set S, say V(S), is obtained by the summation $\sum_{i \in S} c_i$ and this solution set S is said to be feasible if this sum does not exceed GOAL.

The steps to solve a given problem is demonstrated belove. The matrix Val(i, j) shows the values of the solutions $\{ i \setminus S(t_j), j=1, \ldots, i-1 \}$, where Val(i, 1) equals c_i. If infeasibility occurs in any entry of Val(i, j), this infeasible value and the new value calculated for this node are given in the same entry separated by a comma. The alphanumeric codes at the right of infeasible entries specify the place where the calculations for a new value for that entry are done.

In this particular example , where GOAL is taken to be 40, the optimal value is 40 which is the maximum of the node values and the optimal solution yielding this value is { 1, 2, 3, 5 }. Although the GOAL is achieved at node 5, in order to catch the multiple solutions the algorithm did not terminate.

This procedure is coded in Pascal and run on an IBM compatible PC. In this application a lowerbound is used to shorten the bactracking mechanism. The lowerbound is given in Observation 4.2.1.

Example Problem : Pre-Assigned = null set Termination Node = 6

Piece(i)	6	7	9	10	11	12
j i	6	5	4	3	2	1
0	6	7	9	10	11	12
1	18	19	21	22	23	
2	29	30	32	33		
3	39	40	42, 31 A			
4	38	39				
5	46, 36 B					
Node Value	39	40	32	33	23	12

Pre-Assigned = {4}
Termination Node = 3

			A			
				19	20	21
				31	32	
				42, 30 A1		
				31	32	21

Pre-Assigned = {4, 3}
Termination Node = 2

				A1		
					30	31
					42, 00 A11	
					30	31

Pre-Assigned = {4, 3, 2}
Termination Node = 1

					A11	
						42, 00
						00

Pre-Assigned = {6}
Termination Node = 5

	B					
		13	15	16	17	18
		25	27	28	29	
		36	38	39		
		46, 35	48, 37 B1			
		45, 34 B2				
		36	38	39	29	18

Piece(i)	6	7	9	10	11	12
1	6	5	4	3	2	1

Pre-Assigned = {6, 4}
Termination Node = 3

B1

25	26	27
37	38	
48, 36 B11		
37	38	27

Pre-Assigned = {6, 4, 3}
Termination Node = 2

B11

36	37
43, 00 B111	
36	37

Pre-Assigned = {6, 4, 3, 2}
Termination Node = 1

B111

43, 00
00

Pre-Assigned = {6, 5}
Termination Node = 4

B2

22	23	24	25
34	35	36	
45, 33	46, 34 B22	46, 00 B21	
44, 33 B23			
34	35	36	25

Pre-Assigned = {6, 5, 2}
Termination Node = 1

B21

46, 00
00

Pre-Assigned = {6, 5, 3}
Termination Node = 2

B22

34	35
46: 00 B221	
34	35

Piece(i)	6	7	9	10	11	12
i	6	5	4	3	2	1
Pre-Assigned = {6, 5, 3, 2} Termination Node = 1					B221	46, 00 00
Pre-Assigned = {6, 5, 4} Termination Node = 3			B23	32 44, 00 43, 00 B232 32	33 45, 00 B231 33	34 34
Pre-Assigned = {6, 5, 4, 2} Termination Node = 1					B231	45, 00 00
Pre-Assigned = {6, 5, 4, 3} Termination Node = 2					B232	44, 00 00

OBSERVATION 4.2.1. Arrange pieces in nondecreasing order of piece sizes. If $\sum_{i=1}^{L+1} c_i > GOAL$ and $\sum_{i=1}^{L} c_i \leq GOAL$ then L is a lowerbound to the number of pieces that an optimal solution will posess.

Also, an upperbound to the number of pieces in the optimal solution can be imposed in the same manner as follows.

OBSERVATION 4.2.2. Arrange pieces as in Observation 4.2.1. If $\sum_{i=U}^{n} c_i > GOAL$ and $\sum_{i=U+1}^{n} c_i \leq GOAL$, then (n-U) is an upperbound to the number of pieces that an optimal solution will possess.

Although the lowerbound should explicitly be stated in the algorithm, the upperbound is implicitly overtaken in the algorithm STSA by terminating the recursion if { i \ Pre-Assigned } is infeasible.

4.3. Solving the Bin Packing Problem Using STSA

As formulated in Chapter 3, Bin Packing problem is a two dimensional zero-one integer programming problem. This section will be devoted to apply STSA to solve this well-known problem.

First of all, STSA as it is presented in Section 4.1. depends on calculation of Val(i, j) where indices i and j represent root nodes of trees related to each variable. However, bin packing is a two dimensional problem and a transformation function should be set to find out the bin and the piece related to every node . The following functions may be used for this purpose .

Given a Node

$$\text{Bin (Node)} = \left\lfloor \frac{(\text{Node} - 1)}{n} \right\rfloor$$

$$\text{Piece (Node)} = \text{Node} - (\text{Bin} - 1) * n \ .$$

Therefore, the first n nodes will be related to the first bin and the second n of them to the seond bin, etc, as shown in Figure 3.

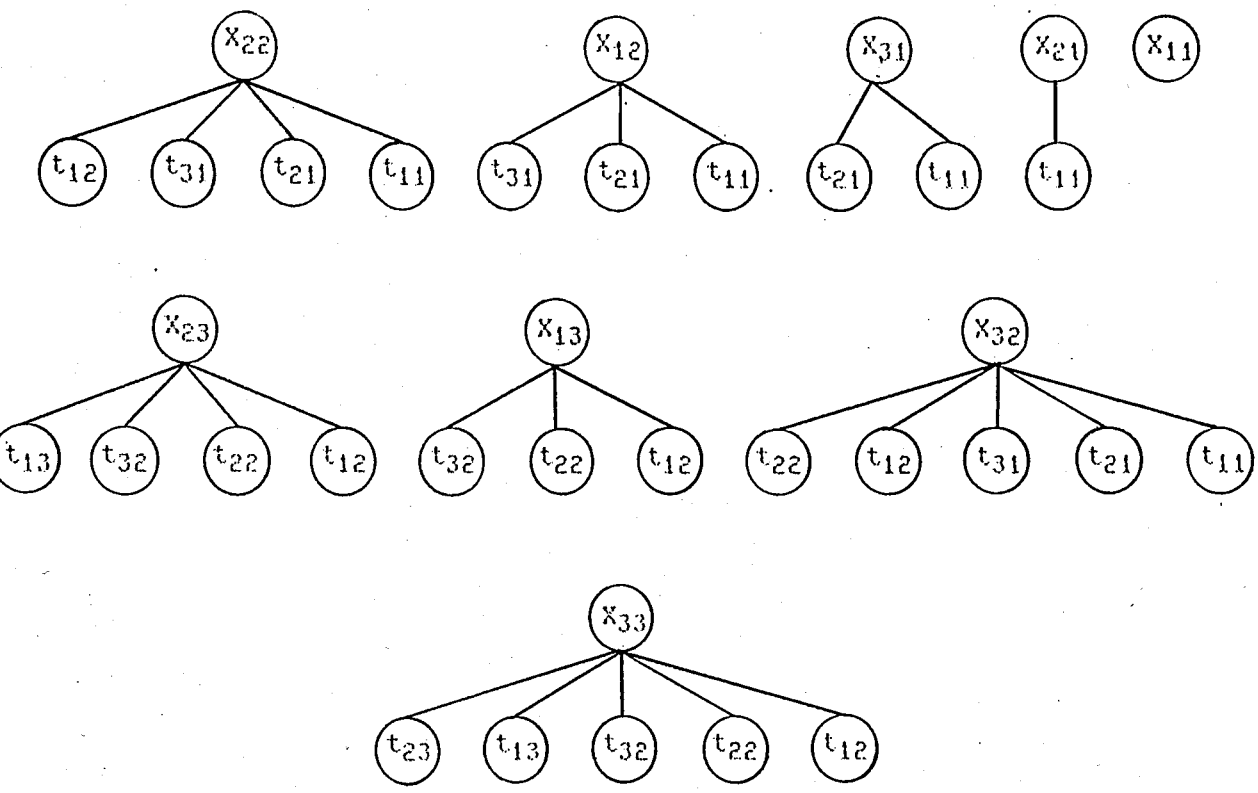

Figure 3. Solution Space of the Bin Packing Problem

The second step may be to define the feasibility check procedure and the value function. In this application, the value of a solution is determined by two distinct measures: The number of pieces packed into bins and the total value of the pieces packed (that is $\sum_{(i,j)\in S} c_{ij}$). While comparing two alternative solutions, the first criterion is the number of pieces packed in each solution because our main concern is to maximize the number of pieces packed into the existing bins (This

duality will be discussed extensively later). If two solutions donot compete on this measure, the solution in which the summation of the values of the pieces packed is larger should be prefered.

In STSA, beginning from the first node, all the nodes are calculated in turn. In this particular case, this corresponds to filling up the bins in turn beginning from the first one. Therefore, if the dual problem, maximizing the number of pieces packed to a given number of bins, is solved at each node, the number of bins at the present node when all the pieces are packed will be the optimal.

To summarize the procedure: Begin from the first node and calculate the other nodes in turn. At each node try to maximize the number of pieces packed. The bin number corresponding to the node at which all the pieces are packed is the optimum number of bins.

Noticing that in this particular case we pack identical bins, we can state the following observation.

OBSERVATION 4.3.1. Let the bins be indexed as $B_1, B_2, \ldots$ Filling up bin B_i while B_j is filled to level zero is meaningless if $i > j$.

So, as shown in Figure 3, at $t_{3,1}$ there is no need to consider the trees related to the first bin.Besides, a fact can easily be seen about the backtracking mechanism.

OBSERVATION 4.3.2. The recursive algorithm backtracks at most n number of nodes.

This is true because there are n pieces to be packed and none of them can be packed more than once. So, after n-1 backtracks, at the n'th trial, say at node i, the solution { i \ Pre-Assigned } will be infeasible because { Pre-Assigned } set has n pieces packed , and at the first check of Algorithm BACKTRACK the recursion will be terminated.

Another improvement in the algorithm can be achieved by tracing a lowerbound to the number of pieces packed into bins: If at any node, say at node j, LB number of pieces are packed, and at node i, i>j, only L number of pieces are packed where L < LB, and this solution is infeasible, then there is no need to backtrack to find a feasible solution to this node because it won't yield an optimal solution. However, we have to keep the information that the particular node was left infeasible, because while backtracking in the following nodes we may need the feasible solution to this node. In such a case, the feasible solution to this node will be calculated. That is, by implementing this procedure we eliminate some unnecessary backtracking and activate the backtracking routine only when generating a competitive solution is promised.

Besides, further fathoming of nodes is possible by implementing a lower and an upper bound on the number of bins packed. A strict upper bound can be obtained by using the FFD heuristic algorithm presented in Chapter 2. In algorithm STSA, any node i is an alternative solution to the first BIN(i) number of bins, and by solving FFD using unpacked pieces in that node, an upper bound, UB(i), for the number of bins that

will be packed using that solution can be found. The overall upper bound upto node j will be $OUB(J) = \min_{i<j} UB(i)$.

The lower bound to the number of bins that will be packed in any node can be found by the following formula:

$$LB(i) = BIN(i) + \left\lceil \sum_{j=1}^{n} c_j - \sum_{\substack{j \in \text{pieces} \\ \text{packed}}} c_j \right\rceil ,$$

where BIN(i) is the bin related to node i. The lower bound is calculated at nodes numbered (n*k+1), k = 2, ..., m-1, for the nodes (m*k-1) + h, h = 1, ..., n, and node i is fathomed if $OUB(i) \leq LB(i)$. The reason for not calculating the lower bound at every node is that the solution obtained at that node may be used to generate solutions in the preceding nodes related to that bin. Therefore, lower bound calculations of nodes related to a bin are done after the solution to the last node of that bin is found.

The most time consuming part of the algorithm is the recursion mechanism. So, prevention of recursion is an important time saving. For this purpose a prevention rule can be implemented as follows :

If Val(i, j) > Bin Capacity then calculate LB(i) by adding value Bin Capacity - Val(i, j) into total value of pieces unpacked and donot initiate recursion if $OUB(i) \leq LB(i)$.

In Chapter 3 using Observation 3.1 the variable number in the bin packing problem was reduced. This fact can again be used to reduce the

problem size while implementing STSA. If we pack pieces with sizes larger than one half before starting to the regular algorithm, this will change the capacities of those bins pre-packed and change the problem to a special case of variable sized bin packing problem, where the bins with capacities less then one should not be left empty if it is not necessary.

In this case, Observation 4.3.1 does not hold because the bins are not identical any more and all the succeeding nodes should be taken into consideration at every node, like in the previous example on the Knapsack Problem.

4.4. An Optimal Smooth Packing Algorithm (OSP)

The Smooth Packing problem can be defined as packing a number of pieces into bins keeping the level of bins as smooth as possible. The objective can be stated as maximizing the minimum or minimizing the maximum bin level. Which one to use depends on the area of application. For example, if the piece sizes stand for a physical weight, solving a minimax problem leads to a more preferable solution. However, if the levels represent the level of raw material in a deep container which will be picked up by a laborer, it is wise to solve the maximin problem. Below, the linear programming formulation of the problem is given for the minimax case.

$$\min \quad y$$

s.t.

$$\sum_{i=1}^{n} c_i x_{ij} \leq y \qquad j = 1 \ldots OPT,$$

$$\sum_{j=1}^{OPT} x_{ij} = 1 \qquad i = 1 \ldots n,$$

where

$$x_{ij} = \begin{cases} 1 & \text{If i'th piece is packed into j'th bin} \\ 0 & \text{Otherwise} \end{cases}$$

y is a real variable , and OPT is the optimal solution of problem (3) in Chapter 3.

In this section, an optimal algorithm, OSF, by which the maximum bin level will be minimized when the pieces are packed into optimal number of bins will be presented. This algorithm depends on solving the algorithm STSA for binpacking as shown in Section 4.3. repetitively while changing the capacity of bins at each step. The flow chart of the algorithm is given in Figure 4.

Example Problem :

In this section, the problem with SEED = 14 and n = 10 will be solved by OSF.

Iteration 1 . Bin Capacity = 1000

Bin	Pieces	Level
1	822, 134, 40	996
2	761	761
3	745, 241	986
4	742	742
5	589	589
6	545, 412	957
	Maximum Level =	996

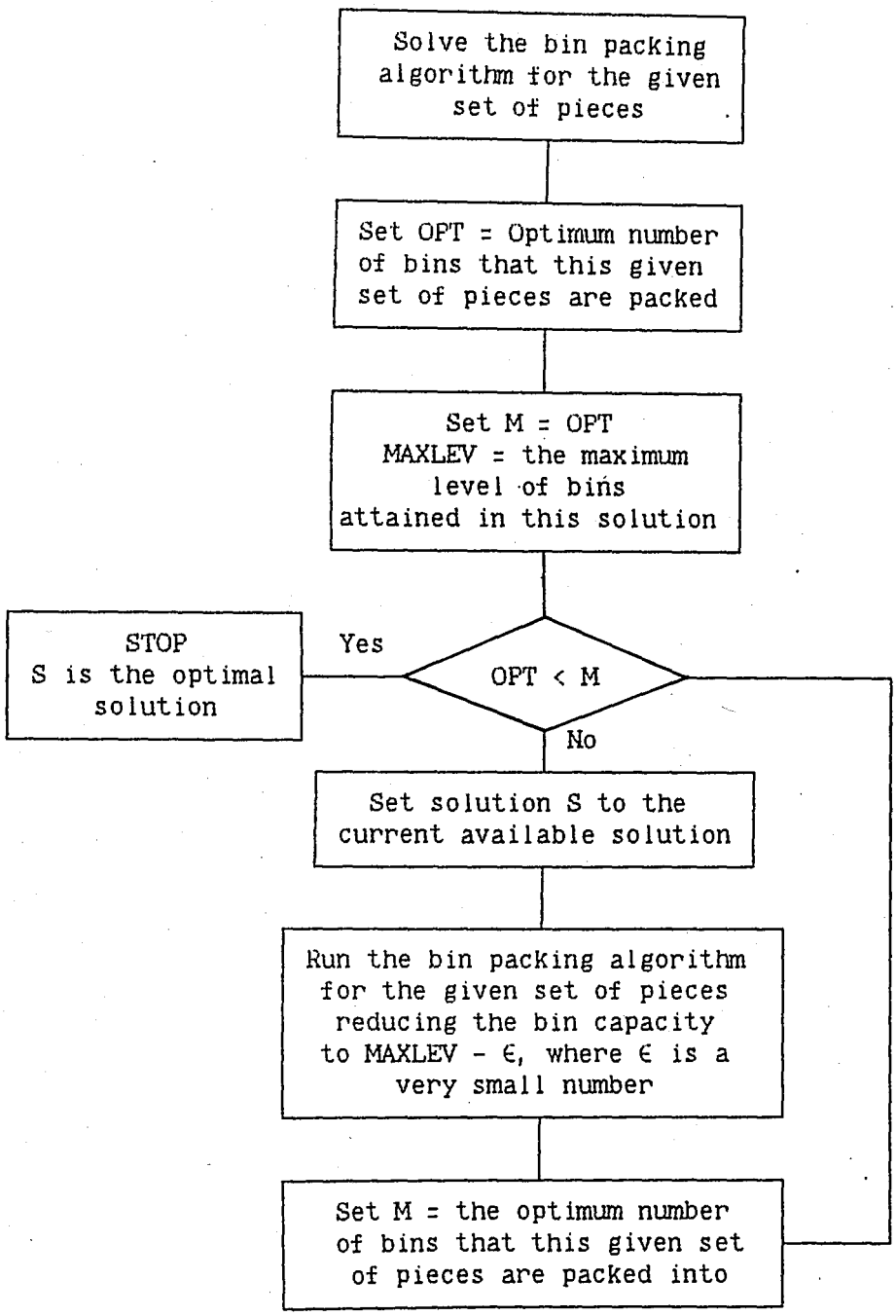

Figure 4. Flow Chart of Optimal Smooth Packing Problem

Iteration 2 . Bin Capacity = 995

Bin	Pieces	Level
1	822, 134	956
2	761, 40	801
3	745, 241	986
4	742	742
5	589	589
6	545, 412	957
	Maximum Level =	986

Iteration 3 . Bin Capacity = 985

Bin	Pieces	Level
1	822, 134	956
2	761, 40	801
3	745	745
4	742, 241	983
5	589	589
6	545, 412	957
	Maximum Level =	983

Iteration 4 . Bin Capacity = 982

Bin	Pieces	Level
1	822, 134	956
2	761, 40	801
3	745	745
4	742	742
5	589, 241	830
6	545, 412	957
	Maximum Level =	957

Iteration 5 . Bin Capacity = 956

Pieces are packed into seven bins in iteration 5. Therefore, in the optimal solution the maximum bin level is equal to 957.

V. HEURISTIC ALGORITHMS

5.1. Heuristic Algorithm SBT for Bin Packing

The well-known heuristic algorithms for the bin packing problem, First-Fit (FF), Best-Fit (BF), First-Fit Decreasing (FFD), Best-Fit Decreasing (BFD)were extensively discussed in Chapter 2.In this section a new heuristic bin packing algorithm, Select the Best in Turn (SBT) which uses the Selection Routine in Section 4.2, will be introduced. The basic idea the algorithm stands on is that " among a given set of pieces, select a group whose cumulative value is as close to the capacity of the bins as possible but not exceeding it, and assign that group as a bin ". It is for sure that SBT is an off-line algorithm just like FFD and BFD. For selecting the best group, the algorithm given in Section 4.2. will be used. The flow chart of the algorithm is given in Figure 5.

Example Problem :

In this section, the problem with SEED = 16 and n = 10 will be solved by SBT.

Iteration 1. L = {486, 298, 110, 67, 753, 484, 417, 478, 431, 714}

B = {486, 431, 67}

Iteration 2. L = {298, 110, 753, 484, 417, 478, 714}

B = {478, 484}

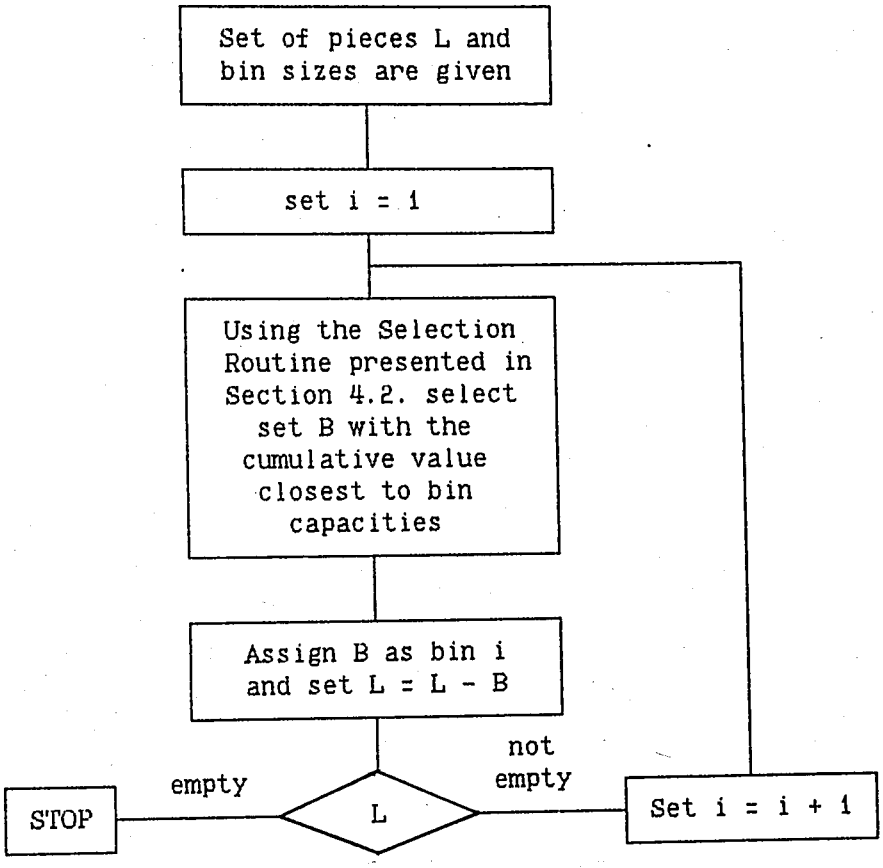

Figure 5. Flow Chart of Algorithm SBT

Iteration 3. L = {298, 110, 753, 417, 714}

B = {110, 298, 471}

Iteration 4. L = {753, 714}

B = {753}

Iteration 4. L = {714}

B = {714}

Therefore, the solution is

Bin	Pieces
1	486, 431, 67
2	478, 484
3	110, 298, 471
4	753
5	714

5.2. Algorithm HEUSTSA for Bin Packing

Another heuristic algorithm can be obtained by changing the optimal algorithm STSA. STSA is an exponential time algorithm and the bactracking mechanism is the routine where exponentially increasing number of operations are performed. In the heuristic algorithm HEUSTSA, the backtracing mechanism of STSA is eliminated and the procedure is reorganized as follows :

Algorithm HEUSTSA

STEP 1 : Set $i = 1$,

$V(t_1) = c_1$,

$S(t_1) = \{1\}$.

STEP 2 : Set $i = i + 1$,

$V(t_j) = 0$, for all j, $j<i$, if $\{ i \setminus s(t_j) \}$ is infeasible .

$V(t_i) = \max \{ c_i, V(t_j) + c_i \mid j<i \}$.

If the maximum value is c_i then

set $S(t_i) = \{i\}$.

If the maximum value is $V(t_K) + c_i$ then

set $S(t_i) = \{ i \setminus S(t_K) \}$.

STEP 3 : If $i < n$ then go to STEP 2.

STEP 4 : The solution is

$$S = \{S(t_j) \mid V(t_j) \geq V(t_k),\ k = 1...n\}$$

HEUSTSA is of time complexity $O((nm)^2)$ if there are n pieces to be packed and these pieces are packed into m number of bins.

Example Problem :

When the problem presented in the previous section is solved using HEUSTSA the following solution is obtained.

Bin	Pieces
1	468, 298, 110, 67
2	753
3	484, 471
4	478, 431
5	714

5.3. Heuristic Algorithms for Smooth Packing

In this section , two heuristic algorithms will be presented generated to solve the smooth packing problem by transforming the idea behind the algorithms BF and BFD. These two new algorithms will be called Worst-Fit (WF) and Worst-Fit Decreasing (WFD) in the sense that they proceed using arguments which are exactly opposite to that of BF and BFD.

Worst-Fit : Let the bins be indexed as $B_1, B_2, \ldots,$ with each initially filled to level zero. The pieces $a_1, a_2, \ldots, a_n$ will be placed

in that order. To place a_i, find the least j such that B_j is filled to level $0 < \alpha \leq 1-a_i$ and α is as small as possible, and place a_i in B_j if possible. If it doesnot fit, find the least j such that B_j is filled to level 0 and place a_i in B_j.

Worst-Fit Decreasing : Arrange $L = (a_1, a_2, \ldots, a_n)$ into nondecreasing order and apply Worst-Fit algorithm to the derived list.

VI. RESULTS

In this chapter, results obtained by running some randomly generated problems to evaluate the performance of algorithm STSA and to compare algorithms WF and WFD against the existing simple list processing algorithms FF, BF, FFD, BFD are given as tables. The example problems are generated by using the following random number generator:

For a given SEED ,

c_1 = SEED

$c_i = [\{(c_{i-1} * 25173) + 13849\} \text{ Mod } 32767] / 32767$, i = 2..n.

In order to evaluate the performance of the fathoming rules applied in STSA besides the overall efficiency of the algorithm, the number of nodes fathomed and number of prevented recursion attempts versus total recursive backtracks and total nodes generated are given below for some example problems. In the table, a problem is represented by the number of pieces to be packed, n, and SEED used to initiate the random number generator.

SEED	Nodes Fathomed	Total Nodes Gen.	Total Rec. Backtracks	Prevented Rec. Attempts
n = 10				
10	3	6	0	0
11	0	6	0	0
12	0	4	0	0
13	0	0	0	0 *
14	0	0	0	0 *
15	0	0	0	0 *
16	0	0	0	0 *
17	0	0	0	0
18	6	12	0	0
19	3	6	0	0

SEED	Nodes Fathomed	Total Nodes Gen.	Total Rec. Backtracks	Prevented Rec. Attempts
n = 15				
20	23	136	4	29
21	0	0	0	0 *
22	0	0	0	0 *
23	0	0	0	0 *
24	5	22	2	0
25	43	109	6	4
26	0	0	0	0 *
27	0	0	0	0 *
28	32	173	4	22
29	63	626	56	4
n = 20				
30	0	0	0	0 *
31	0	0	0	0 *
32	15	503	26	10
33	7	480	3	0
34	8	103	13	0
35	0	0	0	0 *
36	0	0	0	0 *
37	0	0	0	0 *
38	8	16	0	0
39	10	40	4	10
n = 30				
40	13	127	7	2
41	88	1330	47	24
42	142	1240	23	44
43	63	9036	301	28
44	12	88	3	0
45	63	1208	45	1
46	29	1969	71	46
47	85	4440	427	183
48	86	552	11	19
49	13	128	11	0
n = 40				
50	191	727	8	6
51	17	34	0	0
52	37	76	0	0
53	333	7963	321	553
54	18	36	0	0

SEED	Nodes Fathomed	Total Nodes Gen.	Total Rec. Backtracks	Prevented Rec. Attempts	
n = 40 (Continued)					
55	18	283	30	0	
56	87	258	1	63	
57	163	9170	386	90	
58	18	36	0	0	
59	87	749	9	95	
n = 50					
60	155	846	6	21	
61	500	22833	102	0	
62	21	492	19	0	
63	620	41012	217	4	
64	328	2527	14	10	
65	213	32789	122	0	
66	21	42	0	0	
67	21	42	0	0	
68	246	20599	95	0	
69	476	39488	156	1	
n = 60					
70	57	145	0	0	
71	26	52	0	0	
72	0	0	0	0	*
73	261	57998	243	0	
74	36	9003	116	2	
75	0	0	0	0	*
76	23	91	1	0	
77	370	9254	57	92	
78	422	46014	356	0	
79	723	23418	142	3	

The lower and the upper bound of the problems with a star at the right of the related rows are found equal at the beginning of the algorithm , and the optimal solution is detacted before the recursive procedure STSA is run. In these examples, it is interesting to notice that the standard deviation of the number of nodes generated is very high.

The solutions of some example problems using the heuristics FF, BF, WF, FFD, BFD and WFD are presented in the following table. The number of bins packed and the maximum bin level achieved at that solution are given seperated by a comma.

SEED	FF	BF	WF	FFD	BFD	WFD
n = 10						
11	7 , 971	7 , 988	7 , 971	7 , 988	7 , 988	7 , 971
12	8 , 914	8 , 914	8 , 914	8 , 962	8 , 970	8 , 914
13	6 , 893	6 , 907	6 , 924	5 , 987	5 , 995	5 , 984
14	7 , 996	7 , 996	7 , 822	6 , 996	6 , 997	6 , 957
15	5 , 987	5 , 987	5 , 987	5 , 994	5 , 994	5 , 994
16	5 , 957	5 , 982	5 , 957	5 , 970	5 , 970	5 , 970
17	8 , 959	8 , 959	8 , 959	7 , 996	7 , 996	7 , 990
18	6 , 960	6 , 960	6 , 960	6 , 960	6 , 960	6 , 960
19	7 , 981	7 , 998	7 , 998	7 , 998	7 , 998	7 , 981
20	5 , 977	5 , 977	5 , 977	4 , 982	4 , 982	5 , 977
n = 15						
21	6 , 957	6 , 997	7 , 957	6 , 996	6 , 996	6 , 897
22	9 , 996	9 , 996	9 , 972	8 , 996	8 , 996	8 , 996
23	9 , 976	8 , 980	9 , 921	8 , 995	8 , 995	8 , 924
24	10 , 949	10 , 999	10 , 983	10 , 990	10 , 999	10 , 939
25	8 , 986	8 , 995	8 , 957	8 , 996	8 , 996	8 , 920
26	5 , 996	5 , 998	6 , 916	5 , 998	5 , 998	5 , 996
27	7 , 990	7 , 990	7 , 991	6 , 1000	6 , 997	7 , 891
28	8 , 976	8 , 976	8 , 976	8 , 997	8 , 997	8 , 976
29	9 , 1000	9 , 978	9 , 948	9 , 1000	9 , 990	9 , 912
30	7 , 977	7 , 977	8 , 901	7 , 986	7 , 987	7 , 957
n = 20						
31	9 , 1000	9 , 996	9 , 953	8 , 1000	8 , 999	8 , 949
32	12 , 994	12 , 998	12 , 994	11 , 1000	11 , 998	11 , 994
33	14 , 956	13 , 981	14 , 956	13 , 993	13 , 998	13 , 956
34	13 , 977	12 , 998	13 , 932	12 , 1000	12 , 999	12 , 961
35	12 , 991	12 , 991	12 , 991	11 , 998	11 , 998	12 , 954
36	11 , 993	11 , 998	11 , 968	11 , 999	11 , 999	11 , 968
37	12 , 987	11 , 993	13 , 979	11 , 1000	11 , 996	11 , 990
38	13 , 981	13 , 981	13 , 981	12 , 1000	12 , 999	12 , 981
39	12 , 967	12 , 992	13 , 951	11 , 995	11 , 995	11 , 961
40	13 , 997	13 , 997	13 , 997	12 , 1000	12 , 997	12 , 997

SEED	FF	BF	WF	FFD	BFD	WFD
n = 25						
41	14 , 998	14 , 998	15 , 987	13 , 997	13 , 997	13 , 987
42	15 , 993	15 , 998	15 , 993	14 , 998	14 , 998	14 , 993
43	18 , 992	18 , 998	18 , 992	17 , 998	17 , 998	17 , 997
44	15 , 971	15 , 998	15 , 971	14 , 998	14 , 999	14 , 971
45	17 , 999	16 , 999	17 , 992	16 , 999	16 , 999	16 , 992
46	14 , 989	14 , 988	14 , 993	13 , 1000	13 , 996	13 , 958
47	18 , 982	18 , 982	19 , 962	17 , 998	17 , 998	17 , 962
48	15 , 1000	15 , 991	16 , 966	14 , 1000	14 , 999	14 , 988
49	15 , 934	15 , 995	15 , 914	14 , 999	14 , 999	14 , 954
50	16 , 1000	16 , 998	17 , 995	15 , 998	15 , 998	15 , 995
n = 50						
51	32 , 1000	33 , 999	33 , 998	30 , 1000	30 , 999	30 , 994
52	31 , 990	30 , 999	31 , 999	28 , 999	28 , 999	28 , 994
53	26 , 1000	25 , 999	28 , 987	25 , 999	25 , 999	25 , 992
54	27 , 995	27 , 997	29 , 986	25 , 1000	25 , 999	25 , 999
55	27 , 998	27 , 997	29 , 976	26 , 1000	26 , 998	26 , 979
56	23 , 999	23 , 996	24 , 993	22 , 1000	22 , 999	23 , 993
57	29 , 996	29 , 998	31 , 973	28 , 1000	28 , 998	28 , 995
58	30 , 1000	30 , 998	31 , 998	29 , 1000	29 , 999	29 , 998
59	26 , 999	26 , 999	28 , 988	24 , 1000	24 , 999	25 , 999
60	26 , 999	26 , 999	27 , 999	24 , 1000	24 , 999	24 , 997
n = 100						
61	55 , 998	54 , 999	60 , 980	52 , 1000	52 , 999	52 , 993
62	52 , 999	51 , 999	57 , 999	49 , 1000	49 , 999	49 , 999
63	56 , 1000	56 , 999	60 , 993	55 , 1000	55 , 999	55 , 999
64	51 , 1000	51 , 999	55 , 998	49 , 1000	49 , 999	49 , 997
65	58 , 995	57 , 999	62 , 994	55 , 1000	55 , 999	55 , 997
66	59 , 998	57 , 999	63 , 995	57 , 999	57 , 999	57 , 996
67	52 , 1000	51 , 999	57 , 995	50 , 1000	50 , 999	51 , 995
68	58 , 999	58 , 999	61 , 999	56 , 1000	56 , 999	56 , 999
69	57 , 999	57 , 999	61 , 998	56 , 1000	56 , 999	56 , 997
70	57 , 1000	56 , 999	59 , 992	54 , 1000	54 , 999	54 , 999
n = 250						
71	126 , 1000	125 , 999	136 , 993	119 , 1000	119 , 999	120 , 999
72	137 , 1000	135 , 999	147 , 996	128 , 1000	128 , 999	128 , 999
73	139 , 1000	137 , 999	153 , 999	132 , 1000	133 , 999	133 , 999
74	139 , 1000	138 , 999	152 , 999	132 , 1000	132 , 999	132 , 998
75	141 , 1000	140 , 999	153 , 999	131 , 1000	131 , 999	132 , 999
76	136 , 1000	135 , 999	149 , 994	132 , 1000	132 , 999	132 , 998

SEED	FF	BF	WF	FFD	BFD	WFD

n = 250 (Continued)

77	128 , 1000 126	999 137	997 122	1000 122	999 123	999
78	133 , 1000 131	999 147	999 128	1000 128	999 128	999
79	138 , 1000 135	999 149	998 131	1000 131	999 131	998
80	137 , 1000 135	999 146	992 129	1000 129	999 129	999

n = 500

81	278 , 1000 274	999 300	999 268	1000 268	999 268	999
82	265 , 1000 260	999 284	999 250	1000 251	999 252	999
83	266 , 1000 263	999 289	998 256	1000 256	999 256	999
84	275 , 1000 271	999 302	996 264	1000 265	999 265	999
85	266 , 1000 263	999 295	999 255	1000 255	999 255	999
86	283 , 1000 279	999 308	999 275	1000 276	999 276	999
87	258 , 1000 255	999 288	999 246	1000 247	999 248	999
88	275 , 1000 272	999 297	999 263	1000 263	999 263	999
89	273 , 1000 270	999 296	998 262	1000 263	999 263	999
90	260 , 1000 254	999 287	997 246	1000 246	999 247	999

SEED	FF	BF	WF	FFD	BFD	WFD

n = 1000

91	504 , 1000 498	999 564	999 483	1000 483	999 486	999
92	521 , 1000 514	999 581	999 501	1000 502	999 503	999
93	534 , 1000 531	999 584	999 509	1000 509	999 509	999
94	510 , 1000 502	999 569	999 494	1000 494	999 495	999
95	514 , 1000 507	999 577	999 497	1000 498	999 499	999
96	524 , 1000 516	999 582	999 507	1000 507	999 507	999
97	518 , 1000 512	999 573	997 500	1000 500	999 500	999
98	547 , 1000 540	999 602	999 526	1000 527	999 527	999
99	504 , 1000 498	999 569	999 488	1000 489	999 491	999
100	522 , 1000 515	999 584	998 502	1000 502	999 502	999

As it is seen from the examples solved, WF is not a good algorithm for the smooth packing case. However, WFD packs pieces into same number of bins as FFD and BFD do and reduces the maximum level of bins considerably .

VII. CONCLUSION

In this study, some optimal and heuristic algorithms are generated for the solution of ordinary bin packing and newly defined smooth packing problems. Although, the optimal algorithms STSA, for the ordinary bin packing case, and OSP, for the smooth packing case, are algorithms of exponential time complexity, by the lower bounds generated they work well in small sized problems but cannot hadle big problems because they require large amount of computer memory. The logic behind Similar Tree Search Method is the main cotribution of this study to this area.

Besides, four new heuristic procedures are developed where two of them, HEUSTSA and SBT, are related to the ordinary bin packing problem. HEUSTSA and SBT are examples on how the similar tree search procedures can be rectified to obtain heuristic algorithms. However, the existing bin packing heuristics are superior with respect to time complexity functions and these algorithms are presented to inspire new perpectives in solution procedures to the bin packing problems.

WF and WFD algorithms are the smooth packing versions of the BF and BFD algorithms. WF uses more bins then FF and BF does and seems to be unsatisfactory in smooth packing problem. Besides, WFD performs well and while packing the pieces into same number of bins as FFD and BFD does, it reduces the maximum level of bins considerably for most of the cases and can be represented as a good algorithm for the smooth packing case.

REFERENCES

1. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M.R., Graham, R. L., " Worst Case Performance Bounds for Simple One Dimensional Packing Algorithms ", SIAM J. Computing, Vol.3, pp.299-325, 1974.
2. Charles, U.M., " A Linear Time Bin Packing Algorithm ", Operations Research Letters, Vol.4, pp.189-192, 1985.
3. Brown, A.R., Optimum Packing and Depletion, American Elsevier, New York, 1971.
4. Garey, M. R., Johnson, D. S., Computers and Intractability, W. H. Freeman and Company, San Fransisco, 1979.
5. Ong, H. L., Magazine, M. J., Wee, T. S., " Probabilistic Analysis of Bin Packing Heuristics ", Operations Research, V.32, pp.983-998, 1984.
6. Andrew, C. Y., " New Algorithms for Bin Packing ", ACM, Vol.27, pp.207-227, 1980.
7. Coffman, Jr. E. G., So, K., Hofri, M., Yao, A. C., " A Stochastic Model of Bin Packing ", Information and Control, Vol.44, pp.105-115, 1980.
8. Loulou, R., " Probabilistic Behaivour of Optimal Bin Packing Solutions ", Working Paper # 82-29, McGill University, 1982.
9. Friesen, D. K., Langston, M. A., " Variable Sized Bin Packing ", SIAM J. Computing, Vol.15, pp.222-230, 1986.

10. Assmann, S. F., Johnson, D. S., Kleitman, D. J., Lenug, J. Y. T., " On Dual Version of the One-dimensional Bin Packing Problem ", Journal of Algorithms, Vol.5, pp.502-525, 1984.

11. Bruno, J. L., Downey, P. J., " Probabilistic Bounds for Dual Bin Packing ", ACTA Informatica, Vol.22, pp.333-345, 1985.

12. Dudzinski, K., Walukiewicz, S., " Exact Methods for the Knapsack Problem and its Generalizations ", EJOR, Vol.28, pp.3-21, 1987.

REFERENCES NOT CITED

Cook, S. A., " The Complexity of the Theorem-Proving Procedures ", Proceedings Third Annual ACM Symposium on Computing, pp.151-158, 1971.

Karp, R. M., " On the Computational Complexity of Combinatorial Problems ", Networks, Vol.5, pp.45-68, 1975.

Brucker, P., " NP-Complete Operations Research Problems and Approximation Algorithms ", Zeitshrift fur Operations Research, Vol. 23, pp.73-94, 1979.

Lewis, H. R., Papadimitriou, C. H., " Efficiency of Algorithms ", Scientific American, Vol.238, pp.96-109, 1978.

Graham, R. L., " The Combinatorial Mathematics of Scheduling ", Scientific American, Vol.238, pp.124-132, 1978.

Karp, R. M., " Reducibility Among Combinatorial Problems ", Complexity of Computer Computations, R. E. Miller and J. W. Thatcher eds., Plenum Pres, New York, pp.85-104, 1972.