Title: Bin Packing Problem

REGESTRATION NO: - 17BCB0056

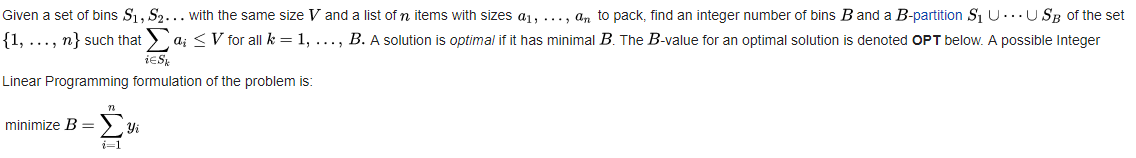
NAME – NACHIKETH JANAPAREDDY

ABSTRACT-

In the **bin packing problem**, objects of different volumes must be packed into a finite number of bins or containers each of volume *V* in a way that minimizes the number of bins used. There are many variations of this problem, such as 2D packing, linear packing, packing by weight, packing by cost, and so on. They have many applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups in media and technology mapping

The problem is an example of an NP-hard Problem, hence algorithms are to be developed keeping in mind to provide if not perfect, optimal solution.

Introduction



Literature review [4]

In this chapter, we recall the literature of the GBPP and of its related problems. These are: the BPP, the VSBPP,etc. The most classical bin packing problem addressed by the GBPP is the BPP. The BPP is the simplest mono-dimensional bin packing problem, introduced by Ullman [1971], which consists in finding the minimum number of bins. In particular, in [Johnson, 1973a], he proposed the Next Fit (NF) algorithm and proved that its performance ratio is 2. In [Johnson et al., 1974], he proposed the First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD), and Best Fit Decreasing (BFD) algorithms and showed that their performance ratios are, 17/10 for FF and BF and 11/9 for FFD and BFD.Yao [1980] presented the Refined First Fit algorithm, with performance ratio 5/3, and proved that any on-line algorithm must have a performance ratio of at least 3/2. Another problem addressed by the GBPP is the VSBPP, where bins with different sizes are available and the goal is to minimize the wasted space. This problem was first investigated by Friesen and Langston [1986]. The authors provided one on-line and two off-line algorithms and proved that their worst case ratio is respectively 2, 3/2, and 4/3 . In the stochastic variant of the problem, named the Stochastic Knapsack Problem (SKP), the source of uncertainty is usually the item profit, and strong hypotheses on its probability distribution are made.

Why Bio inspired algorithm?

Algorithms such as particle swarm optimization and simulated annealing are now becoming powerful methods for solving many tough optimization problems . The vast majority of heuristic and metaheuristic algorithms have been derived from the behaviour of biological systems and/or physical systems in nature. For example, particle swarm optimization [1].

Bat algorithm

In this project I will be focusing on Bat algorithm .The Bat algorithm is nature inspired algorithm proposed by xin-she yang in 2010. The algorithm exploits the so-called echolocation of the bats.[1]

The bat use sonar echoes to detect and avoid obstacles. Sound pulses are transformed into a frequency which reflects from obstacles. The bats navigate by using the time delay from emission to the reflection. After hitting and reflecting, the bats transform their own pulse into useful information to explore how far away the prey is. The pulse rate can be simply determined in the range from 0 to 1, where 0 means that there is no emission and 1 means that the bat’s emitting is their maximum .Bat sends signal with frequency f. Echo signal used to calculate the distance.

All bats use echolocation to sense distance, and they ‘know’ the difference between food/prey. Bats fly randomly with velocity vi at position xi with a fixed frequency fmin, varying wavelength λ and loudness A0 to search for prey.

IDEALIZED RULES OF BAT - They can automatically adjust the wavelength of their emitted pulses and adjust the rate of pulse emission r[0,1], depending on the proximity of their target. Generating new solutions is performed by moving virtual bats. The current global best location (solution) which is located after comparing all the solutions among all the bats. The current best solution according the equation: while At is the average loudness of all the best at this time step. As the loudness usually decreases once a bat has found its pray, while the rate of pulse emission increases, the loudness can be chosen as any value of convenience.

LOUDNESS AND PULSE EMISSION VS ITERATION Frequency [20KHZ-500KHZ] Wavelength [0.7mm-17mm]

Bat algorithm Pseudocode:-

Objective functions f(x),x=[x1,x2,….,xd]T

Initialise the bat population xi (i=1,2,..,n) and vi

Define pulse frequency fi at xi , initialise the pulse rate ri and the

loudness Ai

While (t< Max number of iterations )

Generate new solutions by altering the frequencies

and updating velocities and locations.

If (rand >ri)

Select the solution among the best solution

Generate the local solution among the best solutions

shortlisted

End if

Generate a random solution by flying randomly

If ( rand<Ai & f(xi)<f(x\*))

Accept the new solutions

Increase ri and reduce Ai

End if

Rank the bat and find the best x\*

End while

Post process results and visualisations [6]

Bat Motion

Each bat is associated with a velocity vti and a location xti, at iteration t, in a d-dimensional search or solution space. Among all the bats, there exists a current best solution x\_. Therefore, the above three rules can be translated into the updating equations for xti and velocities vti:

fi = fmin + (fmax-fmin)β\_;

vti= vt-1i + (xt-1i- x\*)fi;

xti= xt-1i + vti;

where β= [0; 1] is a random vector drawn from a uniform distribution.

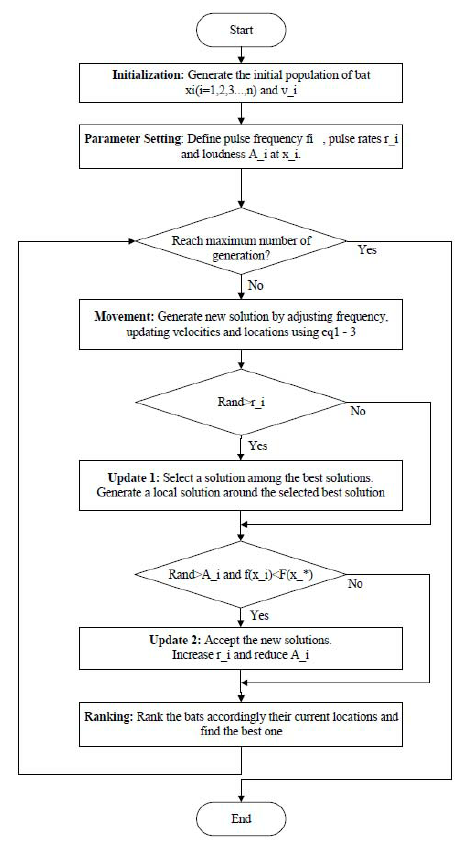
Xin-She Yang (2010) developed the bat algorithm with the following three idealised rules:

1. All bats use echolocation to sense distance, and they also `know' the difference between food/prey and background barriers in some magical way;

2. Bats y randomly with velocity vi at position xi with a frequency f (or wavelength \_) and loudness A0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission r= [0; 1], depending on the proximity of their target;

3. Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) A0 to a minimum constant value Amin.

Flow Chart

.

[7]

Bat Algorithm in Bin Packing (Pseudocode)

->Define the size of the bin (size) and the number of elements (n).

->Set the weights of individual items that has to be stored .

->initialise an empty list of size of the number of elements and i=0.

->for k from 0 to n

->-> set every cell of the array as binsize. (binlist[k] =size)

->while (i<n)

->->for j from 0 to n

->->->if (wt. [i] is lesser than or equal to binlist[j])

->->->->then check for the available bin which can accommodate the items

weight. If the existing bin can’t accommodate create a new bin or

accommodate in one of the existing bins which has remaining

space.

->->->->subtract the weight of the item from the capacity of the bin and

Increment i. if the loop get executed break the for loop.

->for k from 0 to n

->if any of the cells doesn’t have a value that is equal to size of the bin

increment bins.

->The value bins is the number of bins required to store the items

Comparison with the bat algorithm

The random motion of the bats is similar to random weights of the items after the array has been randomised and picked without sorting. Then checking the existing bin having remaining space to nearest location checked by the bats. The BIN PACKING decision problem asks the question whether – given a set of objects of distinct sizes, and a set of bins with specific capacity – there is a distribution of items to bins such that no item is left unpacked nor the capacity of any bin is exceeded. The corresponding optimization problem searches for the minimum number of bins to pack all objects.

Conditions

This solution is for single bat specie.

Weight of the item can’t be more than the capacity of a bin.

CODE FOR BIN PACKING USING BAT ALGORITHM

#include<stdio.h>

#include<stdlib.h>

int a[1];

int n;

int size;

int assignbin(int[],int);

int bininp()

{

printf("enter the size of the bin");

scanf("%d",&size);

return size;

}

void itemsinp()

{

int i;

printf("enter the number of items ");

scanf("%d",&n);

a[n];

for (i=0;i<n;i++)

{

printf("enter the weights ");

scanf("%d",&a[i]);

if(a[i]==0 || a[i]>size)

{

printf("invalid input");

exit(0);

}

}

}

int main()

{

bininp();

itemsinp();

int bins=impliment();

printf("number of bins are : %d",bins);

}

int assignbin(int a[],int l)

{

int i,j,rank,k,insert,bins=1;

int maxpos,maxrank=0;

int binsize[l];

for(i=0;i<l;i++)

binsize[i]=0; //initial

binsize[0]=a[0];

for(i=1;i<l;i++)

{

maxpos=0;

maxrank=0;

for(j=1;j<l;j++)

{

if(a[j]==0)

continue;

rank=size-a[0]+a[j]-j;

if(rank>maxrank)

{

maxrank=rank;

maxpos=j;

}

}

insert=12; //random

for (k=0;k<bins;k++)

{

if(a[maxpos]<= (size-binsize[k]))

{

binsize[k]+=a[maxpos];

insert=-1;

break;

}

}

if(insert!=-1)

{

bins++;

binsize[bins]+=a[maxpos];

}

a[maxpos]=0;

}

return bins+1;

}

int impliment()

{

int x,i,j,z,num,flag,curbin,k=0;

int minbin = assignbin(a,n);

int shuffle[n],arr[n];

for(x=0;x<100;x++)

{

k=0;

srand(time(0)); //initial

while(k<n)

{

for(z=0;z<n;z++)

arr[z]=4689; //random

flag=0;

num=(rand()+0)%(n+1);

for(j=0;j<=k;j++)

{

if(num==arr[j])

{

flag=1;

break;

}

}

if(flag!=1)

{

arr[k]=num;

k++;

}

}

for (i=0;i<n;i++)

{

shuffle[i]=a[arr[i]];

}

curbin=assignbin(shuffle,n);

if(curbin<minbin)

{

minbin=curbin;

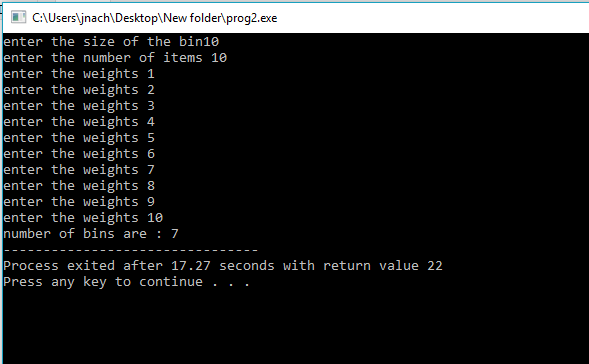
}

}

return minbin;

}

OUTPUT:



Referece

1. A New Metaheuristic Bat-Inspired Algorithm

**DOI** https://doi.org/10.1007/978-3-642-12538-6\_6

**Publisher Name**Springer, Berlin, Heidelberg

1. BISON: A FAST HYBRID PROCEDURE FOR EXACTLY SOLVING THE ONE-DIMENSIONAL BIN PACKING PROBLEM Armin Schollt:l:, Robert Klein§ and Christian Jtirgens

PII: S0305-0548(96)00082-2

1. APPROXIMATION ALGORITHMS FOR BIN-PACKING -E.G. Coffman, Jr. M.R. Garey D.S. Johnson
2. Mauro BALDI\_thesis
3. <https://www.researchgate.net/figure/270592442_fig1_Fig-1-Flowchart-of-BAT-optimization-algorithm>
4. <https://www.researchgate.net/figure/Pseudocode-of-the-bat-algorithm-BA-3_263430240>
5. IJBIC\_bat\_review\_53924\_final