3D bin packing using NSGA-II and Heuristic Decoder

Hamzeh Alzweri†
Computer Science and Engineering
Michigan State University
East Lansing, MI, US
alzwerih@msu.edu

Sinuo Fan
Electrical and Computer Engineer
Michigan State University
East Lansing, MI, US
fansinuo@msu.edu

ABSTRACT

In this paper we are utilizing NSGA-II and the difference process described in [5] and [2], respectively, to solve the 3D bin packing problem. We extend the classical problem objective of saving space and add one more objective that we call priority penalty. This objective is regarding the sort of the items in terms of delivery priority, whichever item that is to be delivered first needs to be packed at the front (beginning) of the container. We use a heuristic decoder to handle all solution feasibility issues related to box placement, e.g boxes overlapping each other, boxes overflowing the container, etc. We show the results we got using the two objective results and visualizations of the phenomes obtained (the actual packed solution).

KEYWORDS

3D bin packing, multi objective optimization, NSGA-II, empty maximal spaces, heuristic decoder.

Introduction

3D bin packing is a classic NP-hard problem, the goal is to place the boxes inside containers while making sure the placement saves as much space as possible. To save space, the specified box should be packed in as few bins or containers as possible, see figure 1. In this project, we will add another goal to the space-saving object. Sort the boxes according to their packing priority (the first box removed from the container). We decided to solve this problem using a heuristic decoder, the reason is to move the feasibility and complexity issues of this problem into the objective evaluation. So, now we can use sequence operators like two point crossover and shuffle indices mutation without worrying about the feasibility of the produced solutions. If we use the GA to change the boxes' locations directly then it's difficult to produce feasible solutions as the initialization and operators need to account for boxes not to overlap with each other in the container space. This makes the problem really complex, and the search space will be difficult to explore properly.

(1)

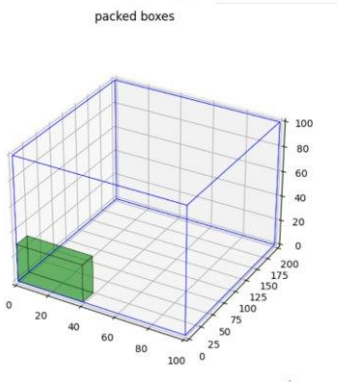
Empty Maximal Spaces

The maximum empty space is the empty space of the cube that is not included in other EMSs and basically indicates where to place the box. The maximum empty slot is created or updated each time a new box is placed, we keep track of a list of all EMSs throughout the decoding process to find empty locations to place the boxes.. When the placed box intersects the EMS, it will be used to create and delete 6 new EMSs. EMS is created in front of, behind, above, below, left, and right at the intersection of the placed box and EMS. After creating a new EMS, some of these EMSs may be infeasible as follows:

1- Infinitely thin EMS : An ems has one of it's dimensions equal to 0, e.g. EMS that is created behind a box when its back is touching another box or the container. See figure 2.

2- An EMS inscribed within another EMS: this is considered a duplicated EMS since I already have an EMS that indicates that the particular space at hand is empty, having another EMS inside of that EMS to tell me that the space is empty there again is redundant so we discard that EMS.

3D bin packing problem in multiple levels using the Genetic Algorithm



(2)

Methodology

We will use the NSGA-II to find the order and orientation of the boxes we have, whereas the heuristic decoder is used to translate this genome (order and orientations) into a phenome (packed solution) in order to evaluate the individual during the NSGA-II.. We move feasibility issues of the problem to a heuristic decoder that will handle the box placement process, this decoder should take as input an individual and return a packed solution (list of containers along with the boxes packed inside of them).

The decoder consists of the following :

1- Placement heuristic function: Try fitting the box in each EMS to determine which EMSs are feasible for this box.

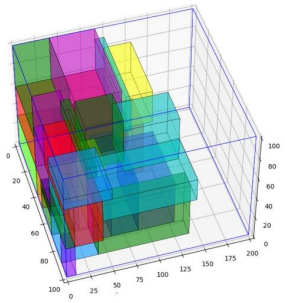
2- Chooses the EMS according to the Back-Bottom-Left (BBL) heuristic rule (choose the spot with the lowest Z, Y, X coordinates in order) and places packs the box in that location.

3- Empty Maximal Space creation function: After packing a box into a location, this function checks which existing EMSs intersect with the packed box in order to create the new EMSs.

4- Empty Maximal Space update function: Because some created empty spaces will have infinite thinness or might be totally inscribed within other EMSs, this function is meant to find such EMSs and remove them as they are unusable and only add overhead to the program in the next iterations. We also make sure to remove EMSs that do not fit any of the remaining boxes, i.e they have less volume than any of the remaining boxes or their dimensions don't fit any of the remaining boxes. We apply those two checks in sequence as in [6], the removal of these EMSs reduces computational complexity as these EMSs won't be used in any future box placement.

Initialization

First, initialize a set of boxes with random dimensions, priorities, weights, and orientations, see figure 3. Create an initial container to use for packing the boxes. The decoder handles opening additional containers when space is needed. Each individual consists of two parts, the first part is the order in which the boxes are packed, and the second part is the orientation of each box. For example: **[3,9,8,10,7,4,6,5,1,2,1,1,1,0,0,0,0,1,2,2, 1]** Here, the first 10 digits are the ID of each box, and the last 10 digits (0, 1, 2) are the arrangement of each box on the left side of the chromosome. The 0 direction means the original direction (no direction) in which the box was created, 1 means the front and back are up and down, and 2 means the sides are up and down.



(3)

Penalty Evaluation

We have created the following merit functions, all of which need to be minimized:

Space Penalty: This function performs really well for the 3D bin packing problem according to [6]. NB means the number of containers, BV is the volume of the last container's box, CV is the volume of the last container. This function represents how well the solution is using the available space (individually). Check only the last container because it contains enough information. This is because if it is better to pack the boxes using the previous container, the last container will receive less boxes. Use NB parts to punish the solution with more containers. If you use only the BV / CV of the last container, you cannot distinguish between a solution with two containers and a solution with three containers, see figure 4.

$$SP = NB + \frac{BV}{CV} \quad (4)$$

$$PP = \frac{\sum_{n=0}^{N} \left| \frac{P-I}{N} \right|}{N} \quad (5)$$

Priority Penality, figure 5, PP is the priority penalty, P is the priority, N is the number of boxes, and I is the index (order) of the person's boxes. This function shows how wrong the priority is. Therefore, there are 50 boxes, with the 50th priority (highest priority) box first. Get 50-0/50 = 1.00. This is the maximum penalty, which is a percentage of how far the boxing order is from

the correct position. We sum this penalty over every box in the chromosome and after that we divide them by N again, and this way, instead of getting some arbitrary number, we get a percentage that represents how off mark the individual is.

Overview

We first Create a set of N boxes with random dimensions, weights, priorities, orientations and IDs. These boxes are to be sorted by the algorithm in containers. Then we initialize the population using the created boxes. Every time we need to evaluate an individual, we need to send it through the decoder first to get our phenome which is fed into our objectives along with the individual. The decoder opens a container and starts the packing process, a placement heuristic is applied to every box in order to determine where to pack it. Then, an EMS creation function is run, this function checks the intersection between the placed box and the EMSs that we already have in order to create new EMSs. After that an update EMSs function is run in order to remove the invalid EMSs we talked about earlier. Another thing that happens in this function is that we also remove the EMSs that don't fit any of the remaining boxes by checking their volume and dimensions. Discarding them saves computational time as they won't be utilized in the future, see [6] for more info. Now we check if there are any remaining unpacked boxes, if there are then we check if we already looped over all boxes in an attempt to place them for this container. If that is the case, then we save this container in our containers list and open up a new container and redo the packing process. If there are no boxes remaining then we exit and return the resulting containers, if there are boxes remaining but we did not attempt all the boxes yet then we simply move to the next box. After we get the containers from the decoder we feed them into the evaluation function which then returns the penalties of the particular individual at hand. We now have the fitness and can proceed with the NSGA-II as follows:

- Assign crowding distance to each individual.
- Start the evolution by applying a tournament selection (R, CD) to create an offspring.
- Use the offspring to create children through crossover and mutation. To do that, split the individual into two chromosomes, first half is the sequence of boxes' IDs and the second half is the orientation bits (rotations). The reason we do this is we apply different operators to each half of the chromosome since they are of different representations.
- Apply Cut and Crossfill crossover and Shuffle indices mutation to the IDs sequence.
- Apply two point crossover and bit flip mutation to the sequence of orientations.
- After creation of children is done, we send them to the decoder, evaluate their fitness and finally combine them with the current population.
- Now individuals are sorted according to the pareto front, then a new population is selecting with the help of TS(R,CD).

For every generation G we keep a reference of the best and worst individual, we keep iterating through the process above until the target number of generations is met. Now the algorithm will return the list of the best and worst individuals across every generation, along with their phenome simulations. For a simple representation of the overview explained above refer to **Appendix 1** at the end of the paper.

Pseudo-code for the decoder

1. Create a set of N boxes with random dimensions, weights, priorities, orientations and IDs.
2. Initialize a population of size M chromosomes using the set of boxes created in step 1.
3. Send all individuals to the heuristic decoder:
    a. While there are still boxes to pack (len(remaining_boxes) > 0):
        i. Create a container with a specified width, height and depth.
        ii. Create an initial EMSs list that have an EMS with the same dimensions as the container created in I.
        iii. For BoxID in individual:
            1. Get box with ID = BoxID from the remaining_boxes pool
            2. Apply rotation to the box according to its orientation in the individual chromosome
            3. Try fitting the box in each of the EMSs in the EMSs list, combine valid EMSs in a list valid_EMSs, if valid_EMSs is empty we skip to the next box, and keep a reference of the current box to pack it in the next container.
            4. Apply BBL heuristic selection on valid_EMSs to select an EMS to pack the box in.
            5. Pack the box in the selected EMSs from iv.
        iv. Add the box to the current container.
        v. Remove the packed box from the remaining_boxes pool.
        vi. For EMS in EMS list:
            1. Check for intersection between placed box and EMS
            2. If there is intersection then use intersection coordinates to create 6 new EMSs (before,

after, above, below, to the right and to the left of the intersection)

3. Add EMSs from ii to EMSs list

vii. For EMS in EMS list:

1. If EMS volume is not bigger than at least 1 remaining box volume, then remove EMS
2. If EMS dimensions do not fit at least 1 remaining box, then remove EMS
3. For target_EMS in EMSs

a. if (x1>x3) and (y1 >ty3) and (z1>z3) and (x2~<x4) and (y2~<y4) and z2 ~< z4) then EMS (x1, y1, z1) (x2, y2, z2) is inscribed within target EMS (x3, y3, z3) (x4, y4, z4) so remove it.

viii. Add container to list of containers.

b. Return list of containers

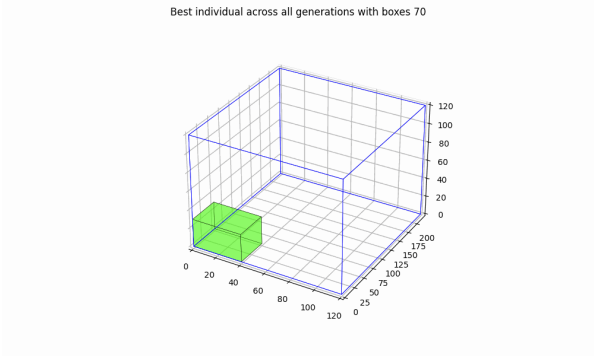
Tableau

Generations	Population size	Crossover rate
50	32	0.55
Mutation Rate	**Mutation**	**Crossover**
0.3	Shuffle indices (for box IDs) and Bit flip (for orientations, here if the bit is 0 it becomes either 1 or 2 and if its 1 it becomes either 0 or 2 and if its 2 it becomes 0 or 1	Cut and cross fill for box IDs and Two-point crossover for orientations.
Parents Selection	**Survivor Selection**	**Termination**
Tournament selection using Rank and Crowding Distance	Select from the pareto front using Rank and Crowding Distance (NSGA-II selection)	Reach target number of generations

Results

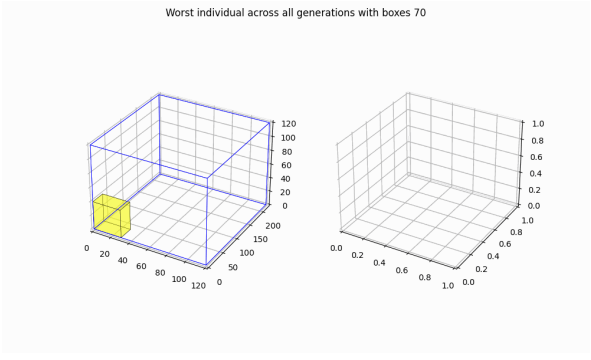
In terms of results, we are testing the solution for every run using 3 different sets of boxes of sizes 70, 140 and 200. We show the visualization of the phenomes (which are genomes that were passed to the decoder) for every different run. We also run the NSGA-II while optimizing for a single objective at a time to check whether the good results we got when optimizing for 2 objectives are legit or not. Our plots show enough evidence that our prioritization of fineness is successful.

In the first test, we run the algorithm using 70 boxes and the visualizations below show the difference between the best and worst individuals' results. The best individual over all generations sorted the 70 boxes into 1 container using 0.856 of its space and with 0.30 (or 30%) off the mark priority. Whereas the worst individual used 2 containers and was 0.37 off the mark. See figures 6 and 7 below for the best and worst individual respectively.



(7)

best individual penalties: (space 1.856, priority 0.308)

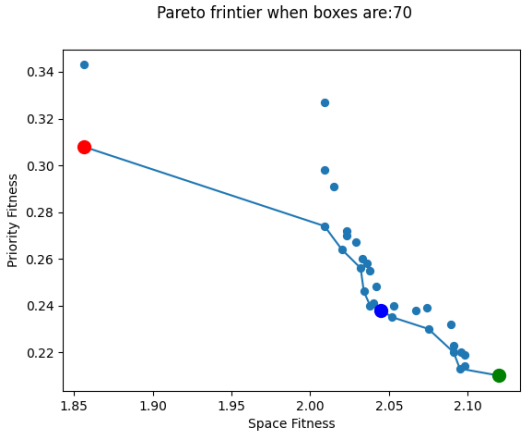


(8)

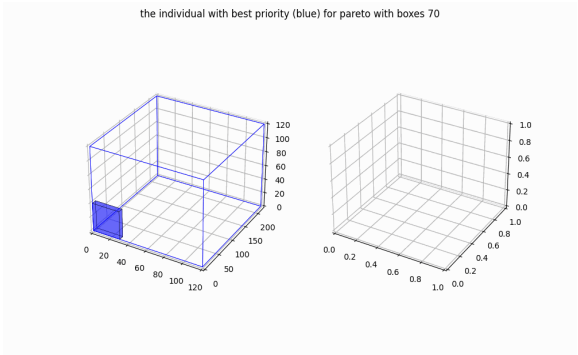
worst individual penalties: (space 2.143, priority 0.37)

The data in figure 9, 10, 11 and 12 show the Pareto graph for a 70 boxes run along with visualization of the colored individuals on the plot, the extremes from both ends and an average individual on the pareto front respectively.

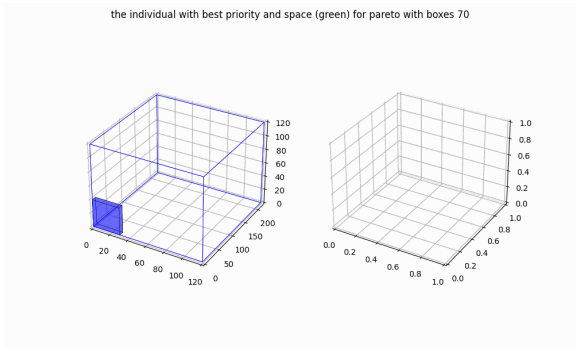
3D bin packing problem in multiple levels using the Genetic Algorithm



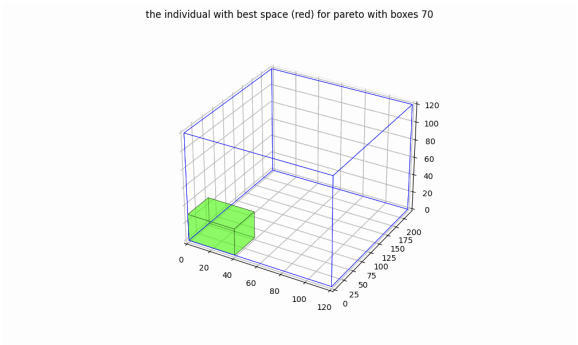
(9)



(10)

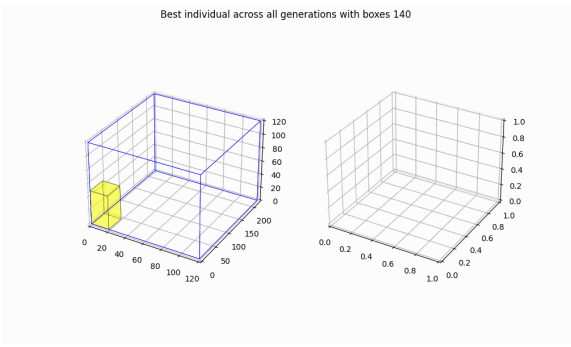


(11)



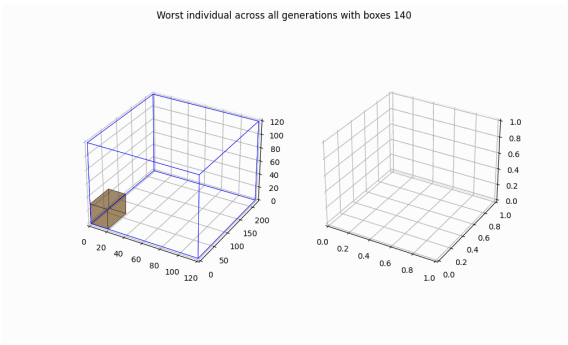
(12)

Now we show the same results as above for 140 boxes:



(13)

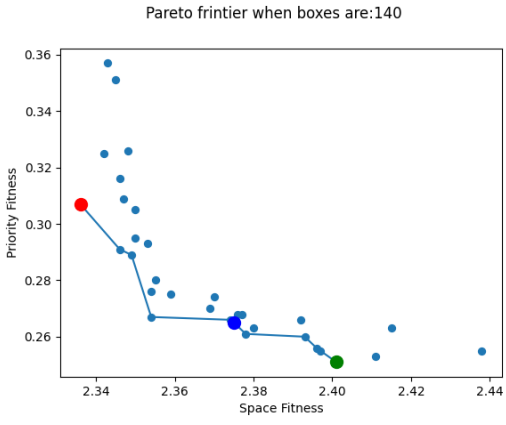
best individual penalties: (space 2.354, priority 0.267)



(14)

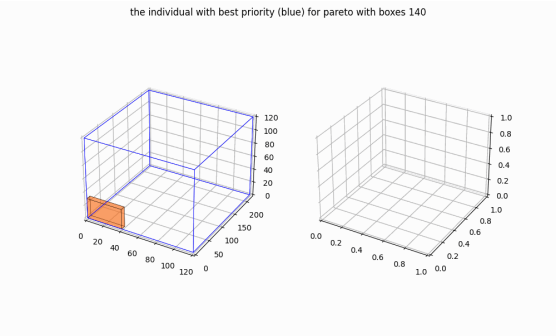
worst individual penalties: (space 2.41, priority 0.354)

And again, the pareto front along with the visualizations of the target colored individuals red, blue green are figures 16, 17 and 18 respectively :

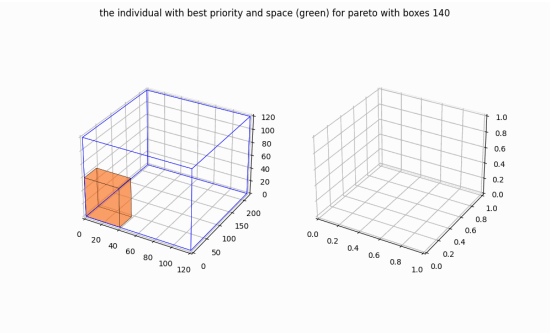


(15)

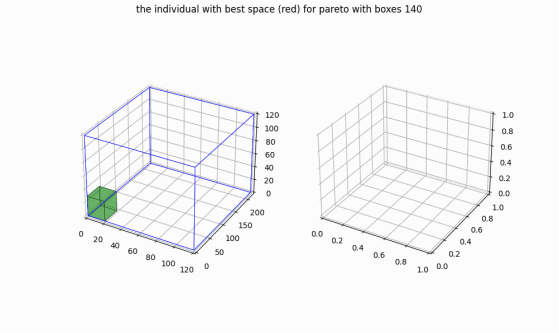
3D bin packing problem in multiple levels using the Genetic Algorithm



(16)

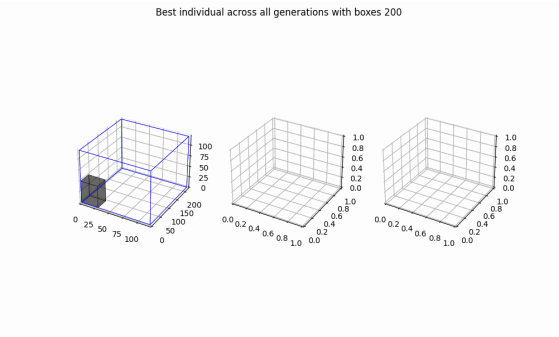


(17)



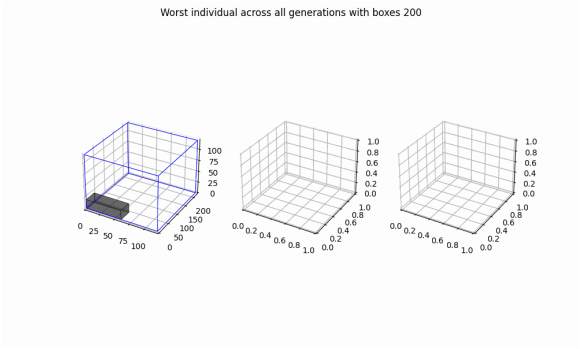
(18)

We run the algorithm for 200 boxes and show the same plots as above in the same order again to get:



(19)

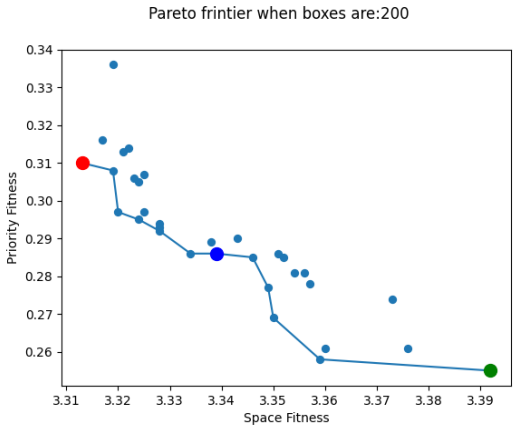
best individual penalties: (space 3.32, priority 0.297)



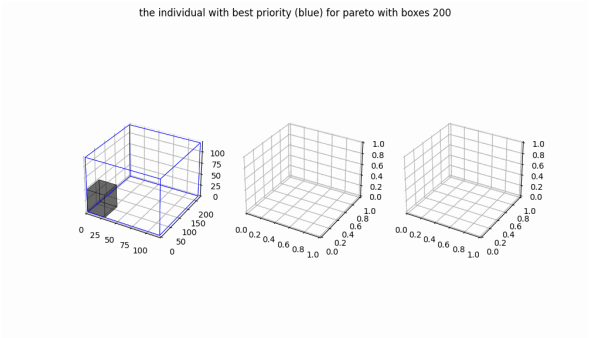
(20)

worst individual penalties: (space 3.448, priority 0.342)

The pareto front and sampled individuals:

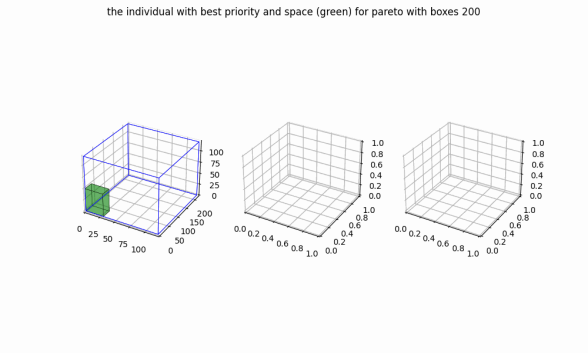


(21)

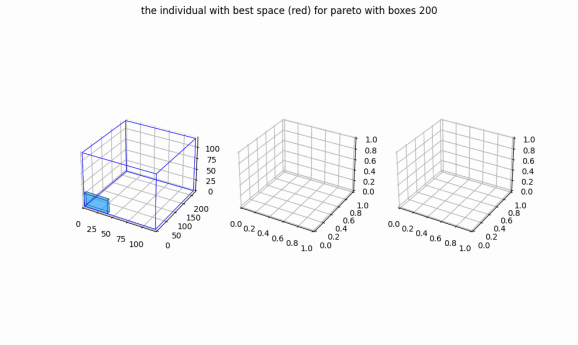


(22)

3D bin packing problem in multiple levels using the Genetic Algorithm



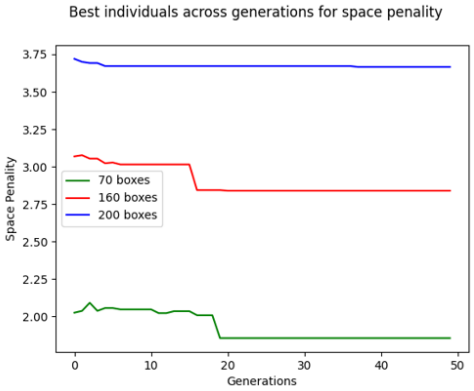
(23)



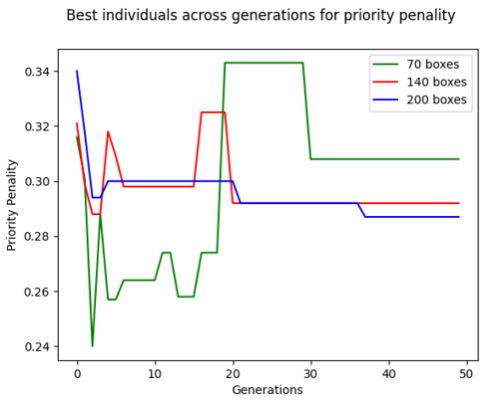
(24)

Scalability

Now to show the scalability of the algorithm we show the objective penalties over generations for the 3 different runs of different box sizes in figure 25 and 26.



(25)

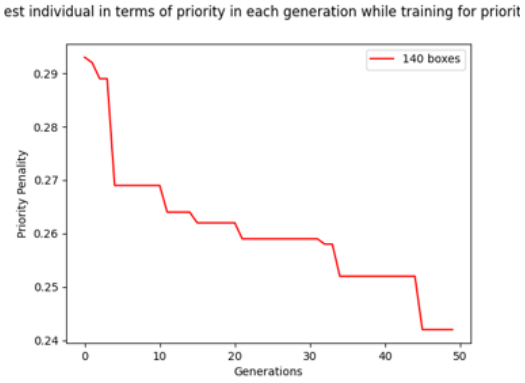


(26)

As we can see from the figures above, the solution wa scalable for all the box size, although it might not be apparent for the space objective run for size of 200 boxes, the space is decreasing slightly but the reason it is not reduced as much is that when you increase the number of boxes you will inevitably need more space to fit them in. This all depends on the sizes of the boxes and the size of the container at hand, so "how good my solution is" really relies on "how good a solution for this set of boxes can be". In general, we say that the solution is scalable as long as it's actually decreasing the penalty even for small amounts.

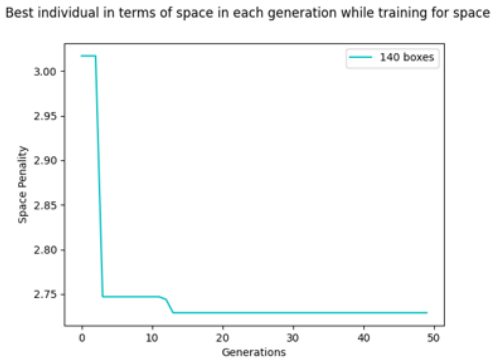
Solutions legibility

In this test, we run the algorithm to optimize for a single objective at a time. We do this because we want to confirm that the solutions obtained above are reasonable, reproducible and not obtained by coincidence.



(27)

The best individuals across all generations in terms of priority.



(28)

The best individuals across all generations in terms of space while optimizing for space penalty.

As we can see from figures 27 and 28, the algorithm is still finding good solutions for each of the objectives separately so we can say that the algorithm will be reaching a minimum that optimizes both the objectives everytime we run it.

Conclusion and future work

Using the NSGA-II and a heuristic decoder got reasonable results for solving the 3D bin packing problem, and the algorithm is scalable for an increased number of boxes.

We haven't compared it to any bench-mark yet, so we can't give a statement on whether it is the best or not at the moment but this is a future consideration. We believe that the performance of this algorithm can be further optimized by trying out different placement heuristics instead of the Back-Bottom-Left heuristic.

In our next work, we will be considering a weight fitness function for stabilizing the boxes in the container (making sure the boxes are placed in a stable way so that they won't fall off
while the containers are in-transit) by assuring the heavier boxes are placed in the bottom and lighter boxes are placed on the top.

REFERENCES

[1] Bean, J., 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 6, 154–160.

[2] Lai, K., Chan, J., 1997. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 32, 115–127.

[3] DEAP. (n.d.). Deap/nsga2.py at master · DEAP/deap. GitHub. Retrieved December 7, 2021, from https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY.

[5] J, Gonçalves. M. Resende. 2013. A biased random key genetic algorithm for 2D and 3D bin packing problems, Int. J. Production Economics 145 (2013) 500–510.

3D bin packing problem in multiple levels using the Genetic Algorithm

Appendix 1

NSGA-II and the Heuristic Decoder workflow