AMS 545 Final Project Report: Packing Anchored Rectangles

Sophia Nolas

5/5/23

1 Introduction

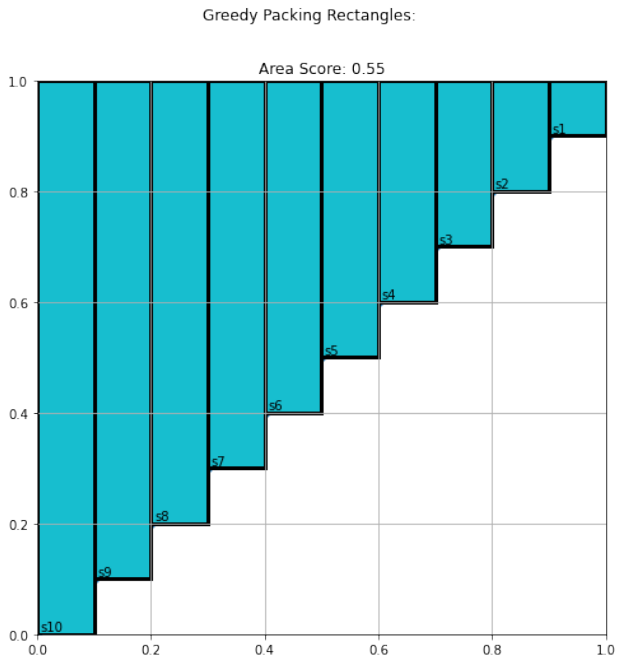
For this problem, I impelmented one of the algorithms in the article "Packing Anchored Rectangles" **Combinatorica - July 2011**; Adrian Dumitrescu (University of Wisconsin, Milwaukee) and Csaba D Toth (California State University, Northbridge). This article proposes two new algorithms to solve the Anchored Rectangle Packing Problem, and analyzes their performance, in order to propose and prove a new minimum coverable area. The problem itself can be expressed in the form of a game, popularized by Peter Winkler:

- Alice chooses n points in the unit square $[0, 0]$x$[1, 1]$ (including the origin)
- Bill chooses an axis-parallel rectangle anchored at each point s $\in$ S – that is, the lowest left point of the rectangle is the point s

A conjecture for this problem is that for any finite set S in the planar unit square, Bill can choose rectangles that cover, all together, half of the unit square (net area = 0.5). This conjecture, despite years of effort, has never yet been rigorously proven. In fact, it has not even been proven that Bill can always score a positive constant area coverage!

For instance, to see that he is not guaranteed more than half coverage, if Alice chooses the points equally spaced along the diagonal, Bill can cover at most $\frac{1}{2} + \frac{1}{2n}$; so if $\epsilon > 0$ is fixed, the constant $\frac{1}{2} + \epsilon$ cannot guaranteed to be covered, by choosing n large enough that $\frac{1}{2n} < \epsilon$.

To visualize this with $n = 10$, you can see that the rectangles chosen using Greedy Packing algorithm cover about half of the unit square:



2 The Algorithms

This paper proposes two algorithms that give Bill a guaranteed score of 0.09121 area covered.

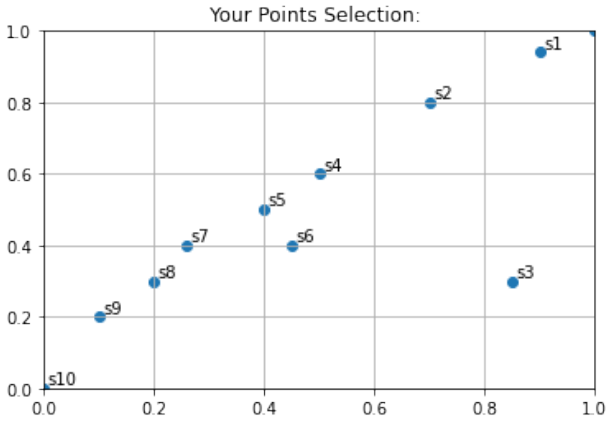
1. GREEDYPACKING
2. TILEPACKING

In this project, I implemented the Greedy Packing algorithm in Python.

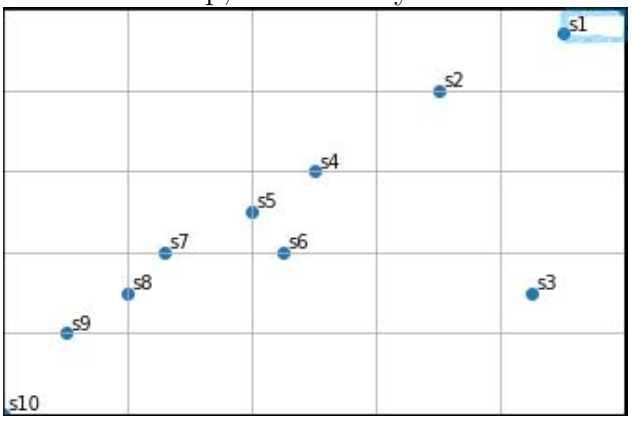
- First, I sort the points in descending order from (1,1) to (0,0) such that for points s_i, s_j, if $i < j$, then $x_i + y_i > x_j + y_j$ (and note that $s_n = (0, 0)$
- Starting with the point furthest from (0,0), s_1, choose the largest rectangle so that s_1 is the lower left vertex.
- For each s_i in increasing order, choose the largest anchored rectangle that does not overlap any previous rectangle.

I chose the largest rectangle at each set by simply naively testing every possible rectangle, that does not overlap a previous one. I choose a height from the current point to the height of any previous point, and draw a rectangle rightward until it hits the x value of another rectangle that is lower than that height.

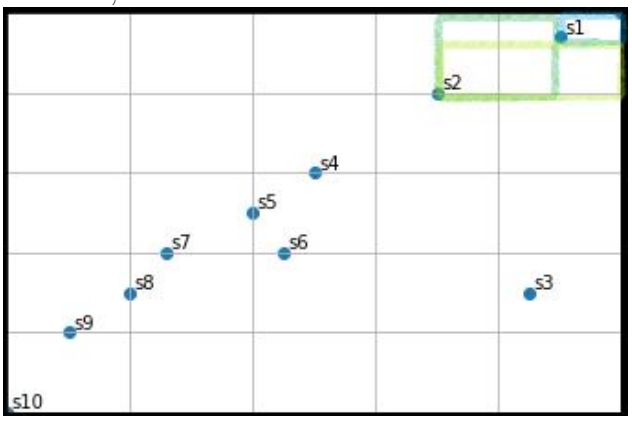
To visualize this, observe the following example. The points are chosen as follows:



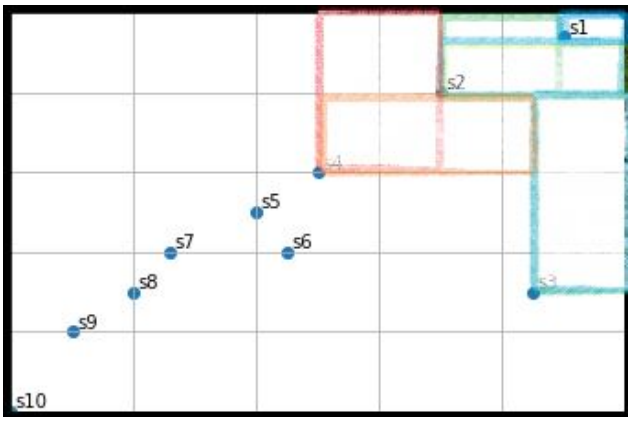
At the first step, there is only one anchored rectangle to choose from.



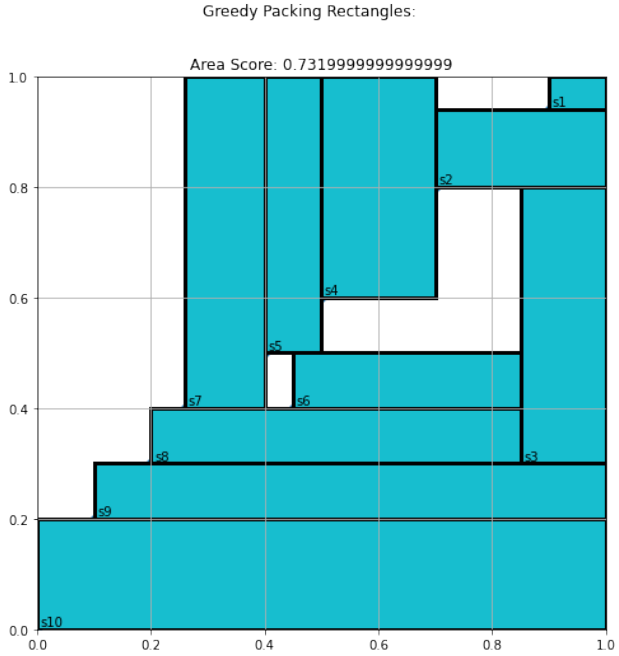
At the next step, the rectangle that is anchored at s_2 can be chosen either to be the one that hits the top, $y = 1$, or the one that hits the right side, $x = 1$. This second option yields higher area, so it is chosen.



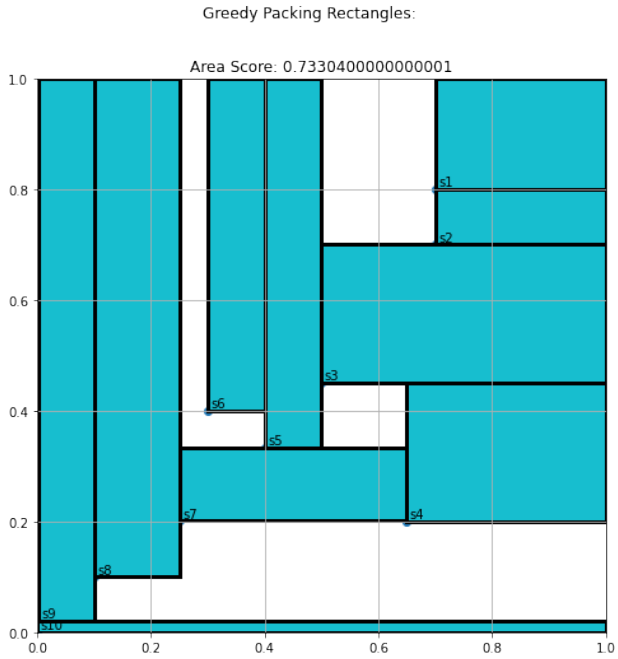
At the next step, the only anchored rectangle possible is the rightward one. Then at the next step for s_4, another comparison must be made, between the "upward" and "sidways" layout of the anchored rectangle; and the decision is made by comparing the areas covered by each.



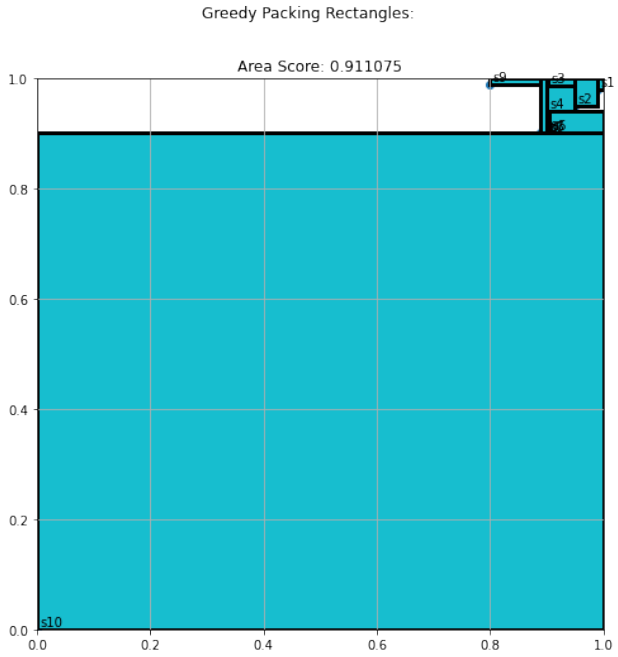
And so on, until all anchored rectangles are chosen. The total area covered is 0.732:

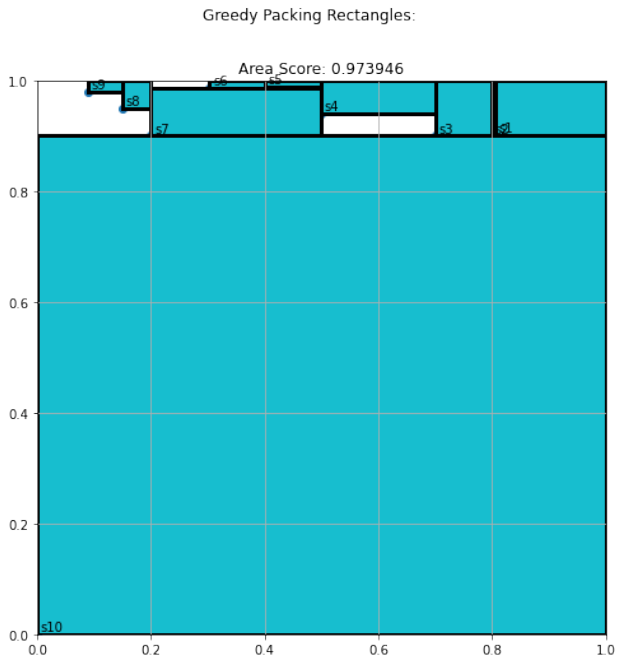


For another example with points close to the diagonal:



And two more examples with points chosen near $(1, 1)$ and near the upper boundary:





Since $(0, 0)$ must always be included, trying to place the points in a tight cluster tends to create at least one large rectangle, giving Bill more area covered; meanwhile it seems that the point set along the diagonal tends to give the most difficulty, leading toward approaching only 0.5 coverage.

The Greedy Packing algorithm, which I implemented here, always yields greater covered area than Tile Packing. The paper "Packing Anchored Rectangles" goes through the proof in a more rigorous way, but the key idea is that the Tile Packing algorithm relies on the (non overlapping) dominant rectangles of each point; which means that the area that each covers will inherently be less than the Greedy Packing rectangles, since these use the same dominant rectangles, while also potentially covering more space, since at each step the actual largest non-overlapping rectangle is chosen, rather than simply one limited in the non-overlapping dominant rectangles. That is, the Tile Packing algorithm chooses tiles in the dominating rectangles; and the greedy packing algorithm chooses rectangles that contain the dominanting rectangles.

Once they have demonstrated that the Greedy Packing algorithm always yields more coverage, the article goes on to prove that the Tile Packing algorithm has a guaranteed coverage bound of 0.09121 (by using properties of the tiles); which means this is also a lower bound for the Greedy Packing algorithm. But, given my limited trials of implementation, it does seem like this bound is potentially (much) higher.

3 Runtime Analysis

For these algorithms, their Time and Space implementation have the costs as follows.

1. Time

   (a) Greedy Packing: sort points in order of distance from origin; for each s_i of n

points, compare $n - i - 1$ rectangles & choose largest; O(n(n-1)/2); total cost of $O(n^2)$.

(b) Tile Packing: sort points in order; compute dominant rectangles; then search through the binary tree holding the dominant rectangles, and choose the largest rectangle in each; for a total cost of O(nlogn)

2. Space: both only need to store the n points, x and y values, so they have a space cost of O(n).

I am also including my implementation code in Python.