Lab Project: Anchored Rectangle Packing

Anton Lorenzen

University of Bonn

June 23, 2022

The problem

Definition

Given a finite set of points $P \subset [0,1]^2$, find a rectangle $R^p = [p_x, r_x^p) \times [p_y, r_y^p) \subseteq [0,1]^2$ for each $p \in P$ such that $R^p \cap R^q = \varnothing$ for all $p \neq q \in P$.

Area: $\sum_{p \in P} (r_x^p - p_x)(r_y^p - p_y)$

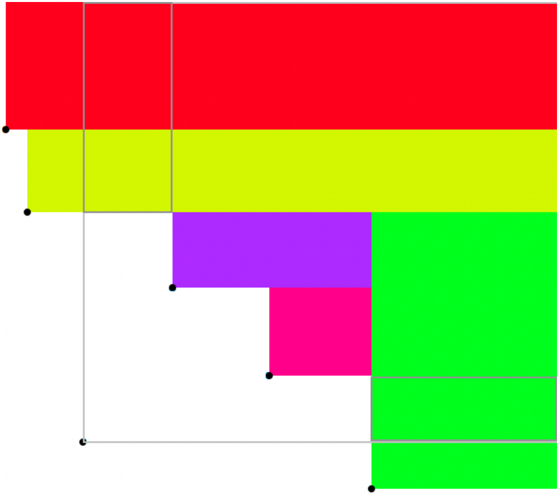
Q: Can you always cover 50% if $(0,0) \in P$?

Why 50% ?

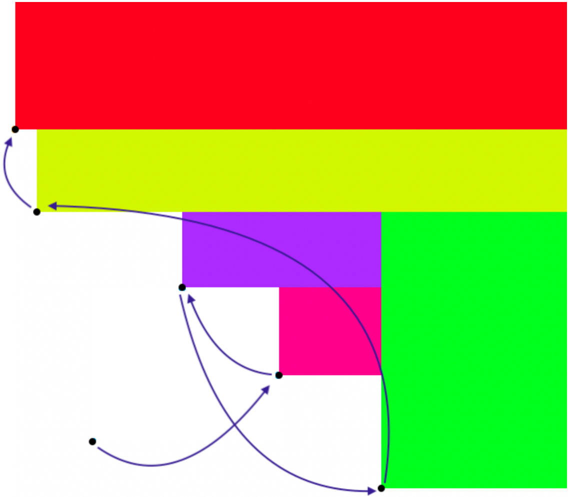
(interactive)

Contents

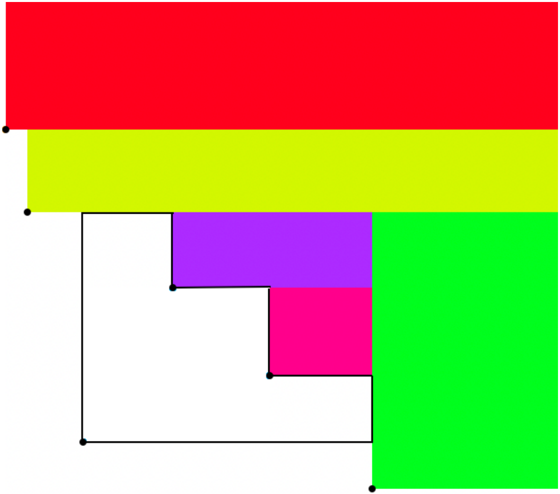
Ordering



Ordering



Tile



Greedy Choices

Basic Algorithm

covered := 0
For all permutations π of P,
    R := pack rectangles greedily in order π
    covered := max(s, coverage(R))
return covered

Dynamic Programming

If $\pi = \pi', x$ is the optimal permutation, then π' is optimal for $P \backslash \{x\}$.

Idea: Inductively compute π for all subsets of P.

Held-Karp Dynamic Programming Solver for TSP $O(2^n n \log n)$.

Improving it with Heuristics

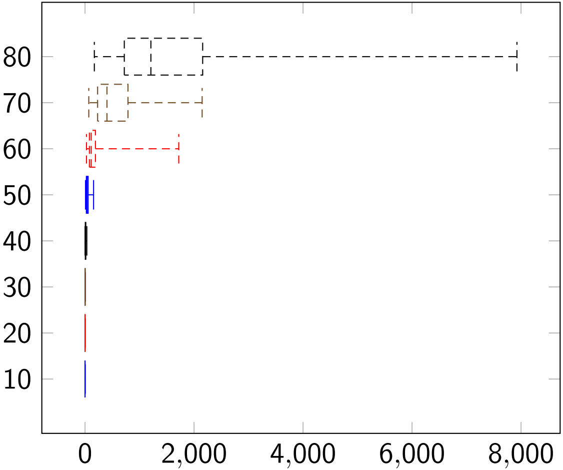


Figure: Time (ms) for the Optimal Algorithm on n uniformly distributed points; 100 instances

TilePacking

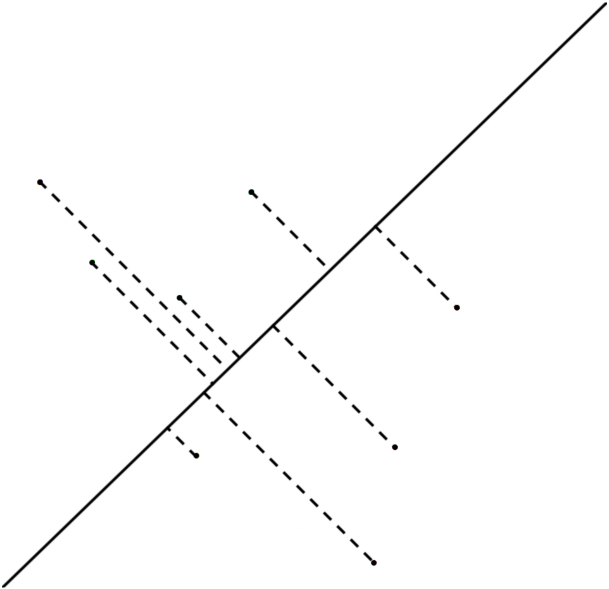
Choose a permutation through a sweep and pack accordingly

Easy to implement in time $O(n \log n)$

Can not cover more than 43% in some instances

TilePacking: Points

TilePacking: Permutation



Problem with TilePacking in Practice

(interactive)

Performance in Theory

GreedyPacking is no better in theory

Reduction by adding 2 points $(x - \epsilon, y), (x, y - \epsilon)$ for each $(x, y) \in P$

Corollary: GreedyPacking can not reach 50% either!

Performance in Practice

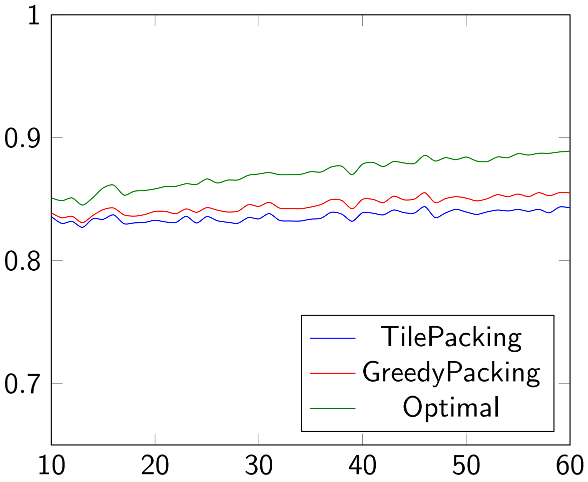


Figure: Average coverage of the algorithms on n uniformly random points; 100 instances

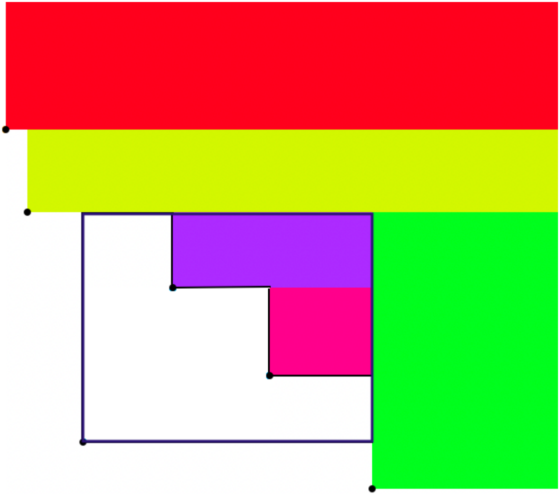
GreedyPacking

Best previous algorithm: $O(n^2 \log n)$ time, $O(n^2)$ space

Implemented: $O(n^2)$ time, $O(n)$ space

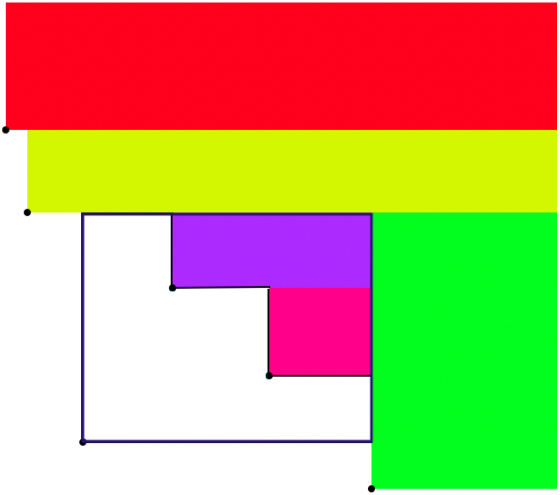
Described: $O(n \log^2 n + k)$ time, $O(n \log^2 n)$ space

Find Tile Rectangle



$O(n)$ time, no extra storage

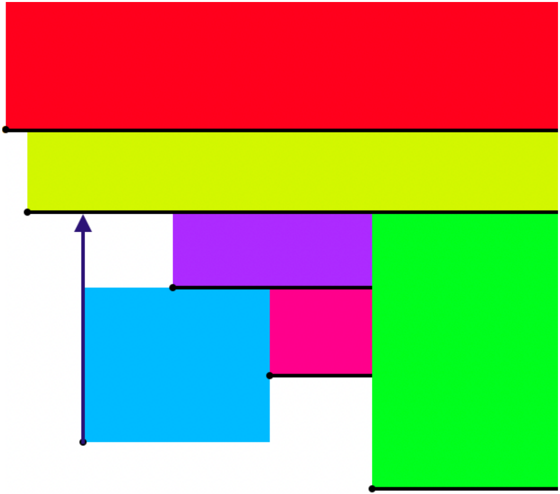
Make greedy choice



$O(n)$ time, no extra storage

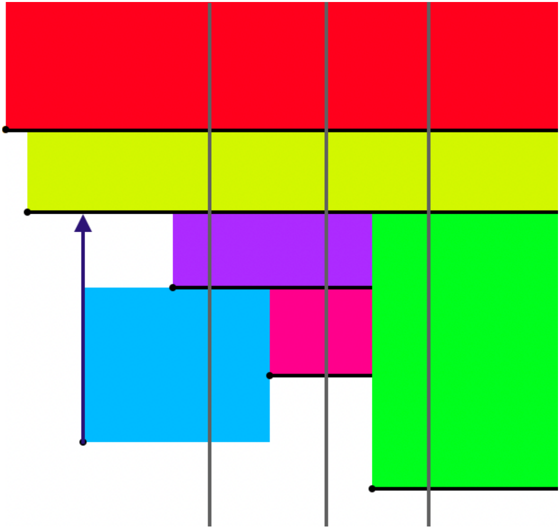
$O(\log^2 n + k)$ time for k points in the tile rect. using a range tree

Find Tile Rectangle (better)



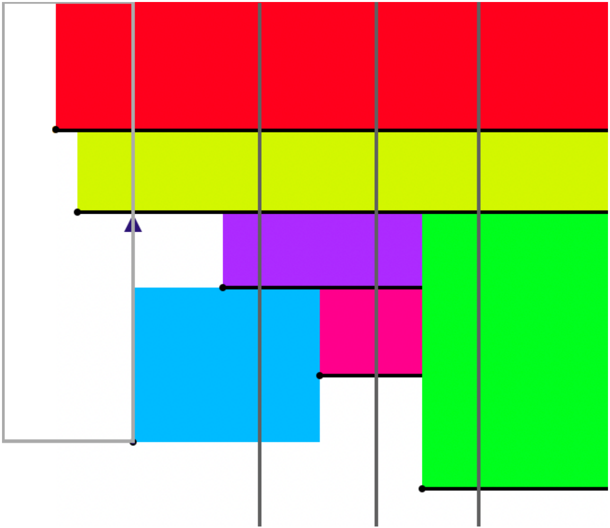
$O(\log^2 n)$ time using interval and priority search tree

Find Tile Rectangle (better)



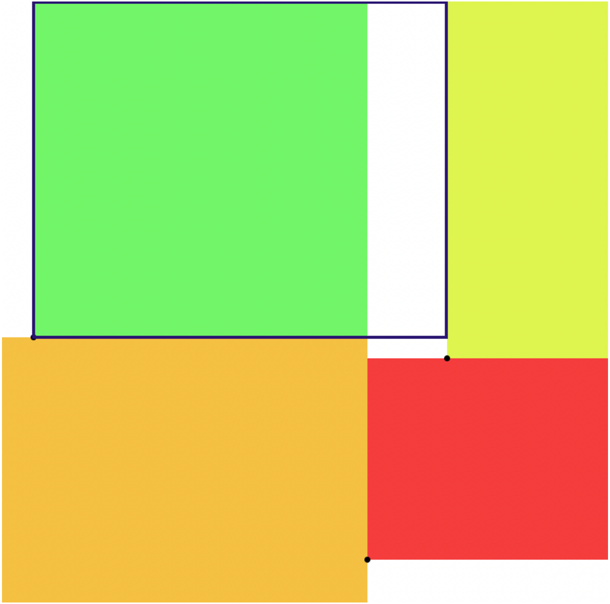
$O(\log^2 n)$ time using interval and priority search tree

Find Tile Rectangle (better)



$O(\log^2 n)$ time using interval and priority search tree

Solves the TilePacking Problem



GreedyPacking: runtime

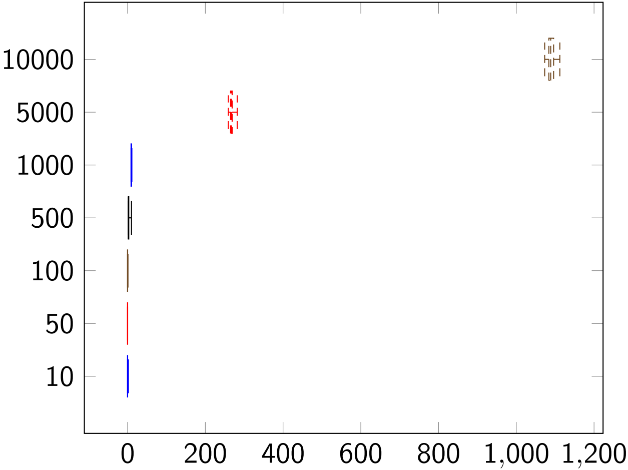
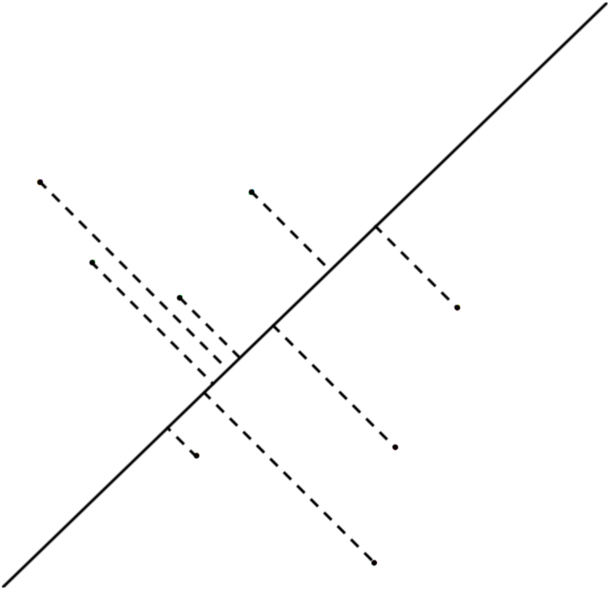
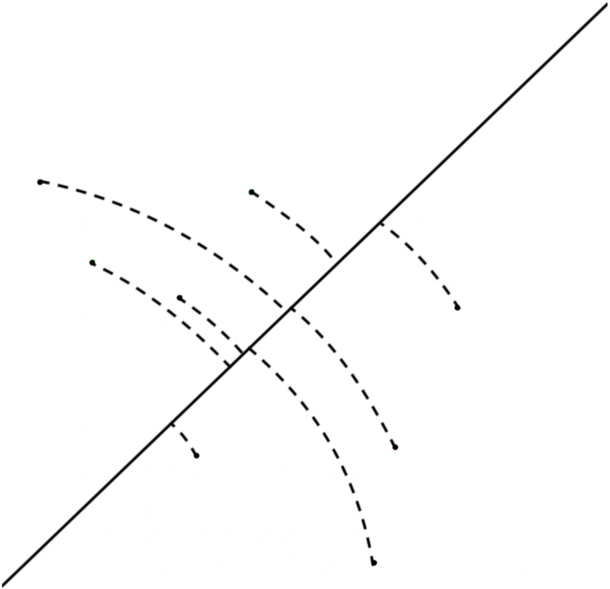


Figure: Time (ms) for the Greedy Algorithm on n uniformly distributed points; 100 instances

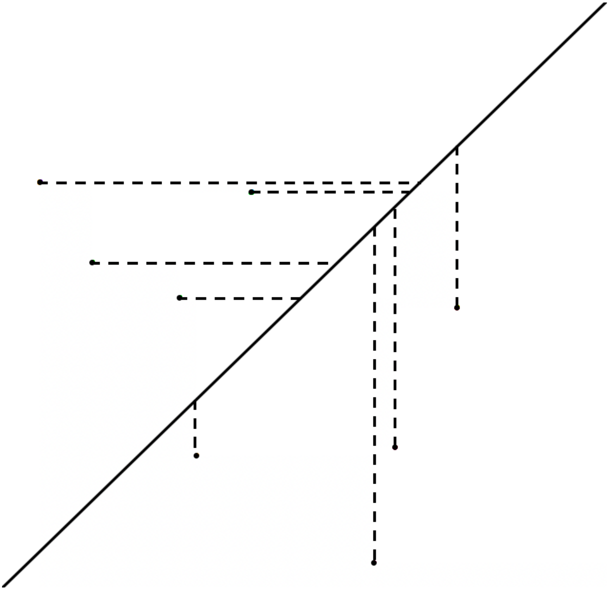
GreedyPacking: Manhattan



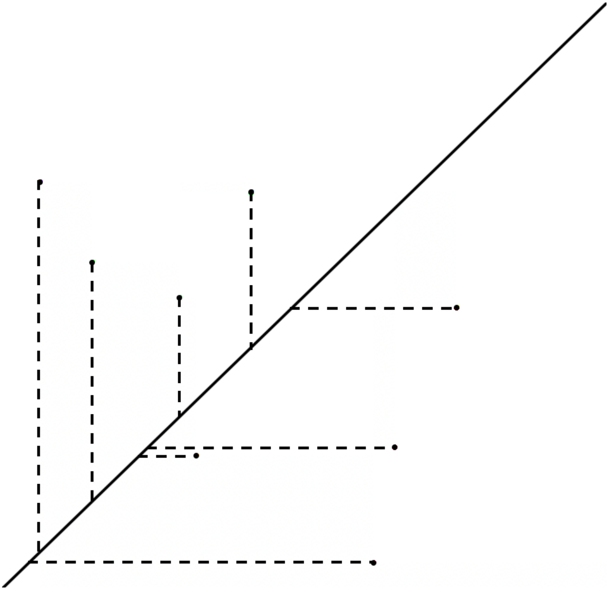
GreedyPacking: Euclidean



GreedyPacking: Infinity



GreedyPacking: Minus Infinity



Norms in Practice

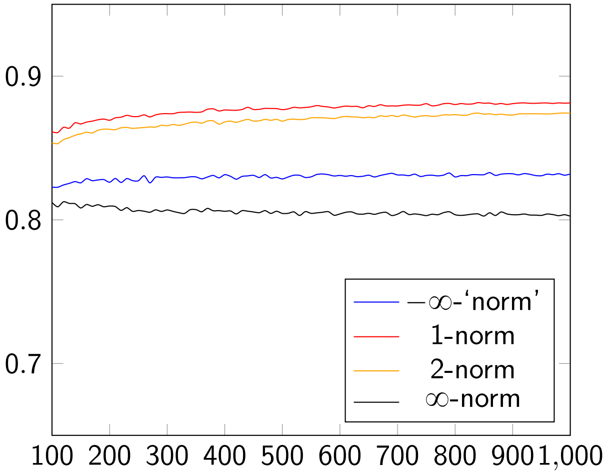


Figure: Average coverage of the greedy algorithm on n uniformly random points; 100 instances

Why are other norms interesting?

No norm is better than any other!

Damerius found no counter-example for the $(-)\infty$-norms

Interactive

Questions?