Geometry Helps to Compare PDs

(part 2)

11/16/2022

Plan:

hand wavy {
- Recall Wasserstein distance
- Combinatorial Methods to compute it (auction algos)
- Geometric Adaptations (k-d trees)
}
- Experimental findings

Bottleneck Distance (∞-Wasserstein)

$$W_\infty(X,Y) = \inf_{\eta: X \to Y} \sup_{x \in X} \|x - \eta(x)\|_\infty,$$

q-Wasserstein Distance

$$W_q(X,Y) = \left[\inf_{\eta: X \to Y} \sum_{x \in X} \|x - \eta(x)\|_\infty^q \right]^{1/q}.$$

Fix $q \geq 1$ and compute q-Wasserstein cost in bipartite weighted graph $G = (U \sqcup V, E, w : E \to \mathbb{R}^+)$

(Idea) Auction Algorithm (Bertsekas, 1974)

$u \in U$ are "Bidders"

$v \in V$ are "Objects"

$w(e)$, $e \in E$, are benefits between Bidders + Objects

price $p_v = 0$ initially

value $(u, v) = (w(u,v) - p_v)$

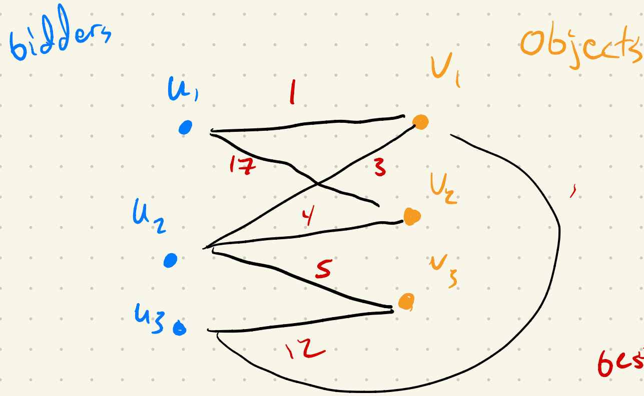
At each iteration, an unassigned bidder u chooses object v with maximal value.

object v is assigned to u.

Let $\Delta p_{u,v}$ be the difference between highest value at U and second highest.

Then $p_v = p_v + \Delta p_{u,v}$ for next iteration.

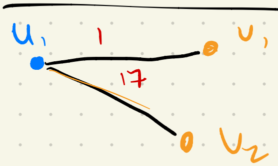
Eg)



best 2nd best value

$17 - 1 = 16$

Iteration 1:



$w(u_1, v_1) = 1$

$w(u_1, v_2) = 17$

$p_{v_1} = 0$, $p_{v_2} = 0$

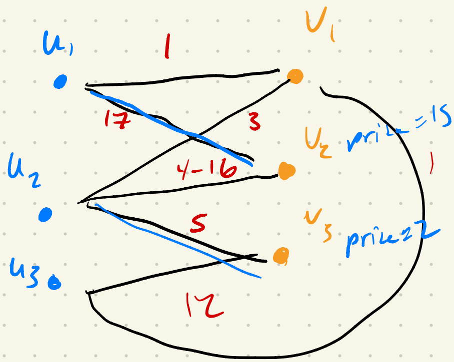
$\text{value}(u_1, v_1) = 1$

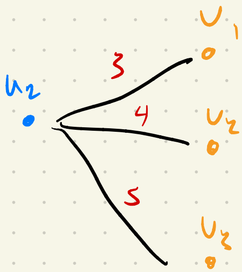
$\text{value}(u_1, v_2) = 17$

choose u_1, v_2.

$p(v_2) = 0 + 16 = 16 + \varepsilon$ for next round.

Iteration 2:





$\text{Value}(u_2, v_1) = 3 - 0 = 3$

$\text{value}(u_2, v_2) = 4 - 16 = -12$

choose → $\text{value}(u_2, v_3) = 5 - 0 = 5$

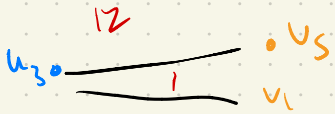
best 2nd best

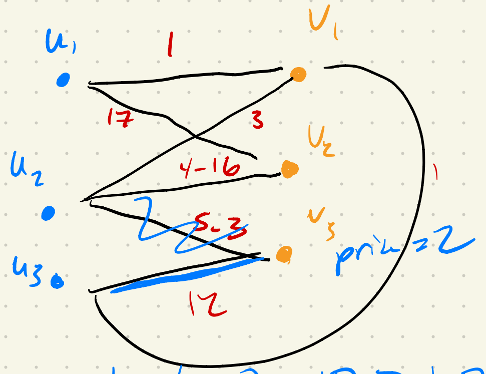
$p[v_3] = 5 - 3 = 2 + \varepsilon$

$p(v_3) = 0 + 2 + \varepsilon$

initial

Iteration 3:



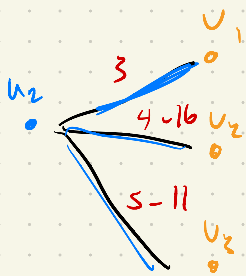


$p_{v_5} = 12 - 1 = 11 + \varepsilon$ for next iteration,

value$(v_3) = 10 > 1 - 0$ = val(v_1)

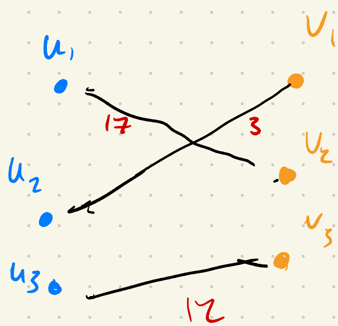
u_2 is now out bid

Iteratation 4:



pair u_2, v_1

and we're done!

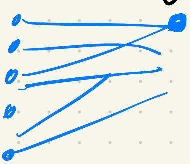


Remarks:

Another version: **Jacobi Auction**, every unassigned bidder makes a bid at each iteration. If several bidders want the same object, it goes to whoever has highest price increase.

This version is more expensive on average

(e.g.) many objects have same price for many bidders.



auction algo from 1974 is way 2 go!

Choosing ε: Note, the smaller the ε, the longer the time of convergence for the auction. But, the more precise a solution.

can compute q-Wasserstein distance with this procedure.

Fixing an approximation parameter $\delta \in (0,1)$, let d denote the qth root of the value of an obtained matching in the algorithm.

Conclude algorithm if:

$$d^q \leq (1+\delta)^q (d^q - n\varepsilon),$$

and return d.

We can turn these matchings into q-Wasserstein distance by taking qth roots + qth powers of L_∞ distances.

Conclude algorithm when matching is optimal, up to error term δ.

Formally,

ALGORITHM 1: AUCTION ALGORITHM

Input: Two persistence diagrams X, Y with $|X|, |Y| \leq n$, $q \geq 1$, $\delta > 0$ (maximal relative error)
Output: δ-approximate q-Wasserstein distance $W_q(X, Y)$
Initialize $d \leftarrow 0$ and $\varepsilon \leftarrow \frac{5}{4} \cdot$ (max. edge length)
while $d^q > (1+\delta)^q(d^q - n\varepsilon)$ **do**
    $\varepsilon \leftarrow \varepsilon/5$
    Let M be an empty matching
    while *there exists some unassigned bidder* i **do**
        Find the best object j with value v_{ij} and the second best object k with value v_{ik} for i
        Assign j to i in M and increase the price of j by $(v_{ij} - v_{ik}) + \varepsilon$
    $d \leftarrow q$-th root of the cost of the (perfect) matching M
return d

What is the crux?

Bidding: Computing max value object + price increase is tricky!

Brute Force: Exhaustive search over all objects per bidder

Lazy Heaps: Keep a heap for each bidder of object values

uses $O(n^2)$ space

Geometry: Use a k-d tree with $O(n)$ space

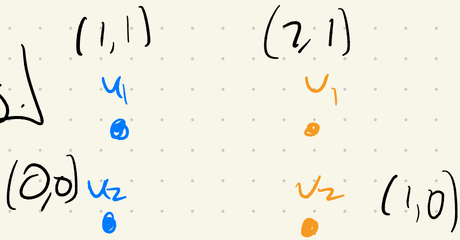
Geometry:

Need to configure a k-d tree

- Initially, when prices are zero,

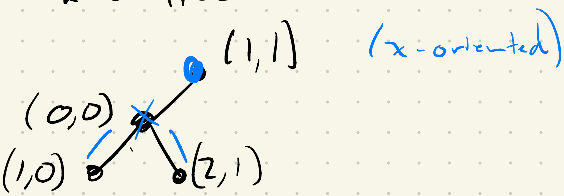
U find best two objects by proximity search.

E.g.



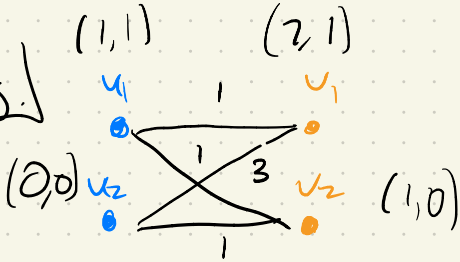
(when all prices are zero)

k-d tree

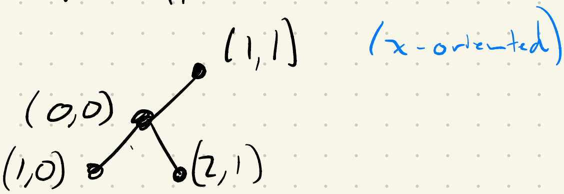


Need to take changing prices into account

Eg.

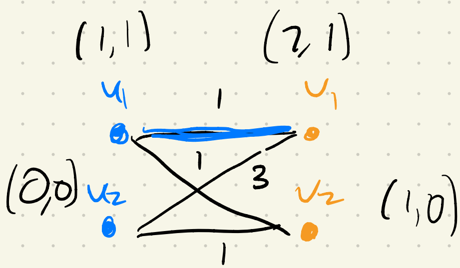


k-d tree

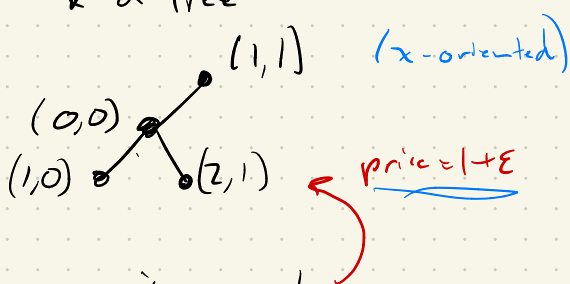


Store prices as weights in k-d tree!

Record minimum weight of any node in subtree of internal nodes.



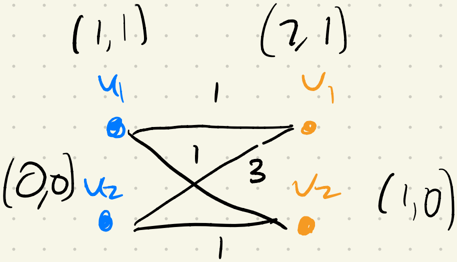
k-d tree



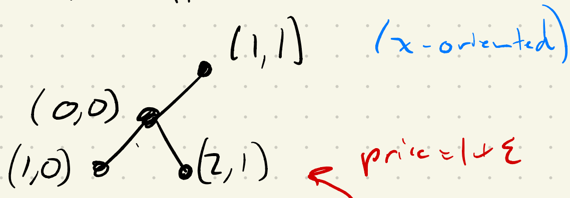
pair (1,1), (2,1)
u_1 v_1

$\text{price}_{u_1} = 1$

value $v_1 = 0$



k-d tree

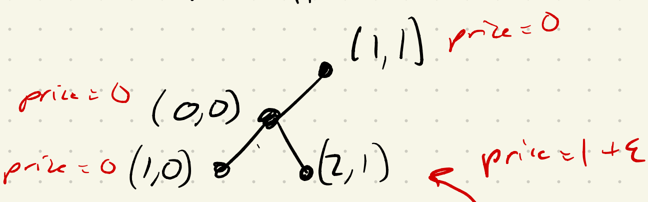


$price_{u_1} = 1$

value $v_1 = 0$

Save minimum weight of any node in subtree of internal nodes

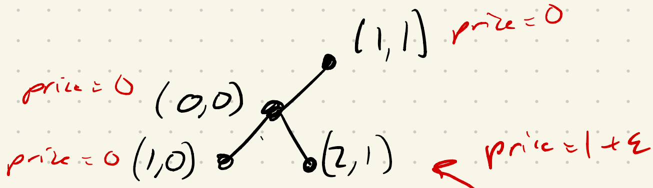
k-d tree



$price_{u_1} = 1$

value $v_1 = 0$

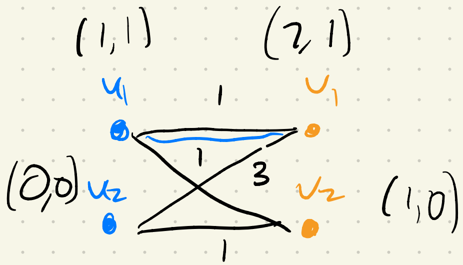
k-d tree

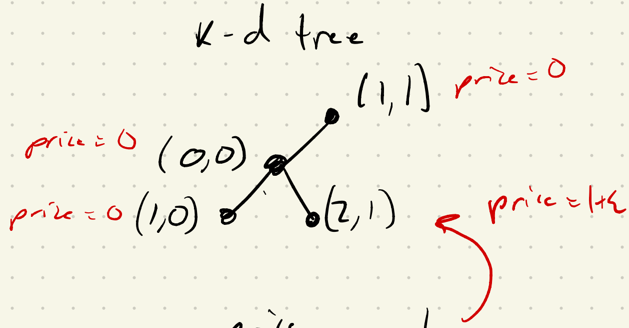


$\text{price}_{u_1} = 1$

value $v_1 = 0$

prune subtrees if qth power of distance from query point to the box containing subtree plus minimum length in subtree exceeds second best candidate





$price_{u_1} = 1$

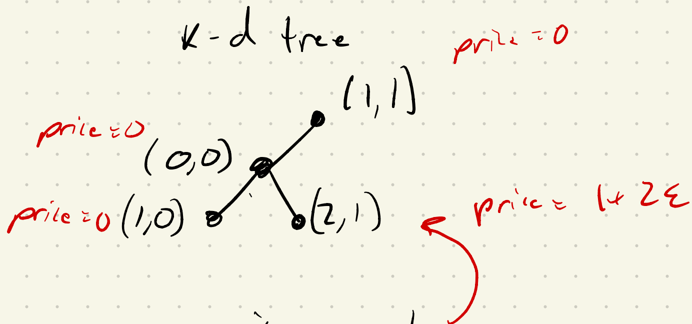
value $v_1 = 0$

Pair

u_2, v_1

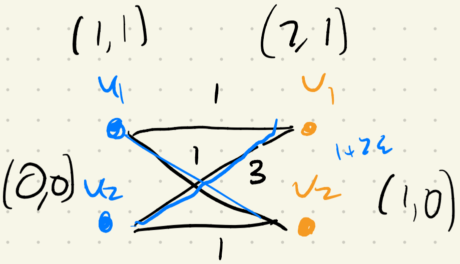
price v_1

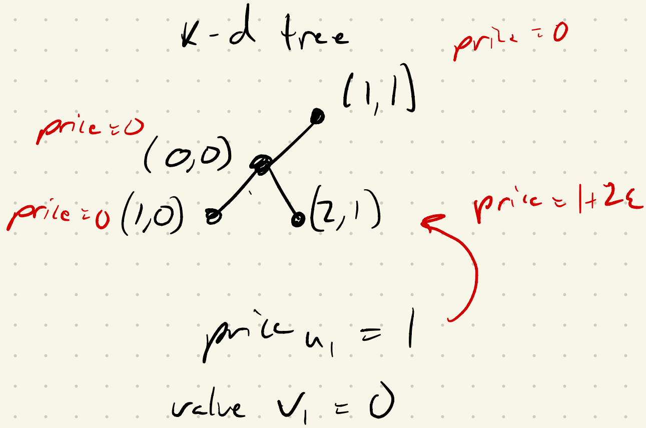
$= 1 + \varepsilon + \varepsilon$



$price_{u_1} = 1$

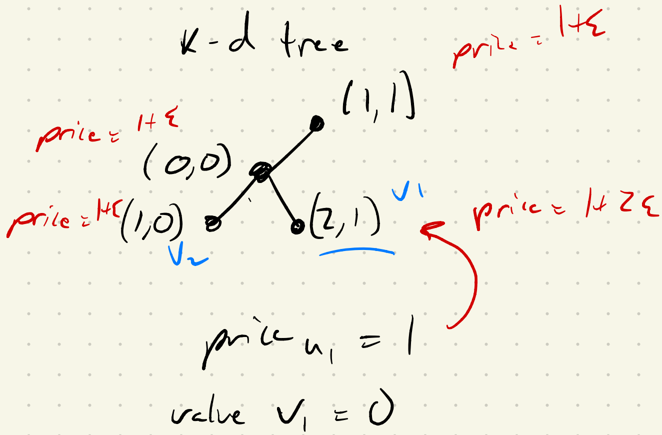
value $v_1 = 0$


(1,1)
(2,1)
u1
v1
1
1+2ε
(0,0)
u2
1
3
v2
(1,0)
1


k-d tree
price = 0
(1,1)
price = 0
(0,0)
price = 0
(1,0)
(2,1)
price = 1+2ε
price_u1 = 1
value v1 = 0

pair

u_1, v_2


k-d tree
price = 1+ε
(1,1)
price = 1+ε
(0,0)
price = 1+ε
(1,0)
v2
(2,1)
v1
price = 1+2ε
price_u1 = 1
value v1 = 0

Persistence Diagrams Need Extra Care

↳ Diagonal bidders can only bid for one object

↳ Off-diagonal bidders can bid for every off-diagonal object but only one on-diagonal object (its projection)

Experimental Results:

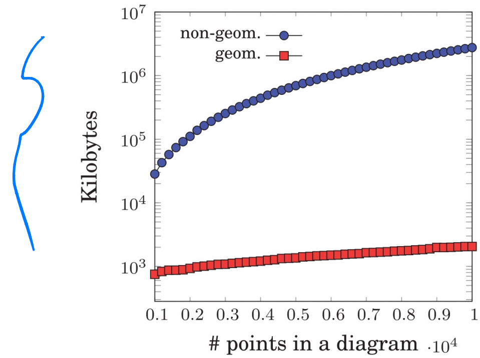


Fig. 6. Comparison of memory consumption of geometric and non-geometric versions of auction algorithm on normal instances.

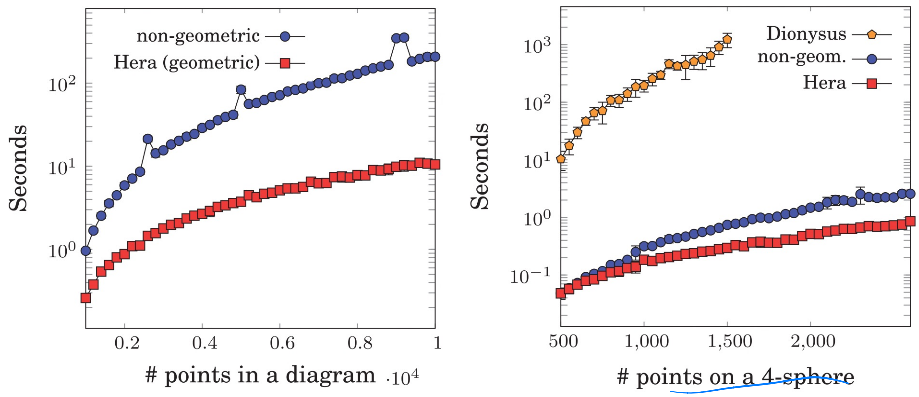


Fig. 7. Comparison of non-geometric and geometric variants of the auction algorithm on normal (left) and real (right) input, also with Dionysus on the real input.

More on repeated points in paper...

Similar techniques implemented using k-d trees.

Takeaways:

- In matching-related problems, k-d trees are a great data structure to use if matching occurs in a metric space
- One can tweak k-d trees to consider changing variables relevant for matching (price, for example)
- The best algorithms on paper might be awful to implement!