No-collision Transportation Maps: Approximating the Optimal Transportation map in $O(n \log n)$ time

Alexander Iannantuono

The University of British Columbia - Okanagan

alexander.iannantuono@ubc.ca

Previously done at McGill University with
Levon Nurbekyan (UCLA) and Prof. Adam M. Oberman (McGill)

Funding by NSERC + FRQNT

March 4 2021

Overview

Original Formulation (by Monge)

Given two seperable metric spaces X, Y that are also Radon spaces, $c : X \times Y \to [0, \infty]$ Borel-measurable, and μ, ν probability measures on X and Y respectively, we seek a transportation plan $T : X \to Y$ such that

$$\inf_T \left\{ \int_X c(x, T(x))\, d\mu(x) \;\middle|\; T\#\mu = \nu \right\}$$

is attained.

Maps, Plans and No-collisions

- What are the differences between OT maps and plans?
- The *no-collision* property is

$$(1-\lambda)x_i+\lambda T(x_i) \neq (1-\lambda)x_j+\lambda T(x_j), \forall x_i \neq x_j \in X, \lambda \in (0,1)$$

- Both OT maps and plans are guaranteed to have this property

Our Question: *Is it possible to obtain a 'good' approximation that is easy to compute and ensures the no-collision property?*
In other words, how 'well' can we approximate

$$\inf_{T\#\mu=\nu} \int c(x, T(x))$$

such that the no-collision property is satistfied for all pairs $(x, y) \in X^2$. If so, how fast?

The Assignment Problem

In combinatorial optimization, the assignment problem is discrete:

$$\arg \min_{f:X \to Y} \sum_{i=1}^{n} C_{x_i, f(x_i)}$$

where C is some associated cost matrix that denotes the cost of 'worker' $1 \leq i \leq |X|$ doing task $1 \leq j \leq |Y|$
In our discrete case, we end up constructing a bijection $f : \mathbb{N} \to \mathbb{N}$ that pairs points via an *ordered* index $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. It is defined explicitly by its function values.

Current Algorithms and an Idea

- The Hungarian algorithm solves the assignment problem **exactly** in $O(n^3)$ time, but does not guarantee no-collision
- Lexicographical ordering in $\mathbb{R}^k$ provides the no-collision but isn't optimal...
- Turns out there is (sometimes) something in the middle!
- Discrete case idea: Build k-d trees for source and target points...with a catch: we need 2^l points to form a complete binary tree
- What about if we don't have a power of 2?

Applications

While the map provided is (almost always) not optimal, it can be useful in domains where:

- Hidden costs exists are more important than costs being considered objectively (flying drones...we don't want crashing)
- Need quick re-computation/adaptation over optimality (can be done since $O(n \log n)$)

(Discrete) Algorithm Description

The idea is as follows:

- Given two sets $X_1, X_2 \subset \mathbb{R}^k$, $n = |X_i| = 2^l$, $l > 0$, build two k-d trees (for each set)
- Keep track of positions of points in the (resp.) trees
- Pair points via their location in the tree
- Possible to do this we will have a complete binary tree of height N, each leaf in the tree will be a singleton

This actually guarantees the no-collision property. A proof can be found in the paper.

Pseudo-code

Algorithm 1: k-d Tree Building

Result: Discrete Bijection between sets $X_1, X_2 \subset \mathbb{R}^k$

Initialize registry $R = (\texttt{id, val, point})$ of size $n = 2^l$;

for $X = X_i, i = 1, 2$ **do**

  m = median(X, j = 0);

  $X_< = \{x \in X | x_j < m\}$, $X_> = \{x \in X | x_j > m\}$;

  for $x \in X_<$, $y \in X_>$ **do**

    $2 \cdot \text{val}(x)$;

    $2 \cdot \text{val}(y) + 1$;

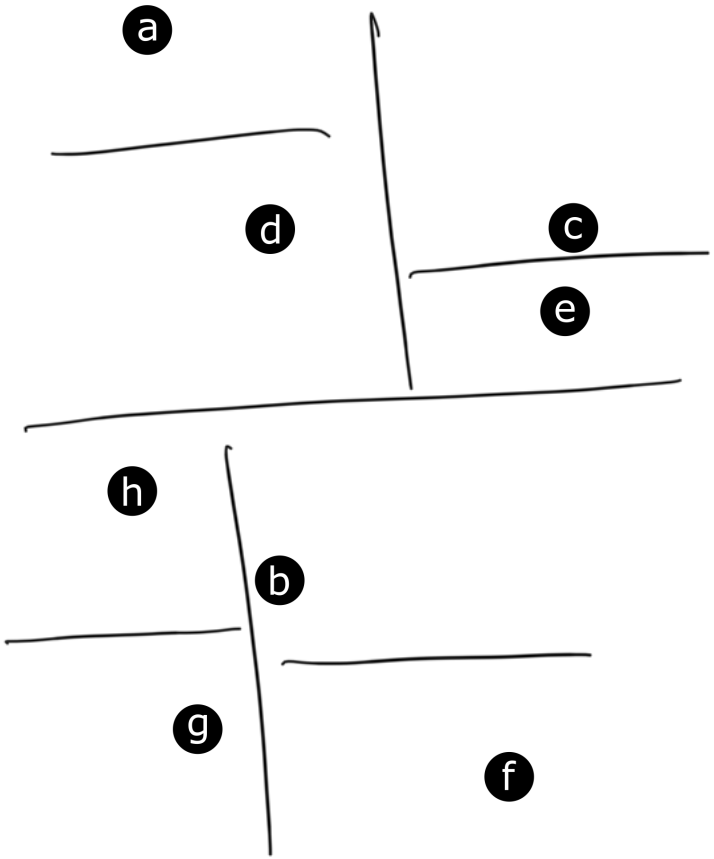
  end

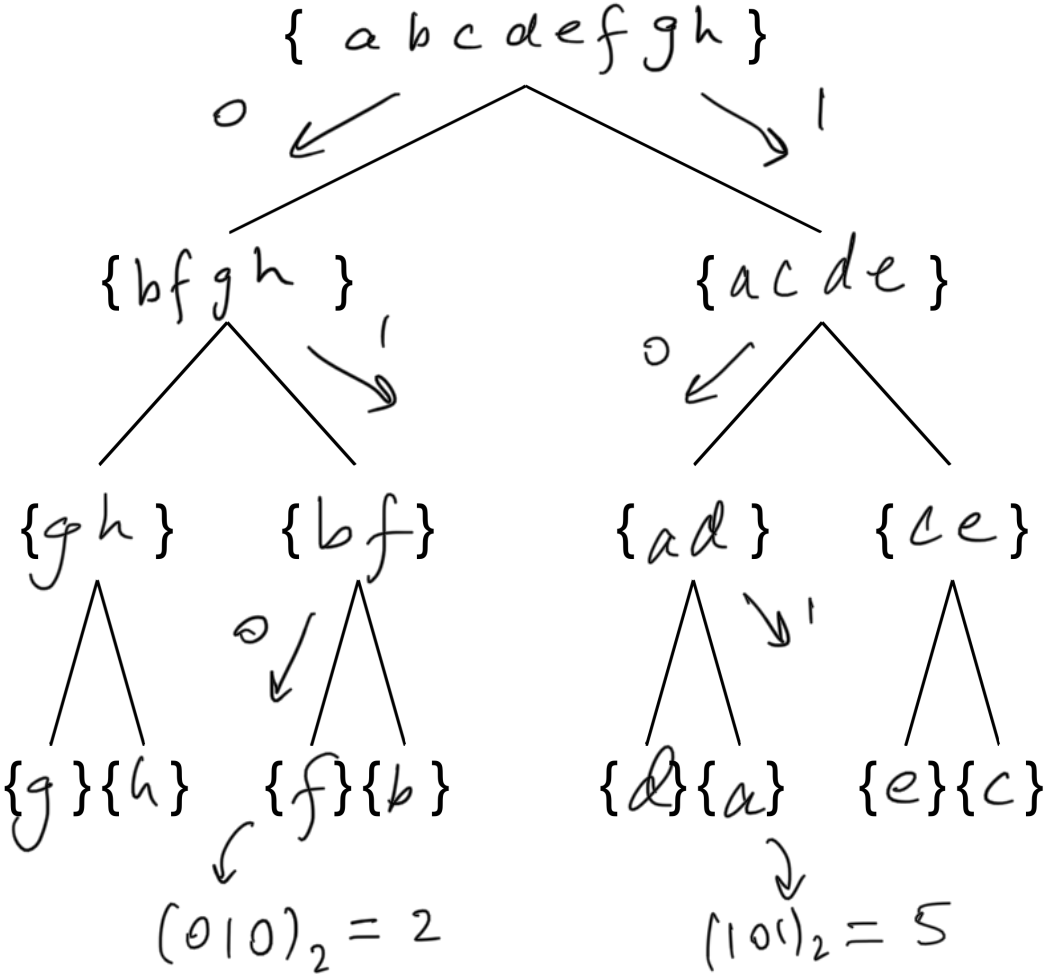
  j = (j + 1) % k;

  Recursively act on $X_<, X_>$ until $|X_\sigma| = 1, \sigma \in \{<, >\}^l$

end

A Step-by-step Example on 2^3 Points in $\mathbb{R}^2$





Algorithmic Analysis

- For each $x \in \mathbb{R}^k$, store a triple `(id, value, x)` with $2n = 2 \cdot 2^l$ points, so $O(n)$ space i.e. linear
- It is known that k-d tree building can be done in $O(n \log n)$ time (need $O(n)$ median finding algorithm).
- Can be seen via the closed form of $\sum_{i=1}^{l} 2^i = 2^{l+1} - 1$.
- Pairing up the points can be done in $O(n)$ time

Some Comparative Results

Cost	Method		n = 64		256		1024		4096	
$\|\cdot\|_2^2$	OT		0.97		1.02		1.23		1.14	
	HV	(ratio)	1.49	(1.54×)	1.23	(1.21×)	1.32	(1.07×)	1.16	(1.02×)
	SH	(ratio)	0.98	(1.02×)	1.05	(1.03×)	1.26	(1.03×)	1.16	(1.02×)
	LEX	(ratio)	6.52	(6.72×)	6.59	(6.46×)	7.00	(5.72×)	7.13	(6.27×)
$\|\cdot\|_2$	OT		1.01		0.77		0.78		0.84	
	HV	(ratio)	1.15	(1.13×)	0.83	(1.08×)	0.81	(1.04×)	0.85	(1.02×)
	SH	(ratio)	1.04	(1.02×)	0.80	(1.03×)	0.81	(1.04×)	0.87	(1.04×)
	LEX	(ratio)	2.25	(2.22×)	2.18	(2.83×)	2.27	(2.92×)	2.31	(2.75×)
$\|\cdot\|_1^2$	OT		1.27		1.47		1.22		1.09	
	HV	(ratio)	1.74	(1.38×)	1.75	(1.19×)	1.37	(1.13×)	1.15	(1.05×)
	SH	(ratio)	1.28	(1.01×)	1.49	(1.01×)	1.25	(1.03×)	1.10	(1.01×)
	LEX	(ratio)	9.42	(7.44×)	10.37	(7.07×)	11.27	(9.27×)	11.13	(10.21×)
$\|\cdot\|_\infty^2$	OT		1.39		1.23		1.06		1.00	
	HV	(ratio)	1.79	(1.29×)	1.49	(1.21×)	1.19	(1.12×)	1.05	(1.05×)
	SH	(ratio)	1.41	(1.01×)	1.26	(1.02×)	1.08	(1.03×)	1.03	(1.02×)
	LEX	(ratio)	6.58	(4.75×)	5.97	(4.85×)	5.89	(5.57×)	5.97	(5.94×)

Table: Average costs of sets of points sampled from $\mathcal{N}(0,1)$ to a new set of points sampled from $\mathcal{N}(0,1)$ (then transformed to a 3:1 aspect ratio and rotated 90° counter-clockwise) measured in various cost functions

Total cost is calculated as $\sum_{x \in X} \|x - T(x)\|_p^q, q = 1, 2, p = 1, 2, \infty$

Strengths and Shortcomings

Strengths:

- Fast
- Easily implementable in the discrete case
- Divide-and-conquer allows for (computational) parallelization
- Does not suffer from the *curse of dimensionality*

Shortcomings:

- Limited insight as to why it provides good approximations
- Limited to maps and no plans (for now at least)
- If $N = 2^l + 1$ for some l, we need to introduce 'fake' points to increase the problem size to $\tilde{N} = 2^{l+1}$
- Can perform poorly on some problems

References

Levon Nurbekyan, Alexander Iannantuono, Adam M. Oberman (2020)
No-collision Transportation Maps
Journal of Scientific Computing 82, 45.

Thank you.