On the Computation of the Kantorovich Distance for Images

Constantinos Alexopoulos[1] and Vassileios Drakopoulos[2]

[1] University of Athens, Panepistimioupolis, 15784 Athens, Greece
(E-mail: calexop@di.uoa.gr)
[2] University of Athens, Panepistimioupolis, 15784 Athens, Greece
(E-mail: vasilios@di.uoa.gr)

Abstract. We consider the theory and applications of the Kantorovich metric in fractal image compression. After surveying the most important approaches for its computation, we highlight its usefulness as a mathematical tool for comparing two images and improve its performance by means of more appropriate data structures.
Keywords: Fractals, Hutchinson metric, Image comparison, Kantorovich metric.

1 Introduction

In many fields of computer science like pattern recognition and image processing, it is important to have an efficient way to compare geometric objects. The natural approach to this problem is to define a metric in the space of the geometric objects and use this metric to compute the distance between them. Considering digitized images as geometric objects, we can use that metric to compare them.

The *Kantorovich* (or *Hutchinson*) *metric*, a.k.a. Wasserstein (or Vasershtein), earth mover's or match metric, takes into account the spatial structure of the compared images and, hence, corresponds more closely than other metrics to our notion of the visual differences between two images. John E. Hutchinson[6] used the Kantorovich distance to measure the distance between self-similar probability measures obtained as limiting distributions for a fairly simple type of Markov chains induced by affine, contractive mappings. He used the Kantorovich metric to prove an existence and uniqueness theorem of such limit measures.

The Kantorovich metric is also used by Michael F. Barnsley[2] and coworkers to approach the convergence of *iterated function systems*, which were introduced by Hutchinson. In trying to solve the so-called "inverse problem" or "image encoding problem", i.e. find an IFS that generates a predetermined image, it is natural to use this metric as an objective function to be minimised. Moreover, this metric appears to be a good indicator of the perceived difference between two images.

Considering digitized images as a set of pixels, the problem of computing the Kantorovich distance between them is equivalent to the formulation of a *linear programming problem* called the *balanced transportation problem.*

According to Michael Werman *et al.*[9] the computational complexity of standard algorithms for transportation problems are of order $O(N^3)$, where N denotes the total number of pixels in the compared images. An algorithm for the computation of the Hutchinson metric in the case of finite one-dimensional sequences is presented in [3].

Thomas Kaijser[7] presented a variation of the primal-dual algorithm for computing the Kantorovich distance function. To decrease the computational complexity for updating the values of the dual variables for both transmitting and receiving images, he always increases them by a constant value of 1. Unfortunately, this is applicable, only if the underlying pixel distance value is the L_1-metric. Moreover, he developed two methods for fast determination of new admissible arcs, one for the L_1-metric and one for the L_2-metric. Kaijser's method was implemented by Niclas Wadströmer[8] in the context of his PhD thesis, but the data structures used to implement the above mentioned method as well as the way that the labelling procedure was implemented are not so clear.

Another work on the computation of the Kantorovich distance is the one of Drakopoulos V. *et al.*[5]. In this work the problem of computing the Kantorovich distance is transformed into a linear programming problem which is solved using the simplex method. To decrease the computational complexity of the method, they developed an approximation algorithm for "large images". Yuxin Deng *et al.*[4] give a brief survey of the applications of the Kantorovich distance in probabilistic concurrency, image retrieval, data mining and bioinformatics.

The main purpose of the present paper is to improve the algorithm presented by Thomas Kaijser for computing the Kantorovich distance function by means of more appropriate data structures. The metric we are using as the underlying distance-function between pixels is the L_1-metric. Using kd-trees we don't have to use different methods, but only to change the metric for the construction of the appropriate kd-tree.

2 Problem formulation

We are interested in computing the *Kantorovich distance* between grey-scale images. There are three types of image models: Measure spaces, pixelated data and functions. Using this approach, we consider an image as a measure space. Therefore, by an image P with support K we mean an *integer-valued* nonnegative function $p(i, j)$ defined on K, i.e. $P = \{p(i, j) : (i, j) \in K\}$. We define as a *Borel measure* on the space of grey-scale images the *pixel value* $p(i, j)$, where i and j are the Cartesian coordinates of the pixel.

For a compact metric space (X, d), let P_1 and P_2 be two Borel probability measures on X and define $\Theta(P_1, P_2)$ as the set of all probability measures P on $X \times X$ with *fixed marginals* $P_1(\cdot) = P(\cdot \times X)$ and $P_2(\cdot) = P(X \times \cdot)$. Next,

let

$$\mathrm{Lip}(X) = \{f: X \to \mathbb{R} \,\big|\big|\, f(x) - f(y) \,| \leq d(x,y), \forall x, y \in X\}$$

and define the distance between P_1 and P_2 as

$$B_d(P_1, P_2) = \sup\left\{\left|\int_X f(x)P_1(dx) - \int_X f(x)P_2(dx)\right|, f \in \mathrm{Lip}(X)\right\}.$$

The images considered are sets of finite collection of pixels, so they constitute compact metric spaces.

Let K_1 and K_2 be two images, S_n, $1 \leq n \leq N$ be the pixels of K_1 and R_m, $1 \leq m \leq M$ the pixels of K_2. Using the terminology of Kaijser we call K_1 the *transmitting image* and K_2 the *receiving image*; S_n, $1 \leq n \leq N$ denote *sources* whereas R_m, $1 \leq m \leq M$ denote *sinks* or *destinations*. By a *flow* we mean the amount of goods sent from the source S_n to the sink R_m denoted by $x(n,m)$ whereas $c(n,m)$, $1 \leq n \leq N, 1 \leq m \leq M$ denote the cost of transferring goods from S_n to R_m. In our case the cost corresponds to the distance between S_n and R_m. If $a(n)$ denote the amount of goods available in a source and $b(n)$ the amount of goods needed in a sink, the Kantorovich distance between K_1 and K_2 can be formulated as a balanced transportation problem as follows:

$$\text{Minimize} \sum_{n=1}^{N}\sum_{m=1}^{M} c(n,m) \cdot x(n,m)$$

subject to $x(n,m) \geq 0, 1 \leq n \leq N, 1 \leq m \leq M$,

$$\sum_{m=1}^{M} x(n,m) = a(n), \quad 1 \leq n \leq N \tag{1}$$

$$\sum_{n=1}^{N} x(n,m) = b(m), \quad 1 \leq m \leq M \tag{2}$$

and

$$\sum_{n=1}^{N} a(n) = \sum_{m=1}^{M} b(m).$$

The distance can be any of the following distances: L_1-metric or L_2-metric. For each source and each sink we define two quantities $\alpha(n)$ and $\beta(m)$ respectively, called *dual variables*. If

$$c(n,m) - \alpha(n) - \beta(m) \geq 0, \quad 1 \leq n \leq N, 1 \leq m \leq M,$$

we call the set of dual variables *feasible*. A pair of indices (n,m), where n is an index of a source S_n and m is an index of a sink R_m, is called an *arc*. If an arc (n,m) satisfies the condition

$$d(n,m) - \alpha(n) - \beta(m) = 0, \tag{3}$$

where $d(n,m)$ is the underlying distance-function between the pixels S_n and R_m, it is called an *admissible arc*; otherwise it is called *nonadmissible*. We say that a flow is *optimal* if Equations (2) and (3) hold.

The dual version of the transportation problem is

$$d_K(P,Q) = \mathrm{Max}\left\{ \sum_{n=1}^{N} \alpha(n)\cdot a(n) + \sum_{m=1}^{M} \beta(m)\cdot b(m) \right\} \tag{4}$$

when the set of dual variables is feasible.

3 The proposed algorithm

Our algorithm is based on the well known *primal-dual algorithm* which solves the balanced transportation problem on the plane. We make several enhancements, however, that improve the efficiency of the algorithm. Our improvements are based on the data structures used to store image data and on the fact that the transportation cost is the distance between the pixels. The latter allows us to use some spatial data structures which facilitate the computations and minimise the complexity of the problem. Before describing our method in detail, we give the main steps of the primal-dual algorithm:

1. Determine an initial value of the dual variables, find the corresponding set of admissible arcs and their flow.
2. Check if the current admissible flow is maximal. If it is go to (4), else go to (3).
3. Update the admissible flow and go to (2).
4. Check if the current maximal flow is optimal. If it is go to (7), else go to (5).
5. Update the dual variables.
6. Find the new admissible arcs and go to (2).
7. Stop.

Let us define as *total transporting grey mass* the summation of the grey value of all pixels in the transporting image. Similarly, we define as *total receiving grey mass* the summation of the grey value of all pixels in the receiving image. In order to convert the Kantorovich distance problem between images to a balanced transportation problem on the plane, both transporting and receiving total grey values must be equal. In general, these two amounts are different and in order to make them equal we change both masses accordingly applying the following formula on every single pixel value of both images:

$$p_{new}(n) = p(n)\cdot \hat{L}(K_2), \quad \hat{L}(K_2) = \Big(\sum_{m=1}^{M} q(m) \Big) / GCD(L,Q),$$

$$q_{new}(m) = q(m)\cdot \hat{L}(K_1), \quad \hat{L}(K_1) = \Big(\sum_{n=1}^{N} p(n) \Big) / GCD(L,Q),$$

where $p(n)$ and $q(m)$ are the pixel values of the transmitting and receiving images respectively, $L = \sum_{n=1}^{N} p(n)$, $Q = \sum_{m=1}^{M} q(m)$ and $GCD(L, Q)$ is the greatest common divisor of L and Q. In the following we shall describe our algorithm as well as the data structures we use to facilitate our computations and image storage.

3.1 Dual variables and the flow of the current admissible arcs

After having made the total grey masses of both images equal we have to initialise the dual variables. We set as initial values $\alpha(n) = \min\{d(n, m), 1 \leq m \leq M\}, i \leq n \leq N$ and $\beta(m) = 0, 1 \leq m \leq M$. From the above equations we observe that the initial values of the dual variables $\alpha(n)$ associated with the transmitting image pixels, are the distances of their nearest neighbour pixels of the receiving image. In order to compute this quantity we create a kd-tree structure[1] using the coordinates of the receiving image pixels and we search for the nearest neighbour of every single transmitting pixel. So, if n is a transmitting pixel and m one of its nearest neighbours in the receiving image, then (n, m) is an admissible arc. Therefore, the initial flow along this arc is $x(n, m) = \min\{p(n), q(m)\}$, whereas the new pixel values are $p(n) - x(n, m)$ and $q(m) - x(n, m)$.

3.2 Increasing the flow along the current set of admissible arcs

We call *surplus source* a transmitting pixel with $p(n) > 0$; otherwise, it is called a *zero source*. A receiving pixel having $q(m) > 0$ is called a *deficient sink*; otherwise, it is called *zero sink*. We define as *augmenting path* a set of admissible arcs connecting sources and sinks starting from a surplus source and ending with a deficient sink running through zero sinks and sources interchangeably. Moreover, the flow along admissible arcs connecting zero sinks with zero sources must be positive. In this step we use a labelling procedure to determine augmenting paths. It is clear that we can have flow increment only along augmenting paths. The labelling procedure is described as follows.

Start by labelling all surplus sources and then label all sinks that are connected to those sources with admissible arcs. Then, using the last labelled sinks, label all sources that are not labelled yet and are connected to those sinks with admissible arcs of positive flow. Repeat the above procedure until either a deficient sink is labelled or no more nodes can be labelled. If a deficient sink is labelled, then proceed to flow augmentation along the path that has been found. If no such path is found, the current admissible flow is maximal. For faster labelling procedure, we don't use any extra data structure. We reorder the pixels of both the transmitting and the receiving image in the initial data structure depending on whether they are labelled

[1] http://www.cs.umd.edu/∼mount/ANN/

or unlabelled. To speed up the reordering process, we store the pixel data in doubly linked lists which need $O(1)$ to move the nodes along the list.

Let $\theta_1 = \min\{x(m,n)\}$ be the minimum value of the positive flows belonging to the augmenting path connecting a labelled source and a label sink directed from sink to source. We define by

$$\theta = \min\Big\{a(n) - \sum_{j=1}^{M} x(n,j), b(m) - \sum_{i=1}^{N} x(i,m), \theta_1\Big\}.$$

Then, we can increase the flow along the path by setting the value of the starting source pixel to $p(n) - \theta$, the value of the ending sink to $q(m) - \theta$, by increasing the flows directed from source to sink by θ and by decreasing the flows from sink to source by the same amount. A drawback of this labelling procedure is that, after increasing the flow along an augmenting path, we may obtain *cycles*. In order to avoid them, we change the way we apply the labelling procedure by using only positive admissible arcs during the whole procedure. In that way, however, we cannot find all the augmenting paths. So, we use a *flow tuning procedure* which finds all possible augmenting paths for the current set of admissible arcs without having to store and use all the zero flow admissible arcs.

3.3 Flow tuning procedure

We define as *surplus flow tree* a set of paths starting from a surplus source and ending to zero sinks. A *zero flow tree* is a flow tree with a zero source as starting node. The main purpose of the flow tuning procedure is to find admissible arcs that connect zero sources belonging to surplus flow trees and unlabelled deficient sinks. To do that, a kd-tree is constructed using the coordinates of the unlabelled deficient sinks. Then, using the kd-tree structure for each zero source belonging to a surplus flow tree, we locate all the deficient unlabelled sinks that lay within a distance $\alpha(n)$ from itself. After that, a new augmenting path has been located and the flow is augmented as described in the previous subsection. According to the definition of the augmenting path, there is no reason to search for arcs that connect zero sources that belong to zero flow trees with unlabelled zero sinks. In such a way we decrease the number of sinks as well as the number of considered sources. The first one leads to a faster construction of the kd-tree whereas the second one minimises the number of input points.

3.4 Dual variable update and the new set of admissible arcs

When no more augmenting paths can be located for the current set of admissible arcs, we proceed to the dual variable update procedure. The main reason for updating the dual variables associated with both sources and sinks

is to create new admissible arcs in order to achieve the maximal and also the optimal flow. According to Kaijser[7], if the underlying metric is the L_1-metric, the dual variable can be changed by $\delta = 1$. In order to preserve the current flow along the current set of admissible arcs, the dual variables are changed as follows:

$$\alpha_{new}(n) = \alpha_{old}(n) + \delta, \quad n \in M_1, \quad \alpha_{new}(n) = \alpha_{old}(n), \quad n \in U_1,$$
$$\beta_{new}(m) = \beta_{old}(m) - \delta, \quad m \in M_2, \quad \beta_{new}(m) = \beta_{old}(m), \quad m \in U_2,$$

where M_1 and M_2 denote the sets of indices of labelled sources and sinks, respectively, whereas U_1 and U_2 denote the sets of indices of unlabelled sources and sinks, respectively. To improve the dual variable update, we define a new variable Δ as the running total of the dual variable changes as the algorithm evolves; see also [1]. We apply the above mentioned dual variable change routine using Δ instead of δ. Because of the way we change the dual variables, new positive flow admissible arcs are created between the labelled surplus sources and the unlabelled deficient sinks. To find out the new set of admissible arcs, a kd-tree is constructed using the coordinates of the unlabelled deficient sinks. Then, for each surplus source, we locate all the deficient sinks that lay within a distance of $\alpha(n) + \Delta$ from it. After finding out the new set of positive flow admissible arcs, the algorithm is applied again until no more surplus nodes exist.

4 Results

We now present typical results from the application of our algorithm to real images, aiming to demonstrate its applicability to the demanding problems inherent in the image compression area and its performance. The original images used as our reference point in the experiments presented here are the $256 \times 256 \times 8$ bpp Lena and Barbara images shown in Figure 1. We examine for each original image how close it is to a filtered or compressed replica of it. In other words we seek to measure the difference (i.e. the error) between two images by computing the Kantorovich distance between the original image and each of the associated filtered ones.

	μ, μ_1	μ, μ_2	μ, μ_3	ν, ν_1	ν, ν_2	μ, ν
d_K	2,789,456	8,562,357	4,532,730	3,125,789	8,998,678	15,853,930
t_K	27:26	42:22	41:15	31:52	44:27	1:12:36

Table 1. The Kantorovich distance d_K between the real-world images and the computation time in hour:min:sec format.

The correspondence between the images of Lena and the indices is the following: μ = original image, μ_1 = wavelet compression, μ_2 = JPEG compression and μ_3 = 8:1 fractal compression. The correspondence between the

Fig. 1. The original images of Lena (left) and Barbara (right) used in our experiments ($256 \times 256 \times 8$ bpp).

images of Barbara and the indices is the following: ν = original image, ν_1 = 64:1 compression and ν_2 = JPEG compression. Time results are given in CPU minutes on a Core$^{\text{TM}}$ 2 Duo PC with a 2.13 GHz CPU clock, 4 GB RAM and running Windows 7 Ultimate. Looking at Table 1 from left to right we can see, which of the images are closer to the originals. The runtime of our algorithm is better than the one presented in [7].

References

1.D.S. Atkinson and P.M. Vaidya. Using geometry to solve the transportation problem in the plane. *Algorithmica*, 13:442461, 1995.

2.M.F. Barnsley. *Fractals everywhere*, 2nd ed., San Diego, 1993. Academic Press Professional.

3.J. Brandt, C. Cabrelli and U. Molter. An algorithm for the computation of the Hutchinson distance. *Information Processing Letters*, 40:113–117, 1991.

4.Y. Deng and W. Du. The Kantorovich metric in Computer Science: A brief survey. *Electronic Notes in Theoretical Computer Science*, 253:73–82, 2009.

5.V. Drakopoulos and N. P. Nikolaou. Efficient computation of the Hutchinson metric between digitized images. *IEEE Transactions on Image Processing*, 13:1581–1588, 2004.

6.J.E. Hutchinson. Fractals and self similarity. *Indiana University Mathematics Journal*, 30: 713–747, 1981.

7.T. Kaijser. Computing the Kantorovich distance for images. *Journal of Mathematical Imaging and Vision*, 9:173–191, 1998.

8.N. Wadströmer. Coding of fractal binary images with contractive set mappings composed of affine transformations, *PhD thesis*, Linköping University, 2001.

9.M. Werman, S. Peleg and A. Rosenfeld. A distance metric for multidimensional histograms. *Computer Vision Graphics Image Processing*, 32:328–336, 1985.