Summary of the lecture „Data Science and Big Data“ by Dr. Tamas Horvath (SoSe 2020)

Fabrice Beaumont
Matrikel-Nr: 2747609
Rheinische Friedrich-Wilhelms-Universität Bonn

29. Mai 2020

Inhaltsverzeichnis

1 A: Wasserstein Weisfeiler-Lehman Graph Kernels

A lot of use of graph-structured data. E.g.:
Social networks, sensor networks, chemo-informatics, bio.informatics.

Graph kernels - deal with complexity of graphs. Mostly R-Convolution = decomposition into substructures which are then compared. Problems:

1. Simple aggregation of the similarities of the substructures might limit the ability to capture complex characteristics of the graph
   (Solutions: Fröhlich et a. [15], Kriege et al. [25])
2. Most proposed variants do not generalist to graphs with high-dimensional continuous node attributes

Method of the paper: Vectorial graph representations + optimal transport theory (Wasserstein distances)

1.0 Definitions

DEF: Kernel

Kernels are a class of similarity functions.
Let X be a set and $k : X \to X \to \mathbb{R}$ be a function associated with a Hilbert space $\mathcal{H}$ such that

$$\exists \phi : X \to \mathcal{H} \text{ s.t.} \qquad k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$$

Then $\mathcal{H}$ is a **reproducing kernel Hilbert space** (**RKHS**) and k is said to be a **positive definite kernel**.

A positive definite kernel can be interpreted as a dot product in a high-dimensional space. Recal SVMs and the kernel trick.

DEF: Graph

A **graph** is a tuple $G = (V, E)$ where V is a set of nodes and $E \subseteq 2^V$ is a set of undirected edges.
Set $n_G := |V|$, $m_G := |E|$ and

$$\mathcal{N}(v) := \{u \in V | (u, v) \in E\}$$

and $|\mathcal{N}(v)| = \deg(v)$ (**first- order neighbourhood**).

G is **labelled**, if there exists a labelling function $l : V \to \sum$ for a finite label alphabet $\sum$.
G is **attributed**, if for each node $v \in V$ there exists an associated vector $a(v) \in \mathbb{R}^m$.
We call $a(v)$ **node attributes** (high-dimensional continuous vectors) and $l(v)$ **node labels** (integer).
G is called **weighted** if there exists a weight function on its edges $w : E \to \mathbb{R}$.

DEF: $\mathcal{R}$-Convolution

Decompose graph G into substructures and define a kernel value $k(G, G')$ as a combination of substructure similarities.

Often $\mathcal{R}$-Convolution kernels discard valuable information such as the distribution of the substructures.

DEF: Wasserstein distance

The **Wasserstein distance** W_p (earth mover distance) is a distance function between probability distributions defined on a given metric space.
Let σ and μ be two probability distributions on a metric space M equipped with a ground distance d such as the Euclidean distance. For $p \in [1, \infty)$ it is:

$$W_p(\sigma, \mu) := \Big(\inf_{\gamma \in \Gamma(\sigma,\mu)} \int_{M \times M} d(x,y)^p d\gamma(x,y) \Big)^{\frac{1}{p}}$$

where $\Gamma(\sigma, \mu)$ is the set of all transportation plans $\gamma \in \Gamma(\sigma, \mu)$ over $M \times M$ with marginals σ and μ on the first and second factors respectively.

d is a metric $\implies$ W_p is a metric (Villani [44], chapter 6, for a proof)
Focus on $p = 1$ (L^1-Wasserstein distance).

HERE: Finite sets of node embeddings (no continuous probability distributions). Therefore formulate as a sum rather than an integral: Given two sets of vectors $X \in \mathbb{R}^{n \times m}$ and $X' \in \mathbb{R}^{n' \times m}$ it is:

$$W_1(X, X') := \min_{P \in \Gamma(X,X')} \langle P, M \rangle$$

M - distance matrix containing $d(x, x')$
$P \in \Gamma$ - transport matrix (or joint probability). Contains fractions that indicate how to transport the values from X to X' with minimal total transport effort. Because we assume that the total mass to be transported equals 1 and is evenly distributed across the elements of X and X', the row and column values of P mus sum up to $\frac{1}{n}$ and $\frac{1}{n'}$ respectively.
$\langle \dot{;} \rangle$ - Frobenius dot product

DEF: Optimal transport problem

„Find the most inexpensive way“ in terms of ground distance
(to transport all the probability mass from distribution σ to match distribution μ.)

Intuition for 1D: two probability distribution as piles of sand. Now the minimum effort required to move the content of the first pile to reproduce the second pile = Wasserstein distance.

1.0.1 The Method

1. Transform each graph into a set of node embeddings (definition 1.0.1)
2. Measure the Wasserstein distance between each pair of graphs
3. Compute a similarity matrix to be used in the learning alg.

DEF: Graph Wasserstein Distance

Given two graphs $G = (V, E)$ and $G' = (V', E')$ a graph embedding scheme $f : G \to \mathbb{R}^{|V| \times m}$ and a ground distance $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ we define the **Graph Wasserstein Distance** (**GWD**) as

$$D_W^f(G, G') := W_1(f(G), f(G'))$$

DEF: Graph Embedding Scheme

Given a graph $G = (V, E)$, a graph embedding scheme $f : G \to \mathbb{R}^{|V| \times m}$, $f(G) = X_G$ is a function that outputs a fixed-size vectorial representation for each node in the graph.
For each $v_i \in V$, the i-th row of X_G is called the node embedding of v_i.

m = dimensionality of node attributes (e.g. $m = 1$ for only labels).

Concrete embedding: **Weisfeiler-Lehman scheme** (**WL scheme**)

1.1 Weisfeiler-Lehman = Node embedding scheme

Designed for labelled non-attributed graphs. Looks at similarities among subtree patterns, iteratively compares labels on the nodes and their neighbours: Create new labels for each

node, which encode the labels of their neighbors by hashing its old label and a string of labels of its neighbours.

Consider a graph $G = (V, W)$, let $l^0(v) = l(v)$ be the initial node label of v for each $v \in V$ and let H be the number of WL iterations. Then, we can define a recursive scheme to compute $l^h(v)$ for $h = 1, \ldots, H$ by looking at the ordered set of neighbours labels $\mathcal{N}^h(v) = \{l^h(u_0), \ldots, l^h(u_{\deg(v)-1})\}$ as

$$l^{h+1}(v) = \text{hash}(l^h(v), \mathcal{N}^h(v))$$

Use perfect hashing for the hash function.

Nodes at iteration $h+1$ will have the same label iff their label and those of their neighbours are identical at iteration h.
Since with each iteration the encoded neighbourhood increases, the nodes necessary to encode all labels decreases over time.

For graphs with continuous attributes $a(v) \in \mathbb{R}^m$ we need to improve the WL refinement step.
Idea: Create an explicit propagation scheme that uses the average over the attributes and weight of the neighbourhoods.
Suppose we have a continuous attribute $a^0(v) = a(v)$ for each node $v \in G$. Define recursively:
Idea: $a^{h+1}(v) = a^h(v) + \sum_{u \in \mathcal{N}(v)} w((v,u)) a^h(u)$
When edge weights are not available set $w(u, v) = 1$.
Realization:

$$a^{h+1}(v) = \frac{1}{2}\Big(a^h(v) + \frac{1}{\deg(v)} \sum_{u \in \mathcal{N}(v)} w((v,u)) a^h(u)\Big)$$

(The weighted average instead of simple sum, and the $1/2$ shall ensure a similar scale of the features across iterations / such tricks lead to better empirical results.)

» Intuitive extension for WL subtree kernel. Applicable to all types of graphs! (Without

additional hashing)

FUTURE WORK: Extension for high-dimensional edge attributes (and dual graphs).

(Similar approaches for **node-level kernel similarities** rely on additional hashing steps for the continuous features.)

1.2 Graph embedding scheme

DEF: WL features

Let $G = (V, E)$ and let H be the number of WL iterations. Then for every $h \in \{0, \ldots, H\}$ we define the WL features as

$$X_G^h = [x^h(v_1), \ldots, x^h(v_{n_G})]^T$$

where $x^h(\dot{}) = l^h(\dot{})$ for categorically labelled graphs and $x^h(\dot{}) = a^h(\dot{})$ for continuously attributed graphs. We refer to $X_G^h \in \mathbb{R}^{n_g \times m}$ as the **node features** of graph G at iteration h. THen the node embeddings of graph G at iteration H are defined as $f^H : G \to \mathbb{R}^{n_G \times (m(H+1))}$:

$$G \to \text{concatenate}(X_G^0, \ldots, X_G^H)$$

FUTURE WORK: Both categorically labelled + continously attributed. Information can be joined and used in an extended formula, but the question of appropriate distance measures between categorical and continuous data remains.

2 Wasserstein distance

Ground distance between each pair of embedded nodes.
For **categorical node features** use Hamming distance:

$$d_{\text{Hamm}}(v, v') = \frac{1}{H+1} \sum_{i=1}^{H+1} \rho(v_i, v_i')$$

$$\rho(x,y) = \begin{cases} 1, & x \neq y \\ 0, & x = y \end{cases}$$

Use, since the Weisfeiler-Lehman features are categorical!! (Values carry no meaning). For **continous node feature** use Euclidean distance:

$$d_E(v,v') = ||v - v'||_2$$

Time-Complexity: n = cardinality of indexed set / number of nodes in the two graphs:

$$\mathcal{O}(n^3 \log(n))$$

Speed-up: Approximations using Sinkhorn regularisation:

near-linear time (while accurate)

3 Kernel

DEF: Wasserstein Weisfeiler-Lehman kernel

Given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_N\}$ and the GWD defined for each pair of graphs on their WL embeddings, we define the **Wasserstein Weisfeiler-Lehman** (WWL) kernel as

$$K_{\text{WWL}} = \exp\Big(- \lambda D_W^{f_{WL}} \Big)$$

Laplacian kernel > favourable conditions for positive definiteness with non-Euclidean distances: **Categorical WWL is positive definite** for all $\lambda > 0$
(Open problem for continuous WWL!!)

(Wasserstein distance is not isometric/ no metric-preserving mapping to an L^2-norm / thus not necessarily possible to derive a positive definite kernel. Laplacian kernel fortunately did this.)

For continuous WWL - treat as if indefinite kernel:

- (reproducing kernel) **Krein spaces** (RKKS)
  (Generalisation of reproducing kernel Hilbert spaces)
- Krein SVM (KSVM) as classifier

1.1 Experimental evaluation

1.1.1 Set-Up:

Data sets:

- Categorical labels:
  - MUTAG
  - PTC-MR
  - NCI1
  - D&D
- Categorical labels and continuous attributes:
  - ENZYMES
  - PROTEINS
- Continuous attributes:
  - IMDB-B
  - BZR
  - COX2
- Continuous attributes and edge weights:
  - BZR-MD
  - COX2-MD

(Source: Kersting et al.)

Comparison of the method with other graph kernel methods:

- Categorical labels:
  - WL
  - WL-OA (superior to previous approaches)
  - Vertex histogram **FRAGE: WasDas?** (V)
  - Edge histogram (E)
- Continuous:
  - Hash graph kernel (HGK-SP, HGK-WL)
  - GraphHopper (GH)
  - Continuous vertex histogram (VH-C)
    (RBF kernel)
  - RBF-WL (own method, but RBF kernel replaces Wasserstein distance)

Classifier: SVM (KSVM for WWL because of the possible indefiniteness of the kernel - negligible differences),
10-fold cross-validation, selecting the parameters on the training set only.

Implementation: Python. 'Original implementations' of the other compared methods. Ubuntu14.04.5 LTS, with 4 CPUs (Intel Xeon E7-4860 v2 @ 2.60GHz) each with 12 cores and 24 threads, and 512 GB of RAM.

1.1.2 Results:

BEMERKUNG: Seltsame Plazierung der Tabellen.

Categorical labels

WWL comparable to WL-OA,
WWL better than WL
BEMERKUNG: Erneut komische Reihenfolge der Ergebnisse.

Categorical labels

WWL significantly better 4/7.
WWL better 1/7.
WWL equal 2/7.
On average always better. $\implies$ New state of art.

Empirical observation: WWL kernel might be positive definite.

Clear benefits over the additive hashing (hash graph kernel HGK).

2 B: GOT: An Optimal Transport framework for Graph comparison

Motivation: Lack of efficient measures for comparing graphs.

Goal of the paper:

- Distance between two graphs (compare graphs)
  Use: Smooth graph signal distribution
- Transportation plan for data from one graph to another (align permuted graphs)
  (if graphs have same nr. of vertices)
  Use: Wasserstein distance (captures the main structural informations of the graphs), new **graph alignment problem**

Optimal Transport (OT) by Monge and later Kantorovich.

Benefits of the method in representative tasks (noisy graph alignment, graph classification, graph signal transfer). Possibility to predict graph signals.
Outperforms Gromov-Wasserstein and Euclidean distance in graph alignment and graph clustering.

Graph matching = NP-hard. Search of Approximations:

- spectral clustering
  (Problem: Matching accuracy is suboptimal)
- Semi-definite programming relaxation

- Gumbel-sinkhorn network
  (for the non-convex quadratic problem version)
- Flamary et al.: Optimal transport for empirical distributions
- Gu et al.: Spectral distance (does not use the full graph structure)
- Nikolentzos et al.: Graph embeddings
- Memoli: Gromov-Wasserstein distance (object matching)
- Peyré et al.:Gromov-Wasserstein distance and barycenter of dissimilarity matrices (+ Sinkhorn projections)
- Vayer et al.: distance for graphs + signals

2.1 Graph alignment - optimal transport

DEF: Graphs and Laplacian matrix

Graph $G = (V, E)$, $|V| = N$, connected, undirected, edge weighted.
Adjacency matrix $W \in \mathbb{R}^{N \times N}$.
$d(i)$ degree of $i \in V$. Degree matrix $D \in \mathbb{R}^{N \times N}$:

$$D_{i,j} = \begin{cases} d(i), & i = j \\ 0, & i \neq j \end{cases}$$

Laplacian matrix $L := D - W$

2.1.1 Smooth graph signals

graphs = probability distributions of signals
G_1, G_2 graphs with Laplacian matrices L_1 and L_2.

Singals follow the normal distribution with zero mean and the transposed Laplacian matrices as covariance matrix:

$$\nu^{G_1} = \mathcal{N}(0, L_1^\dagger) \qquad \mu^{G_2} = \mathcal{N}(0, L_2^\dagger)$$

(Graph signal values vary slowly between strongly connected nodes.)

2.1.2 Wasserstein distance

compare: signal distributions...

DEF: Wasserstein distance

The 2-Wasserstein distance between two graphs G_1 and G_2 with distributions ν^{G_1} and μ^{G_2} is:

$$W_2^2(\nu^{G_1}, \mu^{G_2}) = \inf_{T_\# \nu^{G_1} = \mu^{G_2}} \int_X ||x - T(x)||^2 d\nu^{G_2}(x)$$

Where $T_\# \nu^{G_1}$ denotes the push forward of ν^{G_1} by the transport map $T : X \to X$ defined on a metric space X.

For normal distributions such as ν^{G_1} and μ^{G_2} the 2-Wasserstein distance can be explicitly written in terms of their covariance matrices:

$$W_2^2(\nu^{G_1}, \mu^{G_2}) = \mathrm{Tr}(L_1^\dagger + L_2^\dagger) - 2\mathrm{Tr}\Big(\sqrt{L_1^{\dagger\frac{\dagger}{2}} L_2^\dagger L_1^{\dagger\frac{\dagger}{2}}}\Big)$$

And the optimal transportation map is
$T(x) = L_1^{\dagger\frac{\dagger}{2}} \Big(L_1^{\dagger\frac{\dagger}{2}} L_2^\dagger L_1^{\dagger\frac{\dagger}{2}}\Big)^{\dagger\frac{\dagger}{2}} L_1^{\dagger\frac{\dagger}{2}} x$

The distance is sensitive to differences that cause a global change in the connection between graph components. (No sensitive to differences hat have small impact on the whole graph structure.)
FRAGE: Was sind lower graph frequencies?? Vielleicht sind die signale / die Frequenzen auf die komprimierten Darstellungen aller Knoten attribute??

2.1.3 Graph alignment

Signal distributions depend on the enumeration of nodes (in L_1 and L_2)!

The graph alignment problem: Given two graphs G_1, G_2 with N distinct vertices each (and different sets of edges). Find the optimal transportation map T from G_1 to G_2.
FRAGE: Was ist eine transportation map?

Define the probability measure of a permuted representation for the graph G_2 as

$$\mu_P^{G_2} = \mathcal{N}(0, (P^T L_2 P)^\dagger) = \mathcal{N}(0, (P^T L_2^\dagger P))$$

where $P \in \mathbb{R}^{N \times N}$ is a permutation matrix.
FRAGE: Was tut das? Ist es die WKeit, dass G_2 eine Permutation von G_1 ist?

Thus the graph alignment pb. = find the permutation that minimizes the mass transportation between ν^{G_1} and $\mu_P^{G_2}$:

...

$$W_2^2(\nu^{G_1}, \mu_P^{G_2}) = \mathrm{Tr}(L_1^\dagger + P^T L_2^\dagger T) - 2\mathrm{Tr}\Big(\sqrt{L_1^{\frac{\dagger}{2}} P^T L_2^\dagger P L_1^{\frac{\dagger}{2}}}\Big)$$

2.2 The Method (GOT)

Stochastic gradient descent (alg 1).

Main difficulty: permutation matrix P leads to a discrete optimization problem with a factorial number of feasible solutions.

Idea: Enforce constraints implicitly by using the Sinkhorn operator. The operator normalizes the rows and columns of $\exp(P/\tau)$:

$$S_\tau(P) = A \exp(P/\tau) B$$

Iterative computation of A and B ...
For $\tau \to 0$ the operator yields a permutation matrix.

Changed problem formulation - leads to differentiability - solvable with gradient descent (although highly non-convex - may converge towards local minima).
FRAGE: Ist ersichtlich, warum ohne den Sinkhorn Operator es noch nicht differnezierbar/ für gradient descent lösbar war??

Stochastic exploration

Idea: Optimize the expecation of the cost function w.r.t. the parameters of some distribution q_0: ...
Reformulation (reparameterization trick) ... (17)
Reformulation/ Approximation (parameterless distribution) ... (18)

Alg. converges almost surely to a critical point (no guaranteed global minimum).

Computational complexity:
- naive: $\mathcal{O}(N^3)$ per iteration (due to matrix square-root in singular value decomposition (SVD))
- better: approximate matrix square-root with Newton's method (results in only matrix multiplications)
- better: this + avoid pseudo-inverse computation by adding small diagonal shift to Laplacian matrices and directly compute the inverse

Seite 7 - Kapitel 4

2.3 Experiments

2.3.1 Graph alignment

Use grid search to chose the parameters ρ (Sinkhorn) and γ (learning rate) - while the sample size S was fixed empirically.
For all experiments: $\rho = 5, \gamma = 0.2, S = 30$.
Maximal 10 Sinkhorn iterations, and 3000 iterations for stochastic gradient descent (- typically convergence after around 1000).

Test-graphs: Stochastic block model graph with 40 nodes and 4 communities. Create a noisy version of this graph by removing edges with...

- ...probability $p = 0.5$ within communities
- ...increasing probabilities $p \in [0, 0.6]$ between communities

and generate a random permutation to change the order of nodes.

Compares methods:

- GOT
- Stochastic alg (Euclidean disance)
- State-of-art Gromov-Wasserstein distance for graphs (GW)
  (Based on Euclidean distance between shortest path matrices)

Adjusting parameters for all methods - repeat each experiment 50 times.

GOT outperforms all two methods - but is a bit worse than GW if the error measurement of GW is used.

2.3.2 Graph classification

Creation of 20 test graphs (20 nodes each - randomly permuted, similar numbers of edges) for each of these five models:

- Stochastic Block Model (2 blocks - SBM2)
- Stochastic Block Model (3 blcosk - SBM3)
- Random regular graph (RG)
- Barabasy-Albert model (BA)
- Watts-Strogatz model (WS)

GOT: Align graphs
Non-Parametric 1-NN classification algorithm / several methods:

- GW
- FGM
- IPFP
- RRWM
- NetLSD

Results as confusion matrices. GOT clearly outperforms (general accuracy).

2.3.3 Graph signal transportation

MNIST dataset (60000 images of size 28×28 of handwritten digits - 6000 per class). For each digit class stack all available images into a feature matrix (6000×784) and build a graph over the $784 = 28^2$ feature vectors:

- 20-nearest neighbour binary graph
- square it (results in 2-hop distances / more meaningful weights)

Hence each of the ten classes of digits is represented by a graph of 784 nodes (pixels).

Each image of a given class can be seen as a smooth signal $x \in \mathbb{R}^{784}$ that lives on the corresponding graph.

Construct transportation plan T between a source graph and all nine other graphs.
Result: Transport yields in recognizable images with similarity to the original digits.

Same experiment for Fashin MNIST.
Also recognizable results plus texture preservation (images of clothes). Potential of GOT in graph signal prediction through adaptation of a graph signal to another graph!

3 C: A Graph Theoretic Additive Approximation of Optimal Transport

Method: Adaptation of the classical graph algorithm of Gabow and Tarjan. Execution time $\mathcal{O}\left(\frac{n^2C}{\delta} + \frac{nC^2}{\delta^2}\right)$.
Arbitrarily small constant ϵ: $\lfloor \frac{2C}{(1-\epsilon)\delta} \rfloor + 1$ iterations
(each iteration: Dijkstra-type search + DFS)
Method does not slow down for very small values of δ (unlike Sinkhorn alg).

Transportation cost as a similarity measure between data sets (point clouds, probability distributions, images):
Set A of demand nodes. Set B of supply nodes. Demand $0 < d_a$ at node $a \in A$, supply $0 < s_b$ at node $b \in B$.
Let $G(A, B)$ be a complete bipartite graph on A and B with $n = |A| + |B|$ where $c(a, b) \geq 0$ denotes the cost of transporting one unit of supply from b to a. Let C be the largest cost of any edge in the graph. Assume the cost is symmetric ($c(a, b) = c(b, a)$). W.l.o.g. assume that total supply is at most the total demand (no over saturation). Let $U := \sum_{b \in B} s_b$.

DEF: Transport Plan

A **transport plan** is a function $\sigma : A \times B \to \mathbb{R}_{\geq 0}$ that assigns a non-negative value $\sigma(a, b)$ to every edge (a, b) with the constraints that

$$\forall a \in A : \quad \sum_{b \in B} \sigma(a, b) \leq d_a$$

(i.e. the total supply coming into any node $a \in A$ is at most d_a) and

$$\forall b \in B: \quad \sum_{a \in A} \sigma(a,b) \leq s_b$$

(i.e. the total supply leaving a node $b \in B$ is at most s_b).

A **maximum transport plan** is one where

$$\forall b \in B: \quad \sum_{a \in A} \sigma(a,b) = s_b$$

(i.e. every available supply is transported).

The cost incurred by any transport plan σ is

$$w(\sigma) =: \sum_{(a,b) \in A \times B} \sigma(a,b) c(a,b)$$

DEF: Transportation Problem

Input: A complete bipartite graph $G(A,B)$
Output: Minimum-cost maximum transport plan.

Special cases:

- All demands and supplies positive integers (**Hitchcock-Koopmans transportation problem**)
- Demand or supply at every node is 1 (**Assignment problem**)
- A and B are discrete probability distributions and $U = 1$ (**Optimal transport distance between distributions**)
- Transporting costs is a metric: **Earth Mover's distance** (EMD)
- Transporting cost is p-th power of some metric cost ($p > 1$): **p-Wasserstein's distance**

Related work - Exact methods:

Name	Task	Method	Time	Difficulty
Hungarian method	Assignment pb.	Linear programming duality	$\mathcal{O}(n^3)$	
Gabow, Tarjan	Assignment pb.	Cost scaling paradigm	$\mathcal{O}(n^{2.5}\log(nC))$	
Gabow, Tarjan	Transportation pb. (Integer demands, supplies and edge costs)	Cost scaling paradigm	$\mathcal{O}(n^2\sqrt{U} + U\log(U))$	Scaling demands and supplies to integers leads to U in $\Omega(n)$ - thus runtime $\Omega(n^{2.5})$ again
	Minimum-cost maximum transport plan (integer demands and supplies) = minimum-cost maximum flow		$\tilde{\mathcal{O}}(n^{2.5}\text{poly}\log(U))$ ($\tilde{\mathcal{O}}$ notation suppresses additional logarithmic terms in n)	
	Assignment problem		$\mathcal{O}(n^\omega C)$ (ω is the exponent of matrix multiplication time complexity)	

Execution time polynomial in n, $\log(U)$ and $\log(C)$ - but all quite slow in practise.

Idea: Approximate transport plans.

DEF: δ-approximate Transportation Problem

Input: A complete bipartite graph $G(A, B)$
Output: A maximum transport plan with cost $(1 + \delta)w(\sigma^*)$.

σ^* is the maximum transport plan with the smallest cost.

DEF: δ-close Transportation Problem

Input: A complete bipartite graph $G(A, B)$
Output: A maximum transport plan with cost $w(\sigma) \leq w(\sigma^*) + U\delta$.

σ^* is the maximum transport plan with the smallest cost.
(For discrete probability distributions is is $U = 1$ and thus the solution within an additive error of δ from the optimum.)

Related work - Approximative methods:

Name	Task	Method	Time	Difficulty
	δ-approximate transport plans, d-dim Euclidean assignment pb. (Euclidean transport pb.)	$n(\log(n)/\delta)^{\mathcal{O}(d)}$	$\log(U)$	
	Metric costs, δ-approximate transport plan		$\tilde{\mathcal{O}}(n^2)$	
Cuturi, Altschuler	δ-close transport plan, arbitrary costs	Sinkhorn projection technique	$\tilde{\mathcal{O}}(n^2(C/\delta))$	
THIS PAPER	δ-close transport plan, arbitrary costs		$\mathcal{O}(n^2(C/\delta) + n(C/\delta)^2)$	
	δ-close transport plan, arbitrary costs	Exploit structure of squared Euclidean distances	$\mathcal{O}(n(C/\delta)^d \log^d(n))$	

Execution time near-linear in the input size and polynomial in $\log(n)$, $\log(U)$ and $\log(C)$.
No known δ-approximation algorithms for arbitrary costs with near-linear time.
For the δ-close transport plan pb. Blanchet et al. showed that any algorithm with less than quadratic dependence on C/δ can be used to compute a maximum cardinality matching in any arbitrary bipartite graph in $o(n^{5/2})$ time.

(Note that again $\tilde{\mathcal{O}}$ hides poly-logarithmic factors in the runtime. Some of them are artifacts of worst-case analyses of the algorithms and one cannot avoid them all-together in any practical implementation.)

All these implementations: **Significant increase in running time and numerical instability for smaller values of** δ.
Ideas to tackle this: parallel algorithms or exploiting the cost structure.

3.0.4 Method

Deterministic primal-dual algorithm to compute a δ-close solution in $\mathcal{O}(n^2(C/\delta) + n(C/\delta)^2)$ time.
Adaptation of a single scale of Gabow and Tarjan's scaling algorithm for the transportation problem (with scaled integer demand, supply and cost functions).
Key contribution: diameter-sensitive analysis.
(For geometric costs even $\tilde{\mathcal{O}}(n(C/\delta)^2)$.)
Transformation to integer demands and supplies in $\mathcal{O}(n^2)$.

Two step alg. (per iteration):

- Hungarian Search: Dijkstra's algorithm ($\mathcal{O}(n^2)$); adjust the weights corresponding to a dual linear program; find an augmenting path (consisting of zero slack edges)
- DFS from every node with excess supply; find augmenting paths of zero slack edges to route the supplies ($\mathcal{O}(n^2)$).

(The total length of paths found during the course of the algorithm is bound by $\mathcal{O}(n/\epsilon(1-\epsilon)(C/\delta)^2)$)

Note: Gabow and Tarjan's algorithm computes an optimal solution only when the total supply is equal to the total demand.
The presented algorithm works also for the unbalanced case (and has unlike the original alg. no dependencies on the cost of the optimal solution).

Unlike Sinkhorn projection based approaches, the presented algorithm is numerically stable and executes efficiently for smaller values of δ.

Scaling

- Scale demands and supplies to integer demands and supplies.
- Allows to apply the traditional framework of augmenting paths
- Find an approximate solution for the transformed pb. in $\mathcal{O}(\frac{n^2C}{\delta} + \frac{nC^2}{\delta^2})$.
- Map solution to a feasible solution for the original demands and supplies (Loss of accuracy due to transformation $\leq \epsilon U\delta$)

Scaling: $0 < \epsilon < 1$,

$$\alpha := \frac{2nC}{\epsilon U\delta}$$

for $a \in A$ and $b \in B$ use the following transformations:

$$d_a \to \overline{d}_a = \lceil d_a\alpha \rceil \qquad\qquad s_b \to \overline{s}_b = \lceil s_b\alpha \rceil$$

(Recall that the total supply is no more than the total demand!)
It follows

$$U = \sum_{b\in B} \overline{s}_b = \sum_{b\in B} \lfloor s_b\alpha \rfloor \leq \alpha \sum_{b\in B} s_b = \alpha U$$

Let σ^* be a feasible maximum transport plan for the transformed pb. $\mathcal{I}'$. Define a new transport plan pointwise as: $\sigma(a,b) = \sigma'(a,b)/\alpha$ (not necessarily feasible of maximum of $\mathcal{I}$). σ has costs $w(\sigma) = w(\sigma')/\alpha$.

Convert σ into a feasible and maximum transport plan in two steps:

1. Feasibility: Reduce excess supply κ_a for each demand node to zero by reducing $\sigma(a,b)$ (arbitrary adjacent node b) and κ_a by $\min\{\kappa_a, \sigma(a,b)\}$
   Total remaining (excess) supply: $2n/\alpha$. (Cost $w(\sigma) \leq w(\sigma')/\alpha$ - only cost reductions)
2. Maximum: Match the remaining $2n/\alpha$ to leftover demands of at most cost C per unit of supply:

$$w(\sigma) \leq w(\sigma')/\alpha + \frac{2nC}{\alpha} \leq w(\sigma')/\alpha + \epsilon U\delta$$

(Proof in supplement): The optimal solution for the original pb. is with factor α better than the $w(\sigma^*)$.

Alg for scaled demands and supplies

Alg. output: σ' with

$$w(\sigma') \leq w(\sigma'_{\mathrm{OPT}}) + (1-\epsilon)\delta U$$

in $\mathcal{O}(\frac{n^2 C}{(1-\epsilon)\delta} + \frac{nC^2}{\epsilon(1-\epsilon)\delta^2})$.

DEF: Free vertices

We say that a vertex $a \in A$ is **free** w.r.t. a transportation plan σ if

$$d_a - \sum_{b \in B} \sigma(a,b) > 0$$

(i.e. a is able to receive more supply than σ dictates.)

The set of free demand nodes is A_F.
(Analouge definition for the supply and B_F.)

Define $\delta' = (1-\epsilon)\delta$ and the scaled costs $\overline{c}(a,b) = \lfloor 2c(a,b)/\delta' \rfloor$. $\overline{w}(\sigma)$ denotes the cost of any transport plan w.r.t. the scaled cost function $\overline{c}$.

Algorithm: primal-dual approach. At all times the transport plan satisfies the dual feasibility conditions.

DEF: 1-Feasible transport plans

A transport plan σ along with a dual weight $y(v)$ for every $v \in A \cup B$ is 1**-feasible**,

if $\forall a \in A, b \in B$:

$$y(a) + y(b) \begin{cases} \leq \overline{c}(a,b) + 1 & \text{if } \sigma(a,b) < \min\{s_b, d_a\} \\ \geq \overline{c}(a,b) & \text{if } \sigma(a,b) > 0 \end{cases}$$

(Identical conditions to Gabow and Tarjan - but for costs scaled by $2/\delta'$ and rounded down.)

DEF: 1-Optimal transport plans

A transport plan σ along with a dual weight $y(v)$ for every $v \in A \cup B$ is 1**-optimal**, if it is 1-feasible and maximum.

Since the unbalanced case is considered, some conditions may not be fulfilled jet. One needs to demand that $\forall a \in A$ the dual weight $y(a) < 0$. And if $a \in A_F$ then $y(a) = 0$. For such a transport plan σ and the minimum cost maximum transport plan σ' it is

$$w(\sigma) \leq w(\sigma') + \delta' U$$

GOAL: Construct such a 1-optimal transport plan, fulfilling the condition.

DEF: Residual graph

Let σ be a 1-feasible transport plan. The **residual graph** G_σ on the vertex set $A \cup B$ has the following edges:

- (b,a) with capacity $\min\{\overline{d}_a, \overline{s}_b\}$ if:
  $(a,b) \in A \times B: \quad \sigma(a,b) = 0$
- (a,b) with capacity $\sigma(a,b)$ if:
  $(a,b) \in A \times B: \quad \sigma(a,b) = \min\{\overline{d}_a, \overline{s}_b\}$
- (a,b) with capacity $\sigma(a,b)$ and (b,a) with capacity $\min\{\overline{d}_a, \overline{s}_b\} - \sigma(a,b)$ if:
  $0 < \sigma(a,b) < \min\{\overline{d}_a, \overline{s}_b\}$

(a,b) is called a **backward edge**. (b,a) is called a **forward edge**.

Set the cost of all edges:

$$\overline{c}(a,b) = \lfloor 2c(a,b)/\delta' \rfloor$$

Any directed path in the residual network starting from a free supply vertex to a free demand vertex is called an **augmenting path** (alternates between forward and backward edges - with the first and last edges forward edges).

We can augment the supplies transported by $k \geq 1$ units along an augmenting path P as follows:
Raise the flow in every forward edge by k and reduce the flow in every backward edge by k.
Define **slack** on any edge in the residual graph:

$$s(a,b) = \begin{cases} \overline{c}(a,b) + 1 - y(a) - y(b) & \text{if } (a,b) \text{ is a forward edge} \\ y(a) + y(b) - \overline{c}(a,b) & \text{if } (a,b) \text{ is a backward edge} \end{cases}$$

Any edge in the residual graph is called **admissible** if it has zero slack. The **admissible graph** $\mathcal{A}_\sigma$ is the subgraph of the residual graph consisting of only its the admissible edges.

The Algorithm

Set-up: $\forall (a,b) \in A \times B : \quad \sigma(a,b) = 0$
Dual weights $\forall v \in A \cup B : \quad y(v) = 0$
σ and y form a 1-feasible transportation plan.

Start phases - terminate when σ is a maximum transport plan.
Each phase consists of two steps:

1. **Hungarian Search** = Adjust dual weights, create augmenting path of admissible edges:

- **Augmented residual network** G_σ:
  Add a source node s (connected to every free supply node with $\sum_{a \in A} \sigma(a,b) < \overline{s}_b$, weight 0),
  and target node t (every free demand vertex is connected to it, weight 0)
- The weight of every other edge of the residual network is set to its slack
- $\forall v \in A \cup B$ let l_v be the shortest path from s to v in G_σ.
  Perform **dual weight adjustment**: $\forall v \in A \cup B$

$$y(v) = \begin{cases} y(v) & \text{if } l_v \geq l_t \\ y(v) - l_t + l_v & \text{if } l_v < l_t \ \wedge v \in A \\ y(v) - l_t - l_v & \text{if } l_v < l_t \ \wedge v \in B \end{cases}$$

(Now σ and y still form a 1-feasible solution and there is at least one augmenting path in the admissible graph.)
Executable in time $\mathcal{O}(n\Theta(n)\log^2(n))$

2. **Partial DFS**: Compute at least one augmenting path and update σ along it. Afterwards there is no augmenting path of admissible edges left:
   - $\mathcal{A} = \mathcal{A}_\sigma$. Let X denote the set of free supply nodes in $\mathcal{A}$
   - Initiate DFS from each supply node of X in $\mathcal{A}$:
     - If the DFS visits a free demand node - then an augmenting path P is found
     - Delete all visited edges except the ones of P
     - **Augment** σ along P and update X to denote the free supply nodes in $\mathcal{A}$:
       * Define the **bottleneck edge set** as the set of all edges (u,v) on P with the smallest residual capacity $\text{bc}(P)$.
       * Define the **bottleneck capacity** r_P of P as

$$\min\{\overline{s}_b - \sum_{a' \in A} \sigma(a',b), \overline{d}_a - \sum_{b' \in B} \sigma(a,b'), \text{bc}(P)$$

        * Update P by adding r_P to the transport of every forward edge and subtracting r_P to every backward edge

    - If no free demand nodes was found - delete all visited edges and vertices and update X to the remaining free supply nodes.

- If X is empty - go to the first step 1.

Executable in time $\mathcal{O}(n\Theta(n))$

Omitted: page 7 - Correctness and Efficiency of the alg.
... Theoretical execution time of

$$\mathcal{O}(\frac{n^2C}{(1-\epsilon)\delta} + \frac{nC^2}{\epsilon(1-\epsilon)\delta^2})$$

$$\mathcal{O}(\frac{n^2C}{\delta} + \frac{nC^2}{\delta^2})$$

3.1 Experimental Results

Implementation in Java.
MNIST data set of handwritten digits. Supply and demand based on pixel intensities - normalized such that total supply and demand are both 1. Edge cost = squared-Euclidean distance between pixel coordinates, scaled such that maximum edge cost $C = 1$. ($28 \times 28 = 784$ pixels.)

First set of tests: Confirm theoretical analysis.
Result: Significantly less iterations, shorter than worst case augmenting paths (reason: the theoretical results were tight, and the flow increase in practise is higher). The augmentation time was negligible ($\leq$ 2% of the execution time).

Second set of test: Compare with Sinkhorn, Greenkhorn, APDAGD alg.
Divide algs in iterations = portions that take $\mathcal{O}(n^2)$ time.
Result: Unlike others no significant increase in iterations for smaller values of δ (= better

numerical stability).
Outperforms Sinkhorn (the others are not optimized for actual running time) w.r.t. run time (even for added error parameter 5δ to Sinkhorn (relaxation)). But the results (cost of σ) where not always better - but better, if Sinkhorn is relaxed (5δ).

3.2 Extensions

Sinkhorn scales well in parallel settings (parallelize row and column operations).
The presented method - first step = Dijkstra's search is inherently sequential. But there exists an alternate implementaiton of a single scale of Gabow and Tarjan's algorithm - which may be more amenable to parallel implementation.

4 Comparisons

	Pros	Cons
A	Clear definitions	Bad placement of tables and also content Shift of attention (categorical, continuous WWL)
B		„Challenging problem“ - Abstract, not scientific (?)
		Less formal definitions (see (5) and (6), graph alignment pb.)
		Strange typos (page 4 - last paragraph „=of the same size“)
		Redundancies in the Algorithm 1 requirements
		„mild assumptions“ converges „almost surely“ to a critical point - non scien
		Paragraph before 4-Experimental results is extremely vague
		No specifications of the implementation(?)
C		