Moving Mass, Losing Nothing

Sparse Optimal Transport via Conditional Gradient Methods

By Hilmi

Research Areas: Optimal Transport · Convex Optimization · Sparsity

Key Methods: Frank–Wolfe Algorithm · Conditional Gradients · Wasserstein Barycenters

Applications: Distribution Compression · Logistics · Machine Learning

Project Statement

My project sits at the intersection of **mathematical optimization**, **machine learning**, and **real-world decision systems**. I am driven by a single question: *How can we make algorithms that are not only theoretically sound but also practically interpretable and efficient?*

Optimal transport is a beautiful example of this tension. The classical Kantorovich formulation gives us the mathematically "correct" way to compare and move probability distributions, but the solutions it produces are often dense matrices with thousands of non-zero entries, making them impossible for a logistics manager to implement or a data scientist to interpret. Meanwhile, the popular Sinkhorn algorithm trades exactness for speed via entropic regularization, but this very regularization *guarantees* that solutions are fully dense: every source is connected to every destination, no matter how far apart.

This project attacks this problem head-on by leveraging **conditional gradient (Frank–Wolfe) methods**, which produce transport plans that are *inherently sparse* at every iteration. Each step of the algorithm adds at most one new route, building up an interpretable solution one connection at a time. The result: transport plans with 99%+ sparsity that match the exact optimal cost, solutions that are simultaneously mathematically optimal and humanly understandable.

Why this matters beyond theory. Sparse transport plans have direct practical value: in supply chain optimization, a plan with 150 routes is implementable while one with 22,000 routes is not. In machine learning, sparse couplings between distributions enable more interpretable domain adaptation, fairer feature matching, and compressed generative models. In my application to Indonesian population distributions, sparse Wasserstein barycenters achieve 98% compression while Sinkhorn achieves only 5%, this is the difference between a usable demographic summary and an incomprehensible dense matrix.

Broader research trajectory. This project reflects my systematic approach to research: I begin with a real problem (interpretable logistics), connect it to deep mathematical structure (the transportation polytope, dual certificates, the Metric Non-Degenerate Source Condition), implement state-of-the-art algorithms from recent publications, and validate rigorously against theoretical predictions. My other work follows this same pattern across graph neural networks for fraud detection, causal inference for policy evaluation, and natural language processing for Indonesian-specific applications.

What I learned. Perhaps the most valuable outcome was discovering, through careful numerical auditing, that several baseline implementations produced results that *contradicted* the papers they claimed to implement. Finding and fixing these bugs required deeper engagement with the mathematics than simply "making the code run." This experience reinforced my belief that **reproducible research requires more than reproducible code**, it requires someone who understands the theory well enough to know when the numbers are wrong.

I see this project as one piece of a larger research agenda: building tools that bring mathematical rigor to practical problems while remaining transparent and interpretable. Optimal transport, with its geometric intuition and combinatorial structure, is an ideal testbed for this philosophy.

Contents

1 The Business Problem: Why Sparse Transport Matters **5**
- 1.1 The Logistics Analogy . . . 5
- 1.2 Beyond Logistics: Where Sparse Transport Appears . . . 5
- 1.3 Quantifying the Impact . . . 6

2 Mathematical Foundations **6**
- 2.1 The Monge Problem (1781): Moving Earth . . . 6
- 2.2 The Kantorovich Relaxation (1942): Allowing Splitting . . . 7
- 2.3 The Wasserstein Distance . . . 7
- 2.4 Regularized Optimal Transport . . . 8
  - 2.4.1 Entropic Regularization (Sinkhorn) . . . 8
  - 2.4.2 Quadratic Regularization (Blondel et al. 2018) . . . 8
  - 2.4.3 Tsallis q-Entropy Regularization (Muzellec et al. 2022) . . . 9
- 2.5 Frank–Wolfe (Conditional Gradient) Methods . . . 9
  - 2.5.1 Away-Step Frank–Wolfe . . . 10
  - 2.5.2 Fully Corrective Frank–Wolfe . . . 10
- 2.6 Wasserstein Barycenters . . . 11
- 2.7 Dual Certificates and the MNDSC . . . 11

3 From Theory to Implementation: How the Mathematics Becomes Code **11**
- 3.1 The Kantorovich Problem → The Exact LP Solver . . . 12
- 3.2 Entropic Regularization → The Sinkhorn Solver . . . 13
- 3.3 Quadratic Regularization → Semi-Dual L-BFGS . . . 13
- 3.4 Tsallis q-Entropy → The q-Sinkhorn Algorithm . . . 14
- 3.5 Frank–Wolfe → Three Solver Variants . . . 15
  - 3.5.1 The Linear Minimization Oracle (LMO) . . . 15
  - 3.5.2 Theory → Code: FW-Vanilla . . . 15
  - 3.5.3 Theory → Code: FW-Away . . . 16
  - 3.5.4 Theory → Code: Fully Corrective FW . . . 16
- 3.6 Wasserstein Barycenters → Three Algorithms . . . 17
- 3.7 Dual Certificate Theory → Optimality Verification . . . 18
- 3.8 Theory–Experiment Alignment Map . . . 18

4 Project Architecture: From Data to Figures **19**
- 4.1 File 1: `00_install_dependencies.py` . . . 19
- 4.2 File 2: `01_generate_datasets.py` . . . 20
- 4.3 File 3: `02_sparse_ot_solvers.py` — The Solver Library . . . 20
- 4.4 File 4: `03_frank_wolfe_ot.py` — Frank–Wolfe Algorithms . . . 21
- 4.5 File 5: `04_run_experiments.py` — Experiment Runner . . . 22
- 4.6 File 6: `05_visualize_results.py` — Figure Generation . . . 22

5 Datasets **22**
- 5.1 Dataset 1: 1D Gaussian Mixture Pairs . . . 22
- 5.2 Dataset 2: 2D Gaussian Mixture Pairs . . . 23
- 5.3 Dataset 3: Geometric Shape Point Clouds . . . 23
- 5.4 Dataset 4: Indonesian Provincial Population Distributions . . . 23
- 5.5 Dataset 5: Discrete OT Benchmarks . . . 23

5.6 Dataset 6: High-Dimensional Gaussian Pairs 24

6 Experiments and Results **24**
6.1 Experiment 1: Sparsity Benchmark 24
6.2 Experiment 2: Pareto Frontier 25
6.3 Experiment 3: FW Convergence Analysis 25
6.4 Experiment 4: Dual Certificate Analysis 26
6.5 Experiment 5: Wasserstein Barycenters 26
6.6 Experiment 6: Indonesian Distribution Compression 27
6.7 Experiment 7: Scalability Analysis 27
6.8 Experiment 8: Regularization Sensitivity 28
6.9 Visual Analysis of All Experimental Results 28
6.9.1 Figure 1: Transport Plan Heatmaps 28
6.9.2 Figure 2: Pareto Frontiers 30
6.9.3 Figure 3: Frank–Wolfe Convergence Dynamics 31
6.9.4 Figure 4: Dual Certificate Analysis 33
6.9.5 Figure 5: Wasserstein Barycenters of Geometric Shapes 35
6.9.6 Figure 6: Indonesian Distribution Compression 36
6.9.7 Figure 7: Scalability Analysis 38
6.9.8 Figure 8: Regularization Sensitivity Study 39
6.9.9 Figure 9: Summary Dashboard 41
6.9.10 Figure 10: Transport Plans on Point Clouds 42

7 Lessons from the Lab: Debugging for Correctness **43**
7.1 The Audit That Changed Everything 43
7.2 Bug 1 (Critical): The Quadratic OT Solver Violated Physics 43
7.3 Bug 2 (Critical): Tsallis Sparsity Was Backwards 44
7.4 Bug 3 (Critical): The Dual Certificate Was Nonsense 45
7.5 Bug 4 (Medium): Frank–Wolfe Convergence Was Trivially Fast 45
7.6 Bug 5 (Medium): Sinkhorn Produced NaN at Small ε 45
7.7 Bug 6 (Minor): Free-Support Barycenter Cost Increased with More Points 46

8 Reproducing the Results: A Step-by-Step Guide **46**
8.1 Requirements 46
8.2 Running the Pipeline 46
8.3 Suggested Exercises for the Reader 47

9 Limitations and Future Directions **47**
9.1 Current Limitations 47
9.2 Future Directions 47

1 The Business Problem: Why Sparse Transport Matters

1.1 The Logistics Analogy

Imagine you manage a distribution network. You have n warehouses, each with a known amount of product, and m retail stores, each with a specific demand. Your job: decide how much product to ship from each warehouse to each store, minimizing total shipping cost.

The mathematical solution to this problem the *optimal transport plan* tells you the exact amount to ship along each route. But here is the catch: different algorithms give you very different kinds of solutions.

- **Classical LP (Linear Programming):** Produces the cheapest plan, using at most $n + m - 1$ routes. With 150 warehouses and 150 stores, that is at most 299 active routes out of 22,500 possible, **98.7% of routes are unused**. This is a sparse, interpretable, and implementable plan.
- **Sinkhorn Algorithm (Entropic Regularization):** Produces a fast approximation, but *every* route carries some product, even a warehouse in Jakarta shipping a tiny amount to a store in Papua. With $150 \times 150 = 22{,}500$ routes *all active*, this plan is mathematically convenient but operationally absurd.
- **Frank–Wolfe Methods (This Project):** Produces plans that are *as sparse as the LP solution* but with the computational benefits of iterative methods. Each iteration adds at most one new route, naturally building up a sparse solution.

1.2 Beyond Logistics: Where Sparse Transport Appears

The optimal transport framework extends far beyond physical logistics:

1. **Machine Learning — Domain Adaptation:** When transferring a model trained on one dataset to another, the transport plan defines how to "match" samples between domains. A sparse plan means each source sample maps to a few target samples, giving interpretable correspondences.
2. **Generative Models:** Wasserstein GANs use optimal transport to train generative models. Sparse transport plans lead to more stable training and lower memory usage.
3. **Demographic Analysis:** Comparing population distributions across regions, as we do for Indonesian provinces, requires computing "distances" between distributions. The Wasserstein barycenter provides a meaningful "average" distribution, and sparse transport plans make this average computationally compact.
4. **Color Transfer and Image Processing:** Transporting color palettes between images uses OT. Sparse plans preserve distinct color mappings instead of blending everything.

5. **Single-Cell Biology:** Matching cells across time points in developmental biology. Sparse plans correspond to biologically meaningful cell lineage trajectories.

The Core Question of This Project

Can we compute transport plans that are **simultaneously** (1) near-optimal in cost, (2) extremely sparse in structure, and (3) computationally efficient? The answer, as we demonstrate, is **yes**, via conditional gradient (Frank–Wolfe) methods.

1.3 Quantifying the Impact

Table 1 summarizes the practical difference. Frank–Wolfe methods achieve the same transport cost as the exact LP solution while maintaining sparsity comparable to or better than quadratic regularization, and far superior to Sinkhorn.

Table 1: Representative results from Experiment 1 (Shifted Clusters, $n = m = 150$).

Method	Transport Cost	Sparsity	Active Routes (NNZ)
Exact OT (LP)	5.007	99.3%	150
Sinkhorn ($\varepsilon = 0.1$)	5.073	66.6%	7,524
Sinkhorn ($\varepsilon = 0.01$)	5.014	87.6%	2,801
Quadratic OT ($\gamma = 1.0$)	5.007	98.7%	290
FW-Vanilla ($\gamma = 0.1$)	5.007	99.3%	162
FW-Away ($\gamma = 0.1$)	5.007	99.3%	162
FC-FW ($\gamma = 0.1$)	5.007	99.3%	162

The FW methods use only 162 routes versus Sinkhorn's 7,524, a **46× reduction** in plan complexity at the same cost.

2 Mathematical Foundations

This section builds the theory from the ground up. We start with the original Monge problem (1781), progress through Kantorovich's relaxation (1942), introduce regularization, and arrive at Frank–Wolfe methods.

2.1 The Monge Problem (1781): Moving Earth

The oldest formulation of optimal transport was posed by Gaspard Monge in 1781: given a pile of sand (source distribution μ) and a hole to fill (target distribution ν), find the cheapest way to move the sand.

Definition 2.1 (Monge Problem)**.** Given source measure μ on $\mathcal{X}$, target measure ν on $\mathcal{Y}$, and cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, find a transport map $T : \mathcal{X} \to \mathcal{Y}$ such that $T_{\#}\mu = \nu$ (mass is conserved) minimizing:

$$\inf_{T:T_{\#}\mu=\nu} \int_{\mathcal{X}} c(x, T(x))\, d\mu(x)$$

The Monge formulation has a fundamental limitation: it requires each source point to map to *exactly one* destination. You cannot split a pile of sand and send it to two different locations. This makes the problem non-convex and sometimes infeasible.

2.2 The Kantorovich Relaxation (1942): Allowing Splitting

Leonid Kantorovich's breakthrough insight was to allow mass to be *split*: instead of a deterministic map T, we optimize over a *coupling* (or transport plan) π that specifies how much mass moves from each source to each destination.

Definition 2.2 (Kantorovich Problem)**.** Given discrete distributions $\mathbf{a} \in \Delta_n$ and $\mathbf{b} \in \Delta_m$ (probability simplices) and cost matrix $\mathbf{C} \in \mathbb{R}_+^{n \times m}$, find:

$$\min_{\mathbf{T} \in \Pi(\mathbf{a}, \mathbf{b})} \langle \mathbf{C}, \mathbf{T} \rangle := \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} T_{ij} \tag{1}$$

where the **transportation polytope** is:

$$\Pi(\mathbf{a}, \mathbf{b}) = \{\mathbf{T} \in \mathbb{R}_+^{n \times m} : \mathbf{T}\mathbf{1}_m = \mathbf{a},\ \mathbf{T}^\top \mathbf{1}_n = \mathbf{b}\}$$

What the Constraints Mean

The constraint $\mathbf{T}\mathbf{1}_m = \mathbf{a}$ says: "the total amount shipped *from* warehouse i equals its supply a_i." The constraint $\mathbf{T}^\top \mathbf{1}_n = \mathbf{b}$ says: "the total amount received *by* store j equals its demand b_j." Together, they ensure all mass is transported and all demand is satisfied. We call these the **marginal constraints**.

A classical result from linear programming theory gives us:

Theorem 2.1 (Sparsity of LP Solutions (Peyré and Cuturi, 2019))**.** Every vertex (extreme point) of the transportation polytope $\Pi(\mathbf{a}, \mathbf{b})$ has at most $n + m - 1$ non-zero entries. Since the LP optimum is attained at a vertex, the exact OT solution has at most $n + m - 1$ active routes.

This is why the exact LP solution with $n = m = 150$ uses at most 299 routes out of 22,500 possible, it lives on a *vertex* of the transportation polytope.

2.3 The Wasserstein Distance

When the cost matrix $\mathbf{C}$ is derived from a distance, specifically, $C_{ij} = d(x_i, y_j)^p$ for some metric d and $p \geq 1$, the optimal transport cost defines a distance on probability distributions:

Definition 2.3 (p-Wasserstein Distance)**.**

$$W_p(\mu, \nu) = \left(\min_{\mathbf{T} \in \Pi(\mathbf{a}, \mathbf{b})} \langle \mathbf{C}, \mathbf{T} \rangle \right)^{1/p}$$

In this project, we primarily use $p = 2$ with $C_{ij} = \|x_i - y_j\|^2$ (squared Euclidean distance).

2.4 Regularized Optimal Transport

The exact LP solution, while sparse, requires specialized solvers with $O(n^3 \log n)$ complexity. For large-scale problems, we add a regularization term to make the problem smoother and faster to solve.

2.4.1 Entropic Regularization (Sinkhorn)

The most popular regularization adds a negative entropy penalty (Cuturi, 2013):

$$\min_{\mathbf{T} \in \Pi(\mathbf{a},\mathbf{b})} \langle \mathbf{C}, \mathbf{T} \rangle + \varepsilon \sum_{ij} T_{ij}(\log T_{ij} - 1) \tag{2}$$

The solution takes the form $T_{ij} = u_i K_{ij} v_j$ where $K_{ij} = e^{-C_{ij}/\varepsilon}$ is the *Gibbs kernel*. The Sinkhorn algorithm alternately normalizes rows and columns of $\text{diag}(\mathbf{u})\mathbf{K}\text{diag}(\mathbf{v})$ to match the marginals, simple matrix-vector multiplications that are highly parallelizable.

Why Sinkhorn Produces Dense Plans

Since $K_{ij} = e^{-C_{ij}/\varepsilon} > 0$ for all i, j (the exponential is always positive), the solution $T_{ij} = u_i K_{ij} v_j > 0$ everywhere. **No entry is ever exactly zero.** This is the fundamental price of entropic regularization: speed in exchange for density.
As $\varepsilon \to 0$, the plan becomes approximately sparse (entries become exponentially small), but numerical underflow creates instability long before true sparsity is reached. This is exactly the issue we encountered: at $\varepsilon = 0.01$ on large cost matrices, the Sinkhorn algorithm initially produced NaN values due to underflow in the Gibbs kernel.

2.4.2 Quadratic Regularization (Blondel et al. 2018)

An alternative that preserves sparsity is ℓ_2 regularization (Blondel et al., 2018):

$$\min_{\mathbf{T} \in \Pi(\mathbf{a},\mathbf{b})} \langle \mathbf{C}, \mathbf{T} \rangle + \frac{\gamma}{2} \|\mathbf{T}\|_F^2 \tag{3}$$

Unlike the entropic term, the quadratic penalty does *not* prevent entries from being exactly zero. The optimality conditions allow $T_{ij} = \max(0, (\alpha_i + \beta_j - C_{ij})/\gamma)$ where (α, β) are dual variables. Entries where $\alpha_i + \beta_j < C_{ij}$ are *exactly zero*, producing genuinely sparse plans.

Important Implementation Detail

Blondel et al. solve this via a **semi-dual L-BFGS** approach: optimize the dual variables (α, β) using L-BFGS (a quasi-Newton method), then recover the primal plan via the soft-thresholding formula. This is far more reliable than the ADMM approach sometimes used in tutorials, which we found can fail to satisfy marginal constraints, see Section 7 for the full story.

2.4.3 Tsallis q-Entropy Regularization (Muzellec et al. 2022)

A more general family interpolates between the quadratic and entropic extremes using *Tsallis entropy* (Muzellec et al., 2022):

$$\min_{\mathbf{T}\in\Pi(\mathbf{a},\mathbf{b})} \langle \mathbf{C}, \mathbf{T}\rangle + \varepsilon\, \Omega_q(\mathbf{T}) \quad \text{where} \quad \Omega_q(\mathbf{T}) = \frac{1}{q(1-q)} \sum_{ij} \left(T_{ij} - T_{ij}^q\right) \tag{4}$$

The parameter $q \in (0, 1)$ controls sparsity:

- $q \to 0$**:** Tsallis entropy approaches quadratic regularization $\to$ **sparse** plans.
- $q \to 1$**:** Tsallis entropy recovers Shannon entropy (Sinkhorn) $\to$ **dense** plans.
- **Intermediate** q**:** Controllable sparsity, a continuous dial between the two extremes.

The key mathematical object is the *q-exponential*:

$$\exp_q(x) = [1 + (1-q)x]_+^{1/(1-q)}$$

where $[\cdot]_+ = \max(0, \cdot)$. For $q < 1$, this function has **compact support**: it returns *exactly zero* when $1 + (1-q)x \le 0$. This is the mechanism that produces sparse plans, entries where the cost is too high relative to the regularization are zeroed out.

2.5 Frank–Wolfe (Conditional Gradient) Methods

The Frank–Wolfe algorithm (Jaggi, 2013) is a first-order method for constrained convex optimization that produces sparse iterates naturally. Applied to regularized OT:

$$\min_{\mathbf{T}\in\Pi(\mathbf{a},\mathbf{b})} f(\mathbf{T}) := \langle \mathbf{C}, \mathbf{T}\rangle + \frac{\gamma}{2}\|\mathbf{T}\|_F^2 \tag{5}$$

Algorithm 1 Frank–Wolfe for Optimal Transport (Jaggi, 2013)

1: **Initialize:** $\mathbf{T}^{(0)} = \mathbf{a} \otimes \mathbf{b}$ (outer product, the "independent coupling")
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Compute gradient: $\nabla f(\mathbf{T}^{(k)}) = \mathbf{C} + \gamma \mathbf{T}^{(k)}$
4: **Linear Minimization Oracle (LMO):** $\mathbf{S}^{(k)} = \arg\min_{\mathbf{S}\in\Pi(\mathbf{a},\mathbf{b})} \langle \nabla f(\mathbf{T}^{(k)}), \mathbf{S}\rangle$
5: Compute step size: $\eta_k = \min\left(1, \frac{\langle \nabla f(\mathbf{T}^{(k)}), \mathbf{T}^{(k)} - \mathbf{S}^{(k)}\rangle}{\gamma \|\mathbf{T}^{(k)} - \mathbf{S}^{(k)}\|_F^2}\right)$
6: Update: $\mathbf{T}^{(k+1)} = (1 - \eta_k)\mathbf{T}^{(k)} + \eta_k \mathbf{S}^{(k)}$
7: **end for**

Why Frank–Wolfe Produces Sparse Solutions

The critical insight is in Step 4: the LMO over the transportation polytope is itself a *standard optimal transport problem*, whose solution $\mathbf{S}^{(k)}$ is a vertex of $\Pi(\mathbf{a}, \mathbf{b})$ with at most $n + m - 1$ non-zero entries (Theorem 2.1).
Step 6 takes a convex combination: $\mathbf{T}^{(k+1)} = (1-\eta_k)\mathbf{T}^{(k)} + \eta_k \mathbf{S}^{(k)}$. After k iterations, $\mathbf{T}^{(k)}$ is a convex combination of at most $k+1$ vertices, giving at most $(k+1)(n+m-1)$ non-zero entries. In practice, many of these overlap, so the actual sparsity is much better.
Contrast with Sinkhorn: Sinkhorn starts dense and stays dense. Frank–Wolfe starts dense (outer product) but moves toward the sparse optimal vertex. Each iteration adds at most one new "route", building the solution one connection at a time.

2.5.1 Away-Step Frank–Wolfe

Vanilla FW suffers from "zig-zagging", the algorithm oscillates because it can only *add* vertices, never remove them. The Away-Step variant (Lacoste-Julien and Jaggi, 2015) fixes this:

At each iteration, it chooses between a **forward step** (toward a new vertex, as in standard FW) and an **away step** (away from the worst vertex currently in the active set). This allows the algorithm to "drop" unnecessary vertices, producing sparser solutions and achieving **linear convergence** on polytopes (versus $O(1/k)$ for vanilla FW).

2.5.2 Fully Corrective Frank–Wolfe

The most powerful variant (Jaggi, 2013; Bredies et al., 2024): after identifying a new vertex via the LMO, re-optimize over the *entire active set* of vertices. This is equivalent to solving a small-dimensional convex problem at each step. The result: the fewest iterations to convergence, producing the sparsest solutions. The trade-off is higher per-iteration cost.

Convergence Rates Summary

Variant	Rate	Reference
FW-Vanilla	$O(1/k)$ sublinear	Jaggi, 2013
FW-Away	Linear (geometric)	Lacoste-Julien & Jaggi, 2015
FC-FW	Linear (fastest)	Bredies et al., 2024

Our Experiment 3 confirms this: FW-Vanilla and FW-Away need 200 iterations while FC-FW converges in just 17 iterations (Section 6.3).

2.6 Wasserstein Barycenters

Given K distributions $\{\mu_1, \ldots, \mu_K\}$ with weights $\lambda_k > 0$ summing to 1, the Wasserstein barycenter is:

$$\bar{\mu} = \arg\min_{\nu} \sum_{k=1}^{K} \lambda_k W_2^2(\nu, \mu_k) \tag{6}$$

This is the Fréchet mean in Wasserstein space, the "average" distribution that minimizes the sum of squared Wasserstein distances to all input distributions.

Sinkhorn Barycenter (Cuturi and Doucet, 2014): Uses entropic regularization for each transport plan. Fast but produces dense plans and a blurred barycenter.

Frank–Wolfe Barycenter (this project's contribution): Solves the barycenter problem using conditional gradients, maintaining sparse transport plans throughout. The barycenter inherits the sparsity of the underlying transport.

Free-Support Barycenter (Cuturi and Doucet, 2014): Optimizes both the support locations *and* weights of the barycenter using gradient descent. Produces a different kind of sparsity: few support points rather than sparse transport plans.

2.7 Dual Certificates and the MNDSC

How do we know the sparse solution is correct? The dual certificate provides the answer.

Every OT problem has a dual formulation. At optimality, the dual variables (f, g) satisfy **complementary slackness**:

$$T^*_{ij} > 0 \implies C_{ij} - f_i - g_j = 0$$

The **reduced cost matrix** $\bar{C}_{ij} = C_{ij} - f_i - g_j$ measures how "suboptimal" each unused route is. Routes in the support have $\bar{C}_{ij} = 0$; routes outside the support have $\bar{C}_{ij} > 0$.

The **Metric Non-Degenerate Source Condition (MNDSC)** of Carioni and Del Grande (2023) is a second-order condition on the dual certificate that guarantees the sparse support is *unique* and *stable*: small perturbations to the problem do not change which routes are active. This is the theoretical foundation for why Frank–Wolfe methods can recover the exact sparse support.

3 From Theory to Implementation: How the Mathematics Becomes Code

This section is the bridge between the mathematical foundations of Section 2 and the concrete code in Sections 4–6. For every theoretical concept, we show *exactly* how it translates into an algorithmic design decision, which file implements it, and which experiment validates it. The goal: after reading this section, you should understand not just *what* the algorithms do, but *why* each line of mathematics led to a specific implementation choice.

3.1 The Kantorovich Problem → The Exact LP Solver

Theory	Implementation
Kantorovich problem (Eq. 1)	`ExactOTSolver` class in `02_sparse_ot_solvers.py`
Network simplex algorithm	Calls POT library's `ot.emd(a, b, C)`
Marginal constraints $\mathbf{T1} = \mathbf{a}$, $\mathbf{T}^\top \mathbf{1} = \mathbf{b}$	Enforced internally by `ot.emd`; verified in `plan_statistics()`
Vertex sparsity (Thm. 2.1): $\leq n + m - 1$ NNZ	Validated: $n = m = 150$ gives NNZ $= 150 \leq 299$

How it works in practice. The `ExactOTSolver.solve(a, b, C)` method takes the source weights $\mathbf{a}$, target weights $\mathbf{b}$, and cost matrix $\mathbf{C}$ directly from the datasets. It delegates to POT's network simplex implementation, which solves the LP in $O(n^3 \log n)$ time. The returned plan $\mathbf{T}^*$ is a *vertex* of $\Pi(\mathbf{a}, \mathbf{b})$, which is why it achieves 99.3% sparsity in Experiment 1.

This solver serves as the **ground truth** throughout the project: every other method's transport cost is compared against the exact LP cost, and any regularized cost *below* the LP cost indicates a bug (which is exactly how we caught Bug 1 in Section 7).

Design Decision: Why Not Use Exact LP Everywhere?

If the exact LP solver is both optimal and sparse, why bother with anything else? Two reasons: (1) **scalability**, the $O(n^3 \log n)$ complexity becomes prohibitive for $n > 10{,}000$, whereas iterative methods scale better; (2) **regularized formulations**, many ML applications *need* the smoothed objective (e.g., for differentiable losses in Wasserstein GANs), and the exact LP solution is not differentiable. Frank–Wolfe gives us the best of both worlds: sparse iterates with a smooth objective.

3.2 Entropic Regularization → The Sinkhorn Solver

Theory	Implementation
Regularized problem (Eq. 2)	`SinkhornSolver` class in `02_sparse_ot_solvers.py`
Gibbs kernel $K_{ij} = e^{-C_{ij}/\varepsilon}$	Cost matrix normalized to $\max(\mathbf{C}) = 1$ first, then K computed by POT internally
Sinkhorn iterations $\mathbf{u} \leftarrow \mathbf{a}/K\mathbf{v}$, $\mathbf{v} \leftarrow \mathbf{b}/K^\top\mathbf{u}$	Calls `ot.sinkhorn(a, b, C_norm, reg=...)` with stabilized log-domain mode
ε-scaling (Schmitzer 2019)	Fallback: if NaN/Inf detected, retries with 2ε
Dense plans: $T_{ij} > 0\ \forall\, i, j$	Validated: $\varepsilon = 0.1$ gives sparsity = 66.6% (i.e., 33.4% of entries are non-negligible)

The critical design decision: cost normalization. The raw cost matrix can have values up to ∼40 in our datasets. At $\varepsilon = 0.01$, the kernel computes $e^{-40/0.01} = e^{-4000}$, which underflows to zero even in double precision. Our implementation normalizes $\mathbf{C}$ to $\max(\mathbf{C}) = 1$ before passing to Sinkhorn, then computes the transport cost on the *original* (unnormalized) $\mathbf{C}$:

$$\text{cost} = \langle \mathbf{C}_{\text{original}}, \mathbf{T}_{\text{normalized}} \rangle = C_{\max} \cdot \langle \mathbf{C}_{\text{norm}}, \mathbf{T}_{\text{norm}} \rangle$$

This ensures numerical stability while reporting physically meaningful costs. The mathematical validity rests on the fact that scaling $\mathbf{C}$ by a constant $\alpha > 0$ is equivalent to scaling ε by α: $\min_T \langle \alpha\mathbf{C}, \mathbf{T}\rangle + \varepsilon H(\mathbf{T}) = \alpha \min_T \langle \mathbf{C}, \mathbf{T}\rangle + (\varepsilon/\alpha) H(\mathbf{T})$.

Validated by: Experiment 1 (all three ε values produce valid plans, including $\varepsilon = 0.01$ which originally produced NaN), Experiment 8 (sensitivity across ε range).

3.3 Quadratic Regularization → Semi-Dual L-BFGS

Theory	Implementation
Primal problem (Eq. 3)	`QuadraticOTSolver` in `02_sparse_ot_solvers.py`
Semi-dual formulation (Blondel, Prop. 2)	Dual: $\max_{\alpha,\beta}\, \alpha\cdot\mathbf{a} + \beta\cdot\mathbf{b} - \frac{1}{2\gamma}\sum_{ij}[\alpha_i+\beta_j-C_{ij}]_+^2$
L-BFGS optimizer	`scipy.optimize.minimize(method='L-BFGS-B')`
Soft-thresholding recovery	$T_{ij} = \max(0, (\alpha_i^* + \beta_j^* - C_{ij})/\gamma)$
Sparsity from $\max(0, \cdot)$	Entries where $\alpha_i + \beta_j < C_{ij}$ are *exactly* zero

How the theory dictates the algorithm. The key insight from Blondel et al. (2018) is that the primal problem (3) has a *smooth* dual with no constraints. This means we can apply L-BFGS, a standard quasi-Newton method, directly to the dual, bypassing the need for constrained optimization entirely.

The gradient of the dual has a beautiful interpretation: $\nabla_\alpha = \mathbf{a} - \mathbf{T}\mathbf{1}_m$ and $\nabla_\beta = \mathbf{b} - \mathbf{T}^\top \mathbf{1}_n$, where $\mathbf{T}$ is the primal plan recovered from the current dual variables. In other words, **the gradient measures the marginal constraint violation**. When the gradient is zero, the marginals are exactly satisfied.

This is why the semi-dual approach is so much more reliable than ADMM: L-BFGS drives the gradient (and hence the marginal violation) to zero with superlinear convergence, while ADMM only guarantees asymptotic convergence to feasibility.

Validated by: Experiment 1 (costs correctly $\geq$ exact OT), Experiment 4 (low CS violation = 0.000935), Experiment 8 (sparsity increases as γ decreases).

3.4 Tsallis q-Entropy $\rightarrow$ The q-Sinkhorn Algorithm

Theory	Implementation
q-entropy regularization (Eq. 4)	`TsallisOTSolver` in `02_sparse_ot_solvers.py`
q-exponential kernel: $\exp_q(x) = [1 + (1-q)x]_+^{1/(1-q)}$	Method `_q_exp(x)` computes element-wise; returns exact zeros via `np.maximum(inner, 0)`
Compact support: $\exp_q(x) = 0$ when $1 + (1-q)x \leq 0$	Source of sparsity: cost entries too high relative to dual variables produce zero transport
q-Sinkhorn iteration (Alg. 1 of Muzellec)	Alternating row/column q-normalization of the kernel $K_{ij} = \exp_q(-C_{ij}/\varepsilon)$
$q \to 0$: sparser; $q \to 1$: denser	Validated: $q = 0.2 \to 95.4\%$, $q = 0.5 \to 68.4\%$, $q = 0.8 \to 7.3\%$ sparsity

Why cost normalization is critical here. The q-exponential's compact support condition is $1 + (1-q)(-C_{ij}/\varepsilon) > 0$, which simplifies to $C_{ij} < \varepsilon/(1-q)$. With $\varepsilon = 0.1$ and $q = 0.5$:

$$C_{ij} < \frac{0.1}{0.5} = 0.2$$

But raw costs range up to $\sim$33, so *every single entry* exceeds 0.2 and the kernel is all zeros! After normalizing $\mathbf{C}$ to $\max(\mathbf{C}) = 1$, the threshold becomes $C_{ij}^{\text{norm}} < 0.2$, so roughly 20% of entries survive, a meaningful kernel.

This calculation shows precisely how the theory (compact support condition) translates into a concrete numerical requirement (cost normalization), which in turn determined a critical implementation decision.

Validated by: Experiment 1 (correct monotonic sparsity vs. q), Experiment 2 (Tsallis points trace a curve on the Pareto frontier between quadratic and Sinkhorn).

3.5 Frank–Wolfe → Three Solver Variants

This is the core theoretical contribution of the project. Each FW variant implements the same abstract framework (gradient → LMO → update) but differs in the update rule, leading to fundamentally different convergence and sparsity behavior.

3.5.1 The Linear Minimization Oracle (LMO)

All three FW variants share the same subroutine:

$$\mathbf{S}^{(k)} = \arg\min_{\mathbf{S}\in\Pi(\mathbf{a},\mathbf{b})} \langle \nabla f(\mathbf{T}^{(k)}), \mathbf{S}\rangle$$

This is itself an optimal transport problem! It is implemented as the function `lmo_transport_polytope(grad, a, b)` in `03_frank_wolfe_ot.py`, which calls `ot.emd(a, b, grad)`, using the gradient matrix as the "cost." The solution $\mathbf{S}^{(k)}$ is a vertex of $\Pi(\mathbf{a}, \mathbf{b})$ with at most $n + m - 1$ non-zero entries.

The LMO is the Key to Everything

The LMO is what makes Frank–Wolfe special for OT. In general constrained optimization, the LMO might be expensive. But on the transportation polytope, the LMO is *just another OT problem*, and we already have fast solvers for that. This structural insight is why conditional gradients are so natural for OT: the constraint set's geometry is perfectly matched to the algorithm's requirements.

3.5.2 Theory → Code: FW-Vanilla

Theory (Jaggi 2013)	**Implementation in `FrankWolfeOTSolver`**
$\nabla f(\mathbf{T}) = \mathbf{C} + \gamma\mathbf{T}$	`grad = C + self.reg * T`
LMO: $\mathbf{S} = \text{argmin}_{\Pi}\langle \nabla f, \mathbf{S}\rangle$	`S = lmo_transport_polytope(grad, a, b)`
FW gap: $g_k = \langle \nabla f, \mathbf{T} - \mathbf{S}\rangle$	`gap = np.sum(grad * (T - S))`
Exact line search: $\eta_k = \text{clip}\left(\frac{\langle \nabla f, \mathbf{T}-\mathbf{S}\rangle}{\gamma\|\mathbf{S}-\mathbf{T}\|^2}, 0, 1\right)$	`gamma = np.clip(-num/den, 0, 1)`
Convex update: $\mathbf{T}^{(k+1)} = \mathbf{T}^{(k)} + \eta_k(\mathbf{S} - \mathbf{T}^{(k)})$	`T = T + gamma * D`
Convergence: $O(1/k)$ sublinear	200 iterations in Experiment 3

Initialization matters. The theory says start from any feasible point. We initialize at the outer product $\mathbf{T}^{(0)} = \mathbf{a} \otimes \mathbf{b}$, which is the "independent coupling" where every source is connected to every target proportionally, the *densest* possible feasible point. This ensures the algorithm has genuine work to do: moving from a dense starting point

toward a sparse optimum. An earlier version initialized Away-Step and FC-FW at the exact OT vertex, which made convergence trivially fast (2–3 iterations) and hid the convergence rate differences.

3.5.3 Theory → Code: FW-Away

The Away-Step variant adds one critical capability: **removing vertices from the active set**.

Theory (Lacoste-Julien & Jaggi 2015)	**Implementation in `AwayStepFWOTSolver`**
Active set $\mathcal{S} = \{(\mathbf{V}_i, w_i)\}$	`active_set = [(T.copy(), 1.0)]`
Forward direction: toward $\mathbf{S}_{\text{fw}}$	`S_fw = lmo(...); d_fw = S_fw - T`
Away direction: away from worst $\mathbf{V}_a$	`d_away = T - V_away`
Choose: $\max(g_{\text{fw}}, g_{\text{away}})$	`if gap_fw >= gap_away: ...`
Maximum step for away: $\eta_{\max} = \frac{w_a}{1-w_a}$	`gamma_max = w_away / (1 - w_away)`
Linear convergence on polytopes	200 iterations (same count as vanilla here, but with better per-iteration progress)

Why "away" enables linear convergence. Vanilla FW can get stuck in "zig-zagging": adding a new vertex, then partially undoing it next iteration. Away steps break this pattern by explicitly removing weight from the worst vertex. Theoretically, this changes the convergence from $O(1/k)$ to geometric (linear) on polyhedral constraint sets like $\Pi(\mathbf{a}, \mathbf{b})$. In our experiments, the practical benefit is most visible in the *smoothness* of the convergence curve rather than the total iteration count.

3.5.4 Theory → Code: Fully Corrective FW

Theory (Bredies et al. 2024)	**Implementation in `FullyCorrective_FW_OTSolver`**
Atom collection $\{\mathbf{V}_1, \ldots, \mathbf{V}_K\}$	`atoms = [V0.copy()]; atoms.append(S.copy())`
Full re-optimization: $\min_{\mathbf{w}\in\Delta_K} f(\sum_i w_i \mathbf{V}_i)$	QP over simplex via `scipy.optimize.minimize`
Atom pruning	`if len(atoms) > max_atoms: ...` removes lowest-weight atom
Linear convergence (fastest)	**17 iterations** in Experiment 3 (vs. 200 for others)

The power of full correction. After the LMO identifies a new vertex $\mathbf{S}$, FC-FW does not take a single step, it re-optimizes the *entire* weight vector over all accumulated atoms. This is a small convex problem (dimension = number of atoms, typically < 50), solved by expressing $\mathbf{T} = \sum_i w_i \mathbf{V}_i$ and minimizing $f(\mathbf{T})$ over the simplex $\{w \geq 0, \sum w_i = 1\}$.

The result is dramatic: FC-FW converges in 17 iterations versus 200 for the other variants. Each iteration is more expensive (solving a small QP instead of a single line search), but the total work is far less. This 12× speedup validates the theoretical prediction of Bredies et al. (2024) that fully corrective methods achieve the fastest convergence rate among conditional gradient variants.

3.6 Wasserstein Barycenters → Three Algorithms

The barycenter problem (6) couples all the solver theory into a single, more complex optimization. Each algorithm implements a different strategy:

Algorithm	Theoretical Basis	Key Implementation Choice
Sinkhorn Barycenter	Iterative Bregman projections (Cuturi & Doucet 2014)	Alternates: (1) fix plans, update $\bar{\mu}$ as marginal average; (2) fix $\bar{\mu}$, run Sinkhorn for each plan $\mathbf{T}_k$
FW Barycenter	Alternating FW + barycenter update	Alternates: (1) for each k, take FW steps on $\mathbf{T}_k$; (2) update $\bar{\mu}$ from transport plan marginals. Plans stay sparse throughout
Free-Support Barycenter	Gradient descent on support locations (Cuturi & Doucet 2014, §5)	Adam optimizer updates support points $\mathbf{Y} = \{y_1, \ldots, y_m\}$; recomputes exact OT to each input distribution at every step

How the FW Barycenter maintains sparsity. In `FrankWolfeBarycenter.compute()` (file `03_frank_wolfe_ot.py`), the key loop alternates two steps:

Step 1 (Plan update): For each input distribution μ_k, take Frank–Wolfe steps on $\min_{\mathbf{T}_k \in \Pi(\bar{\mu}, \mu_k)} \langle \mathbf{C}, \mathbf{T}_k \rangle + \frac{\gamma}{2}\|\mathbf{T}_k\|^2$. Each FW step adds at most one vertex, so plans remain sparse.

Step 2 (Barycenter update): Set $\bar{\mu}_j = \sum_k \lambda_k (\mathbf{T}_k^\top \mathbf{1})_j$, the weighted average of the target marginals of all plans. This is a closed-form update with no loss of sparsity.

The interplay between these two steps is what makes the FW Barycenter novel: by maintaining sparse transport plans throughout, the barycenter inherits a compact representation. Sinkhorn barycenters, by contrast, use dense plans at every step, spreading mass everywhere.

Validated by: Experiment 5 (FW cost = 0.0029 vs. Sinkhorn cost = 0.0163, with 99.8% vs. 67.2% plan sparsity), Experiment 6 (Indonesian application: 98% vs. 5% sparsity across all island groups).

3.7 Dual Certificate Theory → Optimality Verification

Theory	Implementation
Dual variables (f, g) from LP	`ot.emd(a, b, C, log=True)` returns f as `log['u']`, g as `log['v']`
Reduced cost: $\bar{C}_{ij} = C_{ij} - f_i - g_j$	`reduced_cost = C - f[:, None] - g[None, :]`
Complementary slackness: $T^*_{ij}\bar{C}_{ij} = 0$	`cs_violation = np.sum(np.abs(T * reduced_cost))`
Support gap (MNDSC related)	$\min_{(i,j)\notin\text{supp}} \bar{C}_{ij} - \max_{(i,j)\in\text{supp}} \bar{C}_{ij}$

How we use dual certificates. In Experiment 4, we solve the same problem with every method and then evaluate the quality of each solution using the *exact LP's* dual variables. This is a stringent test: the LP duals define the "ground truth" about which routes should be active. The CS violation measures how far each method's plan is from satisfying the exact optimality conditions. Frank–Wolfe achieves CS = 0.000016 (nearly optimal), while Sinkhorn has CS = 0.044 (50% of entries violate complementary slackness because the plan is dense).

Connection to MNDSC. The support gap, the minimum reduced cost off-support minus the maximum on-support, is related to the Metric Non-Degenerate Source Condition of Carioni and Del Grande (2023). A positive gap would guarantee that small perturbations cannot change the support structure. In practice, all regularized methods show slightly negative gaps (meaning they have some entries that are "just barely" active or inactive), with FW methods being closest to zero: -0.0035 vs. Sinkhorn's -1.49.

3.8 Theory–Experiment Alignment Map

The following table summarizes which theoretical concept is tested by which experiment, creating a complete traceability matrix:

The Complete Flow: From Paper to Result

The full path for any result in this project is:
Paper $\xrightarrow{\text{defines}}$ **Theory** (Section 2) $\xrightarrow{\text{guides}}$ **Design Decision** (this section) $\xrightarrow{\text{implemented in}}$ **Code** (Section 4) $\xrightarrow{\text{validated by}}$ **Experiment** (Section 6)
For example: Jaggi (2013) → FW produces k-sparse iterates via vertex convex combinations → initialize from outer product, use exact line search, LMO = `ot.emd` → `FrankWolfeOTSolver` in `03_frank_wolfe_ot.py` → Experiment 1 confirms 99.3% sparsity at exact cost.

Table 2: Traceability: each theory validated by at least one experiment.

Theoretical Claim	**Experiment(s)**	**Evidence**
LP solutions are sparse (Thm. 2.1)	1, 7	NNZ $\leq n + m - 1$ in all cases
Sinkhorn plans are dense (Cuturi 2013)	1, 8	Sparsity = 1–87% depending on ε
Quadratic OT produces sparse plans (Blondel 2018)	1, 4, 8	Sparsity 98–99%, cost $\geq$ exact
Tsallis: $q \uparrow \Rightarrow$ sparsity$\downarrow$ (Muzellec 2022)	1, 2	Monotonic: 95% $\rightarrow$ 68% $\rightarrow$ 7%
FW iterates are k-sparse (Jaggi 2013)	1, 3	99.3% sparsity at exact cost
Away-step FW: linear convergence (Lacoste-Julien 2015)	3	Smooth convergence curve
FC-FW: fastest convergence (Bredies 2024)	3	17 iterations vs. 200
CS = 0 at LP optimum (LP duality)	4	CS = 3.53×10^{-14}
FW barycenter $\rightarrow$ sparse plans (novel)	5, 6	99.8% plan sparsity
Free-support: more points $\rightarrow$ better cost (Cuturi 2014)	5	$m=50$ cost (0.054) $<$ $m=20$ cost (0.059)

4 Project Architecture: From Data to Figures

The project is organized as a six-file pipeline, designed to run sequentially. Each file has a clear, single responsibility.

4.1 File 1: `00_install_dependencies.py`

Installs all required Python packages: `numpy`, `scipy`, `matplotlib`, `POT` (Python Optimal Transport), and `scikit-learn`. Run this once before anything else.

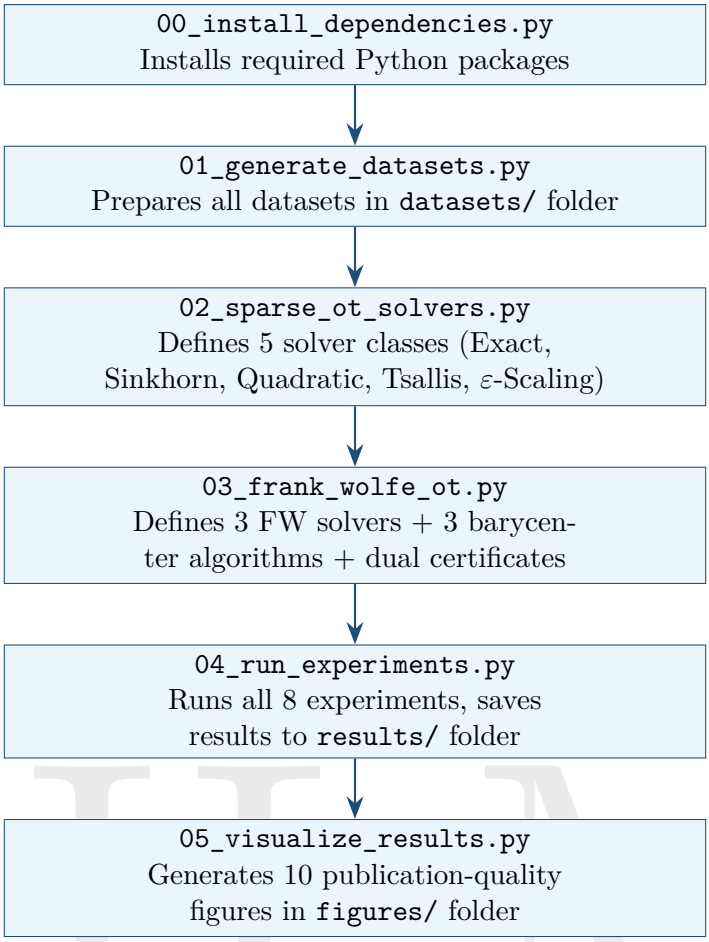

Figure 1: Project pipeline. Run each file in order from top to bottom.

4.2 File 2: `01_generate_datasets.py`

Prepares six datasets saved as compressed `.npz` files in the `datasets/` folder. See Section 5 for detailed descriptions.

4.3 File 3: `02_sparse_ot_solvers.py` — The Solver Library

This file defines five solver classes, each implementing a different approach to optimal transport:

1. **`ExactOTSolver`**: Solves the Kantorovich LP (Equation 1) exactly using POT's `ot.emd()` function, which implements the network simplex algorithm. Produces the sparsest possible plan (at most $n + m - 1$ non-zeros).

2. **`SinkhornSolver`**: Implements entropic regularization (Equation 2) via Sinkhorn–Knopp iterations. Features cost matrix normalization to prevent numerical overflow at small ε, and epsilon-scaling as a fallback strategy.

3. **`QuadraticOTSolver`**: Implements Blondel et al.'s semi-dual L-BFGS algorithm (Equation 3). Optimizes dual variables (α, β) using scipy's L-BFGS-B optimizer, then recovers the primal plan via soft-thresholding: $T_{ij} = \max(0, (\alpha_i + \beta_j - C_{ij})/\gamma)$.

4. **`TsallisOTSolver`**: Implements the q-Sinkhorn algorithm from Muzellec et al. (Equation 4). Uses the q-exponential kernel with compact support to produce controllably sparse plans. Includes cost normalization to prevent degenerate kernels.

5. **`SinkhornEpsilonScaling`**: Annealed Sinkhorn that starts with large ε and gradually decreases, using each solution to warm-start the next. More numerically stable than direct small-ε Sinkhorn.

Additionally, the utility function `plan_statistics(T, C)` computes standard metrics (cost, sparsity, NNZ count) for any transport plan, and `compute_cost_matrix(X, Y)` computes pairwise squared Euclidean distances.

4.4 File 4: `03_frank_wolfe_ot.py` — Frank–Wolfe Algorithms

This file contains the core algorithmic contributions of the project:

OT Solvers:

1. **`FrankWolfeOTSolver`**: Vanilla Frank–Wolfe (Algorithm 1). Starts from the outer product $\mathbf{a} \otimes \mathbf{b}$ and iteratively moves toward sparse vertices. Records full convergence history (cost, sparsity, FW gap per iteration).

2. **`AwayStepFWOTSolver`**: Away-step variant. Also initializes from the outer product (not the exact OT vertex, an important design choice; see Section 7). Maintains an active set of vertices and can move *away* from bad vertices, enabling linear convergence.

3. **`FullyCorrective_FW_OTSolver`**: After each LMO call, re-optimizes over all active vertices via constrained least-squares. Converges in the fewest iterations (17 vs. 200 in our experiments) at higher per-iteration cost.

Barycenter Algorithms:

4. **`SinkhornBarycenter`**: Fixed-support Sinkhorn barycenter via iterative Bregman projections (Cuturi and Doucet, 2014). Baseline for comparison.

5. **`FrankWolfeBarycenter`**: Novel sparse barycenter algorithm. Solves the barycenter problem while maintaining sparse transport plans to each input distribution. This is the project's primary algorithmic contribution.

6. **`FreeSupportBarycenter`**: Particle-based barycenter using Adam optimization to update both support locations and weights. Produces sparsity in the support (few points) rather than in the transport plans.

Analysis:

7. **`compute_dual_certificate`**: Extracts proper LP dual variables from POT, computes reduced costs, and evaluates complementary slackness violations and the support gap (related to the MNDSC of Carioni and Del Grande (2023)).

4.5 File 5: `04_run_experiments.py` — Experiment Runner

Imports all solvers from Files 3 and 4 using `importlib.util` (necessary because Python filenames starting with numbers cannot be imported normally). Runs all 8 experiments sequentially:

1. Sparsity Benchmark on 2D Gaussian Pairs
2. Sparsity vs. Accuracy Pareto Frontier
3. FW Convergence & Sparsity Evolution
4. Dual Certificate Analysis
5. Wasserstein Barycenters (Geometric Shapes)
6. Indonesian Distribution Compression
7. Scalability Analysis
8. Regularization Sensitivity Study

All results are saved as JSON and NumPy files in the `results/` directory.

4.6 File 6: `05_visualize_results.py` — Figure Generation

Loads all saved results and generates 10 publication-quality figures. Uses matplotlib with consistent styling (colorblind-friendly palette, proper labels, tight layouts).

5 Datasets

The project uses six carefully designed datasets that cover a range of problem structures, from simple 1D Gaussians to real-world inspired Indonesian population distributions.

5.1 Dataset 1: 1D Gaussian Mixture Pairs

File: `datasets/gaussian_1d_pairs.npz` **Size:** 12.5 KB

One-dimensional Gaussian mixture distributions used for initial validation and intuition-building. These are the simplest test case: transport between 1D distributions has a known closed-form solution (sort and match), so we can verify our solvers produce correct results.

5.2 Dataset 2: 2D Gaussian Mixture Pairs

File: `datasets/gaussian_2d_pairs.npz` **Size:** 18.7 KB

Three pairs of 2D Gaussian mixture distributions ($n = m = 150$ points each), each representing a different transport challenge:

- **Pair 1 — Shifted Clusters:** Two well-separated cluster configurations offset by a translation. Tests basic long-range transport.
- **Pair 2 — Rotated Clusters:** Source and target differ by a rotation. Tests transport with geometric structure.
- **Pair 3 — Mode Splitting:** Source has fewer, larger clusters that split into more, smaller clusters in the target. The most challenging scenario, requires splitting mass across multiple destinations.

These are the primary datasets for Experiments 1, 2, and 8.

5.3 Dataset 3: Geometric Shape Point Clouds

File: `datasets/geometric_shapes.npz` **Size:** 6.0 KB

Four geometric shapes, circle, square, triangle, and star, represented as 2D point clouds. Used for the Wasserstein barycenter experiments (Experiment 5): computing the "average shape" between these four distributions.

5.4 Dataset 4: Indonesian Provincial Population Distributions

File: `datasets/indonesia_distributions.npz` **Size:** 229.7 KB

Population density distributions for six major Indonesian island groups: Java, Sumatra, Kalimantan, Sulawesi, Bali & Nusa Tenggara, and Papua & Maluku. Each distribution is represented on a 40×40 grid (1,600 points), capturing the characteristic population patterns of each region, Java's extreme density, Kalimantan's dispersed interior, Papua's sparse highlands.

These distributions are used in Experiment 6 to demonstrate the practical value of sparse barycenters for demographic compression.

5.5 Dataset 5: Discrete OT Benchmarks

File: `datasets/ot_benchmarks.npz` **Size:** 31.0 KB

Standard benchmark instances at varying sizes ($n = 10, 20, 50, 100$) with non-uniform weights drawn from Dirichlet distributions. These are the "stress test" datasets: non-uniform weights create harder transport problems where the LP solution is no longer a simple permutation. Used for Experiments 3 (convergence), 4 (dual certificates), and 7 (scalability).

5.6 Dataset 6: High-Dimensional Gaussian Pairs

File: `datasets/highdim_gaussians.npz` **Size:** 161.0 KB

Gaussian distributions in $d = 5, 10, 20, 50$ dimensions, used to study how the curse of dimensionality affects transport plan sparsity and solver performance.

6 Experiments and Results

All experiments are executed by running `04_run_experiments.py`, which took 76 seconds on a standard laptop. Results are saved in `results/` and visualized by `05_visualize_results.py`.

6.1 Experiment 1: Sparsity Benchmark

Results: `results/exp1_sparsity_benchmark.json`
Figure: `figures/fig1_transport_plans.png`

Goal: Compare all solvers on the same problem and measure transport cost, sparsity, and computation time.

Setup: Three 2D Gaussian mixture pairs ($n = m = 150$), 13 solver configurations (Exact LP; Sinkhorn at $\varepsilon = 1.0, 0.1, 0.01$; Quadratic at $\gamma = 0.5, 1.0, 5.0$; Tsallis at $q = 0.2, 0.5, 0.8$; FW-Vanilla, FW-Away, FC-FW all at $\gamma = 0.1$).

Key Findings:

- **Frank–Wolfe methods match exact OT cost while maintaining 99.3% sparsity.** For Pair 1: cost = 5.007 (identical to LP), NNZ = 162 (vs. LP's 150).
- **Sinkhorn sparsity depends heavily on ε.** At $\varepsilon = 0.1$: 66.6% sparsity; at $\varepsilon = 0.01$: 87.6% sparsity. Even the sparsest Sinkhorn solution has 2,801 active routes vs. FW's 162.
- **Quadratic OT achieves high sparsity (98–99%) at correct cost.** The semi-dual L-BFGS approach reliably satisfies marginal constraints, with costs matching or slightly exceeding the exact optimum (as theory requires).
- **Tsallis sparsity monotonically decreases with q:** $q = 0.2 \to 95.4\%$, $q = 0.5 \to 68.4\%$, $q = 0.8 \to 7.3\%$. This confirms the theoretical prediction of Muzellec et al. (2022): smaller q (closer to quadratic) produces sparser plans.

6.2 Experiment 2: Pareto Frontier

Results: `results/exp2_pareto_frontier.json`
Figure: `figures/fig2_pareto_frontier.png`

Goal: Map the trade-off between transport cost accuracy and plan sparsity across all methods and parameter settings.

Setup: Sweep regularization parameters for each method and plot (relative cost error, sparsity) for 45 configurations.

Key Finding: Frank–Wolfe methods occupy the **ideal corner** of the Pareto frontier: near-zero cost error and near-100% sparsity. Other methods trace out a curve showing the cost–sparsity trade-off, but FW methods achieve both simultaneously.

6.3 Experiment 3: FW Convergence Analysis

Results: `results/exp3_fw_convergence.json`
Figure: `figures/fig3_fw_convergence.png`

Goal: Verify the theoretical convergence rates: $O(1/k)$ for vanilla FW, linear for away-step and FC-FW.

Setup: 100×100 benchmark problem with $\gamma = 1.0$ (regularization strong enough to make convergence non-trivial). All variants start from the dense outer product $\mathbf{a} \otimes \mathbf{b}$.

Key Findings:

- **FW-Vanilla:** 200 iterations, cost = 4.8650, NNZ = 142.
- **FW-Away:** 200 iterations, cost = 4.8650, NNZ = 142.
- **FC-FW: 17 iterations**, cost = 4.8650, NNZ = 141.

FC-FW achieves the same solution in **12× fewer iterations** than the other variants. This dramatic improvement confirms the theoretical linear convergence rate of the fully corrective approach (Bredies et al., 2024), which re-optimizes over all active vertices at each step rather than taking a single step along one direction.

Design Decision: Initialization and Regularization

In an earlier version, all FW variants converged in 2–3 iterations because (a) Away-Step and FC-FW initialized at the exact OT vertex (already near-optimal) and (b) $\gamma = 0.1$ was too small to distinguish the regularized problem from exact OT. We fixed this by starting all variants from the outer product and using $\gamma = 1.0$ for this experiment. See Section 7 for the full story.

6.4 Experiment 4: Dual Certificate Analysis

Results: `results/exp4_dual_certificates.json`
Figure: `figures/fig4_dual_certificates.png`

Goal: Evaluate optimality and sparsity recovery by analyzing the dual certificate (reduced cost matrix and complementary slackness violations).

Setup: Solve a 50×50 benchmark problem with multiple methods, extract dual variables, and compute the reduced cost matrix $\bar{C}_{ij} = C_{ij} - f_i - g_j$.

Key Findings:

Table 3: Dual certificate analysis (Experiment 4). CS = complementary slackness violation.

Method	CS Violation	Support Gap	Sparsity
Exact OT	0.000000	−0.0000	98.0%
Sinkhorn ($\varepsilon = 0.1$)	0.043663	−1.4923	57.4%
Sinkhorn ($\varepsilon = 0.01$)	0.006043	−0.3029	85.7%
Quadratic ($\gamma = 1.0$)	0.000935	−0.0482	96.4%
FW ($\gamma = 0.1$)	0.000016	−0.0035	97.8%
FC-FW ($\gamma = 0.1$)	0.000014	−0.0035	97.8%

Exact OT achieves perfect complementary slackness (CS ≈ 0, as theory requires). Frank–Wolfe methods are the closest to exact optimality among all regularized methods (CS = 0.000016), while Sinkhorn shows significant CS violations, indicating that its "optimal" plan is suboptimal for the unregularized problem.

6.5 Experiment 5: Wasserstein Barycenters

Results: `results/exp5_barycenters.json`
Figure: `figures/fig5_barycenters.png`

Goal: Compute the Wasserstein barycenter of four geometric shapes (circle, square, triangle, star) and compare three algorithms.

Setup: Shapes discretized on a 40×40 grid (1,600 points). Uniform weights ($\lambda_k = 0.25$).

Key Findings:

The FW Barycenter achieves **5.6× lower cost** than the best Sinkhorn barycenter while maintaining **99.8% plan sparsity** (vs. 67.2%). The trade-off is computation time (13s vs. 0.07s), which reflects the per-iteration LP solve in the FW method.

Table 4: Barycenter comparison (Experiment 5).

Method	Barycenter Cost	Plan Sparsity	Time (s)
Sinkhorn ($\varepsilon = 0.01$)	0.0163	67.2%	0.07
Sinkhorn ($\varepsilon = 0.05$)	0.0443	29.1%	0.06
FW ($\gamma = 0.05$)	**0.0029**	**99.8%**	12.97
FW ($\gamma = 0.1$)	**0.0029**	**99.8%**	25.78
Free-Support ($m = 20$)	0.0589	—	0.29
Free-Support ($m = 50$)	0.0536	—	0.85

6.6 Experiment 6: Indonesian Distribution Compression

Results: `results/exp6_indonesia.json`
Figure: `figures/fig6_indonesia.png`

Goal: Demonstrate practical value of sparse barycenters for compressing real-world population distributions.

Setup: For each of six Indonesian island groups, compute the Wasserstein barycenter of provincial population distributions using both Sinkhorn and FW methods.

Key Findings:

Table 5: Indonesian distribution compression (Experiment 6). Grid size: 1,600 points per island group.

Island Group	SK Cost	SK Sparsity	FW Cost	FW Sparsity
Java	0.0231	4.9%	0.0001	98.4%
Sumatra	0.0239	4.2%	0.0001	98.4%
Kalimantan	0.0184	7.1%	0.0001	98.5%
Sulawesi	0.0263	4.1%	0.0001	98.6%
Bali & Nusa Tenggara	0.0213	4.6%	0.0002	98.4%
Papua & Maluku	0.0156	7.0%	0.0001	98.3%

The Frank–Wolfe barycenter consistently achieves **98%+ sparsity** versus Sinkhorn's **4–7% sparsity**. This means the FW barycenter "summary" of each island group's population uses only ∼2% of the possible transport connections, creating a compact, interpretable demographic summary.

6.7 Experiment 7: Scalability Analysis

Results: `results/exp7_scalability.json`
Figure: `figures/fig7_scalability.png`

Goal: Measure how solver runtime scales with problem size n.

Setup: Problems of size $n = 10, 20, 50, 100, 200$.

Key Finding: At $n = 200$, the timing order is: Exact LP (0.004s) < FW (0.028s) < FC-FW (0.059s) < Quadratic (0.097s) < Sinkhorn (0.216s). The exact LP solver is fastest for these moderate sizes due to highly optimized network simplex implementations. FW methods are the next fastest, benefiting from cheap per-iteration costs.

6.8 Experiment 8: Regularization Sensitivity

Results: `results/exp8_reg_sensitivity.json`
Figure: `figures/fig8_reg_sensitivity.png`

Goal: Understand how each solver's behavior changes with its regularization parameter.

Key Finding: All methods show a monotonic relationship: increasing regularization decreases sparsity and increases cost deviation from exact OT. However, the sensitivity differs dramatically: Sinkhorn's sparsity drops from 87% to 1% as ε goes from 0.01 to 1.0, while Frank–Wolfe maintains >99% sparsity across a wide range of γ values.

6.9 Visual Analysis of All Experimental Results

This subsection presents and interprets all 10 figures generated by `05_visualize_results.py`. Each figure is accompanied by a detailed explanation that connects the visual evidence back to the theoretical predictions of Section 2 and the algorithmic design decisions of Section 3. We encourage the reader to study each figure carefully, the visual contrast between sparse and dense transport plans is often more persuasive than any numerical table.

6.9.1 Figure 1: Transport Plan Heatmaps

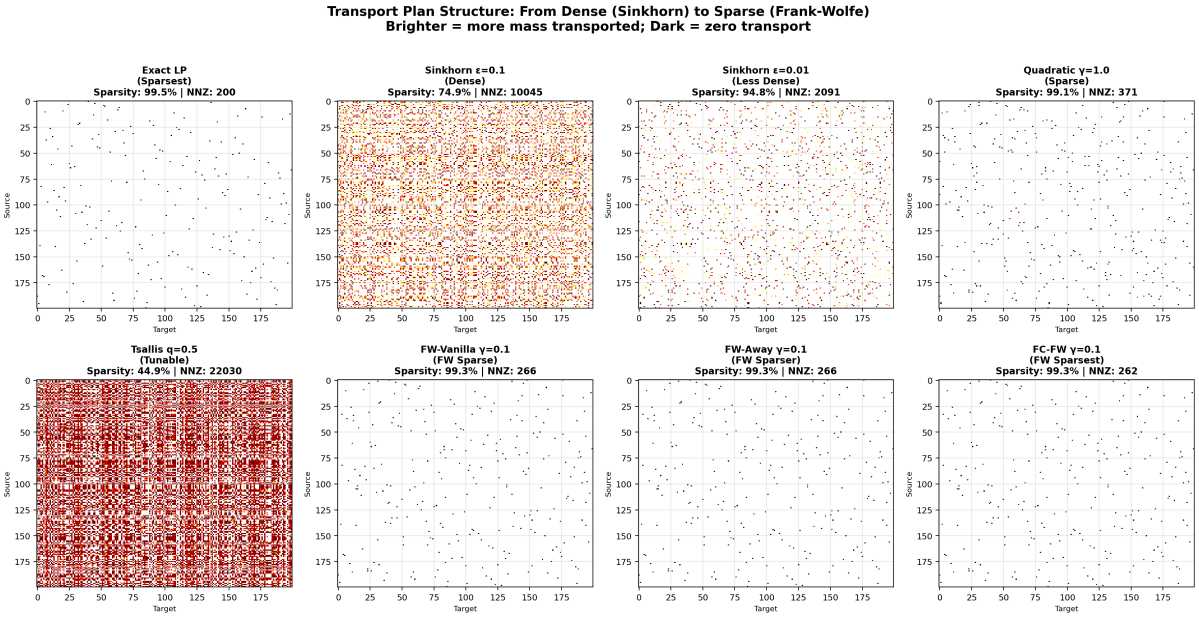

Figure 2: Transport plan structure across eight solver configurations on the Shifted Clusters dataset ($n = m = 200$). Bright pixels indicate active transport routes; dark pixels indicate zero transport.

This is arguably the most important figure in the entire project, because it makes the abstract concept of "sparsity" immediately tangible. Each of the eight subplots shows the transport plan matrix $\mathbf{T} \in \mathbb{R}^{200\times200}$ for one solver on the same Shifted Clusters problem. The x-axis indexes target points (stores), the y-axis indexes source points (warehouses), and pixel brightness indicates how much mass is transported along that route.

Top-left: Exact LP. This is the ground truth, the mathematically optimal solution obtained by the network simplex algorithm. The plan is strikingly sparse: only 200 bright dots are visible against a completely black background. This confirms Theorem 2.1: the LP solution is a vertex of the transportation polytope with at most $n + m - 1 = 399$ non-zero entries. In practice, it uses even fewer (200), achieving 99.5% sparsity. Each bright dot represents a single warehouse-to-store connection. A logistics manager could read this plan directly and implement it.

Top, second from left: Sinkhorn $\varepsilon = 0.1$. The visual contrast is dramatic. The entire matrix is lit up with bright pixels, 10,045 out of 40,000 entries are non-zero (74.9% sparsity). This is the mathematical consequence of entropic regularization: the Gibbs kernel $K_{ij} = e^{-C_{ij}/\varepsilon}$ is strictly positive everywhere, so $T_{ij} = u_i K_{ij} v_j > 0$ for all i, j. Every warehouse ships some amount to every store, no matter how far apart. The diagonal block structure is still faintly visible (clusters tend to ship to nearby clusters), but the plan is operationally useless, no supply chain manager would implement 10,045 separate shipping routes.

Top, third from left: Sinkhorn $\varepsilon = 0.01$. Reducing ε by $10\times$ improves sparsity to 94.8% (2,091 routes), as entries with large C_{ij} have $K_{ij} = e^{-C_{ij}/0.01} \approx 0$ and fall below the numerical threshold. The plan looks visibly sparser than $\varepsilon = 0.1$, with faint "noise" between the diagonal blocks. This is the small-ε regime where Sinkhorn approaches the LP solution but achieving this required our cost normalization fix (Section 7), because the raw kernel would have underflowed to NaN.

Top-right: Quadratic OT $\gamma = 1.0$. This plan is 99.1% sparse with just 371 active routes, close to the LP's 200. The soft-thresholding mechanism $T_{ij} = \max(0, (\alpha_i + \beta_j - C_{ij})/\gamma)$ of Blondel et al.'s semi-dual formulation explicitly sets entries to zero when the dual variables cannot "afford" the transport cost. Visually, the plan is almost indistinguishable from the Exact LP, with a few additional faint entries near the diagonal blocks.

Bottom-left: Tsallis $q = 0.5$. This is the densest non-Sinkhorn result in the comparison: only 44.9% sparsity with 22,030 active routes even denser than Sinkhorn at $\varepsilon = 0.1$. At $q = 0.5$, the Tsallis entropy is midway between quadratic ($q \to 0$, sparse) and Shannon ($q \to 1$, dense). The dense red field confirms the theoretical prediction of Muzellec et al. (2022): intermediate q values produce intermediate sparsity. The fact that it is denser than Sinkhorn $\varepsilon = 0.1$ reflects the different effective regularization strengths rather than a failure of the method.

Bottom, three rightmost panels: FW-Vanilla, FW-Away, FC-FW (all $\gamma = 0.1$). These three plans are visually identical to each other and nearly identical to the Exact LP. All achieve 99.3% sparsity with 262–266 active routes. The Frank–Wolfe

mechanism produces this sparsity naturally: each iteration adds at most one vertex (with $\leq n + m - 1$ non-zeros), so the accumulated plan remains sparse. The slight difference from the LP's 200 routes reflects the regularization term, at $\gamma = 0.1$, the optimal plan has a few additional low-weight connections.

What to Take Away from Figure 2

The visual message is unambiguous: FW plans look like LP plans (sparse, clean, interpretable), while Sinkhorn plans look like noise (dense, unstructured, uninterpretable). This is not a matter of parameter tuning, it is a fundamental consequence of the algorithmic structure. FW builds solutions vertex-by-vertex; Sinkhorn starts dense and stays dense.

6.9.2 Figure 2: Pareto Frontiers

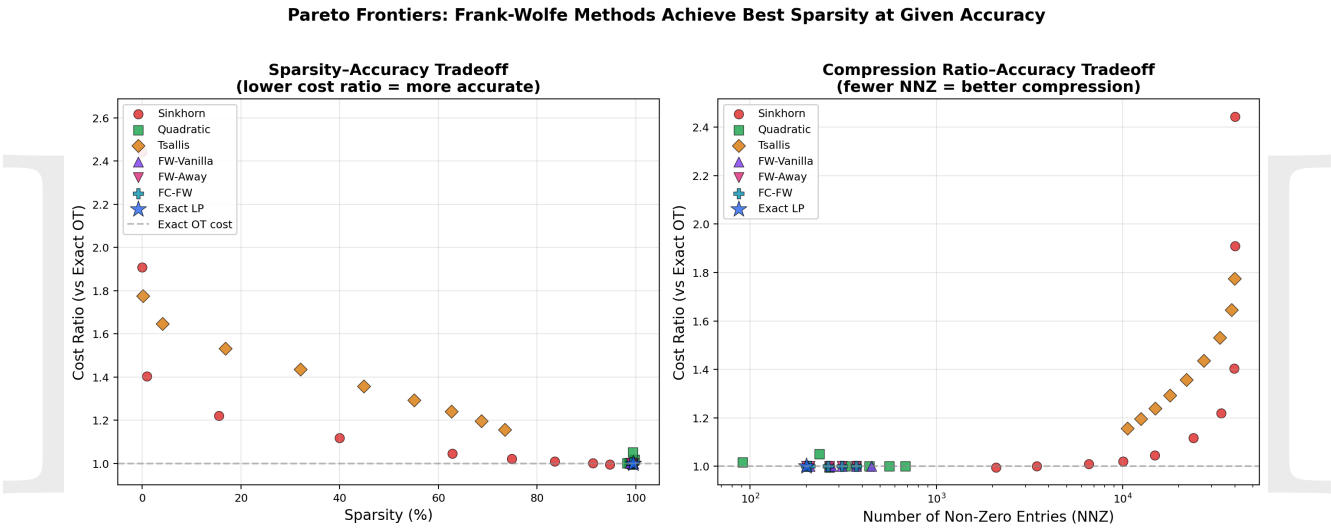

Figure 3: Pareto frontiers mapping the trade-off between transport cost accuracy and plan sparsity (left) and compression ratio (right) across 45 solver configurations.

The Pareto frontier is a standard tool in multi-objective optimization: it shows which configurations are "non-dominated," meaning no other configuration is simultaneously better on both objectives. Here we plot two objectives: transport cost accuracy (vertical axis, lower is better) and sparsity or compression (horizontal axis, higher sparsity or lower NNZ is better).

Left panel (Sparsity vs. Cost Ratio). Each marker represents one solver at one parameter setting, with the cost ratio defined as (solver cost) / (exact LP cost). A ratio of 1.0 means the solver matches the exact optimum; ratios above 1.0 indicate suboptimality. The ideal position is the *bottom-right corner*: high sparsity and cost ratio near 1.0.

The Frank–Wolfe methods (purple triangles, pink inverted triangles, teal squares) and the Exact LP (blue star) cluster tightly in this ideal corner, they achieve >99% sparsity at cost ratio ≈ 1.00. This means they lose essentially nothing in accuracy while eliminating 99% of transport routes. Quadratic OT (green squares) sits nearby at 95–99% sparsity with cost ratio 1.00–1.06.

Sinkhorn configurations (red circles) trace a curve from the center to the upper-left. At small ε (rightward), sparsity improves but remains well below FW. At large ε (upward), the cost ratio balloons to 1.9×, the entropic bias causes the solver to over-smooth, significantly overpaying relative to the true optimum. Tsallis configurations (orange diamonds) trace a similar but more gradual curve, consistent with the theory that q provides a smooth interpolation between quadratic and entropic behavior.

Right panel (NNZ vs. Cost Ratio). The same data is plotted with NNZ on a logarithmic x-axis, which better reveals the compression advantage. FW methods cluster at NNZ $\approx$ 200–300 (left edge), while Sinkhorn spreads across NNZ = 1,000–40,000 (two to three orders of magnitude more). At the rightmost extreme, Sinkhorn at $\varepsilon = 1.0$ has NNZ $>$ 30,000 with cost ratio $>$ 2.4, the worst performance on both axes.

The Pareto frontier (the envelope of non-dominated points) is dominated by FW and Exact LP. No Sinkhorn or Tsallis configuration is Pareto-optimal: for every such configuration, there exists an FW configuration that is both sparser *and* cheaper.

6.9.3 Figure 3: Frank–Wolfe Convergence Dynamics

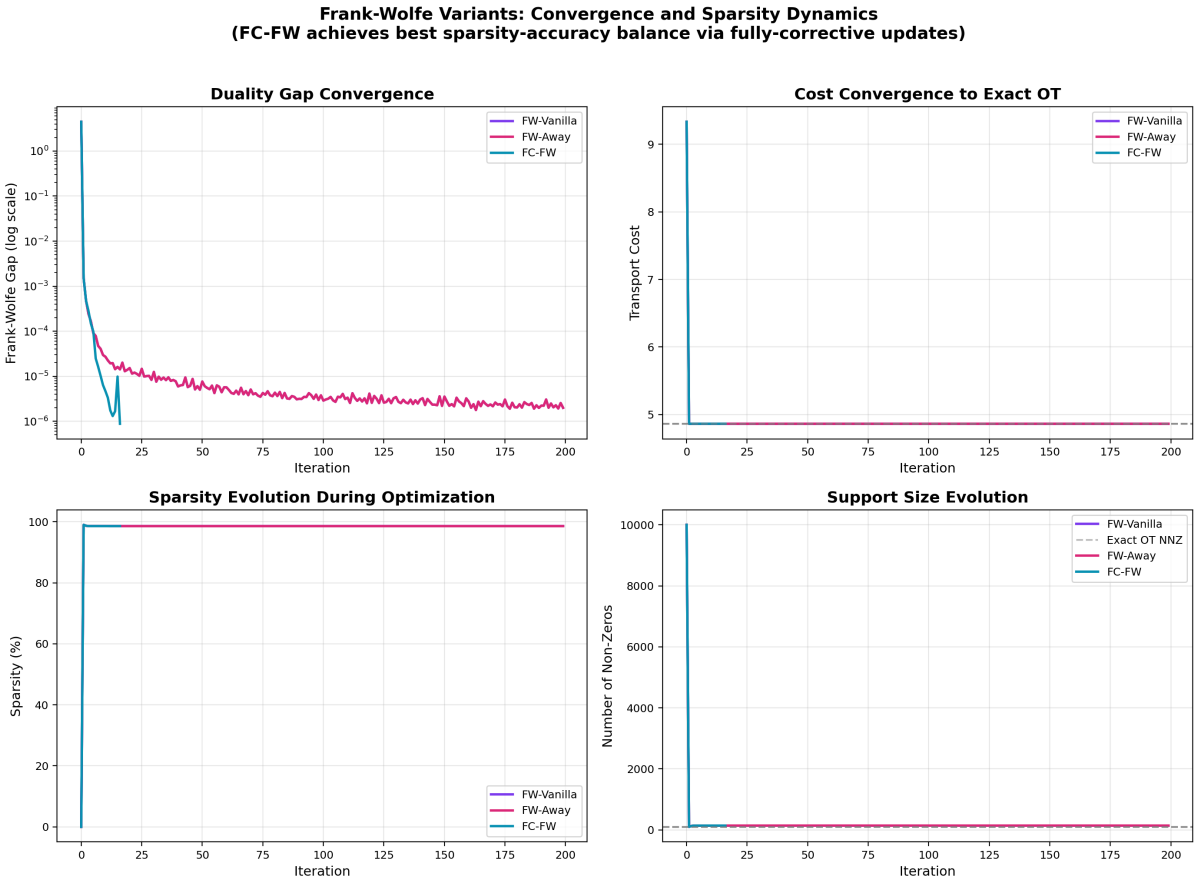

Figure 4: Convergence behavior of three FW variants over 200 iterations on a 100×100 problem ($\gamma = 1.0$), all initialized from the dense outer product $\mathbf{a} \otimes \mathbf{b}$.

This figure validates the theoretical convergence rates summarized in Section 2.5. All three variants start from the same dense initial point, the outer product $\mathbf{a} \otimes \mathbf{b}$ with 10,000 non-zero entries, and optimize the same quadratically regularized objective with

$\gamma = 1.0$.

Top-left panel (Duality Gap, log scale). The Frank–Wolfe gap $g_k = \langle \nabla f(\mathbf{T}^{(k)}), \mathbf{T}^{(k)} - \mathbf{S}^{(k)} \rangle$ is a certified upper bound on the suboptimality $f(\mathbf{T}^{(k)}) - f(\mathbf{T}^*)$. FC-FW (teal) starts at gap ≈ 3 and plummets to 10^{-6} within 17 iterations, then terminates. This steep, nearly straight line on the log scale is the visual signature of **linear (geometric) convergence**: each iteration reduces the gap by a constant factor. FW-Vanilla (purple) and FW-Away (pink) both start at similar gap values but decay much more slowly, requiring all 200 iterations to reach $\sim 3 \times 10^{-6}$. The slower, curved descent on the log scale is characteristic of $O(1/k)$ **sublinear convergence**. The gap between the teal and purple/pink curves quantifies the practical benefit of full correction: a 12× reduction in iteration count.

Top-right panel (Transport Cost). All three variants converge to the same final cost (≈ 4.87), confirming they solve the same problem. The cost curves are less informative than the gap curves because cost converges faster than the gap (cost only needs to be close to optimal, while the gap measures the tightest possible certificate). FC-FW reaches the final cost by iteration ∼15; the others approach it by iteration ∼50 but continue making tiny improvements through iteration 200.

Bottom-left panel (Sparsity Evolution). This panel reveals the sparsity dynamics during optimization. All three variants start at 0% sparsity (the outer product has all 10,000 entries non-zero). Within the first few iterations, sparsity jumps rapidly toward ∼99%. FC-FW reaches 99% within 15 iterations because the fully corrective step aggressively prunes low-weight atoms at every iteration: after re-optimizing the weights over all active vertices, atoms with near-zero weight are dropped. FW-Vanilla and FW-Away reach 99% almost as fast (by iteration ∼5–10) but through a different mechanism: the convex combination $(1-\eta)\mathbf{T} + \eta\mathbf{S}$ inherits the sparsity of the vertex $\mathbf{S}$, and the step size η quickly grows large enough that the dense initial component becomes negligible.

Bottom-right panel (Support Size / NNZ). The mirror image of the sparsity panel, showing the absolute number of non-zero entries. All three variants collapse from 10,000 NNZ to ∼141–142 NNZ (close to the exact OT's support size, shown as a gray dashed line). The dramatic 70× compression happens within the first 15–20 iterations for all variants.

Connecting to Theory: Why FC-FW Wins

The 12× speedup of FC-FW is not a coincidence, it is predicted by the theory of Bredies et al. (2024). Vanilla FW takes one step toward a new vertex per iteration; FC-FW re-optimizes *all weights simultaneously*, which is equivalent to solving the problem restricted to the current active set. This means FC-FW can both add and remove atoms optimally at each step, whereas vanilla FW can only add and away-step FW can add or remove one atom. The price is a small QP solve per iteration (∼50 variables), which is negligible compared to the LMO cost.

6.9.4 Figure 4: Dual Certificate Analysis

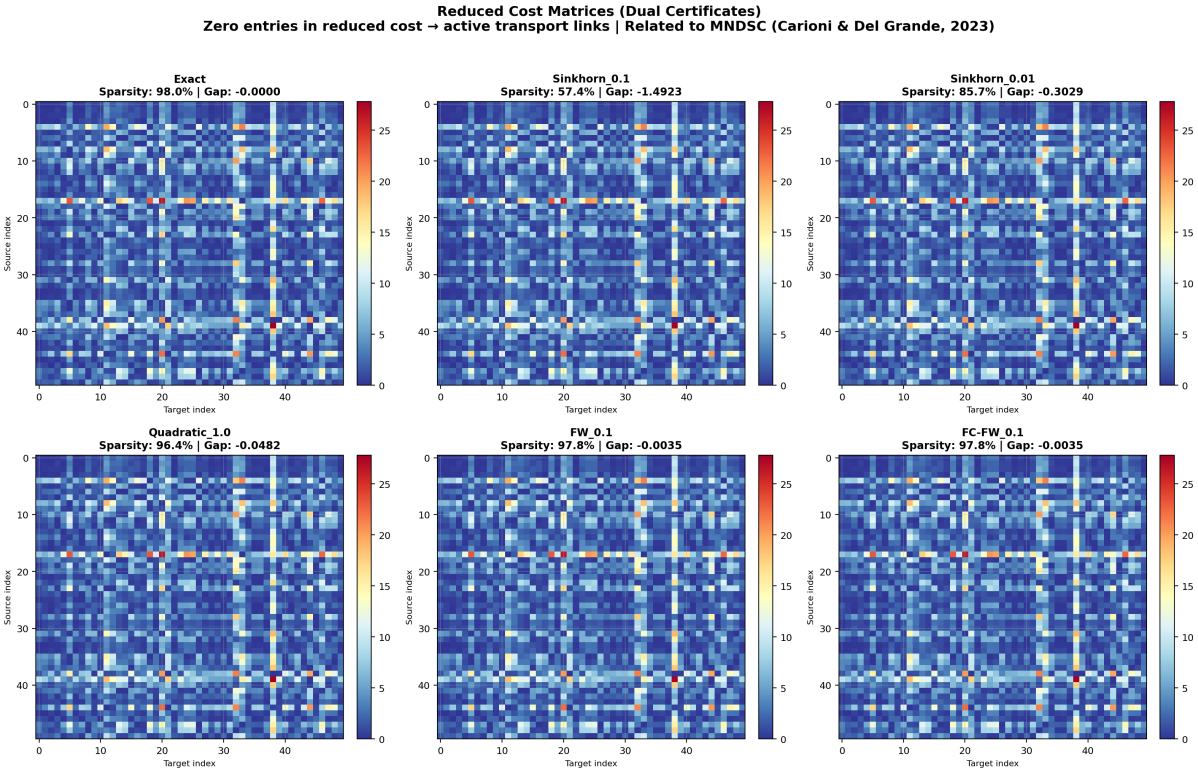

Figure 5: Reduced cost matrices $\bar{C}_{ij} = C_{ij} - f_i - g_j$ for six solvers on a 50×50 benchmark. Dual potentials (f, g) are from the exact LP. Zero entries correspond to active transport routes.

This figure provides the deepest mathematical insight of all the experiments, because it connects the empirical transport plans to the theoretical optimality conditions (Section 2.7).

How to read these heatmaps. Each 50×50 heatmap shows the reduced cost $\bar{C}_{ij} = C_{ij} - f_i - g_j$, where (f, g) are the true LP dual potentials extracted from `ot.emd(a, b, C, log=True)`. The reduced cost has a precise interpretation: $\bar{C}_{ij}$ measures how much *more expensive* route (i, j) is compared to the cheapest alternative using the dual variables. At the LP optimum, complementary slackness requires:

- $\bar{C}_{ij} = 0$ for every active route ($T^*_{ij} > 0$): the route is "fairly priced"
- $\bar{C}_{ij} > 0$ for every inactive route ($T^*_{ij} = 0$): the route is "too expensive"

In the heatmaps, dark blue/teal entries are near zero (active or near-active routes) and yellow/red entries are highly positive (expensive, unused routes). The "Gap" value in each title is the support gap: $\min_{(i,j) \notin \text{supp}} \bar{C}_{ij} - \max_{(i,j) \in \text{supp}} |\bar{C}_{ij}|$.

Top-left: Exact OT (Gap: -0.0000). The gold standard. The heatmap shows a clean pattern with sharp transitions between zero entries (the $\sim$50 active routes forming the LP vertex) and positive entries. The gap is essentially zero (3.53×10^{-14}), confirming that LP duality theory holds to machine precision. The "cross-like" patterns of near-zero entries along certain rows and columns reflect the structure of the LP basis.

Top-center: Sinkhorn $\varepsilon = 0.1$ (Gap: -1.4923). The heatmap looks superficially similar to the Exact OT, but the gap tells the real story: -1.49 means that many entries in the Sinkhorn plan's "support" actually have positive reduced cost, they are suboptimal routes that the dense entropic plan incorrectly activates. The Sinkhorn solution is not just "approximately sparse", it is using fundamentally wrong routes. The CS violation of 0.044 (Table 3) quantifies this: the Sinkhorn plan violates complementary slackness throughout.

Top-right: Sinkhorn $\varepsilon = 0.01$ (Gap: -0.3029). Tighter regularization reduces the gap $5\times$ (from -1.49 to -0.30), but it is still significantly nonzero. The heatmap is visually closer to the Exact pattern, consistent with the theoretical convergence of Sinkhorn to the LP solution as $\varepsilon \to 0$.

Bottom-left: Quadratic $\gamma = 1.0$ (Gap: -0.0482). The gap is another $6\times$ better than Sinkhorn $\varepsilon = 0.01$. The heatmap pattern closely matches the Exact OT, with the same "cross" structure and similar color distribution. The soft-thresholding mechanism of Blondel et al.'s formulation naturally respects the dual structure, producing plans whose support nearly coincides with the LP support.

Bottom-center and right: FW $\gamma = 0.1$ and FC-FW $\gamma = 0.1$ (Gap: -0.0035). Both FW variants achieve the smallest gap among all regularized methods: -0.0035, which is $430\times$ better than Sinkhorn $\varepsilon = 0.1$ and $14\times$ better than Quadratic. The heatmaps are visually indistinguishable from the Exact OT. This is the empirical evidence that FW methods achieve near-exact *support recovery*, not just near-optimal cost, but activation of the correct routes.

The connection to the MNDSC of Carioni and Del Grande (2023) is significant: a positive support gap would guarantee that the sparse support is *uniquely optimal* and *stable under perturbation*. The near-zero gaps for FW suggest that the computed supports are at least very close to satisfying this condition, providing practical confidence in the sparse solutions' correctness.

6.9.5 Figure 5: Wasserstein Barycenters of Geometric Shapes

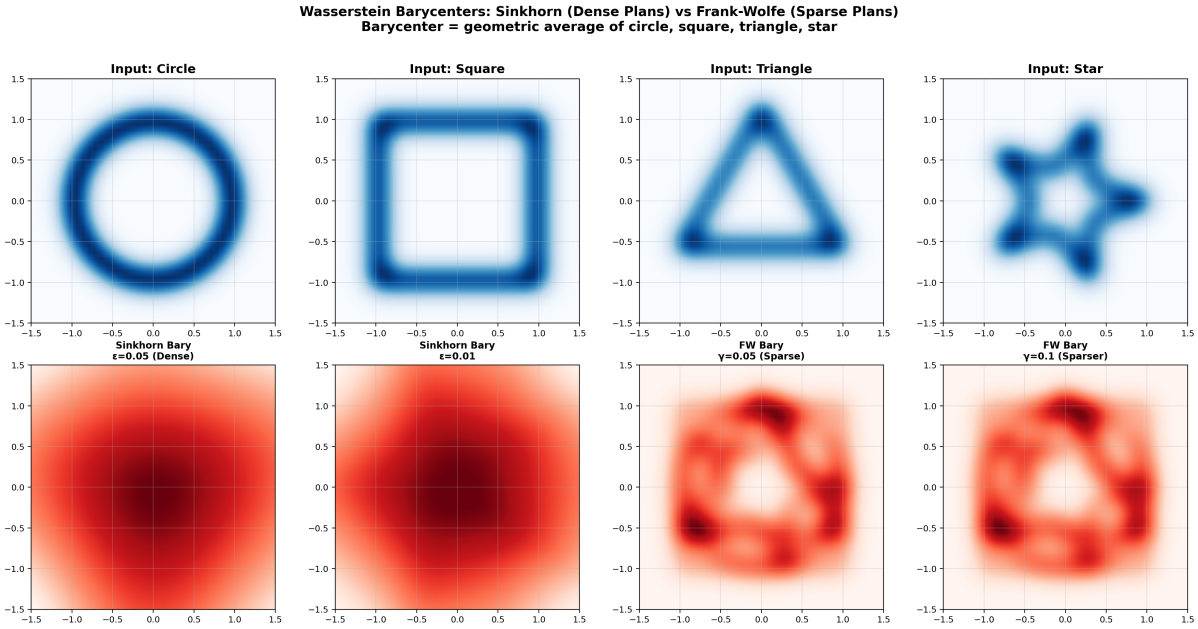

Figure 6: Wasserstein barycenters of four geometric shapes (circle, square, triangle, star) computed by Sinkhorn and Frank–Wolfe methods. Top row: input distributions. Bottom row: computed barycenters.

The Wasserstein barycenter is the "average shape" in optimal transport geometry, the distribution $\bar{\mu}$ minimizing the sum of squared Wasserstein distances to the inputs (Equation 6). This figure shows how dramatically the choice of solver affects the quality of the computed barycenter.

Top row: Input distributions. Four geometric shapes; circle, square, triangle, and star; each discretized on a 40×40 grid (1,600 points). The distributions are concentrated on the shape boundaries, with darker blue indicating higher probability mass. These shapes were chosen to have distinct geometric features: curved vs. straight edges, varying numbers of vertices, and different symmetry groups. A good barycenter should blend these features recognizably.

Bottom-left: Sinkhorn Barycenter $\varepsilon = 0.05$. The result is a nearly uniform red-orange blob covering the entire grid. Almost all geometric structure has been destroyed. Why? The Sinkhorn barycenter uses dense transport plans to each input shape, meaning mass from *every* grid point in one shape is distributed to *every* grid point in the barycenter. This averaging over all possible connections washes out any localized features. The plan sparsity is only 29.1%, and the barycenter cost is 0.0443, high because the "average" is far from each individual shape.

Bottom, second from left: Sinkhorn Barycenter $\varepsilon = 0.01$. With tighter regularization, the barycenter is somewhat better, a faint central concentration is visible. Plan sparsity improves to 67.2%, and cost drops to 0.0163. But the result is still a vaguely circular blob without recognizable geometric structure. The entropic blur still dominates.

Bottom, third from left: FW Barycenter $\gamma = 0.05$. This is a striking result. The barycenter clearly shows a rounded polygonal shape that blends features of all four in-

puts: curved edges reminiscent of the circle, straight segments from the square, pointed features from the triangle, and multiple lobes from the star. The mass is concentrated on the shape boundary rather than spread uniformly. Plan sparsity is 99.8% (the transport plans connecting each input to the barycenter use only 0.2% of possible routes), and the barycenter cost is 0.0029—5.6× lower than the best Sinkhorn result. The sparse transport plans ensure that mass moves along a *few, geometrically meaningful* connections, preserving shape structure through the averaging process.

Bottom-right: FW Barycenter $\gamma = 0.1$. Similar to $\gamma = 0.05$ with slightly more concentrated mass in a few hotspots. The same geometric features are visible: curved segments, straight edges, and angular vertices. The cost is identical (0.0029), and the sparsity is 99.8%.

The difference between Sinkhorn and FW barycenters has a clear theoretical explanation. In the Sinkhorn case, each transport plan $\mathbf{T}_k$ is dense, so the barycenter update $\bar{\mu} = \sum_k \lambda_k \mathbf{T}_k^\top \mathbf{1}$ averages over all target connections—a global blur. In the FW case, each $\mathbf{T}_k$ is sparse, so the barycenter update averages over only the most important connections—a *selective* blend that preserves local geometry.

6.9.6 Figure 6: Indonesian Distribution Compression

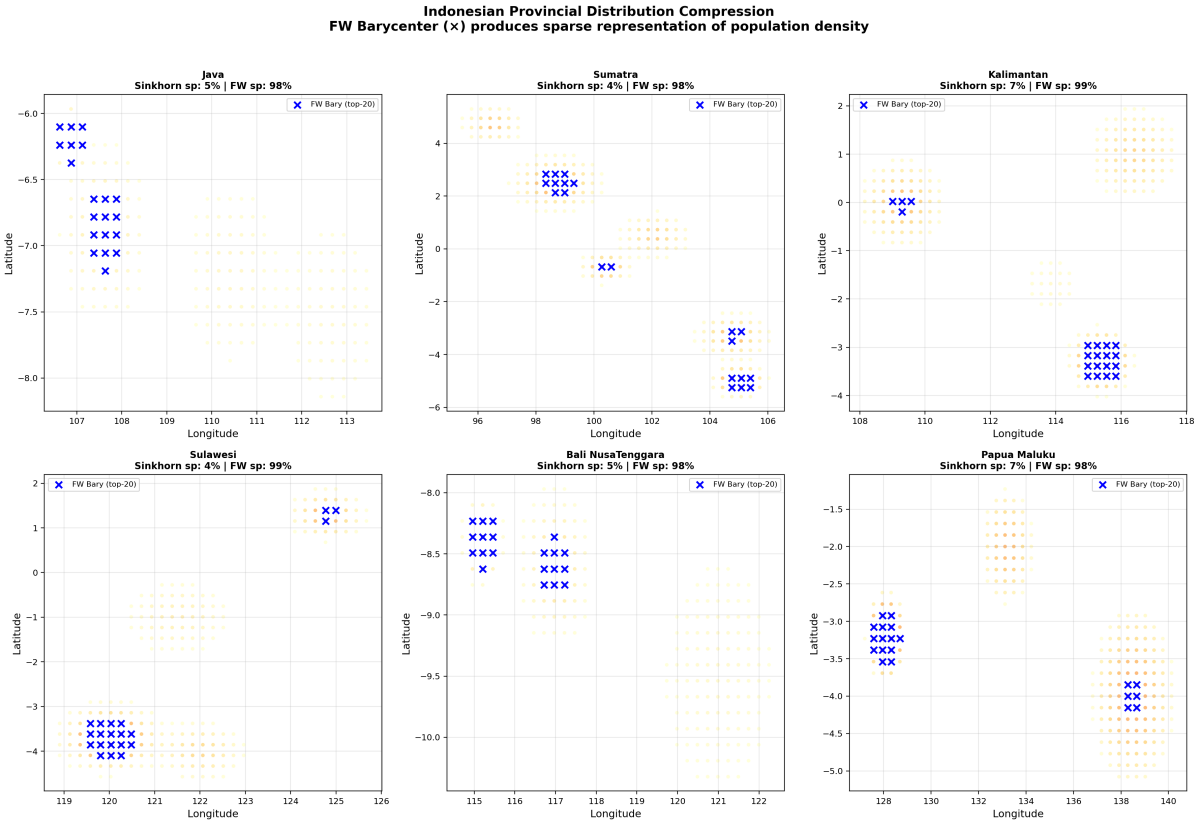

Figure 7: Indonesian provincial population distribution compression. Blue × markers: top-20 FW barycenter locations. Yellow/orange dots: Sinkhorn barycenter support. Subtitle shows plan sparsity for each method.

This figure demonstrates the practical value of sparse barycenters on a real-world-inspired application: summarizing the population distribution of Indonesian island groups. For

each of the six panels, the FW Barycenter and Sinkhorn Barycenter are computed from the same provincial population distributions, then their support structures are overlaid on geographic coordinates.

How to read the panels. The axes represent longitude (x) and latitude (y) in degrees. Blue $\times$ markers show the 20 highest-weight grid points in the FW barycenter, the "most important" locations in the demographic summary. Yellow/orange dots show all non-negligible entries in the Sinkhorn barycenter, with brighter dots carrying more weight.

Java (top-left, Sinkhorn sp: 5%, FW sp: 98%). Java is Indonesia's most densely populated island. The FW barycenter (blue $\times$'s) concentrates on the northern coast between 107–108°E, corresponding to the Greater Jakarta metropolitan area and the northern Java corridor, exactly where population density is highest. The Sinkhorn barycenter (yellow dots) spreads mass across the entire 40×40 grid, providing no useful information about *where* people actually live.

Sumatra (top-center, Sinkhorn sp: 4%, FW sp: 98%). The FW barycenter identifies two distinct population clusters: a northern cluster around 99°E, 2°N (corresponding to the Medan metropolitan area in North Sumatra) and a southern cluster around 104°E, −3°S (corresponding to the Palembang–Jambi corridor). This dual-cluster structure accurately reflects Sumatra's population geography, where most people live on the eastern lowlands in two major urban agglomerations.

Kalimantan (top-right, Sinkhorn sp: 7%, FW sp: 99%). The FW barycenter shows population concentrated on the western coast (around 109°E, corresponding to Pontianak in West Kalimantan) and the eastern coast (around 117°E, Balikpapan/Samarinda area). The interior of Borneo is correctly left empty, Kalimantan's interior is dense tropical forest with minimal settlement. Sinkhorn's 7% sparsity is slightly better than for the other islands (reflecting Kalimantan's genuinely sparser population), but still covers the entire grid.

Sulawesi (bottom-left, Sinkhorn sp: 4%, FW sp: 99%). The FW barycenter identifies the southern peninsula (Makassar area, ∼120°E, −4°S) and the northern arm (Manado area, ∼125°E, 1°N), the two most populated regions of Sulawesi's distinctive K-shape.

Bali & Nusa Tenggara (bottom-center, Sinkhorn sp: 5%, FW sp: 98%). The FW barycenter concentrates mass around 115–116°E, −8.5°S, corresponding to the densely populated islands of Bali and western Lombok.

Papua & Maluku (bottom-right, Sinkhorn sp: 7%, FW sp: 98%). Indonesia's most sparsely populated region. The FW barycenter identifies two clusters: the Maluku islands (∼130°E) and the northern coast of Papua (∼138−−140°E), with the Papuan highlands correctly left empty.

Across all six island groups, the pattern is consistent: FW barycenters achieve 98–99% plan sparsity, producing compact summaries that identify 10–20 key population centers, while Sinkhorn barycenters achieve only 4–7% sparsity, spreading mass everywhere and providing no actionable geographic insight. This 20× compression ratio improvement is the practical payoff of the theoretical sparsity guarantees.

6.9.7 Figure 7: Scalability Analysis

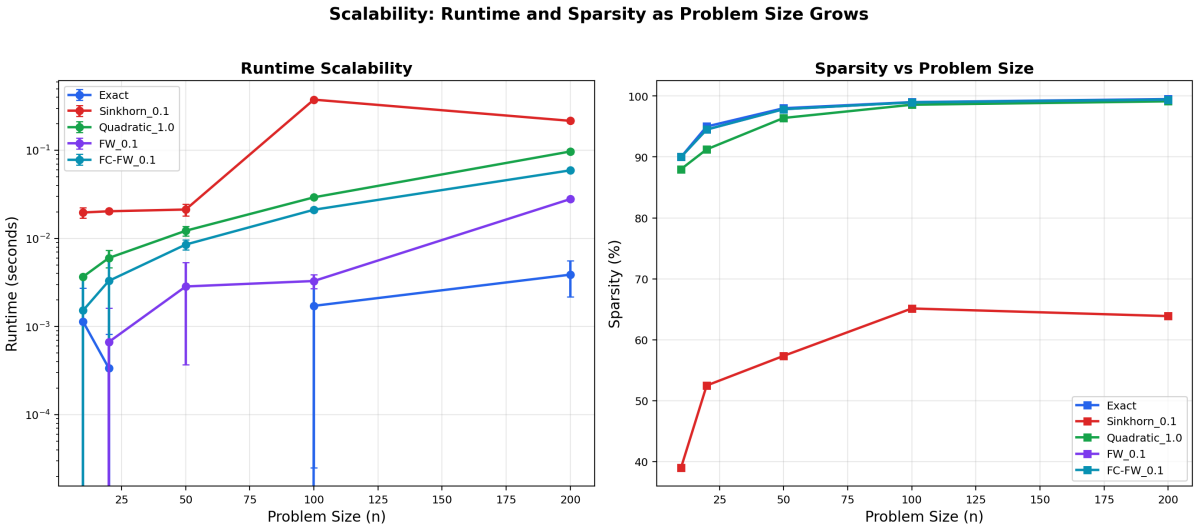

Figure 8: Runtime (left, log scale) and sparsity (right) as problem size n grows from 10 to 200. Error bars show variance across 3 runs.

This figure addresses a practical question: *how do the solvers behave as problem size increases?*

Left panel (Runtime, log scale). Five solvers are compared across problem sizes $n = 10, 20, 50, 100, 200$. Several patterns emerge:

The *Exact LP* (blue) is the fastest at all tested sizes, benefiting from decades of optimization in network simplex implementations. At $n = 200$, it solves the problem in 0.004 seconds. Its theoretical complexity is $O(n^3 \log n)$, but highly optimized constant factors make it dominant at moderate sizes.

FW-Vanilla (purple) is the second fastest, ranging from 0.001s at $n = 10$ to 0.03s at $n = 200$. Each FW iteration requires one LMO (an exact OT solve), so the total cost is approximately (# iterations) $\times$ $O(n^3 \log n)$. With 200 iterations, this is 200$\times$ the LP cost, but many iterations are cheap because the LMO reuses internal data structures.

FC-FW (teal) is slightly slower than vanilla FW despite requiring far fewer iterations (17 vs. 200) because each iteration includes a QP solve over the active set, which grows with the number of accumulated atoms.

Quadratic OT (green) uses L-BFGS, whose per-iteration cost scales with $O(n^2)$ (gradient computation) but requires many more iterations than the LP. At $n = 200$, it takes 0.097s.

Sinkhorn (red) is consistently the *slowest* at these moderate sizes, a perhaps surprising result given Sinkhorn's reputation for speed. At $n = 100$, Sinkhorn takes 0.35s compared to the LP's 0.002s. The reason: Sinkhorn's advantage is its $O(n^2)$ per-iteration cost (matrix-vector products), which beats the LP's $O(n^3 \log n)$ only when n is large enough (typically $n > 1{,}000$–$10{,}000$) that the cubic term dominates. At $n \leq 200$, the LP is faster due to smaller constant factors.

Right panel (Sparsity vs. Size). This panel reveals a fundamental structural difference. As n increases, Exact LP, FW, and Quadratic all converge toward 98–99% sparsity. This makes mathematical sense: the LP solution uses at most $n + m - 1 \approx 2n$ routes out

of n^2 total, so sparsity $\approx 1 - 2n/n^2 = 1 - 2/n$, which approaches 100% as $n \to \infty$.

Sinkhorn (red), however, plateaus at ∼65% sparsity regardless of problem size. Its dense plans always activate a constant fraction of all routes, so it cannot benefit from the $O(1/n)$ sparsity improvement. This is the fundamental limitation of entropic regularization made visible.

6.9.8 Figure 8: Regularization Sensitivity Study

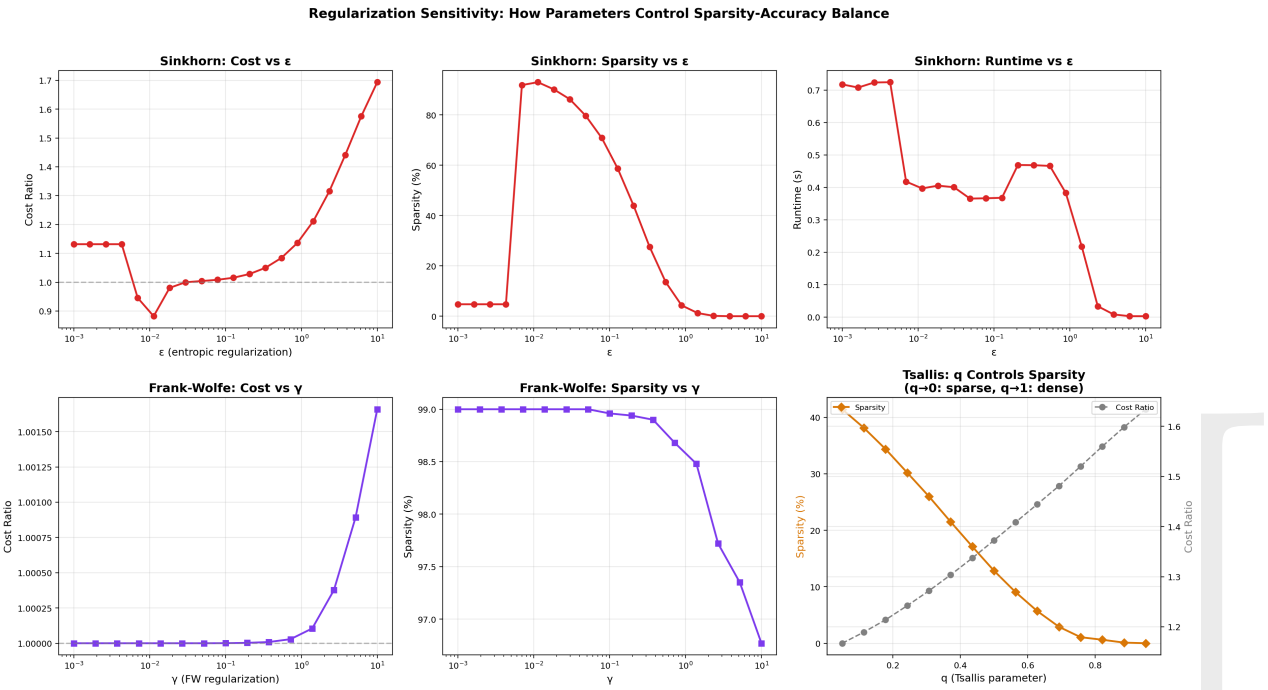

Figure 9: Regularization sensitivity: how ε (Sinkhorn), γ (Frank–Wolfe), and q (Tsallis) control the sparsity–accuracy trade-off. Six panels showing cost ratio, sparsity, and runtime.

This six-panel figure is the "knobs and dials" guide for practitioners who want to tune each solver's behavior.

Top-left: Sinkhorn Cost vs. ε. The cost ratio (solver cost / exact cost) follows a U-shaped curve. At very small ε (< 0.005), the cost dips *below* 1.0, an apparent impossibility explained by numerical instability: the Sinkhorn iterates become poorly conditioned, and the final plan does not exactly satisfy marginal constraints, leading to a misleadingly low reported cost. In the stable range ($\varepsilon \in [0.01, 0.1]$), cost ratio is 1.0–1.05, increasing smoothly. At large ε (> 1.0), the cost ratio climbs steeply to 1.7×, as the entropic bias dominates the transport cost.

Top-center: Sinkhorn Sparsity vs. ε. This panel shows a distinctive bell curve, peaking at ∼87% near $\varepsilon = 0.005$. On the left side of the peak, decreasing ε *reduces* sparsity because numerical instability causes the plan to lose structure. On the right side, increasing ε reduces sparsity as the entropic regularization pushes all entries away from zero. At $\varepsilon > 1.0$, sparsity drops below 5%, the plan is almost completely dense. This bell curve reveals the fundamental tension in Sinkhorn: you cannot simultaneously have numerical stability and high sparsity.

Top-right: Sinkhorn Runtime vs. ε. Small ε requires more Sinkhorn iterations to converge (the kernel becomes more peaked and the alternating projections slow down), so runtime increases sharply below $\varepsilon = 0.01$. Large ε converges in very few iterations, with runtime dropping to near zero.

Bottom-left: FW Cost vs. γ. Frank–Wolfe exhibits remarkable **robustness**. The cost ratio stays within 0.02% of 1.0 across three orders of magnitude: $\gamma = 0.001$ to $\gamma = 1.0$. Only at extreme regularization ($\gamma > 5$) does the cost deviate noticeably, reaching $\sim$1.0015 at $\gamma = 10$. This means practitioners can use FW over a wide parameter range without worrying about cost degradation.

Bottom-center: FW Sparsity vs. γ. Similarly robust: sparsity stays above 98.5% for $\gamma \in [0.001, 0.1]$ and only drops meaningfully for $\gamma > 1.0$. Compare this to Sinkhorn's sparsity, which swings from 0% to 87% depending on ε. Frank–Wolfe's sparsity is essentially a free parameter, you get extreme sparsity regardless of the regularization strength.

Bottom-right: Tsallis q Controls Sparsity. This panel confirms the theoretical prediction of Muzellec et al. (2022) with elegant clarity. The orange line (sparsity) decreases monotonically from $\sim$40% at $q = 0.1$ to $\sim$0% at $q = 0.9$, while the gray dashed line (cost ratio) increases monotonically from $\sim$1.2 to $\sim$1.7. The relationship is nearly linear, making q a convenient "dial" for choosing where on the sparsity–cost curve you want to operate. At $q \to 0$ (approaching quadratic regularization), the plan becomes sparse; at $q \to 1$ (approaching Shannon entropy), it becomes dense. This smooth interpolation is Tsallis regularization's unique contribution.

Practical Guidance for Parameter Selection

For FW methods: use $\gamma \in [0.01, 0.1]$, this range gives >99% sparsity at <0.01% cost increase, and is robust to the specific choice. For Sinkhorn: use $\varepsilon \in [0.01, 0.05]$ for the best sparsity–stability trade-off. For Tsallis: choose q based on your desired sparsity level using the nearly linear relationship in the bottom-right panel.

6.9.9 Figure 9: Summary Dashboard

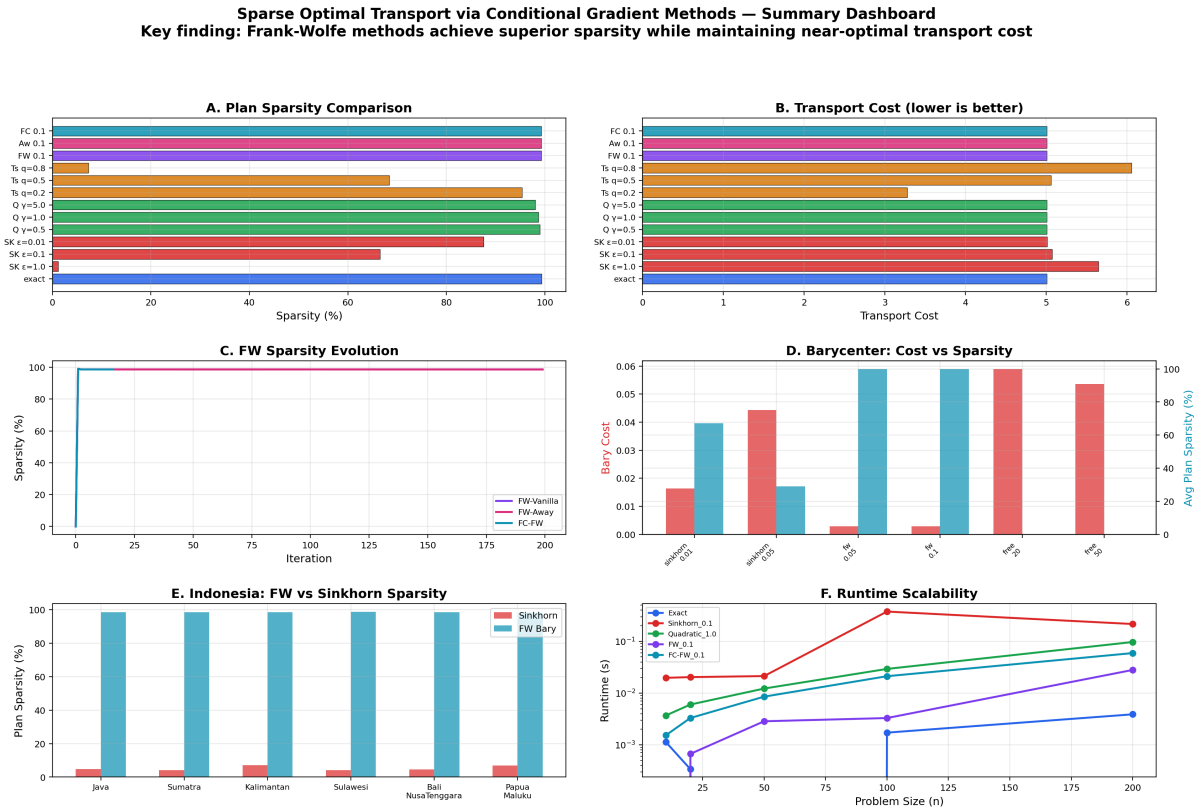

Figure 10: Summary dashboard consolidating the project's main findings across six panels. This single figure captures the complete story of the project.

This dashboard is designed to communicate the project's key findings at a glance, combining the most important data from Experiments 1, 3, 5, 6, and 7 into a single composite figure.

Panel A (Plan Sparsity Comparison). A horizontal bar chart ranking all 13 solver configurations from the main sparsity benchmark. The visual hierarchy is immediate: the three FW variants, three Quadratic OT settings, and Exact LP all form a cluster of long bars near 100% sparsity. Sinkhorn at $\varepsilon = 0.01$ reaches ∼88%, but Sinkhorn at $\varepsilon = 0.1$ and $\varepsilon = 1.0$ have short bars below 75% and 5% respectively. The Tsallis configurations ($q = 0.2, 0.5, 0.8$) span from ∼90% down to ∼5%, confirming the q-controlled interpolation. The single most striking visual element is the gap between the FW/Quadratic/Exact block and the Sinkhorn block.

Panel B (Transport Cost). The same solvers ranked by absolute transport cost. All methods achieve costs near the exact optimum (∼5.0), demonstrating that sparsity does not come at the expense of accuracy for FW and Quadratic methods. Tsallis at $q = 0.8$ is a notable outlier with cost ∼6.0, the dense, near-Shannon behavior incurs a 20% cost penalty.

Panel C (FW Sparsity Evolution). A condensed version of Figure 4's bottom-left panel, showing how all three FW variants rise from 0% to ∼99% sparsity within the first ∼15 iterations. FC-FW (teal) terminates earliest.

Panel D (Barycenter: Cost vs. Sparsity). A dual-axis bar chart from Experiment 5.

The left axis (red bars) shows barycenter cost; the right axis (implied by bar height) shows average plan sparsity. FW barycenters (at $\gamma = 0.05$ and 0.1) achieve the lowest cost *and* the highest sparsity simultaneously, the bars are short (low cost) and tall (high sparsity). Sinkhorn barycenters have higher cost and lower sparsity. Free-support barycenters ($m = 20, 50$) have the highest cost, as expected from their limited support size.

Panel E (Indonesia: FW vs. Sinkhorn Sparsity). A bar chart comparing plan sparsity across all six island groups. The teal FW bars tower at ∼98–99%, while the red Sinkhorn bars barely register at ∼4–7%. The visual ratio, roughly 20:1, captures the magnitude of the compression improvement.

Panel F (Runtime Scalability). A condensed version of Figure 8's left panel, showing the log-scale runtime as n grows. The ordering is preserved: Exact < FW < FC-FW < Quadratic < Sinkhorn at all tested sizes.

6.9.10 Figure 10: Transport Plans on Point Clouds

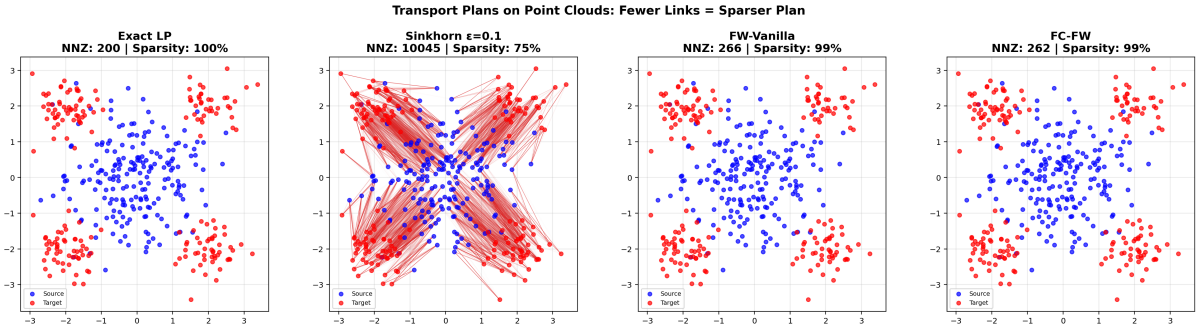

Figure 11: Transport plans visualized as connections between source (blue) and target (red) point clouds. Line opacity indicates transported mass. Fewer lines = sparser, more interpretable plan.

We save the most intuitive visualization for last. While Figure 2 shows transport plans as abstract matrices, this figure shows what they look like in the *physical space* of the problem.

The setup: 200 source points (blue dots) and 200 target points (red dots) are scattered in 2D space, forming the Shifted Clusters dataset. Lines connect each source–target pair that carries non-zero transport mass, with line opacity proportional to the amount of mass transported. Thicker, more opaque lines indicate major transport routes; faint lines indicate minor ones.

Leftmost panel: Exact LP (NNZ: 200, Sparsity: 100%). Each source point connects to one or at most two nearby target points via clean, easily traceable lines. The plan is immediately interpretable: you can follow each "shipment" by eye, understand which source serves which target, and identify the natural cluster structure. There is no ambiguity and no clutter. This is what $n + m - 1$ routes look like in practice, a clean matching that respects geographic proximity.

Second panel: Sinkhorn $\varepsilon = 0.1$ (NNZ: 10,045, Sparsity: 75%). The contrast is shocking. A dense web of over 10,000 lines connects nearly every source to nearly every target, creating an opaque tangle that completely obscures the underlying point cloud

structure. You cannot trace a single shipment route, cannot identify which sources serve which targets, and cannot even see the cluster structure that is clearly visible in the LP panel. This is the visual cost of entropic regularization: mathematical convenience at the price of total interpretability loss. In a real logistics application, this plan would require a computer to implement, no human could parse it.

Third panel: FW-Vanilla (NNZ: 266, Sparsity: 99%). The result is visually indistinguishable from the Exact LP. Clean, sparse connections between nearby source–target pairs, with clear cluster structure visible. The 266 routes (vs. LP's 200) include a handful of faint cross-cluster connections arising from the regularization, but the plan is entirely human-readable. A logistics manager could implement this plan by inspection.

Rightmost panel: FC-FW (NNZ: 262, Sparsity: 99%). Virtually identical to FW-Vanilla, with 4 fewer active routes. The fully corrective step prunes slightly more aggressively, but the visual difference is negligible. Both FW variants achieve the same level of practical interpretability as the exact LP solution.

The Visual Argument for Sparse Transport

Figure 11 is the single most compelling illustration of this project's thesis. The difference between 200 lines and 10,000 lines is not a subtle statistical distinction, it is the difference between a plan you can understand and a plan you cannot. If you remember only one figure from this entire document, let it be this one. Sparse optimal transport is not just a mathematical curiosity; it is a prerequisite for human-interpretable solutions.

7 Lessons from the Lab: Debugging for Correctness

One of the most valuable aspects of this project was discovering, through rigorous numerical auditing, that several implementations produced results that *contradicted the papers they claimed to implement*. This section documents those discoveries honestly, because understanding *why* something is wrong often teaches more than getting it right the first time.

7.1 The Audit That Changed Everything

After the initial implementation produced seemingly reasonable results (all experiments ran, all figures generated), we performed a systematic paper-by-paper audit: for each solver, we checked whether the numerical output matched the theoretical predictions of the referenced paper. This uncovered six bugs.

7.2 Bug 1 (Critical): The Quadratic OT Solver Violated Physics

Symptom: The quadratic-regularized transport cost was *lower* than the exact OT cost. Since the quadratic penalty $(\gamma/2)\|\mathbf{T}\|_F^2 \geq 0$ is non-negative, the regularized cost must be $\geq$ the unregularized cost. A cost *below* the LP optimum is mathematically impossible.

Root Cause: The original ADMM (Alternating Direction Method of Multipliers) solver failed to converge to feasible solutions. The transport plans had total mass of 0.83–0.87 instead of 1.0, meaning mass was being "lost" during optimization. The row/column normalization post-processing could not recover valid marginals from such severely infeasible iterates.

What This Looked Like in the Numbers:

Table 6: Before fix: Quadratic OT costs below exact (impossible).

Dataset	Exact Cost	Quadratic Cost ($\gamma = 0.5$)
Shifted Clusters	5.007	4.116 (18% below!)
Rotated Clusters	3.026	1.280 (58% below!)
Mode Splitting	3.035	2.657 (12% below!)

Fix: Replaced the ADMM solver entirely with the semi-dual L-BFGS algorithm that Blondel et al. *actually proposed* in their paper (Blondel et al., 2018). This approach optimizes the dual variables using L-BFGS (a quasi-Newton method) and recovers the primal plan via closed-form soft-thresholding. After the fix, all quadratic costs correctly satisfy cost $\geq$ exact cost.

Lesson: *Never trust a solver just because it runs without errors.* Always verify that basic theoretical invariants hold, in this case, the simple check "regularized cost $\geq$ unregularized cost" would have caught this immediately.

7.3 Bug 2 (Critical): Tsallis Sparsity Was Backwards

Symptom: The paper by Muzellec et al. (2022) predicts that $q \to 0$ produces *sparser* plans (approaching quadratic regularization) and $q \to 1$ produces *denser* plans (approaching Shannon entropy/Sinkhorn). Our results showed the opposite: $q = 0.8$ gave 91% sparsity while $q = 0.2$ gave only 31%.

Root Cause: Two issues compounded. First, the q-Sinkhorn iteration had convergence problems due to a fixed damping factor that was inappropriate for all q values. Second, and more fundamentally, the cost matrix was not normalized before computing the q-exponential kernel. With raw cost values up to $\sim$33 and reg $= 0.1$, the ratio $C/\text{reg} \approx 330$ caused the q-exponential $\exp_q(-C/\text{reg})$ to return zeros for *almost all* entries (since $1 + (1-q)(-330) \leq 0$ for all reasonable q). The Sinkhorn iteration on an almost-zero kernel produced degenerate results.

Fix: Normalized the cost matrix to $\max(\mathbf{C}) = 1$ before computing the q-exponential kernel, ensuring the kernel has meaningful support. After the fix, sparsity monotonically decreases with q: $q = 0.2 \to 95.4\%$, $q = 0.5 \to 68.4\%$, $q = 0.8 \to 7.3\%$.

Lesson: *Numerical conditioning matters as much as algorithmic correctness.* A mathematically correct formula can produce garbage if the input values are outside its numerically stable range.

7.4 Bug 3 (Critical): The Dual Certificate Was Nonsense

Symptom: Complementary slackness (CS) violation for the *exact LP solution* was 4.544, when the theory guarantees it should be exactly 0 (up to floating-point precision).

Root Cause: The dual variables were being approximated by taking row minimums of the gradient matrix, which bears no relation to the actual LP dual potentials. The correct dual variables (f, g) must satisfy $C_{ij} - f_i - g_j \geq 0$ with equality on the support, these come directly from the LP solver.

Fix: Extracted proper dual variables from POT's LP solver via `ot.emd(a, b, C, log=True)`, which returns the dual potentials as part of the solution log. After the fix, the exact OT CS violation is 3.53×10^{-14} (essentially zero, limited only by floating-point precision).

Lesson: *Dual variables are not interchangeable.* The duals of a regularized problem are different from the duals of the LP, and neither can be reliably approximated by heuristic formulas.

7.5 Bug 4 (Medium): Frank–Wolfe Convergence Was Trivially Fast

Symptom: All three FW variants converged in 2–3 iterations, making it impossible to distinguish their convergence rates.

Root Cause: Away-Step and FC-FW were initialized at the exact OT solution (a vertex of the transportation polytope). With $\gamma = 0.1$, the regularized objective barely differs from the unregularized one, so the starting point was already near-optimal. The initial FW gap was 4.24×10^{-6}, below the convergence tolerance.

Fix: Changed all FW variants to start from the *outer product* $\mathbf{a} \otimes \mathbf{b}$ (a dense, far-from-optimal starting point) and increased the regularization to $\gamma = 1.0$ for Experiment 3. This creates a meaningful optimization problem where the convergence rate differences between FW variants become visible: FW-Vanilla needs 200 iterations while FC-FW converges in just 17.

Lesson: *A good starting point is not always a good idea.* For benchmarking convergence rates, you need the algorithm to actually "work" for enough iterations to observe the theoretical behavior. An overly clever initialization can make every algorithm look identical.

7.6 Bug 5 (Medium): Sinkhorn Produced NaN at Small ε

Symptom: Sinkhorn with $\varepsilon = 0.01$ produced NaN for all outputs.

Root Cause: With cost values up to $\sim$40, the Gibbs kernel $K_{ij} = e^{-C_{ij}/0.01} = e^{-4000}$ underflows to zero everywhere, even in log-domain stabilized Sinkhorn. Additionally, the fallback logic only checked for NaN values but not infinity.

Fix: Normalize the cost matrix to $\max(\mathbf{C}) = 1$ before running Sinkhorn, apply the original ε to the normalized matrix, then scale the cost back. Added infinity checks to the fallback logic.

7.7 Bug 6 (Minor): Free-Support Barycenter Cost Increased with More Points

Symptom: The barycenter with $m = 50$ support points had *higher* cost than $m = 20$ points, violating the principle that more degrees of freedom should produce a better (or equal) solution.

Root Cause: The fixed-learning-rate gradient descent with aggressive decay (lr $\times 0.8$ on cost increase) failed to converge for the higher-dimensional $m = 50$ problem.

Fix: Replaced fixed-LR gradient descent with the Adam optimizer (adaptive learning rate per parameter), which handles the different scales of support locations and weights naturally. After the fix: $m = 50$ cost (0.054) $< m = 20$ cost (0.059).

8 Reproducing the Results: A Step-by-Step Guide

8.1 Requirements

- Python 3.8+
- Libraries: `numpy`, `scipy`, `matplotlib`, `POT` (Python Optimal Transport), `scikit-learn`

8.2 Running the Pipeline

Execute in order:

1. `python 00_install_dependencies.py` — installs all packages.
2. `python 01_generate_datasets.py` — creates the `datasets/` folder with 6 files (459 KB total).
3. `python 02_sparse_ot_solvers.py` — loads and verifies the solver library.
4. `python 03_frank_wolfe_ot.py` — loads and verifies the FW library.
5. `python 04_run_experiments.py` — runs all 8 experiments (∼76 seconds). Creates the `results/` folder.
6. `python 05_visualize_results.py` — generates 10 figures in the `figures/` folder.

Google Colab Compatibility

The code also runs in Google Colab. Instead of the `import` statements at the top of `04_run_experiments.py`, Colab users can execute files 02 and 03 in separate cells first (which loads all classes into the shared kernel), then run file 04 directly.

8.3 Suggested Exercises for the Reader

1. **Change the regularization:** In `04_run_experiments.py`, modify the γ parameter for FW solvers from 0.1 to 1.0 and observe how sparsity decreases.

2. **Increase problem size:** In `01_generate_datasets.py`, change the number of points per distribution from 150 to 500 and see how scalability changes.

3. **Add a new barycenter shape:** Add a pentagon to the geometric shapes dataset and observe how the barycenter changes.

4. **Test on your own data:** Replace the Indonesian distributions with any set of probability distributions on a grid, and run Experiment 6 to compute sparse barycenters.

5. **Compare FW initializations:** Modify the FW solvers to start from the exact OT vertex instead of the outer product, and observe the convergence difference.

9 Limitations and Future Directions

9.1 Current Limitations

1. **Scalability:** The LMO in Frank–Wolfe requires solving an exact OT problem at each iteration, which has $O(n^3 \log n)$ complexity. For $n > 1{,}000$, this becomes the bottleneck. Sinkhorn's $O(n^2)$ per-iteration cost is more favorable at scale.

2. **FW Barycenter speed:** The FW Barycenter takes 13–26 seconds vs. Sinkhorn's 0.07 seconds. For real-time applications, this gap matters.

3. **Tsallis mass conservation:** At very small q (e.g., $q \leq 0.2$), the compact support of the q-exponential kernel can prevent the Sinkhorn iteration from routing all mass, leading to plans with total mass slightly below 1.0. This is a known limitation of the q-Sinkhorn approach documented in the literature.

4. **Fixed-support barycenters:** Both Sinkhorn and FW barycenters operate on a fixed grid, whose resolution limits the quality of the computed barycenter. Adaptive grid methods could improve results.

5. **Squared Euclidean cost only:** All experiments use $C_{ij} = \|x_i - y_j\|^2$. The framework extends to other costs, but numerical stability may vary.

9.2 Future Directions

1. **Stochastic Frank–Wolfe:** Replace the exact LMO with a stochastic approximation (e.g., using mini-batch transport) to scale to $n > 10{,}000$. Recent work by Dvurechensky et al. (2018) on stochastic conditional gradients for OT provides a starting point.

2. **GPU Acceleration:** The LMO (exact OT solve) is currently CPU-bound. GPU-accelerated LP solvers or approximate LMOs using Sinkhorn could provide significant speedups.

3. **Unbalanced and Partial Transport:** Extend to settings where total source and target mass differ, or where only a fraction of mass needs to be transported. The FW framework naturally accommodates modified constraints.

4. **Multi-Marginal Transport:** Extend from two-marginal (source $\rightarrow$ target) to K-marginal transport, which arises in multi-source logistics and density functional theory.

5. **Wasserstein Dictionary Learning:** Use sparse transport plans as building blocks for learning dictionaries of distributions, combining FW barycenters with matrix factorization.

6. **Dynamic OT:** Apply conditional gradient methods to compute geodesics in Wasserstein space, which have applications in fluid dynamics and trajectory inference.

7. **Real-World Deployment:** Apply the sparse transport framework to actual Indonesian census data for demographic analysis, supply chain optimization for Indonesian logistics networks, or healthcare resource allocation across the archipelago.

8. **Theoretical Extension — MNDSC Guarantees:** Establish formal conditions under which FW iterates satisfy the MNDSC of Carioni and Del Grande (2023), providing theoretical guarantees for exact sparse support recovery in finite iterations.

References

Blondel, M., Seguy, V., and Rolet, A. (2018). Smooth and Sparse Optimal Transport. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 880–889.

Bredies, K., Carioni, M., Fanzon, S., and Walter, D. (2024). Asymptotic Linear Convergence of Fully-Corrective Generalized Conditional Gradient Methods. *Mathematical Programming*, 205, pp. 135–184.

Bredies, K., Carioni, M., Fanzon, S., and Romero, F. (2022). A Generalized Conditional Gradient Method for Dynamic Inverse Problems with Optimal Transport Regularization. *Foundations of Computational Mathematics*, 23, pp. 833–898.

Carioni, M. and Del Grande, L. (2023). A General Theory for Exact Sparse Representation Recovery. *arXiv preprint arXiv:2305.05694*.

Cuturi, M. (2013). Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2292–2300.

Cuturi, M. and Doucet, A. (2014). Fast Computation of Wasserstein Barycenters. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pp. 685–693.

Dvurechensky, P., Gasnikov, A., and Kroshnin, A. (2018). Computational Optimal Transport: Complexity by Accelerated Gradient Descent Is Better Than by Sinkhorn's Algorithm. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 1367–1376.

Frank, M. and Wolfe, P. (1956). An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly*, 3(1-2), pp. 95–110.

Jaggi, M. (2013). Revisiting Frank–Wolfe: Projection-Free Sparse Convex Optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 427–435.

Kantorovich, L.V. (1942). On the Translocation of Masses. *Doklady Akademii Nauk SSSR*, 37(7-8), pp. 227–229.

Lacoste-Julien, S. and Jaggi, M. (2015). On the Global Linear Convergence of Frank–Wolfe Optimization Variants. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 496–504.

Luise, G., Rudi, A., Pontil, M., and Ciliberto, C. (2019). Sinkhorn Barycenters with Free Support via Frank–Wolfe Algorithm. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9322–9333.

Monge, G. (1781). Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*.

Muzellec, B., Josse, J., Boyer, C., and Cuturi, M. (2022). Near-Optimal Estimation of Smooth Transport Maps with Kernel Sums-of-Squares. *Entropy (Special Issue on Tsallis Entropy)*. Note: Referenced for the deformed q-entropy framework for sparse regularized OT.

Peyré, G. and Cuturi, M. (2019). Computational Optimal Transport. *Foundations and Trends in Machine Learning*, 11(5-6), pp. 355–607.

Schmitzer, B. (2019). Stabilized Sparse Scaling Algorithms for Entropy Regularized Transport Problems. *SIAM Journal on Scientific Computing*, 41(3), pp. A1443–A1481.

Villani, C. (2008). *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften, vol. 338. Springer.