A Sparse Interior Point Method for Linear Programs arising in Discrete Optimal Transport

Filippo Zanetti* Jacek Gondzio†

Abstract

Discrete Optimal Transport problems give rise to very large linear programs (LP) with a particular structure of the constraint matrix. In this paper we present an interior point method (IPM) specialized for the LP originating from the Kantorovich Optimal Transport problem. Knowing that optimal solutions of such problems display a high degree of sparsity, we propose a column-generation-like technique to force all intermediate iterates to be as sparse as possible. The algorithm is implemented nearly matrix-free. Indeed, most of the computations avoid forming the huge matrices involved and solve the Newton system using only a much smaller Schur complement of the normal equations. We prove theoretical results about the sparsity pattern of the optimal solution, exploiting the graph structure of the underlying problem. We use these results to mix iterative and direct linear solvers efficiently, in a way that avoids producing preconditioners or factorizations with excessive fill-in and at the same time guaranteeing a low number of conjugate gradient iterations. We compare the proposed sparse IPM method with a state-of-the-art solver and show that it can compete with the best network optimization tools in terms of computational time and memory usage. We perform experiments with problems reaching more than a billion variables and demonstrate the robustness of the proposed method.

Keywords: Optimal Transport, Interior Point Method, Conjugate Gradient, Sparse Approximation, Chordal Graphs.

1 Introduction

Interior point methods (IPMs) [35] are among the most efficient optimization solvers for linear programs and are often able to outperform their competitors when it comes to very large problems. They rely on the solution of a linear system to compute the Newton direction that allows to find the next approximation and often employ iterative Krylov methods with a proper preconditioner (e.g. [3, 5, 8, 13, 25, 33]). They are well suited for the solution of problems with some underlying structure that allows for a simplified formulation and efficient preconditioning (e.g. [6, 7, 12, 17]); they have been used in combination with column-generation approaches (see e.g. [16]) to solve problems with many more variables than constraints; they can be formulated in a matrix-free way for

*School of Mathematics, University of Edinburgh, Edinburgh, UK. f.zanetti@sms.ed.ac.uk
†School of Mathematics, University of Edinburgh, Edinburgh, UK. j.gondzio@ed.ac.uk

solving very large problems (e.g. [12, 15]) and they have been used to solve a variety of sparse approximation problems (e.g. [10]).

Linear programs arising from discrete Optimal Transport (OT) possess many of the attractive characteristics (see e.g. [27]): they can have an extremely large number of variables, but relatively few constraints and as a consequence of this, their optimal solution is extremely sparse; they have a very particular constraint matrix with a Kronecker structure which leads to a simplified formulation of the normal equations; they need to be dealt with in a matrix-free way, to avoid forming the huge, very sparse and highly structured constraint matrix.

Linear programs with many more variables than constraints are well suited to be solved using a column-generation approach (see e.g. [23, 31, 32]). In this paper, we propose a hybrid interior point-column generation method for general purpose discrete optimal transport problems: it is highly specialized and exploits the structure of the underlying problem and the known properties of the optimal solution to efficiently mix iterative and direct solvers for the Newton linear system. It is implemented without ever forming the constraint matrix, but only accessing it via matrix-vector products, exploiting its Kronecker structure; the only matrix that is formed is the much smaller Schur complement of the normal equations. The method uses a specialized sparse linear algebra in order to tackle problems of very large dimension.

In particular, the normal equations within the IPM are further reduced to the Schur complement and then solved either with the Conjugate Gradient (CG) method [19] or with a general sparsity-exploiting Cholesky factorization [14]. This is possible because the fill-in of the Cholesky factor of the Schur complement gets smaller and smaller when the IPM approaches optimality; this is rigorously proven using the graph interpretation of the OT problem and confirmed by extensive computational evidence. Moreover, the proposed theoretical result allows to characterize some non-trivial chordal sparsity patterns in a new way.

Many methods have been designed for solving OT problems, see e.g. [2, 9, 11, 18, 22, 24, 28] and the comprehensive summary [29]. A similar idea to the one proposed here, where a sparse version of IPM is used to solve optimal transport problems, was recently proposed in [34], for a particular subset of OT problems. The strengths of the algorithm proposed in this paper, compared to the previously mentioned approaches, are:

- Very general formulation, able to deal with many types of problems; indeed, the sparse IPM is tested on a large and varied collection of problems.
- Adaptability to multiple cost functions, while other methods are often specialized only to one particular metric.
- A highly specialized strategy to solve linear systems, which allows for lower and scalable requirements, both for time and memory.

The proposed IPM implementation is compared with the IBM ILOG Cplex network simplex solver [1, 4, 26], which is a highly optimized commercial software that has been shown to be very fast and reliable when dealing with OT problems. The computational experiments are performed on the DOTmark collection of images [29], considering cost functions given by the $1-$norm, $2-$norm and $\infty-$norm; they show that the proposed method requires less time and less memory than Cplex, while finding a very accurate solution. We performed tests

with extremely large problems, with up to 4.3 billion variables and show that the proposed method is scalable both in terms of computational time and memory requirements.

The rest of the paper is organized as follows: in Section 2 we introduce the discrete optimal transport problem formulation; in Section 3 we present the sparse interior point method in all its key features; in Section 4 we analyze the structure of the normal equations and introduce the mixed iterative-direct approach for their solution; in Section 5 we present the test problems and show the computational results.

1.1 Notation

We denote by $\text{vec}(\cdot)$ the operator that takes a $k_1 \times k_2$ matrix and reshapes it column-wise into a $k_1 k_2$ vector. We use the notation $\lceil x \rceil$ for the ceiling function, i.e. the smallest integer larger than or equal to x.

We define the special modulo function $(x \mod k)$ as

$$(x \mod k) = \begin{cases} (x \mod k) & \text{if } (x \mod k) \neq 0 \\ k & \text{if } (x \mod k) = 0 \end{cases}.$$

We denote by I_k and $\mathbf{e}_k$ respectively the identity matrix and the vector of all ones of size k. We denote by $\mathbb{R}^k_+$ the set of $k-$vectors with nonnegative components.

2 From optimal transport to optimization

Below we recall the Kantorovich formulation [21] of the discrete Optimal Transport problem: given a starting vector $\mathbf{a} \in \mathbb{R}^m_+$ and a final vector $\mathbf{b} \in \mathbb{R}^n_+$, such that $\sum \mathbf{a}_j = \sum \mathbf{b}_j$, find a coupling matrix $\mathcal{P}$ inside the set

$$U(\mathbf{a}, \mathbf{b}) = \left\{ \mathcal{P} \in \mathbb{R}^{m \times n}_+,\ \mathcal{P}\mathbf{e}_n = \mathbf{a},\ \mathcal{P}^T \mathbf{e}_m = \mathbf{b} \right\}$$

that is optimal with respect to a certain cost matrix $\mathcal{C} \in \mathbb{R}^{m \times n}_+$; i.e. find the solution of the following optimization problem

$$\min_{\mathcal{P} \in U(\mathbf{a}, \mathbf{b})} \sum_{i,j} \mathcal{C}_{ij} \mathcal{P}_{ij}. \tag{1}$$

We can interpret this OT problem as minimizing the cost of moving some mass in the configuration $\mathbf{a}$ into the configuration $\mathbf{b}$: $\mathcal{C}_{ij}$ gives the cost of moving a unit of mass from $\mathbf{a}_i$ to $\mathbf{b}_j$ and the optimal solution $\hat{\mathcal{P}}_{ij}$ tell us how much we should move from $\mathbf{a}_i$ to $\mathbf{b}_j$. The constraints given by the set $U(\mathbf{a}, \mathbf{b})$ impose three conditions: we only move positive quantities of mass; we ensure that from each bin i of configuration $\mathbf{a}$, we move out exactly a quantity $\mathbf{a}_i$ overall; we ensure that for each bin j of configuration $\mathbf{b}$, we move in exactly a quantity $\mathbf{b}_j$ overall.

In practice, the "mass" of $\mathbf{a}$ and $\mathbf{b}$ could be anything, from a probability distribution to actual physical quantities that need to be moved. If the cost matrix $\mathcal{C}$ is chosen appropriately (see e.g. [27, Proposition 2.2]), then the optimal

solution of (1) defines a distance between $\mathbf{a}$ and $\mathbf{b}$, called the *q-Wasserstein* distance:

$$W_q(\mathbf{a}, \mathbf{b}) = \big(\sum_{i,j} \mathcal{C}_{ij} \hat{\mathcal{P}}_{ij} \big)^{1/q}. \tag{2}$$

2.1 Kantorovich linear program

We can rewrite the optimization problem (1) as a standard Linear Program (LP):

$$\begin{aligned} \min_{\mathbf{p} \in \mathbb{R}^{mn}} \quad & \mathbf{c}^T \mathbf{p} \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{e}_n^T \otimes I_m \\ I_n \otimes \mathbf{e}_m^T \end{bmatrix} \mathbf{p} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} = \mathbf{f} \\ & \mathbf{p} \geq 0, \end{aligned} \tag{3}$$

where $\otimes$ denotes the Kronecker product and $\mathbf{c} \in \mathbb{R}^{mn}$ and $\mathbf{p} \in \mathbb{R}^{mn}$ are the vectorized versions of $\mathcal{C}$ and $\mathcal{P}$ respectively, $\mathbf{c} = \text{vec}(\mathcal{C})$ and $\mathbf{p} = \text{vec}(\mathcal{P})$.

The matrix of constraints in (3) has the following structure

$$\begin{bmatrix} 1 & & & & & 1 & & & & & 1 & & & & & \\ & 1 & & & & & 1 & & & & & 1 & & & & \dots \\ & & 1 & & & & & 1 & & & & & 1 & & & \\ & & & 1 & & & & & 1 & & & & & 1 & & \\ & & & & \ddots & & & & & \ddots & & & & & \ddots & \\ 1 & 1 & 1 & 1 & \dots & & & & & & & & & & & \\ & & & & & 1 & 1 & 1 & 1 & \dots & & & & & & \dots \\ & & & & & & & & & & 1 & 1 & 1 & 1 & \dots & \\ & & & & & & & & \vdots & & & & & & & \end{bmatrix}.$$

If we call it

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} = \begin{bmatrix} \mathbf{e}_n^T \otimes I_m \\ I_n \otimes \mathbf{e}_m^T \end{bmatrix}, \tag{4}$$

then A_1 is an operator that computes the sum of the entries of the rows of $\mathcal{P}$, while A_2 computes the sum of the entries of the columns. Notice that matrix A is rank deficient by 1.

These operators can be applied in a matrix-free way. Recall the following properties of the Kronecker product [20]:

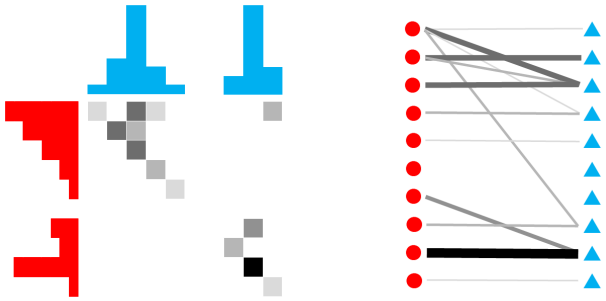
$$(Q \otimes R)^T = Q^T \otimes R^T,$$
$$(Q \otimes R)\mathbf{x} = \text{vec}(RXQ^T),$$

where $\text{vec}(X) = \mathbf{x}$. Then, we can apply matrices A_1, A_2, A_1^T and A_2^T to a vector as follows:

$$\begin{aligned} A_1 \mathbf{x} &= (\mathbf{e}_n^T \otimes I_m)\mathbf{x} = \text{vec}(I_m X \mathbf{e}_n) = X\mathbf{e}_n, \\ A_2 \mathbf{x} &= (I_n \otimes \mathbf{e}_m^T)\mathbf{x} = \text{vec}(\mathbf{e}_m^T X I_n) = X^T \mathbf{e}_m, \\ A_1^T \mathbf{u} &= (\mathbf{e}_n \otimes I_m)\mathbf{u} = \text{vec}(I_m \mathbf{u}\mathbf{e}_n^T) = \text{vec}(\mathbf{u}\mathbf{e}_n^T), \\ A_2^T \mathbf{w} &= (I_n \otimes \mathbf{e}_m)\mathbf{w} = \text{vec}(\mathbf{e}_m \mathbf{w}^T I_n) = \text{vec}(\mathbf{e}_m \mathbf{w}^T), \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^{mn}$, $X \in \mathbb{R}^{m \times n}$, $\mathbf{u} \in \mathbb{R}^m$, $\mathbf{w} \in \mathbb{R}^n$. Notice that the matrices involved X, $\mathbf{u}\mathbf{e}_n^T$ and $\mathbf{e}_m \mathbf{w}^T$ are all of dimension $m \times n$, and thus much smaller than matrix A.

Figure 1: A small example of discrete optimal transport and the corresponding bipartite graph formulation; here the intensity of the color is proportional to the quantity of mass to be moved.



2.2 Graph formulation

We can reformulate the optimal transport problem as a minimum cost flow problem over a bipartite graph: we have m source nodes, each one with an output $\mathbf{a}_i$, connected to n sink nodes, each requiring an input $\mathbf{b}_j$. The incidence matrix of such a graph is very similar to the constraint matrix A:

$$\begin{bmatrix} I_n \otimes \mathbf{e}_m^T \\ -\mathbf{e}_n^T \otimes I_m \end{bmatrix}. \tag{5}$$

A known result of the optimal solution of an optimal transport problem is that the bipartite graph corresponding to the optimal flow, where we have removed the edges corresponding to zero flows, is acyclic (see e.g. [27, Proposition 3.4]). In particular, this implies that there cannot be more than $m + n - 1$ nonzero entries in the optimal solution $\hat{\mathcal{P}}$.

Figure 1 shows a small example of discrete optimal transport and the corresponding bipartite graph formulation: the problem is to move the red configuration (on the left) onto the blue configuration (on the top), where the cost is given by the physical distance between the bins of the histogram.

3 Interior point method for optimal transport

Problem (3) is a standard linear program, that can be solved using an interior point method [35]. However, the constraint matrix can be extremely large for large values of m and n: indeed, the number of variables is $m \cdot n$ and the number of constraint is $m + n$. A linear program with such a structure of the constraint matrix is well suited to be solved by a column generation approach (see e.g. [16, 23, 31, 32]). We propose a method that mixes these two techniques: an interior point method that keeps the iterates sparse at each iteration and updates the list of variables that are allowed to be non-zero in a similar way to the column generation method. We use the term *support* for this list of presumed "basic" variables, to avoid confusion with the notion of *basis* in the simplex and IPM communities.

With this mixed approach, we take advantage of the speed and accuracy of a second order method, while keeping the linear algebra operations cheap thanks to the use of sparse vectors.

3.1 Interior point method

In an oversimplified interpretation (see [35] for more details), an interior point method solves problem (3) by adding a logarithmic barrier to enforce the non-negativity constraint, to obtain the Lagrangian:

$$\mathcal{L}(\mathbf{p}, \mathbf{y}, \mu) = \mathbf{c}^T\mathbf{p} - \mathbf{y}^T(A\mathbf{p} - \mathbf{f}) - \mu \sum_{i=1}^{mn} \mathbf{p}_i,$$

where the parameter μ is slowly driven to zero during the IPM iterations.

At each IPM iteration, one step of the Newton method is performed; the linear system which needs to be solved is

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I_{mn} \\ S & 0 & P \end{bmatrix} \begin{bmatrix} \mathbf{\Delta p} \\ \mathbf{\Delta y} \\ \mathbf{\Delta s} \end{bmatrix} = \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{r_3} \end{bmatrix} = \begin{bmatrix} \mathbf{f} - A\mathbf{p} \\ \mathbf{c} - A^T\mathbf{y} - \mathbf{s} \\ \sigma\mu\mathbf{e}_{mn} - PS\mathbf{e}_{mn} \end{bmatrix}. \tag{6}$$

When reduced to the normal equations approach, it takes the form

$$A\Theta A^T\mathbf{\Delta y} = \mathbf{r_1} + A\Theta(\mathbf{r_2} - P^{-1}\mathbf{r_3}) \tag{7}$$

where $\Theta = \text{diag}(\theta)$ is a diagonal matrix; its entries are $\theta_j = \mathbf{p}_j/\mathbf{s}_j$, $\mathbf{s}$ being the dual slack variable associated with the LP (3), $P = \text{diag}(\mathbf{p})$ and $S = \text{diag}(\mathbf{s})$, σ is a parameter responsible for the reduction of μ.

Notice that the linear system (7) is of dimension $(m + n)$ and thus much smaller than (6), that is of dimension $(m+n+2mn)$. However, (7) contains blocks that would be fully dense, as will be demonstrated in Section 3.3. Moreover, the IPM needs to work with many vectors of size $m \cdot n$, which requires excessive storage for large values of m and n.

3.2 Sparse IPM

It has been observed (see e.g. [15]) that the entries of matrix Θ can be divided in two groups: for "basic" indices $j \in \mathcal{B}$, for which $\mathbf{p}_j \to \hat{\mathbf{p}}_j > 0$, θ_j becomes very large as the IPM approaches convergence; for the "nonbasic" indices $j \in \mathcal{N}$, such that $\mathbf{p}_j \to \hat{\mathbf{p}}_j = 0$, θ_j becomes very small. Moreover, in the non-degenerate case, the optimal solution $\hat{\mathbf{p}}$ is expected to have only $m + n - 1$ non-zeros, out of $m \cdot n$ entries: in the common scenario where $m = n$, this means that the density of the optimal solution decreases as $1/m$ and that at optimality most entries of θ are close to zero, and only a small fraction of them is large.

We can exploit these properties to derive a sparse interior point method, based on the developments of the column generation techniques. In the algorithm that we propose, each iteration is comprised by two phases: an IPM phase on the reduced problem and an update phase.

In the *IPM phase*, we apply a standard IPM to a reduced form of the problem. We start from a vector `index`, that contains the indices of the support, i.e. of the variables that are allowed to attain nonzero values; entries $\mathbf{p}_j$ and

$\mathbf{s}_j$ that are not in the `index` vector are forced to be zero and the same applies to the components of the Newton direction, since we are not updating these components. We define as $\mathbf{p}_{\text{red}}$ and $\mathbf{s}_{\text{red}}$ the reduced $\mathbf{p}$ and $\mathbf{s}$ vectors that contain only the nonzero components. The logarithmic barrier is applied only to the set of variables defined by the index vector and the complementarity measure μ is thus computed as $\mu = (\mathbf{p}_{\text{red}}^T \mathbf{s}_{\text{red}})/\psi$, where $\psi = |\texttt{index}|$. If we call A_{red} the submatrix of A obtained considering only the columns in `index`, then the linear system to solve becomes

$$\begin{bmatrix} A_{\text{red}} & 0 & 0 \\ 0 & A_{\text{red}}^T & I_\psi \\ S_{\text{red}} & 0 & P_{\text{red}} \end{bmatrix} \begin{bmatrix} \mathbf{\Delta p}_{\text{red}} \\ \mathbf{\Delta y} \\ \mathbf{\Delta s}_{\text{red}} \end{bmatrix} = \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2}_{\text{red}} \\ \mathbf{r_3}_{\text{red}} \end{bmatrix} = \begin{bmatrix} \mathbf{f} - A_{\text{red}} \mathbf{p}_{\text{red}} \\ \mathbf{c}_{\text{red}} - A_{\text{red}}^T \mathbf{y} - \mathbf{s}_{\text{red}} \\ \sigma\mu \mathbf{e}_\psi - P_{\text{red}} S_{\text{red}} \mathbf{e}_\psi \end{bmatrix}. \quad (8)$$

The normal equations matrix takes the same form as in (7); indeed $A_{\text{red}} \Theta_{\text{red}} A_{\text{red}}^T = A\Theta A^T$, because Θ coincides with Θ_{red} on the indices in `index` and is zero elsewhere. However, the matrix is much sparser than before.

In the *update phase*, we update the vector `index` and expand or reduce the support; this can happen in two ways:

- We add variables to the support according to their reduced cost $\mathbf{c} - A^T\mathbf{y}$; if any of them is negative, the variables with the most negative reduced cost are added to the support. The new entries are initialized with $(\mathbf{p}_{\text{red}})_j = (\mathbf{s}_{\text{red}})_j = \sqrt{\mu}$, in order to maintain the overall complementarity measure unaltered.

- We remove variables based on the value of $\mathbf{p}_{\text{red}}$; if $(\mathbf{p}_{\text{red}})_j$ is smaller than a certain threshold when the IPM is near convergence, we assume that this component should not belong to the support and remove it.

In order to not perturb too much the IPM algorithm, only m variables are allowed to enter or leave the support at every iteration. In practice, in the first iterations of IPM many variables enter the support, while no entry of $\mathbf{p}$ is small enough to leave it; in the late phase of IPM instead many variables leave the support, while almost no index is added to it.

The initial vector `index` can be chosen in many ways according to heuristics that try to capture the sparsity pattern of the optimal solution. A simple approach is to include in the initial support the entries that correspond to a small cost $\mathbf{c}_j$, according to a predetermined threshold, since the optimal solution will try to allocate as much mass as possible in these low-cost variables. We know that the optimal support should include only $n + m - 1$ entries, but at the beginning we choose the vector `index` with a larger number of entries, usually between three and ten times more than the optimal solution.

3.2.1 Pricing method

In order to update the support, we need to compute the full reduced cost vector $\mathbf{c} - A^T\mathbf{y}$; if m and n are large, this can be very expensive. To reduce the cost of this operation, we propose a heuristic pricing approach. Notice the following

$$(\mathbf{c} - A^T\mathbf{y})_j = \mathbf{c}_j - \mathbf{y}^T\mathbf{a}_j = \mathbf{c}_j - \mathbf{y}_{k_{1j}} - \mathbf{y}_{k_{2j}},$$

where $\mathbf{a}_j$ is the $j-$th column of A and k_{1j} and k_{2j} are defined as

$$k_{1j} = (j \mod m), \qquad k_{2j} = \left\lceil \frac{j}{m} \right\rceil.$$

We make the assumption that a large portion of the smallest (most negative) reduced costs correspond to small values of $\mathbf{c}_j$; the accuracy of this statement depends on the values attained by the Lagrange multiplier $\mathbf{y}$, but we observed in practice that it is true in most of the iterations. Therefore, before starting the IPM algorithm, we can compute a subset of indices J, such that if $j \in J$, $\mathbf{c}_j < C_{\max}$, for some fixed value $C_{\max}$, and the corresponding indices k_{1j} and k_{2j}. Then, during the update phase, we can quickly compute the subset of the entries of the reduced cost vector corresponding to the indices in J and use only these to update the support. Of course, this method misses some of the variables to be added; thus, after a certain number of IPM iterations where we use this method, we should compute the full vector of reduced costs. In this way, we keep the low cost of a single IPM iteration, but we likely increase the number of iterations required. The number of consecutive iterations in which we use the heuristic needs to be chosen carefully: we would like it to be large, to reduce the computational cost as much as possible, but if it is too large we risk performing useless iterations, since the next update with the full reduced costs might change drastically the support. Numerical evidence suggests that the reduced costs should be refreshed every 3 or 4 IPM iterations.

Algorithm SparseIPM summarizes the method just described.

Algorithm SparseIPM

Input: m, n, $\mathbf{f} = [\mathbf{a}^T \quad \mathbf{b}^T]^T$, $\mathbf{c}$, `index`
Initialize $\mathbf{p}^0_{\text{red}}$, $\mathbf{y}^0$, $\mathbf{s}^0_{\text{red}}$

while $\max\left(\frac{\|\mathbf{r_1}\|}{1+\|\mathbf{f}\|}, \frac{\|\mathbf{r_{2\,red}}\|}{1+\|\mathbf{c}_{\text{red}}\|}, \mu\right) >$ `tol` **do**
  $\mathbf{r_1} = \mathbf{f} - A_{\text{red}}\mathbf{p}_{\text{red}}$
  $\mathbf{r_{2\,red}} = \mathbf{c}_{\text{red}} - A^T_{\text{red}}\mathbf{y} - \mathbf{s}_{\text{red}}$
  $\mu = (\mathbf{p}^T_{\text{red}}\mathbf{s}_{\text{red}})/\psi$
  Choose σ and compute $\mathbf{r_{3\,red}} = \sigma\mu\mathbf{e}_\psi - P_{\text{red}}S_{\text{red}}\mathbf{e}_\psi$
  Compute Newton direction $(\mathbf{\Delta p}_{\text{red}}, \mathbf{\Delta y}, \mathbf{\Delta s}_{\text{red}})$ by solving (8)
  Compute stepsizes α_p and α_s
  $\mathbf{p}_{\text{red}} \leftarrow \mathbf{p}_{\text{red}} + \alpha_p\mathbf{\Delta p}_{\text{red}}$
  $\mathbf{y} \leftarrow \mathbf{y} + \alpha_s\mathbf{\Delta y}$
  $\mathbf{s}_{\text{red}} \leftarrow \mathbf{s}_{\text{red}} + \alpha_s\mathbf{\Delta s}_{\text{red}}$
  Perform pricing, with heuristic or full reduced costs
  Update `index`, $\mathbf{p}_{\text{red}}$, $\mathbf{s}_{\text{red}}$
end while

3.3 Structure of the normal equations matrix

The normal equations matrix in (7) takes the form

$$A\Theta A^T = \begin{bmatrix} M & V \\ V^T & N \end{bmatrix} \tag{9}$$

where $V \in \mathbb{R}^{m\times n}$, $\mathrm{vec}(V) = \theta$, $M \in \mathbb{R}^{m\times m}$ and $N \in \mathbb{R}^{n\times n}$ are diagonal and

$$M_{ii} = \sum_{t=0}^{n-1} \theta_{i+tm},\ i = 1, \ldots, m, \qquad N_{jj} = \sum_{t=1}^{m} \theta_{(j-1)m+t},\ j = 1, \ldots, n.$$

To see this, notice that

$$A\Theta A^T = \begin{bmatrix} A_1\Theta A_1^T & A_1\Theta A_2^T \\ A_2\Theta A_1^T & A_2\Theta A_2^T \end{bmatrix}.$$

Let us compute the $(1,1)$ term.

$$(A_1\Theta A_1^T)_{ij} = \sum_{k=1}^{mn} (A_1)_{ik}(A_1)_{jk}\theta_k.$$

Given the structure of A_1 in (4), the only nonzero terms in the summation are obtained if $i = j = (k \mod m)$; in this case $(A_1)_{ik} = (A_1)_{jk} = 1$ and we obtain

$$(A_1\Theta A_1^T)_{ij} = \delta_{ij} \sum_{t=0}^{n-1} \theta_{j+tm},$$

where δ_{ij} is the Kronecker delta. Similarly, we can compute the $(2,2)$ term

$$(A_2\Theta A_2^T)_{ij} = \sum_{k=1}^{mn} (A_2)_{ik}(A_2)_{jk}\theta_k.$$

In this case, the nonzero terms are given by $i = j = \lceil k/m \rceil$. Therefore

$$(A_2\Theta A_2^T)_{ij} = \delta_{ij} \sum_{t=1}^{m} \theta_{(j-1)m+t}.$$

Let us now compute the off-diagonal terms

$$(A_1\Theta A_2^T)_{ij} = \sum_{k=1}^{mn} (A_1)_{ik}(A_2)_{jk}\theta_k.$$

For any combination of i and j, there is only one value of k that produces a nonzero coefficient in the summation, so

$$(A_1\Theta A_2^T)_{ij} = \theta_{k(i,j)},$$

where $k(i,j)$ is identified by $i = (k \mod m)$ and $j = \lceil k/m \rceil$. With this in mind, it should be clear that

$$\mathrm{vec}(A_1\Theta A_2^T) = \theta.$$

As an example, if $m = 2$ and $n = 3$, the normal equations matrix would take the following form

$$\begin{bmatrix} \theta_1+\theta_3+\theta_5 & 0 & \theta_1 & \theta_3 & \theta_5 \\ 0 & \theta_2+\theta_4+\theta_6 & \theta_2 & \theta_4 & \theta_6 \\ \theta_1 & \theta_2 & \theta_1+\theta_2 & 0 & 0 \\ \theta_3 & \theta_4 & 0 & \theta_3+\theta_4 & 0 \\ \theta_5 & \theta_6 & 0 & 0 & \theta_5+\theta_6 \end{bmatrix}.$$

Notice the meaning of matrices M and N:

$$V\mathbf{e}_n = M\mathbf{e}_m, \qquad V^T\mathbf{e}_m = N\mathbf{e}_n. \tag{10}$$

i.e. M_{kk} is the sum of row k of V, while N_{kk} is the sum of column k of V.

In a standard IPM, matrix V would be completely dense, since the entries θ_j are all strictly positive. In the sparse version of IPM instead most entries of θ are exactly zero, making matrix V extremely sparse.

4 Solution of the normal equations

To solve a linear system involving matrix (9), we can use the Schur complement approach: the solution of

$$\begin{bmatrix} M & V \\ V^T & N \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$$

is given by either

$$\alpha_1 = S_N^{-1}(\beta_1 - VM^T\beta_2), \quad \alpha_2 = N^{-1}(\beta_2 - V^T\alpha_1),$$

or

$$\alpha_2 = S_M^{-1}(\beta_2 - V^TM^{-1}\beta_1), \quad \alpha_1 = M^{-1}(\beta_1 - V\alpha_2),$$

where $S_N = M - VN^{-1}V^T$ and $S_M = N - V^TM^{-1}V$ are the two Schur complements. This approach is convenient in this case since matrices M and N are diagonal and thus very easy to invert. Notice that both Schur complements are rank deficient by 1, but this is not a problem in practice: both iterative methods and direct solvers can deal with singular matrices by solving the corresponding least squares problem.

Let us analyze S_M: if we sum the rows of matrix $V^TM^{-1}V$ we obtain, exploiting (10):

$$V^TM^{-1}V\mathbf{e}_n = V^TM^{-1}M\mathbf{e}_m = V^T\mathbf{e}_m = N\mathbf{e}_n.$$

The Schur complement S_M is equal to matrix N minus the previous matrix. Therefore

$$\begin{aligned} |(S_M)_{kk}| - \sum_{i\neq k}|(S_M)_{ki}| &= |N_{kk} - (V^TM^{-1}V)_{kk}| - \sum_{i\neq k}|(V^TM^{-1}V)_{ki}| \\ &= N_{kk} - (V^TM^{-1}V)_{kk} - \sum_{i\neq k}(V^TM^{-1}V)_{ki} \\ &= N_{kk} - \sum_{i=1}^{n}(V^TM^{-1}V)_{ki} \\ &= N_{kk} - (V^TM^{-1}V\mathbf{e}_n)_k = 0. \end{aligned} \tag{11}$$

This means that the Schur complement S_M is weakly diagonally dominant.

If we consider the other Schur complement $S_N = M - VN^{-1}V^T$, then

$$VN^{-1}V^T\mathbf{e}_m = VN^{-1}N\mathbf{e}_n = V\mathbf{e}_n = M\mathbf{e}_m$$

and we conclude similarly that

$$|(S_N)_{kk}| - \sum_{i\neq k}|(S_N)_{ki}| = 0. \tag{12}$$

4.1 Sparsity pattern of the Schur complement

We use results from graph theory to study the sparsity pattern of the Schur complement. Given a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$, we call *adjacency matrix* of the undirected graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ a symmetric matrix $\mathcal{A}$ such that $\mathcal{A}_{ij} \neq 0$ if there exists an edge between nodes i and j.

An undirected graph is *chordal* if every cycle of length greater than three has a chord, i.e. an edge between two non-consecutive nodes in the cycle. We say that the sparsity pattern of a matrix is chordal if the matrix can be interpreted as the adjacency matrix of a chordal graph. The following Theorem relates chordal graphs and positive definite matrices.

Theorem 1. *A sparsity pattern $\mathcal{S}$ is chordal if and only if for every positive definite matrix with sparsity pattern $\mathcal{S}$ there exists a symmetric permutation of its rows and columns that produces a Cholesky factor with zero fill-in.*

Proof. See [30, Theorem 9.1]. □

An undirected bipartite graph has an adjacency matrix of the kind

$$\mathcal{A} = \begin{bmatrix} 0 & \mathcal{M} \\ \mathcal{M}^T & 0 \end{bmatrix} \tag{13}$$

where $\mathcal{M}$ is called the *biadjacency matrix*. Equivalently, given any matrix $\mathcal{M}$, we can build a corresponding bipartite graph for which $\mathcal{M}$ is its biadjacency matrix.

The following result is important for the matrices considered in this paper:

Theorem 2. *Given a matrix $\mathcal{M}$ for which the corresponding bipartite graph is acyclic, $\mathcal{M}\mathcal{M}^T$ and $\mathcal{M}^T\mathcal{M}$ have chordal sparsity patterns.*

Proof. Let us call $\mathcal{N}_0$ and $\mathcal{N}_0'$ the two groups of nodes in the bipartite graph; we call this graph the primary graph and indicate it as $\mathcal{PG}(\mathcal{N}_0, \mathcal{N}_0'; \mathcal{E}_0)$. This graph is acyclic by hypothesis: a trivial property of acyclic graphs is that, if there are t nodes, there cannot be more than $t-1$ edges.

Since it is a bipartite graph, we can construct its adjacency matrix as in (13). We notice that $\mathcal{A}^2$ has two nonzero blocks

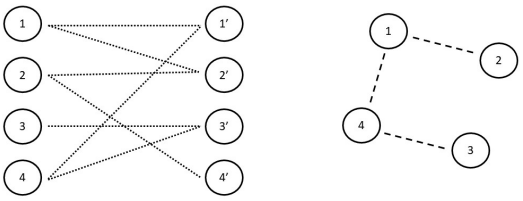
$$\mathcal{A}^2 = \begin{bmatrix} \mathcal{M}\mathcal{M}^T & 0 \\ 0 & \mathcal{M}^T\mathcal{M} \end{bmatrix}.$$

A known fact is that $\mathcal{A}^2$ tells us which nodes are connected by paths of length two; in this case, it is possible only if both nodes belong to $\mathcal{N}_0$ or to $\mathcal{N}_0'$, thus explaining the structure of $\mathcal{A}^2$.

Let us focus on the first of the two blocks $\mathcal{M}\mathcal{M}^T$. We can build a secondary graph using only the nodes $\mathcal{N}_0$; two nodes are connected by an edge in the secondary graph if there is a path of length two between them in the primary graph. We denote the secondary graph as $\mathcal{SG}(\mathcal{N}_0, \mathcal{E}_2)$, where the subscript 2 indicates that the edges correspond to 2-paths in $\mathcal{PG}$. Notice that $\mathcal{M}\mathcal{M}^T$ has the same sparsity pattern as the adjacency matrix of $\mathcal{SG}$.

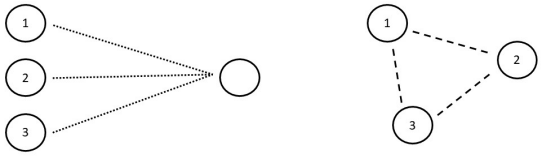
Figure 2 shows an example of an acyclic primary graph and the corresponding secondary graph. We have not represented self-loops, which are always present in $\mathcal{SG}$ by construction. In the following pictures, we use dotted lines to indicate the primary graph and dashed lines for the secondary graph.

Figure 2: Primary graph $\mathcal{PG}$ and associated secondary graph $\mathcal{SG}$.



We want to prove that any secondary graph associated with an acyclic primary graph is chordal. We start by analyzing cycles in the secondary graph. A trivial way to have a cycle in $\mathcal{SG}$ is if a subset of nodes in $\mathcal{N}_0$ are all connected to the same node in $\mathcal{N}_0'$, in the graph $\mathcal{PG}$. Figure 3 shows an example of this. In particular, any subset of nodes in $\mathcal{N}_0$ that are all connected to the same node in $\mathcal{N}_0'$ forms a complete subgraph of $\mathcal{SG}$.

Figure 3: Construction of a cycle in $\mathcal{SG}$.



We call primary cycle and secondary cycle respectively a cycle in $\mathcal{PG}$ and in $\mathcal{SG}$. Similarly, we define primary edges and secondary edges.

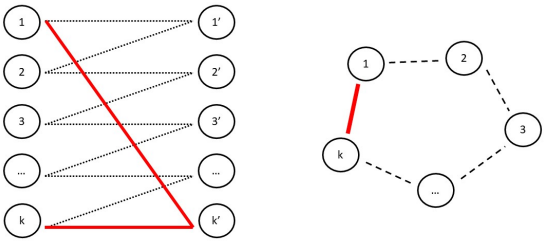
Let us try to construct a chordless secondary cycle of length at least 4 (the existence of such a cycle would imply the non-chordality of $\mathcal{SG}$). Let us suppose that the cycle is formed by a subset of nodes $\mathcal{Q} \subseteq \mathcal{N}_0$. We know that if there are three or more nodes in $\mathcal{Q}$ connected to the same node in $\mathcal{N}_0'$ in the primary graph, then there exists a chord in the secondary cycle; thus, let us assume that this does not happen. This means that we need an *auxiliary* node in $\mathcal{N}_0'$ to connect in the primary graph each couple of subsequent nodes in the secondary cycle.

Suppose that the chordless cycle in the secondary graph is $q_1 \to q_2 \to \cdots \to q_k \to q_1$, with $k \geq 4$. To form the secondary edge between q_1 and q_2, we need a path of length two in the primary graph which involves a node $q_1' \in \mathcal{N}_0'$. To form the secondary edge between q_2 and q_3, we need a path of length two in the primary graph which involves a node $q_2' \in \mathcal{N}_0'$, since we cannot use q_1' again or we would produce a chord. Continuing this reasoning, to have a chordless cycle we need to use a different auxiliary node $q_j' \in \mathcal{N}_0'$ for each pair of subsequent nodes in the secondary cycle. This implies that any edge in $\mathcal{SG}$ corresponds to two edges in $\mathcal{PG}$. To construct the chordless cycle, we would need k auxiliary nodes in $\mathcal{N}_0'$ and $2k$ edges in the primary graph. However, this would mean that $\mathcal{PG}$ has a subset of $2k$ nodes with at least $2k$ edges, which would imply the presence of a primary cycle. This is a contradiction and we conclude that a chordless secondary cycle of length more than 4 cannot exist. The proof for the second block $\mathcal{M}^T\mathcal{M}$ is identical to the one shown here, but the secondary graph

is constructed considering the nodes $\mathcal{N}_0'$.

Figure 4 shows this reasoning: it is possible to construct a chordless path using $k-1$ auxiliary vertices, but it is impossible to close the secondary cycle without creating a cycle in the primary graph. □

Figure 4: Construction of a chordless cycle.



We will use this Theorem to analyze the sparsity pattern of the Schur complements obtained using the optimal solution $(\hat{\mathbf{p}}, \hat{\mathbf{y}}, \hat{\mathbf{s}})$; however, when the algorithm approaches optimality and the complementarity products get close to zero, the entries of Θ tend to zero or infinity. For the sake of formalism, define the matrix $\mathcal{V}$ as the sparsity pattern of V; then during the IPM iterations, $\mathcal{V} \to \hat{\mathcal{V}}$, where

$$\begin{cases} \hat{\mathcal{V}}_{ij} = 1 & \text{if} \quad \hat{\mathcal{P}}_{ij} > 0 \\ \hat{\mathcal{V}}_{ij} = 0 & \text{if} \quad \hat{\mathcal{P}}_{ij} = 0 \end{cases},$$

where $\text{vec}(\hat{\mathcal{P}}) = \hat{\mathbf{p}}$. At a certain IPM iteration, the Schur complements have the same sparsity pattern as VV^T or V^TV; therefore, getting close to optimality, they tend to have sparsity pattern $\hat{\mathcal{V}}\hat{\mathcal{V}}^T$ or $\hat{\mathcal{V}}^T\hat{\mathcal{V}}$. The next result shows that these are chordal matrices.

Corollary 1. *Matrices $\hat{\mathcal{V}}\hat{\mathcal{V}}^T$ and $\hat{\mathcal{V}}^T\hat{\mathcal{V}}$ have chordal sparsity patterns.*

Proof. Recall the graph formulation presented in Section 2.2: $\hat{\mathcal{P}}$ is the biadjacency matrix of the bipartite graph corresponding to the optimal solution, which is known to be acyclic. By construction, $\hat{\mathcal{V}}$ and $\hat{\mathcal{P}}$ have the same sparsity pattern. Therefore, matrix $\hat{\mathcal{V}}$ satisfies the assumptions of Theorem 2. □

4.2 Mixing direct and iterative solvers

We consider two options for solving system (7) with matrix (9): direct approach using Cholesky factorization [14] and iterative one using preconditioned conjugate gradient [19].

We can identify two stages of the IPM. In the early iterations, the index set is far from the optimal one, so we expect that if we compute the Cholesky factorization of the Schur complement, there would be a lot of fill-in; in this phase though, Θ is well conditioned and we can expect a simple preconditioner like incomplete Cholesky to work well. In the late iterations instead, we know that the normal equations matrix becomes extremely ill-conditioned due to the behaviour of matrix Θ, which makes it difficult to find a good preconditioner for the conjugate gradient method; however, computing a full Cholesky factorization

of the Schur complement would be extremely cheap since both S_M and S_N get closer and closer to a sparse chordal matrix, as was showed in Corollary 1.

The proposed approach first applies the conjugate gradient with incomplete Cholesky as preconditioner, with a value of the drop tolerance that is lowered every time too many linear iterations are performed; then, based on a switching criterion, when we decide that the Schur complement is "close enough" to the optimal one, we switch to a direct solution using an exact factorization with approximate minimum degree ordering.

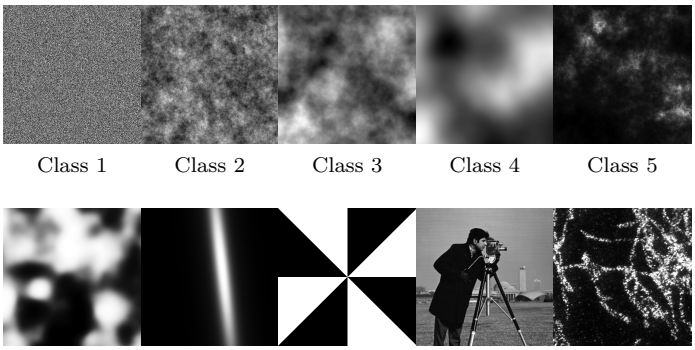
The Schur complements are diagonally dominant, as shown in (11)-(12) and this guarantees that the incomplete factorization never breaks down in exact arithmetic; however, since the matrices are only weakly diagonally dominant, we may need to lift their diagonal with a small coefficient, in order to prevent from numerical inaccuracies making the matrices lose diagonal dominance property.

When using a full factorization, we employ the LDL^T algorithm; since both Schur complements are singular, we expect one entry of matrix D to be zero. We deal with this problem in two ways: we employ Matlab backslash operator, which takes care of the rank deficiency solving a least-squares problem instead; we add a small shift to the diagonal of D as well, since the entry that should be zero could become negative due to numerical errors.

4.3 Switching strategy

In order to switch between the iterative solver and the direct factorization, we need to be sure that the level of fill-in in the Cholesky factor is not going to be too large, or otherwise the method may lose efficiency. Matrix V which is used to build the Schur complement is the biadjacency matrix of a bipartite graph that is not perfectly acyclic (this happens only at optimality), but gets closer and closer to being so; we can expect the number of cycles found in the graph to decrease and correspondingly the fill-in level of the Cholesky factor of the Schur complements to be reduced when the IPM gets close to optimality. One strategy to switch between iterative and direct solver can be based on the number of cycles that are found in the bipartite graph associated with matrix V at any IPM iteration; however, counting the number of cycles in the graph can be quite expensive. A simpler approach is to count the number of edges in the bipartite graph to decide when to switch: intuitively, the more edges there are, the more likely it is to find cycles; moreover, if the number of edges is decreasing fast, it means that the IPM is getting close to optimality and thus we can expect the Cholesky factor to be sufficiently sparse. Notice that each edge corresponds to a variable that is allowed to be nonzero; the number of edges is thus readily available, since it is the number of variables in the support. Therefore, we can switch from iterative to direct solver as soon as we detect that the number of variables in the support is decreasing fast enough: to do so, we compute its relative variation in the last 5 iterations and switch method as soon as this number is sufficiently large.

Figure 5: Example of images from each class of the DOTmark collection.



5 Numerical results

5.1 Test problems

We tested the sparse IPM algorithm on the *DOTmark* (Discrete Optimal Transport benchmark) collection of problems [29]. It includes 10 classes of images, each containing 10 images; the images come from simulations based on various probability distributions (class 1-7), geometric shapes (class 8), classic test images (class 9) and scientific observations using microscopy (class 10). Figure 5 shows one example from each class.

Within a certain class, we can solve the OT problem between any couple of images, giving 45 problems per each class and a total of 450 instances overall. The images are available at different resolutions: we considered mainly the case 32×32 and 64×64 pixels, and show partial results also with 128×128 and 256×256. The problems that arise for a certain resolution `res` have a number of constraints equal to $2\,\texttt{res}^2$ and a number of variables $\texttt{res}^4$; Table 1 shows the dimension of the problems considered.

Table 1: Size of the problems solved

`res`	Constraints	Variables ($\times 10^6$)
32	2,048	1.0
64	8,192	16.8
128	32,768	268.4
256	131,072	4,295.0

As cost function, we consider the $1-$distance, $2-$distance and $\infty-$distance: the vectors $\mathbf{a}$ and $\mathbf{b}$ represent the vectorized forms of two images $\mathcal{A}$ and $\mathcal{B}$; $\mathcal{C}_{ij}$ is the cost of moving mass from position i to position j; in the images, position i corresponds to the entry $\mathcal{A}_{\alpha_i,\beta_i}$, while position j corresponds to $\mathcal{B}_{\alpha_j,\beta_j}$. Then, the distances used are

$$\mathcal{C}^1_{ij} = |\alpha_i - \alpha_j| + |\beta_i - \beta_j|,$$

$$\mathcal{C}^2_{ij} = \sqrt{(\alpha_i - \alpha_j)^2 + (\beta_i - \beta_j)^2},$$

$$\mathcal{C}_{ij}^{\infty} = \max(|\alpha_i - \alpha_j|, |\beta_i - \beta_j|).$$

Notice the following: if we consider an image with k pixels per side, the maximum possible distance between two pixels is approximately $2k$, $\sqrt{2}k$ or k, respectively for the 1, 2 and ∞ distance; all of these values grow linearly with k. Therefore, the parameter $C_{\max}$ in Section 3.2.1 should also scale linearly with the size of the image considered: in this way, the pricing heuristics considers always the same fraction of the total number of variables for each resolution considered.

5.2 Solvers

There are many methods that have been derived for the solution of the Kantorovich linear program (3) (see e.g. [2, 9, 18, 24, 28]); a comparison of them can be found in [29]. Based on the results and the conclusions drawn there, we decided to compare the sparse IPM to the IBM Cplex [1] network simplex method [4, 26]: this solver exploits the network structure of the problem to perform very cheap simplex iterations. Its main advantages are that it is a very efficient solver, that showed extreme consistency of computational times throughout all the 10 classes of the DOTmark collection; it does not need to be adapted to solve different problems, as it can deal with any kind of test problem and cost function chosen (which may not be true for other specially developed optimal transport solvers). Even if it may not always be the fastest approach, Cplex has been demonstrated to be the first choice in terms of reliability and versatility.

Both methods are called through Matlab: the IPM is completely written in Matlab, while the network simplex is only called through Matlab, but it exploits a highly optimized code written in C. In order to use the network solver of Cplex, we form the incidence matrix of the graph (5) and pass it to Cplex, which then extracts the network information. In order to not perturb the network structure, we switched off the preprocessing.

Although the network extraction operation in Cplex can be time consuming, it is a necessary initial phase which later greatly benefits the network-based optimization solver because this allows it to exploit the graph structure and thus avoid many numerical operations. To give the reader an idea of the cost of this operation we will report the results of Cplex runs both with and without the network extraction time. On the other hand, the proposed IPM requires only to store the vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. The matrix A in (4) is implicit.

The parameters for the IPM are: feasibility and optimality tolerance 10^{-6}; conjugate gradient tolerance for predictors 10^{-6} and correctors 10^{-3} (we used a different tolerance for predictors and correctors, as was shown in [36]); maximum number of IPM iterations 200; maximum number of conjugate gradient iterations (for each call) 1000; maximum number of correctors 3.

5.3 Results

We show the results obtained running the two solvers on a computer with an i5-8350U quad-core processor @1.7 GHz and 16 GB of RAM.

The IPM is an inexact method, while the network simplex finds the exact solution. In the results, we show how close the IPM solution is to the exact one

in terms of the relative Wasserstein error (RWE):

$$RWE(\mathbf{a},\mathbf{b}) = \Big|\frac{W_2^{\text{IPM}}(\mathbf{a},\mathbf{b}) - W_2^{\text{Cplex}}(\mathbf{a},\mathbf{b})}{W_2^{\text{Cplex}}(\mathbf{a},\mathbf{b})}\Big|$$

where the 2-Wasserstein distance is defined in (2). Notice that

$$\begin{aligned} RWE(\mathbf{a},\mathbf{b}) &= \Big|\frac{\sqrt{\mathbf{c}^T\hat{\mathbf{p}}^{\text{IPM}}} - \sqrt{\mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}}{\sqrt{\mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}}\Big| \\ &= \Big|\frac{\mathbf{c}^T\hat{\mathbf{p}}^{\text{IPM}} - \mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}{\mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}} + \sqrt{\mathbf{c}^T\hat{\mathbf{p}}^{\text{IPM}}}\sqrt{\mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}}\Big| \\ &\leq \Big|\frac{\mathbf{c}^T\hat{\mathbf{p}}^{\text{IPM}} - \mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}{\mathbf{c}^T\hat{\mathbf{p}}^{\text{Cplex}}}\Big| \end{aligned}$$

We expect the last quantity to be of the order of magnitude of the IPM tolerance and thus the RWE to be even smaller.

Table 2 reports the average results for each class, for resolutions 32×32 and 64×64 and for each cost function considered, in terms of IPM iterations, IPM time, Cplex time (including the network extraction time) and relative Wasserstein error. Table 3 instead reports the average over all the three cost functions for each class of the number of CG iterations performed (`CG`), the maximum fill-in percentage level of the factorization of the Schur complement (`fill-in`) and the number of iterative (`iter-it`) and direct (`dir-it`) iterations performed.

Remark 1. *The reader should keep in mind that the number of iterations presented refers to the overall process of adding/removing variables and optimizing; it is thus not a surprise that there are instances where the iteration count is higher than what would be expected from a standard IPM. The cost of a single iteration however is considerably lower, given the high degree of sparsity of the vectors and matrices involved.*

Figures 6-8-9 show on the left (figures **(a)** and **(c)**) the distribution of the computational times for the IPM and Cplex for each problem of each class, and on the right (figures **(b)** and **(d)**) the performance profiles of the computational time.

Figure 10 shows the performance profile for the resolution 64×64 when the network extraction time is ignored, for the $1-$distance **(a)**, $2-$distance **(b)** and $\infty-$distance **(c)**; the last figure **(d)** shows the overall performance profile when considering the 1,350 problems arising from all three cost functions.

From these results, it is clear that for the smaller problems (32×32) Cplex has an advantage, but for the larger instances (64×64) the IPM becomes very competitive, for all three cost functions considered. Given that Cplex is a highly optimized commercial software while the IPM is implemented in Matlab, we consider these results very promising. Even ignoring the network extraction time (which gives a great advantage to Cplex), we can see that the IPM is very competitive with the network simplex method, in particular for the $1-$distance and $\infty-$distance. Moreover, all the solutions computed with the IPM are very accurate, since the average relative Wasserstein error is of the order of $10^{-6}-10^{-8}$ in all cases.

Table 2: Average results for each class and cost function

		32×32				64×64			
Dist	Class	Iter	IPM t	Cplex t	RWE	Iter	IPM t	Cplex t	RWE
1	1	11.4	0.35	0.62	1.2e-07	14.4	2.18	20.92	5.5e-08
	2	11.7	0.39	0.60	1.4e-07	18.1	3.46	20.64	4.5e-08
	3	15.9	0.59	0.61	2.4e-08	26.8	6.02	20.83	2.1e-08
	4	20.3	0.85	0.57	2.0e-08	38.4	9.69	20.69	2.1e-08
	5	25.6	1.16	0.61	1.4e-08	40.8	10.78	21.84	1.6e-08
	6	18.8	0.72	0.64	3.3e-08	36.2	9.04	23.25	1.3e-08
	7	30.8	1.47	0.57	3.8e-08	72.2	39.11	21.80	2.3e-08
	8	17.4	0.65	0.58	3.8e-08	52.5	21.69	18.55	8.7e-08
	9	14.9	0.52	0.60	2.8e-08	25.0	5.24	21.27	1.4e-08
	10	22.4	0.92	0.62	2.0e-08	40.8	10.48	18.33	2.1e-08
2	1	18.7	0.61	0.73	7.6e-08	26.0	4.66	21.67	1.2e-07
	2	29.5	1.00	0.76	4.3e-07	74.1	14.07	24.31	1.0e-06
	3	53.0	1.92	0.81	1.1e-06	89.1	21.69	27.10	9.9e-07
	4	60.1	2.38	0.81	1.4e-06	102.4	31.06	28.59	1.2e-06
	5	59.0	2.30	0.83	6.8e-07	95.4	28.25	30.02	5.5e-07
	6	46.0	1.98	0.88	6.9e-07	89.4	22.70	29.45	7.8e-07
	7	66.6	3.03	0.82	8.8e-07	108.4	35.47	29.48	1.3e-06
	8	37.3	1.37	0.74	5.0e-07	79.9	20.85	23.62	4.6e-06
	9	43.8	1.57	0.81	8.5e-07	89.7	17.48	27.66	1.0e-06
	10	47.3	1.81	0.81	5.1e-07	99.2	24.57	22.17	8.2e-07
∞	1	12.3	0.51	0.70	3.8e-08	15.6	3.18	19.90	3.3e-08
	2	12.4	0.53	0.69	2.2e-07	17.5	4.42	20.43	9.3e-08
	3	15.7	0.76	0.67	5.5e-08	24.9	7.74	21.05	2.1e-08
	4	18.7	0.99	0.67	2.7e-08	35.8	12.53	21.01	1.6e-08
	5	24.0	1.35	0.67	3.5e-08	36.4	12.69	21.62	2.0e-08
	6	18.8	0.95	0.71	4.9e-08	30.0	9.98	23.11	1.4e-08
	7	28.7	1.74	0.66	4.7e-08	52.5	20.92	21.53	1.7e-08
	8	16.5	0.99	0.65	5.1e-08	37.9	23.08	18.14	2.1e-08
	9	15.0	0.70	0.69	1.4e-07	24.6	7.33	21.05	2.1e-08
	10	21.0	1.11	0.68	2.8e-08	35.9	12.04	18.58	2.2e-08

The IPM behaves particularly well for problems in classes 1 and 2, taking almost always considerably less time than Cplex; it behaves well also for classes 3, 6, 9 and 10, especially for the larger problems; class 7 is the one that challenges the IPM the most, since only in a few occasions it is able to outperform Cplex for this class. Observing the images from these classes (Figure 5), it seems that the IPM performs better when the mass of the image is distributed evenly everywhere, while it struggles when it is concentrated in a narrow region.

From Table 3 we can see that the maximum fill-in level is independent of the class of the images and gets smaller when the problem becomes larger, since the density of the optimal solution decreases as $1/\texttt{res}^2$. We can also see that on average only the very last IPM iterations employ a direct factorization, which is what we wanted to achieve with the switching criterion proposed.

5.4 Results for large instances

When considering larger problems with resolution 128×128, Cplex runs out of memory when generating the network structure, while the IPM does not. This highlights that the proposed method scales better with dimension and uses less memory, since it does not have to store the huge network structure of the

Table 3: Details of the IPM method for each class

	32×32				64×64			
Class	`CG`	`fill-in`	`iter-it`	`dir-it`	`CG`	`fill-in`	`iter-it`	`dir-it`
1	1011.8	10.0	13.6	0.5	1704.1	4.2	17.2	1.5
2	857.6	11.9	15.0	2.9	1867.4	4.3	27.4	9.2
3	1035.9	12.0	23.6	4.6	2392.8	2.9	42.5	4.5
4	1207.9	11.6	29.8	3.2	2913.4	2.9	55.1	3.7
5	1070.3	12.4	29.5	6.7	2801.6	3.1	49.7	7.8
6	1183.0	11.5	26.1	1.7	2810.4	3.1	49.1	2.7
7	1546.0	11.9	38.8	3.2	4179.9	3.7	74.0	3.7
8	1170.6	11.5	22.2	1.5	3650.2	3.8	53.2	3.6
9	1061.8	11.5	22.3	2.3	2504.5	2.9	42.4	4.1
10	1104.5	13.5	26.2	4.0	3430.2	4.3	45.9	12.8

problem. The largest memory requirement comes from storing the cost vector $\mathbf{c}$ and the factorization of the Schur complement.

We performed tests with larger problems using the University of Edinburgh School of Mathematics computing server, which is equipped with four 3.3GHz octa-core Intel Gold 6234 processors and 500GB of RAM. Unfortunately, we are not able to perform tests with Cplex on this computer.

We show the results for the whole collection of problems at resolution 128×128; for resolution 256×256, given the huge size of the problems, we show only partial results for classes 1 and 2.

5.4.1 Resolution up to 128

In this section we demonstrate how the computational time depends on the resolution of the images (32, 64 and 128 pixels) and the different cost functions (1, 2 and $\infty-$norm). We also highlight how our method scales with the dimension.

For each class, Figure 11 shows in logarithmic scale the comparison of the time taken by the various problems for each resolution considered. Each increase in resolution produces a problem with 16 times more variables than the previous one and this is reflected in the times shown; the results remain well clustered with some exceptions for classes 7 and 8, which present a concentration of mass in narrow regions. Overall, the times appear to scale proportionally with the dimension of the problem; the behaviour of the algorithm is not affected by considering huge scale problems with 128 pixels.

5.4.2 Resolution up to 256

For the problems in classes 1 and 2 of the collection, we performed tests with resolution up to 256×256. In Figure 7 we show the average of the computational time and memory required against the number of variables, in a logarithmic plot, when considering the 270 problems coming from all three cost functions and from both classes 1 and 2.

The growth appears linear in this plot; the slope of these lines is slightly smaller than 1, indicating that the growth is linear also if we consider a standard plot. This implies that the proposed specialized IPM is scalable both in terms of computational time and memory.

6 Conclusion

In this paper, a sparse interior point method was introduced for solving linear programs arising from discrete optimal transport. A careful analysis of the matrices involved allows for an efficient way of solving the Newton linear systems that arise within the IPM. Experimental results show that the proposed approach is able to outperform one of the best off-the-shelf software available, even when the time taken for the construction of the network model is ignored. Results with huge scale problems, with up to four billion variables, confirm the robustness of the method.

Further research can potentially improve the performance of the sparse IPM even more, in particular in relation to:

- The choice of the starting subset of nonzero variables, which could be obtained by some more complicated heuristics, based on the initial and final distribution of mass; a better starting point could allow for a smaller size of the support, which would lead to faster computations.
- The switching strategy from iterative method to direct factorization, which could reduce even further the maximum fill-in level of the Cholesky factor and/or the number of overall conjugate gradient iterations.
- The pricing algorithm.

Moreover, the entropic regularized OT problem can be trivially dealt with using an interior point method, just by adding regularization to matrix Θ; however, the structure of the optimal solution changes and the considerations about the linear solver made in this paper need to be reconsidered for that specific case.

References

[1] *IBM ILOG CPLEX*. https://www.ibm.com/analytics/cplex-optimizer.

[2] F. Aurenhammer, F. Hoffmann, and B. Aronov, *Minkowski-Type Theorems and Least-Squares Clustering*, Algorithmica, 20 (1998), pp. 61–76.

[3] L. Bergamaschi, J. Gondzio, M. Venturin, and G. Zilli, *Inexact constraint preconditioners for linear systems arising in interior point methods*, Computational Optimization and Applications, 36 (2007), pp. 137–147.

[4] D. Bertsekas, *Network Optimization: Continuous and Discrete Models*, Athena Scientific, 1998.

[5] S. Bocanegra, F. Campos, and A. Oliveira, *Using a Hybrid Preconditioner for Solving Large-Scale Linear Systems arising from Interior Point Methods*, Computational Optimization and Applications, 36 (2007), pp. 149–164.

[6] J. Castro, *A specialized interior-point algorithm for multicommodity network flows*, SIAM Journal on Optimization, 10 (2000), pp. 852–877.

[7] J. Castro and J. Cuesta, *Quadratic regularizations in an interior point method for primal block-angular problems*, Mathematical Programming, 130 (2010), pp. 415–445.

[8] J.-S. Chai and K.-C. Toh, *Preconditioning and iterative solution of symmetric indefinite linear systems arising from interior point methods for linear programming*, Computational Optimization and Applications, 36 (2007), pp. 221–247.

[9] M. Cuturi, *Sinkhorn distances: Lightspeed computation of optimal transport*, Proc. NIPS, (2013), pp. 2292–2300.

[10] V. De Simone, D. Di Serafino, J. Gondzio, S. Pougkakiotis, and M. Viola, *Sparse approximations with interior point methods*, 2022. To appear in SIAM Review, arXiv:2102.13608.

[11] E. Facca and M. Benzi, *Fast iterative solution of the optimal transport problem on graphs*, SIAM J. Sci. Comput., 43 (2021), pp. A2295–A2319.

[12] K. Fountoulakis, J. Gondzio, and P. Zhlobich, *Matrix-free interior point method for compressed sensing problems*, Mathematical Programming Computation, 6 (2014), pp. 1–31.

[13] C. T. L. S. Ghidini, A. R. L. Oliveira, J. Silva, and M. I. Velazco, *Combining a hybrid preconditioner and a optimal adjustment algorithm to accelerate the convergence of interior point methods*, Linear Algebra and its Applications, 436 (2012), pp. 1267–1284.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1996.

[15] J. Gondzio, *Matrix-Free Interior Point Method*, Computational Optimization and Applications, 51 (2012), pp. 457–480.

[16] J. Gondzio, P. Gonzalez-Brevis, and P. Munari, *New developments in the primal–dual column generation technique*, European Journal of Operational Research, 224 (2013), pp. 41–51.

[17] J. Gondzio, M. Lassas, S.-M. Latva-Aijo, S. Siltanen, and F. Zanetti, *Material-separating regularizer for multi-energy x-ray tomography*, Inverse Problems, 38 (2022).

[18] C. Gottschlich and D. Schuhmacher, *The Shortlist Method for Fast Computation of the Earth Mover's Distance and Finding Optimal Solutions to Transportation Problems*, PLoS ONE, 9 (2014), p. e110214.

[19] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Research of NIST, 49 (1952), pp. 409–436.

[20] R. Horn and C. Johnson, *Matrix Analysis*, Cambridge University Press, 1990.

[21] L. Kantorovich, *On the translocation of masses*, Manag. Sci., 5 (1958), pp. 1–4.

[22] H. Ling and K. Okada, *An efficient earth mover's distance algorithm for robust histogram comparison*, IEEE Trans. Pattern Anal. Mach. Intell., 29 (2007), pp. 840–853.

[23] M. E. Lubbecke and J. Desrosiers, *Selected Topics in Column Generation*, Operations Research, 53 (2005), pp. 1007–1023.

[24] Q. Merigot, *A Multiscale Approach to Optimal Transport*, Computer Graphics Forum, 30 (2011), pp. 1583–1592.

[25] A. R. L. Oliveira and D. C. Sorensen, *A New Class of Preconditioners for Large-Scale Linear Systems from Interior Point Methods for Linear Programming*, Linear Algebra and its Applications, 394 (2005), pp. 1–24.

[26] J. Orlin, *A polynomial time primal network simplex algorithm for minimum cost flows*, Mathematical Programming, 78 (1996), pp. 109–129.

[27] G. Peyre and M. Cuturi, *Computational Optimal Transport: With Applications to Data Science*, Foundations and Trends in Machine Learning, 11 (2019), pp. 355–607.

[28] B. Schmitzer, *A Sparse Multiscale Algorithm for Dense Optimal Transport*, Journal of Mathematical Imaging and Vision, 56 (2016), pp. 238–259.

[29] J. Schrieber, D. Schuhmacher, and C. Gottschlich, *DOTmark - A Benchmark for Discrete Optimal Transport*, IEEE Access, 5 (2017), pp. 271–282.

[30] L. Vandenberghe and M. Andersen, *Chordal Graphs and Semidefinite Optimization*, Foundations and Trends in Optimization, 1 (2014), pp. 241–433.

[31] F. Vanderbeck, *Implementing mixed integer column generation*, in Column Generation, J. Desaulniers, J. Desrosiers, and M. M. Solomon, eds., Springer, US, 2005, pp. 331–358.

[32] F. Vanderbeck and L. A. Wolsey, *Reformulation and Decomposition of Integer Programs*, in 50 Years of Integer Programming 1958–2008, M. Junger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, eds., Springer, Berlin Heidelberg, 2010, pp. 431–502.

[33] M. I. Velazco, A. R. L. Oliveira, and F. F. Campos, *A note on hybrid preconditioners for large-scale normal equations arising from interior-point methods*, Optimization Methods and Software, 25 (2010), pp. 321–332.

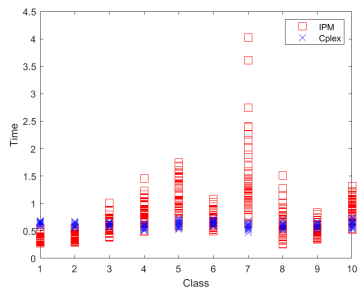
[34] J. Wijesinghe and P. Chen, *Matrix-free interior point methods for point set matching problems*, 2022. arXiv:2202.09763[math.OC].

[35] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, 1997.

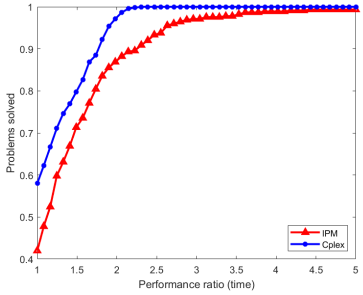
[36] F. Zanetti and J. Gondzio, *A new stopping criterion for Krylov solvers applied in Interior Point Methods*, 2021. arXiv:2106.16090 [math.OC].

A Result figures

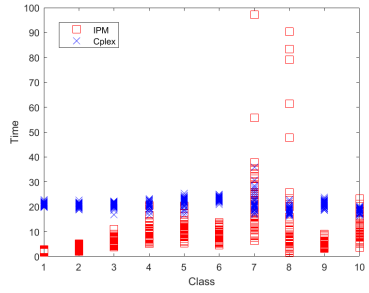
Figure 6: Comparison of the time of Cplex and SparseIPM for the 1−distance



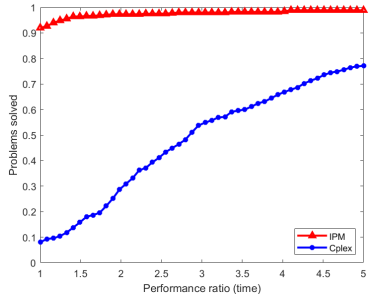
(a) Results 32×32



(b) Performance profile 32×32

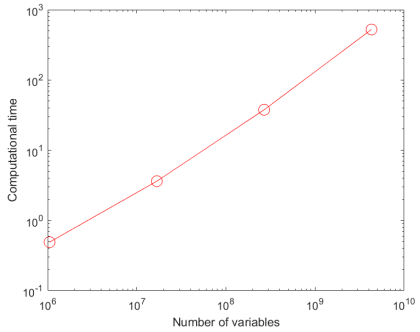


(c) Results 64×64

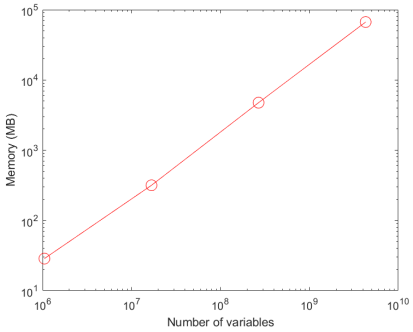


(d) Performance profile 64×64

Figure 7: Logarithmic plot of computational time and memory requirement for the problems in class $1-2$, for resolution up to 256 pixels

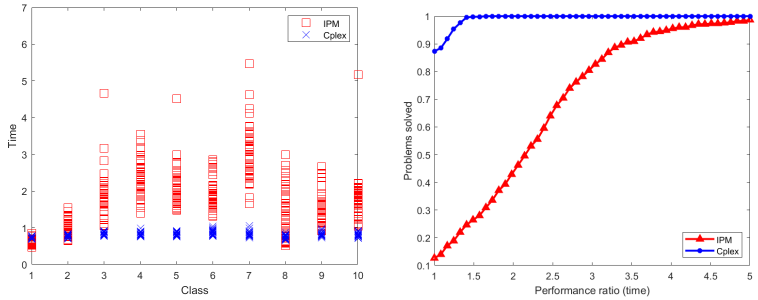


(a) Time (s)



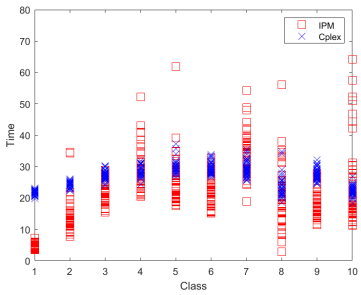
(b) Memory (MB)

Figure 8: Comparison of the time of Cplex and SparseIPM for the 2−distance

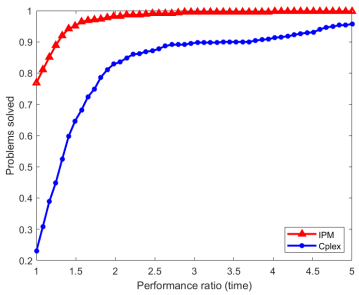


(a) Results 32×32

(b) Performance profile 32×32

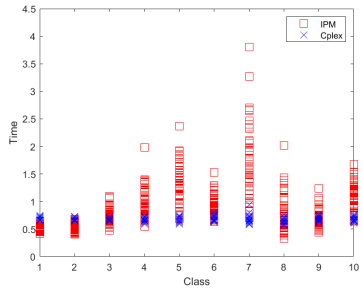


(c) Results 64×64

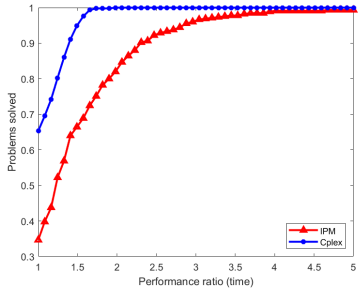


(d) Performance profile 64×64

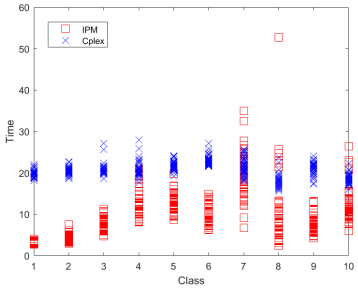
Figure 9: Comparison of the time of Cplex and SparseIPM for the ∞−distance



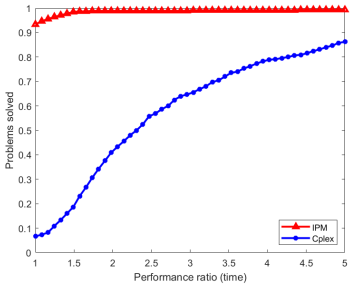
(a) Results 32×32



(b) Performance profile 32×32

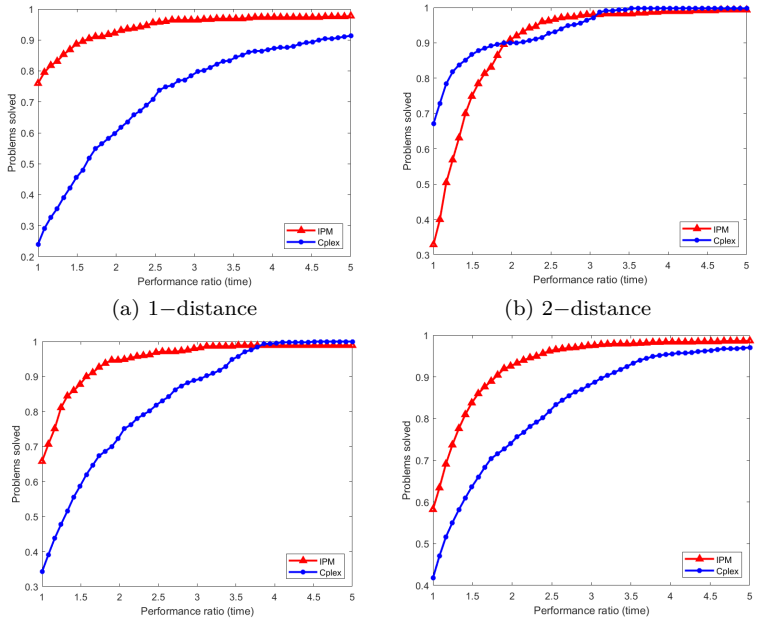


(c) Results 64×64



(d) Performance profile 64×64

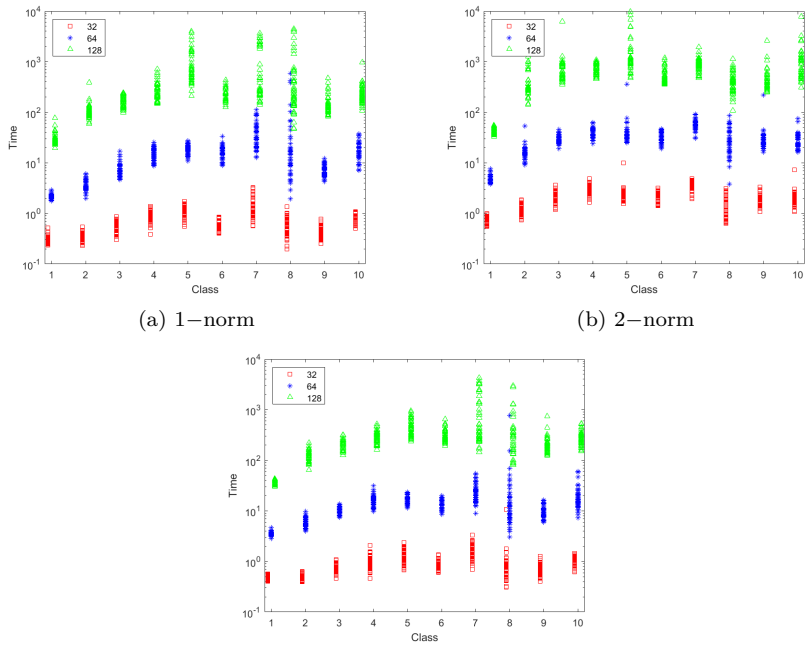
Figure 10: Comparison of the time of Cplex and SparseIPM without network extraction time



(a) 1−distance (b) 2−distance

(c) ∞−distance (d) Overall

Figure 11: Computational time of SparseIPM for resolution up to 128 pixels



(a) 1−norm (b) 2−norm

(c) ∞−norm