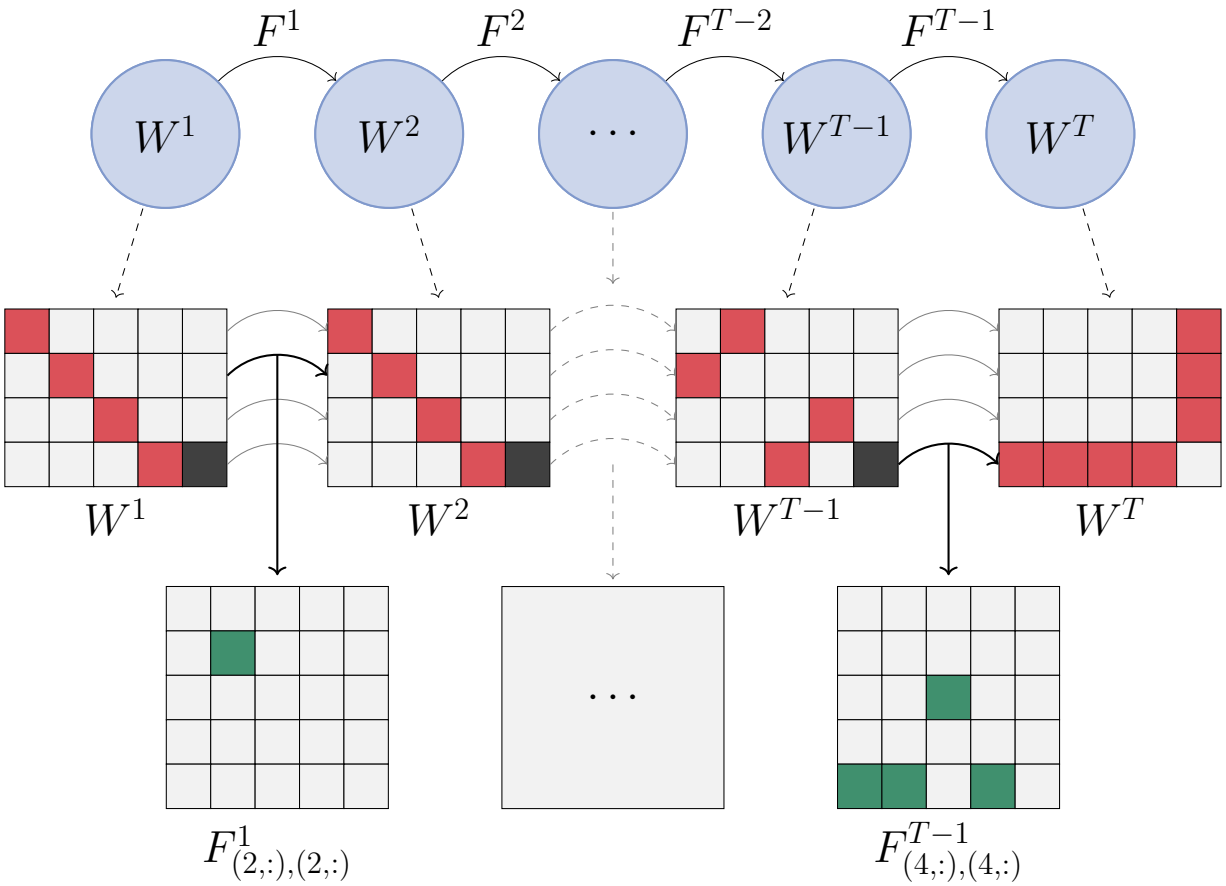
CHALMERS
UNIVERSITY OF TECHNOLOGY



Efficient Evaluation of Target Tracking Using Entropic Optimal Transport

Scalable Computation of the GOSPA Metric for Trajectories

Master's Thesis in Engineering Mathematics and Computational Science

VIKTOR NEVELIUS WERNHOLM
ALFRED WÄRNSÄTER

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's Thesis 2024

Efficient Evaluation of Target Tracking Using Entropic Optimal Transport

Scalable Computation of the
GOSPA Metric for Trajectories

VIKTOR NEVELIUS WERNHOLM
ALFRED WÄRNSÄTER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2024

Efficient Evaluation of Target Tracking Using Entropic Optimal Transport
Scalable Computation of the GOSPA Metric for Trajectories
VIKTOR NEVELIUS WERNHOLM
ALFRED WÄRNSÄTER

Supervisor: Axel Ringh, Department of Mathematical Sciences
Supervisor: Adam Andersson, Department of Mathematical Sciences and Saab AB
Supervisor: Per Ljung, Saab AB
Examiner: Axel Ringh, Department of Mathematical Sciences

Master's Thesis 2024
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of the network flow formulation of the GOSPA metric for trajectories presented in this thesis. See Section 4.2.1 for more details.

Typeset in LaTeX
Gothenburg, Sweden 2024

Efficient Evaluation of Target Tracking Using Entropic Optimal Transport
Scalable Computation of the GOSPA Metric for Trajectories
VIKTOR NEVELIUS WERNHOLM
ALFRED WÄRNSÄTER
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Multiple target tracking deals with the task of estimating targets which appear, disappear, and move within a scene, given data from noisy measurements. To solve this task, a wide range of algorithms can be employed. In order to assess the performance of such algorithms, the so-called GOSPA metric for trajectories can be applied. This metric is formulated as an optimization problem, which has proven computationally demanding for large problem instances. In this thesis, we reformulate this metric in two different ways to obtain optimization problems with optimal transport structure. Following a recent breakthrough in computational optimal transport, we introduce entropic regularization into these formulations. For the regularized problems, we derive and present two numerical algorithms for finding approximate solutions. We test the performance of each algorithm on simulated data with regards to accuracy and computational efficiency. The numerical results suggest that the regularization can be made small enough to allow for an adequate approximation of the GOSPA metric for trajectories while simultaneously allowing a satisfactory convergence rate. Lastly, we compare the running time of our most efficient algorithm with that of a conventional linear programming solver. If a small approximation error is allowed, we find that our algorithm scales better both when the number of trajectories in the data increases, and when the number of considered time steps in the data increases.

Keywords: multiple target tracking, GOSPA metric, convex optimization, duality, optimal transport, tensor optimization problem, network flow problem, entropic regularization, coordinate ascent, Sinkhorn iterations

Acknowledgements

While writing this thesis, we have received guidance and support from various people, to whom we are truly grateful. First and foremost, we would like to thank our academic supervisor Axel Ringh. Your pragmatic and supportive attitude, combined with a great amount of patience, has made working with you a delight. Further, we would like to thank our supervisors at Saab, Adam Andersson and Per Ljung, for sharing their wealth of experience on several different topics. We would also like to thank everyone at Saab who has helped us with this thesis. Moreover, we would like to thank the other thesis workers at Saab, with whom we have spent a significant amount of time. Last but not least, we would like to thank our friends and family, without whose love and support this thesis would not have been possible.

Viktor Nevelius Wernholm and Alfred Wärnsäter, Gothenburg, June 2024

Contents

1

Introduction

Multiple target tracking deals with the task of estimating targets which appear, disappear, and move within a scene, given data from noisy measurements. The goal is to obtain an estimate that is as close as possible to the actual state of the targets, referred to as the ground truth. To solve this task, a wide range of algorithms can be employed. Naturally, different algorithms produce estimates with at least slight differences for the same ground truth. At times, it can be easy to determine which algorithm performed better, but in most cases, it is more complicated and therefore there is a need for a fair and proper metric that assesses the performances of these algorithms. Several metrics exist, and one that has been proven particularly suitable is the so-called Generalized Optimal Sub-Pattern Assignment metric, or GOSPA metric, introduced in [1]. However, this metric only considers static targets and therefore lacks the temporal aspect of tracking. In [2], the metric was extended to also compare time-dependent estimates and ground truths. Both versions are formulated as optimization problems, and the latter is generally computationally difficult to solve with currently used methods. In this thesis, we investigate the use of so-called entropic optimal transport to compute approximate solutions to this problem efficiently.

Optimal transport is a field of mathematical optimization that concerns the optimal transportation and allocation of resources. In general, the goal of an optimal transport problem is to move mass between given distributions as efficiently as possible with respect to some metric. Even though this field originally concerned problems of transporting physical mass, it has proven useful for modeling problems from several areas not typically associated with transportation as well. Although sometimes numerically tractable, many optimal transport problems are computationally expensive. A breakthrough came in [3], where optimal transport problems were regularized using entropy, giving rise to the field of entropic optimal transport. In brief terms, an entropic regularization term was added to the objective function in order to smooth the problem. For the resulting perturbed problem, an iterative algorithm was derived, solving the regularized problem more efficiently compared to conventional solvers applied to the original problem. The idea is to keep the regularization sufficiently small in order to obtain an adequate approximation of the original problem while still obtaining a more computationally tractable formulation. Here, we apply this technique to compute approximations of the extended GOSPA

metric more efficiently.

In this thesis, we start by providing some necessary optimization preliminaries in Chapter 2. Specifically, a brief overview of optimal transport is included. Then, in Chapter 3, we introduce and define the multiple target tracking metrics introduced in [1] and [2], respectively. We also state the linear programming relaxation of the latter, which is the main optimization problem that is investigated in this thesis. Chapter 4 is the main chapter of this thesis. Here, the problem is reformulated in two different ways. Firstly, as a structured tensor optimization problem, and secondly, as a multimarginal optimal transport problem via a network flow problem formulation. From each formulation, an algorithm is derived, resulting in two algorithms. In Chapter 5, we test the performance of each algorithm on simulated data with regards to accuracy and computational efficiency. We then compare our most efficient algorithm with a conventional linear programming solver with regards to running time. Lastly, we conclude and suggest future studies related to this work in Chapter 6.

2

Optimization Preliminaries

In this chapter, we present the optimization preliminaries that are necessary for the derivations of our algorithms. As the problem of interest is convex, we first mention certain important properties of convex optimization in Section 2.1. In Section 2.2, we introduce the basics of optimal transport. First, the basic bimarginal optimal transport problem is presented, and then this is extended to the more general multimarginal optimal transport problem. We also present a discrete version of the latter. Lastly, we provide a brief explanation of entropic regularization.

2.1 Convex Optimization

This section is based on material from [4] and introduces results from convex optimization that are utilized in the derivations of our algorithms. Convex optimization concerns problems of minimizing convex functions over convex sets. A set S is said to be convex if for all points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$, then also $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2 \in S$, for all $\lambda \in (0,1)$. A function $f\colon S \to \mathbb{R}$ that satisfies

$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \leq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2), \text{ for all } \lambda \in [0,1] \text{ and all } \boldsymbol{x}_1, \boldsymbol{x}_2 \in S,$$

is said to be convex on S. If the inequality is strict when $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, then f is also said to be strictly convex on S. Conversely, a function f that satisfies

$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \geq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2),$$

is said to be concave, and if the inequality is strict when $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, it is strictly concave. One particular type of convex functions are affine functions, which in fact are concave as well. For a function f that is convex on a convex set S, a convex optimization problem is defined by

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}), \\ \text{subject to} \quad & \boldsymbol{x} \in S. \end{aligned}$$

Notice that $\min_{\boldsymbol{x} \in S} f(\boldsymbol{x}) = -\max_{\boldsymbol{x} \in S}(-f(\boldsymbol{x}))$, so maximizing a concave function over a convex set is also a convex optimization problem. Convex optimization has some particularly important properties. One of them is stated in the theorem below, sometimes referred to as the fundamental theorem of global optimality for convex problems.

Theorem 1 ([4, Theorem 4.3])**.** *Consider the problem of minimizing a function $f\colon S \to \mathbb{R}$, where S is a convex set and f is convex on S. Then, every local minimum of f over S is also a global minimum. Furthermore, if f is strictly convex on S, then f has a unique global minimum over S.*

Proof. We use a proof by contradiction as in [4, Theorem 4.3]. Suppose that $\boldsymbol{x}^*$ is a local minimum, but not a global one. Then consider some $\hat{\boldsymbol{x}} \in S$ with the property that $f(\hat{\boldsymbol{x}}) < f(\boldsymbol{x}^*)$. Let $\lambda \in (0,1)$. By the convexity of the set S and the function f, $\lambda\hat{\boldsymbol{x}} + (1-\lambda)\boldsymbol{x}^* \in S$, and

$$f(\lambda\hat{\boldsymbol{x}} + (1-\lambda)\boldsymbol{x}^*) \leq \lambda f(\hat{\boldsymbol{x}}) + (1-\lambda)f(\boldsymbol{x}^*) < f(\boldsymbol{x}^*).$$

Choosing $\lambda > 0$ sufficiently small then leads to a contradiction to the local optimality of $\boldsymbol{x}^*$. □

We end this section by defining the projection $\mathrm{proj}_S(\tilde{\boldsymbol{x}})$ of a point $\tilde{\boldsymbol{x}}$ onto a closed convex set S by

$$\mathrm{proj}_S(\tilde{\boldsymbol{x}}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in S} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2.$$

2.1.1 Duality

Here, we briefly explain the concept of duality, which is important in optimization in general and convex optimization in particular. For a more detailed review, we refer to [4]. Duality means that we consider an optimization problem from two perspectives, either by considering the ordinary problem, called the primal problem, or the so-called dual problem. Consider the convex optimization problem

$$\begin{aligned}
\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{minimize}} \quad & f(\boldsymbol{x}), && \text{(2.1a)}\\
\text{subject to} \quad & g_i(\boldsymbol{x}) \leq 0, \quad i = 1,\dots,M, && \text{(2.1b)}\\
& h_j(\boldsymbol{x}) = 0, \quad j = 1,\dots,N, && \text{(2.1c)}
\end{aligned}$$

for some functions f, $g_1,\dots,g_M$, and $h_1,\dots,h_N$, with $M, N, n \in \mathbb{N}$. Here, all functions $g_1,\dots,g_M$ defining the inequality constraints are convex, and all functions $h_1,\dots,h_N$ defining the equality constraints are affine, otherwise the problem is in general not convex. Relaxing the constraints (2.1b) and (2.1c) results in the Lagrangian function $\mathcal{L}$ of (2.1) given by

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) = f(\boldsymbol{x}) + \sum_{i=1}^{M} y_i g_i(\boldsymbol{x}) + \sum_{j=1}^{N} z_j h_j(\boldsymbol{x}).$$

The corresponding dual problem is in turn given by

$$\begin{aligned}
\underset{\boldsymbol{y}\in\mathbb{R}^M,\, \boldsymbol{z}\in\mathbb{R}^N}{\text{maximize}} \quad & \varphi(\boldsymbol{y},\boldsymbol{z}), && \text{(2.2a)}\\
\text{subject to} \quad & \boldsymbol{y} \geq \boldsymbol{0}_M, && \text{(2.2b)}
\end{aligned}$$

where $\mathbf{0}_M$ is the null vector of size M, and the dual function φ is defined as

$$\varphi(\boldsymbol{y}, \boldsymbol{z}) = \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \;\; \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}).$$

If we denote the optimal objective value of (2.1) with f^* and the optimal objective value of (2.2) with φ^*, then $f^* \geq \varphi^*$ holds. We say that the duality gap is $f^* - \varphi^*$. If the duality gap is zero, that is, $f^* = \varphi^*$, then strong duality holds between the primal problem and the dual problem. For convex problems, there exist several different conditions that, if they hold, imply strong duality. One such condition is that all constraints are affine.

Theorem 2 ([4, Theorem 6.12])**.** *Consider* (2.1)*. If f is bounded from below, there exists at least one feasible solution, and all $g_1, \ldots, g_M$ are affine, then strong duality holds between* (2.1) *and* (2.2)

Proof. See [4, Theorem 6.12]. □

2.1.2 Iterative Optimization Algorithms

In practice, problems in convex optimization are often solved with iterative algorithms. In this section, we briefly introduce the basic ideas behind some of them, and specifically, we explain the coordinate descent (or ascent) method. Again, we refer to [4] for a more detailed review.

A key concept in many optimization algorithms is a so-called descent direction at a given point. Intuitively, a descent direction for a function f at a point $\boldsymbol{x}$ is a direction in which f decrease. Formally, we say that $\boldsymbol{p} \in \mathbb{R}^n$ is a descent direction for a function $f \colon \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x} \in \mathbb{R}^n$ if there exists a $\delta > 0$ such that $f(\boldsymbol{x} + \alpha \boldsymbol{p}) < f(\boldsymbol{x})$ for every $\alpha \in (0, \delta]$. A common basis for several iterative optimization algorithms is to identify a descent direction at a given point, and then move a certain distance along this direction as to minimize f, and then repeat. How far to move is decided via a so-called line search, to which we return later. What usually differs between different iterative optimization algorithms is how they identify a descent direction. The most basic algorithm is the steepest descent method, where the steepest descent direction is identified. A descent direction $\boldsymbol{p}$ is said to be a steepest descent direction if it solves the minimization problem

$$\underset{\boldsymbol{p} \in \mathbb{R}^n \colon \|\boldsymbol{p}\| = 1}{\text{minimize}} \; \nabla f(\boldsymbol{x})^\top \boldsymbol{p}. \tag{2.3}$$

The solution to (2.3) is $\boldsymbol{p} = -\nabla f(\boldsymbol{x}) / \|\nabla f(\boldsymbol{x})\|$. If we drop the condition $\|\boldsymbol{p}\| = 1$, the steepest descent method can use $\boldsymbol{p} = -\nabla f(\boldsymbol{x})$ as descent direction. Another common algorithm is Newton's method. It is based on the second order approximation

$$f(\boldsymbol{x} + \boldsymbol{p}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^\top \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^\top \nabla^2 f(\boldsymbol{x}) \boldsymbol{p}.$$

Here, we set the derivative of this expression with respect to $\boldsymbol{p}$ to zero. If the Hessian is positive definite, then that is the same as solving the problem of minimizing the approximation with respect to $\boldsymbol{p}$. The solution is obtained when

$$\nabla^2 f(\boldsymbol{x})\boldsymbol{p} = -\nabla f(\boldsymbol{x}), \tag{2.4}$$

is solved, and so in Newton's method, we must solve (2.4) for each iteration.

We now return to the line search problem. A line search given a descent direction $\boldsymbol{p}$ at a point $\boldsymbol{x}$ is conducted by solving the minimization problem

$$\underset{\alpha>0}{\text{minimize}} \quad f(\boldsymbol{x} + \alpha\boldsymbol{p}). \tag{2.5}$$

In practice, (2.5) can be difficult to solve analytically. Therefore, one usually considers approximate line search strategies. Such strategies include for example interpolation, as well as the usage of second order approximations as in Newton's method.

Another iterative algorithm is the coordinate descent method. In the coordinate descent method, the problem of minimizing a function depending of several coordinates, or sets of variables, is solved by fixing all coordinates but one, and then minimizing the function with respect to the coordinate that is not fixed. Next, the function is minimized with respect to another coordinate, while the rest of the coordinates are fixed. We keep cycling through the coordinates this way until convergence is reached.

To illustrate this more formally, consider the problem

$$\underset{\boldsymbol{x}\in S_x, \boldsymbol{y}\in S_y}{\text{minimize}} \quad g(\boldsymbol{x}, \boldsymbol{y}), \tag{2.6}$$

where g depends on two sets of variables, or coordinates, namely $\boldsymbol{x} \in S_x$ and $\boldsymbol{y} \in S_y$. For a fixed $\hat{\boldsymbol{y}} \in S_y$, one can consider the problem

$$\underset{\boldsymbol{x}\in S_x}{\text{minimize}} \quad g(\boldsymbol{x}, \hat{\boldsymbol{y}}). \tag{2.7}$$

This problem is in general easier to solve than (2.6). When the optimal solution $\hat{\boldsymbol{x}}$ to (2.7) is found, the problem

$$\underset{\boldsymbol{y}\in S_y}{\text{minimize}} \quad g(\hat{\boldsymbol{x}}, \boldsymbol{y}),$$

is solved, and then this procedure is repeated until convergence. If the objective instead is to maximize a function, this method is called the coordinate ascent method.

2.2 Optimal Transport

Optimal transport is a field of mathematical optimization that concerns the optimal transportation and allocation of resources. In general, the objective of an optimal transport problem is to move mass between given distributions as efficiently as possible with respect to some given metric. Introduced in [5], it originally mostly concerned

problems with an obvious connection to traditional transportation, although it has proven useful to model problems without a clear such connection, see for instance [6, 7]. In the following sections, we give an introduction to optimal transport. In Section 2.2.1, the basic bimarginal optimal transport problem is introduced. This is extended to the multimarginal case in Section 2.2.2. A discrete formulation of such problems is presented in Section 2.2.3. Lastly, in Section 2.2.4, we explain how these problems can be modified slightly to enable efficient solution methods.

2.2.1 Bimarginal Optimal Transport

In the most basic case of optimal transport, the only constraints are on an initial distribution and a final distribution, respectively. A constraint on a distribution is usually referred to as a marginal, and so this particular case of optimal transport is sometimes called bimarginal optimal transport. A common formulation of such a problem was introduced in [8]. This formulation is stated in more modern terms in [9], and here we provide a similar formulation.

Let X and Y be two spaces and let $\mathcal{P}(X)$ and $\mathcal{P}(Y)$ be the sets of probability measures on the respective space. Additionally, let $\mu \in \mathcal{P}(X)$ and $\nu \in \mathcal{P}(Y)$ be two given marginals. Consider a measure $q \in \mathcal{P}(X \times Y)$, and denote by $\mathcal{Q}(\mu, \nu)$ the set of all such measures that satisfies $q(A \times X) = \mu(A)$ and $q(B \times Y) = \nu(B)$ for all measurable sets $A \subseteq X$ and $B \subseteq Y$. We say that q is a transport plan and $\mathcal{Q}(\mu, \nu)$ the set of transport plans between the marginal μ and the marginal ν. The bimarginal optimal transport problem can then be written as

$$\underset{q \in \mathcal{Q}(\mu,\nu)}{\text{minimize}} \int_{X \times Y} c(x, y) \, \mathrm{d}q(x, y), \tag{2.8}$$

where $c\colon X \times Y \to \mathbb{R}$ is the cost function that together with the marginals specifies the problem.

2.2.2 Multimarginal Optimal Transport

Optimal transport can be generalized to problems regarding optimal transportation between several distributions, or marginals. Hence, this generalization of bimarginal optimal transport is called multimarginal optimal transport, see [10]. Given a number of marginals $m \in \mathbb{N}$, we consider the product space $X_1 \times \cdots \times X_m$. Let $\mathcal{Q}(\mu_1, \ldots, \mu_m)$ denote the set of all non-negative measures q on $X_1 \times \cdots \times X_m$, where $\mu_1 \in \mathcal{P}(X_1), \ldots, \mu_m \in \mathcal{P}(X_m)$ are its marginals. Here, $\mathcal{P}(X_i)$ denotes the set of probability measures on X_i, for some $i = 1, \ldots, m$. We say that q is a transport plan and $\mathcal{Q}(\mu_1, \ldots, \mu_m)$ the set of transport plans between the marginals $\mu_1, \ldots, \mu_m$. The multimarginal optimal transport problem, for m marginals, can then be written as

$$\underset{q \in \mathcal{Q}(\mu_1,\ldots,\mu_m)}{\text{minimize}} \int_{X_1 \times \cdots \times X_m} c(x_1, \ldots, x_m) \, \mathrm{d}q(x_1, \ldots, x_m), \tag{2.9}$$

where $c\colon X_1 \times \cdots \times X_m \to \mathbb{R}$ is the cost function. Clearly, when $m = 2$, (2.9) is equivalent to (2.8).

2.2.3 Discrete Optimal Transport

A continuous formulation of optimal transport, like (2.9), is theoretically useful, but not very practical for developing numerical solution methods. In practice, optimal transport problems are often discretized. We here present a common way of discretizing optimal transport problems, inspired by [11]. For a problem with $m \in \mathbb{N}$ marginals, these are discretized as vectors $\mu_\ell \in \mathbb{R}_+^{N_\ell}$, for all $\ell = 1, \dots, m$, where N_ℓ is the number of discretized positions for marginal ℓ. We now have two tensors that are elements of $\mathcal{R}_+ = \mathbb{R}_+^{N_1 \times \cdots \times N_m}$, namely the cost tensor $C \in \mathcal{R}_+$, and the transport plan tensor $M \in \mathcal{R}_+$. A cost element $C_{i_1,\dots,i_m}$ has the interpretation as the cost of moving a unit mass along the path $i_1, \dots, i_m$, and a transport plan element $M_{i_1,\dots,i_m}$ has the interpretation as the amount of mass transported along the path $i_1, \dots, i_m$. Here, $i_\ell \in \{1, \dots, N_\ell\}$ for all $\ell \in \{1, \dots, m\}$. The marginal distributions are given by projections of the transport plan, $P_\ell(M) \in \mathbb{R}_+^{N_\ell}$, defined as

$$P_\ell(M)_{i_\ell} = \sum_{i_1,\dots,i_{\ell-1},i_{\ell+1},\dots,i_m} M_{i_1,\dots,i_m}.$$

Lastly, we define the tensor scalar product between the two tensors C and M as

$$\langle C, M \rangle = \sum_{i_1,\dots,i_m} C_{i_1,\dots,i_m} M_{i_1,\dots,i_m}.$$

The discrete multimarginal optimal transport problem can thus be written compactly as

$$\begin{aligned} \underset{M \in \mathcal{R}_+}{\text{minimize}} \quad & \langle C, M \rangle, \\ \text{subject to} \quad & P_\ell(M) = \mu_\ell, \quad \ell = 1, \dots, m. \end{aligned}$$

While a problem like this technically is a linear programming problem, we note that the number of elements in the cost tensor and transport plan grows exponentially with the number of marginals, meaning that the problem is computationally intractable even for modestly sized m.

2.2.4 Entropic Regularization and Sinkhorn Iterations

Traditionally, optimal transport problems have been solved with linear programming. Although efficient for smaller problems, this approach tends to be computationally demanding for larger problems. One way to tackle this is presented in [3]. There, the problem is perturbed by adding an entropic regularization term to the objective function, leading to effective approximation algorithms. This has later been extended to multimarginal optimal transport problems in [12, 13, 14].

The idea is to perturb the objective function in order to smooth the problem, allowing for iterative methods that are significantly faster than the linear programming approach for larger problems. With a sufficiently small perturbation, the modified problem becomes an adequate approximation of the original problem. The exact definition of the entropy term can vary, and is typically adjusted to the problem

formulation in order to gain as much computational alleviation as possible, without making the approximation too poor. However, it commonly involves an $x\log(x)$ term, for a decision variable x. The impact of the entropy term is usually controlled via a regularization parameter.

The iterative method that such an entropic regularization enables is commonly referred to as Sinkhorn iterations. The name originates from Sinkhorn's theorem, first formulated in [15], which regards matrix scaling. This theory can be connected to regularized optimal transport problems. This is for instance illustrated in [16], where an example of how the Sinkhorn iterations can be derived for a specific problem is given. We now derive an iterative scheme for a problem with a structure similar to that of an optimal transport problem with entropic regularization. Given some matrix $A \in \mathbb{R}^{m\times n}$, consider the problem of minimizing a function $f : \mathbb{R}^m_+ \times \mathbb{R}^n_+ \to \mathbb{R}$ defined as

$$f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top A\boldsymbol{y} + \varepsilon \sum_{i=1}^{m} (x_i \log(x_i) - x_i) + \varepsilon \sum_{j=1}^{n} (y_j \log(y_j) - y_j), \tag{2.10}$$

where $x_1, \ldots, x_m$ are the elements of $\boldsymbol{x} \in \mathbb{R}^m_+$ and $y_1, \ldots, y_n$ are the elements of $\boldsymbol{y} \in \mathbb{R}^n_+$. Here, $\varepsilon > 0$. This function is not convex on $\mathbb{R}^m_+ \times \mathbb{R}^n_+$. However, for a fixed $\boldsymbol{y} \in \mathbb{R}^n_+$, it is convex in $\boldsymbol{x}$ on $\mathbb{R}^m_+$. Conversely, for a fixed $\boldsymbol{x} \in \mathbb{R}^m_+$, it is convex in $\boldsymbol{y}$ on $\mathbb{R}^n_+$. Utilizing this, we create a coordinate descent method where we alternatively minimize f with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. First of all, we fix some $\boldsymbol{y}^{(0)} \in \mathbb{R}^n_+$, and consider the problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^m_+}{\text{minimize}} \quad f(\boldsymbol{x}, \boldsymbol{y}^{(0)}). \tag{2.11}$$

The gradient of f with respect to $\boldsymbol{x}$ is

$$(\nabla_{\boldsymbol{x}} f)_i = \left(A\boldsymbol{y}^{(0)}\right)_i + \varepsilon \log(x_i), \quad i = 1, \ldots, m.$$

Setting it to zero, we obtain

$$x_i^{(1)} = \exp\left(-\frac{(A\boldsymbol{y}^{(0)})_i}{\varepsilon}\right), \quad i = 1, \ldots, m.$$

By convexity and Theorem 1, $\boldsymbol{x}^{(1)}$ is a global minimizer to (2.11). We then consider the problem

$$\underset{\boldsymbol{y}\in\mathbb{R}^n_+}{\text{minimize}} \quad f(\boldsymbol{x}^{(1)}, \boldsymbol{y}). \tag{2.12}$$

With analogous reasoning, a global minimizer $\boldsymbol{y}^{(1)}$ to (2.12) is given by

$$y_j^{(1)} = \exp\left(-\frac{((\boldsymbol{x}^{(1)})^\top A)_j}{\varepsilon}\right), \quad j = 1, \ldots, n.$$

By continuing this process, we arrive at a coordinate descent method with the alternating update rules

$$x_i^{(k+1)} \leftarrow \exp\left(-\frac{(A\boldsymbol{y}^{(k)})_i}{\varepsilon}\right), \qquad i = 1, \dots, m,$$
$$y_j^{(k+1)} \leftarrow \exp\left(-\frac{((\boldsymbol{x}^{(k+1)})^\top A)_j}{\varepsilon}\right), \quad j = 1, \dots, n,$$

where k is the iteration index. These are the Sinkhorn iterations, here used in the context of minimizing (2.10). We emphasize that the connection to optimal transport is via the linear term in (2.10), whereas the logarithmic terms correspond to an entropic regularization. In particular, ε corresponds to a regularization parameter.

Hence, by the addition of a cleverly chosen regularization term, we modify the objective function in order for it to obtain properties that enables coordinate descent with exact line search. As shown in [3], this technique can solve multimarginal optimal transport problems several orders of magnitude faster than previously known methods. This is the foundation of the approach used in this thesis.

3

Multiple Target Tracking and Evaluation of Tracking Algorithms

In this chapter, we introduce the concept of multiple target tracking and the mathematical framework that accompanies it. We then briefly discuss the different types of tracking errors a multiple target tracking algorithm can make. Lastly, with this background, we introduce a metric for the evaluation of such tracking algorithms. This chapter is based on the content in [1, 2, 17, 18], to which the interested reader is referred for more details.

3.1 The Multiple Target Tracking Problem

Given one or several sensors placed in the physical world, target tracking is the process of identifying and following the movements of an object, based on the raw detections made by the sensors. A sensor could for example be a radar, a lidar, or a camera. In short, a tracking system works by taking the signal from a sensor, processing it, and returning an estimate for the state of the tracked object. We refer to the component of the system that takes the raw signal and returns the estimated state as the tracking algorithm.

The state can be anything of interest, usually position and velocity relative to the sensor. Typically, the actual state of the real world object is referred to as the ground truth, and the estimated state of the object is referred to as the estimate. Depending on the quality of the tracking system, the state of the object can be detected and tracked with varying accuracy.

In multiple target tracking, the setting is the same, except for the fact that we are interested in how the states of several objects in this scene evolve through time. Compared to single target tracking, this is a more difficult problem. Since the detections are unlabeled, they have to be associated with specific targets. For multiple target tracking, the terms ground truth and estimate correspond to groups of targets or trajectories.

Formally, we model the ground truth and the estimate as sets of trajectories. Math-

ematically, we define a trajectory X as a pair $X = (t_b, x^{1:\ell})$. Here, $t_b \in \mathbb{N}$ is the time index representing the birth of the trajectory and $x^{1:\ell} = (x^1, \dots, x^\ell)$ a sequence of target states $x^1, \dots, x^\ell \in \mathbb{R}^N$ over $\ell \in \mathbb{N}$ time steps. Throughout this work, we consider only discrete time. Additionally, we limit ourselves to consider trajectories on time indices $\{1, \dots, T\} \subset \mathbb{N}$. That is, trajectories such that

$$(t_b, \ell) \in \left\{(t_b', \ell') \in \mathbb{N}^2 \,:\, 1 \leq t_b' \leq T \text{ and } 1 \leq \ell' \leq T - t_b' + 1\right\}.$$

Let $\boldsymbol{X}$ be a set of such trajectories, and let $\mathcal{T}$ denote the set of all such sets of trajectories. A more precise description of the multiple target tracking problem is therefore to construct an estimate $\boldsymbol{Y} \in \mathcal{T}$, given noisy measurements of a ground truth $\boldsymbol{X} \in \mathcal{T}$, such that the "distance" between $\boldsymbol{X}$ and $\boldsymbol{Y}$ is as small as possible.

In this thesis, we focus on the meaning of the term "distance" in the description above. To objectively quantify how similar an estimate produced by some tracking algorithm is to the ground truth, we want a metric $d\colon \mathcal{T} \times \mathcal{T} \to \mathbb{R}_+$ on the sets of trajectories. The design of such a metric is an active area of research. In [2], such a metric based on the GOSPA metric from [1] is proposed. In Section 3.3 we introduce the GOSPA metric, and in Section 3.4 we introduce the metric on the sets of trajectories from [2]. To better understand these metrics, we first give an overview of the types of errors a tracking algorithm can produce.

3.2 Types of Tracking Errors

The purpose of this section is to give the reader a sense of some of the aspects an evaluation metric for multiple target tracking needs to take into consideration. For a more detailed review, we refer to [2, 18].

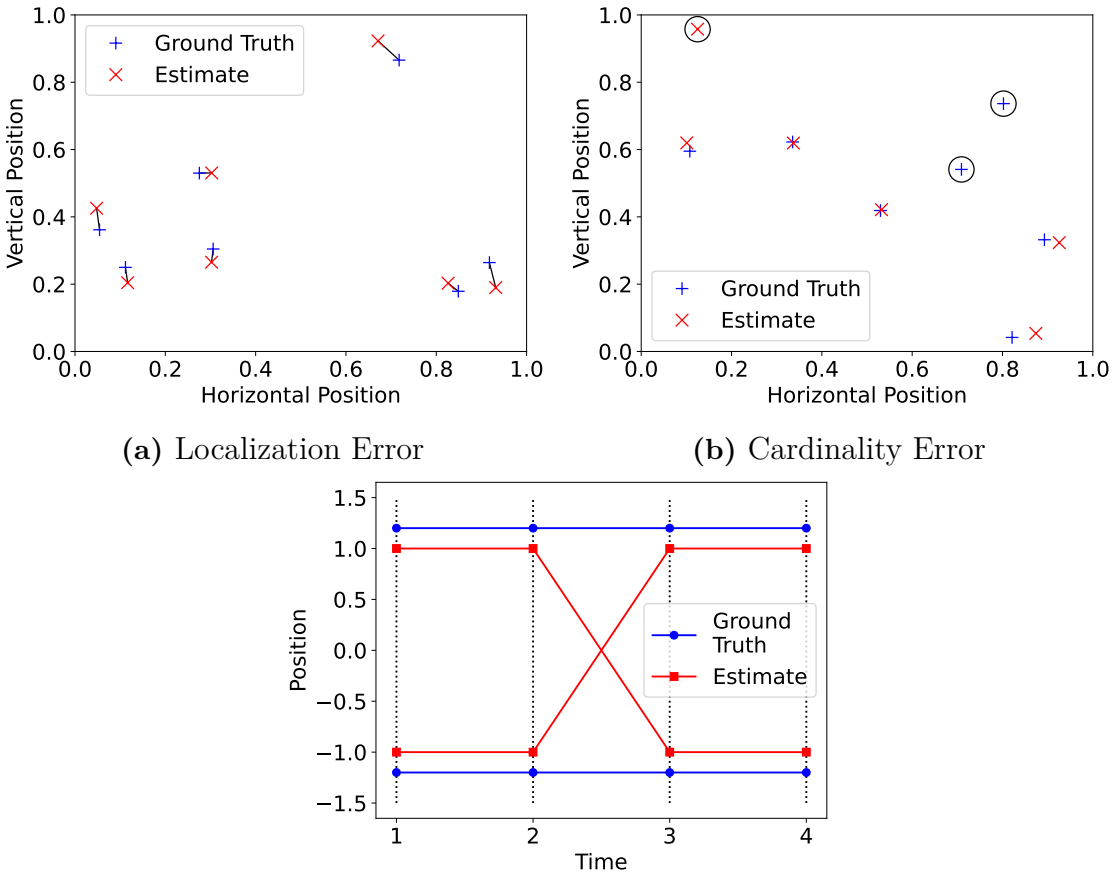
The overall error of an estimate produced by a multiple target tracking algorithm can be subdivided into three different types. These are localization errors, cardinality errors, and track-switching errors. Out of these, the first two appear within a single tracking time step, whereas the last one appears between two consecutive time steps.

We refer to the distance between the estimated target state and the corresponding ground truth state as the localization error. Intuitively, when this error is large, it means that the tracking algorithm has correctly deduced that a target exists, but estimates its state poorly. Figure 3.1 (a) illustrates this type of error.

It is also possible that a tracking algorithm misses to identify a target from the ground truth or gives an estimated target that does not exist in the ground truth, a so-called false target. Collectively, we refer to these types of errors as cardinality errors, since it is usually quantified by considering the cardinality of the set of ground truth targets and the set of estimated targets within one time step. Figure 3.1 (b) illustrates this type of error.

Lastly, if a tracking algorithm erroneously switches the identities of two trajectories, we refer to this type of error as a track-switching error. This error is more subtle

than the previous two, and is best explained by an example. Consider the simple case of two stationary ground truth targets. If the tracking algorithm gives an estimate without localization or cardinality error, but still decides erroneously that that the targets swapped places between two consecutive time steps, it has made a track-switching error. For an illustration of this example using one-dimensional states, see Figure 3.1 (c).



(a) Localization Error

(b) Cardinality Error

(c) Track-Switching Error

Figure 3.1: Illustration of the different types of tracking errors. In figure (a), the localization error is shown by black lines, where the length of the line indicates the magnitude of the error. In figure (b), there are two missed targets and one false target, indicated by black circles. In figure (c), a one-dimensional tracking example with two stationary ground truth targets is shown. The estimate indicates that the targets have swapped places between time step two and three, while this is not the case in the ground truth. This is a track-switching error.

3.3 The GOSPA Metric

Before introducing the extended GOSPA metric on the sets of trajectories, we introduce the GOSPA metric on the sets of targets. We remind the the reader that GOSPA stands for Generalized Optimal Sub-Pattern Assignment. This can be

viewed as a metric on the sets of trajectories where $T = 1$.

Let $d_{\mathrm{b}} \colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_+$ denote a base metric. We then define its cut-off metric as

$$d_c(x, y) = \min(d_{\mathrm{b}}(x, y), c),$$

for some cut-off parameter $c > 0$. Now, let $0 < \alpha \leq 2$, and $1 \leq p < \infty$. Furthermore, let Π_n be the set of all permutations of $\{1, \ldots, n\}$ for $n \in \mathbb{N}$, and let any element $\pi \in \Pi_n$ be a vector $(\pi_1, \ldots, \pi_n)$. Lastly, let $\boldsymbol{x} = \{x_1, \ldots, x_m\}$ and $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ be finite subsets of $\mathbb{R}^N$ corresponding to the ground truth and the estimate, respectively. Then, for $m \leq n$, the GOSPA metric is defined by

$$\tilde{d}_p^{(c,\alpha)}(\boldsymbol{x}, \boldsymbol{y}) = \underset{\pi \in \Pi_n}{\text{minimize}} \left(\sum_{i=1}^{m} d_c(x_i, y_{\pi(i)})^p + \frac{c^p}{\alpha}(n - m) \right)^{1/p}. \tag{3.1}$$

In the case $m > n$, it is defined by $\tilde{d}_p^{(c,\alpha)}(\boldsymbol{x}, \boldsymbol{y}) = \tilde{d}_p^{(c,\alpha)}(\boldsymbol{y}, \boldsymbol{x})$. It can be shown that this indeed is a metric, see [1]. The parameter p determines to what extent outliers are penalized, and α determines the weight for the cardinality mismatch penalty. Additionally, we note that the minimization problem in (3.1) is an assignment problem, and as such it can be solved in polynomial time using the Hungarian algorithm, see [19, 20, 21].

To understand why the metric is formulated as a minimization problem, it helps to compare it to the Euclidean metric. The Euclidean distance between two points $x, y \in \mathbb{R}^N$ can be formulated as the problem of minimizing the length of a curve $u \colon [0, 1] \to \mathbb{R}^N$, subject to $u(0) = x$ and $u(1) = y$. This is analogous to the minimization problem in (3.1). The solution to the question of the shortest curve between x and y has a closed form solution in the form of a straight line connecting x and y. Comparatively, the minimization problem in (3.1) is more difficult, and requires non-trivial algorithms for efficient computation.

The minimization in (3.1) can also be viewed as giving the tracking algorithm the benefit of the doubt, in the sense that we try to find the best assignment of the estimated targets to the ground truth targets without worrying about the intent of the tracking algorithm. For example, consider a case with two ground truth targets and one estimated target, where the estimated target is far from one of the ground truth targets and close to the other one. In this case the metric pairs the estimate with the closest ground truth, even though this estimate might have been the result of a large localization error in estimating the far away ground truth.

It can be argued that the choice of $\alpha = 2$ is the most appropriate one, as this results in the cost of a single missed or false target being the same, whether or not it is associated with another target in the permutation in (3.1). Another reason for setting $\alpha = 2$ is that it allows for a formulation of GOSPA as a minimization problem over so-called assignment sets. For a more detailed motivation, see [1]. An assignment set

θ between the sets $\{1,\dots,m\}$ and $\{1,\dots,n\}$ is a set with the properties

$$\begin{aligned} &\theta \subseteq \{1,\dots,m\} \times \{1,\dots,n\}, \\ &(i,j),(i,j') \in \theta \implies j = j', \text{ and} \\ &(i,j),(i',j) \in \theta \implies i = i'. \end{aligned}$$

The last two properties make sure that each element in one set is assigned to at most one element from the other set. Let Θ denote the set of all assignment sets on the form above. Now, the following proposition can be formulated.

Proposition 1 ([1, Proposition 1])**.** *For $\alpha = 2$, the GOSPA metric can be formulated as an optimization problem over assignment sets. That is*

$$\tilde{d}_p^{(c,2)}(\boldsymbol{x},\boldsymbol{y}) = \underset{\theta\in\Theta}{\text{minimize}} \left(\sum_{(i,j)\in\theta} d_{\text{b}}(x_i,y_j)^p + \frac{c^p}{2}(m+n-2|\theta|) \right)^{1/p}. \tag{3.2}$$

Proof. See [1, Proposition 1]. □

Note the use of d_{b} instead of d_c compared to (3.1). This formulation has a natural interpretation. The sum in the minimization in (3.2) is simply the sum of all localization errors between the ground truth and the estimate. The second term is the cardinality error, where $m+n-2|\theta| = (m-|\theta|)+(n-|\theta|)$ is the sum of the number of missed and false targets, and $c^p/2$ is the penalty for each such error. Note that the GOSPA metric cannot be formulated like this for $\alpha \neq 2$, see [1].

3.4 The T-GOSPA Metric

We now turn to the extended GOSPA metric between sets of trajectories $\boldsymbol{X} = \{X_1,\dots,X_m\} \in \mathcal{T}$ and $\boldsymbol{Y} = \{Y_1,\dots,Y_n\} \in \mathcal{T}$. Since this metric is based on the GOSPA metric, we refer to it as the T-GOSPA metric, for Trajectory-GOSPA. In T-GOSPA, we make use of GOSPA for sets of targets of size at most one. For this special case, the GOSPA metric can be written as

$$\tilde{d}_p^{(c,2)}(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} d_c(x,y), & \boldsymbol{x} = \{x\}, \boldsymbol{y} = \{y\}, \\ 0, & \boldsymbol{x} = \boldsymbol{y} = \varnothing, \\ c/2^{1/p} & \text{else}, \end{cases} \tag{3.3}$$

with the corresponding optimal assignment in (3.2) being

$$\theta^* = \begin{cases} \{(1,1)\}, & |\boldsymbol{x}| = |\boldsymbol{y}| = 1,\ \tilde{d}_p^{(c,2)}(\boldsymbol{x},\boldsymbol{y}) < c, \\ \varnothing, & \text{else}. \end{cases}$$

To simplify the introduction of the T-GOSPA metric, we introduce some additional notation. For a trajectory $X = (t_b, x^{1:\ell})$, we define its state at time step t by

$$S^t(X) = \begin{cases} \{x^{t+1-t_b}\}, & t_b \le t \le t_b + \ell - 1, \\ \varnothing, & \text{else}. \end{cases}$$

If the trajectory is alive at t, this is the singleton set consisting of the state of trajectory X at time index t. Else, it is the empty set. This notation is useful since it allows us to reference the state of a trajectory at a certain time, even if it does not exist. In the context of sets of trajectories $\boldsymbol{X} = \{X_1, \dots, X_m\}$ and $\boldsymbol{Y} = \{Y_1, \dots, Y_n\}$, we also use the shorthand notation $\boldsymbol{x}_i^t = S^t(X_i)$ and $\boldsymbol{y}_i^t = S^t(Y_i)$. Abusing notation slightly, we additionally define

$$S^t(\boldsymbol{X}) = \bigcup_{X \in \boldsymbol{X}} S^t(X),$$

for a set of trajectories $\boldsymbol{X} \in \mathcal{T}$. This is the set of states of all trajectories alive at time step t.

In T-GOSPA, each ground truth trajectory can either be assigned to an estimated trajectory or be left unassigned in each time step. To facilitate this, we use assignment vectors between the sets $\{1, \dots, m\}$ and $\{0, \dots, n\}$. The interpretation of an assignment vector π^t in time step t is

$$\pi_i^t = \begin{cases} j, & \boldsymbol{x}_i^t \text{ is assigned to } \boldsymbol{y}_j^t, \\ 0, & \boldsymbol{x}_i^t \text{ is unassigned.} \end{cases}$$

We let $\Pi_{m,n}$ denote the set of all such assignment vectors. Now we are ready to define the T-GOSPA metric. For some track-switching penalty $\gamma > 0$ and previously introduced parameters, the T-GOSPA metric is defined by

$$d_p^{(c,\gamma)}(\boldsymbol{X}, \boldsymbol{Y}) = \underset{\substack{\pi^t \in \Pi_{m,n} \\ t=1,\dots,T}}{\text{minimize}} \left(\sum_{t=1}^{T} d^t(\pi^t; \boldsymbol{X}, \boldsymbol{Y})^p + \sum_{t=1}^{T-1} s(\pi^t, \pi^{t+1})^p \right)^{1/p}. \tag{3.4}$$

We now explain the terms d^t and s. Firstly, we have

$$d^t(\pi^t; \boldsymbol{X}, \boldsymbol{Y}) = \left(\sum_{(i,j) \in \theta^t(\pi^t)} \tilde{d}_p^{(c,2)}(\boldsymbol{x}_i^t, \boldsymbol{y}_j^t)^p + \frac{c^p}{2}(|S^t(\boldsymbol{X})| + |S^t(\boldsymbol{Y})| - 2|\theta^t(\pi^t)|) \right)^{1/p}, \tag{3.5}$$

with

$$\theta^t(\pi^t) = \left\{ (i, \pi_i^t) \,:\, i = 1, \dots, m,\ |\boldsymbol{x}_i^t| = |\boldsymbol{y}_{\pi_i^t}^t| = 1,\ \tilde{d}_p^{(c,2)}(\boldsymbol{x}_i^t, \boldsymbol{y}_{\pi_i^t}^t) < c \right\}.$$

The quantity in (3.5) is the part of the metric that penalizes localization errors and cardinality errors. These types of errors have in common that they are calculated within one time step. This is highlighted by the fact that (3.5) only takes the assignment vector of a time step t, and that its corresponding sum in (3.4) ranges from 1 to T. Note that (3.5) is simply the objective function of the GOSPA metric used with (3.3). As such, the interpretations of the terms are the same, with the sum corresponding to localization errors and the following term corresponding to cardinality errors. For convenience, the assignment set in (3.4) is used to get only the proper assignments from the assignment vector π^t.

Secondly, we have

$$s(\pi^t, \pi^{t+1}) = \left(\gamma^p \sum_{i=1}^{m} \tilde{s}(\pi_i^t, \pi_i^{t+1}) \right)^{1/p}, \tag{3.6}$$

with

$$\tilde{s}(\pi_i^t, \pi_i^{t+1}) = \begin{cases} 0, & \pi_i^t = \pi_i^{t+1}, \\ 1, & \pi_i^t \neq \pi_i^{t+1}, \pi_i^t \neq 0, \pi_i^{t+1} \neq 0, \\ 1/2, & \text{else.} \end{cases} \tag{3.7}$$

The expression in (3.6) penalizes track-switching. The track-switching errors occur between time steps. As such, it takes the assignment vectors of two consecutive time steps t and $t+1$, and its corresponding sum in (3.4) ranges from 1 to $T-1$. This term is furthermore broken down on an index level in (3.7), where a penalty of γ^p is given if the assignment of a ground truth trajectory changes from one estimated trajectory to another, and a penalty of $\gamma^p/2$ is given if a ground truth trajectory goes from being assigned to an estimated trajectory to being unassigned or vice versa. This special case is called a half-switch, and is an important part of the metric. In short, it assures that the metric is symmetric. This definition of T-GOSPA is introduced along with a proof of that it is a metric in [2], to which we refer the interested reader for additional details.

3.4.1 Formulation Using Assignment Matrices

In this section, we show that the T-GOSPA metric can be formulated as a so-called mixed integer linear programming problem. A mixed integer linear programming problem is simply a linear programming problem where some of the variables have integrality constraints. Such a formulation is useful for several reasons, one of them being that it unlocks the use of many computational techniques developed for such problems. Specifically for this thesis, this formulation also acts as the starting point for the derivations of new algorithms for computing T-GOSPA in Chapter 4.

The key concept in a formulation based on mixed integer linear programming is the use of so-called assignment matrices. The idea is to, for each time step, create a binary matrix W^t, where each element $W_{i,j}^t$ represents an assignment between ground truth trajectory i and estimated trajectory j at time step t. If $W_{i,j}^t = 1$, then ground truth trajectory i and estimated trajectory j are assigned to each other at time step t. If $W_{i,j}^t = 0$, they are not. We now formalize this idea. To this end, we introduce the set of assignment matrices, $\mathcal{W}_{m,n}$, as the set of $(m+1) \times (n+1)$ matrices W^t fulfilling

$$W_{i,j}^t \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \tag{3.8a}$$

$$\sum_{i=1}^{m+1} W_{i,j}^t = 1, \quad j = 1, \ldots, n, \tag{3.8b}$$

$$\sum_{j=1}^{n+1} W_{i,j}^t = 1, \quad i = 1, \ldots, m, \tag{3.8c}$$

$$W_{m+1,n+1}^t = 0. \tag{3.8d}$$

Here, (3.8a) implies that assignments are binary, meaning that there are no in-between states for the assignments. The constraints in (3.8b) imply that each ground truth trajectory is assigned to exactly one estimated trajectory, or unassigned, at each time step. In other words, a ground truth trajectory cannot be assigned to more than one estimated trajectory. Note that the last row is used for representing unassigned trajectories. Analogously, (3.8c) implies the corresponding property among the estimated trajectories, where the last column is used for representing unassigned trajectories. The constraint (3.8d) is of less importance, and just says that the bottom-right corner should be set to zero.

Importantly, we have a bijection between $\Pi_{m,n}$ and $\mathcal{W}_{m,n}$. More precisely, this bijection is given by

$$\begin{aligned} \pi_i^t = j \neq 0 &\iff W_{i,j}^t = 1, \\ \pi_j^t = 0 &\iff W_{i,n+1}^t = 1, \\ \nexists\, i \in \{1,\dots,m\} : \pi_i^t = j \neq 0 &\iff W_{m+1,j}^t = 1, \end{aligned}$$

for $\pi^t \in \Pi_{m,n}$ and $W^t \in \mathcal{W}_{m,n}$. This bijection is understood most easily by an example.

Example 1. *Let $m = 4$ and $n = 5$. For the assignment vector*

$$\pi^t = (2, 0, 5, 0),$$

the corresponding assignment matrix is

$$W^t = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Since $\pi_1^t = 2$, this means that trajectory 1 from the ground truth is assigned to trajectory 2 from the estimate, resulting in $W_{1,2}^t = 1$. Further, $\pi_2^t = 0$ means that trajectory 2 from the ground truth is unassigned, resulting in $W_{2,6}^t = 1$. From $\pi_3^t = 5$, we see that trajectory 3 from the ground truth is assigned to trajectory 5 from the estimate, giving $W_{3,5}^t = 1$. Finally, $\pi_4^t = 0$, implying that trajectory 4, from the ground truth is unassigned, hence $W_{4,6}^t = 1$. Note also that trajectories 1, 3 and 4 from the estimate are unassigned, explaining the last row.

To simplify the notation, let $D_{i,j}^t = \tilde{d}_p^{(c,2)}(\boldsymbol{x}_i^t, \boldsymbol{y}_j^t)^p$, for $i = 1,\dots,m+1$, $j = 1,\dots,n+1$, $t = 1,\dots,T$, where $\boldsymbol{x}_{m+1}^t = \varnothing$ and $\boldsymbol{y}_{n+1}^t = \varnothing$. Then, using assignment matrices, we can formulate the T-GOSPA metric as a mixed integer linear programming problem.

Proposition 2 ([2, Lemma 1])**.** *The T-GOSPA metric, defined in* (3.4)*, can be*

written as

$$d_p^{(c,\gamma)}(\boldsymbol{X},\boldsymbol{Y}) = \underset{\substack{W^t\in\mathcal{W}_{m,n},\\ t=1,\dots,T}}{\text{minimize}} \left(\sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}\left|W_{i,j}^{t+1} - W_{i,j}^t\right| \right)^{1/p}. \tag{3.9}$$

Proof. See [2, Lemma 1]. □

Like before, the first sum is the cardinality and localization error of the metric, while the second sum is the track-switching penalty. Note the indices that the sums range over.

Each change in how a trajectory is assigned is penalized with $\gamma^p/2$. Notice that if two ground truth trajectories and two estimated trajectories that are assigned to each other switch assignments between themselves, this gives a total penalty of $2\gamma^p$ (meaning the penalty is γ^p per switch), as this means that four matrix elements in this sum are changed. If instead a trajectory goes from being unassigned to being assigned or vice versa, only a single element of the top-left $m \times n$ matrix is changed, giving a penalty of $\gamma^p/2$. This corresponds to a half-switch.

The minimization in (3.9) is a combinatorial optimization problem, and as such it can be difficult to compute. To obtain a more computationally tractable problem, we relax the binary constraints. Let $\overline{\mathcal{W}}_{m,n}$ be the set of matrices in $\mathcal{W}_{m,n}$ but with the constraints (3.8a) replaced with $W_{i,j} \in [0,1]$. This results in

$$\bar{d}_p^{(c,\gamma)}(\boldsymbol{X},\boldsymbol{Y}) = \underset{\substack{W^t\in\overline{\mathcal{W}}_{m,n},\\ t=1,\dots,T}}{\text{minimize}} \left(\sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}\left|W_{i,j}^{t+1} - W_{i,j}^t\right| \right)^{1/p}. \tag{3.10}$$

It can be shown that this relaxation still yields a metric.

Proposition 3 ([2, Proposition 2])**.** *The relaxed T-GOSPA metric $\bar{d}_p^{(c,\gamma)}(\boldsymbol{X},\boldsymbol{Y})$, as defined above, is a metric itself.*

Proof. See [2, Proposition 2]. □

For some types of mixed integer linear programming problems, the linear programming relaxation can be such that is has the same optimal solution as the original problem. This is for example true for minimum cost flow problems with integer constraint parameters, see [22, Theorem 8.2]. For such problems, we can solve the linear programming relaxation, which is often easier to solve, in order to obtain a solution to the mixed integer linear programming problem. Unfortunately, this is not the case here.

Proposition 4. *There exist instances when $\bar{d}_p^{(c,\gamma)}(\boldsymbol{X},\boldsymbol{Y}) < d_p^{(c,\gamma)}(\boldsymbol{X},\boldsymbol{Y})$.*

Proof. Consider the case where the states are one-dimensional, the parameters have the values $\gamma = 1$, $c = 2$, and $p = 1$, the number of time steps is $T = 2$, and the ground truth is given by $\boldsymbol{X} = \{X_1, X_2, X_3\}$, where

$$X_1 = \left(1, \left(\frac{5}{2}, 0\right)\right), \quad X_2 = \left(1, \left(\frac{1}{2}, \frac{1}{2}\right)\right), \quad \text{and} \quad X_3 = \left(1, \left(4, \frac{9}{2}\right)\right),$$

whereas the estimate is given by $\boldsymbol{Y} = \{Y_1, Y_2\}$, where

$$Y_1 = \left(1, \left(\frac{9}{2}, \frac{3}{2}\right)\right), \quad \text{and} \quad Y_2 = \left(1, \left(4, \frac{1}{2}\right)\right).$$

That is, the number of ground truth trajectories is $m = 3$ and the number of estimated trajectories is $n = 2$, and all trajectories are alive at all time steps. For this case, the optimal solution for the ordinary T-GOSPA metric is given by

$$W^1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \text{and} \quad W^2 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

with the optimal objective value $d_p^{(c,\gamma)}(\boldsymbol{X}, \boldsymbol{Y}) = d_1^{(2,1)}(\boldsymbol{X}, \boldsymbol{Y}) = \frac{13}{2} = 6 + \frac{1}{2}$, whereas the optimal solution to the relaxed T-GOSPA metric is given by

$$W^1_{\mathrm{LP}} = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad \text{and} \quad W^2_{\mathrm{LP}} = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix},$$

with the optimal objective value $\bar{d}_p^{(c,\gamma)}(\boldsymbol{X}, \boldsymbol{Y}) = \bar{d}_1^{(2,1)}(\boldsymbol{X}, \boldsymbol{Y}) = \frac{25}{4} = 6 + \frac{1}{4}$. Hence, this instance gives $d_p^{(c,\gamma)}(\boldsymbol{X}, \boldsymbol{Y}) < \bar{d}_p^{(c,\gamma)}(\boldsymbol{X}, \boldsymbol{Y})$. □

Specifically, this proves that [23, Conjecture 2.3.1] is false.

Since $(\cdot)^{1/p}$ is an increasing function on $\mathbb{R}_+$ for $p > 0$, the problem of interest for computing the T-GOSPA metric is

$$\underset{\substack{W^t \in \mathcal{W}_{m,n}, \\ t=1,\dots,T}}{\text{minimize}} \quad \sum_{t=1}^{T} \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2} \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} \left| W_{i,j}^{t+1} - W_{i,j}^t \right|.$$

Similarly, for the relaxed T-GOSPA metric, it is

$$\underset{\substack{W^t \in \overline{\mathcal{W}}_{m,n}, \\ t=1,\dots,T}}{\text{minimize}} \quad \sum_{t=1}^{T} \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2} \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} \left| W_{i,j}^{t+1} - W_{i,j}^t \right|. \tag{3.11}$$

In this thesis, we consider (3.11). Specifically, we investigate how approximate solutions can be obtained efficiently using similar techniques to those presented in Section 2.2.4.

4

Novel Algorithms for Approximation of the T-GOSPA Metric

In this chapter, we derive and present two algorithms for efficiently finding approximate solutions to (3.11). The algorithms are derived from two different reformulations of (3.11). The first algorithm is based on a reformulation to a structured tensor optimization problem. The second algorithm is based on a reformulation to a multi-marginal optimal transport problem. Both derivations follow a similar procedure. We refer to the algorithms as Algorithm 1 and 2, respectively.

4.1 Derivation of Algorithm 1

Here, (3.11) is reformulated as a tensor structured optimization problem. Then a regularization term is introduced, after which the Lagrangian dual problem is formulated. For the dual problem, the optimal solution in each variable individually is analytically tractable. This gives rise to an efficient coordinate ascent scheme. The derivation presented in this section is similar to the one conducted in [11], but it can also be found elsewhere.

4.1.1 Formulation as a Linear Programming Problem

The first step of the derivation of Algorithm 1 is to formulate (3.11) as a linear program. Additionally, this serves the purpose of making the relaxed version of T-GOSPA accessible to standard linear programming solvers. To this end, we start by introducing help variables

$$H^t = \left|W^{t+1} - W^t\right| \in \mathbb{R}^{m\times n}, \quad t = 1, \dots, T-1.$$

Here, the absolute value operator is applied element-wise. Using these variables, (3.11) can be formulated as

$$\underset{\substack{W^t \in \overline{\mathcal{W}}_{m,n},\, t=1,\dots,T \\ H^t \in \mathbb{R}^{m \times n},\, t=1,\dots,T-1}}{\text{minimize}} \quad \sum_{t=1}^{T} \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2} \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} H_{i,j}^t, \tag{4.1a}$$

$$\text{subject to} \quad H^t = \left| W^{t+1} - W^t \right|, \quad t = 1, \dots, T-1. \tag{4.1b}$$

Since the optimization problem is to minimize the cost, the constraints (4.1b) can be replaced by the constraints

$$H^t \geq W^{t+1} - W^t, \quad t = 1, \dots, T-1,$$
$$H^t \geq W^t - W^{t+1}, \quad t = 1, \dots, T-1.$$

As a result, we end up with

$$\underset{\substack{W^t \in \overline{\mathcal{W}}_{m,n},\, t=1,\dots,T \\ H^t \in \mathbb{R}^{m \times n},\, t=1,\dots,T-1}}{\text{minimize}} \quad \sum_{t=1}^{T} \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2} \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} H_{i,j}^t, \tag{4.2a}$$

$$\text{subject to} \quad H^t \geq W^{t+1} - W^t, \quad t = 1, \dots, T-1, \tag{4.2b}$$

$$H^t \geq W^t - W^{t+1}, \quad t = 1, \dots, T-1. \tag{4.2c}$$

Note that the objective function (4.2a) and constraints (4.2b) and (4.2c) are comprised of linear functions with respect to the decision variables. This means that (4.2) is a linear program with the same objective value as (3.11).

4.1.2 Formulation as a Structured Tensor Optimization Problem

For the derivation of Algorithm 1, the next step is to write (4.2) as an optimization problem over a high-dimensional tensor with a certain structure. Therefore, let

$$\mathcal{M}_+ = \mathbb{R}_+^{(m+1)^T \times (n+1)^T}$$

be the set of all non-negative tensors with $2T$ indices such that the first T indices have dimension $m+1$, and the following T indices have dimension $n+1$. Furthermore, let $\mu^X \in \mathbb{R}^{m+1}$ and $\mu^Y \in \mathbb{R}^{n+1}$ be marginals defined by

$$\mu^X = \left(\mathbf{1}_m^\top, n \right)^\top, \tag{4.3a}$$

$$\mu^Y = \left(\mathbf{1}_n^\top, m \right)^\top, \tag{4.3b}$$

where $\mathbf{1}_m \in \mathbb{R}^m$ and $\mathbf{1}_n \in \mathbb{R}^n$ are vectors with all elements equal to one. Hence,

$$\sum_{i=1}^{m+1} \mu_i^X = \sum_{j=1}^{n+1} \mu_j^Y = m + n,$$

meaning that we have the same total mass for all marginals. Now, for $i_1, \dots, i_T \in \{1, \dots, m+1\}$ and $i_{T+1}, \dots, i_{2T} \in \{1, \dots, n+1\}$, we define a cost tensor $C \in \mathcal{M}_+$ by

$$C_{i_1,\dots,i_{2T}} = \sum_{t=1}^{T} D^t_{i_t, i_{t+T}}.$$

Following Section 2.2.3, we define the marginal projection of a tensor $M \in \mathcal{M}_+$ onto marginal t for $t = 1, \dots, 2T$ as

$$P_t(M)_{i_t} = \sum_{i_1,\dots,i_{t-1},i_{t+1},\dots,i_{2T}} M_{i_1,\dots,i_{2T}},$$

for appropriate indices i_t. Additionally, for $t = 1, \dots, T$, we define the bimarginal projection of M onto marginals t and $t+T$ by

$$P_{t,t+T}(M)_{i_t,i_{t+T}} = \sum_{i_1,\dots,i_{t-1},i_{t+1},\dots,i_{t+T-1},i_{t+T+1},\dots,i_{2T}} M_{i_1,\dots,i_{2T}},$$

for $i_t = 1, \dots, m+1$ and $i_{t+T} = 1, \dots, n+1$.

Adding the tensor $M \in \mathcal{M}_+$ as a variable to (4.2) and adding constraints

$$W^t = P_{t,t+T}(M), \quad t = 1, \dots, T,$$

yields

$$\begin{aligned}
\underset{\substack{W^t \in \overline{\mathcal{W}}_{m,n},\, t=1,\dots,T,\\ H^t \in \mathbb{R}^{m\times n},\, t=1,\dots,T-1,\\ M \in \mathcal{M}_+}}{\text{minimize}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D^t_{i,j} W^t_{i,j} + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n} H^t_{i,j}, && \text{(4.4a)}\\
\text{subject to} \quad & H^t \geq W^{t+1} - W^t, \quad t = 1, \dots, T-1, && \text{(4.4b)}\\
& H^t \geq W^t - W^{t+1}, \quad t = 1, \dots, T-1, && \text{(4.4c)}\\
& W^t = P_{t,t+T}(M), \quad t = 1, \dots, T. && \text{(4.4d)}
\end{aligned}$$

Here, (4.4) has the same optimal value as (4.2). To see this, we first consider minimizing over all variables except M. Since M is not present in the objective (4.4a), we can simply choose any M such that (4.4d) is satisfied to obtain an optimal solution to (4.4). For more details, see [11]. Substituting W^t for $P_{t,t+T}(M)$ everywhere in (4.4) using (4.4d) yields

$$\begin{aligned}
\underset{\substack{W^t \in \mathbb{R}_+^{(m+1)\times(n+1)},\\ t=1,\dots,T,\\ H^t \in \mathbb{R}^{m\times n},\\ t=1,\dots,T-1,\\ M \in \mathcal{M}_+}}{\text{minimize}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D^t_{i,j} P_{t,t+T}(M)_{i,j} + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n} H^t_{i,j}, && \text{(4.5a)}\\
\text{subject to} \quad & P_t(M) = \mu^X, \quad && t = 1, \dots, T, && \text{(4.5b)}\\
& P_{t+T}(M) = \mu^Y, \quad && t = 1, \dots, T, && \text{(4.5c)}\\
& H^t \geq P_{t+1,t+1+T}(M) - P_{t,t+T}(M), \quad && t = 1, \dots, T-1, && \text{(4.5d)}\\
& H^t \geq P_{t,t+T}(M) - P_{t+1,t+1+T}(M), \quad && t = 1, \dots, T-1, && \text{(4.5e)}\\
& W^t = P_{t,t+T}(M), \quad && t = 1, \dots, T. && \text{(4.5f)}
\end{aligned}$$

The constraints (4.5b) and (4.5c) come from the constraints (3.8b) and (3.8c) of $\overline{\mathcal{W}}_{m,n}$, respectively. Furthermore, we drop the condition (3.8d) of $\overline{\mathcal{W}}_{m,n}$ as $D^t_{m+1,n+1} = 0$. By adding constraints (4.5b) and (4.5c), we have also included the constraints

$$\sum_{j=1}^{n+1} W^t_{m+1,j} = \sum_{j=1}^{n+1} P_{t,t+T}(M)_{m+1,j} = P_t(M)_{m+1} = n, \qquad t = 1, \dots, T,$$
$$\sum_{i=1}^{m+1} W^t_{i,n+1} = \sum_{i=1}^{m+1} P_{t,t+T}(M)_{i,n+1} = P_{t+T}(M)_{n+1} = m, \quad t = 1, \dots, T,$$

on the last row and column of the assignment matrices. Now, since the bottom-right corner element of the assignment matrices has coefficient zero in the objective function, and since we allow this corner element to take any positive value, these constraints do not impose any actual constraints on the problem, meaning that the optimal objective value is unaffected. This might look cumbersome, but this formulation is convenient for reasons apparent in the coming sections.

Note now that constraints (4.5f) can be removed, as we can first optimize over M, and then find the corresponding W^t, $t = 1, \dots, T$. Therefore, these variables can be completely removed from the problem. Finally, since

$$\begin{aligned}
&\sum_{t=1}^{T} \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} D^t_{i,j} P_{t,t+T}(M)_{i,j} \\
&\quad = \sum_{t=1}^{T} \sum_{i_t=1}^{m+1} \sum_{i_{t+T}=1}^{n+1} D^t_{i_t,i_{t+T}} \sum_{i_1,\dots,i_{t-1},i_{t+1},\dots,i_{t+T-1},i_{t+T+1},\dots,i_{2T}} M_{i_1,\dots,i_{2T}} \\
&\quad = \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T} D^t_{i_t,i_{t+T}} M_{i_1,\dots,i_{2T}} \\
&\quad = \sum_{i_1,\dots,i_{2T}} C_{i_1,\dots,i_{2T}} M_{i_1,\dots,i_{2T}} \\
&\quad = \langle C, M \rangle,
\end{aligned}$$

we arrive at

$$\underset{\substack{H^t \in \mathbb{R}^{m \times n}, \\ t=1,\dots,T-1, \\ M \in \mathcal{M}_+}}{\text{minimize}} \quad \langle C, M \rangle + \frac{\gamma^p}{2} \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} H^t_{i,j}, \tag{4.6a}$$

$$\text{subject to} \quad P_t(M) = \mu^X, \qquad t = 1, \dots, T, \tag{4.6b}$$
$$P_{t+T}(M) = \mu^Y, \qquad t = 1, \dots, T, \tag{4.6c}$$
$$H^t \geq P_{t+1,t+1+T}(M) - P_{t,t+T}(M), \quad t = 1, \dots, T-1, \tag{4.6d}$$
$$H^t \geq P_{t,t+T}(M) - P_{t+1,t+1+T}(M), \quad t = 1, \dots, T-1. \tag{4.6e}$$

This is the formulation of T-GOSPA that is the base for the rest of the derivation of Algorithm 1 in the coming sections.

4.1.3 Entropic Regularization

In order to apply the technique described in Section 2.2.4, we need to regularize (4.6) by adding an entropy term $\varepsilon E(M)$ to the objective function. Similarly to [11], we

define such an entropy term by

$$E(M) = \sum_{i_1,\dots,i_{2T}} \left(M_{i_1,\dots,i_{2T}} \log(M_{i_1,\dots,i_{2T}}) - M_{i_1,\dots,i_{2T}} + 1\right),$$

with the regularization parameter $\varepsilon > 0$. The regularized problem therefore becomes

$$\begin{aligned}
\underset{\substack{H^t \in \mathbb{R}^{m\times n},\\ t=1,\dots,T-1,\\ M\in\mathcal{M}_+}}{\text{minimize}} \quad & \langle C, M\rangle + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n} H_{i,j}^t + \varepsilon E(M), && && \text{(4.7a)}\\
\text{subject to} \quad & P_t(M) = \mu^X, && t = 1,\dots,T, && \text{(4.7b)}\\
& P_{t+T}(M) = \mu^Y, && t = 1,\dots,T, && \text{(4.7c)}\\
& H^t \geq P_{t+1,t+1+T}(M) - P_{t,t+T}(M), && t = 1,\dots,T-1, && \text{(4.7d)}\\
& H^t \geq P_{t,t+T}(M) - P_{t+1,t+1+T}(M), && t = 1,\dots,T-1. && \text{(4.7e)}
\end{aligned}$$

It is important to keep in mind that we now have changed the objective function. Therefore, (4.7) possibly has a different optimal objective value compared to (4.6). As explained in Section 2.2.4, we want to keep ε small so that this perturbation is sufficiently small to make (4.7) an adequate approximation of (4.6).

4.1.4 Derivation of the Dual Problem

We now derive a Lagrangian of (4.7) as well as the corresponding dual problem. Relaxing constraints (4.7b) and (4.7c) with dual variables $\lambda_t^X \in \mathbb{R}^{m+1}$ and $\lambda_t^Y \in \mathbb{R}^{n+1}$, respectively, for $t = 1,\dots,T$, and constraint (4.7d) and (4.7e) with dual variables $\Lambda_t \in \mathbb{R}_+^{m\times n}$ and $\tilde{\Lambda}_t \in \mathbb{R}_+^{m\times n}$, respectively, for $t = 1,\dots,T-1$, we obtain a Lagrangian $\mathcal{L}$. For ease of presentation, we split it into three terms according to

$$\mathcal{L}(M, H, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) = L(M, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) + \tilde{L}(H, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) + \hat{L}(\lambda^X, \lambda^Y).$$

In the above, and throughout the rest of this section, we refer to sets of variables with their symbol without subscripts. For example, λ^X stands for $(\lambda_t^X)_i$, for $t = 1,\dots,T$, $i = 1,\dots,m+1$.

Here, L is defined as

$$\begin{aligned}
&L(M, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda})\\
&\quad = \langle C, M\rangle + \varepsilon E(M) - \sum_{t=1}^{T}\sum_{i=1}^{m+1}(\lambda_t^X)_i P_t(M)_i - \sum_{t=1}^{T}\sum_{j=1}^{n+1}(\lambda_t^Y)_j P_{t+T}(M)_j\\
&\qquad + \sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}(\Lambda_t)_{i,j}(P_{t,t+T}(M) - P_{t+1,t+1+T}(M))_{i,j}\\
&\qquad + \sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}(\tilde{\Lambda}_t)_{i,j}(P_{t+1,t+1+T}(M) - P_{t,t+T}(M))_{i,j},
\end{aligned}$$

and contains all terms in $\mathcal{L}$ that depend on M. $\tilde{L}$ is in turn defined as

$$\begin{aligned}&\tilde{L}(H, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda})\\ &\quad= \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n} H_{i,j}^t - \sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}(\Lambda_t)_{i,j}H_{i,j}^t - \sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}(\tilde{\Lambda}_t)_{i,j}H_{i,j}^t,\end{aligned}$$

and contains all terms in $\mathcal{L}$ that depend on H. Lastly, $\hat{L}$ is defined as

$$\hat{L}(\lambda^X, \lambda^Y) = \sum_{t=1}^{T}\sum_{i=1}^{m+1}(\lambda_t^X)_i\mu_i^X + \sum_{t=1}^{T}\sum_{j=1}^{n+1}(\lambda_t^Y)_j\mu_j^Y,$$

and contains all terms that are independent of M and H. Note specifically that L is independent of H, $\tilde{L}$ is independent of M, and $\hat{L}$ is independent of M and H. Hence, for the dual problem

$$\underset{\substack{\lambda_t^X\in\mathbb{R}^{m+1},\,\lambda_t^Y\in\mathbb{R}^{n+1}\\ t=1,\dots,T\\ \Lambda_t,\tilde{\Lambda}_t\in\mathbb{R}_+^{m\times n}\\ t=1,\dots,T-1}}{\text{maximize}}\ \phi(\lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}), \tag{4.8}$$

with the dual function

$$\phi(\lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) = \underset{\substack{H^t\in\mathbb{R}^{m\times n},\\ t=1,\dots,T-1,\\ M\in\mathcal{M}_+}}{\text{minimize}}\ \mathcal{L}(M, H, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}),$$

we can decompose the inner minimization of $\mathcal{L}$ as

$$\begin{aligned}&\underset{\substack{H^t\in\mathbb{R}^{m\times n},\\ t=1,\dots,T-1,\\ M\in\mathcal{M}_+}}{\text{minimize}}\ \mathcal{L}(M, H, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda})\\ &\quad= \underset{M\in\mathcal{M}_+}{\text{minimize}}\ L(M, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) + \underset{H^t, t=1,\dots,T-1}{\text{minimize}}\ \tilde{L}(H,\ \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) + \hat{L}(\lambda^X, \lambda^Y).\end{aligned}$$

We start by minimizing L. Note that for $t = 1, \dots, T$, we have

$$\sum_{i=1}^{m+1}(\lambda_t^X)_i P_t(M)_i = \sum_{i_t=1}^{m+1}(\lambda_t^X)_{i_t}\sum_{i_1,\dots,i_{2T}\setminus i_t} M_{i_1,\dots,i_{2T}} = \sum_{i_1,\dots,i_{2T}}(\lambda_t^X)_{i_t}M_{i_1,\dots,i_{2T}},$$

with a similar expression possible for the case of λ^Y. Further, we have

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j=1}^{n}(\Lambda_t)_{i,j}P_{t,t+T}(M)_{i,j} &= \sum_{i_t=1}^{m}\sum_{i_{t+T}=1}^{n}(\Lambda_t)_{i_t,i_{t+T}}\sum_{i_1,\dots,i_{2T}\setminus i_t,i_{t+T}} M_{i_1,\dots,i_{2T}}\\ &= \sum_{i_1,\dots,i_{2T}}(\Lambda_t)_{i_t,i_{t+T}}M_{i_1,\dots,i_{2T}},\end{aligned}$$

with a similar expression possible for the case of $\tilde{\Lambda}$. Using these expressions, L can

be rewritten as

$$\begin{aligned}
&L(M, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) \\
&\quad = \langle C, M \rangle + \varepsilon E(M) \\
&\qquad - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T} (\lambda_t^X)_{i_t} M_{i_1,\dots,i_{2T}} - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T} (\lambda_t^Y)_{i_{t+T}} M_{i_1,\dots,i_{2T}} \\
&\qquad + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\Lambda_t)_{i_t, i_{t+T}} M_{i_1,\dots,i_{2T}} - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\Lambda_t)_{i_{t+1}, i_{t+1+T}} M_{i_1,\dots,i_{2T}} \\
&\qquad + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_{t+1}, i_{t+1+T}} M_{i_1,\dots,i_{2T}} - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_t, i_{t+T}} M_{i_1,\dots,i_{2T}}.
\end{aligned}$$

Next, we consider the partial derivative of E with respect to $M_{i_1,\dots,i_{2T}}$ for $i_1, \dots, i_T \in \{1, \dots, m+1\}$ and $i_{T+1}, \dots, i_{2T} \in \{1, \dots, n+1\}$. It is given by

$$\begin{aligned}
\frac{\partial E}{\partial M_{i_1,\dots,i_{2T}}} &= \frac{\partial}{\partial M_{i_1,\dots,i_{2T}}} \left(\sum_{i_1,\dots,i_{2T}} M_{i_1,\dots,i_{2T}} \log(M_{i_1,\dots,i_{2T}}) - M_{i_1,\dots,i_{2T}} + 1 \right) \\
&= \log(M_{i_1,\dots,i_{2T}}).
\end{aligned}$$

This means that the partial derivative of L with respect to $M_{i_1,\dots,i_{2T}}$ is

$$\begin{aligned}
\frac{\partial L}{\partial M_{i_1,\dots,i_{2T}}} &= C_{i_1,\dots,i_{2T}} + \varepsilon \log(M_{i_1,\dots,i_{2T}}) - \sum_{t=1}^{T} (\lambda_t^X)_{i_t} - \sum_{t=1}^{T} (\lambda_t^Y)_{i_{t+T}} \\
&\quad + \sum_{t=1}^{T-1} (\Lambda_t)_{i_t, i_{t+T}} - \sum_{t=1}^{T-1} (\Lambda_t)_{i_{t+1}, i_{t+1+T}} + \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_{t+1}, i_{t+1+T}} - \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_t, i_{t+T}}.
\end{aligned}$$

In order to find the minimizer

$$\operatorname*{arg\,min}_{M \in \mathcal{M}_+} L(M, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}),$$

we set

$$\frac{\partial L}{\partial M_{i_1,\dots,i_{2T}}} = 0,$$

and solve for $M_{i_1,\dots,i_{2T}}$. The solution to this equation is

$$\begin{aligned}
M^*_{i_1,\dots,i_{2T}} &= \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon} \right) \\
&\quad \cdots \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon} \right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon} \right) \\
&\quad \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\Lambda_t)_{i_t, i_{t+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(\frac{(\Lambda_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon} \right) \\
&\quad \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\tilde{\Lambda}_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(\frac{(\tilde{\Lambda}_t)_{i_t, i_{t+T}}}{\varepsilon} \right).
\end{aligned} \tag{4.9}$$

Note now that $M^* \geq 0$. By the convexity of L with respect to M together with Theorem 1, we conclude that M^* is optimal. We have thus found a minimizer M^* and expressed it in terms of the dual variables.

We now turn to the problem of minimizing $\tilde{L}$. By writing

$$\tilde{L}(H, \Lambda, \tilde{\Lambda}) = \sum_{t=1}^{T-1} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{\gamma^p}{2} - (\Lambda_t)_{i,j} - (\tilde{\Lambda}_t)_{i,j} \right) H_{i,j}^t,$$

it is clear that $\tilde{L}$ is linear in H. With no constraints on H, we have that

$$\frac{\gamma^p}{2} - (\Lambda_t)_{i,j} - (\tilde{\Lambda}_t)_{i,j} = 0, \tag{4.10}$$

for $i = 1, \ldots, m$, $j = 1, \ldots, n$, $t = 1, \ldots, T-1$, is needed for the minimization problem to have a lower bound. This has implications for the dual problem (4.8), as the outer maximization forces Λ and $\tilde{\Lambda}$ to satisfy (4.10), which in turn makes (4.10) a constraint in the dual problem. Thus, the dual problem becomes

$$\underset{\substack{\lambda_t^X \in \mathbb{R}^{m+1},\, \lambda_t^Y \in \mathbb{R}^{n+1} \\ t=1,\ldots,T \\ \Lambda_t, \tilde{\Lambda}_t \in \mathbb{R}_+^{m \times n} \\ t=1,\ldots,T-1}}{\text{maximize}} \quad \phi(\lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}), \tag{4.11a}$$

$$\text{subject to} \quad \Lambda_t + \tilde{\Lambda}_t = \frac{\gamma^p}{2}, \quad t = 1, \ldots, T-1. \tag{4.11b}$$

We now express $L(M^*, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda})$ in terms of the dual variables. Firstly, note that

$$\begin{aligned}
&\log(M^*_{i_1,\ldots,i_{2T}}) \\
&= \log \Bigg(\exp\left(-\frac{C_{i_1,\ldots,i_{2T}}}{\varepsilon} \right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon} \right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon} \right) \\
&\quad \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\Lambda_t)_{i_t, i_{t+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(\frac{(\Lambda_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon} \right) \\
&\quad \cdots \prod_{t=1}^{T-1} \exp\left(\frac{(\tilde{\Lambda}_t)_{i_t, i_{t+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(-\frac{(\tilde{\Lambda}_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon} \right) \Bigg) \\
&= -\frac{C_{i_1,\ldots,i_{2T}}}{\varepsilon} + \sum_{t=1}^{T} \frac{(\lambda_t^X)_{i_t}}{\varepsilon} + \sum_{t=1}^{T} \frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon} - \sum_{t=1}^{T-1} \frac{(\Lambda_t)_{i_t, i_{t+T}}}{\varepsilon} + \sum_{t=1}^{T-1} \frac{(\Lambda_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon} \\
&\quad + \sum_{t=1}^{T-1} \frac{(\tilde{\Lambda}_t)_{i_t, i_{t+T}}}{\varepsilon} - \sum_{t=1}^{T-1} \frac{(\tilde{\Lambda}_t)_{i_{t+1}, i_{t+1+T}}}{\varepsilon}.
\end{aligned}$$

Secondly, we have that

$$
\begin{aligned}
E(M^*) &= \sum_{i_1,\dots,i_{2T}} \Big(M^*_{i_1,\dots,i_{2T}} \log(M^*_{i_1,\dots,i_{2T}}) - M^*_{i_1,\dots,i_{2T}} + 1 \Big) \\
&= \sum_{i_1,\dots,i_{2T}} \Bigg(M^*_{i_1,\dots,i_{2T}} \Bigg(- \frac{C_{i_1,\dots,i_{2T}}}{\varepsilon} + \sum_{t=1}^{T} \frac{(\lambda^X_t)_{i_t}}{\varepsilon} + \sum_{t=1}^{T} \frac{(\lambda^Y_t)_{i_{t+T}}}{\varepsilon} \\
&\quad - \sum_{t=1}^{T-1} \frac{(\Lambda_t)_{i_t,i_{t+T}}}{\varepsilon} + \sum_{t=1}^{T-1} \frac{(\Lambda_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon} \\
&\quad + \sum_{t=1}^{T-1} \frac{(\tilde{\Lambda}_t)_{i_t,i_{t+T}}}{\varepsilon} - \sum_{t=1}^{T-1} \frac{(\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon} \Bigg) - M^*_{i_1,\dots,i_{2T}} + 1 \Bigg) \\
&= \frac{1}{\varepsilon} \Bigg(- \langle C, M^* \rangle + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T} (\lambda^X_t)_{i_t} M^*_{i_1,\dots,i_{2T}} + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T} (\lambda^Y_t)_{i_{t+T}} M^*_{i_1,\dots,i_{2T}} \\
&\quad - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\Lambda_t)_{i_t,i_{t+T}} M^*_{i_1,\dots,i_{2T}} + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\Lambda_t)_{i_{t+1},i_{t+1+T}} M^*_{i_1,\dots,i_{2T}} \\
&\quad + \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_t,i_{t+T}} M^*_{i_1,\dots,i_{2T}} - \sum_{i_1,\dots,i_{2T}} \sum_{t=1}^{T-1} (\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}} M^*_{i_1,\dots,i_{2T}} \\
&\quad - \varepsilon \sum_{i_1,\dots,i_{2T}} M^*_{i_1,\dots,i_{2T}} + \varepsilon (m+1)^T (n+1)^T \Bigg).
\end{aligned}
$$

This means that

$$
\begin{aligned}
&L(M^*, \lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) \\
&\quad = \langle C, M^* \rangle + \varepsilon E(M^*) - \sum_{t=1}^{T} \sum_{i_1,\dots,i_{2T}} (\lambda^X_t)_{i_t} M^*_{i_1,\dots,i_{2T}} - \sum_{t=1}^{T} \sum_{i_1,\dots,i_{2T}} (\lambda^Y_t)_{i_{t+T}} M^*_{i_1,\dots,i_{2T}} \\
&\qquad + \sum_{t=1}^{T-1} \sum_{i_1,\dots,i_{2T}} (\Lambda_t)_{i_t,i_{t+T}} M^*_{i_1,\dots,i_{2T}} - \sum_{t=1}^{T-1} \sum_{i_1,\dots,i_{2T}} (\Lambda_t)_{i_{t+1},i_{t+1+T}} M^*_{i_1,\dots,i_{2T}} \\
&\qquad + \sum_{t=1}^{T-1} \sum_{i_1,\dots,i_{2T}} (\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}} M^*_{i_1,\dots,i_{2T}} - \sum_{t=1}^{T-1} \sum_{i_1,\dots,i_{2T}} (\tilde{\Lambda}_t)_{i_t,i_{t+T}} M^*_{i_1,\dots,i_{2T}} \\
&\quad = -\varepsilon \sum_{i_1,\dots,i_{2T}} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon} \right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda^X_t)_{i_t}}{\varepsilon} \right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda^Y_t)_{i_{t+T}}}{\varepsilon} \right) \\
&\qquad \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\Lambda_t)_{i_t,i_{t+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(\frac{(\Lambda_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon} \right) \\
&\qquad \cdots \prod_{t=1}^{T-1} \exp\left(\frac{(\tilde{\Lambda}_t)_{i_t,i_{t+T}}}{\varepsilon} \right) \prod_{t=1}^{T-1} \exp\left(-\frac{(\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon} \right) + \varepsilon (m+1)^T (n+1)^T.
\end{aligned}
$$

Therefore, the dual function is

$$\begin{aligned}\phi(\lambda^X, \lambda^Y, \Lambda, \tilde{\Lambda}) = -\varepsilon \sum_{i_1,\dots,i_{2T}} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon}\right) \\ \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\Lambda_t)_{i_t,i_{t+T}}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(\frac{(\Lambda_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon}\right) \\ \cdots \prod_{t=1}^{T-1} \exp\left(\frac{(\tilde{\Lambda}_t)_{i_t,i_{t+T}}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(-\frac{(\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon}\right) \\ + \sum_{t=1}^{T} \sum_{i=1}^{m+1} (\lambda_t^X)_i \mu_i^X + \sum_{t=1}^{T} \sum_{j=1}^{n+1} (\lambda_t^Y)_j \mu_j^Y + \varepsilon (m+1)^T (n+1)^T.\end{aligned}$$

Now we use constraint (4.11b) to eliminate one set of variables. We select $\tilde{\Lambda}$, and for convenience, introduce $\Gamma^t = 2\Lambda_t = \gamma^p - 2\tilde{\Lambda}_t$. The constraints $0 \leq \Gamma^t \leq \gamma^p$ for $t = 1, \dots, T-1$ are now needed since $0 \leq \tilde{\Lambda}_t = \frac{\gamma^p}{2} - \Lambda_t$. Using Γ, the dual function can be written as

$$\begin{aligned}\phi(\lambda^X, \lambda^Y, \Gamma) = -\varepsilon \sum_{i_1,\dots,i_{2T}} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon}\right) \\ \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{\Gamma^t_{i_t,i_{t+T}}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(\frac{\Gamma^t_{i_{t+1},i_{t+1+T}}}{\varepsilon}\right) + \varepsilon (m+1)^T (n+1)^T \\ + \sum_{t=1}^{T} \sum_{i=1}^{m+1} (\lambda_t^X)_i \mu_i^X + \sum_{t=1}^{T} \sum_{j=1}^{n+1} (\lambda_t^Y)_j \mu_j^Y,\end{aligned}$$

and the dual problem becomes

$$\underset{\substack{\lambda_t^X \in \mathbb{R}^{m+1},\, \lambda_t^Y \in \mathbb{R}^{n+1} \\ t=1,\dots,T \\ \Gamma_t \in \mathbb{R}^{m \times n} \\ t=1,\dots,T-1}}{\text{maximize}} \quad \phi(\lambda^X, \lambda^Y, \Gamma), \tag{4.12a}$$

$$\text{subject to} \quad 0 \leq \Gamma_t \leq \gamma^p, \quad t = 1, \dots, T-1. \tag{4.12b}$$

To summarize, we note that since all constraints in (4.7) are affine, strong duality holds between (4.7) and (4.12) by Theorem 2. Therefore, the primal and dual problems have the same optimal objective value. This is important, since it is the objective value we are interested in for T-GOSPA. By virtue of (4.9), we even have an expression for the optimal primal variables given the optimal dual variables. In the next section, we look into solving (4.12) numerically using the coordinate ascent method.

4.1.5 Derivation of the Coordinate Ascent Scheme

We now apply the coordinate ascent method in order to solve (4.12). For this purpose, we start by maximizing with respect to λ^X. Consider the element $(\lambda_\tau^X)_{i_\tau}$ for some

$\tau = 1, \dots, T$ and $i_\tau = 1, \dots, m+1$. We have that

$$\begin{aligned}\frac{\partial \phi}{\partial (\lambda_\tau^X)_{i_\tau}} = -\sum_{i_1,\dots,i_{2T}\setminus i_\tau} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon}\right) \\ \cdots \prod_{t=1}^{T-1} \exp\left(-\frac{(\Lambda_t)_{i_t,i_{t+T}}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(\frac{(\Lambda_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon}\right) \\ \cdots \prod_{t=1}^{T-1} \exp\left(\frac{(\tilde{\Lambda}_t)_{i_t,i_{t+T}}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(-\frac{(\tilde{\Lambda}_t)_{i_{t+1},i_{t+1+T}}}{\varepsilon}\right) + \mu_{i_\tau}^X.\end{aligned} \tag{4.13}$$

As (4.12) is a convex optimization problem, and as there are no restrictions on $(\lambda_\tau^X)_{i_\tau}$, we set this derivative to zero and solve for $(\lambda_\tau^X)_{i_\tau}$ in order to find a maximizer. First however, recall the optimal solution M^* to the primal problem in terms of dual variables, given in (4.9). We identify that the first term in (4.13) is in fact the projection of this solution onto marginal τ, and so

$$\frac{\partial \phi}{\partial (\lambda_\tau^Y)_k} = -P_\tau(M^*)_k + \mu_k^X.$$

Now, the dual variables and the cost tensor are transformed by

$$\begin{aligned} u_t^X &= \exp\left(\frac{\lambda_t^X}{\varepsilon}\right), \quad u_t^Y = \exp\left(\frac{\lambda_t^X}{\varepsilon}\right), && t = 1, \dots, T, \\ U^t &= \exp\left(-\frac{\Gamma_t}{\varepsilon}\right), && t = 1, \dots, T-1, \\ K &= \exp\left(-\frac{C}{\varepsilon}\right), \end{aligned}$$

where all operations are element-wise. Next, we note that

$$\begin{aligned} P_\tau(M^*)_{i_\tau} \Big/ (u_\tau^X)_{i_\tau} &= \sum_{i_t,\dots,i_{2T}\setminus i_\tau} M^*_{i_t,\dots,i_{2T}} \Big/ (u_\tau^X)_{i_\tau} \\ &= \sum_{i_t,\dots,i_{2T}\setminus i_\tau} K_{i_t,\dots,i_{2T}} \prod_{\substack{t=1 \\ t\neq\tau}}^{T} (u_\tau^X)_{i_\tau} \prod_{t=1}^{T} (u_\tau^X)_{i_\tau} \prod_{t=1}^{T-1} \frac{U^t_{i_t,i_{t+T}}}{U^t_{i_{t+1},i_{t+1+T}}} \\ &= (w_\tau^X)_{i_\tau}, \end{aligned}$$

where we define a vector w_τ^X that is independent of u_τ^X. This gives

$$\frac{\partial \phi}{\partial (\lambda_\tau^X)_k} = -(w_\tau^X)_k (u_\tau^X)_k + \mu_k^X.$$

Setting the derivative to zero and solving for $(u_\tau^X)_k$ yields

$$(u_\tau^X)_k = \frac{\mu_k^X}{(w_\tau^X)_k},$$

and by multiplying the numerator and denominator by $(u_\tau^X)_k$, we get

$$(u_\tau^X)_k = \frac{\mu_k^X (u_\tau^X)_k}{(w_\tau^X)_k (u_\tau^X)_k} = \frac{\mu_k^X}{P_\tau(M^*)_k} (u_\tau^X)_k.$$

Note that the right hand side is still independent of $(u_\tau^X)_k$. By convexity, this is a maximizer. This results in the update rule

$$(u_\tau^X)_k \leftarrow \frac{\mu_k^X}{P_\tau(M^*)_k}(u_\tau^X)_k,$$

which can be written on vector form as

$$u_\tau^X \leftarrow \mu^X \oslash P_\tau(M^*) \odot u_\tau^X, \quad \text{for } \tau = 1, \dots, T, \tag{4.14}$$

where $\oslash$ and $\odot$ represent element-wise division and multiplication, respectively. With analogous reasoning, we also obtain

$$u_\tau^Y \leftarrow \mu^Y \oslash P_{\tau+T}(M^*) \odot u_\tau^Y, \quad \text{for } \tau = 1, \dots, T. \tag{4.15}$$

Having obtained update rules for the first two sets of dual variables above, we now turn to Γ. We mainly use the same approach for Γ as we did for λ^X and λ^Y above. Taking the partial derivative of the dual function with respect to $\Gamma_{k,\ell}^\tau$, we get

$$\begin{aligned}
\frac{\partial \phi}{\partial \Gamma_{k,\ell}^\tau} = &\sum_{i_1,\dots,i_{2T} \setminus i_\tau = k, i_{\tau+T} = \ell} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon}\right) \\
&\cdots \prod_{t=1}^{T-1} \exp\left(-\frac{\Gamma_{i_t,i_{t+T}}^t}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(\frac{\Gamma_{i_{t+1},i_{t+1+T}}^t}{\varepsilon}\right) \\
&- \sum_{i_1,\dots,i_{2T} \setminus i_{\tau+1} = k, i_{\tau+1+T} = \ell} \exp\left(-\frac{C_{i_1,\dots,i_{2T}}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^X)_{i_t}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{(\lambda_t^Y)_{i_{t+T}}}{\varepsilon}\right) \\
&\cdots \prod_{t=1}^{T-1} \exp\left(-\frac{\Gamma_{i_t,i_{t+T}}^t}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(\frac{\Gamma_{i_{t+1},i_{t+1+T}}^t}{\varepsilon}\right).
\end{aligned}$$

Similar to before, we use the transformed dual variables, so

$$\begin{aligned}
\frac{\partial \phi}{\partial \Gamma_{k,\ell}^\tau} = &\sum_{i_1,\dots,i_{2T} \setminus i_\tau = k, i_{\tau+T} = \ell} K_{i_1,\dots,i_{2T}} \prod_{t=1}^{T} (u_t^X)_{i_t} \prod_{t=1}^{T} (u_t^Y)_{i_{t+T}} \prod_{t=1}^{T-1} \frac{U_{i_{t+1},i_{t+1+T}}^t}{U_{i_t,i_{t+T}}^t} \\
&- \sum_{i_1,\dots,i_{2T} \setminus i_{\tau+1} = k, i_{\tau+1+T} = \ell} K_{i_1,\dots,i_{2T}} \prod_{t=1}^{T} (u_t^X)_{i_t} \prod_{t=1}^{T} (u_t^Y)_{i_{t+T}} \prod_{t=1}^{T-1} \frac{U_{i_{t+1},i_{t+1+T}}^t}{U_{i_t,i_{t+T}}^t}.
\end{aligned}$$

Remembering (4.9) again, we identify the two terms as bimarginal projections of M^*, yielding

$$\frac{\partial \phi}{\partial \Gamma_{k,\ell}^\tau} = P_{\tau,\tau+T}(M^*)_{k,\ell} - P_{\tau+1,\tau+1+T}(M^*)_{k,\ell}.$$

With respect to Γ, (4.12) is a constrained optimization problem of a convex function on a compact set. We therefore investigate the stationary points, as well as the boundary of the feasible set. To this end, we set this derivative to zero and get

$$P_{\tau,\tau+T}(M^*) = P_{\tau+1,\tau+1+T}(M^*). \tag{4.16}$$

In order to find a stationary point, the next step is to solve (4.16) for $U^{\tau}_{k,\ell}$. To do this, we first rewrite the projections. Notice that the transformed cost tensor can be decoupled as

$$K_{i_t,\dots,i_{2T}} = \prod_{t=1}^{T} k^t_{i_t,i_{t+T}},$$

with

$$k^t_{i_t,i_{t+T}} = \exp\left(-\frac{D^t_{i_t,i_{t+T}}}{\varepsilon}\right).$$

The first projection in (4.16) can therefore be written as

$$P_{\tau,\tau+T}(M^*)_{k,\ell} = \sum_{i_1,\dots,i_{2T}\setminus i_\tau=k,i_{\tau+T}=\ell} \prod_{t=1}^{T} k^t_{i_t,i_{t+T}} \prod_{t=1}^{T} (u^X_t)_{i_t} \prod_{t=1}^{T} (u^Y_t)_{i_{t+T}} \prod_{t=1}^{T-1} \frac{U^t_{i_{t+1},i_{t+1+T}}}{U^t_{i_t,i_{t+T}}}.$$

For convenience, we define $a^t_{i_t,i_{t+T}} = k^t_{i_t,i_{t+T}}(u^X_t)_{i_t}(u^Y_t)_{i_{t+T}}$, and write

$$\begin{aligned}
P_{\tau,\tau+T}(M^*)_{k,\ell} &= \sum_{i_1,\dots,i_{2T}\setminus i_\tau=k,i_{\tau+T}=\ell} \prod_{t=1}^{T} a^t_{i_t,i_{t+T}} \prod_{t=1}^{T-1} \frac{U^t_{i_{t+1},i_{t+1+T}}}{U^t_{i_t,i_{t+T}}} \\
&= a^{\tau}_{k,\ell} \frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}} \sum_{i_1,\dots,i_{2T}\setminus i_\tau=k,i_{\tau+T}=\ell} \prod_{\substack{t=1\\t\neq\tau}}^{T} a^t_{i_t,i_{t+T}} \prod_{\substack{t=1\\t\neq\tau}}^{T-1} \frac{1}{U^t_{i_t,i_{t+T}}} \prod_{\substack{t=1\\t\neq\tau-1}}^{T-1} U^t_{i_{t+1},i_{t+1+T}} \\
&= a^{\tau}_{k,\ell} \frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}} \Bigg[\sum_{i_{\tau+1},i_{\tau+1+T}} a^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} \frac{U^{\tau}_{i_{\tau+1},i_{\tau+1+T}}}{U^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}}} \\
&\quad \cdots \sum_{i_1,\dots,i_{2T}\setminus i_\tau=k,i_{\tau+T}=\ell,i_{\tau+1},i_{\tau+1+T}} \Bigg(\prod_{\substack{t=1\\t\notin\{\tau,\tau+1\}}}^{T} a^t_{i_t,i_{t+T}} \\
&\quad \cdots \prod_{\substack{t=1\\t\notin\{\tau,\tau+1\}}}^{T-1} \frac{1}{U^t_{i_t,i_{t+T}}} \prod_{\substack{t=1\\t\notin\{\tau-1,\tau\}}}^{T-1} U^t_{i_{t+1},i_{t+1+T}} \Bigg) \Bigg].
\end{aligned}$$

Next, we define

$$\begin{aligned}
B^{\tau}_{k,\ell,q,r} = \sum_{i_1,\dots,i_{2T}\setminus i_\tau=k,i_{\tau+T}=\ell,i_{\tau+1}=q,i_{\tau+1+T}=r} \Bigg(& \prod_{\substack{t=1\\t\notin\{\tau,\tau+1\}}}^{T} a^t_{i_t,i_{t+T}} \\
& \cdots \prod_{\substack{t=1\\t\notin\{\tau,\tau+1\}}}^{T-1} \frac{1}{U^t_{i_t,i_{t+T}}} \prod_{\substack{t=1\\t\notin\{\tau-1,\tau\}}}^{T-1} U^t_{i_{t+1},i_{t+1+T}} \Bigg),
\end{aligned}$$

after which we write the bimarginal projection as

$$\begin{aligned}
&P_{\tau,\tau+T}(M^*)_{k,\ell} \\
&= a^{\tau}_{k,\ell} \frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}} \left[a^{\tau+1}_{k,\ell} \frac{U^{\tau}_{k,\ell}}{U^{\tau+1}_{k,\ell}} B_{k,\ell,k,\ell} + \sum_{(i_{\tau+1},i_{\tau+1+T})\neq(k,\ell)} a^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} \frac{U^{\tau}_{i_{\tau+1},i_{\tau+1+T}}}{U^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}}} B_{k,\ell,i_{\tau+1},i_{\tau+1+T}} \right].
\end{aligned}$$

Additionally, we define

$$\hat{B}^{\tau}_{k,\ell} = \sum_{(i_{\tau+1},i_{\tau+1+T})\neq(k,\ell)} a^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} \frac{U^{\tau}_{i_{\tau+1},i_{\tau+1+T}}}{U^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}}} B_{k,\ell,i_{\tau+1},i_{\tau+1+T}}$$

meaning that we can write

$$P_{\tau,\tau+T}(M^*)_{k,\ell} = a^{\tau}_{k,\ell}a^{\tau+1}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau+1}_{k,\ell}}B^{\tau}_{k,\ell,k,\ell} + a^{\tau}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}}\hat{B}^{\tau}_{k,\ell}. \tag{4.17}$$

Analogously, by defining

$$\tilde{B}^{\tau}_{k,\ell} = \sum_{(i_{\tau},i_{\tau+T})\neq(k,\ell)} a^{\tau}_{i_{\tau},i_{\tau+T}} \frac{U^{\tau-1}_{i_{\tau},i_{\tau+T}}}{U^{\tau}_{i_{\tau},i_{\tau+T}}} B_{i_{\tau},i_{\tau+T},k,\ell}$$

we get

$$P_{\tau+1,\tau+1+T}(M^*)_{k,\ell} = a^{\tau}_{k,\ell}a^{\tau+1}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau+1}_{k,\ell}}B^{\tau}_{k,\ell,k,\ell} + a^{\tau+1}_{k,\ell}\frac{U^{\tau}_{k,\ell}}{U^{\tau+1}_{k,\ell}}\tilde{B}^{\tau}_{k,\ell}. \tag{4.18}$$

Substituting (4.17) and (4.18) in (4.16) gives

$$a^{\tau}_{k,\ell}a^{\tau+1}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau+1}_{k,\ell}}B^{\tau}_{k,\ell,k,\ell} + a^{\tau}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}}\hat{B}^{\tau}_{k,\ell} = a^{\tau}_{k,\ell}a^{\tau+1}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau+1}_{k,\ell}}B^{\tau}_{k,\ell,k,\ell} + a^{\tau+1}_{k,\ell}\frac{U^{\tau}_{k,\ell}}{U^{\tau+1}_{k,\ell}}\tilde{B}^{\tau}_{k,\ell},$$

which simplifies to

$$a^{\tau}_{k,\ell}\frac{U^{\tau-1}_{k,\ell}}{U^{\tau}_{k,\ell}}\hat{B}^{\tau}_{k,\ell} = a^{\tau+1}_{k,\ell}\frac{U^{\tau}_{k,\ell}}{U^{\tau+1}_{k,\ell}}\tilde{B}^{\tau}_{k,\ell}.$$

Solving for $(U^{\tau}_{k,\ell})^2$, we get

$$(U^{\tau}_{k,\ell})^2 = \frac{a^{\tau}_{k,\ell}U^{\tau-1}_{k,\ell}\hat{B}^{\tau}_{k,\ell}}{a^{\tau+1}_{k,\ell}(1/U^{\tau+1}_{k,\ell})\tilde{B}^{\tau}_{k,\ell}}.$$

Similarly to before, we multiply the numerator and denominator by $U^{\tau}_{k,\ell}$, giving us

$$(U^{\tau}_{k,\ell})^2 = \frac{U^{\tau}_{k,\ell}a^{\tau}_{k,\ell}U^{\tau-1}_{k,\ell}\hat{B}^{\tau}_{k,\ell}}{U^{\tau}_{k,\ell}a^{\tau+1}_{k,\ell}(1/U^{\tau+1}_{k,\ell})\tilde{B}^{\tau}_{k,\ell}} = (U^{\tau}_{k,\ell})^2\frac{a^{\tau}_{k,\ell}(1/U^{\tau}_{k,\ell})U^{\tau-1}_{k,\ell}\hat{B}^{\tau}_{k,\ell}}{a^{\tau+1}_{k,\ell}U^{\tau}_{k,\ell}(1/U^{\tau+1}_{k,\ell})\tilde{B}^{\tau}_{k,\ell}}, \tag{4.19}$$

where we note that the right hand side does not depend on $U^{\tau}_{k,\ell}$.

We now define the quadmarginal projection of M as

$$P_{t,t+T,t+1,t+1+T}(M)_{i_t,i_{t+T},i_{t+1},i_{t+1+T}} = \sum_{i_1,\ldots,i_{2T}\setminus i_t,i_{t+T},i_{t+1},i_{t+1+T}} M_{i_1,\ldots,i_{2T}}.$$

By unraveling the definitions of $a^\tau_{k,\ell}$, $\hat{B}^\tau_{k,\ell}$, and $\tilde{B}^\tau_{k,\ell}$, we identify that the numerator and denominator can be written as differences of projections of M^*, giving us

$$U^\tau_{k,\ell} = U^\tau_{k,\ell}\sqrt{\frac{P_{\tau,\tau+T}(M^*)_{k,\ell} - P_{\tau,\tau+T,\tau+1,\tau+1+T}(M^*)_{k,\ell,k,\ell}}{P_{\tau+1,\tau+1+T}(M^*)_{k,\ell} - P_{\tau,\tau+T,\tau+1,\tau+1+T}(M^*)_{k,\ell,k,\ell}}}, \tag{4.20}$$

where we have also taken the square root. The step from (4.19) to (4.20) above is verified most easily going in the opposite direction.

This is the basis for our update rule. However, we must keep the constraints (4.12b) in mind. In the transformed variables, this implies that

$$1 \leq U^\tau \leq \exp(\gamma^p/\varepsilon), \quad \tau = 1, \dots, T-1, \tag{4.21}$$

and so we obtain the update rule

$$U^\tau_{k,\ell} \leftarrow \operatorname{proj}_{[1,\exp(\gamma^p/\varepsilon)]}\left(U^\tau_{k,\ell}\sqrt{\frac{P_{\tau,\tau+T}(M^*)_{k,\ell} - P_{\tau,\tau+T,\tau+1,\tau+1+T}(M^*)_{k,\ell,k,\ell}}{P_{\tau+1,\tau+1+T}(M^*)_{k,\ell} - P_{\tau,\tau+T,\tau+1,\tau+1+T}(M^*)_{k,\ell,k,\ell}}}\right), \tag{4.22}$$

where the projection guarantees that (4.21) is satisfied. We thereby have update rules for all sets of dual variables. In the next section, we look at the problem of computing the projections needed for the update rules efficiently.

4.1.6 Message Passing

The update rules (4.14), (4.15), and (4.22) all rely on different types of projections of M^*. Since the number of terms in the sums in these projections grow exponentially with T, naive summation is not feasible, even for very modestly sized T. To this end, we exploit the structure of the problem and derive a recursive scheme that leverages previously calculated quantities to reduce the computational expense significantly. This resembles dynamic programming, but more generally it can be thought of as passing messages along edges in a graph implicitly defined by the structure of the cost tensor, hence the title of this section. See [11, 24] for more details.

To start, we define $\tilde{U}^\tau = 1/U^\tau$, $\tau = 1, \dots, T-1$, and write the quadmarginal projection for $\tau = 1, \dots, T-1$ as

$$\begin{aligned}
&P_{\tau,\tau+T,\tau+1,\tau+1+T}(M^*)_{i_\tau,i_{\tau+T},i_{\tau+1},i_{\tau+1+T}} \\
&\quad= \sum_{i_1,\dots,i_{2T}\setminus i_\tau,i_{\tau+T},i_{\tau+1},i_{\tau+1+T}} \prod_{t=1}^{T} k^t_{i_t,i_{t+T}} \prod_{t=1}^{T}(u^X_t)_{i_t} \prod_{t=1}^{T}(u^Y_t)_{i_{t+T}} \prod_{t=1}^{T-1} U^t_{i_t,i_{t+T}} \prod_{t=1}^{T-1} \tilde{U}^t_{i_{t+1},i_{t+1+T}} \\
&\quad= \hat{\Psi}^\tau_{i_\tau,i_{\tau+T}} k^\tau_{i_\tau,i_{\tau+T}} (u^X_\tau)_{i_\tau} (u^Y_\tau)_{i_{\tau+T}} U^\tau_{i_\tau,i_{\tau+T}} \tilde{U}^\tau_{i_{\tau+1},i_{\tau+1+T}} \Psi^\tau_{i_{\tau+1},i_{\tau+1+T}}.
\end{aligned}$$

Here we have introduced matrices $\hat{\Psi}_\tau$ and Ψ_τ. The former is defined as

$$\begin{aligned}
\hat{\Psi}^\tau_{i_\tau,i_{\tau+T}} = \sum_{i_1,\dots,i_{\tau-1}} \sum_{i_{1+T},\dots,i_{\tau-1+T}} &\prod_{t=1}^{\tau-1} k^t_{i_t,i_{t+T}} \prod_{t=1}^{\tau-1}(u^X_t)_{i_t} \prod_{t=1}^{\tau-1}(u^Y_t)_{i_{t+T}} \\
&\cdots \prod_{t=1}^{\tau-1} U^t_{i_t,i_{t+T}} \prod_{t=1}^{\tau-1} \tilde{U}^t_{i_{t+1},i_{t+1+T}}, \quad \text{for } \tau = 2, \dots, T-1,
\end{aligned}$$

with

$$\hat{\Psi}^1_{i_1,i_{1+T}} = 1,$$

while the latter is defined as

$$\begin{aligned}\Psi^\tau_{i_{\tau+1},i_{\tau+1+T}} = \sum_{i_{\tau+2},\dots,i_T} \sum_{i_{\tau+2+T},\dots,i_{T+T}} \prod_{t=\tau+1}^{T} k^t_{i_t,i_{t+T}} \prod_{t=\tau+1}^{T} (u^X_t)_{i_t} \prod_{t=\tau+1}^{T} (u^Y_t)_{i_{t+T}} \\ \cdots \prod_{t=\tau+1}^{T-1} U^t_{i_t,i_{t+T}} \prod_{t=\tau+1}^{T-1} \tilde{U}^t_{i_{t+1},i_{t+1+T}}, \quad \text{for } \tau = 0,\dots,T-2,\end{aligned}$$

with

$$\Psi^{T-1}_{i_T,i_{2T}} = (u^X_T)_{i_T} k^T_{i_T,i_{2T}} (u^Y_T)_{i_{2T}}. \tag{4.23}$$

Note now that we have the recursive relationships

$$\hat{\Psi}^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} = \tilde{U}^\tau_{i_{\tau+1},i_{\tau+1+T}} \sum_{i_\tau,i_{\tau+T}} \hat{\Psi}^\tau_{i_\tau,i_{\tau+T}} k^\tau_{i_\tau,i_{\tau+T}} (u^X_\tau)_{i_\tau} (u^Y_\tau)_{i_{\tau+T}} U^\tau_{i_\tau,i_{\tau+T}}, \tag{4.24}$$

and

$$\begin{aligned}\Psi^\tau_{i_{\tau+1},i_{\tau+1+T}} = k^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} (u^X_{\tau+1})_{i_{\tau+1}} (u^Y_{\tau+1})_{i_{\tau+1+T}} U^{\tau+1}_{i_{\tau+1},i_{\tau+1+T}} \\ \cdots \sum_{i_{\tau+2}} \sum_{i_{\tau+2+T}} \Psi^{\tau+1}_{i_{\tau+2},i_{\tau+2+T}} \tilde{U}^{\tau+1}_{i_{\tau+2},i_{\tau+2+T}}.\end{aligned} \tag{4.25}$$

This means that given $\hat{\Psi}_\tau$, we can compute $\hat{\Psi}_{\tau+1}$, and given $\Psi_{\tau+1}$, we can compute Ψ_τ. Additionally, for the bimarginal projections, we have

$$P_{\tau,\tau+T}(M^*)_{i_\tau,i_{\tau+T}} = (\hat{\Psi}_\tau)_{i_\tau,i_{\tau+T}} (\Psi_{\tau-1})_{i_\tau,i_{\tau+T}}.$$

In all, this means that we now have a way of computing all projections efficiently. Equipped with these tools, we are now ready to write down Algorithm 1 in its entirety.

4.1.7 Pseudocode for Algorithm 1

Cycling through the update rules (4.14), (4.15), and (4.22), we obtain a coordinate ascent algorithm. Moreover, by using the relationships (4.24) and (4.25), we also have a way of computing the necessary projections efficiently. Putting it all together results in Algorithm 1, which is displayed below.

Algorithm 1

Initialize u^X, u^Y, U and $\tilde{U}$ to one everywhere
while Not converged **do**
  Compute Ψ^{T-1} according to (4.23)
  for $t = T-2, \dots, 0$ **do**
    Compute Ψ^t from Ψ^{t+1} using (4.25)
  end for
  $\hat{\Psi}^1 \leftarrow 1$
  for $t = 1, \dots, T-1$ **do**
    $u_t^X \leftarrow u_t^X \odot \mu_t^X \oslash P_t(M^*)$
    Compute Ψ^t from Ψ^{t+1} using (4.25)
    $u_t^Y \leftarrow u_t^Y \odot \mu_t^Y \oslash P_{t+T}(M^*)$
    Compute Ψ^t from Ψ^{t+1} using (4.25)
    Compute $\hat{\Psi}^{t+1}$ from $\hat{\Psi}^t$ using (4.24)
    for $k = 1, \dots, m$, $l = 1, \dots, n$ **do**
      $U_{k,\ell}^t \leftarrow \text{proj}_{[1,\exp(\gamma^p/\varepsilon)]}\left(U_{k,\ell}^t \sqrt{\frac{P_{t,t+T}(M^*)_{k,\ell} - P_{t,t+T,t+1,t+1+T}(M^*)_{k,\ell,k,\ell}}{P_{t+1,t+1+T}(M^*)_{k,\ell} - P_{t,t+T,t+1,t+1+T}(M^*)_{k,\ell,k,\ell}}}\right)$
    end for
    $\tilde{U}^t \leftarrow 1/U^t$
    Compute $\hat{\Psi}^{t+1}$ from $\hat{\Psi}^t$ using (4.24)
  end for
  $u_t^X \leftarrow u_t^X \odot \mu_t^X \oslash P_t(M^*)$
  Compute Ψ^{T-1} according to (4.23)
  $u_t^Y \leftarrow u_t^Y \odot \mu_t^Y \oslash P_{t+T}(M^*)$
end while

Depending on the application, different types of convergence criteria can be used. One common criteria is to stop the algorithm when the change in the variables between iterations is small. This is discussed more thoroughly in Section 5.1.2.

4.1.8 Computations in the Log-Domain

As explained in Section 4.1.3, we want to keep the regularization parameter ε small. However, as ε appears both in the numerator and denominator of fractions in exponentials, a small ε can result in very small or very large values in the computations. In practice, these computations might break down for ε that are too small due to overflows or underflows, which limits the practical lower bound on ε. In order to allow for as small ε as possible, we therefore transform the computations to the log-domain. For a detailed review of this idea, we refer to [25, 26]. Here, we explain it with an illustration. Assume that we have to calculate the sum of exponentials

$$\mathcal{A} = \sum_{i=1}^{m} \exp(a_i),$$

for a series of elements $a_i, \dots, a_m$. Let $\hat{a} = \max(a_1, \dots, a_m)$ denote the maximum element of this series. We can then rewrite the above summation as

$$\mathcal{A} = \sum_{i=1}^{m} \exp(a_i) = \exp(\hat{a}) \sum_{i=1}^{m} \exp(a_i - \hat{a}).$$

Transforming this to the log-domain yields

$$\log(\mathcal{A}) = \hat{a} + \log\left(\sum_{i=1}^{m} \exp(a_i - \hat{a})\right).$$

Here, the largest term in the right hand side summation is $\exp(0) = 1$. We have thus transformed our calculation so that is has become more computationally stable.

4.2 Derivation of Algorithm 2

Having derived Algorithm 1, we now turn to Algorithm 2. The key to deriving Algorithm 2 is a formulation of (3.11) as a multimarginal optimal transport problem. This is done by first considering an intermediary formulation as a network flow problem. Similar to the previous section, we then add entropic regularization to the problem, after which we derive the dual problem of the regularized problem. Like before, this problem is solved with the coordinate ascent method, and due to strong duality between the primal and dual problem, we can use the obtained objective value as an approximation of T-GOSPA.

4.2.1 Formulation as a Network Flow Problem

In this section, we reformulate (3.11) as a network flow problem. To this end, we let $\mathcal{F}$ be the set of all four-dimensional tensors $F \in \mathbb{R}_+^{((m+1)\times(n+1))^2}$ having indices of dimension $m+1$, $n+1$, $m+1$, and $n+1$, in that order. We index it using the notation $F_{(i,j),(k,l)}$. This is to emphasize the fact that we see this entry as representing the flow from position (i,j) in one assignment matrix, to position (k,l) in another.

Starting from (3.11), we add these flow variables to the problem, along with some additional constraints to get

$$\underset{\substack{W^t \in \mathbb{R}_+^{(m+1)\times(n+1)},\\ t=1,\dots,T,\\ F^t \in \mathcal{F}_+,\\ t=1,\dots,T-1}}{\text{minimize}} \quad \sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D_{i,j}^t W_{i,j}^t + \frac{\gamma^p}{2}\sum_{t=1}^{T-1}\sum_{i=1}^{m}\sum_{j=1}^{n}\left|W_{i,j}^{t+1} - W_{i,j}^t\right|, \tag{4.26a}$$

$$\text{subject to} \quad W^t \mathbf{1}_{n+1} = \mu^X, \quad t = 1, \dots, T, \tag{4.26b}$$

$$(W^t)^\top \mathbf{1}_{m+1} = \mu^Y, \quad t = 1, \dots, T, \tag{4.26c}$$

$$W_{i,j}^{t+1} - W_{i,j}^t = \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^t - \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^t, \quad \begin{matrix} i = 1, \dots, m+1, \\ j = 1, \dots, n+1, \\ t = 1, \dots, T-1, \end{matrix} \tag{4.26d}$$

where we recall that the marginals μ^X and μ^Y are defined by (4.3). Adding the flow variables and the additional constraints does not affect the optimal objective value since we can first consider optimization over W^t, $t = 1, \dots, T$, and then select appropriate values for F^t, $t = 1, \dots, T-1$, to fulfill constraint (4.26d). Here, constraint (4.26d) is a flow constraint. It makes sure that the change in element (i, j) from time step t to time step $t+1$ is the difference between the incoming and outgoing flow at that position.

Now we make an observation. Since the objective function is a function on W^t, $t = 1, \dots, T$, and not on the flows, we can constrain the flow as long as we make sure that it is always possible to reach all feasible W^t, $t = 1, \dots, T$. We make the restriction that each position has either only incoming flows, or only outgoing flows. We model this using the constraint

$$\beta_{i,j}^t \sum_{\substack{k,l=1 \\ (i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^t + (1-\beta_{i,j}^t) \sum_{\substack{k,l=1 \\ (i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^t = 0,$$

for all $t = 1, \dots, T-1$, $(i, j) \in \{1, \dots, m+1\} \times \{1, \dots, n+1\}$, where we have introduced the binary variables $\beta_{i,j}^t$. Note that $\beta_{i,j}^t = 1$ implies that $F_{(k,l),(i,j)}^t = 0$ for all (k, l), and $\beta_{i,j}^t = 0$ implies that $F_{(i,j),(k,l)}^t = 0$ for all (k, l). This means that we have $\beta_{i,j}^t F_{(i,j),(k,l)}^t = F_{(i,j),(k,l)}^t$, as well as $(1-\beta_{k,l}^t) F_{(i,j),(k,l)}^t = F_{(i,j),(k,l)}^t$, which becomes useful below.

Introducing the Iverson brackets, defined as

$$[P] = \begin{cases} 1, & \text{if } P \text{ is true,} \\ 0, & \text{otherwise,} \end{cases}$$

we can now write the track-switching penalty in (4.26a) as

$$
\begin{aligned}
&\sum_{i,j=1}^{m,n} \left| W_{i,j}^{t+1} - W_{i,j}^{t} \right| \\
&\quad = \sum_{i,j=1}^{m+1,n+1} [i \neq m+1] \cdot [j \neq n+1] \cdot \left| W_{i,j}^{t+1} - W_{i,j}^{t} \right| \\
&\quad = \sum_{i,j=1}^{m+1,n+1} [i \neq m+1] \cdot [j \neq n+1] \\
&\qquad \cdots \left((1-\beta_{i,j}) \sum_{\substack{k,l=1 \\ (i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^{t} + \beta_{i,j} \sum_{\substack{k,l=1 \\ (i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^{t} \right) \\
&\quad = \sum_{i,j=1}^{m+1,n+1} \sum_{k,l=1}^{m+1,n+1} [(i,j) \neq (k,l)] \Big([k \neq m+1] \cdot [l \neq n+1] \cdot (1-\beta_{k,l}) \\
&\qquad\qquad + [i \neq m+1] \cdot [j \neq n+1] \cdot \beta_{i,j} \Big) F_{(i,j),(k,l)}^{t} \\
&\quad = \sum_{i,j=1}^{m+1,n+1} \sum_{k,l=1}^{m+1,n+1} [(i,j) \neq (k,l)] \Big([k \neq m+1] \cdot [l \neq n+1] \\
&\qquad\qquad + [i \neq m+1] \cdot [j \neq n+1] \Big) F_{(i,j),(k,l)}^{t}.
\end{aligned}
$$

This motivates the definition of a new cost tensor

$$
\tilde{D}_{(i,j),(k,l)} = \begin{cases} \dfrac{\gamma^p}{2} [(i,j) \neq (k,l)] \cdot \Big([k \neq m+1] \cdot [l \neq n+1] \\ \qquad\qquad + [i \neq m+1] \cdot [j \neq n+1] \Big), & \text{if } i = k, \\ \infty, & \text{else.} \end{cases}
$$

The second case in the definition above restricts the flow to only be between elements of the same row. This does not change the objective value as each row has constant sum by constraint (4.26b), and as such all feasible $W^1, \ldots, W^T$ are still reachable.

This definition allows us to formulate the problem as

$$\underset{\substack{W^t\in\mathbb{R}_+^{(m+1)\times(n+1)},\\ t=1,\dots,T,\\ F^t\in\mathcal{F}_+,\\ \beta\in\{0,1\}^{(m+1)\times(n+1)},\\ t=1,\dots,T-1}}{\text{minimize}} \quad \sum_{t=1}^{T}\sum_{i,j=1}^{m+1,n+1} D_{i,j}^t W_{i,j}^t + \sum_{t=1}^{T-1}\sum_{i,j=1}^{m+1,n+1}\sum_{k,l=1}^{m+1,n+1} \tilde{D}_{(i,j),(k,l)} F_{(i,j),(k,l)}^t, \tag{4.27a}$$

$$\text{subject to} \quad W^t \mathbf{1}_{n+1} = \mu^X, \quad t = 1,\dots,T, \tag{4.27b}$$

$$(W^t)^\top \mathbf{1}_{m+1} = \mu^Y, \quad t = 1,\dots,T, \tag{4.27c}$$

$$W_{i,j}^{t+1} - W_{i,j}^{t+1} = \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^t - \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^t, \quad \begin{matrix} i=1,\dots,m+1,\\ j=1,\dots,n+1,\\ t=1,\dots,T-1, \end{matrix} \tag{4.27d}$$

$$\beta_{i,j}^t \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^t + (1-\beta_{i,j}^t) \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^t = 0, \quad \begin{matrix} i=1,\dots,m+1,\\ j=1,\dots,n+1,\\ t=1,\dots,T-1. \end{matrix} \tag{4.27e}$$

The definition of the cost tensor $\tilde{D}$ has an intuitive explanation. The condition $i = k$ in the definition makes sure that we pay an infinite price for flows between rows, implicitly introducing a constraint. The factor $[(i,j) \neq (k,l)]$ says that flows starting and ending in the same position have no penalty. This makes sense, since this corresponds to the case of not changing the assignment between time steps. The terms $[k \neq m+1] \cdot [l \neq n+1]$ and $[i \neq m+1] \cdot [j \neq n+1]$ indicate when the flow destination and source are on the boundary of the assignment matrix, respectively. If both are on the boundary, there is no penalty. If exactly one is on the boundary, this indicates a half-switch, and a penalty of $\gamma^p/2$ is given. Lastly, if neither the flow destination nor source is on the boundary, this indicates a full switch (equivalent to two half-switches), giving a penalty of γ^p.

Next, we argue that the constraints (4.27e) can be removed. Remember that these constraints makes each position in the assignment matrices either only have outgoing or incoming flows. Consider (4.27) without the constraints (4.27e), and let F^* be an optimal flow in an optimal solution to this problem. Assume that there exists (i,j), (k,l), (r,s) so that $(F^*)_{(k,l),(i,j)}^t > 0$ and $(F^*)_{(i,j),(r,s)}^t > 0$, meaning that position (i,j) has at least one incoming and outgoing flow in the optimal solution. Part of this flow can be replaced by a flow going directly from (k,l) to (r,s) with a cost that is not larger than the original flow. By repeating this augmentation, we see that given a flow that does not fulfill constraint (4.27e), we can always find a flow with the same or lower cost but fulfilling the constraints. The argument for why the flow can be augmented this way is highlighted by the example below.

Example 2. *Consider a part of a larger flow where there is a flow of* $1/2$ *from* $(1,1)$ *to* $(1,2)$*, and a flow of* $1/2$ *to* $(1,2)$ *to* $(1,3)$*. This flow does not fulfill* (4.27e)*. Given that* $m \geq 1$ *and* $n \geq 3$*, the flows* $(1,1) \to (1,2)$*, and* $(1,2) \to (1,3)$ *yield a total cost*

of γ^p, while the alternative of sending $1/2$ from $(1,1)$ to $(1,3)$ directly only has a cost of $\gamma^p/2$. Analogous reasoning can be performed for other cases.

The argument above gives

$$\underset{\substack{W^t\in\mathbb{R}_+^{(m+1)\times(n+1)},\\ t=1,\dots,T,\\ F^t\in\mathcal{F}_+,\\ t=1,\dots,T-1}}{\text{minimize}} \quad \sum_{t=1}^{T}\sum_{i,j=1}^{m+1,n+1} D_{i,j}^t W_{i,j}^t + \sum_{t=1}^{T-1}\sum_{i,j=1}^{m+1,n+1}\sum_{k,l=1}^{m+1,n+1} \tilde{D}_{(i,j),(k,l)} F_{(i,j),(k,l)}^t, \tag{4.28a}$$

$$\text{subject to} \quad W^t \mathbf{1}_{n+1} = \mu^X, \quad t = 1,\dots,T, \tag{4.28b}$$

$$(W^t)^\top \mathbf{1}_{m+1} = \mu^Y, \quad t = 1,\dots,T, \tag{4.28c}$$

$$W_{i,j}^{t+1} - W_{i,j}^t = \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(k,l),(i,j)}^t - \sum_{\substack{k,l=1\\(i,j)\neq(k,l)}}^{m+1,n+1} F_{(i,j),(k,l)}^t, \quad \begin{matrix} i = 1,\dots,m+1,\\ j = 1,\dots,n+1,\\ t = 1,\dots,T-1, \end{matrix} \tag{4.28d}$$

as our final formulation of T-GOSPA using flows. See Figure 4.1 for an illustrative overview of the variables and structure of this formulation. Now that we have a formulation of T-GOSPA using flows, the next step is to rewrite this as a multimarginal optimal transport problem, which we do in the following section.

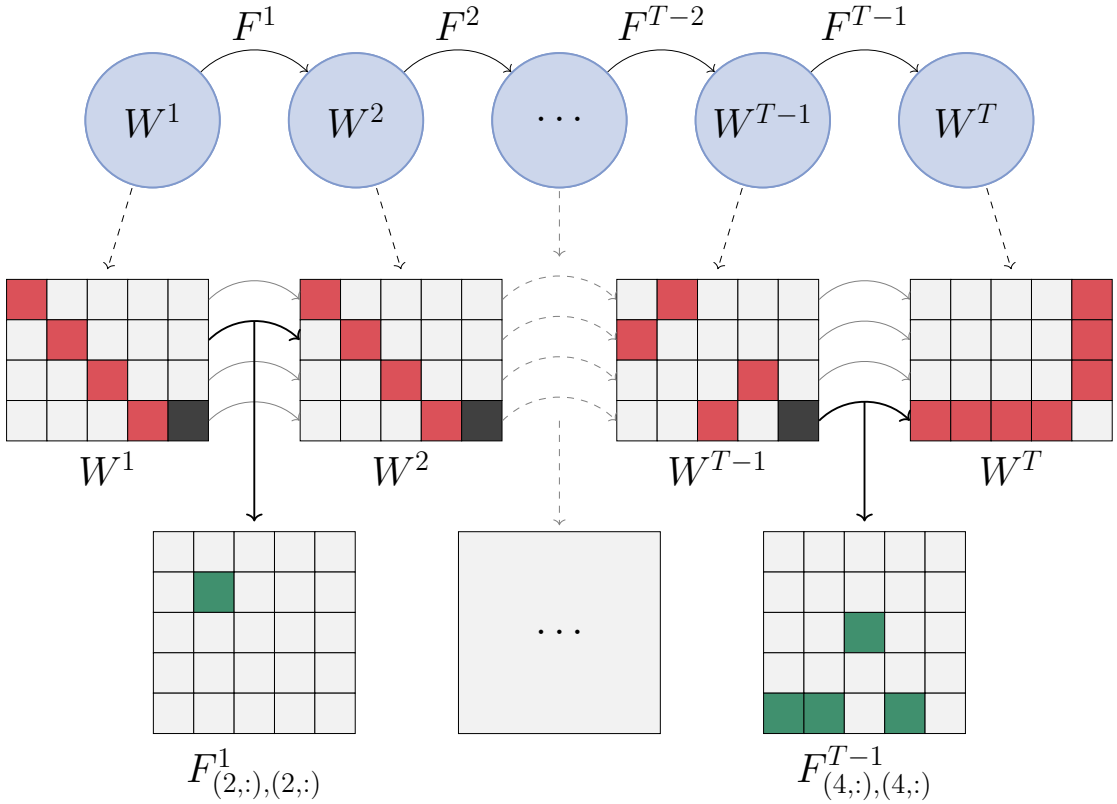


Figure 4.1: An illustration of the network flow formulation of the T-GOSPA metric. On the top, the underlying graphical structure is shown. Each node represents a marginal, while edges represent transport plans between the marginals. The marginals correspond to assignment matrices, showcased in the second row. We limit the flows to be between individual rows of the assignment matrices. The flow between each pair of rows in adjacent assignment matrices can be represented by a slice of the transport plan, showcased on the last row. Here $F^t_{(i,:),(i,:)}$ is the matrix constructed by fixing the first and third indices of F^t to i.

4.2.2 Formulation as a Multimarginal Optimal Transport Problem

In this section, we formulate (4.28) as a multi-marginal optimal transport problem. For this reason, let

$$\hat{\mathcal{M}}_+ = \mathbb{R}_+^{((m+1)\times(n+1))^T}$$

denote the set of all non-negative tensors with $2T$ indices such that odd indices have dimension $m+1$, and even indices have dimension $n+1$. We index such tensors using the notation $M_{(i_1,j_1),\dots,(i_T,j_T)}$ for $i_1,\dots,i_T \in \{1,\dots,m+1\}$ and $j_1,\dots,j_T \in \{1,\dots,n+1\}$. Moreover, we define the projections

$$\hat{P}_t(M)_{(i_t,j_t)} = \sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus(i_t,j_t)} M_{(i_1,j_1),\dots,(i_T,j_T)}, \quad t = 1,\dots,T,$$

and

$$\hat{P}_{t,t+1}(M)_{(i_t,j_t),(i_{t+1},j_{t+1})} = \sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus(i_t,j_t),(i_{t+1},j_{t+1})} M_{(i_1,j_1),\dots,(i_T,j_T)},$$
$$t = 1,\dots,T-1.$$

Analogous to the approach of Section 4.1.2, we make the substitutions

$$W^t = \hat{P}_t(M),$$
$$F^t = \hat{P}_{t,t+1}(M).$$

Note that this substitution renders (4.28d) redundant, and therefore it can be removed from the problem. Note now that we also have

$$\begin{aligned}\sum_{t=1}^{T}\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} D^t_{i,j}\hat{P}_t(M)_{(i,j)} &= \sum_{t=1}^{T}\sum_{i_t=1}^{m+1}\sum_{j_t=1}^{n+1} D^t_{i_t,j_t} \sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus(i_t,j_t)} M_{(i_1,j_1),\dots,(i_T,j_T)} \\ &= \sum_{(i_1,j_1)\dots,(i_T,j_T)} \sum_{t=1}^{T} D^t_{i_t,j_t} M_{(i_1,j_1),\dots,(i_T,j_T)},\end{aligned}$$

and

$$\begin{aligned}&\sum_{t=1}^{T-1}\sum_{i,j=1}^{m+1,n+1}\sum_{k,l=1}^{m+1,n+1} \tilde{D}_{(i,j),(k,l)} F^t_{(i,j),(k,l)} \\ &\quad= \sum_{t=1}^{T-1}\sum_{i_t,j_t=1}^{m+1,n+1}\sum_{i_{t+1},j_{t+1}=1}^{m+1,n+1} \tilde{D}_{(i_t,j_t),(i_{t+1},j_{t+1})} \sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus(i_t,j_t),(i_{t+1},j_{t+1})} M_{(i_1,j_1),\dots,(i_T,j_T)} \\ &\quad= \sum_{(i_1,j_1)\dots,(i_T,j_T)} \sum_{t=1}^{T-1} \tilde{D}_{(i_t,j_t),(i_{t+1},j_{t+1})} M_{(i_1,j_1),\dots,(i_T,j_T)}.\end{aligned}$$

This means that by defining the cost tensor as

$$\hat{C}_{(i_1,j_1),\dots,(i_T,j_T)} = \sum_{t=1}^{T} D^t_{i_t,j_t} + \sum_{t=1}^{T-1} \tilde{D}_{(i_t,j_t),(i_{t+1},j_{t+1})},$$

we can write the problem as

$$\begin{aligned}\underset{M\in\hat{\mathcal{M}}_+}{\text{minimize}} \quad & \langle \hat{C}, M\rangle, && \text{(4.29a)}\\ \text{subject to} \quad & \hat{P}_t(M)\mathbf{1}_{n+1} = \mu^X, \quad t = 1,\dots,T, && \text{(4.29b)}\\ & \hat{P}_t(M)^\top \mathbf{1}_{m+1} = \mu^Y, \quad t = 1,\dots,T. && \text{(4.29c)}\end{aligned}$$

This is a multimarginal optimal transport problem which has the same optimal objective value as (3.11), and it forms the basis for the remainder of the derivation of Algorithm 2.

4.2.3 Entropic Regularization

In this section, we regularize (4.29) in order to apply a technique similar to the one presented in Section 2.2.4, as was done for Algorithm 1. We do this by adding an entropy term $\varepsilon\hat{E}(M)$ similarly to the first derivation. Here, we define

$$\hat{E}(M) = \sum_{(i_1,j_1),\dots,(i_T,j_T)} \left(M_{(i_1,j_1),\dots,(i_T,j_T)} \log M_{(i_1,j_1),\dots,(i_T,j_T)} - M_{(i_1,j_1),\dots,(i_T,j_T)} + 1 \right).$$

Let $\varepsilon > 0$. The resulting entropy regularized version of (4.29) becomes

$$\begin{aligned} \underset{M \in \hat{\mathcal{M}}_+}{\text{minimize}} \quad & \langle \hat{C}, M \rangle + \varepsilon \hat{E}(M), && \text{(4.30a)} \\ \text{subject to} \quad & \hat{P}_t(M)\mathbf{1}_{n+1} = \mu^X, \quad t = 1, \dots, T, && \text{(4.30b)} \\ & \hat{P}_t(M)^\top \mathbf{1}_{m+1} = \mu^Y, \quad t = 1, \dots, T. && \text{(4.30c)} \end{aligned}$$

Next, we derive the Lagrangian dual problem of (4.30).

4.2.4 Derivation of the Dual Problem

In this section, we derive the dual of (4.30). We do this by relaxing the constraints (4.30b) and (4.30c), with the dual variables $\lambda^t \in \mathbb{R}^{m+1}$ and $\hat{\lambda}^t \in \mathbb{R}^{n+1}$, for $t = 1, \dots, T$, respectively. Note that in contrast to (4.7), we only have two sets of constraints here, and thus we only need two sets of dual variables. Especially, note that the constraints (4.30b) correspond to the constraints (4.7b) and that the constraints (4.30c) correspond to the constraints (4.7c), whereas the complicating constraints (4.7d) and (4.7e) are no longer present. This is the key difference between the two formulations, and main point of creating an algorithm based upon a flow problem formulation. In exchange for eluding such complicating constraints, we have accepted a somewhat more complicated cost tensor, and in particular a cost tensor for which some entries are infinite.

With the constraints (4.30b) and (4.30c) relaxed with the dual variables introduced above, the Lagrangian $\hat{\mathcal{L}}$ becomes

$$\begin{aligned} \hat{\mathcal{L}}(M, \lambda, \hat{\lambda}) = \langle \hat{C}, M \rangle + \varepsilon \hat{E}(M) \\ + \sum_{t=1}^{T} \sum_{i=1}^{m+1} \lambda_i^t \left(\mu_i^X - \sum_{j=1}^{n+1} \hat{P}_t(M)_{(i,j)} \right) \\ + \sum_{t=1}^{T} \sum_{j=1}^{n+1} \hat{\lambda}_j^t \left(\mu_j^Y - \sum_{i=1}^{m+1} \hat{P}_t(M)_{(i,j)} \right), \end{aligned}$$

and the corresponding dual problem becomes

$$\underset{\substack{\lambda^t \in \mathbb{R}^{m+1}, \hat{\lambda}^t \in \mathbb{R}^{n+1} \\ t=1,\dots,T}}{\text{maximize}} \ \underset{M \in \hat{\mathcal{M}}_+}{\text{minimize}} \ \hat{\mathcal{L}}(M, \lambda, \hat{\lambda}).$$

Analogously to the derivation of Algorithm 1, we now solve the inner minimization problem in the primal variable M analytically. To this end, we take the first derivative

of $\hat{\mathcal{L}}$ with respect to a single element in M. This gives

$$\frac{\partial \mathcal{L}}{\partial M_{(i_1,j_1),\dots,(i_T,j_T)}} = \hat{C}_{(i_1,j_1),\dots,(i_T,j_T)} + \varepsilon \log(M_{(i_1,j_1),\dots,(i_T,j_T)}) - \sum_{t=1}^{T} \lambda_{i_t}^t - \sum_{t=1}^{T} \hat{\lambda}_{j_t}^t.$$

Setting the derivative to zero and solving for M yields

$$M^*_{(i_1,j_1),\dots,(i_T,j_T)} = \exp\left(-\frac{\hat{C}_{(i_1,j_1),\dots,(i_T,j_T)}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\lambda_{i_t}^t}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\hat{\lambda}_{j_t}^t}{\varepsilon}\right). \tag{4.31}$$

Here, $M^* \geq 0$. By the convexity of $\hat{\mathcal{L}}$ together with Theorem 1, we therefore conclude that M^* is optimal. We thus have the dual problem

$$\underset{\substack{\lambda^t \in \mathbb{R}^{m+1},\, \hat{\lambda}^t \in \mathbb{R}^{n+1} \\ t=1,\dots,T}}{\text{maximize}} \; \hat{\phi}(\lambda, \hat{\lambda}), \tag{4.32}$$

for the dual function $\hat{\phi}(\lambda, \hat{\lambda}) = \hat{\mathcal{L}}(M^*, \lambda, \hat{\lambda})$, where

$$\hat{\mathcal{L}}(M^*, \lambda, \hat{\lambda}) = -\varepsilon \sum_{(i_1,j_1),\dots,(i_T,j_T)} \exp\left(-\frac{\hat{C}_{(i_1,j_1),\dots,(i_T,j_T)}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\lambda_{i_t}^t}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\hat{\lambda}_{j_t}^t}{\varepsilon}\right) + \sum_{t=1}^{T} \sum_{i=1}^{m+1} \mu_i^X \lambda_i^t + \sum_{t=1}^{T} \sum_{j=1}^{n+1} \mu_j^Y \hat{\lambda}_j^t + \varepsilon (m+1)^T (n+1)^T.$$

Similarly to Algorithm 1, we solve the dual problem numerically using a coordinate ascent scheme. This is the topic of the next section.

4.2.5 Derivation of the Coordinate Ascent Scheme

In this section, we derive the update rules needed for the coordinate ascent scheme. Similarly to the derivation of Algorithm 1, we introduce the transformed dual variables

$$u^t = \exp\left(\frac{\lambda^t}{\varepsilon}\right), \quad \hat{u}^t = \exp\left(\frac{\hat{\lambda}^t}{\varepsilon}\right), \quad \text{for } t = 1, \dots, T,$$
$$\hat{K} = \exp\left(-\frac{\hat{C}}{\varepsilon}\right).$$

Since problem (4.32) is convex, we take the derivative of the the dual function with respect to element $\lambda_{i_\tau}^\tau$, for some $\tau = 1, \dots, T$, and $i_\tau = 1, \dots, m+1$, giving

$$\begin{aligned}\frac{\partial \phi}{\partial \lambda_{i_\tau}^\tau} &= -\sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus i_\tau} \exp\left(-\frac{\hat{C}_{(i_1,j_1),\dots,(i_T,j_T)}}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\lambda_{i_t}^t}{\varepsilon}\right) \prod_{t=1}^{T} \exp\left(\frac{\hat{\lambda}_{j_t}^t}{\varepsilon}\right) + \mu_{i_\tau}^X \\ &= -\sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus i_\tau} \hat{K}_{(i_1,j_1),\dots,(i_T,j_T)} \prod_{t=1}^{T} u_{i_t}^t \prod_{t=1}^{T} \hat{u}_{j_t}^t + \mu_{i_\tau}^X.\end{aligned}$$

We now define a new type of projection $\hat{P}_t^X(M)$, for $t = 1, \ldots, T$, as

$$\hat{P}_t^X(M)_{i_t} = \sum_{j_t} \sum_{(i_1,j_1),\ldots,(i_{\tau-1},j_{\tau-1}),(i_{\tau+1},j_{\tau+1}),\ldots,(i_T,j_T)} M_{(i_1,j_1),\ldots,(i_T,j_T)},$$

and note that

$$\begin{aligned}\hat{P}_\tau^X(M^*)_{i_\tau} \Big/ u_{i_\tau}^\tau &= \sum_{j_t} \sum_{(i_1,j_1),\ldots,(i_{\tau-1},j_{\tau-1}),(i_{\tau+1},j_{\tau+1}),\ldots,(i_T,j_T)} \hat{K}_{(i_1,j_1),\ldots,(i_T,j_T)} \prod_{\substack{t=1 \\ t\neq\tau}}^{T} u_{i_t}^t \prod_{t=1}^{T} \hat{u}_{j_t}^t \\ &= w_{i_\tau}^\tau,\end{aligned}$$

where we have defined a vector $w_{i_\tau}^\tau$ that is independent of $u_{i_\tau}^\tau$. With this, we have that

$$\frac{\partial \phi}{\partial \lambda_{i_\tau}^\tau} = -u_{i_\tau}^\tau w_{i_\tau}^\tau + \mu_{i_\tau}^X.$$

Setting this to zero and solving for $u_{i_\tau}^\tau$, we get

$$u_{i_\tau}^\tau = \frac{\mu_{i_\tau}^X}{w_{i_\tau}^\tau} = \frac{\mu_{i_\tau}^X}{\hat{P}_\tau^X(M^*)_{i_\tau}} u_{i_\tau}^\tau.$$

Note that the right hand side is independent of $u_{i_\tau}^\tau$. The update rule is therefore

$$u_{i_\tau}^\tau \leftarrow \frac{\mu_{i_\tau}^X}{\hat{P}_\tau^X(M^*)_{i_\tau}} u_{i_\tau}^\tau,$$

or in vector form,

$$u^\tau \leftarrow \mu^X \oslash \hat{P}_\tau^X(M^*) \odot u^\tau, \quad \text{for } \tau = 1, \ldots, T. \tag{4.33}$$

Analogously, by defining the projection $\hat{P}_t^Y(M)$, for $t = 1, \ldots, T$, as

$$\hat{P}_t^Y(M)_{j_t} = \sum_{i_t} \sum_{(i_1,j_1),\ldots,(i_{\tau-1},j_{\tau-1}),(i_{\tau+1},j_{\tau+1}),\ldots,(i_T,j_T)} M_{(i_1,j_1),\ldots,(i_T,j_T)},$$

we obtain the update rule

$$\hat{u}^\tau \leftarrow \mu^Y \oslash \hat{P}_\tau^Y(M^*) \odot \hat{u}^\tau, \quad \text{for} \quad \tau = 1, \ldots, T. \tag{4.34}$$

Given some feasible initial values, for instance setting u_τ^X and u_τ^Y to one everywhere for all $\tau = 1, \ldots, T$, we iterate using (4.33) and (4.34) alternately. As all constraints for (4.29) are affine, Theorem 2 implies[1] that strong duality holds between (4.29) and (4.32). Therefore, the dual and primal problem have the same optimal objective value, and in particular the optimal solutions are linked by (4.31). Like before, we need to find a way of calculating the projections efficiently. This is the topic of the next section.

[1]Here, it should be noted that Theorem 2 cannot be used directly as some entries in the cost tensor are infinite. However, this can easily be rectified by fixing the corresponding variables to zero and setting the infinite entries in the cost tensor to any real value.

4.2.6 Message Passing

Again, like in Section 4.1.6, we utilize the structure of the problem in order to schedule the computations in such a way that less computation is needed overall. To this end, we decouple the cost tensor as

$$\begin{aligned}\hat{K}_{(i_1,j_1),\dots,(i_T,j_T)} &= \prod_{t=1}^{T} \exp\left(-\frac{D^t_{i_t,j_t}}{\varepsilon}\right) \prod_{t=1}^{T-1} \exp\left(-\frac{\tilde{D}_{(i_t,j_t),(i_{t+1},j_{t+1})}}{\varepsilon}\right) \\ &= \prod_{t=1}^{T} k^t_{(i_t,j_t)} \prod_{t=1}^{T-1} \tilde{k}_{(i_t,j_t),(i_{t+1},j_{t+1})},\end{aligned}$$

where

$$k^t_{(i_t,j_t)} = \exp\left(-\frac{D^t_{i_t,j_t}}{\varepsilon}\right),$$

and

$$\tilde{k}_{(i_t,j_t),(i_{t+1},j_{t+1})} = \exp\left(-\frac{\tilde{D}_{(i_t,j_t),(i_{t+1},j_{t+1})}}{\varepsilon}\right).$$

Next, we observe that the projection $\hat{P}_\tau(M^*)_{(i_\tau,j_\tau)}$ can be written as

$$\begin{aligned}\hat{P}_\tau(M^*)_{(i_\tau,j_\tau)} &= \sum_{(i_1,j_1),\dots,(i_T,j_T)\setminus(i_\tau,j_\tau)} \prod_{t=1}^{T} k^t_{(i_t,j_t)} \prod_{t=1}^{T-1} \tilde{k}_{(i_t,j_t),(i_{t+1},j_{t+1})} \prod_{t=1}^{T} u^t_{i_t} \prod_{t=1}^{T} \hat{u}^t_{j_t} \\ &= \hat{\Phi}^\tau_{(i_\tau,j_\tau)} k^\tau_{(i_\tau,j_\tau)} u^\tau_{i_\tau} \hat{u}^\tau_{j_\tau} \Phi^\tau_{(i_\tau,j_\tau)},\end{aligned} \tag{4.35}$$

where

$$\hat{\Phi}^\tau_{(i_\tau,j_\tau)} = \sum_{(i_1,j_1),\dots,(i_{\tau-1},j_{\tau-1})} \prod_{t=1}^{\tau-1} k^t_{(i_t,j_t)} \prod_{t=1}^{\tau-1} \tilde{k}_{(i_t,j_t),(i_{t+1},j_{t+1})} \prod_{t=1}^{\tau-1} u^t_{i_t} \prod_{t=1}^{\tau-1} \hat{u}^t_{j_t}, \quad \tau = 2,\dots,T,$$

with

$$\hat{\Phi}^1_{(i_1,j_1)} = 1,$$

and

$$\Phi^\tau_{(i_\tau,j_\tau)} = \sum_{(i_{\tau+1},j_{\tau+1}),\dots,(i_T,j_T)} \prod_{t=\tau+1}^{T} k^t_{(i_t,j_t)} \prod_{t=\tau}^{T-1} k_{(i_t,j_t),(i_{t+1},j_{t+1})} \prod_{t=\tau+1}^{T} u^t_{i_t} \prod_{t=\tau+1}^{T} \hat{u}^t_{j_t},$$

$$\tau = 1,\dots,T-1,$$

with

$$\Phi^T_{(i_T,j_T)} = 1.$$

We now note that we have the recursive relationships

$$\hat{\Phi}^\tau_{(i_\tau,j_\tau)} = \sum_{(i_{\tau-1},j_{\tau-1})} \hat{\Phi}^{\tau-1}_{(i_{\tau-1},j_{\tau-1})} k^{\tau-1}_{(i_{\tau-1},j_{\tau-1})} \tilde{k}_{(i_{\tau-1},j_{\tau-1}),(i_\tau,j_\tau)} u^{\tau-1}_{i_{\tau-1}} \hat{u}^{\tau-1}_{j_{\tau-1}}, \tag{4.36}$$

and

$$\Phi^{\tau}_{(i_\tau, j_\tau)} = \sum_{(i_{\tau+1}, j_{\tau+1})} \Phi^{\tau+1}_{(i_{\tau+1}, j_{\tau+1})} k^{\tau+1}_{(i_{\tau+1}, j_{\tau+1})} \tilde{k}_{(i_\tau, j_\tau),(i_{\tau+1}, j_{\tau+1})} u^{\tau+1}_{i_{\tau+1}} \hat{u}^{\tau+1}_{j_{\tau+1}}. \tag{4.37}$$

This means that given $\hat{\Phi}^{\tau-1}$, we can compute $\hat{\Phi}^{\tau}$, and given $\Phi^{\tau+1}$, we can compute Φ^{τ}. By (4.35), we can then compute $\hat{P}_\tau(M^*)$. Given $\hat{P}_\tau(M^*)$, we can compute the projections in the update rules (4.33) and (4.34) by

$$\hat{P}^X_\tau(M^*)_i = \sum_j \hat{P}_\tau(M^*)_{(i,j)},$$
$$\hat{P}^Y_\tau(M^*)_j = \sum_i \hat{P}_\tau(M^*)_{(i,j)}.$$

In summary, this means that we now have a way of computing the projections more efficiently. Equipped with these tools, we are now ready to write down Algorithm 2, which we do in the next section.

4.2.7 Pseudocode for Algorithm 2

By alternating the update rules (4.33) and (4.34), we get a coordinate ascent algorithm. Further, by using the relationships (4.36) and (4.37), we also have a way of computing the necessary projections efficiently. Bringing these parts together, we get Algorithm 2, which is displayed below.

Algorithm 2

Initialize u and $\hat{u}$ to one everywhere
while Not converged **do**
    $\Phi^T \leftarrow 1$
    for $t = T-1, \dots, 1$ **do**
        Compute Φ^t from Φ^{t+1} using (4.37)
    end for
    $\hat{\Phi}^1 \leftarrow 1$
    for $t = 1, \dots, T-1$ **do**
        $u^t \leftarrow \mu^X \oslash \hat{P}^X_t(M^*) \odot u^t$
        $\hat{u}^t \leftarrow \mu^Y \oslash \hat{P}^Y_t(M^*) \odot \hat{u}^t$
        Compute $\hat{\Phi}^{t+1}$ from $\hat{\Phi}^t$ using (4.36)
    end for
    $u^T \leftarrow \mu^X \oslash \hat{P}^X_T(M^*) \odot u^T$
    $\hat{u}^T \leftarrow \mu^Y \oslash \hat{P}^Y_T(M^*) \odot \hat{u}^T$
end while

Like for Algorithm 1, different types of convergence criteria can be used. This is discussed more thoroughly in Section 5.1.2.

5

Numerical Results

In this chapter, we test our algorithms on simulated data. Section 5.1 serves to introduce the experimental setup used. The obtained results are then showcased in Section 5.2.

5.1 Experimental Setup

In this section, we describe the setup of our numerical experiments. Section 5.1.1 is dedicated to explaining the data simulation process. Section 5.1.2 explains the most important quantities that are reported in Section 5.2. Lastly, in Section 5.1.3, some remarks on the implementation are made.

5.1.1 Simulation of Data

We run our algorithms on two types of data. The first type of data is generated by sampling the cost matrices D^t at each time step directly. Throughout this chapter, this is referred to as unstructured data. In the second case we first generate ground truth trajectories. These are then corrupted by adding noise, track-switches, and false trajectories in order to generate a dummy estimate used for testing. We refer to this type of data as structured data.

Unstructured Data

Out of the two types of simulated data, this is the most straightforward. For distance matrices $D^t \in \mathbb{R}^{(m+1)\times(n+1)}$, $t = 1, \dots, T$, we simply sample each element from a uniform distribution. More precisely, we sample

$$D^t_{i,j} \sim \mathcal{U}(0,1),$$

for all elements except the bottom-right corner. Thus, we sample for all

$$(t,i,j) \in \{1,\dots,T\} \times (\{1,\dots,m+1\} \times \{1,\dots,n+1\} \setminus \{(m+1,n+1)\}).$$

Here, $\mathcal{U}(0,1)$ denotes a uniform distribution on the half-open interval $[0,1)$. Moreover, we set $D^t_{m+1,n+1} = 0$, for $t = 1, \dots, T$, since our algorithms cannot handle the case where the bottom-right corners of the distance matrices are non-zero.

While this data does not have the typical characteristics of an instance of T-GOSPA found in the evaluation of multiple target tracking algorithms, it is still a well-defined problem and a valid input to the algorithms. This showcases the most general problem the algorithms can handle. A summary of the parameters used for the generation of unstructured data is provided in Table 5.1.

Table 5.1: Parameters for simulating unstructured data according to the procedure outlined in Section 5.1.1.

Description	Parameter
Number of ground truth trajectories	m
Number of estimated trajectories	n
Number of time steps	T

Structured Data

In order to properly assess how the algorithms perform when evaluating multiple target tracking algorithms, tests need to be done with more realistic data. This section is devoted to explaining how we generate such data. The high level steps of this process are:

Step 1: Generate ground truth trajectories from a certain movement model.

Step 2: Add track-switches.

Step 3: Add missed and false trajectories.

Step 4: Add noise.

We now describe the steps in detail.

Step 1: Generating ground truth trajectories. The first step is to decide the number of true and false trajectories to be included in the ground truth. With a true trajectory we mean a trajectory that exists in the estimate, and with a false trajectory we mean a trajectory that is missed in the estimate. Let m_t and m_f be the number of such true and false trajectories, respectively, and let $m = m_t + m_f$.

Furthermore, we need to determine the sampling rate Δt. This determines how often measurements are taken from the movement model. Instead of setting the sampling rate directly, we parameterize it as $\Delta t = r/T$. The reason for this is that for a fixed r, the trajectories cover about the same distance for any number of time steps T. The value of r decides roughly how much distance the trajectories cover. When testing the algorithms on test cases with different values of T, this is a useful property since it makes the cases more similar, and therefore makes the comparison between them more truthful.

We consider the two-dimensional case with target states in $\mathbb{R}^4$. Here, the first coordinate represents horizontal position, the second horizontal velocity, the third

vertical position, and the fourth vertical velocity. For a state $x \in \mathbb{R}^4$, we denote these by subscripts as x_1, x_2, x_3, and x_4, respectively

We are now ready to describe how a trajectory is generated. To generate a trajectory $X = (t_b, x^{1:\ell})$, we do the following:

1. Sample t_b and $\tilde{t}_d$ from a geometric distribution supported on $\mathbb{N}$ with probability of success q. Let $t_d = T - \tilde{t}_d + 1$. The numbers t_b and t_d denote the time index of birth and death of the trajectory, respectively. This implies $\ell = t_d - t_b + 1$. Here, it is possible that $t_b > t_d$, or that they take values outside of $\{1, \dots, T\}$. In these cases, we repeat the sampling procedure.

2. Initialize the trajectory by sampling the initial position according to

$$\begin{pmatrix} x_1^1 \\ x_3^1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right),$$

   and the initial velocity according to

$$\begin{pmatrix} x_2^1 \\ x_4^1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} -x_1^1 \\ -x_3^1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right).$$

   Here, $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ denotes a multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ. The expected initial velocity is pointed towards the origin. This results in testing data where many trajectories pass through a small region of space around the origin, which is useful for testing the part of the metric that penalizes track-switching.

3. Recursively generate the states for further time steps by sampling

$$x^{t+1} \sim \mathcal{N}(Fx^t, Q),$$

   for $t = t_b, \dots, t_d - 1$ with

$$F = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} (\Delta t)^3/3 & (\Delta t)^2/2 & 0 & 0 \\ (\Delta t)^2/2 & \Delta t & 0 & 0 \\ 0 & 0 & (\Delta t)^3/3 & (\Delta t)^2/2 \\ 0 & 0 & (\Delta t)^2/2 & \Delta t \end{pmatrix}.$$

   The matrices F and Q are constructed to make the simulated trajectories more similar to what might be found in a real world scenario. See the red trajectories in Figure 5.3 for examples of how these trajectories look. This step is based on the data simulation procedure of [17].

4. Add this trajectory to the ground truth $\boldsymbol{X}$.

To obtain the true trajectories of the ground truth, we start with $\boldsymbol{X} = \varnothing$ and repeat the procedure above m_t times.

Step 2: Adding track-switches. The procedure of generating the dummy estimate is started by copying $\boldsymbol{X}$ from the previous time step. Let this copy be $\boldsymbol{Y} = \{Y_1, \dots, Y_{m_t}\}$, where $Y_j = X_j$ for $j = 1, \dots, m_t$. To add N_{ts} track-switches to the dummy estimate $\boldsymbol{Y}$, we do the following:

1. Sample a time step t_s uniformly on the set $\{1, \ldots, T\}$. Sample j_1 and j_2 from the set $\{1, \ldots, m_t\}$.
2. If $j_1 = j_2$, or if t_s, j_1, and j_2 have been tried before, repeat from step 1.
3. If $|\boldsymbol{y}_{j_1}^{t_s}| = |\boldsymbol{y}_{j_2}^{t_s}| = 1$, let $y^1 \in \boldsymbol{y}_{j_1}^{t_s}$, and $y^2 \in \boldsymbol{y}_{j_2}^{t_s}$. If $||(y_1^1, y_3^1)^\top - (y_1^2, y_3^2)^\top||_2 < c_s$, we swap the values of $\boldsymbol{y}_{j_1}^t$ and $\boldsymbol{y}_{j_2}^t$, for $t = t_s, \ldots, T$. That is, we swap the identities of the trajectories from time step t_s and onward. Here, it is possible that the first condition fails, in which case we retry.
4. If the number of successful swaps is less than N_{ts}, and the total number of attempts on step 3 is less than $N_{\max}$, then repeat from step 1. Otherwise, terminate.

The limit on the number of attempts in step 3 is practical since it limits the computational expense and also handles the case when there exist no trajectories satisfying the condition in step 3. For this reason, we are not guaranteed any number of track-switches. For our purposes though, this ad hoc procedure is enough. After completing the steps above, we have introduced track-switches into the dummy estimate $\boldsymbol{Y}$.

Step 3: Adding missed and false trajectories. Here, we simply generate an additional m_f trajectories according to the procedure in the first step and add them to $\boldsymbol{X}$. Additionally, we generate an additional n_f trajectories according in the same way and add them to $\boldsymbol{Y}$. Let $n = m_t + n_f$ be the total number of trajectories in $\boldsymbol{Y}$.

Step 4: Adding noise. Lastly, we add Gaussian noise with mean zero and standard deviation σ to all states in $\boldsymbol{Y}$.

We have now generated a ground truth $\boldsymbol{X}$ with m trajectories, and a dummy estimate $\boldsymbol{Y}$ with n trajectories. In this thesis, we limit ourselves to using the position components of the states, and therefore we drop the velocity components. A summary of all parameters for the generation of structured data is provided in Table 5.2.

Table 5.2: Parameters for simulating structured data according to the procedure outlined in Section 5.1.1.

Description	Parameter
Number of true ground truth trajectories	m_t
Number of false ground truth trajectories	m_f
Number of false estimated trajectories	n_f
Number of time steps	T
Distance parameter for trajectories	r
Birth-death probability	q
Track-switching cutoff	c_s
Max number of track-switches	N_{ts}
Max number of track-switch attempts	$N_{\max}$
Noise for estimate	σ

5.1.2 Reported Quantities

Throughout the remainder of this chapter, we report two quantities when conducting our numerical investigations. After each iteration of the respective algorithm, we record the relative step size

$$\frac{||\boldsymbol{u} - \boldsymbol{u}_{\text{prev}}||_2}{||\boldsymbol{u}_{\text{prev}}||_2}, \tag{5.1}$$

where $\boldsymbol{u}$ and $\boldsymbol{u}_{\text{prev}}$ are vectors constructed by concatenation of all transformed dual variables from two consecutive iterations. This tells us the size of the previous iteration step relative to the magnitude of the previous iterate. If this quantity decreases, it indicates that the algorithm is converging.

Additionally, we record the value of the objective function at the current iterate with the regularization term $\varepsilon E(M)$ omitted. This is the most important value, and the one used in practice to evaluate a multiple target tracking algorithm.

We now provide a brief motivation for omitting the regularization term from our reported figures. Let f be the primal objective function in (4.29), that is, the objective function without the regularization term. Let g be the primal objective function with the regularization. Thus

$$g(M) = f(M) + \varepsilon E(M).$$

Let $\widetilde{M}$ be an approximate solution to the regularized problem found by the algorithm, and let M^* be the optimal solution to (4.29). Furthermore, let f^*_{LP} be the optimal objective value to the linear programming relaxation of T-GOSPA in (4.2). Assuming that $\widetilde{M}$ fulfills the constraints (4.29), we have

$$f^*_{\text{LP}} = f(M^*) \leq f(\widetilde{M}) \leq g(\widetilde{M}) = f(\widetilde{M}) + \varepsilon E(\widetilde{M}),$$

since $E(M) \geq 0$ for all M. Therefore, to obtain a better approximation of f^*_{LP}, it is justified to consider $f(\widetilde{M})$. In practice, $\widetilde{M}$ never fulfills the constraints (4.29) exactly. This is mostly due to limited floating point precision and the fact that the algorithms are never run to complete convergence. From practical experience, the use of $f(\widetilde{M})$ still provides good approximations, and as such is what is used throughout this chapter. The reasoning above was done using the formulation in (4.29) made for Algorithm 2. Analogous reasoning can be done for Algorithm 1.

In order to compare the behavior of the algorithms between test cases, we consider the error relative to the linear programming reference

$$\left| \frac{f(\widetilde{M})}{f^*_{\text{LP}}} - 1 \right|. \tag{5.2}$$

In practice, f^*_{LP} is found by solving (4.2) directly using a linear programming solver.

5.1.3 Implementation Remarks

We now turn to discuss the regularization parameter ε. The value of ε needs to be in relation to the magnitude of the rest of the objective function. For this reason,

we parameterize ε in terms of problem data and a parameter η. More precisely, for Algorithm 1 we take

$$\varepsilon = \eta \cdot T \cdot \max(\max(D), \gamma^p),$$

and for Algorithm 2 we take

$$\varepsilon = \eta \cdot T \cdot \max(\max(D), \max(\tilde{D})).$$

Here, $\max(D)$ denotes the maximum element in D^t, $t = 1, \ldots, T$, $\max(\tilde{D})$ denotes the maximum element that is not infinity in $\tilde{D}$, and η is the normalized regularization magnitude. The reason for choosing ε this way is that now a fixed value of η can be used to provide a similar level of regularization across problem instances where the magnitude of the values in C might be very different. Moreover, it is known from literature that the complexity for similar algorithms to what was derived in the previous chapter depends on the ratio between ε and the largest element of C, see for example [27], making the parameterization above reasonable. When showing our results, we refer to η as the regularization magnitude.

The algorithms were implemented[1] in Python using a combination of the packages NumPy [28], SciPy [29], and PyTorch [30]. Here PyTorch allowed for the usage of both CPU and GPU resources, which we utilized in some experiments. For our linear programming solver, we used the COIN-OR CLP/CBC LP solver [31, 32]. This solver is based on the simplex method. The solver was interfaced from Python by the PuLP package [33]. The hardware used was an Intel Xeon Silver 4210 @ 2.20 GHz (CPU) and a NVIDIA A100 (GPU). Figures were generated in Python using the package Matplotlib [34].

In the experiments in Section 5.2.1 and Section 5.2.2, we use 64-bit floating-point precision in the calculations, and in the experiments in Section 5.2.3, we use 32-bit floating-point precision. This is because the former sections focus on qualitative aspects of the algorithms, whereas the latter section focuses on computational speed. In the latter, the GPU is utilized for some calculations, which is not optimized for 64-bit floating-point numbers.

5.2 Numerical Experiments

In this section, we run our algorithms through several experiments and report the results. In Section 5.2.1, we consider the convergence of the algorithms on different types of data. In Section 5.2.2, we empirically analyze the effect of the regularization parameter with respect to the quality of the solutions. Finally, in Section 5.2.3, we show how the running time scales with the size of the input data. There, we compare our fastest algorithm with the alternative method of using a third-party linear programming solver.

[1]The code produced as part of this thesis can be found on GitHub on the address:

`https://github.com/alfredwarnsater/efficient-mtt-eval-ot-MVEX03`

5.2.1 Convergence

In this section, we run the algorithms on one instance of unstructured data, and one instance of structured data, and analyze the results. Particularly, we study the convergence properties of the algorithms empirically.

Unstructured Data

We run Algorithm 1 and 2 on unstructured data generated according to the procedure outlined in Section 5.1.1. The parameters used for data generation, as well as the parameters used for the algorithms, are shown in Table 5.3. The results of this experiment is shown in Figure 5.1 for Algorithm 1, and Figure 5.2 for Algorithm 2.

Table 5.3: Parameters for the experiments on unstructured data in Section 5.2.1.

Parameter	Value
T	20
m	16
n	15

(a) Data parameters

Parameter	Value
c	Not applicable
p	1
γ	0.1
η	$5 \cdot 10^{-5}$
Convergence criteria	Stop after 10^3 iterations

(b) Algorithm parameters

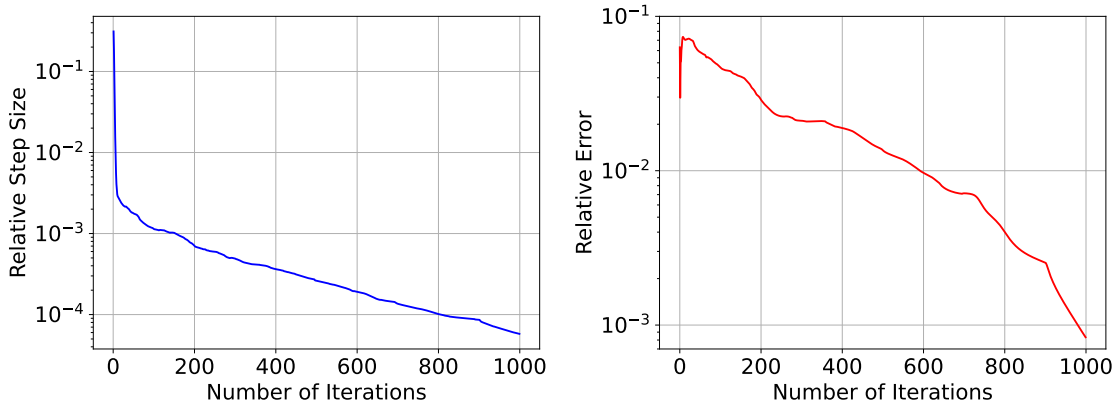


Figure 5.1: Convergence results for Algorithm 1 on the unstructured data. **Left:** The relative step size in (5.1) versus the number of iterations. **Right:** Relative error as defined in (5.2) versus the number of iterations.

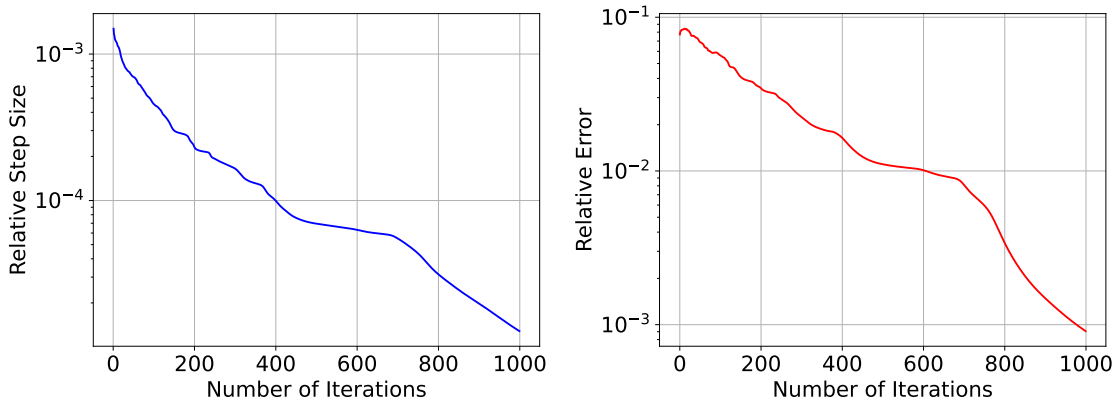


Figure 5.2: Convergence results for Algorithm 2 on the unstructured data. **Left:** The relative step size in (5.1) versus the number of iterations. **Right:** Relative error as defined in (5.2) versus the number of iterations.

Comparing Figure 5.1 with Figure 5.2, we see that Algorithm 1 and 2 behave similarly on the unstructured data if we disregard the first few iterations. The relative step size decreases rapidly in both cases, which indicates that the algorithms converge. For the relative error, we see that both algorithms converge to values close to the linear programming reference. The final approximation of T-GOSPA is 52.6603 for Algorithm 1 and 52.6564 for Algorithm 2. From using a linear programming solver, we get that the exact result is 52.7042, resulting in final relative errors of $8.3361 \cdot 10^{-4}$ (0.0834%) and $9.0801 \cdot 10^{-4}$ (0.0908%) for Algorithm 1 and 2, respectively.

Structured Data

We also run Algorithm 1 and 2 on structured data generated according to the procedure outlined in Section 5.1.1. The parameters used for data generation, as well as the parameters used for the algorithms, are shown in Table 5.4. In Figure 5.3, a visualization of the trajectories comprising the data is shown. The results of this experiment is shown in Figure 5.4 for Algorithm 1, and Figure 5.5 for Algorithm 2.

Table 5.4: Parameters for the experiments on structured data in Section 5.2.1.

Parameter	Value
m_t	14
m_f	2
n_f	1
T	20
r	1
q	0.9
c_s	0.25
N_{ts}	20
$N_{\max}$	100 000
σ	0.01

(a) Data parameters

Parameter	Value
c	0.25
p	1
γ	1
η	10^{-5}
Convergence criteria	Stop after 10^3 iterations

(b) Algorithm parameters

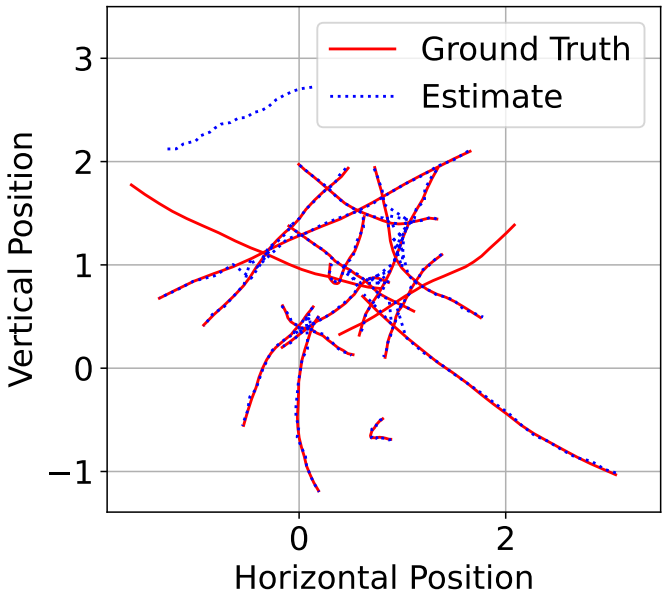


Figure 5.3: Illustration of the structured data ground truth and dummy estimate generated by the procedure in Section 5.1.1. The experiments in this section uses precisely this test case. This shows all time steps at once. Therefore the temporal aspect is not seen (for example, the direction of trajectories is not shown). The parameters used to generate this case is shown in Table 5.4 (a).

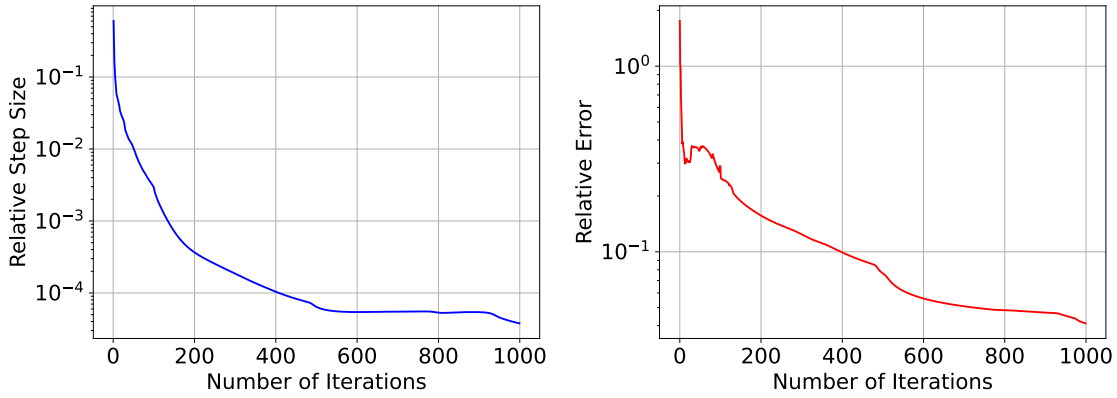


Figure 5.4: Convergence results for Algorithm 1 on the structured data. **Left:** The relative step size in (5.1) versus the number of iterations. **Right:** Relative error as defined in (5.2) versus the number of iterations.

Looking at Figure 5.4, we see that the relative step size decreases more slowly in the structured case compared to the unstructured one for Algorithm 1. This is mirrored in the relative error, where we see that the convergence rate tapers off after about 500 iterations. After 1000 iterations, the obtained value for T-GOSPA is 25.8982, which compared to the the exact solution of 24.8743 from the linear programming solver gives a final relative error of $4.1166 \cdot 10^{-2}$ (4.1166%).

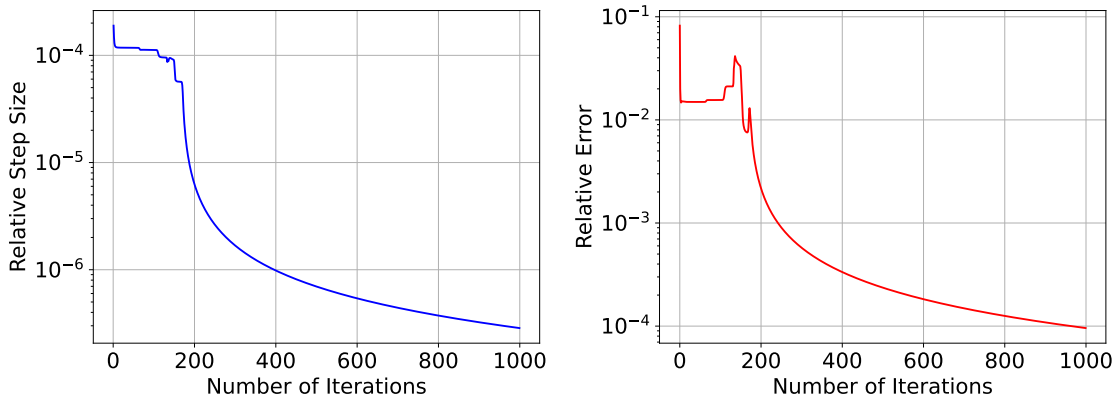


Figure 5.5: Convergence results for Algorithm 2 on the structured data. **Left:** The relative step size in (5.1) versus the number of iterations. **Right:** Relative error as defined in (5.2) versus the number of iterations.

For Algorithm 2, we note from Figure 5.5 that we compared to the unstructured case shown in Figure 5.2 now have a clear transient phase of about 200 iterations where the relative step size plateaus and the objective value fluctuates. After this phase, we get a more well-behaved convergence, but we still see that the rate of convergence decreases with the number of iterations. The final approximation of T-GOSPA is 24.8766 for Algorithm 2, giving a final relative error of $9.5775 \cdot 10^{-5}$ (0.0096%).

Due to Algorithm 2 performing better than Algorithm 1, we focus on Algorithm 2 for the remainder of this chapter.

5.2.2 Regularization Sensitivity

In this section, we investigate the sensitivity of Algorithm 2 with respect to the regularization parameter. Here, the structured data from the previous section is reused and the regularization parameter η is varied. The parameters used for the algorithm is shown in Table 5.5. In Figure 5.6, the result of this experiment is shown.

Table 5.5: Algorithm parameters for the analysis of regularization sensitivity data in Section 5.2.2.

Parameter	Value
c	0.25
p	1
γ	1
η	See Figure 5.6
Convergence criteria	Stop when (5.1) $< 1.5 \cdot 10^{-6}$

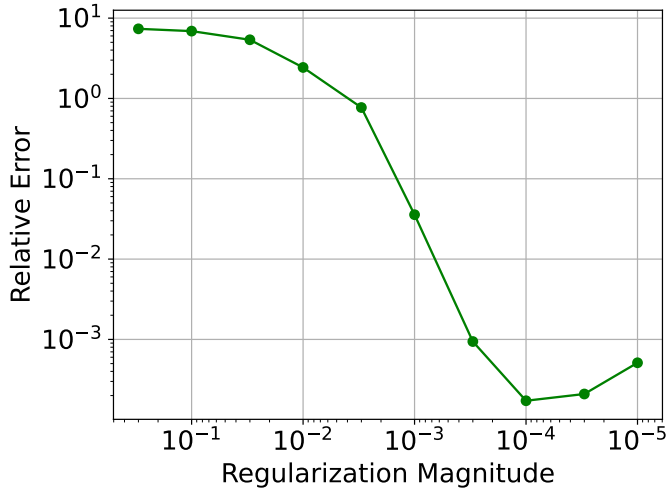


Figure 5.6: Relative error as defined in (5.2) for Algorithm 2 for varying regularization magnitude (η) on the structured data from Section 5.2.1 (and visualized in Figure 5.3).

From Figure 5.6, we see that a decreased magnitude of regularization results in a better estimate of T-GOSPA. This is expected, since it corresponds to letting $\varepsilon \to 0$ in (4.30), and thus making the problem more and more like (4.29), which has the same optimal objective value as the linear programming formulation in (4.2). The takeaway here is that a value of η less than 10^{-3} is needed to get satisfactory solutions to T-GOSPA using Algorithm 2.

The increase in the relative error between $\eta = 10^{-4}$ and $\eta = 10^{-5}$ is confusing from a theoretical standpoint, but not of great concern in practice. Nevertheless, we give some reasons for why this could happen. Firstly, it could be the case that a stopping criteria based on the relative step size is sensitive to changes in the regularization magnitude. Secondly, recall that we do not include the entropy term here, which could affect the results. Lastly, we note that it could be due to the specific structure of this particular problem instance. Further analysis is needed to understand this phenomena.

5.2.3 Running Time Comparisons

In this section we investigate how the running time of Algorithm 2 scales when the size of the input data grows. We consider the case of increasing number of trajectories and increasing number of time steps separately. For reference, a third-party linear programming solver is included. A version of the algorithm which additionally utilizes the GPU for computations is also benchmarked.

Varying the Number of Trajectories

Here we run Algorithm 2 and the linear programming solver on structured data simulated according to the description in Section 5.1.1, with an increasing number of trajectories, and record the wall-clock time and the relative error. For simplicity, we consider cases with $m = n$, where roughly one tenth of the trajectories in the ground truth and estimate are missed and false, respectively. For the different sizes of m, we run 50 instances and compute mean running times. We consider $m \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. The parameters for the experiment are shown in Table 5.6, while results are shown in Figure 5.7.

Table 5.6: Parameters for the running time experiments for increasing number of trajectories in Section 5.2.3. Here $\lfloor x \rceil$ is the rounding of x to the nearest integer.

Parameter	Value
m_t	$m - m_f$
m_f	$\lfloor 0.1 \cdot m \rceil$
n_f	m_f
T	25
r	1
q	0.9
c_s	0.25
N_{ts}	m
$N_{\max}$	$100 \cdot m$
σ	0.01

(a) Data parameters

Parameter	Value
c	0.25
p	1
γ	1
η	10^{-4}
Convergence criteria	Stop when (5.1) $< 10^{-4}$

(b) Algorithm parameters

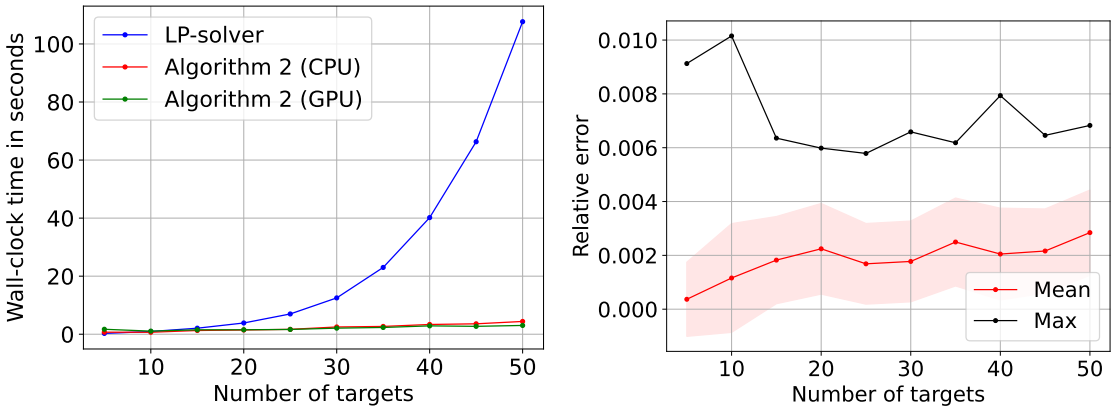


Figure 5.7: Running time results for the tested algorithms for varying number of trajectories (m). Shown values are averages over 50 samples. **Left:** Wall-clock time in seconds until termination for the tested algorithms. **Right:** Relative error as defined in (5.2). The black line shows the maximum relative error over all samples. The red line shows the mean relative error, and the shaded region indicate plus and minus one standard deviation.

In the left part of Figure 5.7, we see that the linear programming solver is faster than our algorithm for small instances. When the size of the input data grows, we see that the linear programming solver scales worse, and for large instances our algorithm is faster. Additionally, we note that for small test cases the CPU version of the algorithm is faster, while for larger cases the GPU version is faster.

In interpreting these results, it is important to remember that the linear programming solver returns an exact solution, while our method is approximate. The approximation error in terms of relative error is shown in the right part of Figure 5.7. We see that in the worst case, we have an error of roughly 1%. Typically though, we see that we have an error of around 0.2%, growing slightly when the number of trajectories increases.

Varying the Number of Time Steps

In this section, we run Algorithm 2 and the linear programming solver on structured data simulated according to the description in Section 5.1.1 with an increasing number of total time steps and record the wall-clock time and the relative error. Similar to the previous test, we here for simplicity consider cases with $m = n$ where one tenth of the trajectories in the ground truth and estimate are missed and false, respectively. For the different sizes of T, we run 50 instances of the problem and compute the mean running times. We consider $T \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. The parameters for the experiment are shown in Table 5.7, while results are shown in Figure 5.8.

Table 5.7: Parameters for the running time experiments for increasing number of time steps in Section 5.2.3.

Parameter	Value
m_t	27
m_f	3
n_f	3
T	See above
r	1
q	0.9
c_s	0.25
N_{ts}	30
$N_{\max}$	3000
σ	0.01

(a) Data parameters

Parameter	Value
c	0.25
p	1
γ	1
η	10^{-4}
Convergence criteria	Stop when (5.1) $< 10^{-4}$

(b) Algorithm parameters

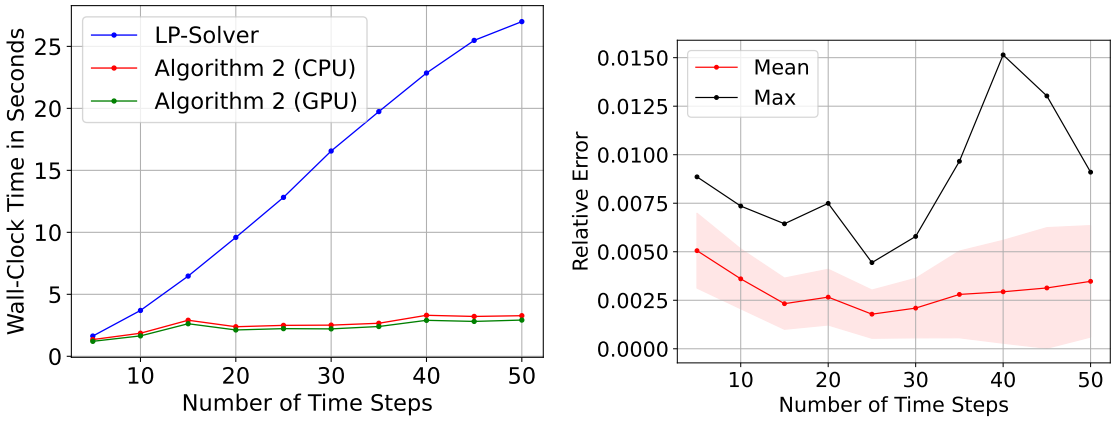


Figure 5.8: Running time results for the tested algorithms for varying number of total time steps. Shown values are averages over 50 samples. **Left:** Wall-clock time in seconds until termination for the tested algorithms. **Right:** Relative error as defined in (5.2). The black line shows the maximum relative error over all samples. The red line shows the mean relative error, and the shaded region indicate plus and minus one standard deviation.

In the left part of Figure 5.8, we see that the running time of both the linear programming solver and our algorithm is roughly linear with respect to the total number of time steps considered. Contrary to the case of increasing number of trajectories, we see that the linear programming solver seems to scale linearly here. This is reasonable, since the number of variables of the linear program increases linearly with T, compared to quadratically with m when $m = n$ in the previous case. Even though both options seem to scale linearly in this case, our algorithm is faster overall. The size of the relative error is similar to before, with a worst case error of 1.5% and mean error of about 0.3%. We also note that using the GPU for computations does not bring any significant acceleration in this case.

Tests on Some Large Problem Instances

To highlight the practical use case of our algorithm more clearly, we run both the linear programming solver and our algorithm on a large instance with $m = n = 75$ and $T = 40$. After more than half an hour, the linear programming solver had not terminated, while our algorithm returned with an approximation after only one minute. Our method also has the advantage of being tunable, since we can adjust the convergence criteria and the regularization parameter to balance the trade-off between accuracy and computational cost.

Lastly, to see if using the GPU can ever bring a significant speed-up, we run an even larger instance with $m = n = 250$ and $T = 25$. This showed that using the GPU was more than three times faster than only using the CPU, indicating that the acceleration from the GPU is most significant when m and n are large.

Overall, the results suggests that our algorithm might be an attractive alternative to using a linear programming solver when the problem instance is large, especially when the number of trajectories in the input data is large. In very large instances of T-GOSPA, an exact solution might be computationally intractable, in which case the approximate solution produced by our algorithm could be of use.

6

Conclusion

To conclude, we have derived two algorithms that we call Algorithm 1 and Algorithm 2 for finding approximate solutions to the relaxed T-GOSPA metric. This was achieved by considering two other formulations of this metric, one for each algorithm. After introducing entropic regularization, each of these formulations gave rise to slightly different iterative schemes for finding approximate solutions to the relaxed T-GOSPA metric.

Theoretically, we have proved that there exist instances when the relaxed T-GOSPA metric is strictly smaller than the ordinary T-GOSPA metric. This disproved a previously conjectured result.

By running the derived algorithms on simulated data, we have shown that both of them converge numerically. Out of the two algorithms, Algorithm 2 performed better in terms of convergence rate. We also showed numerically that the regularization can be made small enough so that adequate approximate solutions can be obtained without making the problem computationally intractable.

Based on a comparison of the running times between Algorithm 2 and a conventional linear programming solver, we found that Algorithm 2 scaled better when the size of the input increased, especially when the number of trajectories in the data increased. It was found that the running time for Algorithm 2 could be further decreased by utilizing a GPU for some calculations. Here it should be noted that while our method is approximate, the approximation error was at most around 1%.

Regarding future studies, it could be of value to theoretically analyze the convergence rate and running time of both algorithms. Additionally, the effect of the initialization could be investigated. Different types of post-processing, such as rounding techniques, could also be investigated to see if performance can be improved. We also believe that the algorithms can be extended to cover a more flexible metric with time-dependent track-switching penalty introduced in [35].

Another matter to be investigated is the use of T-GOSPA computed by our algorithms as a loss function when developing machine learning based multiple target tracking algorithms. Since computational efficiency and approximate methods are central in

machine learning, we believe that future studies in this area could be promising.

Bibliography

[1] Abu Sajana Rahmathullah, Ángel F. García-Fernández, and Lennart Svensson. "Generalized optimal sub-pattern assignment metric". In: *2017 20th International Conference on Information Fusion (Fusion)*. July 2017, pp. 1–8. DOI: 10.23919/ICIF.2017.8009645.

[2] Ángel F. García-Fernández, Abu Sajana Rahmathullah, and Lennart Svensson. "A metric on the space of finite sets of trajectories for evaluation of multi-target tracking algorithms". In: *IEEE Transactions on Signal Processing* 68 (2020), pp. 3917–3928. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2020.3005309.

[3] Marco Cuturi. "Sinkhorn Distances: Lightspeed Computation of Optimal Transportation Distances". In: *Advances in Neural Information Processing Systems 26 (NIPS 2013)*. Advances in Neural Information Processing Systems. Vol. 26. Curran Associates, Inc., June 4, 2013.

[4] Niclas Andréasson, Anton Evgrafov, and Michael Patriksson. *An introduction to continuous optimization: foundations and fundamental algorithms*. Third edition, Edition 3:1, OCLC: 1103765628. Lund: Studentlitteratur, 2016. 492 pp. ISBN: 9789144115290.

[5] Gaspard Monge. *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale, 1781.

[6] Eduardo Fernandes Montesuma, Fred Ngolè Mboula, and Antoine Souloumiac. *Recent Advances in Optimal Transport for Machine Learning*. June 28, 2023.

[7] Cédric Villani. *Optimal Transport*. Red. by M. Berger et al. Vol. 338. Grundlehren der mathematischen Wissenschaften. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 9783540710493. DOI: 10.1007/978-3-540-71050-9.

[8] Leonid Kantorovich. "On the translocation of masses (in Russian)". In: *Proceedings of the USSR Academy of Sciences* (1942).

[9] Filippo Santambrogio. *Optimal transport for applied mathematicians: calculus of variations, PDEs, and modeling*. Progress in nonlinear differential equations and their applications volume 87. Cham Heidelberg New York: Birkhäuser, 2015. 353 pp. ISBN: 9783319208275.

[10] Brendan Pass. *Multi-marginal optimal transport: theory and applications*. Sept. 11, 2014.

[11] Axel Ringh et al. *Graph-structured tensor optimization for nonlinear density control and mean field games*. Sept. 5, 2022.

[12] Jean-David Benamou et al. *Iterative Bregman Projections for Regularized Transportation Problems*. Dec. 16, 2014.

[13] Filip Elvander et al. "Multi-marginal optimal transport using partial information with applications in robust localization and sensor fusion". In: *Signal Processing* 171 (June 1, 2020), p. 107474. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2020.107474.

[14] Isabel Haasler et al. "Multimarginal Optimal Transport with a Tree-Structured Cost and the Schrödinger Bridge Problem". In: *SIAM Journal on Control and Optimization* 59.4 (Jan. 2021), pp. 2428–2453. ISSN: 0363-0129, 1095-7138. DOI: 10.1137/20M1320195.

[15] Paul Knopp and Richard Sinkhorn. "Concerning nonnegative matrices and doubly stochastic matrices." In: *Pacific Journal of Mathematics* 21.2 (Jan. 1967), pp. 343–348. ISSN: 0030-8730.

[16] Bahman Kalantari and Leonid Khachiyan. "On the complexity of nonnegative-matrix scaling". In: *Linear Algebra and its Applications* 240 (June 1, 1996), pp. 87–103. ISSN: 0024-3795. DOI: 10.1016/0024-3795(94)00188-X.

[17] Ángel F. García-Fernández, Lennart Svensson, and Mark R. Morelande. "Multiple target tracking based on sets of trajectories". In: *IEEE Transactions on Aerospace and Electronic Systems* 56.3 (June 2020), pp. 1685–1707. ISSN: 0018-9251, 1557-9603, 2371-9877. DOI: 10.1109/TAES.2019.2921210.

[18] Samuel S. Blackman and Robert Popoli. *Design and analysis of modern tracking systems*. Artech House radar library. Boston: Artech House, 1999. 1230 pp. ISBN: 9781580530064.

[19] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1 (Mar. 1955), pp. 83–97. ISSN: 0028-1441, 1931-9193. DOI: 10.1002/nav.3800020109.

[20] H. W. Kuhn. "Variants of the hungarian method for assignment problems". In: *Naval Research Logistics Quarterly* 3.4 (Dec. 1956), pp. 253–258. ISSN: 0028-1441, 1931-9193. DOI: 10.1002/nav.3800030404.

[21] James Munkres. "Algorithms for the Assignment and Transportation Problems". In: *Journal of the Society for Industrial and Applied Mathematics* 5.1 (1957), pp. 32–38. ISSN: 0368-4245.

[22] Jan T. Lundgren, Mikael Rönnqvist, and Peter Värbrand. *Optimization*. Lund: Studentlitteratur, 2010. 537 pp. ISBN: 9789144053080.

[23] Camilla Carlsson and Hanna Ekelund. "Sequentially connected 2D assignment problems solved using Lagrangian dual methods". In: (2022).

[24] Isabel Haasler et al. "Multi-Marginal Optimal Transport and Probabilistic Graphical Models". In: *IEEE Transactions on Information Theory* 67.7 (July 2021), pp. 4647–4668. ISSN: 1557-9654. DOI: 10.1109/TIT.2021.3077465.

[25] Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. *Accurate Computation of the Log-Sum-Exp and Softmax Functions*. Sept. 8, 2019.

[26] Gabriel Peyré and Marco Cuturi. "Computational Optimal Transport: With Applications to Data Science". In: *Foundations and Trends® in Machine Learning* 11.5 (2019), pp. 355–607. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000073.

[27] Jiaojiao Fan et al. "On the complexity of the optimal transport problem with graph-structured cost". In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. International Conference on Artificial Intelligence and Statistics. PMLR, May 3, 2022, pp. 9147–9165.

[28] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2.

[29] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (Mar. 2020), pp. 261–272. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0686-2.

[30] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[31] John Forrest et al. *coin-or/Clp: Release releases/1.17.9*. Version releases/1.17.9. Oct. 25, 2023. DOI: 10.5281/zenodo.10041272.

[32] John Forrest et al. *coin-or/Cbc: Release releases/2.10.11*. Version releases/2.10.11. Oct. 25, 2023. DOI: 10.5281/zenodo.10041724.

[33] Stuart Mitchell, Michael O'Sullivan, and Iain Dunning. *PuLP: A Linear Programming Toolkit for Python*. 2011.

[34] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (May 2007), pp. 90–95. ISSN: 1558-366X. DOI: 10.1109/MCSE.2007.55.

[35] Ángel F. García-Fernández, Abu Sajana Rahmathullah, and Lennart Svensson. "A time-weighted metric for sets of trajectories to assess multi-object tracking algorithms". In: *2021 IEEE 24th International Conference on Information Fusion (FUSION)*. 2021 IEEE 24th International Conference on Information Fusion (FUSION). Nov. 2021, pp. 1–8. DOI: 10.23919/FUSION49465.2021.9626977.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY