Successive subdivision methods for transport networks of images and other grid graphs[☆]

Michael Muskulus[a,1], J. D. Walsh[b,2]

[a] *Department of Civil and Environmental Engineering, Norwegian University of Science and Technology NTNU, Høgskoleringen 7A, 7491 Trondheim, Norway.*
[b] *School of Mathematics, Georgia Institute of Technology, 686 Cherry Street, Atlanta, GA 30332-0160, USA.*

Abstract

A variety of methods already exist for computing solutions to discrete optimal transport problems, but all of those methods are designed to work on abstract networks. We propose a new optimal transport technique, the method of successive subdivision, or SSD, as a way of leveraging known positional information to reduce computation time and complexity. We then compare the performance of a specific, multigrid-style SSD against some standard methods, and consider the scaling behavior of our software in order to acquire information about the method's average complexity. Our results show that multigrid-style SSD is significantly faster than standard solution methods, especially for large transport problems. We therefore conclude that the SSD approach offers a promising way of using the structure of the embedding space to solve optimal transport problems more efficiently.

Keywords: optimal transport, Monge-Kantorovich, Monge property, minimum cost, network flow, SSD methods

[☆]This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1650044 and by the Research Council of Norway as Project No. 249582/F10. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Research Council of Norway.

Email addresses: michael.muskulus@ntnu.no (Michael Muskulus), jdwalsh03@gatech.edu (J. D. Walsh)

[1] http://www.ntnu.edu/employees/michael.muskulus. Tel.: +47 73593113.

[2] http://people.math.gatech.edu/~jwalsh35. Tel.: +1 404-894-6441.

1. Introduction

1.1. Optimal transport

A transport problem begins with a system of weights, both positive ("sources") and negative ("sinks"), whose sum is zero. These weights can be moved along various routes, but each route has a given cost. A transport plan is a list of weights corresponding to the possible routes. A plan that correctly distributes the weight from the sources to the sinks, balancing the problem, is said to be *feasible*. Each transport plan has an associated total cost, equal to the the weight moved along a given route times the cost incurred during its transport, summed over every route in use. The optimal transport solution is a transport plan of minimum cost.

Today, optimal transport theory is recognized as a vitally important area of mathematics. Practical applications have been found in fields ranging from civil engineering [1] to diagnostic medicine [2, 3, 4]. Optimal transport is used in the earth sciences [5], meteorology [6], and occasionally even cosmology [7]. It comes up in economic theory [8, 9, 10, 11] and quantitative finance [12, 13]. It is repeatedly cited as a technique for solving image registration and image warping problems [14, 15, 16]. And of course, it is commonly applied in mathematics and physics [17, 18, 19, 20]. However, numerical optimal transport is still maturing, with many unsolved computational problems. One particular obstacle is the difficulty of solving large, complex systems.

1.2. Complexity and size

Discrete optimal transport problems can be solved using linear programming techniques or methods based on those used for partial differential equations. The network simplex method and Orlin's algorithm are examples of the former; see [21] and [22], respectively. Auction algorithms are an example of the latter; see [23]. With respect to the total number of sources and sinks, such algorithms scale cubically or worse. For example, Orlin's 1993 algorithm, described as having the "best worst-case" scaling behavior, is $\mathcal{O}(N^4 \log N)$ when solving fully connected transport problems with N sources and sinks [24]. Other, slightly more efficient algorithms exist (as we discuss later), but none scale better than $\mathcal{O}(N^3)$.

This scaling behavior drastically limits the size of problems that can be computed under reasonable time constraints. Unfortunately, the networks involved can be extremely large. The Stanford Large Network Dataset Collection offers data for social networks containing as many as 65 million nodes

and 1.8 billion edges [25]. These are *reduced* data sets. The original version of that network contined 117 million nodes and 2.5 billion directed edges [26]. Solving optimal transport problems for such large graphs is currently very difficult, if not impossible.

1.3. Reducing complexity by successive subdivision

To reduce the time and complexity required to solve such large problems, we look to a so-far neglected source: positional information. The methods for solving discrete problems were designed for abstract networks. Because optimal transport problems involve minimizing some distance, whether real or artificial, most of them can be embedding in some physical space. This is inherently true of grid graphs, which appear naturally in images and many discretized data sets.

We propose to use this positional information to separate large transport problems into smaller subproblems. Because of the worse-than cubic performance of existing solvers, subdividing transport problems can greatly decrease total solution time, and even improve the overall scaling rate.

Many algorithms exist for dividing a physical space, but we argue that the best option is to apply a multigrid method: constructing a hierarchy of discretizations, and solving repeatedly from coarsest to finest. Multigrid methods, while generally used to solve partial differential equations, are directly applicable to discrete problems where positional information matters [27].

At each of the discrete levels, we can use the solution network to subdivide the graph into subgraphs. Each subgraph, when more finely discretized, can again be solved and subdivided. We argue that this process of *successive subdivision*, or *SSD*, offers a substantial improvement in speed and complexity over existing methods. As we show, solutions generated using successive subdivision are near-optimal. They can be used effectively to better initialize exact solution methods, greatly reducing total computation time, or presented as-is, when a quick estimate is desired.

1.4. Our approach

We begin by describing known information: the discrete optimal transport problem, features of the most commonly used solution methods, and the available positional information. Then we describe our proposed SSD method, discuss its properties, and outline the impact of the discretization hierarchy on the problem's geometry. Finally, we present our numerical results:

design principles, time and memory comparisons with common initialization methods, and a comparison of scaling with and without subdivision.

2. Background

2.1. The optimal transport problem

The discrete optimal transport program was originally developed by Gaspard Monge [28], but it was first formulated as a linear programming problem by Frank Hitchcock in [29]. Because transport problems embedded in space are generally fully-connected, we focus on the dense transport problem:
We are given a set of source weights $\{a_i\}_{i=1}^{n}$ and sink weights $\{b_j\}_{j=1}^{m}$ such that $a_i > 0$ for all $i \in \mathbb{N}_n = \{1, \ldots, n\}$ and $b_j > 0$ for all $j \in \mathbb{N}_m$. We are also given a set of costs $\{c_{ij} \mid i \in \mathbb{N}_n,\, j \in \mathbb{N}_m\}$ such that $c_{ij} > 0$ for all i and j. Our goal is to

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij}x_{ij} & \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = a_i & \forall i \in \{1, 2, \ldots, n\} = \mathbb{N}_n \\
& \sum_{i=1}^{n} x_{ij} = b_j & \forall j \in \mathbb{N}_m \\
& \sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j & \\
& x_{ij} \geq 0 & \forall i,\, j \text{ where } i \in \mathbb{N}_n \text{ and } j \in \mathbb{N}_m
\end{aligned}$$

We will assume without loss of generality that $m \leq n$.

2.2. Standard discrete transport solvers

The standard linear programming methods use no positional information. They consider the linear programming problem as a graph with V vertices (nodes) and E edges (arcs). The graph is then solved using some minimum flow algorithm.

Consider how this applies to the discrete transport problem we have defined. We have m sources and n sinks, with $m \leq n$, so

$$V = m + n \leq 2n \qquad \text{and } E = m \cdot n \leq n^2. \tag{1}$$

Suppose m and n are comparable in size. Because we are interested in complexity, we can presume that $V = n$ and $E = n^2$, disregarding constant values.

For the discrete optimal transport problem in linear programming form, the best minimum flow methods have complexity $\mathcal{O}(n^3)$. Examples of methods showing this scaling behavior are Orlin's 2013 algorithm with complexity $\mathcal{O}(VE)$ [22] and the push-relabel minimum cost flow algorithm, with complexity $\mathcal{O}(V^3)$ [30].

The most commonly used linear programming algorithm, the network simplex method, scales as $\mathcal{O}(VE \log V \log VC)$, where C is the maximum possible flow over the arcs [31]. Assuming C is a fixed constant, and thus can be diregarded, this is equal to $\mathcal{O}(n^3 \log^2 n)$.

Auction algorithms use a completely different approach, a relaxation method derived from those used to solve partial differential equations. When applied to discrete optimal transport problems with integer coefficients, the auction algorithm has complexity $\mathcal{O}(VE \log VC)$ [23], where C is defined in the same way it is for the network simplex. Disregarding C gives us a complexity of $\mathcal{O}(n^3 \log n)$.

Each of the algorithms discussed above, require approximately four variables per arc. Let us consider this in the context of the limitations of existing linear programming solvers. SoPlex, a well-known open-source linear solver, can manipulate up to 2 million variables [32, 33].[3] This is borne out by a paper using discrete optimal transport to approximate optimal coupling, published in the same year as the MCF documentation, that describes how "with a 50×50 grid one obtains a linear program with about 6 million variables" [32]. A 50×50 grid corresponds to a graph with $n = m = 1250$, which would have approximately 1.5 million arcs.

The capabilities of SoPlex are fairly typical. According to documentation published in 2000, one optimal-transport specific solver, MCF, quickly handles "several thousand nodes and several million arcs" [35]. While the auction algorithm requires more customized software, tests by the authors reveal comparable limitations.

These limitations are all comparable because they result from the sheer size of the transport problem. To generalize this in the context of our desired grid discretization, suppose we have a $W \times W$ grid. Assuming $m = n$, then

[3]Commercial solvers may be able to handle more; for example, see [34].

we must have $n = W^2/2$. This gives us W^2 vertices and $W^4/4$ arcs. Thus, it is no surprise that these linear solvers have the limits described. Assuming that each arc requires continuous storage of four variables, and that each variable takes up 8 bytes of space (the standard `double`), a 200×200 grid would require nearly 12 gigabytes.

2.3. Available positional information

As we know, the discrete transport problem is not simply an abstract graph. It is embedded in some metric space, even though none of the above methods use that information. In order to consider how best to use the grid, we need to establish what information we have. Then we can consider how best to apply it.

2.3.1. The Monge property

In his 1781 memoir, Monge recognizes an important property: an optimal transport network embedded in a metric space will not contain any intersecting paths.[4]. In short, if A goes to B and X to Y, then the route from A will never intersect the route from X. This is a consequence of the triangle inequality; if such an intersection occurred, a simple transformation of paths reduces the total distance traveled, contradicting the assertion of optimality.

The Monge property is a significant limitation on the number of possible solutions; this is particularly true in two dimensions, because every crossing is an intersection. Hence, for some transport problems it may be profitable to initialize using feasible systems satisfying the Monge property.

2.3.2. The Ham Sandwich theorem

The Ham Sandwich Theorem states that, given N finite measures over an N-dimensional space, it is possible to divide all of them in half simultaneously (with respect to total measure) using a single $(N-1)$-dimensional hyperplane [36, 37]. An optimal transport contains two sets of positive finite measure: the set of all sources and the set of all sinks. Suppose the problem is *feasible*: that is, the total weight of the sources equals the total weight

[4] "Lorsque le transport du déblai se fait de manière que la somme des produits des molécules par l'espace parcouru est un *minimum*, les routes de deux points quelconques A & B, ne doivent pas se couper entre leurs extrémités, car la somme $Ab + Ba$, des routes qui se coupent, est toujours plus grande que la somme $Aa + Bb$, de celles qui ne se coupent pas." p. 667 [28]

of the sinks. If the problem is embedded in a metric space of two or more dimensions, it is possible to subdivide the problem via a ham sandwich cut, creating two smaller, feasible optimal transport problems.

Because each of the two solutions is optimal, neither contains an intersection. Since the solutions are separated by the ham sandwich cut, the two cannot intersect with each other. Therefore the union cannot contain an intersection, and so the combined solution satisfies the Monge property. Similarly, because the individual solutions are feasible, their union must be feasible.

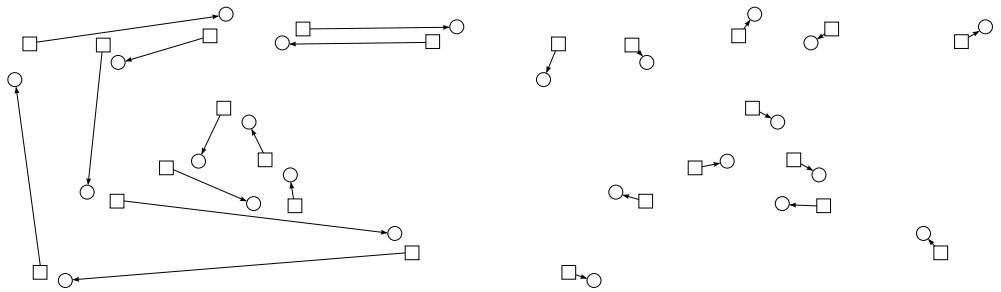
Since each of the two optimal transport problems is feasible and satisfies the Monge property, the original problem can be subdivided further using additional ham sandwich cuts, and each new subproblem will satisfy the Monge property. In fact, the cuts involved need not be ham sandwich cuts. Any sequence of cuts that subdivides the original into feasible transport problems could be used, such as a sequence of subdivisions based on convex hulls. No matter how the subdivisions are made, so long as the subproblems are separated by some algebraic surface, the union of their solutions will be feasible and satisfy the Monge property.

Limits of the Monge property / Ham Sandwich combination. The Monge property is a necessary condition for any solution of an optimal transport problem embedded in a metric space, but it is not a sufficient condition. There is no guarantee that initializing via the Monge property will lead to an optimal solution. In fact, the solution could be arbitrarily far from optimal. To see how this is so, consider the Graham scan.

The Graham scan algorithm, developed in 1972, is the best known algorithm using what we call successive subdivision [38]. Implementation is reasonably straight-forward: it is often taught to upper-level undergraduates as a method for solving the "ghosts and ghostbusters problem." Each source and sink has weight 1, which reduces the transport problem to an *assignment*, or matching, problem. Through cuts made using a series of convex hulls, the Graham scan subdivides the problem until each feasible area is reduced to a single pair of points, a source and a sink, which are then matched. Overall, the algorithm is $O(n^2 \log n)$, where n is the total number of sources and sinks in our assignment problem.

While the Graham scan does guarantee a feasible solution satisfying the Monge property, the results can be arbitrarily far from the optimal cost. See Figure 1 for a specific example. This is because the Graham scan algorithm

does not consider transport cost.



(a) Graham scan result (b) Optimal transport plan

Figure 1: Graham scan result satisfying Monge property with severely suboptimal costs

As it turns out, none of the existing subdivision methods take cost optimization into account. Because our primary goal is computation of the optimal cost, this requires a change of focus. We propose applying successive subdivision in a way that best takes cost minimization into account, even if the results of that method occasionally violate the Monge property. With that in mind, let us consider how multigrid methods might be applied to successive subdivision.

2.3.3. Multigrid method

Multigrid is a technique used to improve the effectiveness of numerical methods for differential equations. It is not in itself a differential equation solver; rather, it is a method of scaling problems to take advantage of differences in the behavior of either the solver or the system of equations. Generally speaking, multigrid is a way of determining an approximate solution on a coarse-grained network, followed by repeated grid refinements and further approximations, converging on an acceptable numerical solution [27].

When the grid is coarsened, viable subdivisions appear that may not have been immediately apparent at finer resolutions. Take, for instance, the center block of Figure 2. It is clear that the upper sink and source form a balanced transport network, with their -20 and $+20$ weights. So do the lower sink and source, with weights of -4 and $+4$. Thus, the center block can be subdivided by a horizontal line across the middle, creating two balanced transport networks. This implies that the left-hand block can also be subdivided by a horizontal line across the middle, even though the network balance is less immediately apparent.

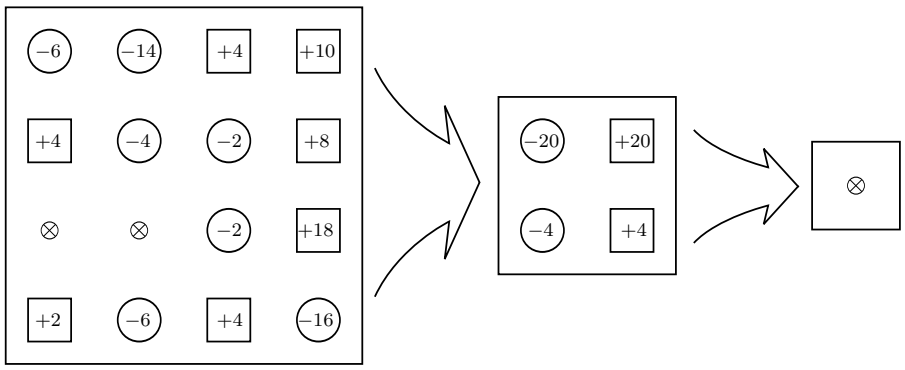


Figure 2: Typical binning progression

If the optimal transport problem splits into multiple components, as Figure 2 does, then viable subdivisions are natural and obvious. But even connected solutions can be subdivided using multigrid methods. A single vertex whose weight is sent along multiple routes allows for a subdivision *through* that point in order to send some of its weight on each side of the partition.

Multigrid subdivision through a vertex is accomplished by treating the embedding as the quotient space of a pseudometric space and the vertex as multiple vertices in a similarity class of identically-located points. (This rationale underlies the other SSD algorithms allowing cuts across vertices, such as the ham sandwich algorithm of Lo, Matoušek, and Steiger; see [39].)

Because the multigrid method is not, itself, a solver, using it frees us to apply any desired optimal transport algorithm. The algorithm minimizes the cost of each individual subproblem, thereby shifting the focus of the successive subdivision method from the preservation of the Monge property to the optimization of the cost. The end result is a near-optimal solution.

3. Mathematics

3.1. Description of the SSD algorithm

Multigrid-style SSD lends itself easily to an optimal transport (OT) algorithm that is both recursive and parallelizable. The general form of the algorithm we propose is the function `RECURSE`, shown in Figure 3. `RECURSE` takes two arguments: a `region` of the underlying space and a non-negative integer indicating the `level`, or number of binning steps, from the original

system. (Thus, `level` 0 indicates the original problem, `level` 1 is the first coarse-grained problem, and so on.)

Starting with the original (finest) level, successively coarsen the grid `level` times, until the desired starting coarse-grain discretization is reached.

The function `RECURSE` can be initialized at any `level` (including zero), but the initial `region` must be large enough to cover the entire occupied space of the original problem after being refined `level` times. In practice, the ideal initial `level` is one less than the smallest integer required to reduce the entire problem to a single point. (The initial `region` is that point, usually the origin.) If this initialization is applied to a transport problem embedded in $\mathbb{R}^d$, the first use of `CONSTRUCT` generates a refined transport problem of size 2^d, the smallest with the potential for subdivision.

`RECURSE`:

1. `CONSTRUCT` the OT problem by refining the previous `level`'s `region`
2. `SOLVE` the refined OT problem for solution `S`
3. if `level` > 0:
   (a) `PARTITION` the solution `S` into a set of regions `R` $= \{$`r`$\}$
   (b) for each region `r` in `R`:
   Let $\mathtt{S_r}$ = `RECURSE` on region `r` at `level` $- 1$
   (c) `COMBINE` the sub-solutions $\{\mathtt{S_r}\}$ into (new) solution `S`
4. return `S`

Figure 3: Recursive algorithm for multigrid-style SSD

Note that each iteration in step 3(b) is completely independent of the others. Because of this, every call to `RECURSE` in 3(b) can be implemented as a separate thread, as can the sub-calls from those `RECURSE`s, up to the limits of the system. In other words, the general structure of `RECURSE` echoes that of a parallelizable `mergesort`, with the same benefits.

The subroutines of `RECURSE` are highly implementation dependent:

- `CONSTRUCT` generates the optimal transport problem associated with the current `level` by performing one refinement of the given `region` (which is associated with the previous `level`).
- `SOLVE` is the chosen optimal transport solver.

- `PARTITION` separates the optimal transport solution into disjoint subregions, taking into account the refinement issues mentioned above.
- `COMBINE` builds the union of the set of optimal transport solutions.

Many of the subroutines can be specialized or parallelized in order to take advantage of the `RECURSE` function's structure. `COMBINE` can be implemented as a parallel `merge` algorithm. The `SOLVE` routine can also be optimized, particularly for intermediate refinements where an optimal solution may be less desirable than a quickly-generated approximation.

3.2. Properties of SSD algorithm

We now consider the standard mathematical properties of the SSD algorithm in Figure 3. For purposes of the arguments below, assume the original transport problem is size $n = \left(2^d\right)^k$. Hence, solving will involve k `levels` with graphs of size: 2^d, 2^{2d}, 2^{3d}, $\ldots$, $2^{(k-1)d}$, n.

3.2.1. Termination

The termination of the SSD algorithm is straight-forward: Suppose the routine `SOLVE` terminates in finite time. Then `RECURSE` will complete each of its k `levels` in finite time. Therefore, the SSD algorithm itself termines in finite time.

3.2.2. Scaling: worst, best, and average case

Because the multigrid SSD method `RECURSE` is dependent on the performance of the underlying solver, the worst and best case scaling estimates are dependent on the complexity of `SOLVE`. For now, assume that `SOLVE` has complexity $S(n)$. This will allow us to easily discuss the relative impacts of multigrid SSD on the various linear programming methods. For simplicity, we will also assume that the scaling of the `PARTITION` and `COMBINE` routines is considered to be bounded above by some complexity $P(n)$, where $\mathcal{O}(n) \leq P(n) \leq S(n)$.

Worst case scaling. Suppose each attempt at subdividing the coarse grained problem returns an *atomic solution*; that is, a solution that has not been subdivided. In other words, at each `level`, `PARTITION` returns a single region $R = S$. This implies that we will use multigrid SSD to recursively solve problems of complexity

$$S(1) + \ldots + S(n/8) + S(n/4) + S(n/2) + S(n). \tag{2}$$

Therefore, the complexity for the worst-case is

$$\mathcal{O}(S(n) \log S(n)). \tag{3}$$

Best case scaling. Consider a discrete transport problem with n vertices that has been split by SSD using ham sandwich cuts. This results in $n/2$ subproblems of constant difficulty. In fact, the difficulty is constant because each problem consists of a single matched pair. Thus, solving the transport problems is $\mathcal{O}(n)$. In practice this means that the time taken by `PARTITION` and `COMBINE` dominates, and the complexity of multigrid SSD becomes $\mathcal{O}(P(n))$.

Average case scaling. Not surprisingly, obtaining a theoretical estimate of average complexity would be difficult. In particular, even if we knew the average complexity of the `SOLVE` routine, we cannot apply the master theorem, because it requires recurrence relations that can be written as

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1,\, b > 1, \text{ both constant}$$

For the typical problem using multigrid-style SSD, both a and b will be functions dependent on n.

Still, we can make a few observations. The potential improvement in complexity is highly dependent on the size and number of subdivisions possible. We know, by the Ham Sandwich theorem, that we can always subdivide the discrete transport problem. Hence, it is always possible to achieve something resembling the best case above, where the complexity is dominated by `PARTITION` and `COMBINE`.

However, recall that our goal is not simply fast computation. Otherwise, we would simply use the Graham scan algorithm. Since our goal is an appropriate balance between the complexity of our computations and the optimality of our solutions, we can choose the desired average complexity, by deciding to favor speed or accuracy. We discuss how this choice can impact average complexity in Section 4.3.

3.3. Geometric complications

As mentioned in Section 2.3.3, subdivsions created by using the multigrid method will result in near-optimal solutions. It is important to consider the ways in which the solutions generated by multigrid SSD come to lose their optimality.

By assumption, the `SOLVE` routine generates an optimal solution for each individual subproblem. The only way that the combined result can be less than optimal is if some set of arcs between subproblems has a lower total cost than the arcs chosen within those subproblems.

Assuming our cost function is a norm, this external lower-cost arc set cannot occur when every subproblem is contained in a convex set, provided those sets are pairwise disjoint. Hence, given such an arrangement, individually optimal sub-solutions combine to give us a solution that is optimal overall. However, if we have overlapping subproblems or concavities in subproblems, the individual sub-solutions may not give us an optimal combined result. If we further refine these sub-solutions, the result may violate the Monge property.

The refinement process of the multigrid method introduces an additional complexity. Simply put, points move. When we coarsen the grid, we combine a group of points into a single, central point. A solution that is optimal at the central point may not be optimal when the weights are distributed to the locations of the original points. This is a direct consequence of points which satisfy the strict triangle inequality:

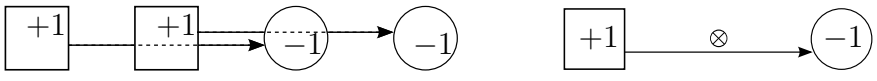
$$|x + y| < |x| + |y|. \tag{4}$$

For a cost function that is a norm, this only occurs when a sub-solution contains nonparallel arcs, and the subproblem is further refined.

Taking our cost to be the Euclidean distance, we consider each possibility in turn: overlapping regions, nonconvex regions, and nonparallel arcs.

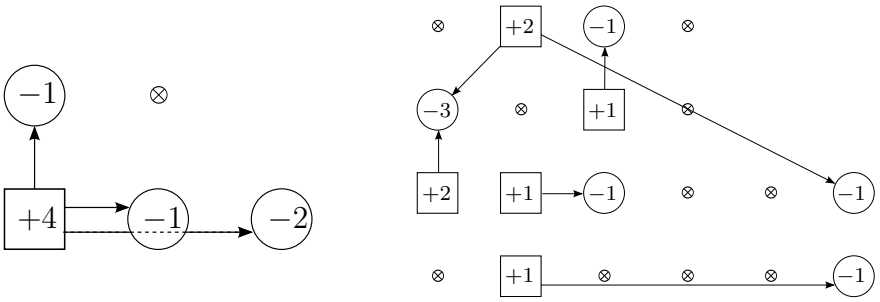
Overlapping regions. A valid optimal transport solution can involve overlapping regions, such as those shown in Figure 4. This can happen when two non-zero regions have overlapping routes, as in the figure on the top, or when a single region crosses a space that has zero weight in the coarse grid, as shown in the bottom figure. (A zero-weight region is self-feasible, but may include non-zero weights when refined.) While such regions may be disjoint on the coarse grid, they cannot be physically separated because there is no guarantee the regions will remain disjoint under the refined grid.

Nonconvex regions. If a region incorporates a concave area, then the solution of the refined subproblem may introduce diagonal arcs that cross routes in other regions. An example is shown in Figure 5.



(a) Routes overlap (b) Crosses zero region

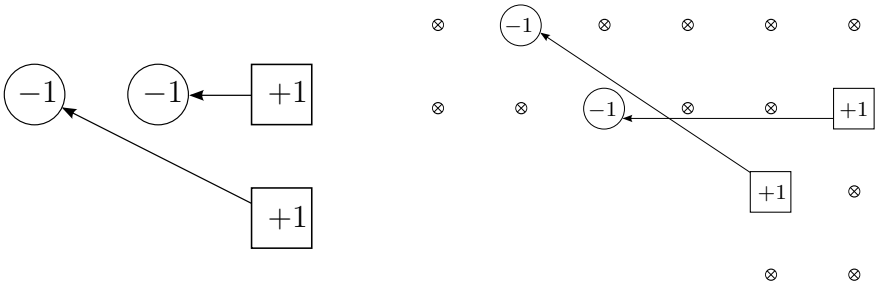
Figure 4: Overlapping disjoint regions in an optimal transport plan



(a) Initially separable (b) Refined regions intersect

Figure 5: Regions that separate in the coarse grid, but intersect in the refined grid

Nonparallel arcs. If adjacent subproblems contain arcs that are not parallel to the coordinate system, refining the grid can introduce changes in the position and angles of such arcs such that physical separation is no longer possible. An example is shown in Figure 6. The system on the left satisfies the Monge property (and is, in fact, optimal), but the system on the right, generated by solving the two refined subproblems separately, does not.



(a) Initially separable (b) Refined routes intersect

Figure 6: Routes that separate in the coarse grid, but intersect in the refined grid

3.3.1. Incorporating geometric concerns into multigrid-style SSD

Having identified the key ways in which optimality might fail, it is worth considering how best to incorporate those into an effective multigrid-style SSD solver. The resolution of these issues is a function of the `PARTITION` routine.

There are two primary ways to deal with geometric concerns arising from regions of questionable structure. The first is to combine all such regions, ensuring that the refined systems will satisfy the Monge property. The second is to ignore potential crossings and any resulting violations of the Monge property, assuming that the refined systems will still be "nearly" Monge and hence that the combined solution will remain close to optimal.

This is the crux of the decision described in Section 3.2.2 under "Average case scaling." Simply put, we have two options:

- The more questionable regions we combine, the larger the individual subproblems become, and the closer the scaling is to the worst case estimate of $\mathcal{O}(S(n) \log S(n))$. However, the final cost computed should be close to optimal. We refer to this as *accurate* multigrid-style SSD.

- The fewer questionable regions we combine, the smaller the individual subproblems are, and the closer the scaling becomes to the best case estimate $\mathcal{O}(P(n))$. However, the final cost computed is likely to be significantly less optimal. We refer to this as *fast* multigrid-style SSD.

We compare these two approaches in Section 4.

4. Numerics

4.1. Design

For our numerical tests, we assumed that our transport problems were embedded in $\mathbb{R}^2$, and the transport cost was determined by the Euclidean distance. With that cost function, the optimal solution may not be unique. We define each problem on the plane. Given a size d, our code generates an assignment problem on a $2^d \times 2^d$ grid. Thus, for any given problem we have 2^{2d-1} sources and an equal number of sinks. The complete bipartite network digraph has $V = 4^d$ nodes and $E = 4^{2d-1}$ arcs.

We implemented the multigrid-style SSD software in C++, compiled with `gcc` 4.1.2, which supports most of the 2003 language standard. Our multigrid-style SSD code was made to run on a single processor (though it does take

advantage of recursion). Because computer hardware changes so quickly, we have focused on direct comparisons between programs run on the same hardware under near-identical conditions. Thus, specific numbers may vary but our comparisons of efficiency should remain otherwise relevant with future computing advances.

Accurate vs. fast SSD. We had two primary considerations in implementing the `PARTITION` function: satisfying the Monge property, in order to measure accuracy, and effective subdivision of the original problem, in order to gauge speed. To manage this, we created two versions of `PARTITION` function: one for accuracy and one for speed. In every other respect, our two SSD implementations are identical.

The auction algorithm for optimal transport problems treats the system as an expanded assignment problem, so it is a simple matter to subdivide through vertices and split components into multiple pieces. However, we chose not to use this capability for either implementation, instead treating each component as atomic. For our fast multigrid-style SSD, these were the components we used to create our partitions.

For accurate multigrid-style SSD, we expanded these components in two ways. When a component crossed over another, we combined the two components into one. When a component included a diagonal (nonparallel) arc, we considered the entire rectangle enclosing the arc as part of the component, combining in any other component that happened to intersect with that space. We did not expand convex regions of components, but only because in our experiments such regions rarely led to violations of the Monge property when the solutions were combined.

Two solvers, three initializers. In general, differences in storage methods make it difficult to directly compare our methods to existing solvers. Positional techniques can always use significantly less memory, with some corresponding penalty to time. Abstract network techniques cannot. We worked around the issue by coding our own versions of existing methods.

In order to compare multigrid-style SSD to existing solvers, we wrote a version of one of the fundamental linear programming techniques, the network simplex method described by Chvátal [21], and one of the most efficient relaxation methods, the auction algorithm of Bertsekas and Castañón [40, 23].

Because one of our goals is to consider the effectiveness of multigrid-style SSD as an initialization method, we also wrote software for three of

the most commonly used linear programming initialization methods: the northwest corner method, the least cost method, and Vogel's approximation method [41, 42]. Together with multigrid-style SSD, these give us four ways to initialize the network simplex.

We took advantage of one positional feature for all our methods: dynamic cost computation. When using an abstract network structure, every single arc must be given an explicit cost that is either loaded into memory or read from a file. When given an embedding and a cost (distance) metric, it is sufficient to give each vertex an explicit position and compute costs as needed using the metric. Because the underlying structure of the transport problem is a complete bipartite graph, most systems have far fewer vertices than arcs, so tracking vertex positions requires far less storage than tracking arc costs. This allows us to greatly reduce the memory requirements for nearly all of our solution and initialization methods.

The down-side of dynamic cost computation is the need to perform complex operations on-the-fly, often calculating with the same values over and over again. This created problems for Vogel's approximation. Because Vogel's method is so dependent on easy repeated access to an array of costs, dynamic cost calculation proved highly inefficient (each initialization would require over 24 hours to complete). As a result, all computations involving Vogel's approach are made with a "fat" Vogel's method that keeps a stored array of cost values.

4.2. Comparison with existing methods

To test their effectiveness at initialization, we put our two multigrid SSD implementations head-to-head against the three network initialization methods — northwest corner, least cost, and Vogel's — to see how well they performed at network approximation and initialization. We also compared six different solvers. Five of them were network simplex methods: our two SSD methods and the three standard techniques. The sixth was the auction method applied directly to the original problem.

Using our generator software, we created ten random assignment problems of size $2^7 \times 2^7$. These problems consist of random permutations of 2^{13} sinks and 2^{13} sources, with equal probability of a sink or source at all locations. No attempt was made to control or structure placement, but random seed variables were stored, allowing us to quickly rebuild the same matrix for head-to-head method comparisons.

The resulting graphs have over 16 thousand vertices and 67 million arcs, so solving them by standard linear programming methods would require more than 268 million variables. Because of the dynamic cost computation, described in Section 4.1, and careful crafting of the "fat" Vogel's method, we were able to keep the number of variables well below this size.

We initialized and solved each problem using the network simplex method with each of our two multigrid SSD implementations and the three standard network initialization methods. As the auction method is a satisfactory solver in its own right, we also applied that algorithm directly to each of the original problems. The median of our ten results were used for our comparisons.

Initialization. Our first consideration was the time and memory requirements for network initialization. These are summarized in Table 1.[5] Error values shown are the percentage by which the estimated network's cost exceeded the optimal cost. For comparison, our network simplex solver required 2.858 MB of memory to solve the initialized problems and the auction solver used 2.514 MB. The memory requirements for all methods except multigrid SSD are independent of the problem generated, so most memory values are constant.

Initialization method	Time	Memory	Cost error
Northwest corner	0.00 sec	1.857 MB	1895.0%
Least cost	1350.52 sec	1.857 MB	53.2%
Vogel's approximation	1820.65 sec	513.3 MB	65.9%
Multigrid SSD (accurate)	18.79 sec	1.502 MB	5.4%
Multigrid SSD (fast)	0.20 sec	1.456 MB	31.5%

Table 1: Time and memory use by initializer

Solving. Our tests also address the question of whether or not multigrid-style SSD effectively solves the original problem, and if so, which of our two SSD approaches would perform better. We considered this from two perspectives: the total solution time and the time and iterations required to obtain the exact cost from the initial network. These results are shown in Table 2.[6]

[5]The auction solver, not listed, is self-initializing.

[6]Auction iterations are not shown because the complexity of a network simplex iteration (a "pivot") differs so substantially from that of an auction iteration (a "bid"). While the slowest network simplex solver required fewer than 5 million pivot steps, the fastest auction solution required nearly a *billion* bids.

Method	Iterations	Solving time	Total time
Auction	—	165.60 sec	165.60 sec
Northwest corner	4704386	298.07 sec	298.07 sec
Least cost	769662	76.80 sec	1425.71 sec
Vogel's approximation	818061	81.43 sec	1903.57 sec
Multigrid SSD (accurate)	389872	52.81 sec	75.71 sec
Multigrid SSD (fast)	596907	66.34 sec	66.57 sec

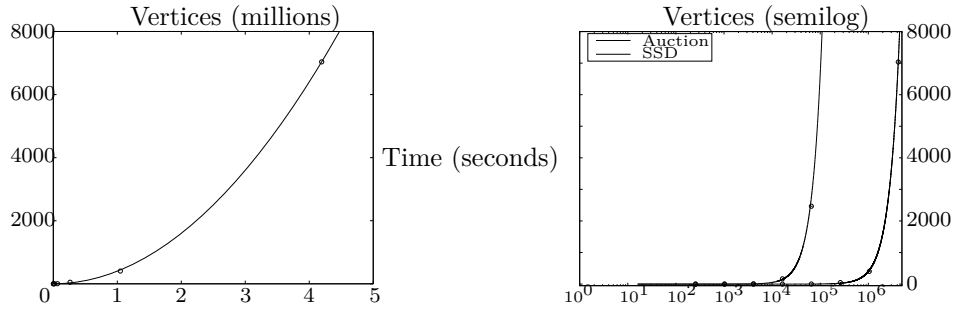
Table 2: Iterations and solving time by method

As can be seen, the SSD approach solves the problem in one-third the time required by the highly-efficient auction algorithm. Although this may not seem like a significant improvement, the difference in solution time grows rapidly when problem size increases, as shown in the next section.

4.3. Scaling

To evaluate the scaling behavior of the fast multigrid-style SSD as problem size increases, we ran time trials on problems with size up to $n = 2^{22}$, running ten random trials and taking the medians. When a trial ran under one second, instead of ten single trials we used aggregated results: ten sets of 100 or 1000 consecutive trials, as appropriate. Even then, problems with fewer than 1000 vertices were solved too quickly for accurate results.

Scaling of SSD with respect to time. Figure 7 shows the time and size relationship from our data in two plots.



(a) Multigrid SSD (linear scale) (b) SSD vs. auction (semilog)

Figure 7: Time complexity for multigrid SSD approximation (by number of vertices)

The left-hand plot, Figure 7(a), shows that our results scaled linearly. Numerically, our fast multigrid-style SSD implementation appears to scale with time complexity $O(n^2)$, with a constant around $c \approx 10^{-10}$ seconds. Hence, even for low values of n, subdividing the optimal transport problem significantly reduces the auction method's computation time. Because we have developed multigrid-style SSD as an initialization technique, beyond a certain point the final solver — in our case the network simplex method — is likely to dominate asymptotic complexity. Nonetheless, these improvements in speed and scaling demonstrate the viability of an SSD approach.

The right-hand plot, Figure 7(b), compares our multigrid-style SSD approximation to the fastest exact solver we identified, the auction method for optimal transport, using a semilogarithmic scale. Our implementation of the auction method appears to scale with time complexity $O(n^2 \log n)$, with a constant around $c \approx 10^{-8}$.

Scaling of SSD with respect to memory. Because our `SOLVE` function's auction algorithm dominates our memory requirements, we theorized that memory use would remain linear. That seems to be the case, as the graph in Figure 8 illustrates. While memory use is somewhat dependent on network structure, every test we ran gave a nearly identical linear relationship, regardless of size: approximately $9n \times 10^{-5}$ megabytes of memory. (Note that the constant is approximately half that of the standard auction implementation.) This is not surprising, considering the way dynamic cost computation controls memory use in the implementation of the SSD method.

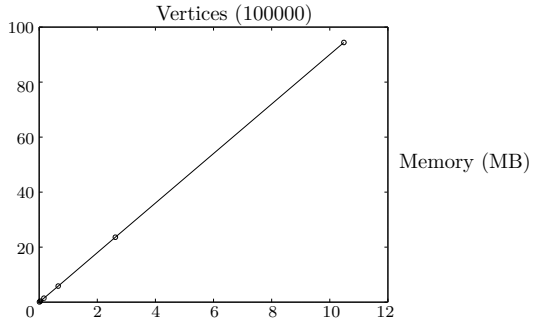


Figure 8: Memory use for multigrid SSD approximation (by number of vertices)

5. Conclusion

We have proposed a new approach to the discrete optimal transport problem, multigrid SSD, which uses positional information in tandem with existing, more abstract, network graph techniques. We have described our numerical results, which compare favorably with existing techniques. Furthermore, they suggest that the average complexity of SSD methods is significantly better than that of standard techniques. Our results suggest that multigrid-style SSD is a promising way of using the structure of the embedding space to solve optimal transport problems more efficiently.

References

[1] M. Muskulus, Distance-based methods for structural damage detection in offshore wind turbine installations, in: M. Singh, R. B. K. N. Rao, J. P. Liyanage (Eds.), Proceedings of the 24th Congress on Condition Monitoring and Diagnostics Engineering Management, COMADEM International, COMADEM International, U.K., 2011.

[2] S. Alagoz, B. B. Alagoz, A distance-based dynamical transition analysis of time series signals and application to biological systems, J. Biol. Phys. 38 (2012) 293–303. doi:10.1007/s10867-011-9248-2.

[3] D. Lombardi, Inverse problems for tumor growth modeling, Ph.D. thesis, Institut de Mathématiques de Bordeaux (September 2011).

[4] M. Muskulus, A. M. Slats, P. J. Sterk, S. Verduyn-Lunel, Fluctuations and determinism of respiratory impedance in asthma and chronic obstructive pulmonary disease, Journal of Applied Physiology 109 (2010) 1582–1591.

[5] U. Ehret, E. Zehe, Series distance — an intuitive metric to quantify hydrograph similarity in terms of occurrence, amplitude and timing of hydrological events, Hydrology and Earth System Sciences 15 (2011) 877–896.

[6] M. J. P. Cullen, R. J. Purser, An extended Lagrangian theory of semi-geostrophic frontogenesis, Journal of the Atmospheric Sciences 41 (1984) 1477–1497.

[7] U. Frisch, S. Matarrese, R. Mohayaee, A. Sobolevski, A reconstruction of the initial conditions of the universe by optimal mass transportation, Nature 417 (6886) (2002) 260–262.

[8] G. Carlier, A general existence result for the principal-agent problem with adverse selection, Journal of Mathematical Economics 35 (2001) 129–150.

[9] G. Carlier, F. Santambrogio, A continuous theory of traffic congestion and Wardrop equilibria, Journal of Mathematical Sciences 181 (2012) 792–804.

[10] A. Figalli, Y.-H. Kim, R. J. McCann, When is multi-dimensional screening a convex program?, Journal of Economic Theory 146 (2011) 454–478.

[11] A. Galichon, B. Salanié, Matching with trade-offs: Revealed preferences over competing characteristics, Columbia University Department of Economics Discussion Paper Series (0910-14).

[12] M. Beiglböck, P. Henry-Labordére, F. Penkner, Model-independent bounds for option prices: a mass transport approach, Finance and Stochastics 17 (2013) 477–501.

[13] A. Galichon, P. Henry-Labordére, N. Touzi, A stochastic control approach to no-arbitrage bounds given marginals, with an application to loopback options, The Annals of Applied Probability 24 (2014) 312–336.

[14] R. Chartrand, B. Wohlberg, K. R. Vixie, E. M. Bollt, A gradient descent solution to the Monge-Kantorovich problem, Applied Mathematical Sciences (Ruse) 3 (2009) 1071–1080.

[15] S. Haker, A. Tannenbaum, Optimal mass transport and image registration, in: IEEE Workshop on Variational and Level Set Methods in Computer Vision: Proceedings: 13 July, 2001, Vancouver, Canada, IEEE Computer Society, Los Alamitos, California, 2001, pp. 29–36.

[16] L. Zhu, Y. Yang, S. Haker, A. Tannenbaum, An image morphing technique based on optimal mass preserving mapping, IEEE Transactions on Image Processing 16 (6) (2007) 1481–1495.

[17] L. A. Caffarelli, X. Cabré, Fully nonlinear elliptic equations, Vol. 43 of American Mathematical Society Colloquium Publications, American Mathematical Society, Providence, R.I., 1995.

[18] L. Chacón, G. L. Delzanno, J. M. Finn, Robust, multidimensional mesh-motion based on Monge-Kantorovich equidistribution, Journal of Computational Physics 230 (2011) 87–103.

[19] R. Jordan, D. Kinderlehrer, F. Otto, The variational formulation of the Fokker-Planck equation, SIAM Journal on Mathematical Analysis 29 (1998) 1–17.

[20] F. Otto, The geometry of dissipative evolution equations: the porous medium equation, Communications in Partial Differential Equations 26 (2001) 101–174.

[21] V. Chvátal, Linear Programming, W.H. Freeman, New York, 1983.

[22] J. B. Orlin, Max flows in $\mathcal{O}(nm)$ time, or better, in: Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC '13, ACM, New York, NY, USA, 2013, pp. 765–774.

[23] D. P. Bertsekas, D. A. Castañón, The auction algorithm for the transportation problem, Annals of Operations Research 20 (1) (1989) 67–96.

[24] J. B. Orlin, A faster strongly polynomial minimum cost flow algorithm, Operations Research 41 (1993) 338–350.

[25] J. Leskovec, A. Krevl, SNAP Datasets: Stanford large network dataset collection, `http://snap.stanford.edu/data` (2014).

[26] The Internet Archive, Friendster social network dataset: Friends, `http://archive.org/details/friendster-dataset-201107` (2011).

[27] U. Trottenberg, C. Oosterlee, A. Schüller, Multigrid, Academic Press, Cambridge, Massachusetts, 2001.

[28] G. Monge, Mémoire sur la théorie des déblais et des remblais, in: Histoire de l'Académie Royale des Sciences de Paris, avec les Mémoires de Mathématique et de Physique pour la même année, Académie des sciences (France)., 1781, pp. 666–704, in French.

[29] F. L. Hitchcock, The distribution of a product from several sources to numerous localities, Journal of Mathematics and Physics 20 (1941) 224–230.

[30] A. V. Goldberg, R. E. Tarjan, Finding minimum-cost circulations by canceling negative cycles, Journal of the Association for Computing Machinery 36 (1989) 873–886.

[31] R. E. Tarjan, Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm, Mathematical Programming 78 (1997) 169–177.

[32] L. Rüschendorf, L. Uckelmann, Numerical and analytical results for the transportation problem of Monge-Kantorovich, Metrika 51 (3) (2000) 245–258.

[33] Zuse Institute Berlin, SoPlex: The Sequential object-oriented simPlex, `http://soplex.zib.de`, accessed: 2016-02-11.

[34] International Business Machines Corporation, Guidelines for estimating CPLEX memory requirements based on problem size, `http://www-01.ibm.com/support/docview.wss?uid=swg21399933`, accessed: 2016-02-11.

[35] Zuse Institute Berlin, MCF homepage, `http://www.zib.de/opt-long_projects/Software/Mcf`, accessed: 2015-11-25.

[36] W. D. Smith, N. C. Wormald, Geometric separator theorems and applications, in: K. Kelly (Ed.), Proceedings 39th Annual Symposium on Foundations of Computer Science: November 8–11, 1998 Palo Alto, California, The Institute of Electrical and Electronics Engineers, Inc., Los Alamitos, California, 1998, pp. 232–243. `doi:10.1109/SFCS.1998.743449`.

[37] A. H. Stone, J. W. Tukey, Generalized "sandwich" theorems, Duke Mathematical Journal 9 (2) (1942) 356–359. `doi:10.1215/S0012-7094-42-00925-6`.

[38] R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, Information processing letters 1 (4) (1972) 132–133.

[39] C.-Y. Lo, J. Matoušek, W. Steiger, Algorithms for ham-sandwich cuts, Discrete & Computational Geometry 11 (4) (1994) 433–452. doi:10.1007/BF02574017.
URL http://dx.doi.org/10.1007/BF02574017

[40] D. P. Bertsekas, A new algorithm for the assignment problem, Mathematical Programming 21 (1) (1981) 152–171.

[41] G. B. Dantzig, M. N. Thapa, Linear Programming 1: Introduction, Springer Series in Operations Research, Springer-Verlag, New York, 1963.

[42] N. V. Reinfeld, W. R. Vogel, Mathematical Programming, Prentice-Hall, Englewood Cliffs, N.J., 1958.