Branching optimization for transporting tasks in the plane *

Jaroslaw Piersa†
piersa.jaroslaw@gmail.com
2015-05-03

Abstract

In this work we propose a simulated annealing based search algorithm for optimal ramified or branched trans-
portation route in \( \mathbb{R}^2 \). The cost function of the transport network allows the system to merge small roads into larger
highways, carry the goods together and separate them near the destination. As a result the transportation route
has multiple branching-in and -out points, to some extent similar to nerve system of the tree leaf. The problem is
known to be NP-complete even for finite set of branching points and metric spaces.

Keywords: Transport network, ramification algorithm, simulated annealing optimization.

1 Introduction

In this paper we investigate transport optimization problem, in which the roads are allowed to merge into larger
highways. Consider two sets: sources and destinations (sinks). The sources produce some amount of goods, that needs
to be transported to one of the destinations from \( D \). The obvious solution is to build a separate road for every pair of
points, between which we need to transport something, but in most cases it is more efficient to build a single highway
combining few different transports.

The above specification resembles a bit the famous Gauss problem to find the shortest road map connecting angles
of equilateral triangle. The cost of the road is two-fold. On one hand it should scale linearly with its length. On the
other hand it should scale sub-linearly with the transport intensity.

Assume for a while, that there is only one source node, so that the final network resembles nerves of the tree
leaf. At first glance the problem might seem to be a variation of minimum spanning tree [1]. Note, that the whole
transportation network must be connected (after forgetting directions of the edges). Immediately we see, that the
problem degenerates to the geometric Steiner tree problem, that is minimum spanning tree with set of mandatory
vertices to be included (sources and destinations) and set of possible vertices (the whole plane). The Steiner tree
problem, even for finite sets of possible vertices and metric distance function is known to be NP-complete[2].

Xia discussed this problem in the context of single source cases (where it was clearly a leaf) [3, 4]. Matuszak et al.
described a bayesian-network based algorithm, which worked fine assuming limited size of output and input sets [5].

In [6, 7] we proposed a randomized search algorithm for the optimal solution. In this work we refine this algorithm,
inspect its theoretical backgrouds using Markov chain Monte Carlo (MCMC) theory and generalise it, so it can be
applied to ‘indistinguishable goods’ variant.

Most of the paper is devoted to construction of the solution space, transition rules, and their theoretical properties.
We provide more precise specification in Section 2 then the algorithm itself in Section 3. The Section 4 is devoted
to theoretical explanation, while Secion 5 addresses its technical aspects. The results are presented in Section 6. On
top of that, we introduce a possibility to redirect the flows between destinations in Section 7. Finally the paper is
concluded with Section 8.

2 Problem specification

Let \( S \) denote the set of sources, and \( D \) the set of destination points in Euclidean plane \( \mathbb{R}^2 \). Each source has its own
production rate, and each destination has its consumption rate. We represent it by transport setup \( t_{sd} \geq 0, s \in S, d \in D \),
which indicates how much of goods from \( s \) is transported to \( t \). The total production rate of sources \( S \) must be equal
to the total consumption rate of \( D \).

Let \((a,b)\) denote a straight road from \( a \) to \( b \). Length of the road will be denoted by \( \text{len}(a,b) = |(a,b)| = l \), the
transported quantity \( t((a,b)) = q \) goods.

*This is author’s draft version. The final paper is available at International Journal on Artificial Intelligence Tools webpage.
†Mickiewicza 5, 07415 Olszewo Burki, Poland.
We assign a concave cost function of transport of goods through a road segment \((a, b)\): 

\[
\text{Cost}((a, b)) = \text{Cost}(\text{len}(a, b)), t((a, b)) = q^\alpha \cdot l,
\]

where \(0 \leq \alpha \leq 1\) is cost exponent. The formula does not account for unfavorable ground (wetlands or mountains).

We represent the solution network as a directed graph \(G = (V, E)\) and transport function \(t : E \rightarrow \mathbb{R}_0\). We require: 

\[
\forall s \in S \sum_{v \in Adj(s)} t_{sv} = \sum_{d \in D} t^0_{sd} \quad \text{for sources and symmetrically for destinations}: \forall d \in D \sum_{v \in Adj(d)} t_{vd} = \sum_{s \in S} t^0_{sd}.
\]

Now the cost of the solution is defined as:

\[
\text{Cost}(G, t) = \sum_{e \in E} \text{Cost}(t(e), \text{len}(e)).
\]

**Degenerate case** For \(\alpha = 1\) road merging cannot bring about any savings due to triangle inequality. For \(\alpha > 1\) the optimal path would split into tiny rivers, which merge at the destination. While this case might be interesting per se, it is not realistic and hard to rationalize, so throughout the rest of the paper we assume \(0 \leq \alpha < 1\).

### 3 Search algorithm

The input of the algorithm consist of: set of sources \(S\), set of destinations \(D\), and transport setup \(t^0\).

We refer to the \(S \cup D\) as terminal nodes, as the path road structure terminates in them. The set \(V \setminus (S \cup D)\) consists of inner nodes, as none of them can be a leaf of the network. The inner nodes serve as 'crossroads', where two roads can merge into a highway. While the most real world crossings are incident to four streets, the optimal route of the inner nodes is: the total transport entering the node must be equal to the total transport leaving the node. For the source nodes \(v \in S\):

\[
\sum_{w: v \rightarrow w \in E} t(u \rightarrow v) = \sum_{w: v \rightarrow w \in E} t(v \rightarrow w),
\]

that is: the total transport entering the node \(v\) must be equal to the total transport leaving \(v\).

If we define the energy (cost) of the solution as

\[
E(G, t) := \text{Cost}(G, t),
\]

then, the problem can now be seen as minimization of this function over the space of all directed graphs including \(S \cup D\). The pair \((G, t)\) is allowed to evolve under the energy-driven simulated annealing-like dynamic in order to reach the global minimum or at least a \(G\) which is close enough to the optimal solution.

The algorithm is carried out as follows:

1. begin with starting network \(G^1, t^1\), assign iteration number \(i := 1\),
2. repeat
   (a) create \((G', t')\) from \((G^i, t^i)\) by performing a random perturbation (to be discussed in Sec. 5),
   (b) search \((G', t')\) for possible constraint violations (anomalies); if found any, dissolve them,
   (c) calculate \(\text{Cost}(G', t')\) and \(\Delta \text{Cost} = \text{Cost}(G', t') - \text{Cost}(G^i, t^i)\),
   (d) if \(\Delta \text{Cost} < 0\) then accept the better solution, otherwise assign with probability

\[
\mathbb{P}(G^{i+1}, t^{i+1}) := (G', t') = \exp(-\beta \Delta \text{Cost}),
\]

and with complimentary probability leave the current solution

\[
(G^{i+1}, t^{i+1}) := (G^i, t^i),
\]

(7)
(e) update rates and parameters,
(f) assign $i := i + 1$ (timesteps),

until the result is satisfactory,

3. return the obtained network $G^i, t^i$.

While the outline seems simple and resembles Boltzmann machines, some of the subroutines, in particular random perturbation, are delicate and require a closer look.

4 Theoretical foundations

Below we present sketch of proof of the algorithm. It consists of three steps.

- the algorithm is a simulated annealing variation for graph space,
- the transition between solutions is a Markov chain, which is irreducible,
- the Markov chain is aperiodic.

4.1 Definition of the Markov chain

The transition between a pair of solutions via a random perturbation can be seen as a random walk over the possible solution space $\Sigma = \{G = (\mathcal{V}, \mathcal{E}), t : \mathcal{V} \to \mathbb{N} : G \text{ is directed}, \mathcal{S} \subset \mathcal{V}, \mathcal{D} \subset \mathcal{V}, t \text{ satisfies the constraints Eq. 3, 4 and 5} \}$.

Graph perturbations, which were mentioned in the second step of the algorithm, indicate a possibility of transition from state $(\sigma_1, t_1) \in \Sigma$ to another $(\sigma_2, t_2) \in \Sigma$. Upon freezing $\beta$ (simulation annealing parameter), we obtain constant in time transition probabilities $P((\sigma_1, t_1) \rightarrow (\sigma_2, t_2))$, for $(\sigma_1, t_1), (\sigma_2, t_2) \in \Sigma$.

4.2 Irreducibility

![Figure 1: Main steps of reaching the solution. (a) starting setup, (b) paths divided into short segments, (c) merging of two unconnected paths, (d) final step of merging, (e) next path being merged into to the existing network, (f) final solution.](image)

This is the most challenging step, which is also reflected in Section 5 where we describe the technical implementation.
The Markov chain is \textit{irreducible} if for every two states \( \sigma, \tau \), the chain can reach \( \tau \) from \( \sigma \) in finite number of steps with non-zero probability. We need to ensure that for every two possible solutions \((\sigma_1, t_1), (\sigma_2, t_2)\) it is possible to obtain \((\sigma_2, t_2)\) from \((\sigma_1, t_1)\) in a number of steps.

We need the following theorem from \cite{2}.

\textbf{Theorem} For every inner node in optimal geometrical Steiner tree, its degree is equal to three, and angles between incident edges are equal to \(2\pi/3\).

For every inner node \( v \in V \setminus (S \cup D) \) we only need to allow \( \text{deg}_{in}(v) \in \{1, 2\} \) and symmetrically \( \text{deg}_{out}(v) \in \{1, 2\} \).

Irreducibility also requires backward transitions from \((\sigma_2, t_2)\) to \((\sigma_1, t_1)\) to be possible. We satisfy this requirement by organizing the perturbations into pairwisely reversible pairs:

\begin{itemize}
  \item vertex merging and vertex splitting
    \begin{itemize}
      \item \( X \)-shaped,
      \item \( Y \)-shaped,
      \item \( \lambda \)-shaped,
    \end{itemize}
  \item branch forward jump and branch backward jump,
    \begin{itemize}
      \item output branch \( (Y \)-shaped),
      \item input branch \( \lambda \)-shaped,
    \end{itemize}
  \item vertex insertion and vertex deletion.
\end{itemize}

On top of that we also introduce self-reversing alteration:

\begin{itemize}
  \item vertex motion.
\end{itemize}

All of the listed alterations must preserve consistency requirements Eq. \ref{eq:consistency} and transport setup \( t^0 \). Their detailed implementations are provided in Sec. \ref{sec:implementations}.

To conclude this step we need to show, that it is possible to reach every possible (or at least \textit{every reasonable}) solution in \( \Sigma \).

Let us begin with the solution, where every source is connected with a direct road with target destination. Suppose, that we need to obtain a solution where routes \( s_1 \rightarrow d_1 \) and \( s_2 \rightarrow d_2 \) are merged. We may (theoretically) proceed as follows:

\begin{itemize}
  \item insert a number of inner vertices into the roads \( s_1 \rightarrow d_1 \) and \( s_2 \rightarrow d_2 \),
  \item merge two inner vertices from both roads \( (X \)-shaped),
  \item continue the merging for remaining inner nodes \( (Y \)-shaped and \( \lambda \)-shaped),
  \item move the merged vertices to desired positions,
  \item if needed, merge more additional road(s) \( s_3 \rightarrow d_3 \) into the obtained structure.
\end{itemize}

The process is illustrated in Fig\ref{fig:process}. By iterating this process, we should be able to reach most of the solution networks within the space \( \Sigma \). This process may fail to reach some graphs, where one or more terminating node are isolated, but these cases are not valid members of solution space (they violate transport setup \( t^0 \)). Since \( \Sigma \) is a set of solutions instead of simply ‘graphs’, it does not affect the requirements.

Since we use only reversible alterations, the above process is reversible. Given a solution \( \sigma \) we can turn it into directly connected sources and destinations by

\begin{itemize}
  \item applying a sequence of vertex splits,
  \item deleting inner nodes from the routes.
\end{itemize}

Given a pair of solutions \( \sigma_1 \) and \( \sigma_2 \), (assuming they have the same transport setup) we can

\begin{itemize}
  \item morph \( \sigma_1 \) into \( \sigma' \) consisting of direct connections by using splits and deletions,
  \item turn \( \sigma' \) onto \( \sigma_2 \) by applying insertions and merges.
\end{itemize}

Hence, it is possible to reach \( \sigma_1 \) starting from \( \sigma_2 \), if both of the solutions share the same \( t^0 \).
4.3 Aperiodicity

Let us begin with a brief revision of Markov chain properties: Let $\sigma$ be a state of Markov chain.

- Let us define set $\text{returns}(\sigma) = \{i \in \mathbb{N} : \text{the chain can reach state } \sigma \text{ starting from } \sigma \text{ in } i \text{ steps} \}$,
- define Period of state $\sigma$ as $\text{period}(\sigma) = \text{GCD}(\text{returns}(\sigma))$,
- in irreducible Markov chain period is the same for every state $\sigma$,
- irreducible Markov chain is aperiodic, if the period of its states is equal to unity.
- in order to show aperiodicity it is enough to show $\exists_{\sigma \in \Sigma} P(\sigma \rightarrow \sigma) > 0$.

We show, that the Markov chain defined by the search algorithm is aperiodic. Let $\sigma_1 := \text{the optimal solution.}$ Let us choose any perturbation $f$, which alters the solution $\sigma$ into $f(\sigma) \neq \sigma$. $f$ can be selected from the list of available modifications with some probability $p > 0$. It is always possible to introduce a new vertex and move its position, so there always at least one $f$ available.

Let $\text{Cost}(\sigma) = C_1$ and $\text{Cost}(f(\sigma)) = C_2$, and $C_1 < C_2$ (because $\sigma$ is optimal). The probability of acceptance is from Eq $[7]$

$$P(\sigma := f(\sigma)) = \exp(-\beta(C_2 - C_1)) < 1,$$

so also probability of staying in $\sigma$

$$P(\sigma := \sigma) = 1 - \exp(-\beta(C_2 - C_1)) > 0.$$

Hence we have probability of staying in the same state:

$$P(\sigma \rightarrow \sigma) \geq p(1 - \exp(-\beta(C_2 - C_1))) > 0.$$  \hspace{1cm} (11)

So the chain is aperiodic.

To summarize:

- the solution space $\Sigma$ and transitions constitute a (huge) Markov Chain, which is stationary, irreducible and aperiodic,
- taken together, the algorithm described in Section 3 can be seen as special case of Metropolis-Hastings algorithm known as simulated annealing,
- the convergence follows from the properties of Metropolis-Hastings algorithm$[9]$.

4.4 Further remarks

A critical reader should now raise a couple of questions concerning our assumptions.

- The graph is a discrete object, but positions of vertices are real, which makes $\Sigma$ infinite.

This is true. We can, however, restrict the positions to a discrete lattice or argue, that in their computer representation they can only be set to floating-point-representable values, which makes the solution space incredibly huge but finite.

- The assumption that the optimal network shares similarities to Steiner tree problem may not be true.

The restrictions concerning the input and output degrees may indeed not hold (though, results of Xia $[3]$ and Matuszak $[5]$ indicate otherwise), but nodes with degree (input or output $\deg = \deg_{in} + \deg_{out}$) $\deg \geq 3$ can approximated by $\deg - 1$ nodes with $\deg = 2$.

The angle between the incident edges was not used in our construction. In fact for $\alpha \neq 0$ the optimal solution does not have to preserve the angle $2\pi/3$.

5 Alterations of the solution network

In this section we describe in details the random alterations of the solution graph, which were introduced in Sections 3 and 4.
5.1 Initial network

The algorithm starts from \((G^1, t^1)\), that need to be constructed manually. This step is illustrated in panel (b) in Fig. 1. The needed paths can be determined from transport input — we build the road if \(t^0_{s,d} > 0\) The path follows linear segment connecting points \(s\) and \(d\).

5.2 Random perturbations

During step 2b. of the algorithm (create \((G', t')\) from \((G^i, t^i)\) by performing random perturbation), we pick one of following operations:

- merge vertices (create a branch),
- separate vertex (remove a branch),
- move a branch forward / backward the path,
- create / delete vertex in the middle of existing edge,
- adjust position of vertex.

They are described below.

5.3 Merging vertices

This modification is responsible for merging a pair of vertices into a single one. Two consecutive merging operations create a highway, which can carry the transport to the destination at lower cost. The pseudo-code follows:

1. select two vertices \(p_1, p_2\), such that: \(p_1 \neq p_2, p_1, p_2 \notin \mathcal{S} \cup \mathcal{D}\) and there exists no directed path between \(p_1\) and \(p_2\),
2. if both \(p_1\) and \(p_2\) have three different in-neighbors or three different out-neighbors, then merging is not possible (formally: if \(|\text{Adj}^{\text{in}}(p_1) \cup \text{Adj}^{\text{in}}(p_2)| \geq 3\) or \(|\text{Adj}^{\text{out}}(p_1) \cup \text{Adj}^{\text{out}}(p_2)| \geq 3\),
3. create \(p_3\) positioned at \(x_j := (x_j + x_j)/2\), for all dimensions \((j = 1, 2)\) (on the line between \(p_1\) and \(p_2\)),
4. for every parent \(p\) of \(p_1\) or \(p_2\), link \(p \rightarrow p_3\) and redirect the transfer \(p \rightarrow p_1\) or \(p \rightarrow p_2\) to the new vertex \(p \rightarrow p_3\),
5. for every child \(c\) of \(p_1\) or \(p_2\), link \(p_3 \rightarrow c\) and redirect the transfer \(p_1 \rightarrow c\) or \(p_2 \rightarrow c\) to the new vertex \(p_3 \rightarrow c\),
6. delete \(p_1\) and \(p_2\) from the network (with all the adjacent edges).

The sketch of the merging can be seen in Fig. 2(a) to (d).

5.4 Splitting vertices

Splitting of branches is a reverse process to merging. The main issue is that one must be careful not to redirect the transport through a wrong node. The pseudo-code follows:

1. choose the vertex to be split \(p \notin \{\mathcal{S} \cup \mathcal{D}\}\),
2. let \(r_1, r_2\) be the parents of \(p\) and \(c_1, c_2\) its children, if there is only one parent and only one child, then the operation is pointless (there is nothing to split),
3. check if there is unique path between $p$ and every single source $s \in S$ and destination $d \in D$, since the representation cannot distinguish which path between nodes carries the transport; if non-unique paths are found, then skip the modification,

4. add a pair of new nodes to $V$: $p_1, p_2$ located at $p_i = \frac{p + c_i + 5r_i + 5r_2}{3}$, for $i = 1, 2$,

5. redistribute the flows $r_1 \rightarrow p \rightarrow c_1$, $r_2 \rightarrow p \rightarrow c_1$, $r_1 \rightarrow p \rightarrow c_2$ and $r_2 \rightarrow p \rightarrow c_2$ so they flow through either $p_1$ or $p_2$; check that the new routing does not violate path existence constraints; if it does, then rollback the modification,

6. delete $p$ from the network (with adjacent edges).

In Figure 3 (c) to (h) we present the splitting procedure. For X-shape, the step 5 requires local optimization problem to be solved. We need to find $t((p_1, c_1))$, $t((p_1, c_2))$, $t((p_2, c_1))$ and $t((p_2, c_2))$, which minimize the cost. Since the cost function is monotone one of these transfers should either fully saturate one of the children $c_1$ or $c_2$ (and spend the rest by sending it to the remaining child) or fully deplete one of the parents $r_1$ or $r_2$ (and take the rest from the other one). Immediately we have, that at least one of transfers is equal to zero: $t((p_1, c_1)) = 0$, $t((p_1, c_2)) = 0$, $t((p_2, c_1)) = 0$ or $t((p_2, c_2)) = 0$. This leaves four possibilities to be checked (we omit the edge braces () for compact notation):

<table>
<thead>
<tr>
<th>$t(p_1, c_1)$</th>
<th>$t(p_1, c_2)$</th>
<th>$t(p_2, c_1)$</th>
<th>$t(p_2, c_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$t(r_1, p_3)$</td>
<td>$t(p_3, c_1)$</td>
</tr>
<tr>
<td>2</td>
<td>$t(r_1, p_3)$</td>
<td>0</td>
<td>rest</td>
</tr>
<tr>
<td>3</td>
<td>$t(p_3, c_1)$</td>
<td>rest</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>rest</td>
<td>$t(p_3, c_1)$</td>
<td>$t(r_2, p_3)$</td>
</tr>
</tbody>
</table>

Note, that the splitting can be pushed forward ($\lambda$) or backwards ($\gamma$) until it reaches source or destination vertex, where the the routes become completely separated.

5.5 Vertex motions

The above points use heuristics, that a group of paths clustered in one area should be merged into a larger highway. Vertex motions, while do not change the graph structure directly, can bring a pair of vertices closer to each other and more likely to be merged.

The subroutine follows

1. select a point $x \in \mathbb{R}^2$ located nearby one of the vertices of the graph $G$, for instance randomly select $u \in V$ and then sample a point from the 2d-ball centered at $u$: $B_2(u, r) = \{ x \in \mathbb{R}^2, |x, u| \leq r \},$

2. find $v \in V \setminus (S \cup D)$, which is the closest to $x$,

3. for $v$ and its neighbors (unless they are in $S$ or $D$) perform

$$w := \gamma w + (1 - \gamma)x,$$

(12)

where for instance $\gamma = i/T$ (current timestep divided by number of all the timesteps).

The restriction of distance of point $x$ makes the motion less chaotic.

The modification is presented in Fig. 4. The presented version is clearly inspired by Kohonen self-organizing maps [10]. Other version of the routine can be simplified to applying a random motion to the vertex. If the computing power allows it, one can include both.

1. select a random vertex $v \in V \setminus (S \cup D),$

2. sample $x, y \sim_{iid} N(0, \sigma^2),$

3. assign $v := v + (x, y)$ (translate $v$ by vector $(x, y)$).
5.6 Branching shifts

This modification allows the branching to jump backward or forward the path. We impose the restriction, that after modification, the graph should still have maximum number of two children and two parents per vertex. There are two branch types, children or parents. They are equivalent, so we discuss only the former. Children branching in turn are divided into:

1. (backward via parent $r_1$) redirect $r_1 \rightarrow v \rightarrow c_i$ to $r_1 \rightarrow c_i$,
2. (backward via parent $r_2$) redirect $r_2 \rightarrow v \rightarrow c_i$ to $r_2 \rightarrow c_i$,
3. (forward via child $c_1$) redirect $v \rightarrow c_2$ to $v \rightarrow c_1 \rightarrow c_2$,
4. (forward via child $c_2$) redirect $v \rightarrow c_1$ to $v \rightarrow c_2 \rightarrow c_1$.

The pseudo-code of children-branching shift follows:

1. randomly select a node $v \in V \setminus (S \cup D)$,
2. let $c_1, c_2$ denote children of $v$; $r_1, r_2$ parents of $v$ (possibly only $c_1$ or only $r_1$ exists),
3. verify possibility of shifts:
   
   (a) if $\text{deg}_\text{out}(v) = 1$ (only one child), then shift is not possible,
   
   (b) if $\text{deg}_\text{out}(r_i) = 2$ backward shift is not possible via parent $r_i$ (otherwise it would result in $\text{deg}_\text{out}(r_i) = 3$),
   
   (c) if $\text{deg}_\text{out}(c_2) = 2$ forward shift is not possible via child $c_2$ (otherwise $c_2$ would have 3 children),
   
   (d) if $\text{deg}_\text{out}(c_1) = 2$ forward shift is not possible via child $c_1$,
4. randomly select and run one of the allowed shifts, skip if no shift is possible.

The parents-branching shifts follow in a similar way. In Figure 5 we present the possible scenarios.

5.7 Vertex deletion and insertion

Most of the above alterations are vertex-wise and require inner nodes (that is from set $V \setminus (S \cup D)$). The following modifications provide vertices needed for sampling routines.

Insertion:

1. sample a random edge $p_1 \rightarrow p_2$ from $E$,
2. with probability $P = \frac{1}{1+\exp(-b|p_1,p_2|-l_a)}$ divide the edge $p_1 \rightarrow p_2$ into a pair of edges $p_1 \rightarrow p_{\text{new}} \rightarrow p_2$, where $p_{\text{new}} := (p_1 + p_2)/2$, and $l_a \geq 0, b \geq 0$ are tuning parameters (the larger $l_a$ is, the longer $(p_1, p_2)$ must be in order to allow the edge division).
Note: if parameter \( l_d \approx 0 \) is too small, then one may encounter exponential growth of very short edges, which leads to numerical instabilities.

Deletion:

1. sample a node \( p \in V \setminus (S \cup D) \), such that \( \deg_{in}(p) = \deg_{out}(p) = 1 \),
2. denote \( r \) — the only parent of \( p \), and \( c \) — the only child of \( p \),
3. with probability \( P = \frac{1}{1 + \exp(-b|r,c|-l_d)} \) merge edges \( r \rightarrow p \) and \( p \rightarrow c \) into edge \( r \rightarrow c \) and delete \( p \); \( l_d \geq 0, b \geq 0 \)

Note: if the parameter \( l_d \gg 0 \) is too large, this operation can rapidly delete most of inner nodes with input and output degrees one, which are the main source of degrees of freedom in the system.

### 5.8 Triangular anomalies

In course of above alterations one can encounter untypical situations. While they are not a direct violation of the network constraints, they are suboptimal and hard to dissolve in a course of above modifications. We refer to these situations as anomalies. The idea is presented in Figure 6(b).

Below is the subroutine dissolving triangular pattern:

1. for every \( c \) — child of node \( p \) do
   - if there exists a two-step path \( p \rightarrow q \rightarrow c \) then redirect \( p \rightarrow c \) through \( p \rightarrow q \rightarrow c \):
     - \( \text{tmp} := t(p \rightarrow c) \), and delete edge \( p \rightarrow c \).
     - \( t(p \rightarrow q) := t(p \rightarrow q) + \text{tmp} \),
     - \( t(q \rightarrow c) := t(q \rightarrow c) + \text{tmp} \).

### 5.9 Quadrilateral anomalies

These anomalies include edges in quadrilateral shape. Again, these are not a direct violation of the network constraints, but can be hard do get rid off. Unlike triangular they come in two versions: common grandchild (or ‘2-2’), and 3-grandchild is a child (‘3-1’).

As it can be seen in Figure 6 they do not add new vertices to the graph. The search for these shapes can be expensive in general case, but if the limits for input and output degrees are held, then it is possible in time \( O(1) \).

Anomaly 3-1 removal:

1. given a vertex \( v \) and its child \( c \),
2. if there exists three-step path \( v \rightarrow p_1 \rightarrow p_2 \rightarrow c \) then redirect \( v \rightarrow c \) through \( v \rightarrow p_1 \rightarrow p_2 \rightarrow c \):
   - \( \text{tmp} := t(v \rightarrow c) \),
   - for edges: \( v \rightarrow p_1, p_1 \rightarrow p_2 \) and \( p_2 \rightarrow c \) increase their \( t() \) value by \( \text{tmp} \),
   - delete edge \( v \rightarrow c \).

Anomaly 2-2 removal:

1. given a vertex \( v \) and its two children \( c_1 \) and \( c_2 \)
2. if both children \( c_1 \) and \( c_2 \) have a common child \( g \) (grandchild of \( v \)) then choose better solution:
   - Redirection \( v \rightarrow c_1 \rightarrow c_2 \rightarrow g \):
     - \( t(v \rightarrow c_1) := t(v \rightarrow c_1) + t(v \rightarrow c_2) \),
     - add new edge \( t(c_1 \rightarrow c_2) := t(v \rightarrow c_2) + t(c_1 \rightarrow g) \),
     - \( t(c_2 \rightarrow g) := t(c_2 \rightarrow g) + t(c_1 \rightarrow g) \),
     - delete edges \( v \rightarrow c_2 \) and \( c_1 \rightarrow g \),
   - or redirection \( v \rightarrow c_2 \rightarrow c_1 \rightarrow g \):
     - \( t(v \rightarrow c_2) := t(v \rightarrow c_2) + t(v \rightarrow c_1) \),
     - add new edge \( t(c_2 \rightarrow c_1) := t(v \rightarrow c_1) + t(c_2 \rightarrow g) \),
     - \( t(c_1 \rightarrow g) := t(c_1 \rightarrow g) + t(c_2 \rightarrow g) \),
Figure 6: Anomaly dissolution: (a) and (b) triangular shape, (c) and (d) quadrilateral 2-2 shapes, (e) and (f) 3-1 shapes.

- delete edges $v \rightarrow c_1$ and $c_2 \rightarrow g$.

This step does not affect the correctness of the algorithm, but can speed up the convergence. The only thing which can be ‘broken’ is possibility of returning to the state before this optimization. There are no direct reversing procedures, but the optimization can be ‘reverted’ a sequence of vertex splittings and jumps.

6 Results

We have tested our computer implementation of the algorithm. We used Java JDK 1.7 and OpenJDK 1.7 both running under Linux OS x64 on i5 CPU 3.1 GHz and 16 GB of RAM (though, we utilized only a fraction of this number). The implementation includes real-time visualization of the found solution, which affects timing results (there are not many performance statistics in the literature to compare).

There are few tuning parameters in the algorithm such as: initial inverse temperature $\beta$ (see Eq. 7), annealing cooling rate, coefficient of the penalty in the cost function (see Eq. 14), thresholds in deletion and insertion of the vertices (see Sec. 5.7) etc. The first test case consists of one source and two destinations located in nodes of equilateral triangle (see Fig. 7). Having set the cost exponent $\alpha := 0$ (see Eq. 1) we end up with metric Steiner tree problem. The exact solution is known in this case, the best network consists of heights of the triangle from intersection point to vertices of the triangle. The algorithm returned accurate result, though local optimization of the final shape took several minutes.

Figure 7: Equilateral triangle case (one source and two sinks, $\alpha = 0$ — geometrical Steiner tree problem). (a) Starting setup (minimum spanning tree solution), $cost = 400u$ ($u$ denotes units). (b) Found solution $cost \simeq 346.77u$. Known analytic solution: $Cost_{opt} = 200\sqrt{3} \simeq 346.41u$.

We opted for linear and geometrical update of inverse temperature $\beta$, that is: $\beta := \beta \cdot q$, for $q > 1$. Unless stated otherwise, the exponent $\alpha$ in the cost function Eq. 1 is $\alpha = 0.5$. The threshold lengths $l_a$ and $l_d$ are dependent on the scale of the graph. A rule of thumb is that there should be enough space for at least one inner node between every
pair of source and destination. By default most of the transports \( t(s, d) \) between a source \( s \) and destination \( d \) in the initial graph are equal to 1. We refrain from floating point values of transport \( t() \) due to numerical instabilities.

The remaining examples are more tricky, as the exact solution is rarely known, and there is a shortage of similar algorithms to compare. The work of Matuszak et al. [5] discusses bayesian influence-based approach but with limited information about timing and quality. The work of Xia [11] discusses a number of cases, but limited to one source node, and misses performance statistics as well.

In Figures 8 and 9 we present other examples organized in varied topologies: a circle with unique source in its center, destination scattered inside a ball and sources and destinations located on parallel segments. Some of these cases give tree-like shapes obtained from Liendenmayer systems and other fractal techniques [12]. Our results resemble the figures obtained by Xia [11], in particular when we selected large number of vertices. The number of sources and destination varies from 10 to a 300.

In panels 8(a) and 8(b) the destinations are organized into a one-dimensional sphere with radius \( R = 100 \). The source in center point has three leaving routes with an angle between them approximately equal to \( 2\pi/3 \), and being further divided as they reach the perimeter. A topologically similar setup, but with the center located on the perimeter is presented in Fig 8(c) and 8(d). The position of the unique sources along with the sources makes the path resemble the topology of the setup. Panels 8(e) and 8(f) are organized into a ball with \( R = 100 \) and the source located on the perimeter, so the embedding space is now clearly 2d. A bush-like network clearly emerges in this case. In panels 9(a) and 9(b) the sources and sinks are located on parallel segments, distanced by 200 units. The algorithm returned a tree like structure with root parts built on top of sources and leaves in destination points.

For larger sets 8(b), 8(d), 8(f) and 9(b) we needed to lock transport swaps and random motions, and focus the computing power on reduction of number of inner nodes by massive merging. Once most of the paths merged together, we enabled these operations again. While this makes the algorithm a bit interactive or supervised, it allowed to reach impressive solution in dozen thousand of iterations. The shapes clearly resemble stochastic-modeled plants [12]. The massive cost reduction compared to smaller samples is due to the exponent \( \alpha \) in formula [1] which gives more savings (compared to linear solution) for large flows.

The total running time (including real-time visualization) varied from a couple of minutes (small samples up) up to several hours (300 nodes case in Fig 8(f)). The number of needed timesteps varied from a couple of hundreds (triangle) to an order of \( 10^5 \) for case 8(f). It might be confusing, when the algorithm should be stopped. Unlike many precise algorithms which give an all or nothing result, simulated annealing and evolutionary-based ones offer a (possibly very poor) solution at every timestep, but the quality improves over time. It is left to the user to decide if the solution is satisfactory.

### 7 Transport redirections

#### 7.1 Further modifications

Consider a variation of the transporting task, where the goods are indistinguishable and can be interchanged between the destinations. For instance: instead of transferring \( x \) amount from source \( p_1 \), the destination \( d \) can accept the goods from source \( p_2 \) and the surplus in \( p_1 \) is redirected somehow else. In other words — we no longer have the transport setup. This variant can result in better global solution.

Without branching this would be a straightforward linear optimization task, but if we account for this option, we end up with an interesting extension of the original problem. It is now reasonable to introduce following alteration, which can affect the amounts transported between \( S \) and \( D \).

In other words, we replace transport setup \( t^0 : S \times D \to \mathbb{N} \), with transport intensities \( t^1 : S \cup D \to \mathbb{N} \). The balance equation follows

\[
\sum_{s \in S} t^1(s) = \sum_{d \in D} t^1(d) \tag{13}
\]

The introduced modification can damage the balance equation (it is not a balance constraint), so we introduce a balance violation penalization to the cost function:

\[
Cost_p(G, t) := \sum_{s \in S} (t(s) - t^1(s))^2 + \sum_{d \in D} (t(d) - t^1(d))^2. \tag{14}
\]

If the real production or consumption rate of any source or destination differs from its optimal value, then the penalty cost is nonzero. The modifications can be seen in Fig. 10.

The modified cost function follows:

\[
Cost^1(G, t) := c_1 \cdot Cost(G, t) + c_2 \cdot Cost_p(G, t). \tag{15}
\]

where \( Cost \) was defined in Eq. 2 scalars \( c_1, c_2 \geq 0 \) are tuning parameters.

For captions we denote:
Figure 8: Non-trivial examples of transporting routes problem with single source. (a) and (b) — single source, multiple sinks cases in spherical topology; (c) to (d) — single source and multiple edges distributed on perimeter of the square; (e) and (f) source and destinations located randomly inside 2d-ball.

- $C_1 = \text{Cost}_1(\mathcal{G}, t)$,
- $C = c_1 \cdot \text{Cost}(\mathcal{G}, t)$,
- $C_p = c_2 \cdot \text{Cost}_p(\mathcal{G}, t)$,
- $\text{error} = C_p/C^1$.

7.2 Perturbations

We introduce two more graph modifications, which interfere in transport between pair source — destination

1. select a source node $s$, and two destination nodes $d_1$ and $d_2$, such that,
   - transport from $s_1$ to $d_1$ is nonzero,
   - there are unique routes from $s_1$ to $d_1$ and from $s_1$ to $d_2$ (this requirement can be lifted, if the graph representation stores the information, which allow us to determine the right path),

2. attempt to redirect $f$ units of goods, from route $s_1 \rightarrow ... \rightarrow d_1$ to route $s_1 \rightarrow ... \rightarrow d_2$, that is:
   - assign $t_{uv} := t_{uv} - f$ for every edge $u \rightarrow v$ on path from source $s$ to destination $d_1$
   - assign $t_{uv} := t_{uv} + f$ for every edge $u \rightarrow v$ on path from $s$ to $d_2$
(a) Number of $N = 11$ sources and destinations located on parallel segments. Cost: $2200 \rightarrow 1393.74$. Approx. 400 iterations.

(b) Same topology with $N = 101$ pairs of source and destination. Cost: $2 \cdot 10^{4} \rightarrow 6.3 \cdot 10^{3}$. Approx. $2.5 \cdot 10^{4}$ iterations. The bottom part of the network is still in clearly suboptimal shape.

Figure 9: Non-trivial examples of transporting routes problem. (a) and (b) sources and destinations located on parallel segments.

Figure 10: Redirection of transport between sources and destinations: (a) and (b) redirection on existing paths, (c) and (d) creation of new path.

- if this modification completely drains edges in the graph, then delete them (they serve no further purpose).

If the destination point $d_2$ is not reachable from $s$, then either the modification is not possible or we need to construct the missing path.

1. given $s$ and $d_1$, as above and $d_2$ which is not reachable via a directed path from $s$,
2. find inner node $p$, which can be reached from $s$ via a directed path and the distance between it and $d_2$ is minimal,
3. if $|p, d_2| \leq |s, d_2|$ add a new edge $p \rightarrow d_2$ to the graph, and redirect $f$ units of good from $s \rightarrow ... \rightarrow d_1$ to newly created $s \rightarrow ... \rightarrow p \rightarrow d_2$ (update $t$ accordingly),
4. otherwise add new edge $s \rightarrow d_2$ to the graph and redirect $s \rightarrow ... \rightarrow d_1$ to $s \rightarrow d_2$,
5. delete edges with transport equal to 0.

7.3 Results

In Figure 11 we present an interesting case, for which the basic variation, as described in Sec 3, turned too weak. This case was interactive. Namely we manually and on-line adjusted values of parameters $c_1$ and $c_2$ in order to focus the algorithms on the most suited updates. Roughly speaking the idea follows:

- begin with balanced $c_1$ and $c_2$ (the costs coefficients should be of the same order),
- reduce $c_1$ to 0, let the algorithm redirect the flows to the nearest neighbours (cheapest solution),
- slowly recover $c_1$ to its original value so that the algorithm develop routes to unsaturated destination points,
- let the algorithm tune the final shape of the network, remove surplus of nodes etc.

The overall process seems 'algorithmizable’ and we consider application of machine learning to adjust the parameters during the optimization phase.
8 Conclusions

In this work we described network searching algorithm along with sketch of its theoretical background. While the proposed algorithm is only a heuristic, it yields satisfactory results for many non-trivial cases with multiple nodes. There are some limitations which seem interesting directions of further development, for instance distinguishable goods (they can be exchanged only among the sources of the same type) or optimization in $\mathbb{R}^3$. An interesting case is varied cost of they area, for instance building a road through the wetlands, across the hill or building a bridge over the river are more expensive than road on plain ground. Finally, the ground is not flat, so we plan to adjust the algorithm to account for hills, valleys, and slope of the road.

Acknowledgments

This work is dedicated to memory of Tomasz Schreiber (1975 – 2010), who first drew my attention to these research areas.


References


