Perturbation Analysis of Practical Algorithms for the Maximum Scatter Travelling Salesman Problem

Emil Biju [*] Sundar Raman P.[†]

Abstract

The Maximum Scatter Traveling Salesman Problem (MSTSP) is a variant of the famous Traveling Salesman Problem (TSP) and finds its use in several real-world applications including manufacturing, imaging and laser melting processes. The objective of this problem is to maximize the cost of the least cost edge in a tour of an input graph. Akin to the TSP and several of its variants, the MSTSP is an NP-hard problem. While several approximation algorithms have been proposed for this problem, many of them suffer from bad worst-case complexities and present challenges in scalability and practical use. Besides, these algorithms have often been designed and evaluated with a sole focus on theoretical approximation quality, while practical applications require detailed experimental evaluations to study the stability, quality and runtime over a large and diverse set of inputs. In this work, we describe six algorithms for MSTSP inspired by prior work in the area, along with improved formulations that enhance their utility in real-world scenarios. Further, we perform experimental studies motivated by smoothed analysis to comprehensively evaluate these algorithms on various performance metrics. We demonstrate that despite having bad worst-case complexities, certain algorithms perform exceedingly well in practical use cases. Our experiments reveal trade-offs among the runtime, quality and stability of different algorithms that must be considered when making a design choice depending on the objectives and constraints associated with the use of the algorithm.

1 Introduction

The Traveling Salesman Problem (TSP) [11] is a well-known and widely studied discrete optimization problem where the objective is to find a tour through all nodes in an input graph such that the total tour cost is minimized. The Maximum Scatter Traveling Salesman Problem (MSTSP) is a modification of this problem where the objective is to find a tour such that cost of the least cost edge is maximized. Formally, the problem can be defined as follows.

Given a set of points $P = \{p_1, ..., p_n\}$, we consider a tour T over these points. The scatter of the tour, T is defined as the cost of the least cost edge in T. The goal of the MSTSP is to find a tour of points in P with the maximum scatter, i.e., to maximize the cost of the least cost edge in a tour of P.

The MSTSP was first introduced in 1997 [2], motivated by problems in riveting and medical imaging. It also has applications in laser melting, additive manufacturing [14], etc. where an optimal placement of items in a sequence needs to be identified so that they are geometrically well-separated to avoid interference. Similar to other variants of TSP, solving MSTSP exactly is NP-hard [2]. Therefore, efficient algorithms identified so far are approximation schemes that compute a tour whose scatter is at least a factor $\epsilon \in (0, 1)$ of the optimum, called as an ϵ-approximation. Asymptotically optimal and linear-time, deterministic weave algorithms [7] work only for graphs with vertices on a regular 2D grid while the complexity of the general MSTSP in a plane remains open. Using Dirac's theorem [4], [2] proposes a 0.5-approximation for MSTSP in the Euclidean plane and raised the question of whether a better approximation ratio can be obtained. [9] proposes a non-trivial (1-ϵ) polynomial-time approximation scheme that constructs multi-graphs to arrive at an optimal tour. However, it has a high dependence on the precision parameter ϵ and becomes extremely slow even for $n \sim 100$, which renders it impractical for several use cases. Recently proposed meta-heuristics like genetic algorithms with different crossover operators [1] are time-consuming and inappropriate choices of control parameters, fitness functions will make it difficult for the algorithm to converge.

Hence, there is a clear need for algorithms that are easy to implement, practical in real world applications and scalable. Besides, we require a thorough experimental evaluation of algorithms based on diverse metrics that influence overall performance to understand scenarios where each of them may be useful. Many algorithms work well on real data, despite having poor complexity under the standard worst-case measure, like the simple 2-opt algorithm [3] as we will demonstrate.

[*]Indian Institute of Technology Madras.
[†]Indian Institute of Technology Madras.

Many proofs of worst-case complexity make use of contrived constructions of 'bad' inputs [6] whose existence and applicability in any real-life setting is uncertain. Perturbation analysis [13] helps explain the behavior of algorithms in practice by subjecting the algorithm inputs to slight random perturbations and evaluating its average performance over the perturbed inputs.

In Section 2, we describe six approximation algorithms for the MSTSP which are derived from previously proposed approaches either for the TSP or MSTSP. We introduce suitable modifications to some of these algorithms to enhance their practical usability in real-world scenarios. In Section 4, we describe the dataset that we use to evaluate our algorithms and perform experiments. Further, in Section 5, we perform a comprehensive perturbation analysis of the six algorithms and evaluate trade-offs in their practical usage. To the best of our knowledge, this is the first body of work that has performed such a detailed comparative study and experimental evaluation of MSTSP algorithms. The main contributions of this paper are as follows.

- We present the Naive Greedy algorithm as a fast and easy-to-implement baseline for MSTSP.
- We present the Naive Weave and Hoffmann Weave algorithms which introduce an improved formulation of the work in [2] and [7] to extend their usability to non-regular grids.
- We extend the work in [8] to introduce Pure 2-opt and Randomised 2-opt as very close approximation algorithms for the MSTSP.
- We use a real-world dataset augmented using five graph perturbations and evaluated with three edge cost metrics to perform a comprehensive perturbation analysis of the algorithms and compare results on three critical performance measures, namely, the quality, runtime and stability of the algorithms.

2 Algorithmic Description

2.1 Naive Greedy Algorithm.

This algorithm presents a greedy approach for approximating the maximum scatter TSP. In this algorithm, we start at a random node in the graph and then travel to the farthest node from it. We iteratively repeat this on visiting each new node by travelling to the farthest unvisited node. Once all the nodes are visited, the last edge connects back to the first visited node to complete the tour. The cost of the least cost edge in this tour is considered as the solution to the Maximum scatter TSP for the given set of points.

2.2 Naive Weave Algorithm.

Here, we devise an algorithm to extend the work by Arkin et al. [2] to points in a 2D plane. First, we present a method to define a grid over a given set of points. Then, we apply Arkin's algorithm to points in each row of the grid iteratively. We also make edges between rows to ensure completion of the tour.

Existing work [7] for approximating the solution to the maximum scatter of points in a 2D plane is limited in application to regular grids. This limitation restricts the usability of the algorithm in several real-world scenarios. Hence, we present a method to define a grid over any given set of points, such that applying Arkin's algorithm to each row separately gives a fair approximation to the maximum scatter TSP solution. We note that it is not necessary for a point to be present in every column of every row of the grid to apply Arkin's 1D algorithm over the row. Specifically, we could leave vacant locations in a row and still travel through the remaining points following the sequence ordering proposed by Arkin.

Given a set of n points, we define d as:

$$d = \alpha \cdot \lfloor \sqrt{n} \rfloor \tag{2.1}$$

where α is a hyperparameter that is finalized through experiments.

We sort the given set of points by their row-coordinates and partition the sorted array into subarrays of d points each. The last partition alone may have less than d points. The points in each partition are assigned to the same row in the grid and every point receives a row index as a result. Next, we sort the points by their column-coordinates and iterate over them starting from the point with the least column coordinate. Each new point is added to the first column until a point having the same row index as a point that is already assigned to the column is encountered. Once this occurs, we create a new column and continue adding further points to it and repeat the process until all points are assigned a column index. Thus, a grid over the points in the 2D plane is defined.

If there are m points in a row of the grid, we visit the points following the sequence ordering prescribed by Arkin's algorithm, i.e,

- If m is even, say, $m = 2k + 1$: $1, k + 2, 2, k + 3, ..., m, k + 1$
- If m is odd, say, $m = 2k$: $1, k + 2, 3, k + 4, ..., k - 1, m, k, m - 1, ..., k + 1$

The points in each row of our grid are traversed based on the ordering above. The rows are visited sequentially and we also connect the last visited node from each row to the first node to be visited from the next row. Finally, to complete the tour, the last visited point from the last row is joined to the first visited point of the tour. The minimum edge cost in this traversal is returned as the algorithm's approximation for the maximum scatter TSP.

Algorithm 1 Procedure for the Naive Weave Algorithm

```
1: Given: Graph $G(v_1, ..., v_n)$
2: max_pts_per_row ← $d = floor(\sqrt{n}) \cdot fact$
3: Initialise: $i \leftarrow 1$
4: Initialise: scatter = $+\infty$
5: Define a grid on these points (num_rows= floor($n/d$) or 1+floor($n/d$))
6: while i≤num_rows do
7:     Traverse the row based on Arkin's ordering of points $[v_{i,1}, ..., v_{i,k+1}]$
8:     if i = num_rows then
9:         Jump from $v_{i,k+1}$ to $v_{1,1}$
10:    else
11:        Jump from $v_{i,k+1}$ to $v_{i+1,1}$
12:    scatter ← min(scatter, highest edge cost in this traversal)
13: return scatter
```

2.3 Hoffmann Weave Algorithm. In this algorithm, we extend the algorithm in [7] which works on regular grids to operate on any given set of points in a 2D plane. We first define a grid over the points using the method presented in Section 2.2. In the naive weave algorithm, rows are traversed one at a time after which the tour moves to the subsequent row. If the edges between rows have low cost, the algorithm will output poor approximations. Besides, it is not necessary that all points in a row are covered together. Instead, we can take advantage of the second dimension to increase the minimum edge cost in the tour.

We first explain the method when num_rows is even. We partition the rows into pairs with each pair being separated by a fixed number of rows. If num_rows $= 2t$, the rows are paired as $(1, t+1), (2, t+2), ..., (t, d)$. Then, the traversal is performed by interleaving through one pair at a time.

For the pair $\{i, t+i\}$, the ordering is given by:

$$HW\{i, t+i\} = (i, \sigma(2)), (t+i, \sigma(3)), (i, \sigma(4)), ... \\ ..., (i, \sigma(d)), (i, \sigma(1)) \tag{2.2}$$

where d is the number of points in each row and $\sigma(j)$ is the j^{th} point in Arkin's 1D ordering of d points.

To switch between pairs, an edge is made from the last visited point of the old pair to the first point to be visited from the new pair. However, switching between pairs requires a different strategy depending on whether d is odd or even.

When d is odd, the ordering of pairs is given by:

$$HW\{1, t+1\}, HW\{t+1, 1\}, HW\{2, t+2\}, HW\{t+2, 2\}, ...,$$

$$..., HW\{t, d\}, HW\{d, t\}$$

When d is even, the ordering of pairs is given by:

$$HW\{1, t+1\}, HW\{2, t+2\}, HW\{3, t+3\}, ..., HW\{t, d\},$$

$$HW\{t+1, 1\}, HW\{t+2, 2\}, HW\{t+3, 3\}..., HW\{d, t\}$$

Next, we explain the method for the case when the number of rows is odd. We partition the rows into pairs and a triplet having rows at a distance of t from each other. First, the traversal through the triplet is completed. While the column index for each visited point is decided by whether d is odd or even as presented earlier, the row index alternates over 3 possible values. If num_rows $= 2t + 1$, the rows may be visited in the order $(1, t+1, m, 1, t+1, m, ...), (t+1, m, 1, t+1, m, 1, ...), (m, 1, t+1, m, 1, t+1, ...)$ or $(1, t+1, m, 1, t+1, m, ...), (m, 1, t+1, m, 1, t+1, ...), (t+1, m, 1, t+1, m, 1, ...)$ depending on the value of num_rows to ensure that no point is visited more than once before the tour is complete. After this, an even number of rows remain and the strategy in Equation 2.2 is used for the remaining traversals. The last visited point is joined back to the first visited point to complete the tour. The algorithmic procedure is detailed in Algorithm 2.

2.4 Dirac Algorithm. The Dirac algorithm is based on Dirac's theorem [5] which states that if every node of a graph having n nodes ($n > 3$) has degree $\geq \lceil \frac{n}{2} \rceil$, then the graph contains a Hamiltonian cycle. This algorithm from the work by [2] is a 0.5-approximation algorithm with $\mathcal{O}(n^2)$ runtime which is accomplished as follows. The problem of finding a Hamiltonian cycle in a pruned graph where each node has degree $\geq \lceil \frac{n}{2} \rceil$ is as difficult as finding a Hamiltonian cycle in the unpruned graph (same order of complexity). So, we start with a random Hamiltonian cycle instead which is trivially found in the initial fully-connected, unpruned graph. The median edge weight of edges from each node in the graph is computed in $O(n^2)$ time. Then, the minimum of all these medians acts as a threshold. Let $E^{'}$ denote the set of all edges which have edge weight greater than or equal to this threshold. Any edge of a Hamiltonian cycle with weight lower than this threshold is pruned

Algorithm 2 Procedure for the Hoffmann Weave Algorithm

```
 1: Given: Graph $\mathbf{G}(v_1, ..., v_n)$
 2: max_pts_per_row $\leftarrow d = floor(\sqrt{n}) \cdot fact$
 3: Initialise: scatter $\leftarrow +\infty$
 4: Define: $\sigma(j) = j^{th}$ point in Arkin's 1D ordering of $d$ points
 5: Define: $HW\{i, t+i\} = (i, \sigma(2)), (t+i, \sigma(3)), (i, \sigma(4)), ..., (i, \sigma(d)), (i, \sigma(1))$
 6: Define a grid on these points (num_rows= floor($n/d$) or 1+floor($n/d$))
 7: if num_rows is odd then
 8:     Complete traversal of rows $1, t+1, m$
 9:     scatter $\leftarrow$ min(scatter, highest edge cost in this traversal)
10:     Ignore these rows to get an even number of remaining rows
11:     num_rows $\leftarrow$ num_rows$-3$
12: if $d$ is odd then
13:     for $i = 1$ to $t$ do
14:         Complete traversals $HW(i, t+i)$ and $HW(t+i, i)$
15:         scatter $\leftarrow$ min(scatter, highest edge cost in this traversal)
16: else
17:     for $i = 1$ to $t$ do
18:         Complete traversal $HW(i, t+i)$
19:         scatter $\leftarrow$ min(scatter, highest edge cost in this traversal)
20:     for $i = t$+1 to $d$ do
21:         Complete traversal $HW(i, i-t)$
22:         scatter $\leftarrow$ min(scatter, highest edge cost in this traversal)
23: return scatter
```

and replaced with edge(s) with weight greater than the threshold through a 2-opt step to result in another cycle. In more detail, if path $\mathbf{P}$=$pv_1v_2 \dots v_{n-2}q$ has an edge (p, q) with edge weight less than the threshold, then delete edges (p, q), (v_i, v_{i+1}) and add edges (p, v_{i+1}), (q, v_i) where $i \in P \cap Q$ where $P = \{i : (p, v_{i+1}) \in E^{'}\}$, $Q = \{i : (q, v_i) \in E^{'}\}$. $P \cap Q$ must be non-empty as $P \cup Q < n$ (as edge pq doesn't belong to $E^{'}$) and as $|P \cap Q| = |P| + |Q| - |P \cup Q|$ where $|P| + |Q| \geq n$. By representing the sets P, Q with bit vectors, all of the bookkeeping can be done in an overall time of $\mathcal{O}(n^2)$.

2.5 2-opt Algorithm. A naive 2-opt algorithm is implemented where a 2-opt step is performed when the current least cost edge (whose weight is the current scatter value) of a Hamiltonian tour can be replaced

Algorithm 3 Procedure for the Dirac Algorithm

```
 1: Given: Graph $\mathbf{G}(v_1, \dots, v_n)$
 2: Initialise: median[n], valid[n] = $\{0, \dots, 0\}$
 3: Initialise: Random Hamiltonian path $\mathbf{P}(p_1, \dots, p_n)$
 4: def $scatter$(Graph $\mathbf{G}$, Path $\mathbf{P}$):
 5:         return scatter of $\mathbf{P}$, scatter edge's vertex id
 6: for $i = 1$ to n do
 7:     median[i] $\leftarrow$ median of $\{d(v_i, x) \mid x \in \mathbf{G}\}$
 8: min_med $\leftarrow$ minimum of $\{$median[i] $\mid i \in [n]\}$
 9: for $i = 1$ to n do
10:     valid[i] $\leftarrow \{x \mid d(v_i, x) \geq$ min_med for $x \in \mathbf{G}\}$
11: for $i = 1$ to n do
12:     $p, q \leftarrow \mathbf{P}[i], \mathbf{P}[(i+1)\%n]$
13:     if $d(p, q) <$ min_med then
14:         val_p $\leftarrow$ { index of $x$ in $\mathbf{P} \mid x \in$ valid[p] }
15:         val_q $\leftarrow$ { index of $x$ in $\mathbf{P} \mid x \in$ valid[q] }
16:         val_q $\leftarrow$ (val_q - 1) % n
17:         $r \leftarrow set\_intersection$(val_p, val_q)[0]
18:         $\mathbf{P} \leftarrow 2opt\_step(\mathbf{P}, i, r)$
19: scatter, scatter_index $\leftarrow scatter(\mathbf{G}, \mathbf{P})$
20: return scatter, $\mathbf{P}$
```

by another edge that connects one of the vertices at the ends of the current scatter edge to another vertex to get an improved scatter value. The initial tour may be randomly initialized, which is referred to as Randomized 2-opt (or) can also be initialized to the Hamiltonian cycle output from the Dirac algorithm as described above, which is referred to as Pure 2-opt. There is a noticeable difference between both variants in terms of average run time as can be inferred from Table 3. A 2-opt step among quadruplets (1,2,3,4) in a cyclic order (where vertices 1, 2 and 3, 4 are adjacent in the cycle) essentially disconnects the edge between 1 and 2, 3 and 4 and creates new edges between 1 and 3, 2 and 4.

3 Smoothed analysis of 2-opt for MSTSP

Let $Y = X + Z$ where X denotes the set of n 2D data points (from input files) reduced (scaled) to a unit-square $[0, 1]^2$ and Z denotes a random gaussian perturbation on X, $\mathcal{N}(0, \sigma^2)$ where $\sigma \leq \frac{1}{2\sqrt{n \log n}}$. The following analysis obtains an upper bound for the expected number of 2-opt steps for convergence. It closely follows the work in [10] on smoothed analysis of 2-opt heuristic for TSP.

Let the total number of iterations taken by the 2-opt algorithm be denoted by n_{iter}, initial tour's scatter value be L_{init} and incremental gain of the scatter value obtained over iterations (which if zero means that the algorithm has halted) be δ. To upper bound n_{iter}, we need to upper bound L_{init} (=$L_{init_{max}}$) and lower bound

Algorithm 4 Procedure for the 2-opt Algorithm

```
1: Given: Graph $\mathbf{G}(v_1, ..., v_n)$, maximum iterations $MAXITER(> 0)$, Hamiltonian path $\mathbf{P}(p_1, ..., p_n)$
2: Initialise: iter $= 0$
3: while iter$< MAXITER$ do
4:     found $\leftarrow 0$
5:     scatter, scatter_id $\leftarrow$ $scatter(\mathbf{G}, \mathbf{P})$
6:     for $i = 1$ to $n$ do
7:         $p, q \leftarrow \mathbf{P}[i], \mathbf{P}[(i+1)\%n]$
8:         $r \leftarrow \mathbf{P}$[scatter_id]
9:         $s \leftarrow \mathbf{P}$[(scatter_id $+ 1)\%n$]
10:        if $d(p, r)$ and $d(q, s) >$ scatter then
11:            $\mathbf{P} \leftarrow 2opt\_step(\mathbf{P}, i,$ scatter_index)
12:            found $\leftarrow 1$
13:            break;
14:    if found==0 then break;
15:    iter $\leftarrow$ iter+1
16: return scatter
```

δ (=δ_{min}) as $n_{iter} \leq \frac{L_{init_{max}}}{\delta_{min}}$. From Lemma 2.3 of Manthey's work [10], we have:

$$P[L_{init} \geq 18n] \leq P[Y \not\subset [-1, 2]^2] \leq \frac{1}{n!}$$

Let $y_1 y_2$ be the current least cost edge in a cycle P, S be the quadruplets (y_1, y_2, y_3, y_4) in a cyclic order (where vertices y_1, y_2 and y_3, y_4 are adjacent in the cycle) and $\delta(S)$ be the incremental gain in the scatter value after performing a 2-opt step on the quadruplets (y_1, y_2, y_3, y_4). If $d(a, b)$ represents the distance between points a and b computed using one of the metrics in Section 4,

$$\begin{aligned}\delta(S) &= min(d(y_3, y_1) - d(y_2, y_1), d(y_4, y_2) - d(y_2, y_1)) \\ w.l.o.g: \quad \delta(S) &= d(y_3, y_1) - d(y_2, y_1) = \Delta_{(3,2)}(1)\end{aligned}$$

From Lemma 4.1, 4.2 in [10] which holds for MSTSP also, we obtain:

$$P[|\Delta_{(3,2)}(1) \leq \epsilon \,|\, d(y_2, y_3) = \delta] \leq \frac{\epsilon}{4\sigma\delta}$$
$$\implies P[\delta_{min} \leq \epsilon] \leq \mathcal{O}(\frac{n^4 \epsilon}{4\sigma^2})$$

Finally, following Theorem 4.3 in [10], the expected number of 2-opt steps on Y required for convergence, $E[n_{iter}] = \mathcal{O}(\frac{n^6 logn}{\sigma})$ which is polynomial in n as opposed to the worst-case exponential nature of the 2-opt algorithm for MSTSP, due to its inherent NP-hardness.

4 Data

To simulate real-world conditions, the data that is used for our experiments is obtained from the files provided for Symmetric TSP in the TSPLIB library [12]. From this library, we take 76 graphs having atmost 2000 nodes. We assume that any two nodes in a graph can be connected by an edge. For each unperturbed input, we create 100 perturbed inputs using each perturbation type mentioned in Table 1 to obtain 7,600 perturbed inputs for each perturbation type. We anonymously make the code and dataset used for our work available[1]. These would be made publicly accessible if our paper is accepted to support further research in this area.

The cost of an edge can be computed using one of three distance metrics. If $a_1 = (x_1, y_1)$ and $a_2 = (x_2, y_2)$ are the coordinate locations of two nodes in a 2D plane, the distance metrics are formulated as follows:

- Euclidean: $d(a_1, a_2) = (x_2 - x_1)^2 + (y_2 - y_1)^2$
- Manhattan: $d(a_1, a_2) = |x_2 - x_1| + |y_2 - y_1|$
- Max 2D: $d(a_1, a_2) = max(|x_2 - x_1|, |y_2 - y_1|)$

We devise a simple method to approximate an upper bound for the maximum scatter TSP solution for each input graph. Consider the second highest cost edge incident on each node in the graph. As a tour visits every node and passes through atleast two of the edges incident on each node, the least cost edge among the second highest cost edge from each node gives an upper-bound for the actual maximum scatter value. We refer to this approximated bound as the *scatter bound*.

Pert. Type	Hyperparameter
Gaussian (μ, σ)	$\mu = 0, \sigma_1 = 0.5\times$ min_edge
	$\mu = 0, \sigma_2 = 0.05\times$ min_edge
	$\mu = 0, \sigma_3 = 0.5 \times$ max_hull $\times \frac{1}{\sqrt{nlogn}}$
Uniform (a,b)	$a = -\sigma_1, b = \sigma_1$
	$a = -\sigma_3, b = \sigma_3$

Table 1: Perturbations created for our dataset. min_edge refers to the cost of the least cost edge in the graph. max_hull is the maximum side length of the square convex hull that covers all the graph points.

5 Experiments

In this section, we perform perturbation analysis studies on the 6 algorithms and analyse results in terms of

[1]Our code and data are available at: https://drive.google.com/drive/folders/13TaSCjObHYNgJ2ysb6kHF1TQanvBjq8z?usp=sharing. Care has been taken to ensure anonymity of the uploader.

the quality of output, runtime, stability under perturbation and identification of worst case inputs. For standardising our results and ensuring reproducibility, all experiments were run on a Google Cloud Virtual Machine with the following specifications: N1 Intel Skylake platform, 2vCPU, 7.5GB RAM, Ubuntu Bionic OS (18.04 LTS). Unless otherwise mentioned, all plots and results are reported on inputs generated from the Gaussian ($\mu = 0, \sigma_1 = 0.5\times$ `min_edge`) perturbation type using the Euclidean distance metric. Nevertheless, we observed that results with other perturbation types and edge cost metrics followed very similar trends and hence, the results that we present are highly generalizable.

5.1 Closeness of algorithm predictions to the scatter bound. First, we study the closeness of the algorithms' predictions for the maximum scatter to the estimated scatter bound by averaging the predictions made over several perturbations of each input.

In Figure 1, we compare the average scatter bound over perturbations of each input x with the average maximum scatter predicted by each algorithm over perturbations of x. We observe that the naive greedy algorithm shows the poorest performance as the obtained maximum scatter values remain small though the ideal values increase. Dirac and weave algorithms increase with the ideal scatter value, but fall short of finding the most optimal tours. The Pure 2-opt and Randomized 2-opt algorithms render the best performance as the mean maximum scatter closely follows the mean scatter bound. This also helps us in establishing that the scatter bound is a fair estimate of the maximum scatter of a graph since the 2-opt algorithm outputs demonstrate that they are nearly realizable for a wide range of inputs.

In Table 2, we find the average of the ratio between the output of each algorithm for each input graph and the respective scatter bound over the complete set of input graph perturbations. We observe that while the Hoffmann Weave algorithm on regular grids has a theoretical approximation ratio of $\frac{\sqrt{10}}{5} \sim 0.63$, the mean approximation ratio is 0.78, which shows that the algorithm performs much better in practice and realizes one of the proposed benefits of our study. The Hoffmann Weave algorithm performs substantially better than Naive Weave which establishes the advantage of utilising the second dimension to increase edge costs. The Pure 2-opt algorithm has the best mean approximation ratio of 0.82 and hence, provides the highest output quality in this experimental evaluation.

Algorithm	**Mean Approximation Ratio**
Naive Greedy	0.09
Naive Weave	0.30
Hoffmann Weave	0.78
Dirac	0.51
Pure 2-opt	0.82
Randomized 2-opt	0.79

Table 2: Mean approximation ratio between algorithm prediction and scatter bound

5.2 Deviation of maximum scatter predictions under perturbation. Next, to study the stability of these algorithms under input perturbation, we analyse the extent of deviation of the maximum scatter value predicted by each algorithm for various perturbed inputs corresponding to each sample x. We quantify the deviation in two ways:

1. **Range of variation (RoV):** For a given sample x, the range of variation is defined as the difference between the largest and smallest maximum scatter predictions made by the algorithm among various perturbed inputs derived from x. If $P(x)$ denotes the set of all perturbed inputs derived from x and s_p is the maximum scatter predicted for an input p,

$$RoV(x) = \max_{p\in P(x)} s_p - \min_{p\in P(x)} s_p \tag{5.3}$$

2. **Scaled Deviation:** If s_x and s_p are respectively the maximum scatter predictions for the original sample x and its perturbation p, the scaled deviation (SD) is defined as

$$SD(x,p) = \frac{s_p - s_x}{s_x}$$

   Since it is possible for the RoV to be larger when the unperturbed scatter is larger, computing the scaled deviation has the effect of normalising by the unperturbed maximum scatter value.

We observe from Figure 2 that the maximum scatter predictions show very large variations from the prediction for the unperturbed sample, x when x has a smaller number of nodes. As the number of nodes increases, the stability of the algorithm appears to improve, particularly for the Naive Greedy, Dirac and 2-opt algorithms. However, Weave algorithms show large variations even when the number of nodes is large. Besides, the Dirac algorithm is highly stable over a wide range of input graph sizes. The 2-opt algorithms are also stable with the RoV values lying below 1500 in most cases. The naive greedy algorithm is stable only when the number of nodes is sufficiently large.

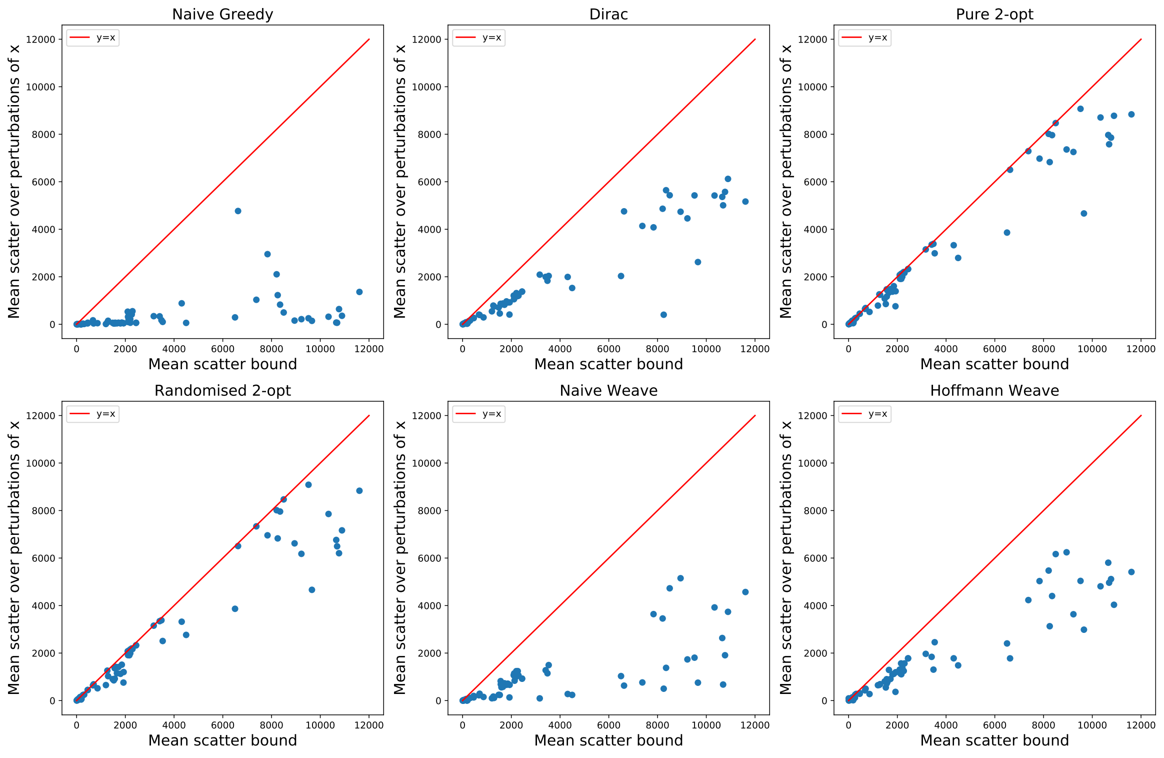

Figure 1: Comparison of the closeness of mean scatter bounds to mean scatter predictions of each algorithm. The mean is computed over Gaussian perturbations of each input.

From Figure 3, we observe that the scaled deviation takes on a larger range of values when the number of nodes is smaller. Each green dot refers to the scaled deviation value for a distinct perturbed input. All perturbed inputs with the same number of nodes are placed on the same vertical line. For most algorithms, the scaled deviation decreases with increasing number of nodes. This is because when there are too few nodes, variations in the extent of perturbation, starting conditions, etc. have larger impact on the obtained max scatter value. When the number of nodes increases, many of these factors get averaged out or are small after normalization. Similar to earlier observations that the RoV for Weave algorithms remains large even when the input graph contains higher number of nodes, we observe here that the scaled deviation also shows large variations. Hence, it is evident that Weave algorithms are less stable, even at higher graph cardinalities. The largest magnitudes for the scaled deviation are observed in the Naive Greedy algorithm, with values ranging upto 50. This is because the maximum scatter predictions remain small even when the number of nodes is large, as observed in Figure 1, leading to a weaker normalization effect from s_x.

In Figure 4, we average the scaled deviation values over all perturbations of each sample x and plot the averages for 3 different edge cost metrics against the number of nodes in x. First, we observe that the mean scaled deviation for the Euclidean and Max 2D edge cost metrics closely follow each other for most algorithms. Second, for Dirac and 2-opt algorithms, the mean scaled deviation when the Manhattan metric is used is higher than for the other metrics. For Weave algorithms, the variation with the edge cost metric is quite minimal. To define a grid over the points for running Weave algorithms, distances between points are largely measured along axis-aligned directions. This reduces the disparity among the distances measured using various edge cost metrics and consequently, the resultant grid locations are highly likely to be unaffected by the edge cost metric used.

5.3 Variation in the runtime of algorithms. In this section, we analyse the variation in the runtime of different algorithms. We plot the runtime for each input against the number of nodes in Figure 5. Firstly, we observe that the runtime increases with the number of nodes, which is expected since the algorithm has to explore a larger number of possible tours in the process of approximating the maximum scatter. For example,

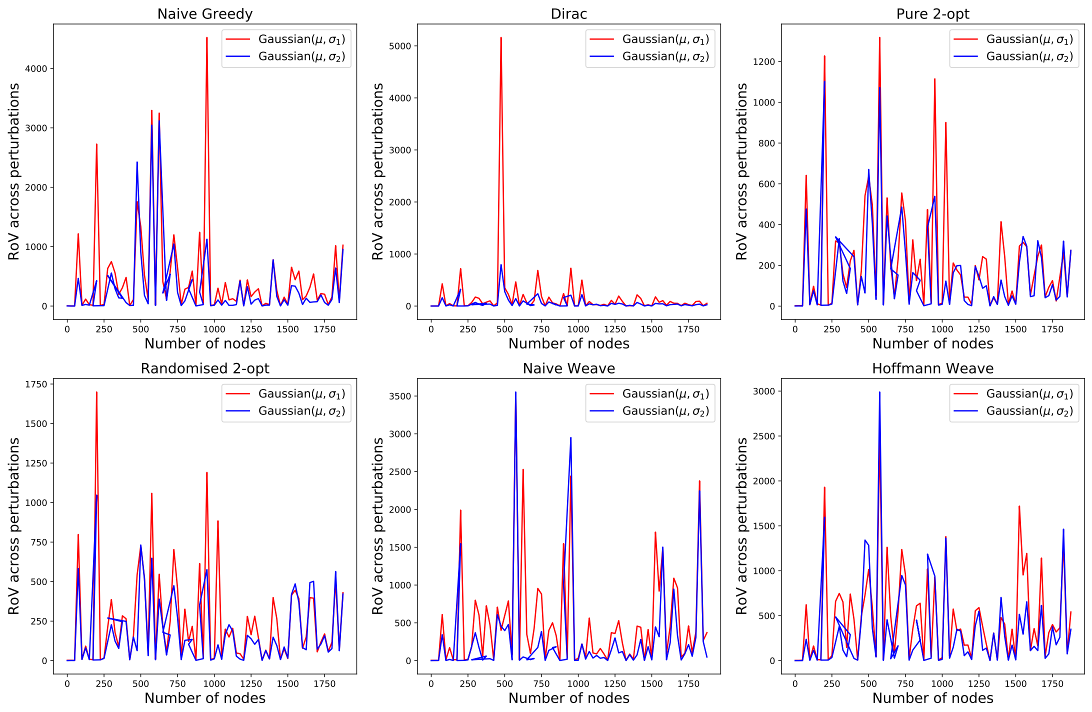

Figure 2: Comparison of the RoV of the maximum scatter values against the number of nodes in the input x. The inputs are arranged in the increasing order of the number of nodes along the X-axis. Note that the Y-axis scale for each plot is different.

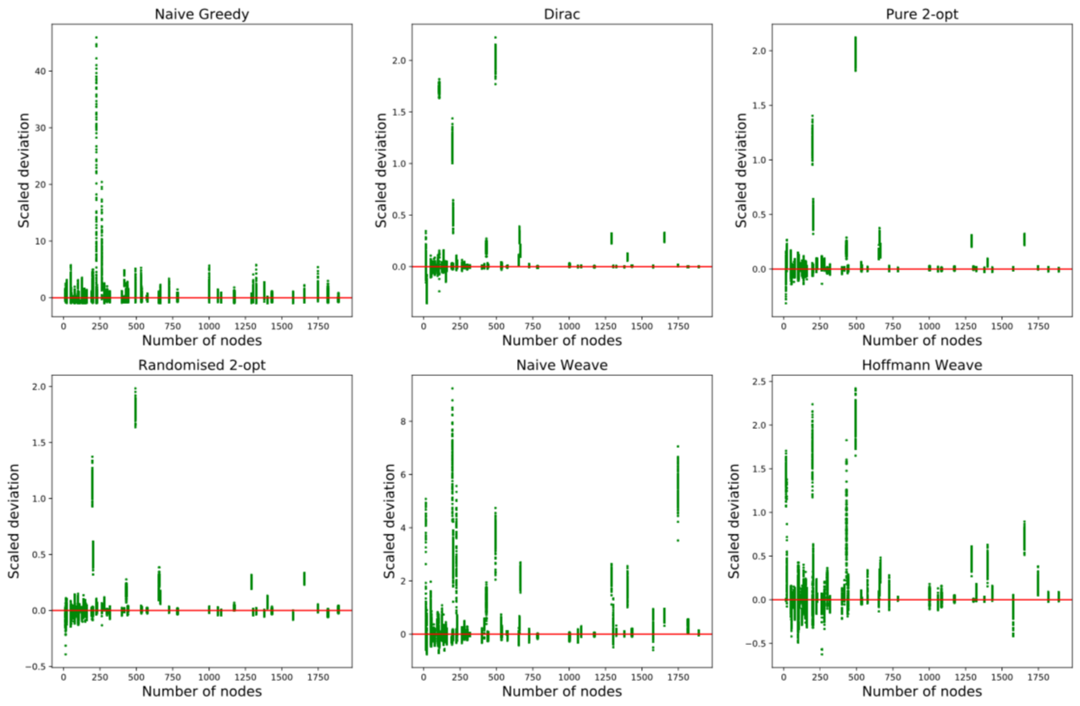

Figure 3: Comparison of the Scaled deviation of the maximum scatter predictions against the number of nodes in x.

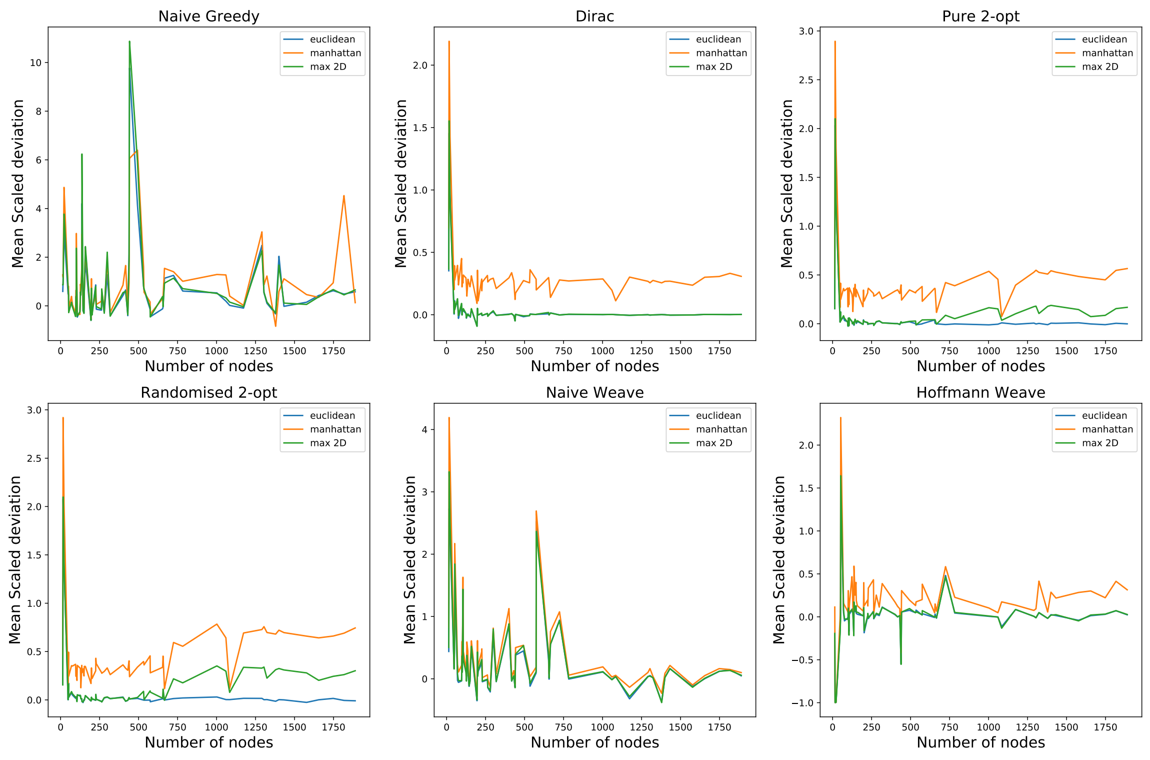

Figure 4: Mean scaled deviation over number of nodes for 3 possible edge cost metrics

in the naive greedy algorithm, after each node is visited, the algorithm checks all the remaining unvisited nodes to identify the farthest unvisited node and determine the next edge in the tour. Hence, the runtime of the algorithm for an input graph varies as $O(n^2)$, if n is the number of nodes and the plot is roughly quadratic for the Naive Greedy algorithm. Figure 5 compares the runtime on two possible values of the perturbation parameter, σ. It is observed that the variation with the perturbation parameter is minimal since the exact location of a perturbed instance doesn't deeply influence the average runtime, unless many bad instances are clustered. The lengths of the vertical lines show the difference between the maximum and minimum runtime among input graphs with the same number of nodes. We notice wider variations in runtime as the number of nodes increases. Naive greedy algorithms show comparatively less variation because the number of iterations required and the steps performed per iteration are fixed. 2-opt algorithms show the largest variations and this is because they are heavily dependent on the initial tour from which the 2-opt steps start. The choice of nodes among which the 2-opt swap occurs also determine how fast the optimal solution is reached. These can vary heavily depending on the initialization and perturbations.

Algorithm	Slowest (per input)	Slowest (on average)	Average runtime
Dirac	6002	64	1.241941
Rand. 2opt	1367	10	1.083100
Hf. Weave	209	2	0.004496
Pure 2opt	17	0	0.493515
Nv. Weave	5	0	0.003096
Nv. Greedy	0	0	0.002770

Table 3: (i) Number of perturbed inputs for which each algorithm has featured as the slowest. (ii) Number of inputs for which the algorithm has the largest mean runtime over all its perturbations. (iii) Average runtime of each algorithm over all perturbations of all inputs.

We also observe that for a given number of nodes in the range considered, the naive greedy and weave algorithms take much less time (by a factor of more than 100) than Dirac and 2-opt algorithms. Hence, we note that though the results of 2-opt are more accurate, it is much slower. Table 3 shows the number of perturbed inputs for which each algorithm has featured as the slowest among the 6 algorithms. It is clear that the Dirac algorithm is the slowest in most cases. It also has the largest average runtime over all inputs. The naive

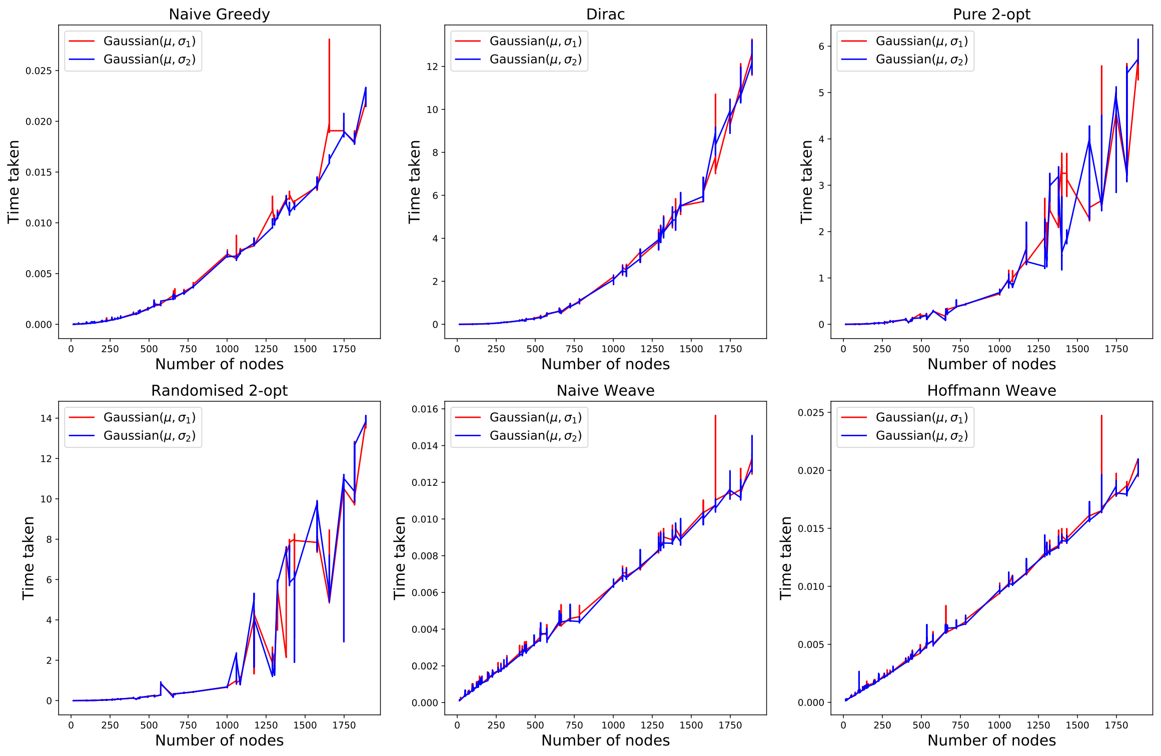

Figure 5: Comparison of the runtime of each algorithm against the number of nodes in the input. The two coloured curves on each plot show the runtime for inputs generated from two different Gaussian perturbations. Each vertical line shows the difference between the maximum and minimum runtime for perturbed inputs with the corresponding number of nodes. Note that the Y-axis scale for each plot is different.

greedy algorithm never features as the slowest, while the naive weave algorithm does so in 5 instances corresponding to input graphs with very few nodes (16 and 22 nodes). When the mean runtime over all perturbations of each sample x is considered, we observe from the second column that the last 3 algorithms never feature as the slowest on average for any input. This emphasizes the advantage of performing perturbation studies as it allows us to analyze the general performance of the algorithm by averaging over several input graphs in the same neighbourhood.

6 Conclusion

In this work, we described six approximation algorithms for MSTSP that are simple, easy to implement, practical and scalable. Further, we performed a perturbation analysis of the algorithms to study their usability in real-world scenarios. We observed that while these algorithms may have bad worst-case time complexity and output accuracy, analysing their average performance across a set of perturbed inputs reveals their practical efficacy. While certain algorithms may have exponential worst-case complexity, they may be useful in practice. For example, while the 2-opt algorithm has worst-case exponential runtime, its expected runtime is polynomial as the worst-case instances are sparse, rare and not clustered together.

Perturbation analysis of these algorithms has also revealed trade-offs that must be taken into consideration when evaluating the performance of an algorithm. We found that while the Pure 2-opt and Randomized 2-opt algorithms provide approximations that are closest to the scatter bound, they are relatively unstable as their output predictions show large deviations for small perturbations to the input graphs. On the other hand, the Dirac algorithm is highly stable and shows only minor variations under input perturbation. While the Naive Greedy algorithm is the fastest, it provides bad approximations that are far off from the scatter bound. The Weave algorithms provide a relatively better balance between speed and quality of output. In conclusion, through this paper, we have presented a comprehensive study of different approximation methods for the MSTSP that highlight critical factors that must influence the choice of an algorithm in practical settings.

Our procedural philosophy can further be extended to the study of other NP-hard problems of practical importance.

References

[1] Z. H. Ahmed, *A comparative study of eight crossover operators for the maximum scatter travelling salesman problem*, International Journal of Advanced Computer Science and Applications, 11 (2020).

[2] E. M. Arkin, Y.-J. Chiang, J. S. B. Mitchell, S. S. Skiena, and T.-C. Yang, *On the maximum scatter tsp*, in Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '97, USA, 1997, Society for Industrial and Applied Mathematics, p. 211–220.

[3] A. Croes, *A method for solving traveling salesman problems*, Operations Research, 5 (1958), pp. 791—-812.

[4] G. A. Dirac, *Some Theorems on Abstract Graphs*, Proceedings of the London Mathematical Society, s3-2 (1952), pp. 69–81.

[5] G. A. Dirac, *Some theorems on abstract graphs*, Proceedings of the London Mathematical Society, 3 (1952), pp. 69–81.

[6] M. Englert, H. Röglin, and B. Vöcking, *Worst case and probabilistic analysis of the 2-opt algorithm for the tsp*, Algorithmica, 68 (2014), pp. 190–264.

[7] I. Hoffmann, S. Kurz, and J. Rambau, *The maximum scatter TSP on a regular grid*, CoRR, abs/1512.02054 (2015).

[8] D. Johnson and L. A. McGeoch, *The traveling salesman problem: A case study in local optimization*, 2008.

[9] L. Kozma and T. Mömke, *A ptas for euclidean maximum scatter tsp.*, CoRR, abs/1512.02963 (2015).

[10] B. Manthey and R. Veenstra, *Smoothed analysis of the 2-opt heuristic for the tsp: Polynomial bounds for gaussian noise*, in International Symposium on Algorithms and Computation, Springer, 2013, pp. 579–589.

[11] A. P. Punnen, *The Traveling Salesman Problem: Applications, Formulations and Variations*, Springer, Boston, MA, USA, 2007, pp. 1–28.

[12] G. Reinelt, *Tsplib—a traveling salesman problem library*, ORSA journal on computing, 3 (1991), pp. 376–384.

[13] D. A. Spielman and S.-H. Teng, *Smoothed analysis: An attempt to explain the behavior of algorithms in practice*, Commun. ACM, 52 (2009), p. 76–84.

[14] I. Stock, *The maximum scatter tsp on a regular grid : How to avoid heat peaks in additive manufacturing.* February 2017.