GalaxyTSP: A New Billion-Node Benchmark for TSP

Anonymous Author(s)
Affiliation
Address
email

Abstract

We approximate a Traveling Salesman Problem (TSP) three orders of magnitude larger than the largest known benchmark, increasing the number of nodes from millions to billions. Previously, the World TSP dataset served as the largest benchmark for TSP approximation with 1.9 million cities. The dataset we use is currently the largest catalog of stars in the Milky Way, which we call Galaxy TSP, consisting of 1.69 billion stars. We use a divide and conquer approach for approximating the TSP by splitting the problem into tiles, approximating each tile, and merging the approximations. We learn to split tiles for efficient computation. We demonstrate our approach on optimization of space telescope target scheduling.

1 Introduction

The traveling salesman problem (TSP) arises in many application areas that involve graphs, such as circuit design and route planning. Since TSP is NP-hard, it is infeasible to solve exactly for large graphs; even approximation algorithms often require intractable running time to achieve adequately accurate solutions when run directly on the entire graph. Currently, the benchmark for the largest TSP problem is World TSP [3], which consists of 1.9 million nodes. However, larger datasets do appear in real-world applications and require novel approaches. In this work, we present Galaxy TSP, a new large benchmark dataset, for TSP approximation consisting of 1.69 billion nodes. Figure 1 shows our computed tour for the Galaxy TSP.

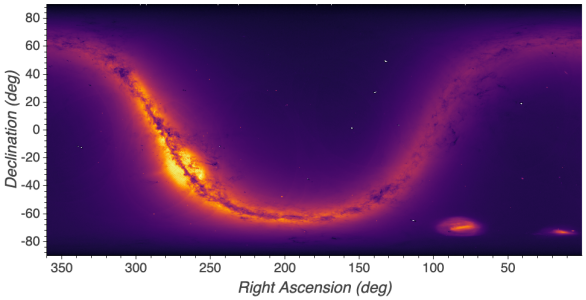

Figure 1: Computed tour for the Galaxy TSP consisting of 1,691,937,135 stars in the Milky Way galaxy projected into 2D. The graph is brighter where there are larger concentrations of stars. The U-shaped feature is the disk of the galaxy as seen from the Earth, which itself is within the disk. The brightest region along the disk, on the left, is the galactic center where the density of stars is the largest. The two isolated features at the bottom right of the plot are, from left to right, the Large Magellanic Cloud and the Small Magellanic Cloud, two satellite galaxies of the Milky Way.

At such a scale, existing approximation algorithms cannot be directly executed on the entire graph. The Lin-Kernighan heuristic (LKH) algorithm [6, 7], for example, would require nearly 30 years of

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

Dataset	Nodes
World TSP [3]	1,904,711
Geonames [12]	12,032,452
Galaxy TSP (Ours)	1,691,937,135

Table 1: Datasets used in this work and corresponding number of nodes. All three datasets use spherical coordinates. Our dataset is $888\times$ larger than the World TSP benchmark.

compute time. Instead, we propose a divide-and-conquer approach that runs LKH on smaller tiles of the graph, using machine learning to guide the tiling split, and merges the tile approximations.

Our algorithm approximates a solution to the TSP on three datasets, summarized in Table 1. World TSP [3] is a dataset and challenge comprising 1,904,711 locations of cities around the world. The dataset consists of latitude and longitude (lat/long) coordinates in the World Geodetic System (WGS84) format. The challenge has been ongoing since 2003, with a new recent record for the lowest TSP length as of June 2020 [3]. GeoNames [12] is a dataset containing 19 different attributes for each of over 12 million unique geographical features found on the Earth. We pre-process the dataset to contain only lat/long coordinates in the WGS84 format. Galaxy TSP, based on the Gaia data release 2 (DR2) [2, 5], is a catalog of 1.69 billion stars in the Milky Way galaxy, consisting of their position and other properties. Here we use the stars' two angular coordinates which describe their position in the sky. Those positions are used by astronomers to plan observations, which are optimized for the usage of telescopes by minimizing the angular distance between sequential stars (or targets). Observations are currently planned for thousands of targets and future space telescopes may require optimizing for millions of targets.

Applications of TSP in astronomy deal with efficient scheduling of telescope observations, whether on the ground or in space. The slew time required to move the telescope from one target and position it on the next target in a sequence of observations is a significant overhead in these observations. Therefore minimizing that time and making the sequence of observations slew-optimized is critical for the efficient use of telescope time. An early solution in the context of observations in radio astronomy was developed by first assigning targets to a partitioned region of the sky and then visiting targets within each region [8]. This algorithm used brute force search for regions with fewer than 10 targets and simulated annealing for more dense regions. Telescope scheduling has had particular significance for research in gravitational waves, which are transient events where the optically observable signal of the source of the waves appears soon after the gravitational waves signal. Therefore detecting the optical signal requires quick follow-up observations of all targets in the area of the sky from which the gravitational waves signal originated. Addressing a further-constrained scheduling problem, other work [11, 10] considers that certain tiles of the sky require longer exposure times than others. Divide and conquer approaches to TSP have considered problem instances of up to 1 million nodes [9] using parallel processing. Previously, the World TSP dataset served as the largest benchmark for TSP approximation with 1.9 million cities. In contrast, we consider a dataset with 1.69 billion nodes.

2 Dataset

The Galaxy TSP dataset is based on the European Space Agency (ESA) Gaia space observatory [2]. Gaia, launched in December 2013, is continuously scanning the sky with the goal of mapping the position and other properties of over a billion stars (which is an order of 1% of the stars in the Milky Way galaxy). Here we use the second release of Gaia data (DR2), published in 2018 [5]. We have downloaded the Gaia DR2 data from the online ESA Gaia Archive, and extracted the on-sky position of the stars, consisting of two angular coordinates: Right Ascension (RA) and Declination (Dec). Those coordinates of a position on the sky are conceptually similar to the latitude and longitude coordinates of a location on the Earth. RA ranges between $0 - 360$ degrees, and Dec ranges between -90 and 90 degrees. Figure 1 shows the density maps of stars in the Gaia DR2 catalog. The large U-shaped feature represents the disk of the Milky Way galaxy, whose location is not aligned with the (RA, Dec) coordinate system. The brightest region in the figures, at (RA, Dec) $\approx (266, -29)$, is the center of the Milky Way galaxy, with the largest density of stars. Also seen in the figures, in the bottom-right area, are two isolated bright regions which are nearby small galaxies, the Large Magellanic Cloud (brighter) and the Small Magellanic Cloud (fainter). We consider pairwise spherical distances between stars.

3 Methods

Running a TSP approximation algorithm, such as LKH, 2-Opt, or Christofides, on the entire graph of 1.69 billion nodes is infeasible. The estimated running time of LKH, the best existing large scale TSP approximation algorithm, on our dataset would be 30 years of computation on a cloud compute instance as shown in Figure 4, which is infeasible. Running LKH in parallel on a few hundred instances would also be inefficient and very expensive. We therefore take a divide-and-conquer approach using machine learning to efficiently handle this extremely large scale approximation. Splitting the entire graph into a uniform grid is inefficient. We divide the problem into sub-problems by adaptively splitting tiles of the dataset into four sub-tiles and efficiently solving the sub-tiles, as shown in Figures 3 and 2. We fit a quadratic to the computation time as a function of the number of nodes, for super-tiles, sub-tiles, and their difference. This is a classical time-accuracy trade-off, sacrificing accuracy for efficient computation and the ability to utilize additional hardware resources. Our approach consists of three parts:

1. **Split**: We first split the entire TSP graph into tiles using a quadtree, shown in Figure 2, learning the decision of whether or not to split a tile.
2. **Approximate**: Next, we approximate a solution for each of the tiles. The approximation is performed in parallel on individual tiles using LKH [1]. Approximating many relatively small tiles is computationally feasible.
3. **Merge**: Finally, we merge the approximations together, resulting in an approximate tour for the entire dataset of 1.69 billion stars.

Split. The input is the set of all stars in the graph. We use a quadtree [4] to partition the space into tiles. Starting with a single tile T containing all stars in the graph, tiles are recursively split into four subtiles of equal size $S_{T_1}, S_{T_2}, S_{T_3}$ and S_{T_4} governed by a splitting criterion. We use a quadtree with at least two million nodes per tile to find a baseline tour length and binary classification on a set of tile features to learn whether to split, as shown in Figure 2. We experiment with a criterion for deciding if a tile should be split. Our first approach is to split any tile with more than 2 million nodes. We improve on this by learning how to split from the data: deciding whether to split a tile by binary classification on extracted graph features from each supertile. As an example of a result of our splitting method we compare between a fixed criterion and our machine learning criterion for the Small Magellanic Cloud shown on the bottom right corner region in Figure 2. Using the fixed splitting criterion (left) results in 7 tiles; whereas using the learning criterion (center) results in four tiles. The length of the tour using the fixed criterion for this region is 63,093,867 arc-seconds; whereas the length of the tour using our algorithm for this region is 63,012,660 arc-seconds.

Approximation. There are many heuristics and approximation algorithms available for solving the TSP. These include the Branch and Cut algorithm, 2-opt, Farthest Insertion, and LKH. We use LKH due to the availability of a fast approximation for geographic coordinates and the small optimality gap. Given a graph and a tour, LKH searches for improved tours by swapping subtours until no additional improvements are made. As shown in Figure 3 time to find an approximation using LKH is quickly intractable with increasing number of nodes. A tile containing 500,000 stars runs in approximately 30 minutes while a tile containing 3,000,000 stars runs in approximately 250 minutes.

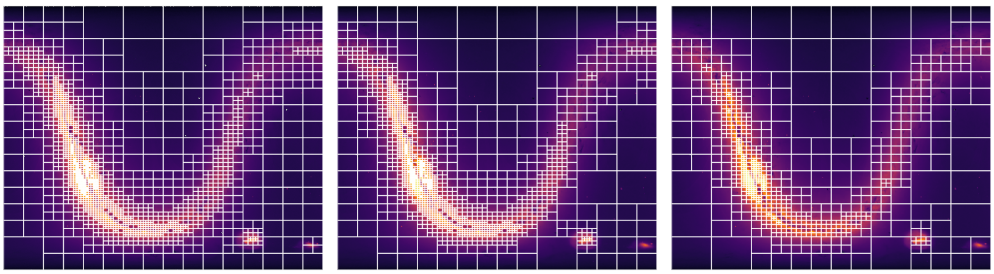

Figure 2: (Left) Tiling using fixed splitting criterion of 2 million stars per tile. (Center) Tiling based on learning a splitting criterion from data. (Right) Tiling using a fixed criterion of 5 million stars per tile. Notice the differences in tilings of the Large and Small Magellanic Clouds at the bottom right.

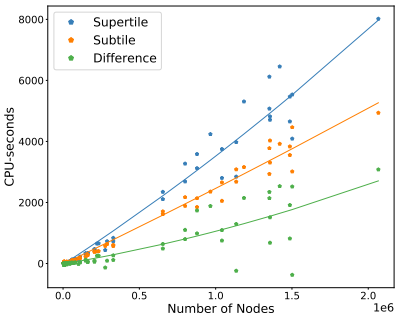

Figure 3: Running times of LKH for graphs of sizes 100-2,000,000 nodes. Plot shows time saved by approximating supertiles (of four tiles) directly using LKH as well as running time of the split, approximation, and merge processes.

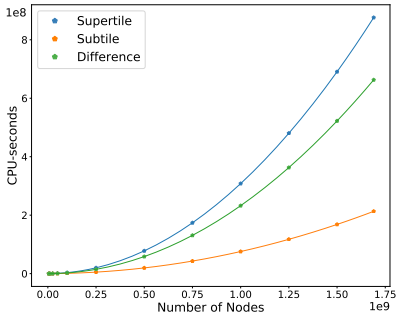

Figure 4: Time complexity of LKH is quadratic, $O(n^{2.2})$. Running LKH on Galaxy TSP would take around 30 years of computation compared with around 30 minutes for a tile with 500,000 nodes.

Merge. We merge the approximations of four individual subtiles $S_{T_1}, S_{T_2}, S_{T_3}, S_{T_4}$ into an approximation of a single merged tile T using the method proposed by Mulder and Wunsch [9]. Only k nodes in a tour nearest the tile boundary are considered as possible join points and serve as tile tour entry points. Using this frontier is necessary as using every star in a tile would change time complexity of the merge from $O(n)$ to $O(n^2)$.

Algorithm. Our approach begins by splitting the graph from coarse-to-fine, then approximates individual small tiles, and finally merges the tiles from fine-to-coarse resolutions. Given an input graph, the algorithm first splits the graph into sub-tiles using a quad tree. An approximation algorithm, LKH, is then run on each tile to approximate a tour for that tile. Starting with the finest level of the quad tree hierarchy and iterating upward, all tile groups in a hierarchy level are merged. Once the top level is merged, a single tile remains, which is the approximate tour of the entire graph.

4 Results

On the World TSP dataset, our approach splits the problem into 7 tiles with a mean of 272,101 nodes each. Running LKH on the tiles required a maximum of 779 seconds on a single tile and a total time of 3027 seconds. Our tour length was 7,587,334,983 meters. In contrast, the current record on this dataset has a tour length of 7,515,770,584, which is within 1% of the length of our tour. On the Geonames dataset, our approach splits the problem into 13 tiles with a mean of 925,573 nodes each. Our algorithm runs in 35,917 CPU-seconds, with a maximum of 6,400 seconds for a single tile, yielding a tour length of 19,718,226,839 meters. In contrast, running LKH on the full dataset resulted in a tour of 19,672,680,637 meters in 85,000 seconds, which is within 0.24% of our tour length.

Our Galaxy TSP dataset is 888 times larger than World TSP. The algorithm splits Galaxy TSP into $1,963$ tiles with an average of $861,913$ nodes per tile and maximum of $1,998,413$ nodes in a single tile. LKH requires a maximum of $17,257$ seconds for a single tile, and $8,128,680$ CPU-seconds, running LKH on tiles in parallel. Merging the tiles takes $344,455$ CPU-seconds. The final tour length is $15,780,464,837$ arc-seconds. Our learning algorithm splits Galaxy TSP into $1,753$ tiles with an average of $965,281$ nodes per tile and maximum of $4,732,478$ nodes in a single tile. LKH requires a maximum of $66,601$ seconds for a single tile, and $8,961,439$ CPU-seconds in parallel. Merging the tiles takes $307,587$ seconds. The final tour length is $15,779,149,332$ arc-seconds.

5 Conclusions

In this work we introduce a new dataset called Galaxy TSP and find an approximate solution for the TSP. Our approach is applicable to other large-scale TSPs and allows trading-off between accuracy and approximation time. We learn a splitting criterion from data which provides an adaptive solution. We demonstrate the benefits of adaptive splitting over a simple partitioning of space on the maximum number of nodes per tile. In future work we would like to use a graph neural network for learning the splitting criteria, improving generalization and providing fine-grained control over the tradeoff between accuracy and running time.

References

[1] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP Solver, 2006. http://www.math.uwaterloo.ca/tsp/concorde.html.

[2] Gaia Collaboration et al. Gaia data release 1. summary of the astrometric, photometric, and survey properties. *Astronomy & Astrophysics*, 595:A2, 2016.

[3] Bill Cook. World TSP, 2003. http://www.math.uwaterloo.ca/tsp/world.

[4] Raphael Finkel and Jon Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[5] Collaboration Gaia, AGA Brown, A Vallenari, T Prusti, JHJ de Bruijne, C Babusiaux, ÁL Juhász, G Marschalkó, G Marton, L Molnár, et al. Gaia data release 2 summary of the contents and survey properties. *Astronomy & Astrophysics*, 616(1), 2018.

[6] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[7] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[8] Walter Max-Moerbeck. Scheduling and calibration strategy for continuous radio monitoring of 1700 sources every three days. In *Observatory Operations: Strategies, Processes, and Systems V*, 2014.

[9] Samuel A. Mulder and Donald C. Wunsch. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks*, 16(5–6):827–832, 2003.

[10] Javed Rana, Shreya Anand, and Sukanta Bose. Optimal search strategy for finding transients in large-sky error regions under realistic constraints. *The Astrophysical Journal*, 876(2):104, 2019.

[11] Javed Rana, Akshat Singhal, Bhooshan Gadre, Varun Bhalerao, and Sukanta Bose. An enhanced method for scheduling observations of large sky error regions for finding optical counterparts to transients. *The Astrophysical Journal*, 838(2):108, 2017.

[12] Mark Wick. GeoNames, 2020. https://www.geonames.org.

6 Appendix

Algorithm 1 Approximate and Merge Tiles

Input: Tiles $T = (T_1, T_2, T_3, T_4)$
Output: Merged super-tile S_T
Compute $LKH(T_1), LKH(T_2), LKH(T_3), LKH(T_4)$
Compute tile centroids $(\mu_1, \mu_2, \mu_3, \mu_4)$
Let current shortest tour $S_T = LKH(T_1)$
for $t = 2, 3, 4$ **do**
  Find m nearest neighbors in S_T to μ_t: $N_{m \in S_T}(\mu_t)$ and let these be join points.
  for each $n_i \in N_m(\mu_t)$ **do**
    Find k nearest neighbors in tile T_t to n_i: $N_{k \in T_t}(n_i)$and let these be tile tour entry points.
    for each $n_j \in N_k(n_i)$ **do**
      Let n_a be node after n_i in S_T
      Let n_b be node before n_j in $LKH(T_t)$
      Remove edges $n_i \rightarrow n_a$ and $n_b \rightarrow n_j$
      Add edges $n_i \rightarrow n_j$ and $n_b \rightarrow n_a$
      Compute new merged tour and length and store
    end for
  end for
  $S_T =$ Shortest tour found, up to tile T_t
end for
return Merged tile solution S_T

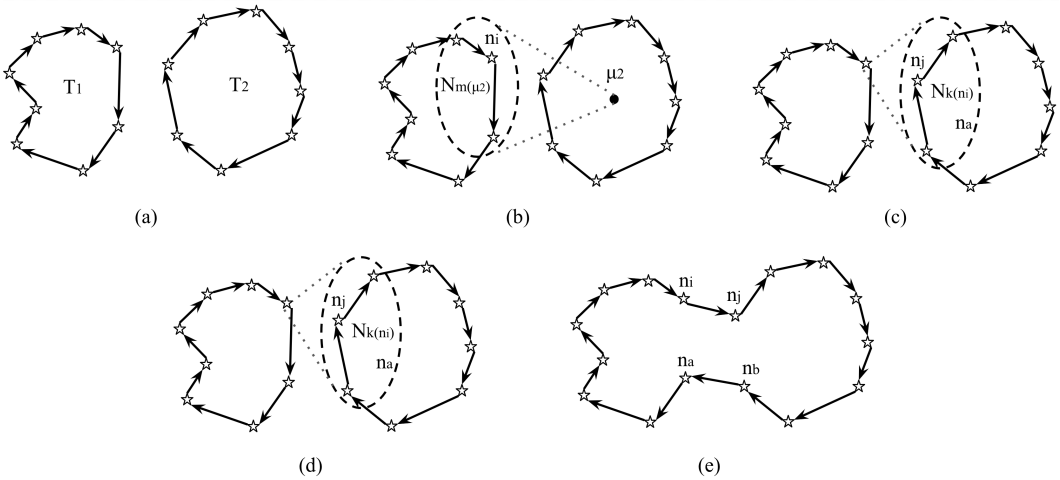

Figure 5: (a) Two adjacent tiles before a merge. T2 is a tile tour to be inserted into the tour of T1. (b) The nearest neighbors $N_m(\mu_2)$ to μ_2 in T_1. (c) The nearest neighbors $N_k(n_i)$ to n_i in T_2. (d) The edge from best join point n_i to the next star n_a and the edge to the best tile tour entry point n_j from the previous tour star n_b are removed. (e) Next, edges from n_i to n_j and n_b to n_a are added. This yields a single merged tour that includes each star from both T1 and T2.

Implementation details. All experiments are performed on a cloud instance with 96 vCPUs and 624 GB of RAM. Our feature set consists of the mean declination, mean right ascension, mean distance from the graph centroid, mean distance from the tile center, and the mean clustering coefficient across all nodes in the 20-nearest-neighbors graph of the tile. We train a logistic regression binary classifier to predict whether a tile should be split or not and use this model to guide our approximation algorithm. The model is trained on 168 super tiles of sizes between 2-5 million nodes using 5-fold cross-validation with an F1-score of 0.922. We set up a binary classification problem to predict whether the $\frac{\Delta dist}{\Delta time}$ from splitting the tile is in the bottom 75th-percentile, in which case, we split the tile. This percentile is chosen as a trade-off between the gained time and the increase in tour length.

Visual results. Figure 6 shows the final tour of World TSP, showing the full graph and zoomed in at x64 resolution centered on Long Island. Figure 7 shows the final tour of Geonames, showing the full graph and zoomed in at x64 resolution centered on Long Island. Figure 8 shows the final tour of Galaxy TSP zooming at increasing resolutions of x1, x4, x16, x64, x256, x1024, and x4096.

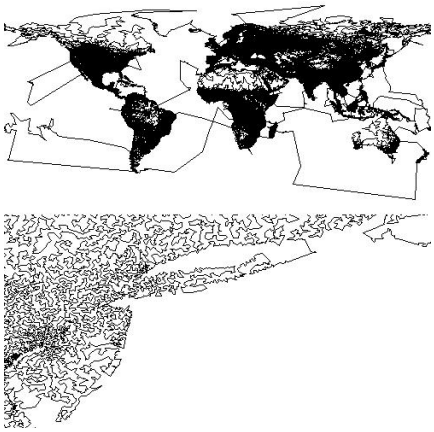

Figure 6: The final tour of World TSP. Showing the full graph and zoomed in x64 centered on Long Island.

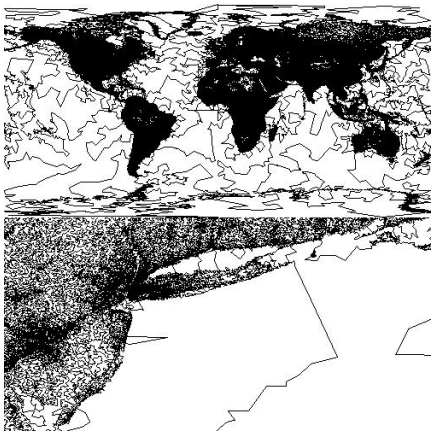

Figure 7: The final tour of Geonames TSP. Showing the full graph and zoomed in x64 centered on Long Island.

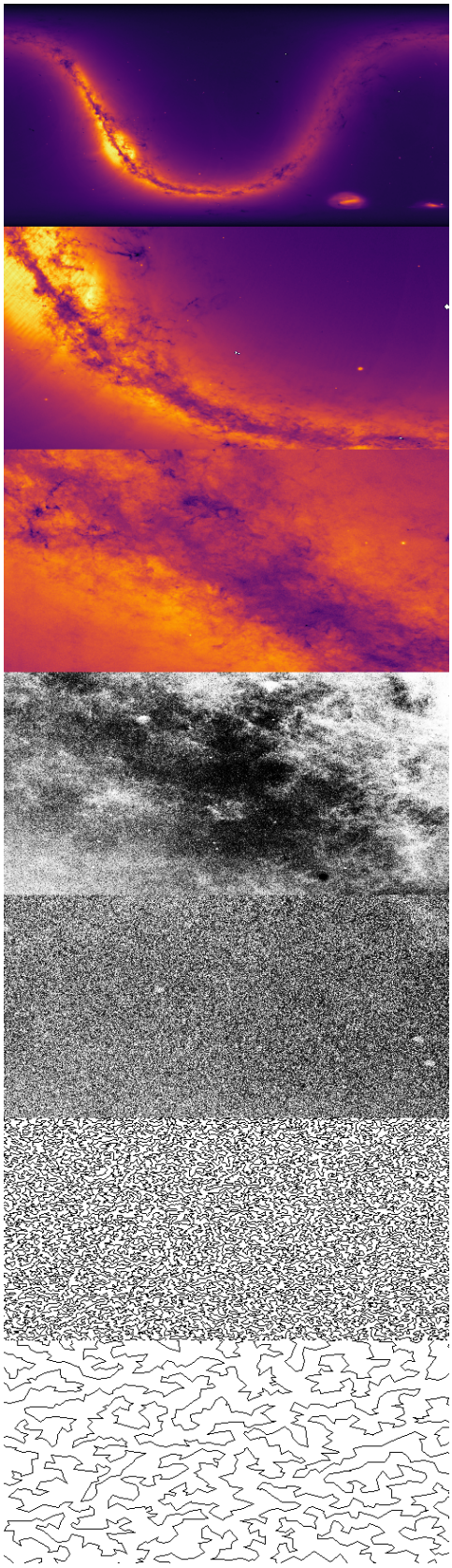

Figure 8: The final tour of Galaxy TSP, zoomed in at resolutions x1, x4, x16, x64, x256, x1024 and x4096.