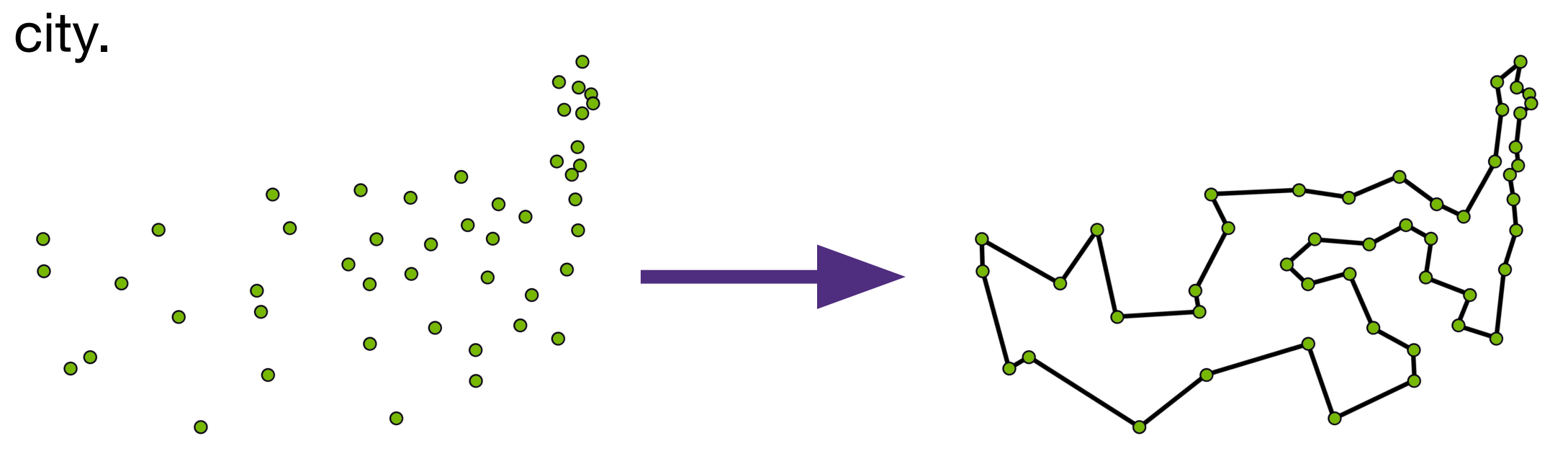
An N-body Approach to the Traveling Salesman Problem Utilizing GPUs

Johnny Seay, Bryant Wyatt, Jesse Crawford
Tarleton State University Mathematics Department

The Traveling Salesman Problem (TSP)

Given a list of cities and either their coordinates or the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the original city.



Applications

The Traveling Salesman Problem has numerous applications, including:

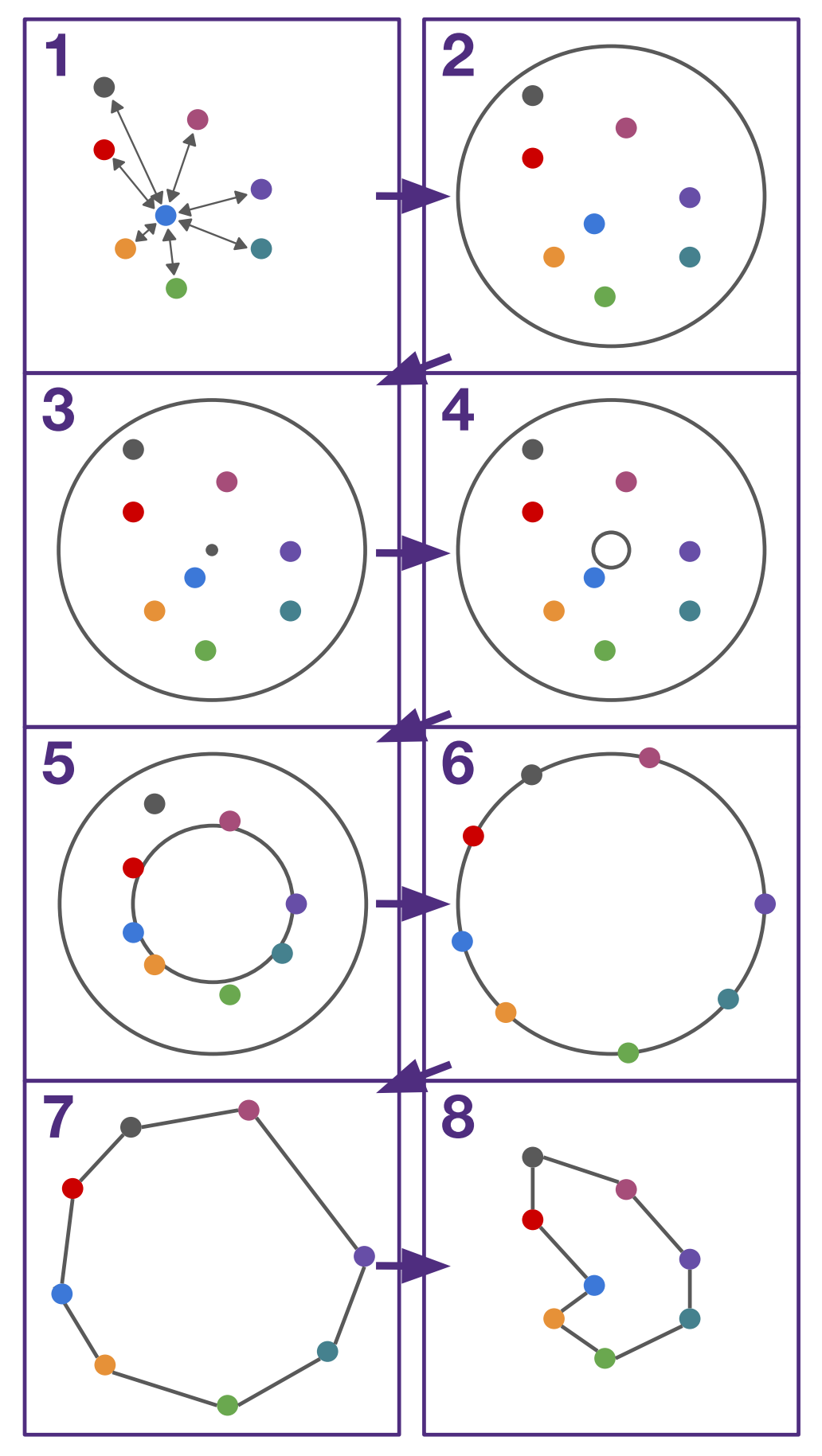
- Logistics
- Circuit Design
- Mapping Genomes
- Job Scheduling
- X-Ray Crystallography
- Organizing Data
- Aiming Massive Telescopes
- Guiding Industrial Machines
- Routing Autonomous Vehicles

Time Complexity

Running times for exact algorithms have a lower bound of $O(n^2 2^n)$ and upper bound of $O(n!)$. So, approximation algorithms are needed for most instances with practical applications.

The Basic Idea

1	Cities interact with each other through an attractive-repulsive force
2	Surround the cities with a bounding circle whose center is the centroid of the cities
3	Insert an additional circle at the centroid of the cities, with initial radius of zero
4	Increase inner circle's radius over time, until it touches the outer circle
5	The circles act as walls pushing on the cities
6	The cities get pushed out into a 1-dimension ring
7	Obtain the path from the ring by recording only the *order* in which they lie on the ring
8	Use the ordering to draw a path on the initial configuration of the cities



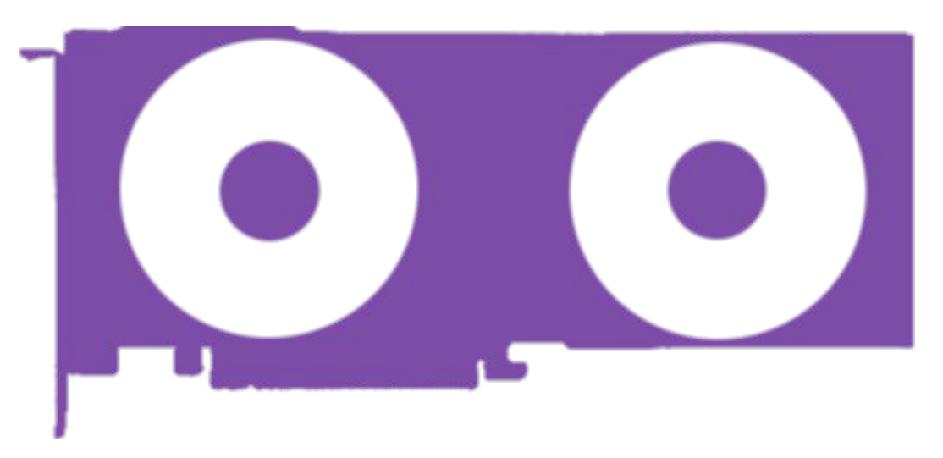
The GPU

Benefits of GPUs

- Allows us to parallelize our N-body simulation, resulting in much smaller running times.
- Runs with different parameter sets can be ran simultaneously, leading to minimal solutions quicker.
- Cost-effective resource for parallelizing code as they are inexpensive and fit easily into standard workstations.
- Can better visualize simulations in real-time.
- Makes it simple to scale the size of instances.

Our Usage

- The N-body simulation is implemented in CUDA C/C++ so that it can run in parallel on Nvidia GPUs.
- In the N-body simulation, each city has its own thread to perform the needed force calculations for that city.
- The generation of the density map is implemented in CUDA C/C++ so that it can run on Nvidia GPUs, allowing us to generate higher quality density maps.
- OpenGL is used to visualize everything in real-time.



Preliminary Results

To gauge the feasibility of our approach, we first created a simple model and ran a number of simulations on small randomly generated instances. We then compared our solutions to the solutions generated by the greedy nearest-neighbor algorithm and the convex hull algorithm. For a baseline, we also found the optimal solutions to these generated instances using the brute-force approach.

Using the optimal solutions, we found the percent errors for each run. For each different instance size, 100 instances were generated. The mean was taken over non-zero error instances.

$$\text{Percent Error} = \frac{\text{Algorithm Cost} - \text{Optimal Cost}}{\text{Optimal Cost}}$$

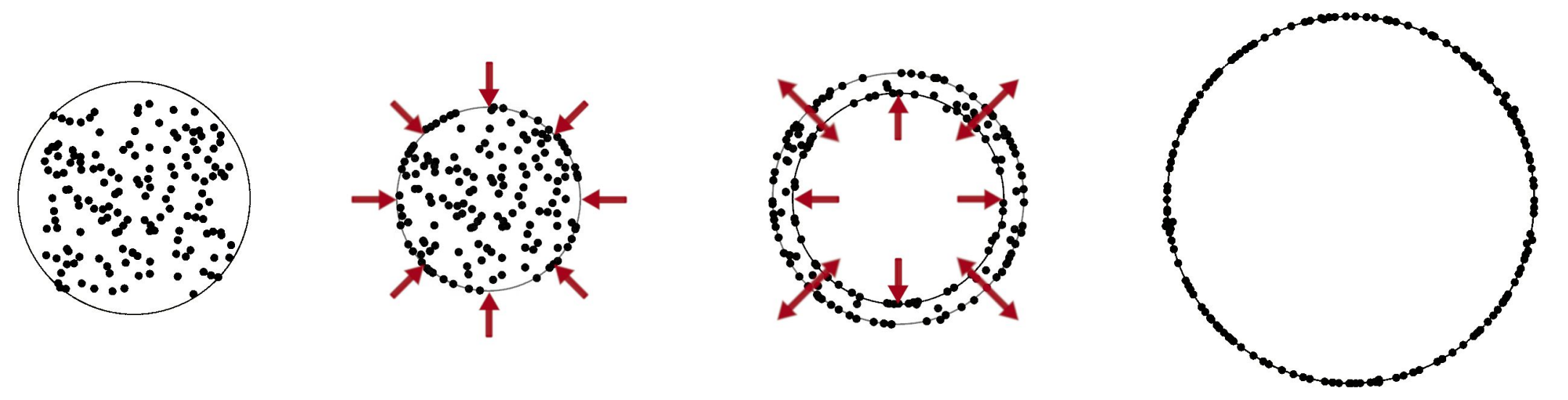
Instance size, *n*	Mean N-body Error	Mean Nearest-Neighbor Error	Mean Convex-Hull Error
$n=7$	1.06%	3.38%	0.06%
$n=8$	0.84%	3.31%	0.04%
$n=9$	1.70%	3.11%	0.19%
$n=10$	1.75%	2.71%	0.12%
$n=11$	2.73%	3.28%	0.26%

Variations

When running our N-body approach on large-sized or non-uniform instances, we observed that they exhibited undesirable behaviour that led to undesirable results. We developed two variations on top of our simple N-body approach to address both types of instances and the challenges they brought.

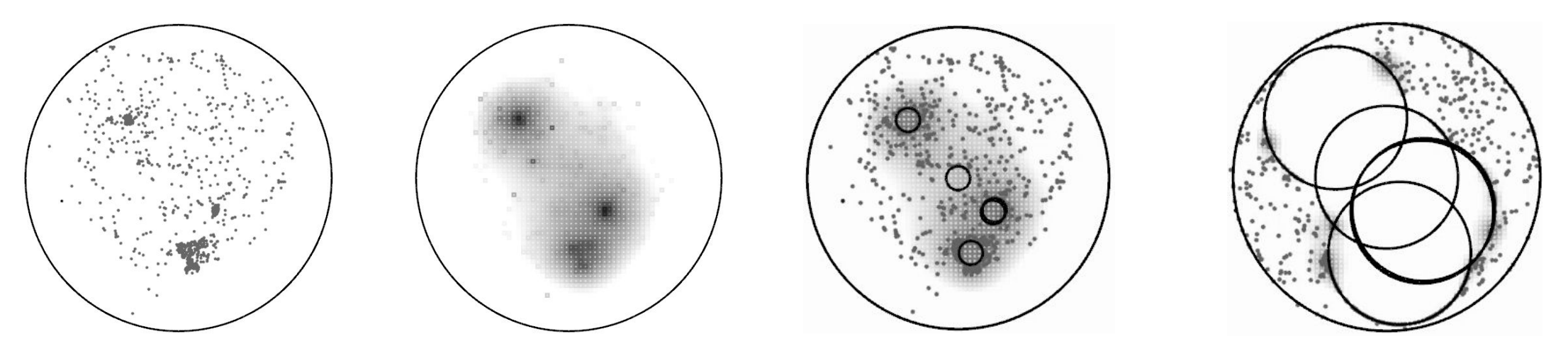
Pressure

- Keeping the pressure on the outer circle within some range helps produce better results.
- When the pressure gets out of range, the outer circle grows or shrinks to get back into range.



Bubbles

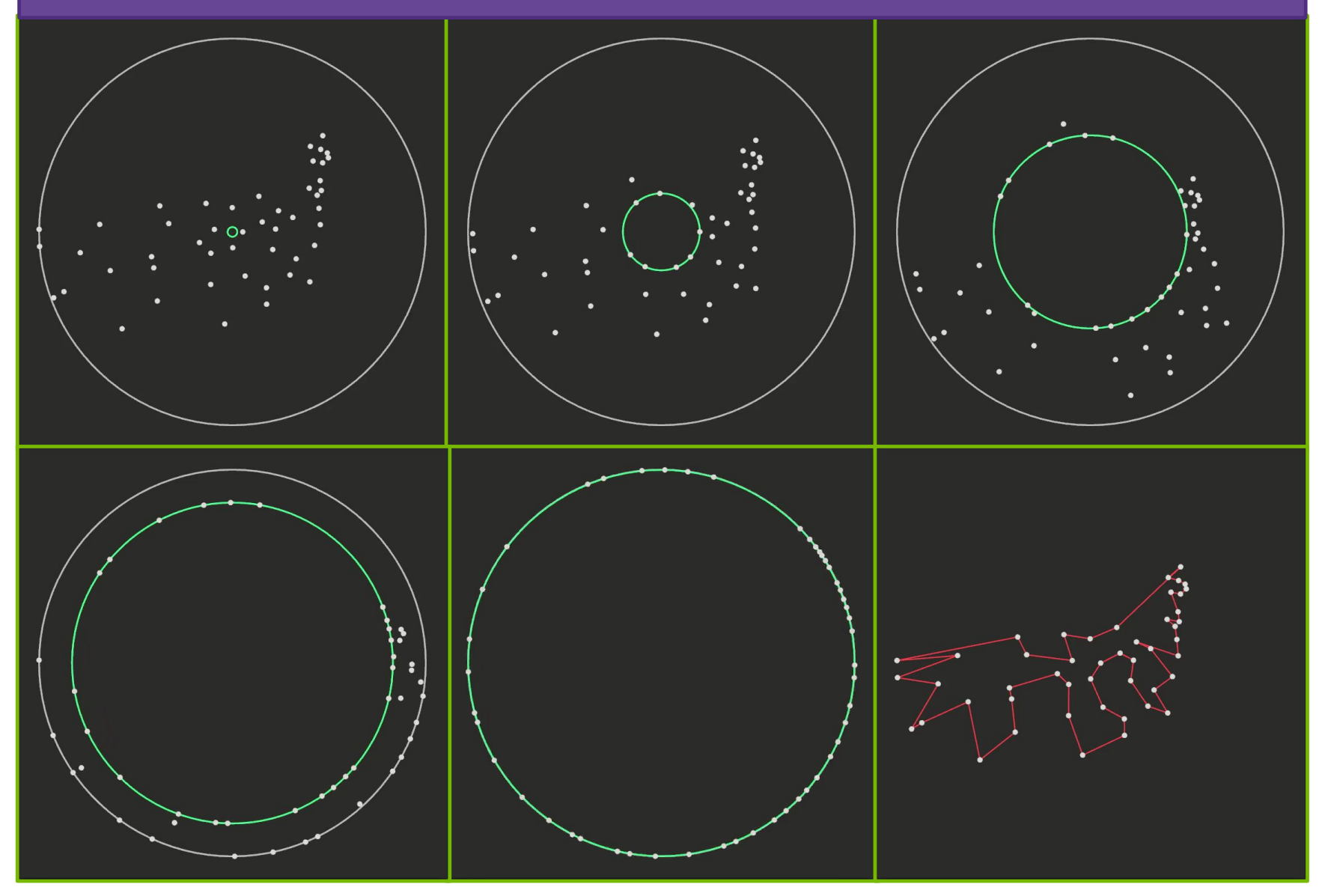
- Additional circles (or bubbles) are inserted in dense areas to break them apart.
- A heat/density map is generated in order to measure density for bubble placement.



Results

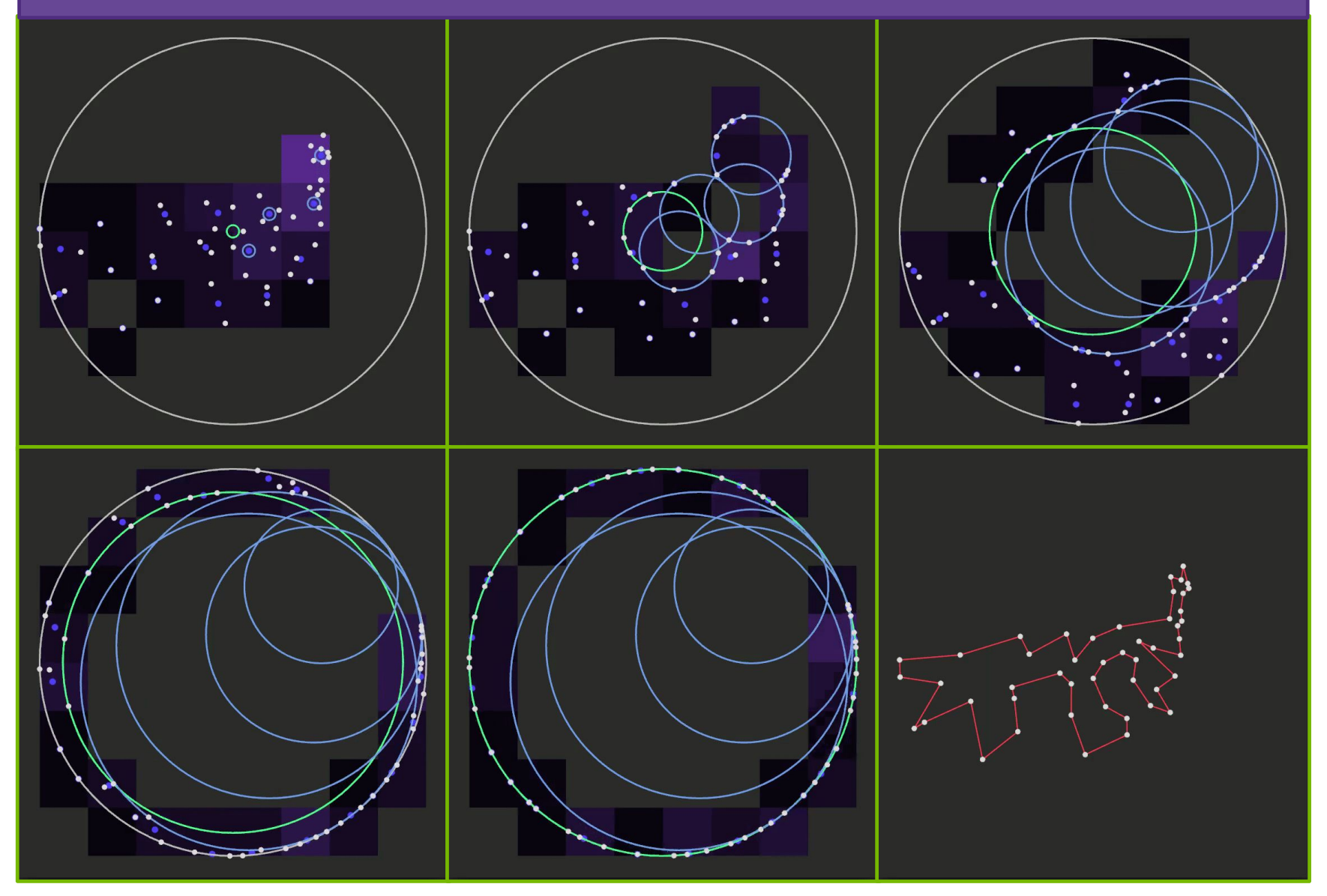
Dataset: *att48* **Optimal Cost:** 33523.708 units

Normal Run



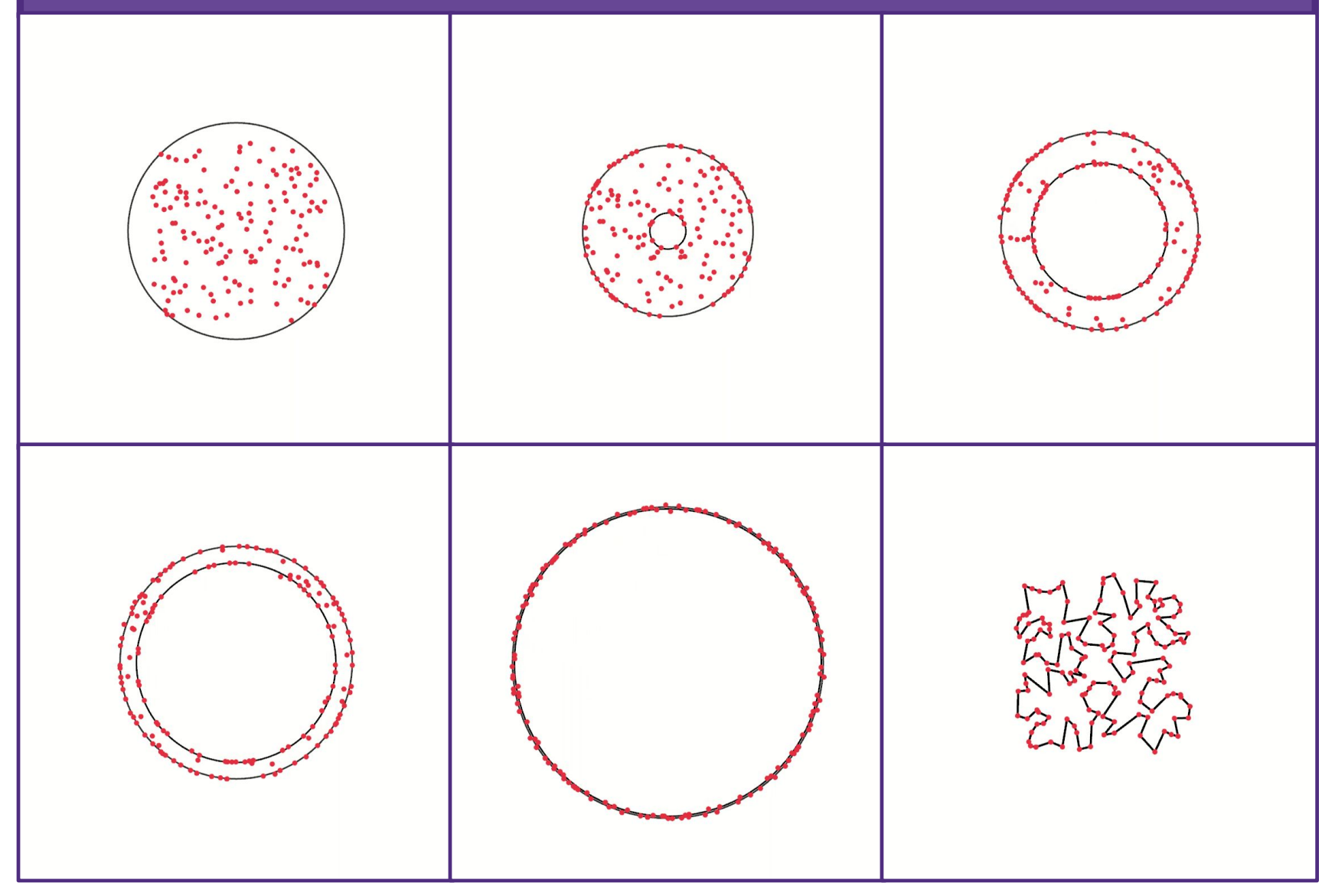
N-body Cost: 39104.852 units
Percent Diff.: 16.65%

Bubble Run



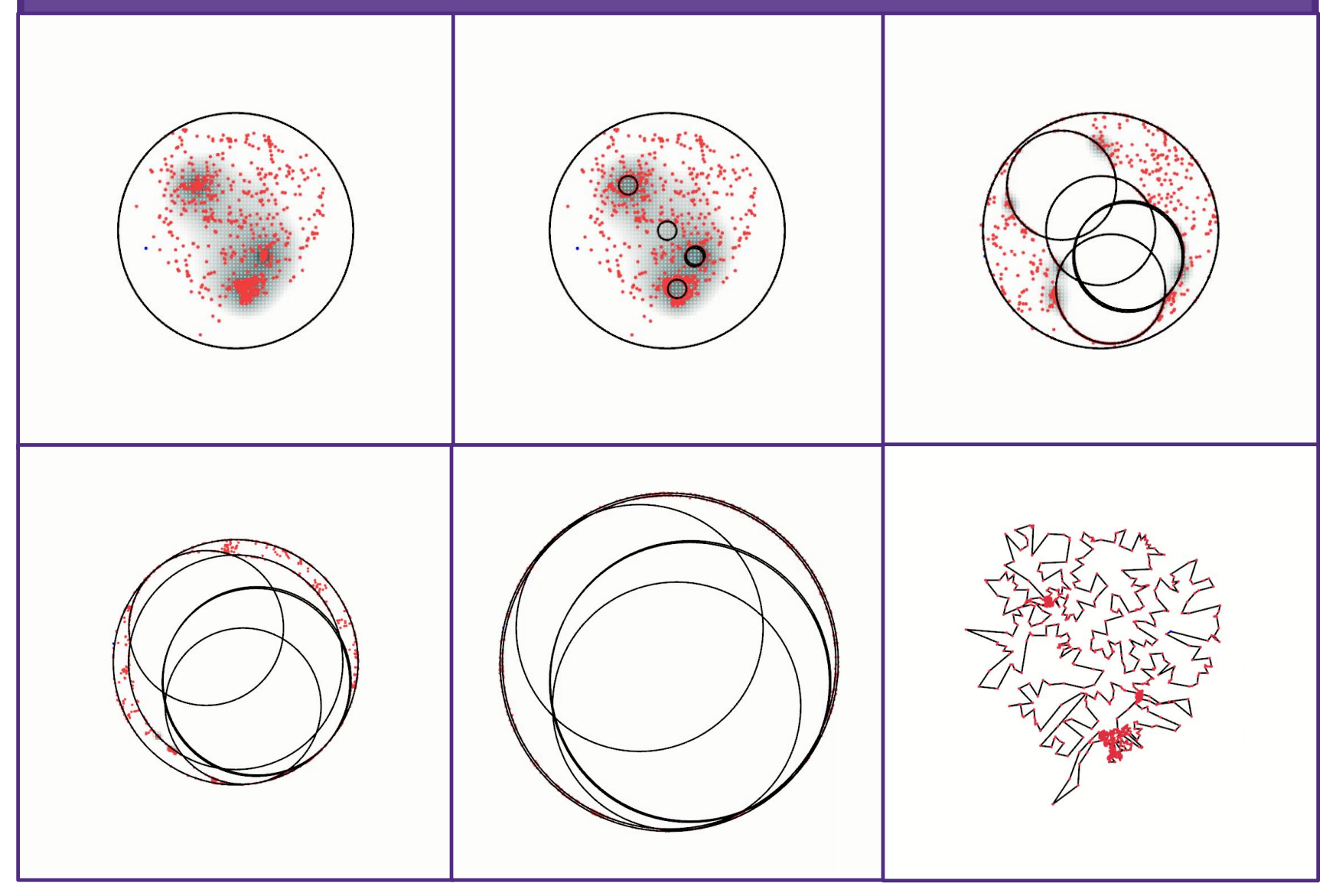
N-body Cost: 35725.371 units
Percent Diff.: 6.57%

ch150



Optimal Cost: 6532.28
N-body Cost: 7370.47
Percent Error: 12.83%

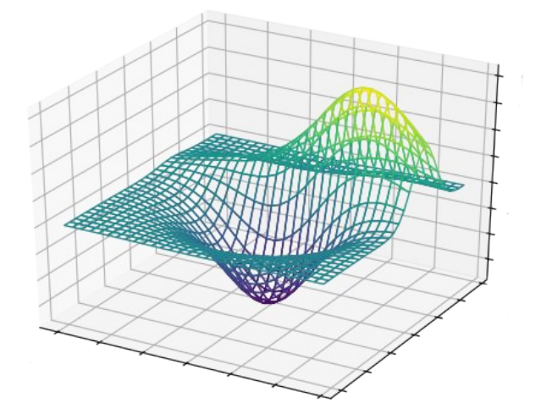
zi929

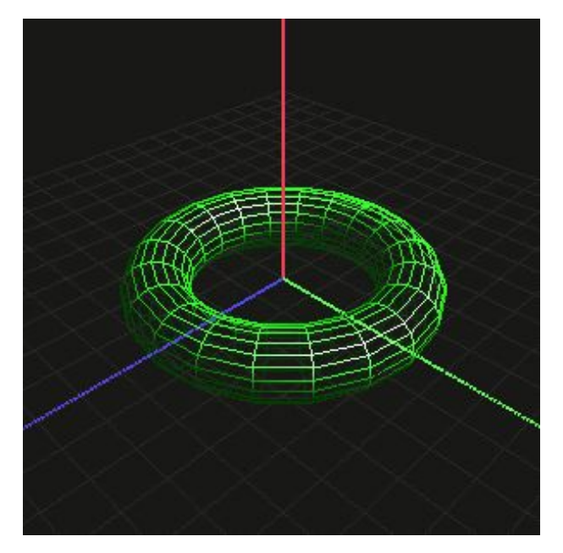


Optimal Cost: 95345
N-body Cost: 117921
Percent Error: 23.68%

Future Work

- Hyperparameter optimization
- Simultaneous runs in parallel
- Improve code efficiency
- Investigate other force functions
- Move into higher dimensional spaces





Acknowledgements/References

- Nvidia
- Tarleton's High Performance Computing Lab
- Tarleton's Office of Research and Innovation

Datasets shown can be found at:
http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html