Developing a More Efficient Fusion Reactor through Computer Modeling

New Mexico Supercomputing Challenge Final Report

April 1, 2015

Team 3

Albuquerque

Team Members
Jorge Antonio Perez
Che Olavarria Gallegos

Teacher
Jim Mims

Project Mentor
Dr. Mark Derzon
Dr. David Metzler

INTRODUCTION

This paper details our project *Developing a more efficient fusion reactor through computer modeling*. In this project, we establish a set of techniques to speed the simulation of the plasma in a fusion reactor. Second, we establish and explain the genetic algorithm we developed in order to optimize the designs of fusion reactors in a computationally-efficient manner. The following introduction explains why fusion is important, how computer modeling contributes to the development of fusion as a source of energy, and how this particular project contributes to research in fusion.

Please note that the section entitled "How this project contributes to research in nuclear fusion" directly concerns our project.

Why is fusion important? [Background]

Since the industrial revolution, one of humanity's greatest concerns is the generation of usable energy to power our various technologies. If we were able to generate significantly more usable energy easily, then a number of world problems, such as the lack of fresh water in some areas near the ocean, would become less problematic.

Most energy is currently produced by the combustion of oil, coal, or natural gas. Some is produced through wind energy or solar energy, and some countries, such as France, produce most of their electrical energy from nuclear fission. Combustion of oil and coal damages the environment unless precautions are taken; and in addition, along with natural gas, they contribute to climate charge. Wind and solar have a smaller environmental impact, however they are not always reliable. Fission power does not produce greenhouse gasses – although mining it does – and is, in fact, dependable; but current fission designs suffer from the danger of a reactor melt down (thankfully, an unlikely event) and the problem of safely storing spent fuel, which is highly radioactive.

By comparison, fusion power has significant overlapping advantages: 1] it is dependable, like coal and oil. 2] Fusion does not produce slowly decaying radioactive waste. 3] Fusion reactors are not capable of any sort of "meltdown" or of producing widespread fallout; and 4] chemicals released from a fusion reactor would be small in quantity and limited to hydrogen and helium – both of which are considered to be safe elements. Global reserves of deuterium, the main fuel in fusion, would last humanity billions of years at current consumption levels (Ongena).

Fusion power is currently not being used as an energy source because no one has succeeded in building a reactor capable of generating more energy than powering it consumes. This is expected to charge in the near future, however. ITER, a fusion reactor currently under construction in France, plans to start Deuterium-Tritium fusion by 2027. ITER's estimated output

will be 500 MW, and will produce 10 times more power than it consumes ("Facts and Figures"). In another example, Lockheed Martin's Skunkworks say that they are currently exploring a compact fusion reactor which confines plasma using a magnetic bottle ("Compact Fusion").

Computer Modeling in relation to Fusion Power [Background]

Constructing fusion reactors for research can be prohibitively expensive, especially if they are of the scale necessary to produce more energy than they consume. ITER, the aforementioned experimental reactor, has an estimated price tag of 13 billion euros, and demonstrates the monetary cost of fusion research ("Facts and Figures").

In many cases, modeling a fusion reactor within the confines of a computer is faster, safer, and less expensive than building a real model. This accelerates the design cycle, allowing scientists and engineers to test more hypothetical designs in a shorter period of time. Better yet, a simulation can help to ensure that a specific reactor design will, in fact, *work* before significant investments are made into building the actual design.

Purpose: How this project contributes to research in nuclear fusion

This project is focused at developing techniques and algorithms which reduce the number of computational resources required to approximate the behavior a plasma under the conditions present in a fusion reactor. Beyond this, it also focuses on demonstrating an algorithm capable of optimizing the design of a fusion reactor with relative efficiency.

Our Simulation

Our goal is to develop a computer program capable of simulating a plasma contained by magnetic fields, in conditions similar to those in a fusion reactor, for a few microseconds simulated time per few hours of real time on a single processor. Using a method similar to Particle-In-Cell [PIC], the current industry standard for plasma simulations, we are able to meet this goal with simulations of 50,000 to 100,000 super-particles. Investigation into the accuracy of the simulation reveals that because we failed to account for the magnetic field produced by currents within the plasma, our simulation is inaccurate. In addition, it is also possible that we used too few super-particles to represent the plasma.

A super-particle is a point-mass used in simulations to represent a clump of trillions of protons, electrons, or other particles. For example, if a super-particle represents a clump of a trillion protons, it will have a trillion times the charge of a proton, a trillion times the mass of a proton, a trillion times the momentum of a proton, and a trillion times the energy of a proton.

Our Optimization Algorithm

Our goal with this part of the project is to develop a genetic algorithm capable of finding a near-optimal solution for a fitness function which takes a "large" number of input variables, e.g., 10 to 20, in fewer than 100 generations, and with fewer than 10,000 function evaluations

per generation. We have far surpassed this goal (more details in the Results section of the paper).

How Genetic Algorithm Relate to Fusion Power

In order to find the optimal design of a fusion reactor, the genetic algorithm uses a simulation of the fusion reactor to estimate the input-energy to output-energy ratio of the reactor. This ratio is the "fitness", and thus the simulation simply becomes a "fitness function". The input variables for a fusion reactor could include placement of coils of wire, the physical dimensions of the reactor, the current flowing through those coils, etc.

In order to better judge our genetic algorithm, we compare it against a Monte-Carlo simulation designed to optimize the same fitness function as the genetic algorithm, within the same range. The Monte-Carlo simulation is our control algorithm.

Engineering Goals & Hypothesis

Our project does not set out to prove or disprove a theory about fusion power or about computer modeling in relation to fusion power. Rather, in this project we –

- Explore a variety of methods for quickly evaluating the efficiency of the design of a fusion reactor through computer simulation.
- Develop a genetic algorithm for optimizing a set of floating-point inputs extremely efficiently.

The main focus of this project is to achieve these goals.

Engineering Goals

Our goals are to –

- Create a genetic algorithm capable of approaching near-optimal solutions for fitness functions which take up to 20 inputs in 100 or fewer generations, with only a few thousand function evaluations in each generation. When optimizing the design of a fusion reactor, one function evaluation is effectively equivalent to one simulation.
- Create a program capable of modeling and evaluating the efficiency of one fusion reactor design on one core within 3 hours. The program will treat the plasma as a discrete collection of charged super-particles.

Hypothesis

A genetic algorithm designed to optimize over a set of floating-point inputs within a specified range will perform more than 10 times faster than Monte-Carlo methods in a similar range for sets of greater than 5 inputs; hence, if those inputs represent the design specifications - such as current, placement of wire coils, etc. - of a fusion reactor, then the genetic algorithm could find a near-optimal design for a fusion reactor faster than the Monte-Carlo method.

Second, simulations of fusion reactors can be made to calculate the efficiency of a reactor based on a set of floating-point input parameters. If these simulations are made to run in under three hours, while still producing precise results, genetic algorithms become a viable option for the optimization of designs of fusion reactors.

Methodologies and Materials

Materials

In the case of our project, the methodologies involved in the workings of the algorithms and in testing and collecting data are more important than the below materials.

- Compiler and editor for C# and C++ code: Visual Studio 2013
- Compiler and editor for Java code: Eclipse
- Computer on which to run simulations and test out the genetic algorithm

Algorithm Description

This section outlines a description of the algorithms we built over the course of the project. It describes how they work, what they do, and what purpose they serve. It also comments on their strengths and weaknesses, and describes improvements that can be made.

Code for simulating fusion reactors

The simulation is very similar to the Particle-in-Cell method [PIC], however it has a few changes. The simulation uses super-particles to represent the plasma. Each super-particle can be thought of as a single, massive, positively charged point-mass representing a clump of atomic nuclei. For example, if the simulation is of deuterium-tritium fusion, each super-particle would represent a clump of either deuterium nuclei or tritium nuclei.

Our code applies an implementation of the leapfrog method of integration to calculate the new position and new velocity of every super-particle in the simulation at each time-step. In our current simulation, each time-step currently lasts 0.1 nanoseconds. In order to approximate the force on the particle, the magnetic field at the location of the particle is calculated based on linear interpolation from a two-dimensional mesh. Forces due to collisions are calculated based off the Barnes-Hut algorithm, then scaled so as to conserve the change in kinetic energy of the super-particle per unit of mass. In contrast to our simulation, many PIC implementations use Monte-Carlo methods to approximate collisions.

Applying the leapfrog method

The leapfrog method is defined by the following equations:

$$\mathbf{s}_{i+1} = \mathbf{s}_i + \mathbf{v}_i\,\Delta t + \frac{1}{2}\mathbf{a}_i\,\Delta t^2$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{1}{2}(\mathbf{a}_i + \mathbf{a}_{i+1})\,\Delta t$$

$$\mathbf{a}_{i+1} = \frac{\mathbf{F}(\mathbf{s}_{i+1})}{m}$$

Where $\mathbf{a}_i$ is the acceleration at time t_i, $\mathbf{v}_i$ is the velocity at time t_i, $\mathbf{s}_i$ is the position at time t_i, $\mathbf{F}(\mathbf{s}_i)$ is the force at point $\mathbf{s}_i$, and $t_i = i\,\Delta t + t_0$

When simulating a plasma, the leapfrog method can be a bit tricky to apply. The leap-frog method requires knowing the new acceleration in order to calculate the new velocity:

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{1}{2}(\mathbf{a}_i + \mathbf{a}_{i+1})\,\Delta t$$

However, because the force on a charged particle is determined by the Lorentz force,

$$\mathbf{a}_{i+1} = \frac{q}{m}[\mathbf{E}(\mathbf{s}_{i+1}) + \mathbf{v}_{i+1} \times \mathbf{B}(\mathbf{s}_{i+1})]$$

calculating the new acceleration requires knowing the new velocity. We are able to solve this problem by creating a system of linear equations relating acceleration and velocity and solving that system in the code.

Calculating the magnetic field of the reactor using interpolation

Because the reactor in our simulation, or many of its components, is rotationally symmetric, the magnetic field of the reactor is pre-calculated on a two-dimensional mesh. Values for the magnetic field at a point are calculated by rotating the point so that it lies on the two-dimensional mesh of pre-calculated values for the magnetic field, then the magnetic field is interpolated by way of two-dimensional linear interpolation for values in between the mesh points. The interpolated value for the magnetic field is then rotated by the same angle the position of the particle was originally rotated when mapping the position of the particle to the mesh.

Magnetic fields are pre-calculated on the mesh through the application of the Biot-Savart Law over all coils of wire in the reactor:

$$\mathbf{B}(\mathbf{s}) = \frac{u_0}{4\pi} I \int_C \frac{d\mathbf{l} \times (\mathbf{s} - \mathbf{l})}{|(\mathbf{s} - \mathbf{l})|^3}$$

where $\mathbf{B}(\mathbf{s})$ is the magnetic field at point $\mathbf{s}$, u_0 is the permeability of free space (magnetic constant), I is the current, $\mathbf{l}$ is the curve representing the wire, and $d\mathbf{l}$ is the differential element of the curve representing the wire. The Biot-Savart Law essentially states that the magnetic field along a closed, curved piece of wire at a point is equal to the line integral over the entire loop of wire of the magnetic field generated by each segment of the wire.

In order to quickly and accurately numerically integrate the Biot-Savart Law, the code applies a method of dynamic numeric integration in which the number of wire elements used is about inversely proportional to the distance between the particle and the location along the loop of wire where the magnetic field is being calculated. Essentially, where the particle is close to the wire, a large number of wire elements are used for integration, but when the particle is

further away from the wire, only a small number of wire elements is used. The integral applies the midpoint rule, with each midpoint representing a section of wire. When more wire elements are needed, a section of wire is subdivided into 3 smaller sections, and when even more wire elements are needed, each of those subsections can be subdivided further.

The Barnes-Hut Algorithm

The Barnes-Hut algorithm is normally applied to simulations of the gravitational interactions between large masses because it provides an efficient and fast way to calculate the force on a mass due to other masses. Conventional gravitational simulations require calculating the force between every pair of particles, then summing all the forces on each particle. As a result, conventional methods commonly have a running time of $O(n^2)$, where n is the number of particles.

The Barnes-Hut algorithm, on the other hand, uses an octree to break up the simulation space (which is three dimensional). The octree is populated with the positions of particles in the simulation, and each node (or cell, as it is also called) in the octree also stores the mass of all the particles in that node or cell. On the next page is an image of a simulation with a cross section of the Barnes-Hut octree shown.

In our simulation of the plasma, we modify the Barnes-Hut algorithm so that it stores charge instead of mass, and so that it calculates the force only between particles in adjacent nodes [cells]. This calculation of force between adjacent particles approximated a collision. That force was then multiplied, or "scaled", by $r^{-2/3}$ so as to conserve the change in the kinetic energy of particles in a collision, where r is the ratio between the actual number of nuclei in the simulation and the number of super-particles that represent nuclei in the simulation.

Using the Barnes-Hut algorithm for plamas in our simulation

1) Place all the particles in the simulation within an octree.
2) Once the octree is constructed, for every node in the octree, add up the charge of all particles within that node, calculate the "center of charge" for each node", and save those values.
3) For a neutral plasma, use the barnes-hut tree to calculate the electric field of the nodes nearest to a particle, calculatet he force on a particle due to those fields, and multiply that force by the scale factor.

The Genetic Algorithm

Given a function that takes *n* inputs, a lower bound for each input, and an upper bound for each input, the genetic algorithm tries to find *n* floating-point values between the lower bound and upper bound which maximize the output of the given function.

It is not hard to apply this process to fusion reactors. The simulation we built determines the design for a fusion reactor based off of a set of floating-point inputs; the bounds for each

input are determined by what we could reasonably build; and the ratio between the intake energy of the fusion reactor and the output energy due to fusion after a certain time is the output of a simulation. This ratio between intake and output energy is called the efficiency of the reactor. When applied to optimize fusion reactors, the genetic algorithm would use a simulator for fusion reactors as a function it was trying to optimize, and then find the design with greatest efficiency.

If the actual efficiency of a fusion reactor is proportional to the efficiency calculated by the computer simulation of that reactor plus a constant –

$$E = mx + b$$

where *E* is the actual efficiency, *x* is the calculated efficiency, and *m* and *b* are constants determined for a specific class of reactor design such as tokomak reactors, or bottle reactors, and $m > 0$ – then the genetic algorithm can be applied without a calculation of *m* and *b*. The reason for this is simple. If $f(x_1, x_2, x_3 \dots x_n)$ and $g(x_1, x_2, x_3 \dots x_n)$ are functions such that

$$f(x_1, x_2, x_3 \dots x_n) = mg(x_1, x_2, x_3 \dots x_n) + b,$$

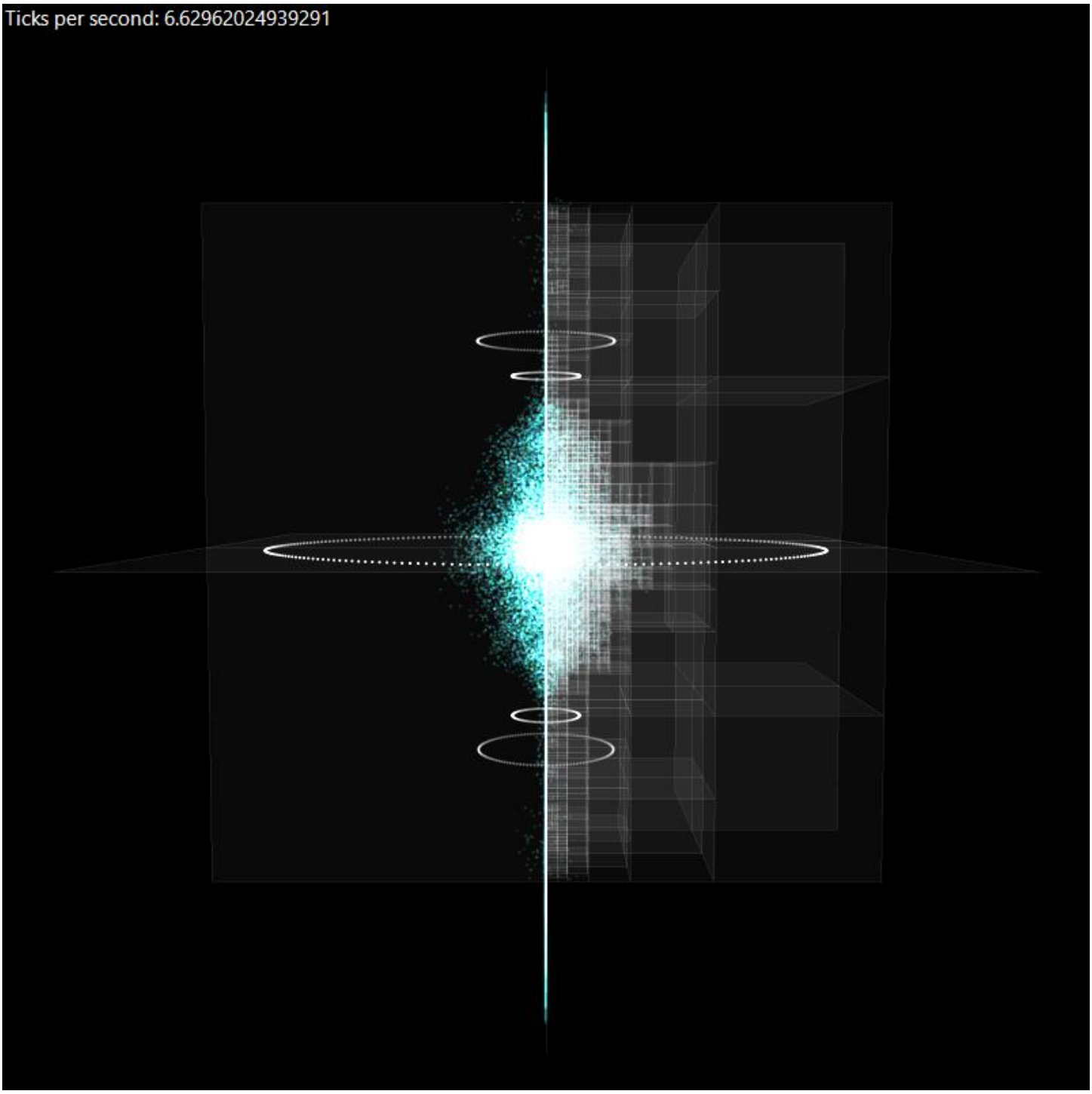
then the set of inputs $\{x_1, x_2, x_3 \dots x_n\}$ producing the maximum of $f(x_1, x_2, x_3 \dots x_n)$ is identical to the set of inputs producing the maximum of $g(x_1, x_2, x_3 \dots x_n)$, provided that $m > 0$.

How our Genetic Algorithm works for a given function that takes *n* inputs:

1) Choose a population size *p*, an upper and lower bound for each input, and a multiplier constant *m*. We found that the multiplier should generally be smaller than 0.5, or else the genetic algorithm will not converge.
2) Generate an initial population of *p* random sets of *n* inputs for the given function. Each input chosen should be between the upper and lower bounds.
3) Evaluate each set of inputs in the current population using the given function.
4) Select the 10 best sets of inputs – the 10 sets of inputs which scored highest under the given function.
5) Use the 10 best sets of inputs from the current population and the 10 best sets of inputs from the previous population to generate a bounding box. If there is no previous population (such as is the case when starting with an initial population, just use the 10 best sets of inputs from the current population).

   NB: the bounding box is not actually a bounding box. It is a box in *n*-dimensional space, where *n* is the number of inputs for the given function, whose size in each dimension is determined by the average distance between solutions within that dimension.
6) Multiply the size of the bounding box in each dimension by the multiplier.
7) Generate a new population of *p* random sets of *n* inputs. Each set of inputs in the new population is generated by randomly selecting and modifying one of the 10 best sets of inputs from the current population. The average size of a random modification to a set of inputs is determined by the size of the bounding box for the current population.

8) Now the new population becomes the current population.
9) Repeat Steps 3 to 8 until the genetic algorithm is halted. In general, the solutions in the new population will be better than those in the current population.



Caption: Above is an image of the simulation with the octree shown. The blue particles represent the plasma, the white rings of particles represent the loops of wires producing the magnetic fields which contain the plasma. Note that in the areas where there are more particles, the octree have more subdivisions. Only 7 layers of subdivisions are shown.

Results

The Simulation of the Fusion Reactor

We originally met our engineering goal of a few milliseconds of simulation time within 3 hours on a single core (CPU), however we learned that our simulation is inaccurate because it failed to account for currents flowing within the plasma. In a simulation with 30,000 super-particles, where the simulation itself is on a single core, we generally achieved speeds of around 6 "ticks" per second, or 0.6 nanoseconds of simulated time for every second of real-world time. Running time seems to vary proportionally to the number of particles in the simulation. The slowest part of the simulation was the calculation of collisions with the Barnes-Hut algorithm. Constructing the octree was fast, however calculating forces on particles was much slower.

Genetic Algorithm

The data for the genetic algorithm is shown in the graphs on the following pages. The genetic algorithm far outperformed Monte-Carlo optimization, and the more generations that elapsed, the better the genetic algorithm does in comparison to Monte-Carlo optimization. Initially, a smaller generation size performs better than a larger generation size.

Suppose that the median fitness found by the genetic algorithm after n function evaluations of the fitness function was x. The graphs plots the number of function evaluations it took Monte-Carlo optimization to have a 50% chance of finding a fitness better than x. We ran each algorithm 1000 times when collecting data.

For a population size of 1000, within or at 35 generations (or 35,000 function evaluations), the genetic algorithm performs better than 1,500,000 function evaluations with Monte-Carlo optimization for functions which take any number of input s less than or equal to 32 inputs.

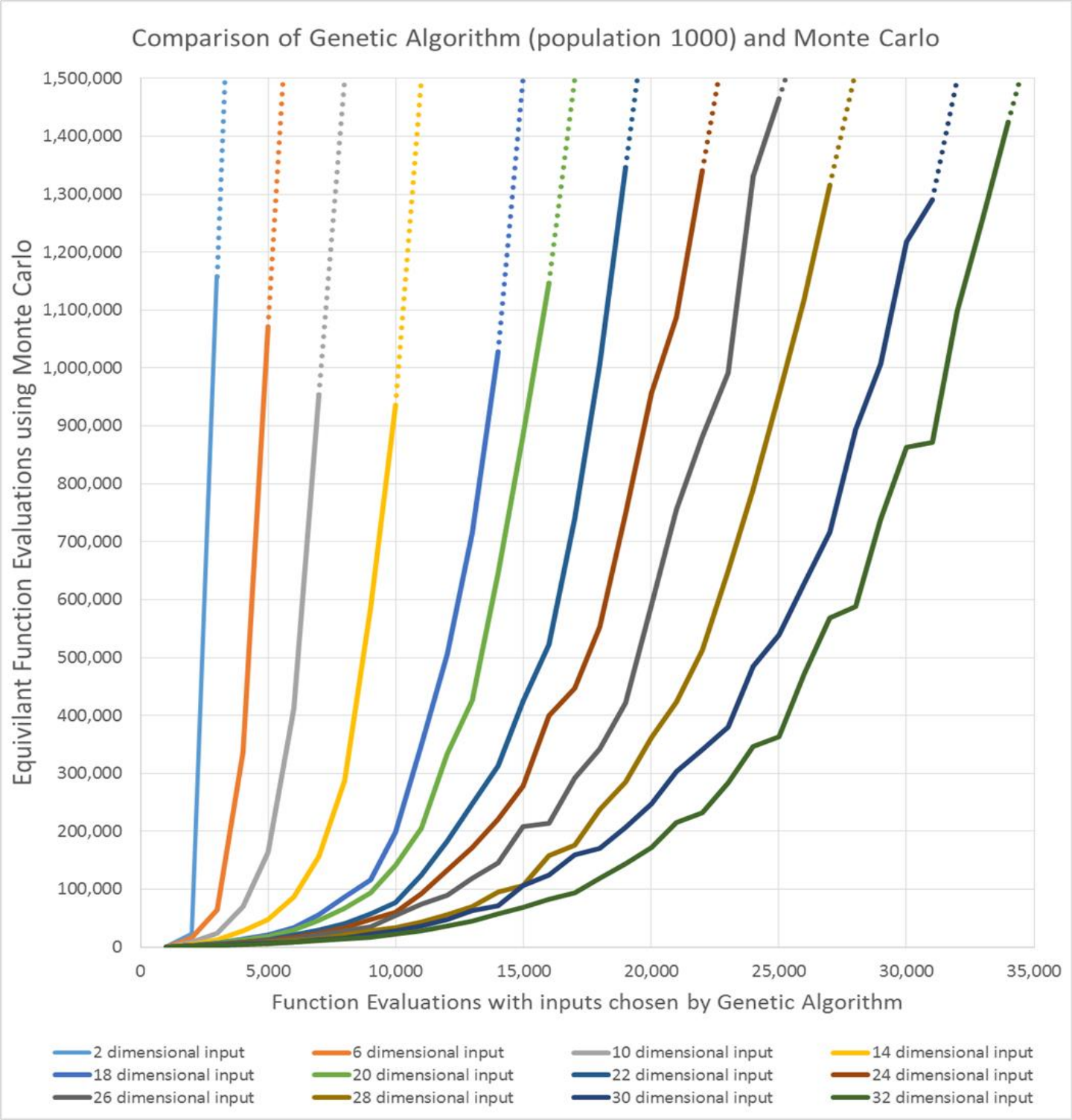


Figure 1: A comparison of the genetic algorithm to Monte Carlo optimization. Each line represents a different number of inputs. For example, "14 dimensional input" means an input that the both the genetic algorithm and Monte Carlo Optimization were searching for the combination of 14 floating-point numbers which maximized the output of the fitness function used for testing. The dotted line indicates projected values.

Takeaway: This graph shows that the genetic algorithm performs far better than Monte Carlo Optimization. For 30 dimensions, the genetic algorithm takes only 30,000 evaluations to find a value Monte Carlo Optimization takes 1,220,000 evaluations to find. Better algorithms require fewer evaluations

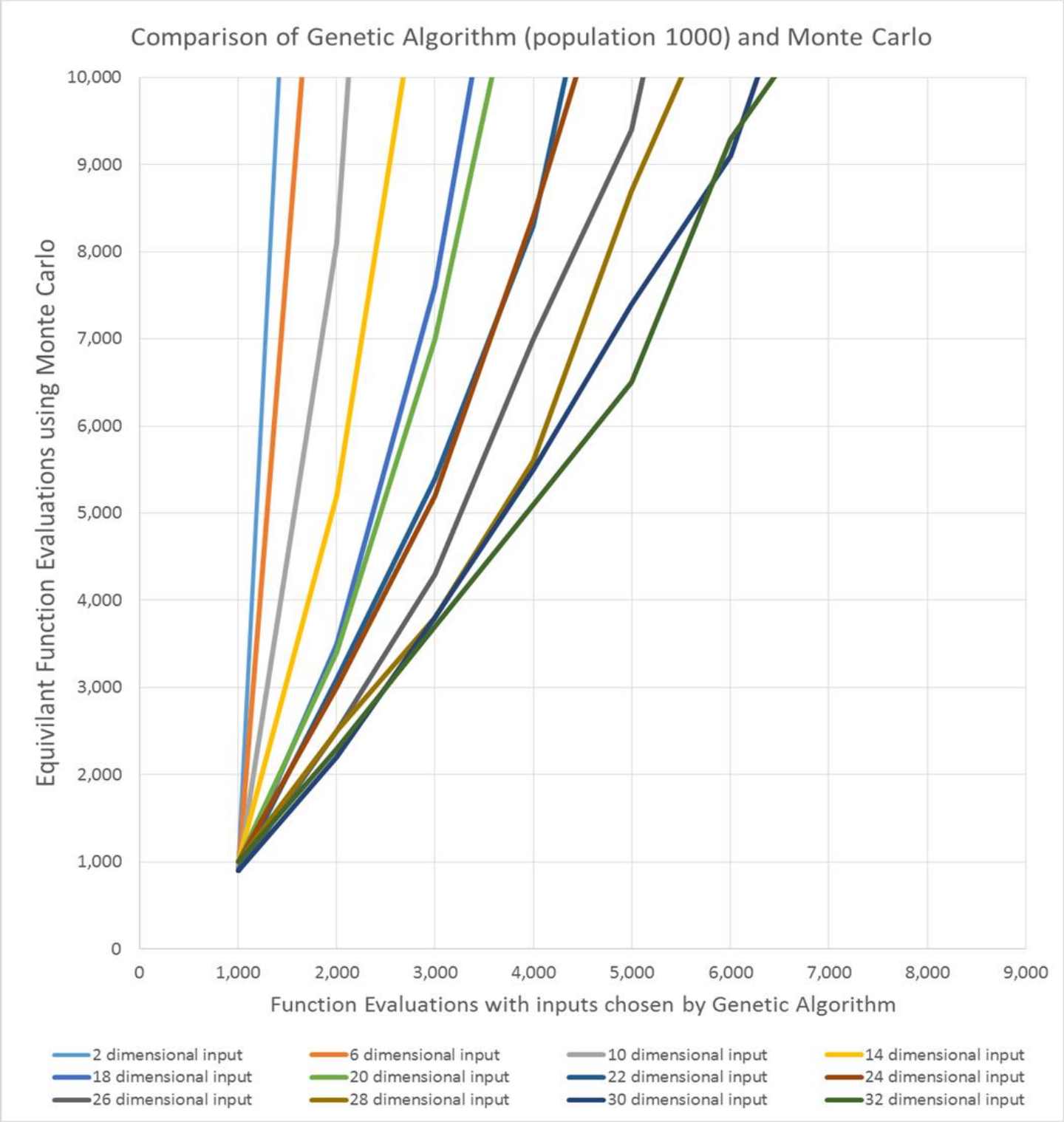


Figure 2: This graph is shows the same data on the previous graph, except at a 1:1 scale. It shows that, at worst, a genetic algorithm with a population size of 1000 per generation will perform as good as Monte Carlo optimization. A population size of 1000 means that each "generation" uses 1000 function evaluations.

Takeaway: This graph shows that, even if you only have the opportunity to run a small number of generations, the genetic algorithm will perform better than Monte Carlo optimization.

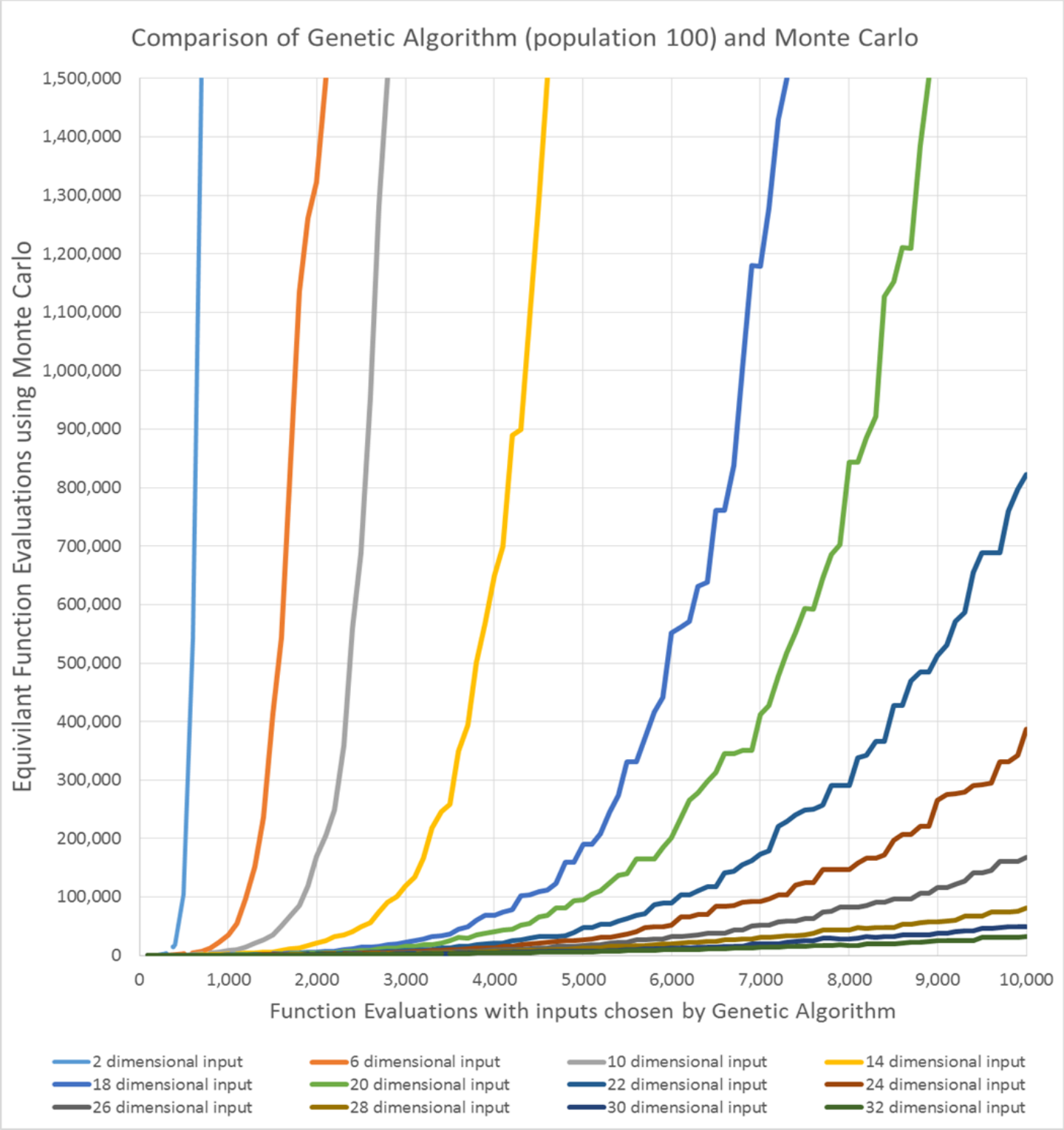


Figure 3: This graph compares the genetic algorithm with a population of 100 per generation (or 100 function evaluations per generation) to Monte Carlo Optimization. If you compare this graph to *Figure 1*, you can see that the genetic algorithm with a population of 100 per generation outperforms the genetic algorithm with a population of 1000 for all numbers of inputs smaller than 28.

Takeaway: the genetic algorithm performs extremely well even with a small population per generation, which might be the case if computing resources to simulate a fusion reactor were limited.

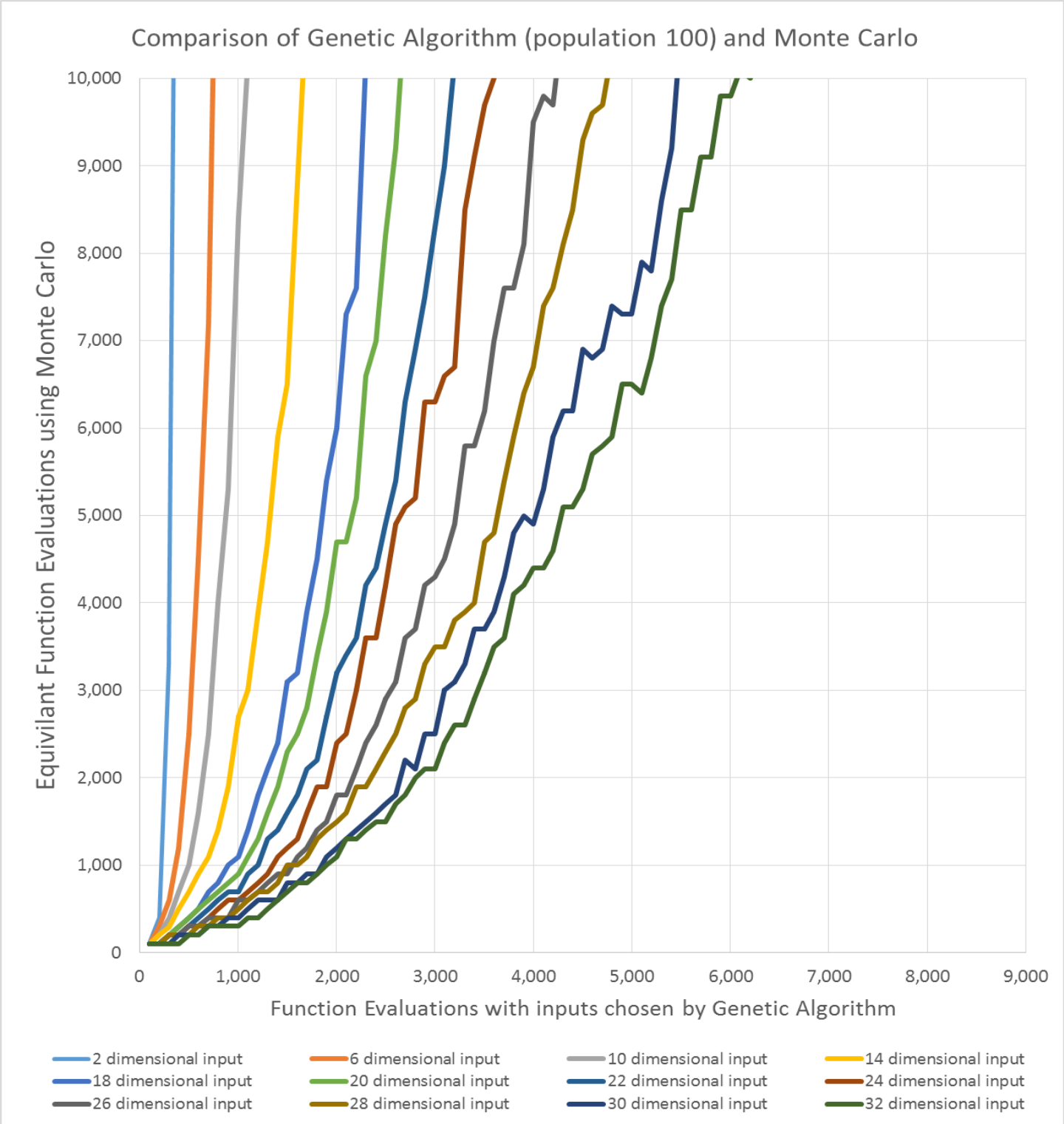


Figure 4: This graph scales the axes at a scale near to 1:1. As you can see, the genetic algorithm with a population of 100 initially performs worse than Monte Carlo Optimization for more than 18 dimensions. This demonstrates the downsides of using a smaller population size, and doesn't recover till 4000 function evaluations have occurred (40 generations with a population of 1000).

Takeaway: Large population sizes can be better than small population sizes for large numbers of inputs and when the number of generations that can be computed is limited.

Conclusion and Discussion

In order to be useful, the program we built in order to simulate plasmas will need to be more accurate. More specifically, it would need to account for magnetic fields in the plasma. And for results to be meaningful, the accuracy of the simulation needs to be verified. Taking steps to improve the accuracy of the simulation, and then verify that accuracy, will be a central focus going into the future. After programming and interacting with the simulation, we believe that it would be possible to account for magnetic fields simply and quickly by altering the Barnes-Hut algorithm to record the current within each cell/node, much as it does with charge.

Given a simulation that runs on 500 cores and takes 300 hours to produce an estimation of the efficiency, as many professional simulations would – we hoped to make our simulation over 10,000 times faster than this – on one of the most powerful current supercomputers in the world (one that has over 1,000,000 cores), our genetic algorithm when running with a population of 2000 would be able to optimize fusion reactors determined on 20 inputs within a year.

Acknowledgements

Thanks to Dr. Derzon, from Sandia, for providing consulting in relation to plasmas and the simulation of fusion reactors, and to Dr. Metzler, for providing consulting in relation to simulations and mathematics.

Bibliography

Bishop, Breanna. "Laser fusion experiment yields record energy at LLNL's National Ignition Facility." *Lawrence Livermore National Laboratory*. Lawrence Livermore National Security, 26 Aug. 2013. Web. 27 Mar. 2015. <https://www.llnl.gov/news/laser-fusion-experiment-yields-record-energy-llnls-national-ignition-facility>.

"Compact Fusion." *Lockheed Martin*. Lockheed Martin Corporation, Web. 27 Mar. 2015. <http://www.lockheedmartin.com/us/products/compact-fusion.html>.

"Facts & Figures." *Iter the Way to New Energy*. Iter Organization, Web. 27 Mar. 2015. <http://www.iter.org/factsfigures>.

Lakoba, T. *Simple Euler Method and Its Modifications*. Print.

Ongena, J., and G. Van Oost. *Energy for Future Centuries Will Fusion Be an Inexhaustible, Safe and Clean Energy Source?* Print.

"The Phases of ITER." *Iter the Way to New Energy*. Iter Organization, Web. 27 Mar. 2015. <http://www.iter.org/proj/iterandbeyond>.

Ventimiglia, Tom, and Kevin Wayne. "The Barnes-Hut Algorithm." *a graph visualization library using web workers and jQuery*. Samizdat Drafting, Web. 27 Mar. 2015. <http://arborjs.org/docs/Barnes-Hut>.

Young, Peter. *The Leapfrog Method and Other "symplectic" Algorithms for Integrating Newton's Laws of Motion*. 2014. Print.