Energy minimization of alkane molecules.

David Himmelstrup, vrs552
University of Copenhagen

Introduction

This report will describe the implementation and execution of a genetic algorithm written in Python for optimizing the energy configuration of three alkane modules ($C_{10}H_{22}$, $C_{20}H_{42}$ and $C_{40}H_{82}$). To make the problem tractable, the algorithm will only optimize the dihedral angles of the molecules. Since the number of dihedral angles linearly increases with the length of the carbon chains, a brute force approach of enumerating all possible configurations quickly becomes infeasible. A genetic algorithm, on the other hand, can search the configuration space and possibly find good (but not guaranteed to be perfect) solutions in a reasonable amount of time. The effectiveness of the algorithm is affected by several parameters which will be explored empirically. The effectiveness of the algorithm is also affected by offspring is produced, and, at the end of the report, a new and improved mating algorithm is introduced.

Genetic Algorithm

The algorithm is based on genetic inheritance. The idea is, if we have two energy configurations, maybe they can be combined in some way that gets the best parts of each parent. By repeatedly combining (mating) configurations over many generations and only letting the fittest (ie. those with the lowest energy) survive, the algorithm can incrementally hone in on good configurations without exploring the entire configuration space.

The algorithm follows three steps:

1. Mating.
   Two configurations of length N can be mated and produce two children. The first child has M dihedral angles from the first parent and M-N dihedral angles from the second. The second child has M-N angles from the first parent and N from the second.
2. Mutation.
   We also want to explore the space immediately surrounding our configuration. This is done by taking a random angle and mutating it to a random value. Too much mutation would be no better than a random search so this is done fairly rarely.
3. Evolution.
   Only energy configurations with the lowest energy survives to be mated in the next generation. However, to avoid stagnation, the algorithm allows for a small change of a high energy configuration replacing a lower energy configuration. The closer the energy

configurations are to each other, the greater the likelihood of replacement. The replacement probability is as follows:

$$P = e^{\frac{-\Delta E}{T}}$$

Greedy Optimization and butane

A simpler approach than the genetic algorithm would be to make a random walk through the configuration space and pick the configuration with the lowest energy.

Butane, with four carbon atoms, has only a single dihedral plane and the configuration space is therefore 1 dimensional and can be plotted easily. Figure 1 shows the energy (in kcal/mol) for dihedral angles ranging from 0 to 180 degrees in butane. The lowest point is at 180 degrees.

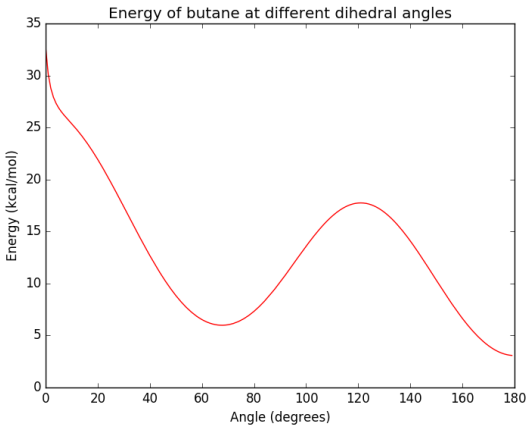

Figure 1

Sampling 2000 random points in this space yielded a lowest energy of 3.3 kcal/mol at angle 179.2. In the 1 dimensional configuration space of butane, this is quite close to optimal.

$C_{10}H_{22}$ has 7 dihedral planes and the configuration space therefore has n^7 points compared to the n^1 points for butane. Doing a random search of this space is not expected to yield results that are anywhere near optimal. For $C_{20}H_{42}$ and $C_{40}H_{82}$, the configuration space is even more vast. The follow table contains energy levels found by random sampling over the three molecules.

Name	Energy (kcal/mol)
$C_{10}H_{22}$	56.9
$C_{20}H_{42}$	231.0
$C_{40}H_{82}$	773.1

Table 1

Simulation parameters and defaults

The parameters of the genetic algorithm are the number of configurations (parents) in each generation, the total number of generations to simulate, the rate of random mutations, and the temperature of the system (which determines how likely a child is to spontaneously outcompete its parent; higher temperatures means children are more likely to survive). How these parameters affect the outcome of the algorithm is explored in the following simulations. By default, the simulations will run over 100 generations, with 20 configurations in each generation, a mutation rate of 10% and a temperature factor of 1.

Simulation 1

In simulation 1, the energy of $C_{10}H_{22}$, $C_{20}H_{42}$ and $C_{40}H_{82}$ using the genetic algorithm and the default parameters. The following three plots show both the mean energy across the parents in each generation as well as the lowest energy.

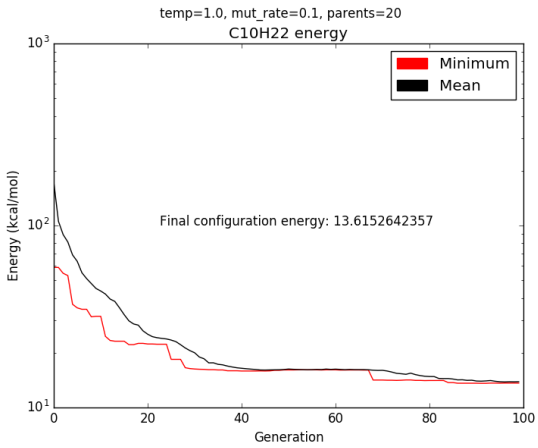

Figure 2

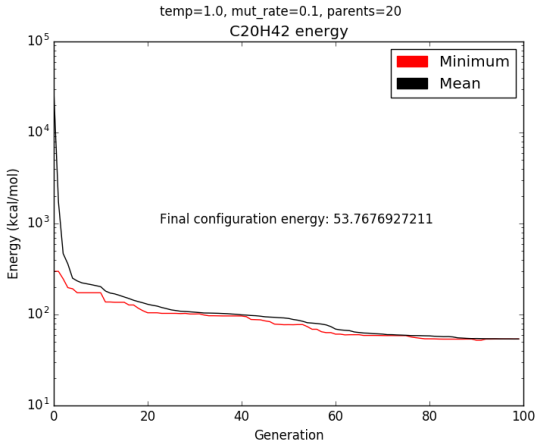

Figure 3

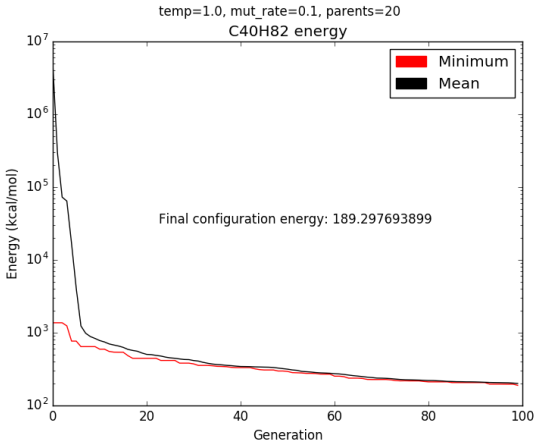

Figure 4

In all three plots, mean energy level starts high. As the parents mate and only the fittest survive, both the mean energy level and the lowest energy drops and eventually converges. A cursory comparison of the date from Table 1, Figure 2, Figure 3 and Figure 4 shows that the genetic algorithm is capable of finding lower energy configurations even though the greedy algorithm goes through 20 times as many generations.

Simulation 2

The default parameters might seem quite arbitrary so, to get a better feel for how they affect the outcome, let's investigate any correlation between the temperature parameter and the energy configuration found. Temperatures from 0.5 to 10 are checked in increments of 0.5.

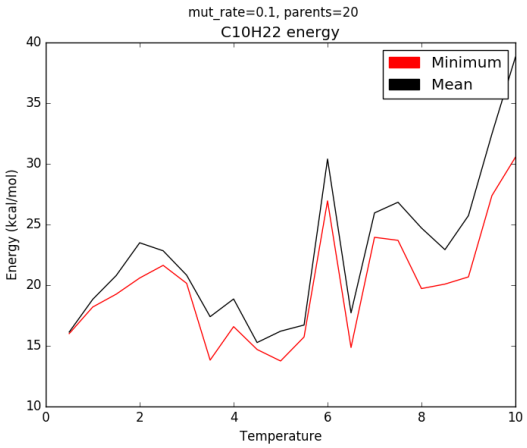

Figure 5

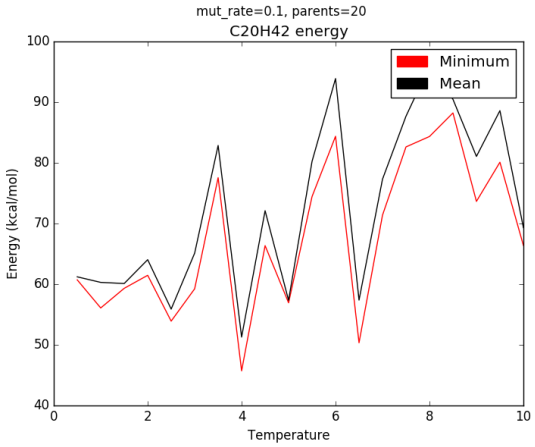

Figure 6

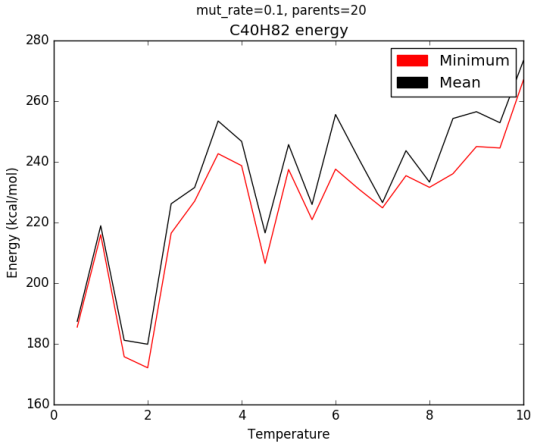

Figure 7

Figure 5, Figure 6 and Figure 7 show no clear correlation between temperature and energy found. The genetic algorithm is inherently noisy and this noise is shown as spikes in the plots with no clear trend lines.

Simulation 3

This simulation looks at correlations between mutation rate and energy levels found. Mutation rates from 0% to 50% are checked with 5% increments.

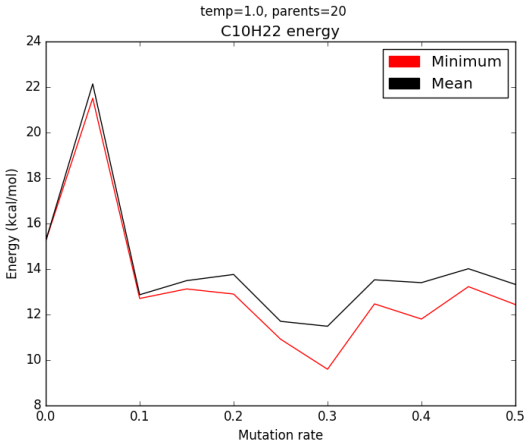

Figure 8

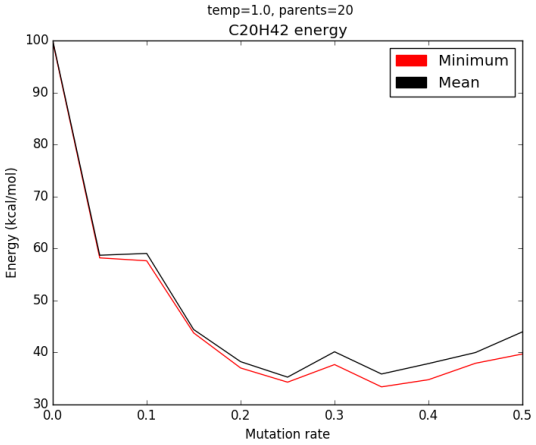

Figure 9

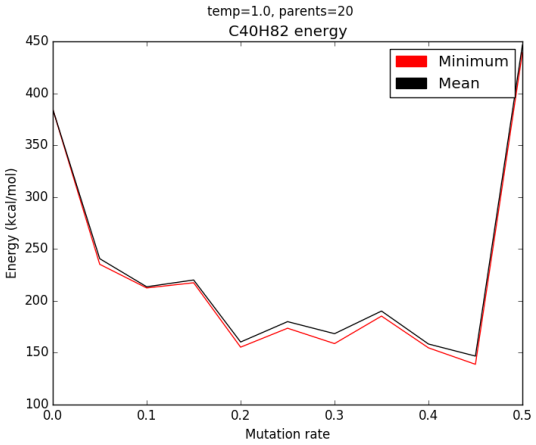

Figure 10

It appears that a too low mutation rate (<10%) makes it more difficult for the algorithm to find good solutions. Possibly because it only has the initial genetic information to work with and because it gets stuck in unfavorable local minimums.

Simulation 4

Comparing the greedy and genetic algorithms highlights the different way they operate. In the three following figures, it can be seen how the greedy algorithm makes big improvement leaps when it finds a new favorable configuration. The genetic algorithm instead makes many small, incremental improvements, honing in on configurations far more favorable than those found by the greedy algorithm.

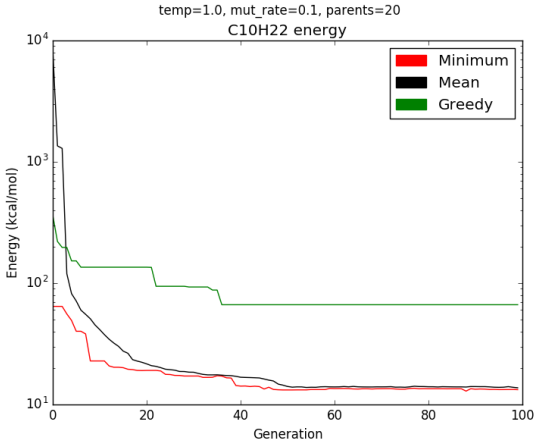

Figure 11

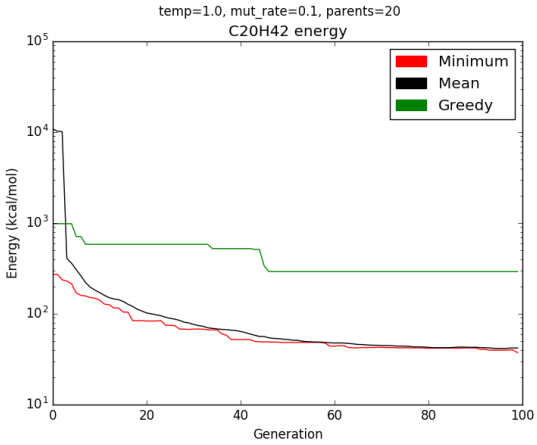

Figure 12

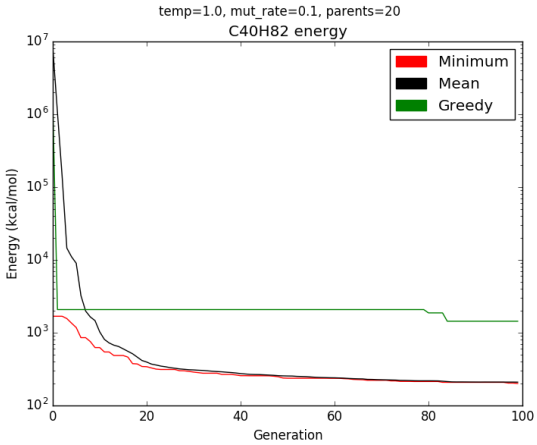

Figure 13

Simulation 5

This simulation looks for a correlation between the size of a generation and how many generations it takes for the energy level to converge. Population sizes are simulated in 10 steps from 4 to 100.

From the following three figures, it can be seen that there's a general trend of larger population sizes (up to a certain limit) requiring more generations to converge. In Figure 14, this limit is reached at a population size of about 35. In Figure 15, the limit is reached at a population size of about 65. In Figure 16, the limit has not been reached.

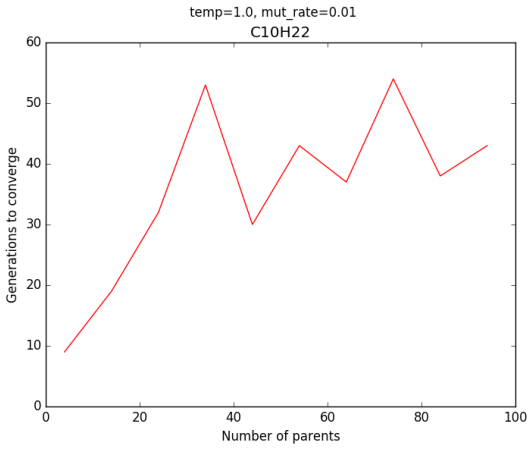

Figure 14

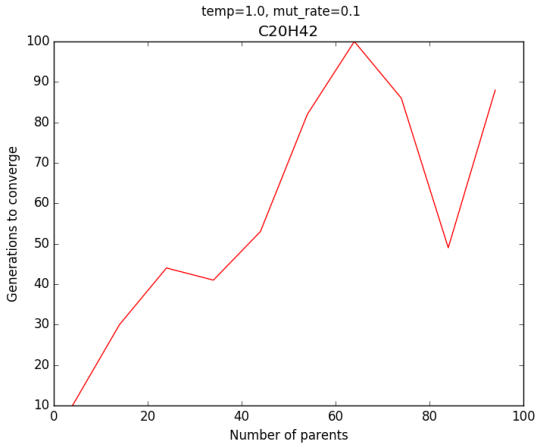

Figure 15

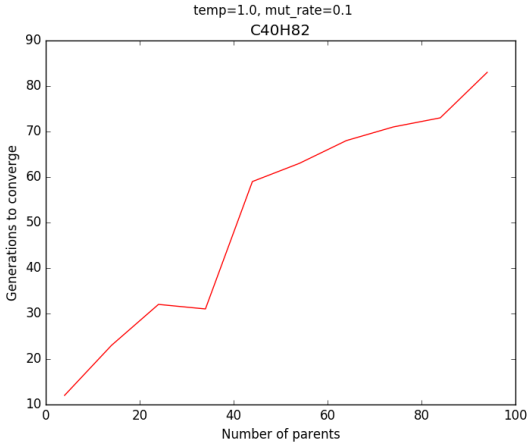

Figure 16

Simulation 6

This final simulation changes the mating algorithm. Instead of slicing both parents at the same location and letting each child inherit two chunks of genetic information, the new mating algorithm splits each piece of genetic information from the parents between the two children. This leads to many more undesirable children but also increases the chance of getting only the good information from each parent.

Compared to simulation 1, the energy configuration found for $C_{10}H_{22}$ is 14.0% lower, for $C_{20}H_{42}$ it is 22.3% lower, and for $C_{40}H_{82}$ it is 19.5% lower.

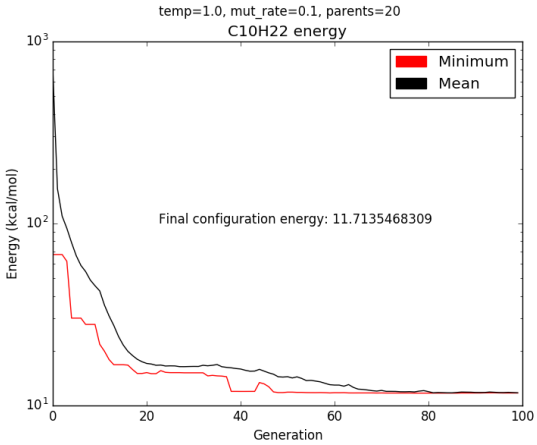

Figure 17: Improved mating

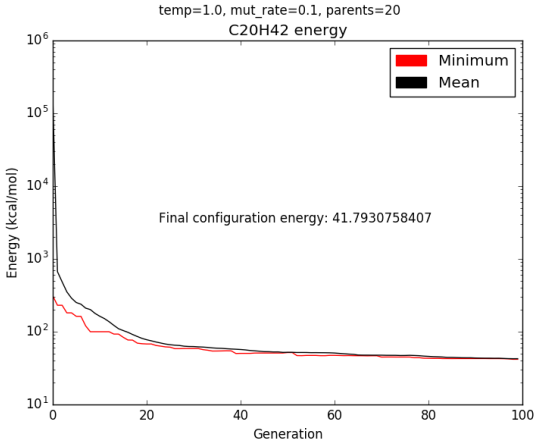

Figure 18: Improved mating

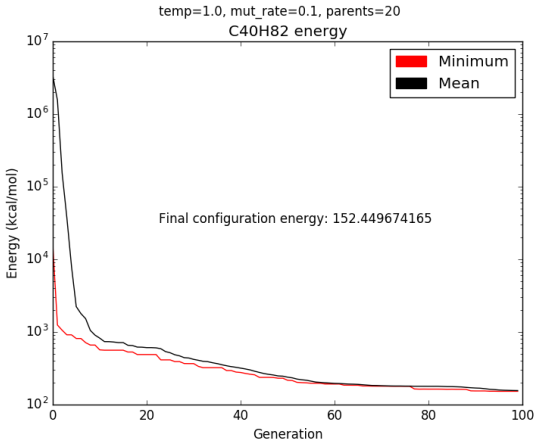

Figure 19: Improved mating