Advanced Algorithms Assignment 2

Zaymon Foulds-Cook

August 25, 2018

Contents

1 Introduction

1.1 Problem Statement

Linear polymers or chain molecules are molecules that exist in the natural world that are linear sequences of bonded atoms where each atom k within the molecule (excluding terminal atoms) is only bonded to atoms k_{k-1} and k_{k-1}. According to Valence-Shell Electron-Pair Repulsion Theory all atoms in a molecule interact with all other atoms regardless of whether the atoms are explicitly bound together through chemical bonds.

The energy of the interaction between two atoms is a function of the displacement between then and is given by the Lennard-Jones potential:

$$V = (\frac{1}{r^{12}} - \frac{2}{r^6}) \tag{1}$$

where

$$r_{ij}^2 = x_{ij}^2 + y_{ij}^2 \tag{2}$$

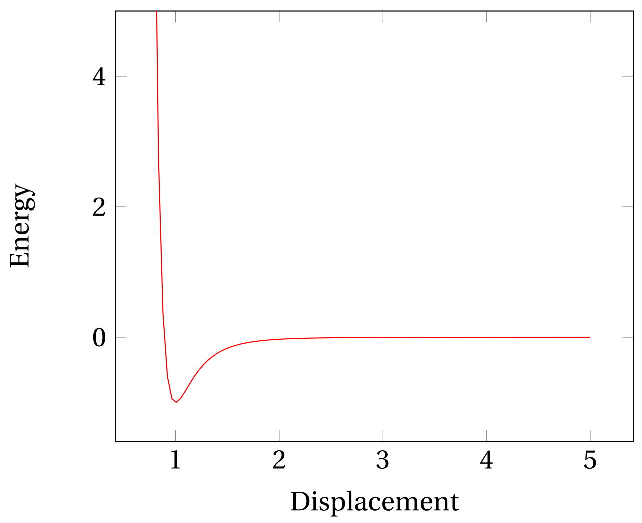

Figure 1: Energy as a function of displacement

The Lennard-Jones or "L-J potential" is simple mathematical approximation of the strength of interaction between a pair of neutrally charges atoms. As the distance between two neutrally charged similar atoms converges to 0 atoms experience Exchange Interaction or Pauli repulsion due to overlapping electron orbitals. The L-J Potential is an effective approximation commonly used as it reliably approximates energies at short and long distances. The L-J Potential is also used due to the computational simplicity since r_{ij}^{12} can be expressed as the square of r_{ij}^6.

For a molecule configuration to be stable it must be in a state of minimum energy. If the molecule is not in a configuration that results in minimum energy the configuration of that molecule would be transient in the physical sense as atom pairs throughout the molecule will be repelled and attracted. To calculate the total energy in a molecule configuration the summation of each atom pair's energy contribution is calculated. This is expressed by:

$$V = \sum_{i<j}^{N} \left(\frac{1}{r^{12}} - \frac{2}{r^{6}}\right) \tag{3}$$

Designing computational methods for estimating and predicting molecule configurations has relevance to the study of: protein folding, molecular medicine, molecular physics, pharmacology and other fields examining or designing molecules and compounds.

1.2 Problem Representation

The problem can be represented by a vector of angles $\alpha_0 ... \alpha_{N-2}$ where each angle is relative to the previous angle and the angle's value ranges from $-\pi$ to π radians. This system is visualized in Figure 2. Each bond in the molecule is of length 1, ensuring the minimum energy between bonded molecules as demonstrated by the function plotted in Figure 1.

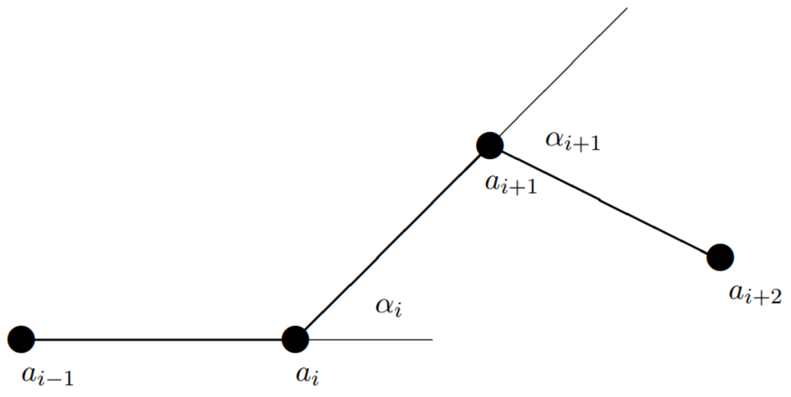

Figure 2: Problem configuration

There are several distinct advantages to this approach compared to using cartesian coordinates:

1. Specifying the angles removes the need to recalculate the positions of atoms after a change (as the angles are relative).

2. The cartesian coordinates can be obtained for any atom via trigonometry.

3. The vector of angles is a simpler representation of the problem and the change in energy for the cluster relative to the change in any α_i can be determined.

2 Literature Review

2.1 BFGS

The Broyden-Fletcher-Goldfarb-Shanno algorithm is an iterative method under the quasi-Newton class of hill climbing algorithms. "Quasi-Newton methods, like steepest descent, require only the gradient of the objective function to be supplied at each iteration" (Jorge Nocedal, 1999) The BFGS algorithm will never make a step in the ‘uphill’ direction. To effectively make use of BFGS in problem spaces with local minima BFGS needs to be used as the optimization step in a higher level optimization algorithm such as a genetic population based search.

2.2 Cross-Entropy Method

The Cross-Entropy method is a generic approach to solving complicated optimization problems such as Max Cut or the travelling salesman problem. "the CE method... defines a precise mathematical frame-work for deriving fast, and in some sense "optimal" updating/learning rules..." (Boer, Kroese, Mannor, & Rubinstein, 2005). The core algorithm of the CE method is quite a simple iterative process:

1. Generate a sample of random data according to some mechanism.
2. Score the sample and take an elite sub sample of the resulting population.
3. Use the values of the elite sub sample to update the parameters of the random mechanism to produce a "better" sample on the next iteration.

The random mechanism can be as simple as specifying a random distribution for each element in the vector to be optimized (in this case the vector of angles) where the mean and standard deviation are changed to reflect the elite sample.

Specifying a smoothing parameter can be useful (Botev, Kroese, Rubinstein, & L‘Ecuyer, n.d.). The smoothing parameter determines the ratio of change between the old distribution and the newly determined updated distribution. This smoothing parameter λ can either be a static value $0 \leq \lambda \leq 1$ or dynamically changed as a function of time, score or another heuristic.

3 Algorithm Description

3.1 Cross Entropy Optimization

3.1.1 Pseudocode

```
ALGORITHM CrossEntropy (sequenceLength):
    populationSize <– ALGORITHMPARAMETER
    parameterBlendRatio <– ALGORITHMPARAMETER
    eliteDistributionPercentage <– ALGORITHMPARAMETER
    epsilon <– ALGORITHMPARAMETER

    distributions <– [vector<normal distribution>] with default parameters
    population <– [vector<Sequence>] generatePopulation(populationSize, distributions)

    lastBest <– infinity
    bestScore <– infinity
    bestSequence <– Sequence

    while True:
        # Sort Population by Cost
        scoredPopulation <– [vector<pair<cost, Sequence>>] sort(population)
        diff <– lastBest – scorePopulation[0].cost
        lastBest <– scoredPopulation[0].cost

        if lastBest < bestScore:
            bestScore = lastBest

        # Update Distributions
        for angle in 0 <= angle < sequenceLength:
            values <– []
            for memberIndex in 0 <= memberIndex < populationSize * eliteDistributionPercentage:
                values <+ scoredPopulation[memberIndex].Sequence[angle]

            mean <– mean(values)
            stdDeviation <– stdDeviation(values)

            distributions[angle].updateParameters(mean, stdDeviation)

        # Check exit conditions
        if (diff < epsilon):
            print bestScore
            break

        # Generate new Population
        population = generatePopulation(populationSize, distributions)
```

3.2 BFGS and Genetic Search

3.2.1 Pseudocode

```
FUNCTION BFGS(sequence):
    gradientVector = sequence.CalcGradients


ALGORITHM GA_BFGS(sequenceLength):
    populationSize <− ALGORITHMPARAMETER
    epsilon <− ALGORITHMPARAMETER

    # Mutation chances
    mutation1 <− ALGORITHMPARAMETER
    mutation2 <− ALGORITHMPARAMETER
    mutation3 <− ALGORITHMPARAMETER

    population <− [vector<Sequence>] generatePopulation(sequenceLength)

    lastBest <− infinity
    bestScore <− infinity
    bestSequence <− Sequence

    while true:
        for member in population:
            member <− BFGS(member)

        scoredPopulation <− [vector<pair<cost, Sequence>>] sort(population)
        diff <− lastBest − scorePopulation[0].cost
        lastBest <− scoredPopulation[0].cost

        if lastBest <− bestScore:
            bestScore <− lastBest

        for member in population:
            if rand > mutation1:
                member.mutateRandomAngles()
                BFGS(member)
            if rand > mutation2:
                member.clearSubsequent()

        for randomPair in population:
            newPairs = crossOver(randomPair)
            randomPair.first = newPairs[0]
            randomPair.last = newPair[1]

        # Check exit conditions
        if (diff < epsilon):
            print bestScore
            break
```

3.3 Cross Entropy Method to Generate Optimization Candidates for BFGS

An experiment to use the Cross Entropy Method as the global optimizer in order to generate optimization candidates for BFGS. Conceptually this method will allow the Cross Entropy Method to optimize the distribution vector to generate good candidates for BFGS optimization.

3.3.1 High level algorithm

1. Generate Population using distribution vector.
2. Run BFGS on population members, generating an additional optimized population.
3. Calculate the scores of the optimized population and sort.
4. Update the distribution vector based on the pre-optimized sequences which correspond to the sequences in the elite sample of the optimized population.
5. Goto step 1.

References

Boer, P.-T. B., Kroese, D. P., Mannor, S., & Rubinstein, R. Y. (2005). A tutorial on the cross-entropy method. *Annals of Operations Research, 134*, 19–67. Retrieved from `https://people.smp.uq.edu.au/DirkKroese/ps/aortut.pdf`

Botev, Z. I., Kroese, D. P., Rubinstein, R. Y., & L'Ecuyer, P. (n.d.). *The cross-entropy method for optimization.* Retrieved from `https://people.smp.uq.edu.au/DirkKroese/ps/CEopt.pdf`

Jorge Nocedal, S. J. W. (1999). *Numerical optimization.* New York: Springer-Verlag. Retrieved from `http://www.bioinfo.org.cn/ wangchao/maa/Numerical_Optimization.pdf`