Genetic Algorithm to search for exotically shaped fullerenes

L. R. N. Ferreira\textsuperscript{1}, J. P. A. Mendonça\textsuperscript{2}, F. Sato\textsuperscript{2}

\textsuperscript{1} Departamento de Ciência da Computação, Instituto de Ciências Exatas, Campus Universitário da UFJF, Juiz de Fora - MG, CEP: 36036-900, Brazil
\textsuperscript{2} Departamento de Física, Instituto de Ciências Exatas, Campus Universitário da UFJF, Juiz de Fora - MG, CEP: 36036-900, Brazil

E-mail: jpalastus@gmail.com, lohanext@gmail.com, fsato01@gmail.com

Abstract. The versatility of carbon-carbon bonds is in charge of various carbon-based structures including numerous possibilities for building fullerenes. Theoretically, it is possible to make any closed surface consisting of C atoms in a number of ways. However, the generation of possible arrangements and, furthermore, calculating the corresponding energetics is a great challenge even for a small molecule. In this context, we develop a genetic-algorithm-based code that can search for exotically shaped fullerenes. Furthermore, we discuss the construction and optimization of the algorithm assisted by some test results.

1. Introduction

Fullerene is the generic terminology used to describe a C-based molecule in a closed or partially closed arrangement. It is well-known that C atoms can adopt various shapes and arrangements thus providing a humongous variety of different patterns of fullerenes itself. Other than that, C atoms can be arranged to form various structures of different dimensions, such as graphene (planar) \cite{1}, nanotubes (tubular) \cite{2}, nanotorus (tubular rings) \cite{3}, buckyballs (spherical/ellipsoidal) \cite{4}, to name a few.

Naively speaking, a fullerene can be built in the shape of any closed surface in the form of graphene-like arrangements of carbon. However, from a computational aspect, several other (secondary) problems arise in the investigation of such processes. For each closed shape, there are infinite and nontrivial C arrangements even for a molecule consisting of a limited number of atoms. Consequently, the search for stable and energetically favorable fullerenes becomes challenging.

The problem of finding, in a 2D continuous surface, a set of \(N\) coordinates that minimize a given function (e.g., the cohesive energy) is, in essence, the NP problem of optimizing a function of \(N^2\) continuous variables \cite{5}. Along with the calculation of cohesive energy, it is a complex problem from computational point of view.

Bio-inspired algorithms \cite{6}, such as genetic algorithms (GA) or swarm optimizations\cite{7}, have been used as a good alternative for NP optimization problems. In addition, it has already been shown that this method is quite reasonable in predicting the experimental/theoretical geometries of organic compounds and complexes. In particular, these methods have already shown good results in finding geometries for organic compounds and complexes from experimental data and simulation comparison \cite{8, 9, 10}.
To handle the problem of building carbon molecules of more general shapes, we develop a software based on a genetic algorithm that can search for fullerenes that can be wrapped around in any exotically shaped surface. In this work, we highlight the main features of the code and show the archived results for trial geometries. We also discuss on the development issues of this kind of software and on the optimization/parallelization process that made these calculations doable.

2. Methodology and Implementation

The main purpose of this work is to build a versatile code that can handle virtually any kind of structure. To do so, our input files should contain the maximum possible information about the C structure to be reproduced. There are many traditional file formats available for 3D geometry modeling, but we choose to work with Polygon file format (.ply). Having said that, vertex positions in this format are expressed as traditional Cartesian coordinates which makes it a more suitable format for reading and handling.

From the initials set of vertex containing the shape in the .ply file, we build a list of carbon structures to populate the first generation of the genetic algorithm. This was done by selecting a random subset of the original vertex set with the desired number of elements to be used as atomic positions. For the physical validity of our energy evaluations, the atoms should be sufficiently far from each other by obeying $r_{ij} < 2r_0$ for all $i$ and $j$ vertex in the selected subset ($r_0$ represents the covalent atomic radius of C).

Once the set of initial structures is generated, their energy is calculated using a combination of Tersoff Potential and Lennard-Jones Potential parametrized for $sp^2$ carbon [11]. Through the genetic algorithm, a given percentage (input parameter) of the lower energy structures is selected to “survive and reproduce”, producing a population for the next generation.

In the input .ply file, the vertex is read in order. We use this order to define the genetic representation of each element of the population as a vector that contains the indices of the vertices that have been selected to be a carbon atom in this structure. In this sense, a structure is genetically represented by an array of integers that can be converted into positions using the data from the .ply file. Crossover has been made by sorting an index where we split the parents’ genetic representation and then combining the parts from both. Mutations can occur in the descendant structures with a given probability. A schematic representation of this genetic algorithm can be seen in figure 1.

![Figure 1](image_url)

The reproduction process is made multiple times until a previously defined number of generations is archived. Since time is an important aspect when talking about the optimization
of algorithms, the final code has been implemented in C++ due to its good performance in memory management and parallelization. A pseudo-code for this implementation can be seen in figure 2.

```c++
// n_structs // number of structures
struct struct_size // 'size' of the structures
steps // total number of generations
structs[n_structs] // structures vector
new_structs[n_structs] // next generation structures vector

invalid_verif < MAX_INT

energy[n_structs] < 0
probs[n_structs] < 0

generation < 0

for i < 0 to i < n_structs
    structs[i] <- initialStructure(energy)
    // To invalid structures energy[i] <- invalid_verif + 1
end for

while generation < steps do
    highest_energy = MAX_INT;
    lowest_energy = MAX_INT;
    for i < 0 to i < n_structs
        if energy[i] < invalid_verif then
            energy[i] <- evaluateEnergy(structs[i])
            if (highest_energy<energy[i]) highest_energy <- energy[i]
            if (lowest_energy>energy[i]) lowest_energy <- energy[i]
        end if
    end for
    probabilities(probs,energy,highest_energy,lowest_energy)
    for i < 0 to i < n_structs
        parent1 <- rand_parent(probs, NULL);
        parent2 <- rand_parent(probs, parent1);
        new_structs[i] <- generate_son(parent1,parent2);
    end for
    structs <- new_structs
    generation++
end while

// Evaluate last generation energies
for i < 0 to i < n_structs
    if energy[i] < invalid_verif then
        energy[i] <- evaluateEnergy(structs[i])
    end if
end for
```

Figure 2. Pseudo-code showcasing the implementation strategy used here.

The parallelism was done by focusing on the energy calculation part during the stages of the generations. Two different types of parallelism were tested: One with the implementation of using the Open Multi-Processing (OpenMP) threads library and other using the Message Passing Interface (MPI) library. For both implementations, the parallelism was applied inside the `evaluateEnergy()` function, shown in the pseudocode, and both can be defined as Data Parallelism since we focused on distributing the number of structures evenly between all processes (or all threads) and then each process/thread computes the full procedure for each of its structures.

3. Results
This software has been able to generate physically reasonable structures for all (or tested) geometries using .ply files. In tests using the Stanford Rabbit (scaled to support 150 atoms,
a population of 1000 molecules and a mutation rate of 0.5%), an average of 10000 generations were done to obtain stable values of fitness. An example of the optimization-curve can be seen in figure 3, while more examples of obtained geometries can be seen in 4.

In a supplementary effort to verify the obtained geometries, we perform PM3 (semi-empirical) optimization using ORCA [12]. In addition, it is shown that the geometries converged in our proposed methodology don’t suffer major deformations (only bond breaking or dihedral angle inversions have been verified) after the implementation of quantum chemical calculations. This is, in fact, combined merit of the effectiveness of or methodology to minimize functions and the accuracy of Tersoff to describe graphene-like systems. Since this analysis demands a full dedicated implementation and more complex numerical treatments, we will not further discuss it here.

Figure 3. To illustrate the data obtained from a run of this code, we show here the energy evolution and final structure result for a fullerene grown over a Stanford Rabbit.

We also notice a strong dependency on the quality of the final geometry on the ratio between the number of atoms and the area of the surface to be reproduced. A good approach identified in the tests is to use the area of the input geometry (that can be approximately calculated using the polygons defined after the vertices at the end of the .ply file) and the graphene superficial density. For a smaller number of atoms than this, final structures can show holes. Besides, for a larger number of atoms, three and four atoms ring can be generated, which are not really stable
Figure 4. Some visual examples of structures obtained from the methodology presented here. The first two are the fruit of closed surface models and the last one have been obtained from a planar square, showing the effects of restricting the structure to the surface shape in the optimizations.

in free optimization and are appearing as minima here as an artifice of the 2D confinement. In fact, this confinement constraint actuates here as a constraint actuates in classical mechanics and can generate “artificial” equilibrium configuration (that don’t exist without it).

Also, our studied subject which is the Stanford Rabbit has shown problematic parts for optimization, especially the ears. The algorithm tries to generate atoms for the sides of ears, however, ends up with a single-side final result as the these sides are very near to each other. Consequently, the methodology should be revised for applications that need a highly detailed geometry. An alternative to avoid this problem is the use of a scaled Stanford Rabbit model, where the thickness of the ear is larger than the van der Waals equilibrium distance among the C atoms.

Regarding the parallelism implementations, we compared the speed to iterate over a specific number of generations (an ellipsoid geometry has been built using 50000 points in the .ply file. 1000 generations have been processed). For this test, the size of each structure (number of atoms) is kept fixed while the number of structures per generation and number of threads/cores is increased on each run. For each run, we recorded the time needed to complete all the iterations for each implementation and the results can be seen in tables 1 and 2.

In general, MPI implementation proved to be slower, which is probably because of the extra memory needed for it to work. Unlike OpenMP (where all the threads have access to the same set of variables), on MPI, a lot of information was copied for each running process plus time spent sending and receiving messages between processes to share the progress.

Both tables show that the program is not scalable. this means that the efficiency (the ratio between speed gain and number of cores/threads used) only decreased when using more resources and did not change even when increasing the number of structures per generation. It is important to point out that only a part of the program was parallelized leaving other steps of the process still sequential, such as the reproduction phase. So, the code has space for improvements that may change the scalability factor.

4. Conclusion
We proposed the utilization of genetic algorithms to build fullerenes in generic shapes. In this regard, some considerations and approach choices have been made in order to implement this methodology. Molecules have been represented as integer arrays (genome) with positions labels, referring to an input .ply file vertices. Crossover and mutation have been implemented as schematized in figure 1.

Our tests pointed out that this methodology is promising since it is able to predict results
Table 1. Average running time (s) for MPI implementation

| cores | n_structs | 400 | 800 | 1600 |
|-------|-----------|-----|-----|------|
| 1     |           | 160 | 331 | 641  |
| 2     |           | 96  | 199 | 380  |
| 4     |           | 57  | 116 | 220  |
| 8     |           | 37  | 75  | 145  |
| 16    |           | 27  | 55  | 110  |
| 32    |           | 23  | 49  | 91   |

Table 2. Average running time (s) for OpenMP implementation

| threads | n_structs | 400 | 800 | 1600 |
|---------|-----------|-----|-----|------|
| 1       |           | 160 | 331 | 641  |
| 2       |           | 89  | 185 | 354  |
| 4       |           | 52  | 108 | 207  |
| 8       |           | 33  | 71  | 135  |
| 16      |           | 23  | 49  | 98   |
| 32      |           | 19  | 43  | 78   |

in affordable computational time using classical methodologies and the obtained geometries were found to be stable after semi-empirical calculation. In addition, it is also shown that the methodology could further be improved by making some adjustments in order to capture the small details and complex geometries. That being said, a general recipe of choice of parameters is difficult (maybe impossible) to point out.

The code has been parallelized and, even with a small speedup in comparison to the ideal, is capable to generate guess geometries in a time much smaller than a single DFT geometry-optimization for this size/kind of system. In this regard, we see this methodology as a good alternative to generate prototypes and initial guesses for carbon materials in various problems, such as graphene coating of nanostructures or creating molecules with desired shapes for a given applications, saving computational time and the human efforts.

Acknowledgements

The author acknowledges support by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), the Financiadora de Estudos e Projetos (FINEP) and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for financial support, as well as UFJF for the infrastructure used.

References

[1] Ferrari A C, Meyer J, Scardaci V, Casiraghi C, Lazzeri M, Mauri F, Piscanec S, Jiang D, Novoselov K, Roth S et al. 2006 Physical review letters 97 187401
[2] Iijima S and Ichihashi T 1993 nature 363 603
[3] Liu L, Guo G, Jayanthi C and Wu S 2002 Physical review letters 88 217206
[4] Rohlfing E A, Cox D M and Kaldor A 1984 The Journal of chemical physics 81 3322–3330
[5] Ausiello G, Crescenzi P, Gambosi G, Kann V, Marchetti-Spaccamela A and Protasi M 2012 Complexity and approximation: Combinatorial optimization problems and their approximability properties (Springer Science & Business Media)
[6] Fister Jr I, Yang X S, Fister I, Brest J and Fister D 2013 arXiv preprint arXiv:1307.4186
[7] Banks A, Vincent J and Anyakoha C 2007 Natural Computing 6 467–484
[8] Deaven D M and Ho K M 1995 Physical review letters 75 288
[9] Li B, Li Y and Gong L 2014 Engineering Applications of Artificial Intelligence 27 70–79
[10] García-Godoy M J, López-Camacho E, García-Nieto J, Del Ser J, Nebro A J and Aldana-Montes J F 2019 Applied Soft Computing 79 30–45
[11] Rajasekaran G, Kumar R and Parashar A 2016 Materials Research Express 3 035011
[12] Neese F 2012 Wiley Interdisciplinary Reviews: Computational Molecular Science 2 73–78