Proceedings of The National Conference
On Undergraduate Research (NCUR) 2006
The University of North Carolina at Asherville
Asherville, North Carolina
April 6 – 8, 2006

Construction of Lists of Neighbors of Points by Means of Delaunay Triangulations

Francisco J. Izquierdo Serrano
Department of Mathematics
University of Puerto Rico at Humacao
100 Tejas Avenue
Humacao, Puerto Rico 00791

Faculty Advisor: José O. Sotero-Esteva

Abstract

Given a set of points in the R^3 space we want to construct efficiently, for each point of the set, the list of all other points that are within a prescribed distance. This problem is related with the computation of short range interactions in molecular dynamics simulations. These neighbors lists could be constructed by the inspection of all possible pairs of points. This naive algorithm takes time $O(n^2)$. On the other hand, we could assume that we have a structure already defined using those points that could help us construct such lists. The data structure tested here was the Delaunay Triangulation (DT). A DT is a triangulation that satisfies the empty circle property, that is, the circumference defined by the vertices of each triangle cannot contain another point of the triangulation. In practice, DTs connect each vertex with their nearest neighbors. Depending on which data structure is used to represent DT, the theorethical construction times vary from $O(n^2)$ for a flat structure, to $O(n \log n)$ for a hierarchical structure.

Once the DT is constructed, we construct the lists of neighbors of one point by doing a depth first search starting from that point until the prescribed distance is reached. In order to measure actual running times for data sets of the order of magnitude and distributions commonly found in molecular dynamics simulations, we compared empirically the three types of constructions mentioned above: naive, flat Delaunay, and hierarchical Delaunay. We used the CGAL package which supply easy access to geometric algorithms and DTs. We observed empirically that the hierarchical structure is better than the others for problem sizes ranging from 10^3 to 10^6 points. Different distributions of points do not seem to affect this performance.

Keywords: Molecular dynamics, Delaunay triangulations, Carbon nanotubes.

1. Introduction

Delaunay triangulations (DT) were developed by Boris Delaunay in the 1930s. Many applications to computer graphics were found during the 1990s. These triangulations (when used in 2D) or simplices (in 3D) maximize the minimum angle between arcs. When used in computer graphics, the result is smoother representation of surfaces or volumes than those produced by other triangulations.

Given the importance of the DTs, the studies of their properties and construction techniques under various assumptions have produced efficient data structures and algorithms for their management. Many commercial and public domain software packages and libraries that implement those data structures and algorithms can be found. One of such libraries is the *Computational Geometry Algorithms Library* (CGAL)[1].

A new application of the DT to molecular dynamics simulations (MD) has been proposed recently[2]. MD simulations computes interactions between atoms or molecules using, among others, a truncated version of the Lennard-Jones potential[3]. As a consequence, considering all the particles in the simulation in order to compute the total force exerted on one particle by the others based on this short range interaction is not longer necessary. Only those particles that are within certain range have to be considered. It is customary in MD simulations to construct,

for each particle, a list of all the neighbors that are within the prescribed range and let run the simulation for several iterations based on those lists. The technique produces a notable improvement in efficiency of the simulation.

As the study of nanostructures grows more important, the use of MD simulations for their study has become widespread. By definition, nanostructures are structures with at least one dimension measuring less than 100 nanometers. Nanofibers are important examples of these structures. They have important applications in micro and nano devices as fluid, heat and electric conductors and as electronic devices[4]. The study of a particular type of nanofibers, carbon nanotubes, are of particular interest for their well known structure and excellent electrical and heat conducting properties that are being studied by means of MD simulations[5].

This paper deals with the application of DT to MD. The study uses data from simulations involving carbon nanotubes in order to empirically measure the performance of different versions of DT-based algorithms as compared to the classical neighbors lists method.

2. Background

2.1. short range interactions in molecular dynamics

A molecular dynamics simulation (MD) is based on an atomistic model of a system. It consists of an iterative process in which equations of motion are numerically solved. Each iteration represents a time step of the simulation.

The Lennard-Jones potential is defined by

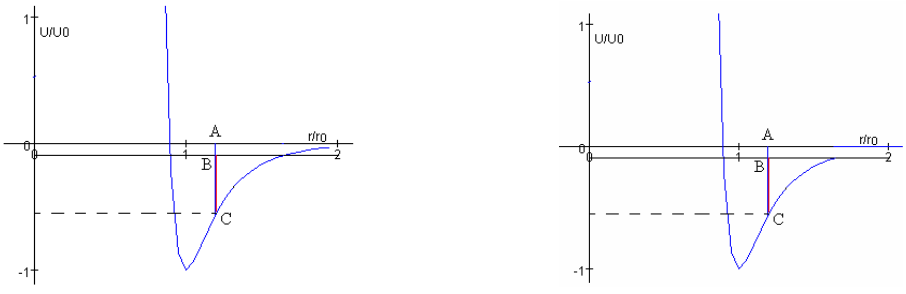
$$E_p(r) = E_0\left[\left(\frac{r_0}{r}\right)^{12} - 2\left(\frac{r_0}{r}\right)^{6}\right]$$

(figure 1a) where E_0 and r_0 are constants that depend on the atoms involved and r is the distance between the pair of atoms. It is used in MD simulations to compute Van der Vals interactions. A similar potential can be used to compute bond interactions. The calculation of the interaction between all the molecules requires the inspection all possible pairs of n atoms, that is,

$$\frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

cases. The complexity is $O(n^2)$.

Two improvements are often made to MD simulations in order to make them more efficient. First, since the potential is very small when the atoms are far apart those computations can be avoided. Therefore, a truncated potential is actually used (figure 1b). Along with this truncated potential the second improvement consists of maintaining a table that associates to each atom the neighboring atoms that are within a distance of r_{cut}. Once the table is built, the simulation runs for a while computing atom interactions by using only the neighbors of each atom according to the table.



(a) (b)

Figure 1: (a) Graph of the Lennard-Jones potential, (b) Truncated potential.

2.2. Delaunay triangulations

A *triangulation* is a subdivision of a geometric object into simplices. The *simplices* are polygons with triangular faces if the object has three dimensions or triangles if the object is of dimension 2. The *vertex set V* of the triangulation consists of the collection of the points of the triangulation that lie at the corner of the simplices. A *Delaunay triangulation* is a triangulation where each vertex is linked to their nearest neighbors in the sense that the circumferences defined by the vertices of a triangle in the Delaunay triangulation do not contain another vertex of the triangulation. Figure 3 shows a triangulation that is not Delaunay and one that is Delaunay.

Figure 3: Examples of triangulations in 2D. (a) not a Delaunay triangulation, (b) Delaunay triangulation.

Different algorithms for constructing Delaunay triangulations exist. A naive method would construct the triangulation by adding each vertex one at a time. The method would require finding an existing triangle where the new points lies, subdividing the triangles in three new triangles, and flipping edges in order to satisfy the empty circle property, similar to the step from figure 3a to 3b. Each new flipping may damage the Delaunay property for other triangles. In the worst case all triangles would have to be inspected, which is O(n). If n vertex are added, then the whole construction would be $O(n^2)$.

Hierarchical constructions of Delaunay triangulation exist. Those constructions maintain data structures that keep track of the triangle subdivisions described above. With this modification the search for the triangles in which a new point lies as well as the fixing of the empty circle property requires O(log n) operations. When n vertices are inserted, a O(n log n) algorithm results. In a previous work flat and hierarchical Delaunay triangulation construction were empirically compared[6].

3. Methods and software

In this work four different methods for constructing neighbors lists are compared: one based on the inspection of all possible atom pairs and three Delaunay-based algorithms. Implementations of those algorithms are executed and running times are measured. The data obtained from the test is fitted to different functions, errors are measured, and the correspondence to theoretical complexities is analyzed.

3.1. lists of neighbors

3.1.1. *classical neighbors lists*

Classical neighbors lists are constructed by inspection all possible atom pairs. An array of atoms is maintained. A data structure consisting of an array of lists of indexes of atoms in the previous array hold the neighbors lists.

3.1.2. *Delaunay-based neighbors lists*

The *Delaunay Triangulations* link each vertex with their nearest neighbors. Based on this property we define three types of neighbors lists: one-level, two-levels, and three-levels neighbors lists. A Delaunay triangulation is constructed for the three methods using the coordinates of the atoms in the system. The one-level lists include as neighbors of an atom the atoms that corresponds to the vertices in the triangulations that are neighbors of the base atom. Two-levels neighbors lists include the neighbors of these neighbors as well. Three-levels neighbors lists include the neighbors of the later. The data structure used to store the neighbors lists is the same as the one used in section 3.1.1. Figure 4 illustrates the three levels of neighbors of an atom. A depth-first search algorithm is used for this consturction is shown in figure 5.

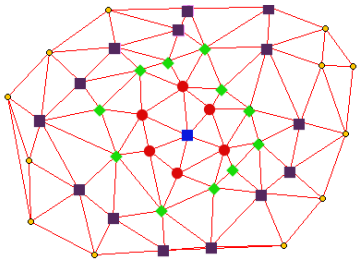


Figure 4: A Delaunay triangulation. A base atom (squared vertex) has one-level neighbors shown as circle vertices, two-level vertices shown as diamond vertices, and three level neighbors shown as dark square vertices.

```
void neighborsOf(const Triangulation &t, const Tds::Vertex_handle
&unVertice, const int level, Vertex_vector &point){
      if(level < 1) return;
      Vertex_circulator vc = t.incident_vertices(aVertex), done(vc);
      do{ if(!hasPoint(vc, point) && !t.is_infinite(vc))
                  point.push_back(vc);
      } while(++vc != done);
      if(!(vc = t.incident_vertices(aVertex))
          do{ if(!t.is_infinite(vc))
                 neighborsOf(t, vc, level-1, point);
          } while(++vc != done);
}
```

Figure 5. Code (in C++) used for Delaunay based neighbors lists construction.

3.2. input data

In order to test the implementation in a realistic setting the input data used for the simulation was a system consisting of a Carbon nanotube inmersed in Argon gas as shown in figure 6. The dimensions of the containing box are 80 x 80 x 80 $\mathring{A}$. The amount of Argon atoms was varied from 10^3 to 10^5 in steps of 2,000 and from 10^5 to 10^6 in steps of 10,000.

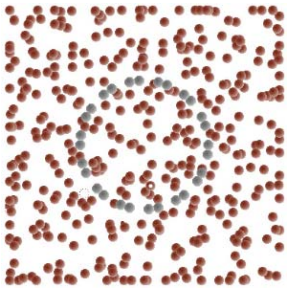


Figure 6. Radial slice of atomic system of the simulation. The inner ring represents a single walled (7,7) carbon nanotube inmersed in Argon gas.

3.3. comparison of execution times and theoretical bounds

The comparisson between theoretical time complexities and the times obtained in these tests was made by means of least-squeres fitting[7,8]. Each data set consisted of m pairs of the form (n_i, t_i) where n_i is the size of the problem and t_i the time spent by the simulation. The following functions were chosen:

$$T_{lin}(n) = a_1 + a_2 n,$$
$$T_{nlog}(n) = b_1 + b_2 n \log n,$$
$$T_{quad}(n) = c_1 + c_2 n^2,$$

and

$$T_{cub}(n)=d_1+d_2 n^3 .$$

A constant term has been added to each function in order to account for the part of the time spent by the algorithm independent of the amount of data, for example, for the initialization of global objects. The determination of the model that more closely represents the data was made based on the error function

$$E[T]=\sqrt{\frac{1}{m}\sum_{i=1}^{m}(T(n_i)-t_i)^2} .$$

3.4. implementation details

3.4.1. *CGAL package*

To construct the Delaunay triangulations the computational geometry C++ library *Computational Geometry Algorithms Library* (CGAL) package was used. Instances of the

`CGAL::Triangulation_hierarchy_3<Dt>`

class was used to construct the triangulation by adding point-vertices one by one. Once the triangulation was built, the algorithm shown in figure 4 constructed the neighbors lists.

3.4.2. *hardware, operating system and compiler*

A C++ implementation of the algorithms was tested on an Itanium2 processor running a Red Hat operating system. The Intel C++ compiler version 8.1 was used to compile both the CGAL library and our implementation of the neighbors lists. Execution times were measured as shown in figure 7.

```
for(long n=10000; n <= 10000000; n += incr){
  MDSimulationClock clock;    clock.start();
  // ... O(1) code
  MDNLDelaunaySimulation sim(ins, parameters);  // contructs NLists
  sim.run(1);  // runs one iteration of the smulation
  cout << "simDelaunay\t" << n << "\t" << clock.stop() << endl;
}
```

Figure 7. C++ code that generates the report of execution times.

3.5. *numerical analysis software*

```
function  [A a b e]=corr2Log(data)
      x=data(:,1)/1000;       y=data(:,2);      m = size(y)(1);
      %A=[ones(m,1) x];                   % uncoment for linear fit
      %A=[ones(m,1) prod([x log(x)]')'];  % uncoment for n log n fit
      %A=[ones(m,1) prod([x x]')'];       % uncoment for n^2 fit
      %A=[ones(m,1) prod([x x x]')' ];    % uncoment for n^3 fit
      %A=[ones(m,1) prod([x x x x]')' ];  % uncoment for n^4 fit
      b=y;
      % compute a= (A' * A) \ (A' * b) using QR factorization
      [Q R]=qr(A);
      a=R\(Q'*b);
      e = norm(A * a – b)/sqrt(size(b)(1));
end
```

Figure 8. *Octave* code for least-square fitting and error computation.

Numerical calculations of least-squares method and errors were made using the Octave language for numerical computions[9]. Figure 8 shows the code used. Linear algebra operations makes the method compact, efficient and numerically stable.

4. Results

In this section the results obtained by comparing the performance of the algorithms based on the classical neighbors lists data structures (NL) and the three variants of the algorithm based on the Delaunay triangulation are presented. Three variants of the Delaunay-based algorithms are analyzed: using one (D1) two (D2) and three (D3) levels of neighbors. Both qualitative analysis of the shape of the graphs of the results and numerical-statistical measurements are used.

4.1. graphical comparison of performances

We ran the implementation of the studied structures and recorded the time that they took for constructing the initial neighbors list and performing one iteration of the simulation. The results are shown in the graph of figure 9. The times reported here are the total system time required to construct the neighbors list. Actual times were measured in seconds but reported in arbitrary time units since they depend on several factors such as the processor speed and the software used. The absolute error of the measures is $\pm$ 0.5. From this graph we conclude that the Delaunay Triangulation with one or two levels is significantly better than the other two for simulations involving more than 50000 atoms.

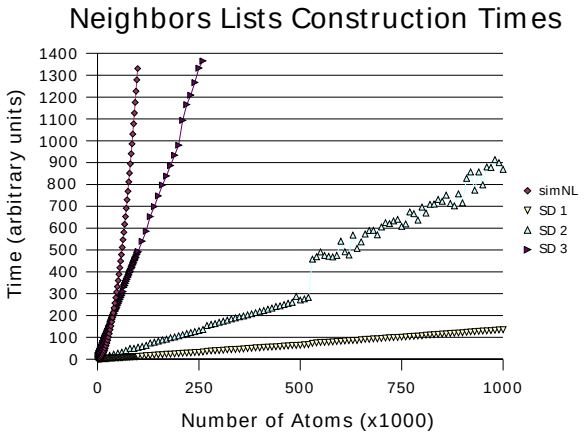


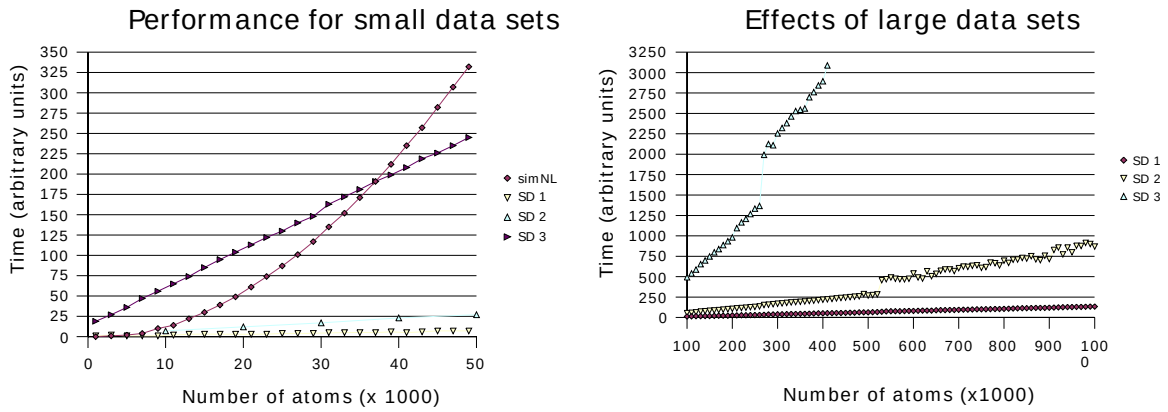
Figure 9: Comparison between the classical neighbors list evaluation (simNL), and to Delaunay construction with one, two or three levels of neighbors (SD 1, SD 2, SD 3 respectively).

The Delaunay based constructions are not always the best choice as figure 10(a) shows. When the problem is small, D3 is not more efficient than the classic NL algorithm. Otherwise any of the Delaunay-based algorithms are better than NL.

Considering the results presented so far we may reach the preliminary conclusion that the experimental results are consistent with the theoretical bounds. We will revisit this subject later in this article.

4.2. the effect of memory utilization on performance

As figure 10(b) shows, large problems degrades the behavior of the DT-based algorithms. By using system monitoring tools we observed that in those cases disk utilization was unusually high. Information obtained by the *top* command (figure 11) suggest that this is caused by virtual memory paging. Kswapd, a daemon that manages paging, significantly increases its activity during such events. Memory utilization analysis, which is beyond the scope of this article, is necessary for large problems.



(a) (b)

Figure 10: Experimental running times for (a) small and (b) large data sets of the classical neighbors list (NL), and Delaunay constructions with one, two or three levels of neighbors (D1, D2, D3 respectively).

```
PRI   SIZE %CPU %MEM    TIME COMMAND
15  10.6G  0.9 89.2  35:29 simDelaunay
15     0   0.3  0.0   7:23 kswapd
16  38496  0.1  0.0   0:09 top
```

Figure 11: Portion of the output of `top` command showing memory use of `simDelaunay`, the implementation of the Delaunay based NL construction, when constructing NLs from three level neighbors in a system of 400,000 atoms.

4.3. least-squares analysis of results

Now correlations between theoretical and empirical complexities are measured. Section 2.1 and 2.2 showed that the theoretical complexities of the NL algorithm and the DT-based ones are $O(n^2)$ and $O(n \log n)$ respectively. The qualitative information presented in section 4.1 suggest that the theoretical bounds are consistent with the empirical results. In order to confirm the qualitative analysis a conventional curve-fitting method is used to objectively test these conclusions.

The least-squares method and the error function as described in section 3.3 is used to fit the experimental data to some of the functions commonly used to describe complexity of algorithms. The data affected by intensive virtual memory paging was discarded for this test. The results of the least-squares method are shown in table 1.

Table 1: Least-squares errors of the fitting of commonly used theoretical complexity bounds and experimental running times. The underlined number shows the minimum error encountered for the corresponding algorithm.

	Linear	***n log n***	***Quadratic***	***Cubic***
Neighbors list	80.015	58.019	<u>2,67</u>	57,38
Delaunay one level neighbors	<u>1.1111</u>	1.8894	10,91	17,75
Delaunay two levels of neighbors	<u>3.8723</u>	5.4031	20,4	32,18
Delaunay three levels of neighbors	17.699	<u>12.365</u>	50,5	136,25

As expected, the NL construction correlates best to the n^2 function, and the D3 to the *n log n* function. What may be surprising is that the linear function shows the least error when fitted to the D2 and D1 algorithms, which is better than expected.

Finally, the performance degradation of the algorithms discussed in section 4.2 is analyzed. When the data that reflects performance degradation is fitted to the same functions as before an increase in the experimental complexity results (table 2). Algorithm D2 increases from linear to quadratic and D3 increases from *n log n* to cubic.

Table 2: Least-squares errors of the fitting of commonly used theoretical complexity bounds and experimental running times. The underlined number shows the minimum error encountered for the corresponding algorithm.

	Linear	***n log n***	***Quadratic***	***Cubic***	***N^4***
D2	29.54	29.1	<u>27.74</u>	29.45	33.62
D3	61.13	59.67	53.22	<u>51.23</u>	55.27

5. Conclusions

After comparing the performance of the algorithms based on the classical neighbors lists data structures (NL) and the three variants of the algorithm based on the Delaunay triangulation are presented, using one (D1) two (D2) and three (D3) levels of neighbors, the most efficient structure for the simulation is the Delaunay-based one-level neighbors lists. Other versions of Delaunay-based algorithms present a significant improvement over classical NL. Empirical results for the MD sistems being simulated show better performance than the theoretical estimates. Very large problems may present significant degradation of the performance probably due to memory paging.

6. Acknowledgments

This work was supported by the *Penn-UPR Partnership for Research and Education in Materials* project (NSF-DMR-353730), *Humacao Undergraduate Research in Mathematics to Promote Academic Achievement* program (NSA-H98230-04-C-0486) and *Humacao Research Scholarships* program (NSF-0123169). The author also thanks the advise of Dr. Pablo Negrón Marrero who suggested the addition of the analysis of degenerate cases in section 4.3.

7. References

1. CGAL Editorial Committee, ``Computational Geometry Algorithms Library", CGAL, http://www.cgal.org.
2. P. Agrawal, *et. al.*, "Algorithmic Issues in Modeling Motion", *ACM Computing Surveys* 34(4), December 2002, pp. 550-572.
3. D. C. Rapaport, *The Art of Molecular Dynamics Simulation*, 2nd edition, Cambridge.
4. N. León, "Electrical Characterization of Electronspun Sb-doped SnO_2 Nanofibers", *Proc. NCUR 2006* (accepted).
5. H. Zhong, J. Lukes, "Thermal Conductivity of Single-wall Carbon Nanotubes", *Proc. IMECE04*, Nov. 13-20, 2004.
6. F. Izquierdo, "Comparación de Construcciones de Triangulaciones de Delaunay para Aplicaciones de Dinámica Molecular", Technical report for HURMA Project, UPR at Humacao.
7. S. Yakowitz, F. Szidarovsky, *An Introduction to Numerical Computations*, 2nd edition,, MacMillan.
8. P. Negrón, *Fundamentos del Analisis Computacional*, (umpublished).
9. J. W. Eaton, "GNU Octave", *Octave Language*, http://www.gnu.org/software/octave.