Efficient Simulation of Interacting Particle Systems in Continuous Space and Time

Tomasz Ożański

Efficient simulation of interacting particle systems in continuous space and time

Tomasz Ożański

Wrocław University of Technology, Department of Electronics, Wrocław, Poland

Abstract. Interacting particle systems models became an increasingly important tool for modeling complex real-world phenomena with applications ranging from ecology, classical physics to tumor evolution. The need for efficient algorithm for simulations becomes therefore apparent. The existing simulation ideas and frameworks mostly make use of lattice models with discrete time. The work presents an efficient algorithm allowing simulation of complicated models while allowing particles positioned in continuous space with continuous timing between events.

Keywords: interacting particle system, event driven simulation, birth and death process, tumor evolution

1 Introduction

The original motivation for interacting particle systems comes from statistical mechanics, with the first models being investigated in XIX century, focusing mainly on dynamics of particles representing real atoms and molecules interacting by physical forces and following Newton's Laws. The notable results obtained by such formulation include Boltzmann's kinetic gas theory and Navier-Stokes equations.

Another class of models were later developed to investigate phenomena in crystal lattice. One of such models was the Ising model. The particles represented magnetic spins, each in one of two possible states -1 or 1 arranged in a grid structure representing metallic lattice. The dynamics of the system was driven by a Hamiltonian function including nearest-neighbor interactions between spin and optional external magnetic field.

While it was still possible to analyze Ising model in one and two dimensions using mathematical tools and characterize a phase transition precisely, it quickly became apparent that answering more complicated questions regarding dynamics of the system becomes difficult with the available mathematical tools.

Spin models can be reinterpreted by assigning an existence of a particle to one of the states and an empty spot to the other to obtain a birth and death process where flipping a spin means creation (birth) or removal (death) of a particle from the system. Such processes can be used to model biological systems. One of the first famous models of this type was Conway's Game of Life, it's aim was to emulate spreading of species on a two-dimensional space approximated by a

spin lattice and a set of simple rules: a birth occurs when an empty spots has exactly 3 neighbors and a death occurs if an existing particle has less than two or more than three neighbors [3].

The growing computational power of the first computers allowed to run simulations allowing the direct observation of system dynamics, leading to development of new tools to analyze such models like Markov Chain Monte Carlo algorithms for the Ising model [4]. Another framework developed at that time to broaden classes of problem that could be analyzed were cellular automata. In the classical cellular automaton the dynamics usually happens in generations, i.e. all the cells are updated simultaneously and the next state of a cell depends only on the state of its nearest neighbors.

Lattice models have their strength in straightforward implementation as a computer program which can be as simple as a loop iterating over an array. However, the simplicity also has its drawbacks – enforcing all particles to lie on a grid is only suitable for models where particles are inherently distributed on a grid, otherwise it produces an unnecessary interference into model's dynamics. Another drawback comes from limitation on simulation space, because of limited memory of a computer the space also becomes limited, this problem becomes especially evident for higher dimensions. For example in a 3D model of length 1024 in each dimension where one byte is required per lattice site one needs 1GB of memory just for storing the grid irregardless of how many particles there exist in the system.

For more advanced models the underlying lattice can be replaced with graph providing generalized neighborhoods for interaction. Many times the state transition rules can also be substituted by a probabilistic ones, what is a standard setting for both Contact Process or Voter Model, making them a Markov Processes. For a biological model more closely resembling nature, one would prefer to avoid using lattice models, but allow the particles to be in an Euclidean space.

Apart from continuous space, the Birth and Death Processes are usually also defined as continuous time processes. In the simplest case the state-changes are Markov process meaning only one event happens during a state change and the temporal spacing between two consecutive events is given as a rate for exponential distributions and depends on interactions between particles.

For simulation of an ecological system one usually considers two-dimensional continuous space with each particle representing an individual on a surface. The birth and death events of a particles directly correspond to start and end of life of modeled individual. The competition for resources between individuals and efficient spreading on large distances requires interactions to be long-ranged with close-ranged component emulating direct interactions between individuals.

Another area of applications which can be described by Birth and Death processes is dynamics of cell ensembles, especially in connection with models solid tumor evolution. In such models a cell is modeled as a particle, the birth and death event becomes a cell division and a cell death respectively. There are both short- and long-range interactions that can be considered here, the first type

comes from mechanical pressure and local competition for glucose and oxygen and the second corresponds to the signals carried over by various chemicals.

Allowing different types of particles and adding intrinsic variables can simulate changes of the genetic profile of the cells. Such simulations would provide a important tool for understanding Darwinian forces between different subclones of a heterogeneously composed tumors. Better understanding of complex processes in tumor evolution would hopefully also advance the search for a cure for a cancer.

2 System description

The system is defined on a set of particles, each having its position in D-dimensional euclidean space (with D being usually 2 or 3) and a set of other model-specific intrinsic parameters (attributes). The evolution of the class of systems can be described as a sequence of events each of them representing a single change to the state. The base events that can occur are:

- birth – creation of a particle to the configuration set
- death – removal of a particle from the configuration set

More complex events can be constructed as a combination of more than one base events, for example:

- jump – removal of a particle and creation of an identical one elsewhere
- division – removal of a particle and creation of two identical ones nearby
- mutation – removal of a particle and a creation of a new different one in the same place

While mutation is listed as complex event, for convenience and performance reasons it is better to consider it as another base event. In many models mutation would usually immediately follow any of the standard events to update the intrinsic parameters.

It is also worth noting that not all the dynamics can be captured by combining finite number of events and has to be approximated by a such, e.g. a continuous time motion can be replaced by a chain of shorter jumps.

The dynamics of the system is entirely described by timings between events which on the other hand depend on interactions between particles. In the deterministic case it means that the waiting time till the next event for a particle can be computed knowing the position of all the other particles and their intrinsic values. It is only required that no two events happen exactly at the same time, i.e. the order of the events is determined.

It is also possible to use stochastic timing where the distribution of the waiting time is known and depends on positions of all particles and their intrinsic values. In the simplest case the distribution is an exponential distribution with probability density function $f(x) = \lambda \exp(-\lambda x)$ with event intensity parameter λ dependent on interaction through functions called kernels.

The simplest death kernel is just a constant making lifespan exponentially-distributed with the rate of dying being same at any moment. An example of a death kernel with rate dependent on local density is

$$m(x;\gamma_t) = \sum_{y\in\gamma_t} e^{-\mathrm{dist}(x,y)}, \tag{1}$$

where γ_t is a configuration of all particles in the system at time t and $\mathrm{dist}(x,y)$ is Euclidan distance between particles x and y. In the presented kernel each neighbor contribution to local density is given by a Gaussian function.

A kernel describing an event in which a parent particle gives a birth to an offspring particle in principle can depend on both positions, however, it is easier to consider only the kernels where the placement of the offspring does not depend on the birth rate. For example a kernel

$$b(x,x';\gamma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(\mathrm{dist}(x,x'))^2}{2\sigma^2}} \sum_{y\in\gamma} e^{-dist(x,y)} \tag{2}$$

describes a birth rate in position x' by particle x under configuration γ. The rate depends on local density around x and the offspring particle x' is placed according to a normal distribution centered around x' with standard deviation σ. For more examples of kernels see for instance [1] or the references therein.

3 Simulation Algorithm

3.1 Naive algorithm

A naive algorithm which is capable of simulating the above process can be as follows

while configuration not empty **do**
  for every particle P from configuration **do**
    compute next event time
  end for
  decide which event comes first in time
  execute event
end while

The algorithm as presented above is not very efficient. Several measures which can be taken to improve the situation are described below.

3.2 Partial Updates

There is no need to do a full update of the event list at each iteration, it is only necessary to update those events which could have possibly been changed by the executed event. Unfortunately not much can be done in a case when every event depends on all particles in the configuration set.

However, very often this is not the case, in many systems one might expect that a particle will have a very little effect on what is happening in a distant location. This is the case if long-range correlations are not present or not important for the evolution of the system.

Assume that the maximum range of interaction is known to be $r_{\max}$. For an event which takes place at position x it is only needed to update all the events which have at least one dependency in ball with radius $r_{\max}$ centered at x.

3.3 Neighborhood Structure

To efficiently decide which events needs update the algorithm must be able to quickly find particles close to the place where the event occurred, therefore a special data structure is required to achieve this goal.

The space is subdivided into tiles, each representing a D-dimensional cube of fixed side-length approximately equal to $r_{\max}$. All particles which spatially belong to the given tile are compactly stored together in a resizable array. This approach fulfills the need for an efficient way to access all the particles in the given volume, but without an additional structure it does not provide an efficient way to find a single particle within the tile.

Non-empty ones are stored in a hash table with keys being their positions in D-dimensional space. Empty tiles are simply non-present elements in the hash table. This approach allows for having virtually no restrictions on simulation space while allowing constant-time tile access.

In contrast to other space-partitioning data structures like octrees or kd-trees an advantage of hash-based structure, apart from not having logarithmic access cost, is simplicity of implementation for multidimensional spaces. To access all the neighbors of a given point it is enough to visit a bounded number of tiles surrounding the point. In a two-dimensional case with tile size equal to $r_{\max}$ one has to visit up to 9 tiles.

3.4 Time Ordering

The update algorithm requires ability to quickly find the next event to be executed. To perform the operation efficiently a priority queue for all the events is required. The existing tile structure is extended with additional features allowing it to maintain a heap property.

For every tile it is enough to keep a pointer to a particle which event comes first within the tile. Whenever a new particle is added to the tile a single check is required to restore the pointer. One of the costly operations is removal of the particle from the tile which requires a linear search. However, in most use cases both insertion or removal is accompanied by an update to all the neighboring particles including those inside the tile. Both the incremental search and particle visitation have the same linear complexity in number of particles in tile therefore doing an incremental search does not increase the overall complexity. In fact an incremental search can be performed during particle visitation.

All the tiles are organized in a efficient heap structure namely a pairing heap which is believed to provide amortized logarithmic asymptotic complexity for all single-particle operations [2]. Adding all costs together a single event execution is linear in size of the neighborhood and has the same complexity in number of tiles as pairing heap.

3.5 Update algorithm examples

This subsection summarizes the ideas for an efficient simulation by giving example algorithms for the two basic events: birth and death.

Helper functions The helper functions used in the example algorithm are explained below

- HashFindTile(p) – finds and returns a tile to which position p belongs to
- CreateTile(p) – creates tiles to which position p will belong and inserts the tile to the hash table and heap-of-tiles
- RemoveTile(T) – removes tile T from hash table and heap-of-tiles and destroys the tile
- Push(T,P) – adds particle P to tile T
- Pop(T) – removes last particle from tile T
- ResetFirstEventPointer(T) – resets first event pointer in tile T
- IncludeInteraction(P,Q) – update particle P by including interaction with particle Q
- ExcludeInteraction(P,Q) – update particle P by excluding interaction with particle Q
- SampleNewEvent(P) – samples a new realization of an event for a particle P
- UpdateFirstEventPointer(T,P) – checks whether particle T would now be the one with first event in tile T and updates the first event pointer accordingly
- HeapOfTilesUpdate(T) – updates position of tile T in heap-of-tiles structure
- SwapContents(Q,R) – exchanges data of both particles including their positions,

Birth event A simple case of execution of a birth event requires updating all the interaction between newly-created particle and its neighbors it is possible to perform all the updates of the time-ordering structure at the same time. An example step-by-step algorithm for executing birth of a particle at position x is presented below:

function ExecuteBirth(position x)
  $T \Leftarrow$ HashFindTile(x)
  if T not found **then**
    $T \Leftarrow$ CreateTile(x)

```
    end if
    P ⇐ Particle(x)
    Push(T,P)
    for non-empty tile U ≠ T which can have particles within r_max from P do
        ResetFirstEventPointer(U)
        for every particle Q within tile U do
            if Distance(P,Q) < r_max then
                IncludeInteraction(P,Q)
                IncludeInteraction(Q,P)
                SampleNewEvent(Q)
            end if
            UpdateFirstEventPointer(U,Q)
        end for
        HeapOfTilesUpdate(U)
    end for
    ResetFirstEventPointer(T)
    for every particle Q in T except the last one do
        if Distance(P,Q) < r_max then
            IncludeInteraction(P,Q)
            IncludeInteraction(Q,P)
            SampleNewEvent(Q)
        end if
        UpdateFirstEventPointer(T,Q)
    end for
    GenerateNewEvent(P)
    UpdateFirstEventPointer(T,P)
    HeapOfTilesUpdate(T)
end function
```

Death event An execution of a death event requires excluding the interaction with the particle to be removed. Again it is possible to update time-ordering structures at the same time. In an example step-by-step algorithm for executing a death of a particle P is the following:

```
function ExecuteDeath(particle P)
    T ⇐ HashFindTile(P)
    for non-empty tile U ≠ T which can have particles within r_max from P do
        ResetFirstEventPointer(U)
        for every particle Q within tile U do
            if Distance(P,Q) < r_max then
                ExcludeInteraction(Q,P)
                SampleNewEvent(Q)
            end if
            UpdateFirstEventPointer(U,Q)
        end for
        HeapOfTilesUpdate(U)
```

```
    end for
    ResetFirstEventPointer(T)
    for every particle Q in T except the last one do
        if P = Q then
            R ⇐ Last(T)
            SwapContents(Q,R)
        end if
        if Distance(P,Q) < r_max then
            ExcludeInteraction(Q,P)
            SampleNewEvent(Q)
        end if
        UpdateFirstEventPointer(T,Q)
    end for
    Pop(T)
    if T = ∅ then
        RemoveTile(T)
    end if
end function
```

4 Summary and outlook

The running time of the algorithm greatly depends on ratio of tile size with to $r_{\max}$.

On one side too large tiles would result in unnecessary visitations of neighboring particles on the other side having too small tiles would mean large number of updates to heap-of-tiles structure. To find the optimal value for tile size one has to perform an empirical search since results might vary for different models and hardware. The optimal tile size search results for an example model are presented on Fig. 1.

The example model has the timings between events given by rates for exponential distributions – birth rate and death rate depend on intrinsic parameters of each cell and local density measured by a Gaussian kernel. The relation between density and birth rate and expected lifespan is given by a normalized Gaussian function. The placement of new particles is normally distributed with respect to the parent particle. After each birth event, with small probability both parent and child particle undergo a mutation modifying particle's intrinsic parameters.

The cost of updating of heap-of-tiles structure increases as number of tiles in the system becomes larger. Due to that fact the time needed for execution of a single event in a growing model also increases. The results for the example model are presented on Fig. 2.

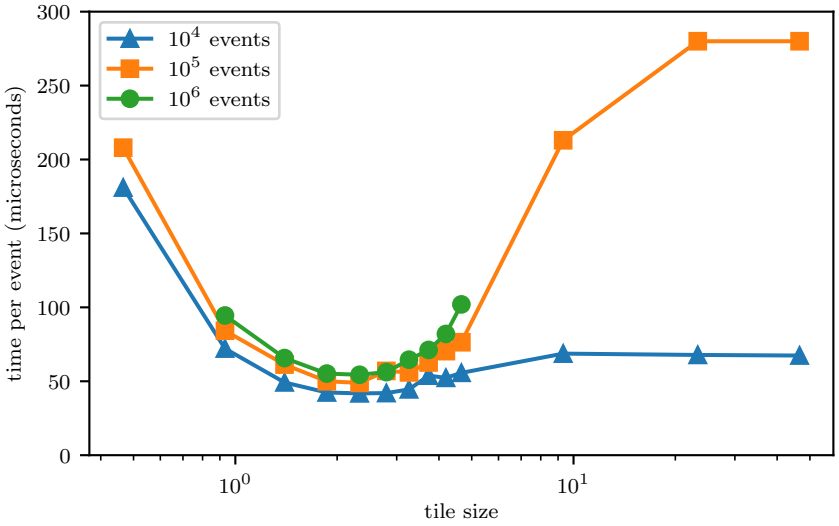

Fig. 1. Execution time per event in relation tile size (in multiplicities of interaction range $r_{\max}$). Tests were performed for runs of different lengths: 10^4, 10^5 and 10^6 events

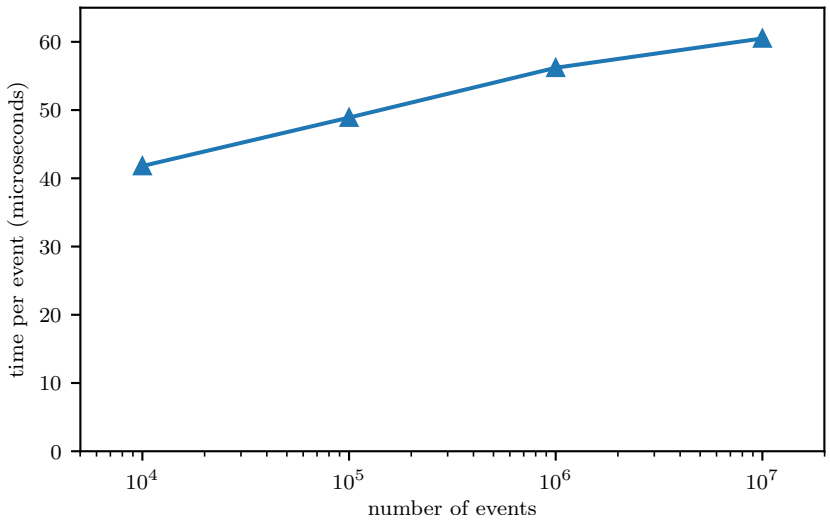

Fig. 2. Execution time per event in relation to number of events executed measured for on IntelCore i5 processor clocked at 2.6GHz. The execution of a million of events requires about one minute

During a run, a simulation gives access to all the particles' positions and their intrinsic values which can be used to produce snapshots of the data for viewing or static analysis. A render of a configuration from a sample cancer evolution model where each particle represents a tumor cell is presented on on Fig. 3.

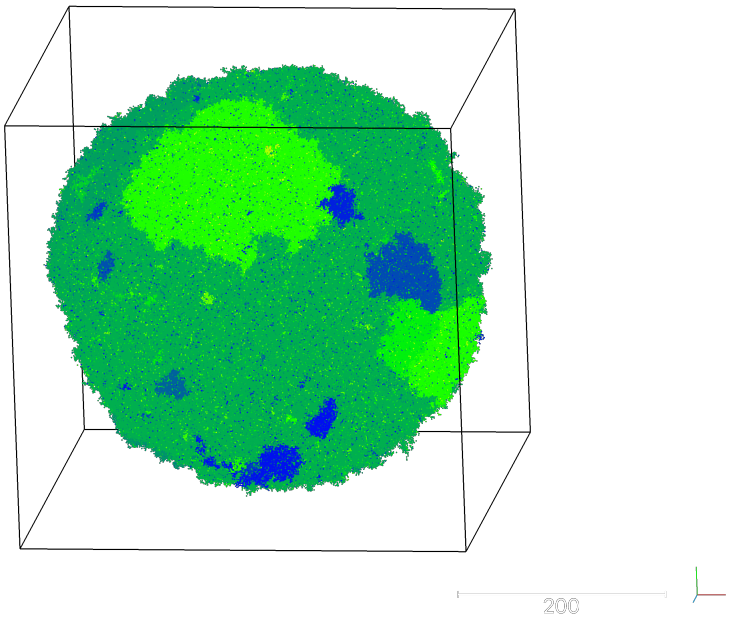

Fig. 3. Render of a sample 3D configuration having about 8.2 million points. Different colors correspond to different values of particles' intrinsic parameters. The render was created using CloudCompare software

The program can be also configured to produce a stream containing all the events in the chronological order which can be used for further analysis. While it is possible to store the stream on the disk and perform an off-line analysis it might be not suitable due to large disk space needs, instead the stream can be redirected to another simpler program which can analyze the data on-line without storing all the intermediate events.

One cubic centimeter of human tissue contains about 10^9 cells. In the development of the simulation framework all measures has been taken to minimize memory footprint, but still some memory overhead is required for keeping the data structures and memory preallocation. A realistic tumor simulation has to be able to handle ensembles of consisting of billions of cells. While the simulation program can handle configurations of this size, the limitation comes from long computing time and memory size of a typical computer. For just few intrinsic

variables the size of a single particle takes tens of bytes which directly translates into tens gigabytes for storing a billion-sized configuration.

The only way to increase capabilities for simulating larger systems seem to be parallelization of the algorithm where the total configuration is distributed across several computers to reduce simulation time and per-machine memory requirements. However, keeping the time order of the events in during distributed computations seems to be major difficulty.

Acknowledgements I thank my supervisor Tyll Krüger for guidance and corrections to this work. I grateful to Ewaryst Rafajłowicz and Ewa Skubalska-Rafajłowicz for valuable discussions and support during my PhD program at Wrocław University of Technology. This research was supported in part by PL-Grid Infrastructure.

I thank for the support of Yuri Kondratiev and hospitality and financial support for several visits in Bielefeld University. I'm thankful to Center for Interdisciplinary Research (ZiF) in Bielefeld where part of the work was carried out in ZiF cooperation group *Multiscale Modelling of Tumour Initiation, Growth and Progression: From Gene Regulation to Evolutionary Dynamics*.

I thank my colleagues Mykola Lebid, Viktor Bezborodov and Marcin Bodych for many discussions on the topic.

References

1. Finkelshtein, D., Kondratiev, Y., Kutoviy, O.: Individual based model with competition in spatial ecology. SIAM Journal on Mathematical Analysis 41(1), 297–317 (2009)
2. Fredman, M.L., Sedgewick, R., Sleator, D.D., Tarjan, R.E.: The pairing heap: A new form of self-adjusting heap. Algorithmica 1(1-4), 111–129 (1986)
3. Gardner, M.: Mathematical games: The fantastic combinations of john conways new solitaire game life. Scientific American 223(4), 120–123 (1970)
4. Liggett, T.: Interacting particle systems, vol. 276. Springer Science & Business Media (2012)