Simulation of a geometric P system

March 20, 2016

Blue parts of the algorithm can wait.
They can be replaced with temporary solutions in red.

Input: A geometric P system (description files in the Inventory directory)

Output: Snapshot files in the Snapshots directory

Data structures:

- list `lFixedObjects` of existing fixed objects, each element corresponding to one copy of a fixed object, initialized to initial objects in the inventory file;
- `Multiset mFloatObjects` – multiset of existing floating objects, one element for each type of object in the inventory file, each object with obligatory fields `String name` a `int numCopies`;
- `Multiset mNewFloatObjects` – multiset of new floating objects created during applications of rules in one step;
- each connector and each metabolic protein has the boolean field `isUsed`
- connection graph of existing fixed objects, each graph vertex corresponding to one (copy of) object, each graph arc to a pair of connectors bound together.

  Proposed graph implementation:
  - each connector has fields (i) `connectedTo` of type `Connection`, containing subfields `fixObject` of type `FixedObject` and `connector` of type `Connector`, (ii) `isActive` - boolean
  - each surface protein has a field `connections` – a list of elements of type `Connection`;
  - initially set all `connectedTo=null`, `isActive=true`, `connections=empty list`;
  - when two connectors on fixed objects get connected, each one sets its `connectedTo` to the reference to the other fixed object and the other connector;
  - when a connector is connected to a surface protein, then the connector sets its `connectedTo` field as above, and the surface protein creates and adds the corresponding `Connection` object to its `connections` list;
  - analogously the fields are updated when a connection is broken or a fixed object is destroyed.

Initialization: Place all initial fixed object in the imventory file to their required positions (posX,posY,posZ), rotate them to angles (angleX,angleX,angleZ) and add them to the `mFixedObjects` list. (NOTE: when moving and rotating an object you must re-calculate coordinates of all its vertices and connectors);

Main loop: repeat

- apply metabolic rules (affects fields `isUsed` in proteins and multisets `mFloatObjects` a `mNewFloatObjects`);
- apply rules destroying fixed objects (sets fields `isUsed` in connectors);
- apply rules creating fixed objects (sets various fields in connectors);
- apply division rules (sets various fields in connectors);
- for all connectors and all metabolic proteins on fixed objects set `isUsed = false`;
- `mFloatObjects` = union of `mFloatObjects` and `mNewFloatObjects`;
- `mNewFloatObjects` = empty multiset;
- if any division rule was applied, try to apply mitosis;

until no rule was applicable

==========

Method: apply metabolic rules

`availFloatRules` = set of all rules for floating objects; /* *set of rules that still have a chance to be applied* */

while `availFloatRules` $\neq \emptyset$ **do**

- choose randomly one rule from `availFloatRules`;
- **if** the rule is applicable (the floating objects and the protein at its left-hand side are available and unused), **then**
  - subtract the multiset of floating objects at its left-hand side from `mFloatObjects`;
  - add the multiset of floating objects at its right-hand side to `mNewFloatObjects`;
  - for the protein at its left-hand side set field `isUsed = true`;
- **end else** remove the rule from `availFloatRules`;

==========

Method: apply rules destroying fixed objects

`availDestructRules` = set of all destruction rules;
while `availDestructRules` $\neq \emptyset$ **do begin**

- choose randomly one rule $us \rightarrow v$ from `availDestructRules`;
- **if** the multiset u is contained in `mFloatObjects` and the fixed object s exists in the list `lFixedObjects` (if there are more s then choose one randomly) **then**
  - subtract u from `mFloatObjects`;
  - for all connectors on other objects connected to s, set `connectedTo=null`, `isActive=true` and `isUsed=true`;

- if s is connected by its connectors also to some surface proteins on other objects, then delete the connection to s from their `connections` lists;
- delete s from the list `lFixedObjects`;

- **end else** remove the rule from `availDestructRules`;

==========

Method: apply rules creating fixed objects

`availCreationRules` = set of all rules creating fixed objects;
while `availCreationRules` $\neq \emptyset$ **do**

- choose randomly one rule $u \to t$ from `availCreationRules`;
- create empty list `candidateConnections` to which t could eventually connect;
- if the multiset u is contained in `mFloatObjects`, then fill the list `candidateConnections` of elements of type Connection as follows:
  1. construct the set P of all proteins to which the object t could connect: for each connector on t with a protein q search the glue relation G for triples $(p, q, v) \in G$ (where $v \in O^*$ can be an arbitrary multiset), and add p to the set P;
  2. search the objects in the list `lFixedObjects`; add each connector with `isUsed=false` and `isActive=true` and protein in P to the list `candidateConnections`;
- `Connection connection = null`;
- **while** `candidateConnections` is not empty and `connection = null`**do**
  - `connection` = random connection from `candidateConnections`; denote p its connector protein, φ its angle on an object s;
  - let v = newObjectDirection(t, `connection`);
  - add t to the `connection` in the direction v (NOTE: when moving and rotating t, you must re-calculate coordinates of all its vertices and connectors);
  - if t would intersect another fixed object in its interconnected graph, then
    - if t is a 1D object, then t is shortened (its tip connector positions also changed!) just to touch the other object
    - else the growth is blocked: remove t, remove `connection` from the list `candidateConnections` and set `connection = null`;

    PUSHING:
    - each interconnected graph of fixed objects is enclosed in a rectangular block;
    - if a growing block overlaps with another, a pushing line connecting their centres (+ small random angle up to $\pi/4$) is established;
    - both blocks are pushed in opposite directions along the line so as not to overlap; the movement of blocks is anti-proportional to their weights (initially the weight is just the number of objects in each block, later fixed objects can have assigned weights);
    - if, during its movement, a block hits another block, a chain pushing reaction proceeds, but this time the direction is fixed to the established pushing line;

- **if** `connection` = null then
  - remove the rule from `availCreationRules`

  else
  - subtract u from `mFloatObjects`;
  - check all free connectors of t whether they would bind to (1) a connector or (2) surface protein located within the `radius` distance on another object in the list `lFixedObjects`;
  - for all newly established connections:
    - fill the field `connectedTo` and set `isUsed=true` in both participating connectors;
    - if a connector connects to a surface protein, add the connector to the list `connections` of the surface protein;
    - add the multiset v of signalling objects due to the glue relation G to `mNewFloatObjects`.

==========

Method: apply division rules

`availDivisionRules` = set of all division rules;
while `availDivisionRules` $\neq \emptyset$ **do**

- choose randomly one rule $(p,q)|u \to (p)(q)$ from `availDivisionRules`;
- **if** the multiset u is contained in mFloatObjects and the pair of connected proteins (p,q) exists in the connection graph (if there are more such pairs (p,q) then choose one randomly) **then**
  - subtract u from `mFloatObjects`;
  - for both connectors containing p and q set `connectedTo=null` and `isActive=false`; if one of p,q is a surface protein, then remove the other from its `connections` lists;
- **else** remove the rule from `availBreakRules`;

==========

Method: tryMitosis

Input: Cell in which a division rule was applied

Output: void

- set isActive=true in all connectors
- each connected subgraph of fixed objects containing all initial objects (in the multiset sense) forms a separate cell;
- if there are other connected subgraphs not containing all initial objects, they become parts of the closest cell (the distance of two blocks is the distance of the two their closest object vertices);
- floating objects are divided regularly among the descendant cells;

==========

Method: newObjectDirection

Input: FixedObject t which has to be connected to Connection c

Output: Direction vector $\mathbf{v}$ for the object t

Denote $s = c$.`fixedObject`;

if c.connector is an edge connector then
    $\mathbf{u}$ = the vector perpendicular to the edge connector of c and lying in s;
else
    if c.position = $\mathbf{0}$ then $\mathbf{u}$ = an arbitrary vector within s;
    else $\mathbf{u}$ = (Vector)c.`connector.position`;

if s is a 2D object then
    $\mathbf{w}$ = the vector perpendicular to s
else set the vector $\mathbf{w}$ as follows:

- let i be the index of the first nonzero components of $\mathbf{u}$;
- set each components of $\mathbf{w}$ except the i-th to a random number between $\langle -1, 1 \rangle$;
- calculate the i-th component of $\mathbf{w}$ so that $\mathbf{w}$ is perpendicular to $\mathbf{u}$.

/* *Vectors* $\mathbf{u}$ *and* $\mathbf{w}$ *define the plane where the vector* $\mathbf{v}$ *will be placed.* */

Set $\mathbf{v}$:= $\mathbf{u}$, then rotate $\mathbf{v}$ by the angle c.connector.φ in the plane given by $\mathbf{u}$ and $\mathbf{w}$.

`return` $\mathbf{v}$;

CLOSEDNESS CONDITION OF A CELL:

- candidate cell = connected graph of 2D fixed objects;
- from the cell centre set 38 half-lines:
  - 6 in basic axes directions;
  - 8 in centres of all octants (direction calculated as the center of triangle);
  - 24 in centres of all small triangles (optional);
- if all intersect some fixed object in the graph, the graph is assumed to be a closed cell.