A Constrained, Force-Directed Layout Algorithm for Biological Pathways

Burkay Genc and Ugur Dogrusoz

Computer Engineering Department and
Center for Bioinformatics, Bilkent Univ., Ankara 06800, Turkey

Abstract. We present a new elegant algorithm for layout of biological signaling pathways. It uses a force-directed layout scheme, taking into account directional and regional constraints enforced by different molecular interaction types and subcellular locations in a cell. The algorithm has been successfully implemented as part of a pathway integration and analysis toolkit named PATIKA, and results with respect to computational complexity and quality of the layout have been found satisfactory.

1 Introduction

As graphical user interfaces have improved, and more state-of-the-art software tools have incorporated visual functions, interactive graph editing and diagramming facilities have become important components in visualization systems [4]. Biology is no exception. In order to make useful deductions about a cell, an inherently complex multi-body system, one needs to consider cellular pathways as an interconnected network rather than separate linear signal routes.

There has been a few studies done specifically for layout of biological pathways as well, focusing on metabolic pathways. Karp and Paley [6] proposed a divide-and-conquer algorithm to identify a number of pre-determined subtopologies such as paths, cycles, and trees so that different layout approaches may be applied on each part. Becker and Rojas [1] improve this approach by supplementing a special force-directed layout algorithm and additional layout heuristics.

PATIKA [3], a pathway database and tool, is mainly intended for signaling pathways whose underlying graph structure can be arbitrarily more complicated and irregular than that of metabolic pathways.

In this paper, we introduce an efficient and powerful layout algorithm devised for pathway graphs as defined by PATIKA ontology [2]. It is based on the spring force directed layout algorithm [5] with regional constraints. It also uses a similar idea to magnetic fields of Sugiyama [7] but employs per edge fields to enforce edge orientation constraints, which are allowed to adaptively change during layout.

2 Pathway Model

The structure of pathway graphs highly depend on the type of the pathways and the ontology used to represent the biological phenomenon. We assume the basics

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 314–319, 2004.

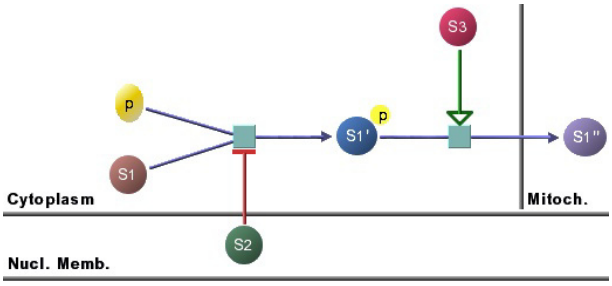

Fig. 1. An example illustrating the basics of the assumed ontology. The states, transitions, and interactions (substrates such as the one with source $S1$, products such as the one with target $S1'$, and effectors such as the one with source $S2$) are represented with ovals, rectangles, and lines of varying types, respectively, and cellular compartments are separated by orthogonal lines.

of the ontology described in [3,2], which represents a cellular process in the form of a directed graph called *pathway graph* (Figure 1). Usually the pathway graphs representing signaling pathways do not possess the uniform properties that those representing metabolic pathways do.

3 Layout Algorithm

We have chosen to use a force-directed layout algorithm with constraints to satisfy the criteria of the specific underlying model as well as the general conventions in pathway graph drawings. Basically, it is a virtual dynamic system in which nodes are assumed to have a certain "mass", connected via "springs" of a pre-specified desired length. Thus each node in a pathway graph is applied both *spring* and *node-to-node repulsion* forces. Spring forces include *relativity constraint* forces that are applied on each substrate, product, activator or inhibitor node to align the corresponding edge to lie towards the left, right, top or bottom of the associated transition, respectively. Furthermore, each horizontal (vertical) compartment separator is part of this physical system, on which the rest of the system can apply forces, moving them in only vertical (horizontal) direction. We also assume "gravitational" forces on compartment separators, disallowing a compartment to unnecessarily expand (Figure 2). Thus the optimal layout is regarded as the state of this system in which total energy is minimal.

The layout algorithm is split into three *major phases*, each of which alternates between odd and even-numbered *minor phases*. The first major phase is mainly for unscrambling the pathway graph with the help of high repulsion force ranges and the concept of *pulsing*. This is achieved by expanding the graph to a much larger area in a new minor phase compared to the previous one, and vice versa.

The second phase is where each edge adapts a best orientation for itself with the concept of "maturity". As an edge stays in a certain orientation (e.g., left-to-

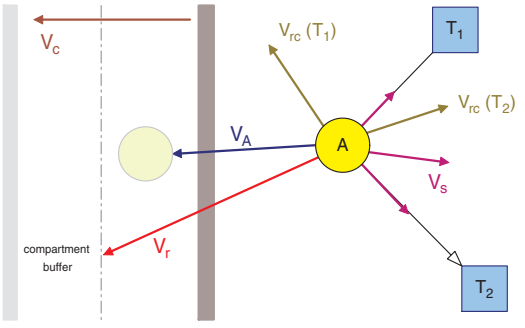

Fig. 2. An example showing various types of forces on a state A (V_s, V_r, and V_{rc}: spring, repulsion, and relativity constraint forces, respectively) and a compartment separator. Both move towards left by total forces V_A and V_c, respectively.

right) over consecutive iterations, its maturity is increased; and after a certain period, it "adapts" this orientation.

The last major phase is the stabilization phase, where all forces are pulled down to a minimum level, and pulsing and adaptive layout are disabled. In this phase compartments are also allowed to shrink.

The following method is used for calculating the relativity constraint forces acting over an edge. The method is clearly of $\Theta(1)$ time complexity.

algorithm ApplyRCF(*Edge* e)
(1) $\{u, v\}$ = {source,target} node of e
(2) **if** this is an adaptive layout **then**
(3) **if** we are at major phase 2 **then**
(4) Increment *maturity* of e
(5) **if** e is mature **and** orientation is not satisfied **then**
(6) Change orientation of e as appropriate
(7) Calculate V_{rc} on e according to its orientation
(8) Split the force into components: V_{rc}^x and V_{rc}^y
(9) Update $\{u, v\}.sf.x$ and $\{u, v\}.sf.y$ by V_{rc}^x and V_{rc}^y, resp.

The next method is of $\Theta(|E|)$ and calculates the general spring forces acting on each edge using $F_s = (\lambda - edgeLength)^2/\eta$, where λ is the ideal edge length and η is the elasticity constant of the edge.

algorithm ApplySpringForces(*Graph* $G = (V, E)$)
(1) **for** $e \in E$ **do**
(2) $\{u, v\}$ = {source,target} node of e
(3) Calculate the spring force V_s acting on e
(4) Split the force into components: V_s^x and V_s^y
(5) Update $\{u, v\}.sf.x$ and $\{u, v\}.sf.y$ by V_s^x and V_s^y, resp.
(6) **call** ApplyRCF(e)

Node-to-node repulsion forces are calculated using the formula $F_m = \alpha/(d_x^2 + d_y^2)$, where α is the repulsion constant and d_x and d_y are the differences in x and y coordinates of the two repulsing nodes, respectively.

```
    algorithm ApplyMassForces(Graph G = (V, E))
(1)     Create empty set S of layout nodes
(2)     for u ∈ V do
(3)         Insert u into S
(4)         for v ∈ V − S do
(5)             if u and v are in repulsion range then
(6)                 Calculate repulsion force V_r acting on u and v
(7)                 Split the force into components: V_r^x and V_r^y
(8)                 Update {u, v}.rf.x and {u, v}.rf.y by V_r^x and V_r^y, resp.
```

Steps 6-8 are handled in $\Theta(1)$ steps executed a total of maximum $O(|V|^2)$ times. However, since a node pair affect each other only when they are below a certain geometric distance, the average complexity is expected to be lower.

The following method controls the compartment constraints.

```
    algorithm CheckCompartmentRules(Graph G = (V, E))
(1)     for u ∈ V do
(2)         Calculate newX, newY, newRx and newRy based on old
                coordinates and V_s^x, V_s^y, V_r^x and V_r^y values of u
(3)         if u is a state then
(4)             if compartment bounds are violated by newRx or newRy then
(5)                 if compartment resizing is enabled then
(6)                     Resize compartment of u
(7)                 else
(8)                     Alter V_r^x, V_r^y so as to keep u within compartment borders
(9)             if compartment bounds are violated by newX or newY then
(10)                Alter V_s^x, V_s^y so as to keep u within compartment borders
(11)        Increment error by V_r^x, V_r^y, V_s^x and V_s^y of u
(12)        Update coordinates of u.x and u.y with newX and newY, resp.
```

Step 6 might require displacement of all nodes taking $O(|V|)$ time to complete. The compartments are normally resized no more than once or twice per iteration. Thus, the overall time complexity is $O(|V|^2)$ in the worst case and $O(|V|)$ on the average.

The main layout algorithm is as follows:

```
    algorithm Layout()
(1)     Set step to 0
(2)     if an incremental layout is to be done then
(3)         Increment step to second major phase
(4)     else
(5)         Set repulsionRange to MAX_REPULSION_RANGE
(6)     while step ≤MAX_ITERATION_COUNT do
(7)         if entering second major phase then
(8)             Set repulsionRange to desiredRange for second major phase
(9)         Set error to 0
(10)        call ApplySpringForces()
(11)        if in an odd minor phase or in third major phase then
(12)            call ApplyMassForces()
(13)        call CheckCompartmentRules()
```

(14) **if** in third major phase **and** *step* mod *shrinkPeriod* == 0 **then**
(15) Shrink all compartments from all sides as much as possible
(16) **if** *error* <*ERROR_THRESHOLD* **then**
(17) Jump to next minor phase by adjusting *step*
(18) **if** in third major phase **then**
(19) Immediately finish layout
(20) Increment *step* by 1

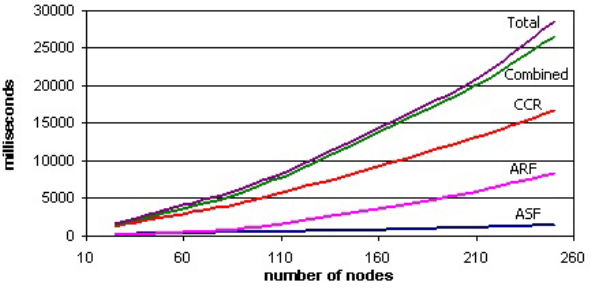

Fig. 3. Graph size vs. execution time.

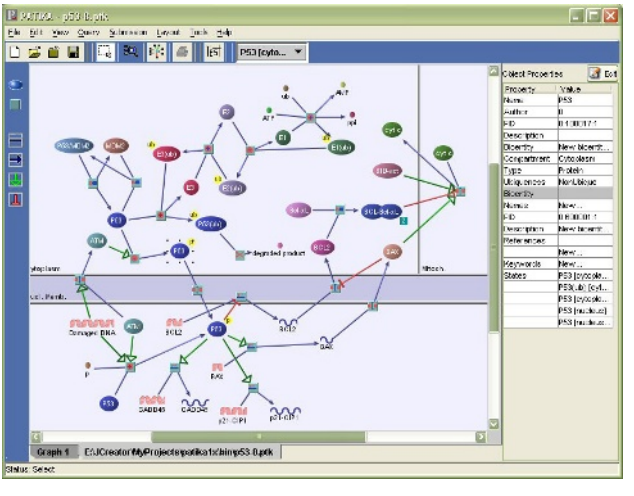

Fig. 4. An example layout for the p53 pathway.

The first and second major phases only differ in the amount of repulsion range considered when calling ApplyMassForces. For the odd-numbered minor phases and first two major phases the overall worst-case time complexity of each layout iteration is $O(|E| + |V|^2 + |V|) = O(|V|^2)$ for sparse graphs. For

the even-numbered phases this is reduced to $O(|E| + |V|)$. In the third major phase, the repulsion forces are always calculated; additionally, a shrink operation is performed at certain periods yielding an overall complexity of $O(|V|^2)$ for sparse graphs. In the worst case if we assume that all phases are executed to the end and all node pairs are considered for repulsion calculations, the overall time complexity is $O(K \cdot |V|^2)$ over a total of K iterations needed for minimizing the total energy of the system.

4 Implementation and Results

The algorithm described above has been implemented and tested within the PATIKA pathway editor [3]. For each test a random graph is generated and all nodes are randomly assigned a compartment. The number of edges per graph is chosen to be linear in the number of nodes as in a typical pathway graph. For similar reasons one in every 20 edges or so are added as a back edge to form a new cycle.

Figure 3 shows the run time behavior of each layout component with increasing number of nodes. It is clear that the time spent inside the ApplySpringForces method is linear with respect to the number of nodes as expected. Execution time of the algorithm is affected by other parameters of the algorithm as suggested by the theoretical analysis.

The quality of the layout is found to be acceptable in terms of general graph drawing criteria (e.g., discovering symmetries, minimizing edge crossings) as well as pathway graph drawing conventions (Figure 4).

References

1. M. Y. Becker and I. Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17:461–467, 2001.
2. E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, A. Ayaz, G. Gulesir, G. Nisanci, and R. Cetin-Atalay. An ontology for collaborative construction and analysis of cellular pathways. To appear in Bioinformatics, 2003.
3. E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk. PATIKA: An integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, 2002.
4. U. Dogrusoz, Q. Feng, B. Madden, M. Doorley, and A. Frick. Graph visualization toolkits. *IEEE Computer Graphics and Applications*, 22(1):30–37, January/February 2002.
5. T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, 1991.
6. P. D. Karp and S. Paley. Automated drawing of metabolic pathways. In *Third International Conference on Bioinformatics and Genome Research*, pages 225–238, Tallahassee, Florida, June 1994.
7. K. Sugiyama and K. Misue. A simle and unified method for drawing graphs: Magnetic-spring algorithm. In R. Tamassia and I. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 364–375. Springer-Verlag, 1995.