Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Sciences 176 (2006) 135–149

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

A layout algorithm for signaling pathways

B. Genc [a,b], U. Dogrusoz [a,c,*]

[a] *Computer Engineering Department and Center for Bioinformatics, Bilkent University, Bilkent 06800, Ankara, Turkey*
[b] *Computer Science Department, University of Waterloo, Waterloo, ON, N2L 3G1 Canada*
[c] *Tom Sawyer Software, Oakland, CA 94612, USA*

Received 29 December 2003; received in revised form 22 November 2004; accepted 27 November 2004

Abstract

Visualization is crucial to the effective analysis of biological pathways. A poorly laid out pathway confuses the user, while a well laid out one improves the user's comprehension of the underlying biological phenomenon.

We present a new, elegant algorithm for layout of biological signaling pathways. Our algorithm uses a force-directed layout scheme, taking into account directional and rectangular 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 visualization and analysis toolkit named PATIKA, and results with respect to computational complexity and quality of the layout have been found satisfactory. The algorithm may be easily adapted to be used in other applications with similar conventions and constraints as well.

PATIKA version 1.0 beta is available upon request at http://www.patika.org.
© 2004 Elsevier Inc. All rights reserved.

[*] Corresponding author. Address: Computer Engineering Department and Center for Bioinformatics, Bilkent University, Bilkent 06800, Ankara, Turkey. Tel.: +90 312 290 1612; fax: +90 312 266 4047.
E-mail address: ugur@cs.bilkent.edu.tr (U. Dogrusoz).

0020-0255/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2004.11.006

Keywords: Bioinformatics; Signaling pathways; Cellular pathway analysis; Information visualization; Pathway visualization; Graph layout

1. Introduction

Graphs are commonly used to model relational information that arises in numerous areas from software engineering to telecommunications to biology. Objects are the *nodes* or *vertices* in a graph; relations or links are the *edges* in a graph. The usefulness of the relational model depends on whether the drawing, or the *layout*, of the graph effectively conveys the relational information to the users. A poorly drawn diagram confuses the user of an application, while a well laid out diagram improves the user's comprehension of the data.

As graph drawing applications have grown larger in terms of the size of graphs displayed, manual layout of graphs has become more difficult and tedious. This has motivated a great deal of research in automatic graph drawing [1]. 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 [2]. Biology is no exception; human genome is expected to create an extremely complex network of information, composed of hundred thousands of different molecules and factors [3,4]. Knowing the map of this network as completely as possible is very important since it will potentially explain the mechanisms of life processes as well as disease conditions. Such knowledge will also serve as a key for further biomedical applications such as development of new drugs and diagnostic approaches. In this regard, a cell can be considered as an inherently complex multi-body system. In order to make useful deductions about such a system, one needs to consider cellular pathways as an interconnected network rather than separate linear signal routes (Fig. 1).

Recently a number of molecular interaction and pathway databases and integrated software tools have been developed to address this need [5–10]. Even though some of these databases support visual interfaces based on graph representations, most of them either use static images or cannot cope well with more complex, non-standard pathways. A multitude of general [1] and constrained [11–14] graph layout algorithms that have been developed in the past, does not seem to be able to directly address the specific needs and established conventions of pathway graphs. This is due to a number of reasons including directional and regional constraints inherent in a pathway graph. In many cases, the relations among pathway elements impose a pattern on the pathway. In the literature, such molecular patterns are drawn in varying ways highly depending on the pathway ontology used. In order to create a meaningful drawing of the given pathway data, one should employ extensive techniques

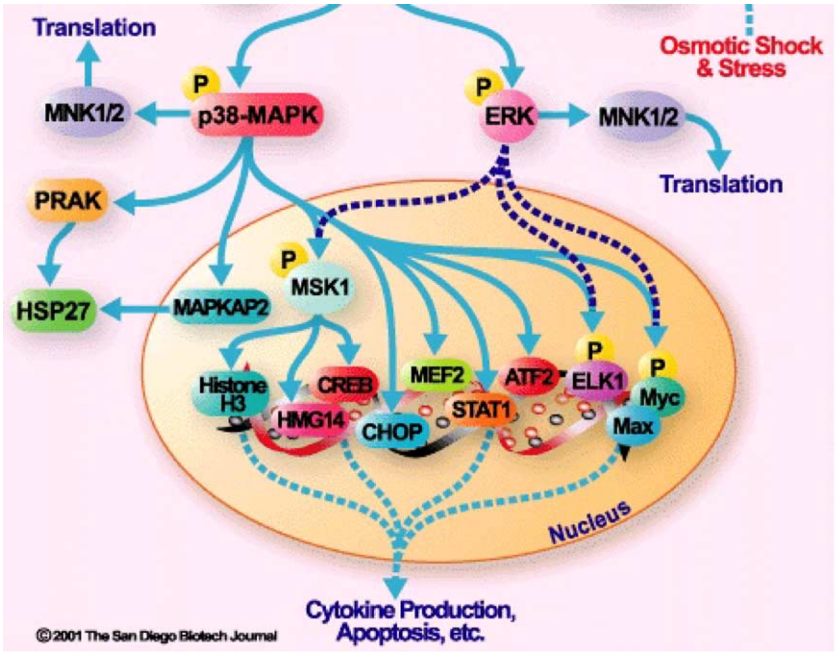


Fig. 1. A (partial) example of a signaling pathway. (For colour see online version.)

of graph layout to visualize such patterns and emphasize the contextual meaning of the data.

There has been a few studies done specifically for layout of biological pathways, focusing on metabolic pathways. Karp and Paley [15] 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 [16] improve this approach by supplementing a special force-directed layout algorithm and additional layout heuristics. Schreiber [17] presents a layout algorithm for drawing biochemical pathways within BioPath, based on the graph drawing algorithm by Sugiyama et al. [18] for the computation of layered layouts of directed acyclic graphs, with extensions for clustering disjoint subgraphs and some domain specific constraints. Unfortunately none of these algorithms may be applied to signaling pathways using a state-transition ontology and a compartment based cell model.

PATIKA [8], a pathway database and tool, is composed of a server-side, scalable, object-oriented database and client-side editors to provide an integrated, multi-user environment for visualizing and manipulating network of cellular events. PATIKA 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 [19]. Our algorithm is based on the spring force directed layout algorithm [20] with regional constraints. Moreover, it uses a similar idea to magnetic fields of Sugiyama et al. [21] but employs per-edge fields to enforce edge orientation constraints, which are allowed to adaptively change during layout. An additional technique called "pulsing" is applied to reduce edge crossings and node overlaps. Finally, it is naturally incremental as an update on the pathway data may be quickly reflected on the previous layout.

2. Pathway model

The structure of pathway graphs highly depend on the type of the pathways (e.g., metabolic or signaling) and the model or ontology used to represent the biological phenomenon. We assume the basics of the ontology described in [8,19], which represents a cellular process in the form of a directed graph called *pathway graph*, using a compartment based state-transition pathway model. Usually the pathway graphs representing signaling pathways do not possess the nice, uniform properties (e.g., can be decomposed into paths, cycles, and trees easily) that those representing metabolic pathways do.

A molecule may have any number of *states* to depict changes in its information context through chemical or physical modifications. Changes in the subcellular location of a molecule are also regarded as a change in its information context. In order to reflect these changes each state is associated with exactly one *compartment* such as cell membrane, nucleus or mitochondria. A *transition*, represented as a distinct type of node, provides a convenient mean for conveying event-specific information like reaction constants or the complex regulatory behavior. Each transition has a number of associated states, which may be *products*, *substrates* or *effectors* of the transition. All these relations are represented by different edge types (Fig. 2).

3. Algorithm

We build up our algorithms based on the model described above. Each state is associated with a compartment, and each compartment is a rectangular area defined by two horizontal and two vertical compartment separators. Whenever possible, we follow the conventions and draw substrate(s) on one side of the transition (to its left by default) and product(s) of a transition on the opposite side (to its right by default), respectively. This naturally leads to left-to-right

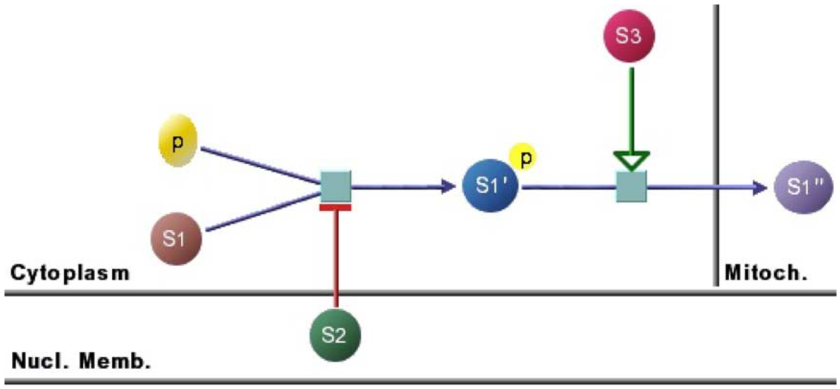


Fig. 2. An example illustrating the basics of the assumed ontology. The states, transitions, and interactions (substrates such as the one with source $S1$, products such as the one with target $S1'$, and effectors such as the one with source $S2$) are represented with ovals, rectangles, and lines of varying types, respectively, and cellular compartments are separated by orthogonal lines. (For colour see online version.)

oriented hierarchical structures for acyclic pathway graphs for which compartmental constraints do not violate this orientation (e.g., product of a transition is in a compartment to the left of the compartment in which the same transition's substrates reside).

3.1. Main idea

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 be physical objects with a certain "electrical charge", 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, along with the associated transition, to align the corresponding edge to lie towards the left, right, top or bottom of the 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 (one per compartment) on compartment separators, disallowing a compartment to unnecessarily expand. Fig. 3 explains varying types of such forces with a sketch. As a result, the optimal layout is regarded as the state of this system in which total energy is minimal.

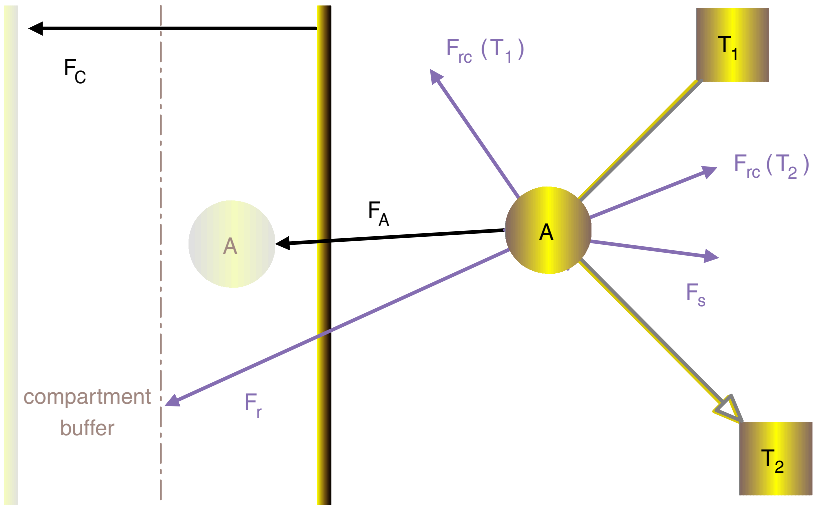


Fig. 3. An example showing various types of forces on a state *A* (F_s, F_r, and F_{rc}: spring, repulsion, and relativity constraint forces, respectively) and a compartment separator. As a result both move towards left as defined by total forces, F_A and F_C, respectively, acting upon them. (For colour see online version.)

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. Here we use the concept of *pulsing* to avoid node overlaps and decrease edge crossings. This is achieved by having high repulsion forces and turning them on and off at alternating minor phases, creating a pulse-like effect, similar to that of the heart of a living being; the graph expands 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. Here we use the concept of "maturity" for the orientation of an edge. As an edge stays in a certain orientation (e.g., left-to-right or top-to-bottom) over consecutive iterations, its maturity is increased; and after a certain period, it "adapts" this orientation. This is especially useful when default orientation cannot be satisfied. For instance, if the flow of the pathway is from mitochondria to ER (i.e., from right-to-left), the edges on the pathway will adapt a right-to-left orientation after a while.

The last major phase is the stabilization phase, where all forces are at a minimum level, and pulsing and adaptive layout are disabled. In this phase, compartments are also allowed to shrink, so unnecessary space around the compartment bounds can be eliminated. In other words, this phase is where we "polish" the overall layout of the pathway.

In what follows we present our layout algorithm in a bottom–up fashion.

3.2. Layout algorithm

We assume that the pathway graph object $G = (V, E)$ to be laid out is implemented using an adjacency list representation and can be referenced through structures named *Graph*, *Node*, and *Edge*. Layout specific data and functionality are assumed to be kept in these structures. In the following pseudo code, location of a biological node u is represented with $L(u)$. In addition, cumulative spring forces F_s, due to incident edges on u are denoted by $F_s(u)$. Similarly, cumulative repulsion forces F_r, acting on u due to other nodes in a certain geometric neighborhood of u are denoted by $F_r(u)$.

The following method is used for calculating the relativity constraint forces, F_{rc}, acting over an edge. These forces are based on the conventions of pathway drawings. For instance, a product of a transition is drawn to its right; thus the product is applied a force proportional to the distance by which it is to the left of the associated transition. These forces are calculated per edge and reflected on its end nodes as follows:

Algorithm. ApplyRCF(*Edge* $e = (u, v)$)

```
(1) if adaptive layout enabled and in major phase 2 then
(2)     $e.maturity += 1$
(3)     $orientDiscrepancy := |e.assignedOrientation - e.orientation|$
(4)     if $e.maturity \geqslant MATURITY\_THRESHOLD$ and
        $orientDiscrepancy \geqslant MAX\_ORIENT\_DISCREPANCY$ then
(5)         Change orientation of $e$ as appropriate
(6)         $e.maturity := 0$
(7) Calculate $F_{rc}$ on $e$ according to its orientation
(8) $F_s(u) += F_{rc}$
(9) $F_s(v) -= F_{rc}$
```

We have an $\Theta(1)$ time processing of each edge's current orientation. If the edge cannot satisfy its current orientation, then we assign a new orientation to it based on its current position vector. The method is clearly of $\Theta(1)$ time complexity.

The next method calculates the general spring forces acting on each edge. The formula for calculating the spring forces acting on an edge is

$$F_s = (\lambda - edgeLength)^2/\eta,$$

where λ is the ideal edge length and η is the elasticity constant of the edge:

Algorithm. ApplySpringForces(*Graph* $G = (V, E)$)

```
(1) for $e = (u, v) \in E$ do
(2)     Calculate the spring force $F_s$ acting on $e$
```

```
(3)     $F_s(u) += F_s$
(4)     $F_s(v) -= F_s$
(5)     call APPLYRCF($e$)
```

The overall time complexity of this method is $\Theta(|E|)$ as all steps inside the for-loop can be processed in $\Theta(1)$ steps.

Node-to-node repulsion forces are calculated using the formula

$$F_r = \alpha/(d_x^2 + d_y^2),$$

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. ApplyRepulsionForces(*Graph* $G = (V, E)$)

```
(1) Create empty set $S$ of layout nodes
(2) for $u \in V$ do
(3)     Insert $u$ into $S$
(4)     for $v \in V - S$ do
(5)         if $dist(u, v)$ in $repulsionRange$ then
(6)             Calculate repulsion force $F_r$ acting on $u$ and $v$
(7)             $F_r(u) += F_r$
(8)             $F_r(v) -= F_r$
```

Steps 6–8 are handled in $\Theta(1)$ steps, which are executed a total of maximum $O(|V|^2)$ times, making the overall complexity of the method $O(|V|^2)$. 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 than this.

The CHECKCOMPRULES method is called to control the compartment constraints as well as updating node coordinates before the next layout step:

Algorithm. CheckCompRules(*Graph* $G = (V, E)$)

```
(1) for $u \in V$ do
(2)         $L(u) += F_r(u)$
(3)         if $u$ is a $state$ and $u$ violates bounds and resizing is enabled then
(4)             Expand compartment of $u$
(5)         $L(u) += F_s(u)$
(6)         if $u$ is a $state$ and $u$ violates bounds then
(7)             Alter $L(u)$ to keep $u$ within borders
(8)         Increment $error$ by total displacement of $u$
```

Either compartment resizing is enabled and the compartment borders are allowed to expand to create enough space for the node displacements or the node movements are "trimmed" to fit in the current compartment bounds. Notice that expansion and shrinkage (which is performed periodically during LAYOUT) of a compartment both affect the geometry of the contents of the neighboring compartments. For instance, if the cytoplasm expands from its bottom, both nucleus membrane and nucleus located below it will shift towards bottom along with its contents.

Step 4 is a bit complicated and possibly requires displacement of all nodes in the graph, thus takes $O(|V|)$ time to complete. Note that the compartments are usually resized no more than once or twice per iteration. This is because, although multiple nodes may request a resize operation, once the first resize is realized, the new compartment bounds will suffice for the following resize requests. Therefore, we can expect the overall time complexity of the method to be $O(|V|^2)$ in the worst case and $O(|V|)$ on the average.

The main method making use of earlier ones to implement the layout algorithm is as follows:

Algorithm. Layout()

```
(1)  step: = 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)     error: = 0
(10)    call APPLYSPRINGFORCES()
(11)    if in an odd minor phase or third major phase then
(12)       call APPLYREPULSIONFORCES()
(13)    call CHECKCOMPRULES()
(14)    if in third major phase and resizing is enabled and step mod shrinkPe-
        riod = 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 end layout
(20)    step: = 1
```

Let us analyze each phase independently. The first and second major phases only differ in the amount of repulsion range considered when calling APPLYREPULSIONFORCES. For the first major phase, a very high repulsion range is used, thus more node pairs are considered in these calculations. Minor phases indeed affect the time complexity more than the major phases, since in even-numbered minor phases the repulsion force calculations are disabled. Thus, 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 complexity 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. The shrink operation is similar to the expand operation and handled in $O(|V|)$ time, making the overall complexity of the third major step $O(|V|^2)$ for sparse graphs. Since there are multiple phases and we may skip the remaining iterations of a phase upon achieving the *errorThreshold* value, it is very difficult to make an average case complexity analysis for the algorithm. However, for 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 complexity of one layout iteration is $O(|V|^2)$. Overall this yields a worst-case time complexity of $O(K \cdot |V|^2)$ over a total of K iterations needed for minimizing the total energy of the system.

4. Implementation

The algorithm described above has been implemented within the PATIKA pathway editor [8]. The development environment was Sun's Java SDK 1.3 and Microsoft Windows XP operating system on an ordinary personal computer.

The theoretical analysis of the algorithms has shown that the overall complexity of the algorithm is $K \cdot O(|V|^2)$ over a total of K iterations. Thus, a quadratic behavior of execution time vs. number of nodes was expected, assuming K does not grow in the order of the graph size. It should be noted that in various modes of the layout (adaptive layout mode, compartment resizing mode, incremental mode, etc.) the number of iterations required to finalize the layout differs significantly. The tests presented here are executed in adaptive mode with compartment resizing enabled. The algorithm converges a lot quicker in incremental mode for relatively minor changes in the topology and/or geometry of an already laid out graph.

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. Each case includes five different test runs with 25, 50, 100, 150, and 200 nodes on the average. The average of results obtained are presented in Table 1. The execution times have been measured for each major component of the layout independently to discover their individual effects. Moreover, the total time of the

Table 1
Test results (ASF: apply spring forces, ARF: apply repulsion forces, CCR: check compartment rules)

$\|V\|$	Iter	Max Iter	ASF (ms)	ARF (ms)	CCR (ms)	Combined (ms)	Total (ms)
25	6948	7500	99	56	600	756	869
50	7415	7500	155	233	1205	1594	1725
100	7462	7500	265	903	2423	3592	3784
150	7467	7500	463	1882	4026	6372	6674
200	7500	7500	474	3500	5351	9326	9694

layout process is compared with the total contribution of these components to observe constant workload on smaller components (such as shrinking the compartments, which is done at certain intervals, independent of the number of nodes) of the algorithm.

Fig. 4 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 (due to the number of edges being a constant multiple of nodes in test graphs).

In theoretical analysis, we have stated that the APPLYREPULSIONFORCES method is quadratic with respect to the number of nodes. Our test results support our claim and reveals the quadratic $O(|V|^2)$ behavior of this component.

The CHECKCOMPRULES method was stated to have a linear contribution to the theoretical time complexity with respect to the number of nodes in the

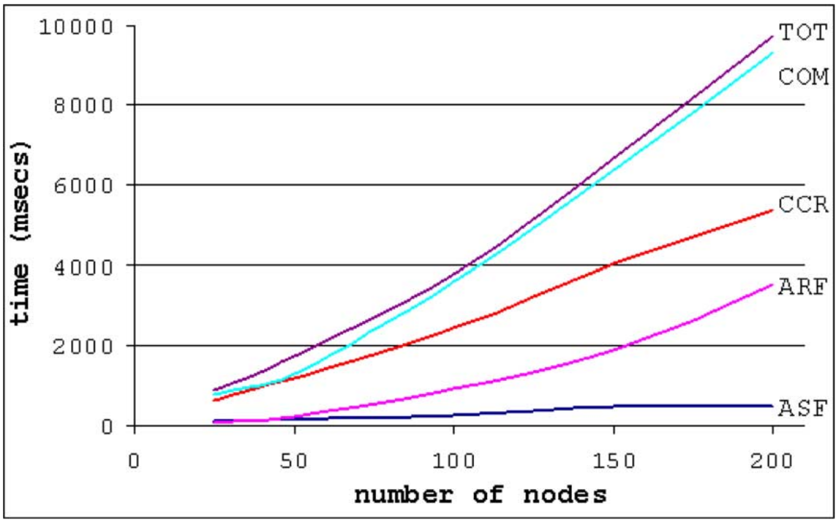
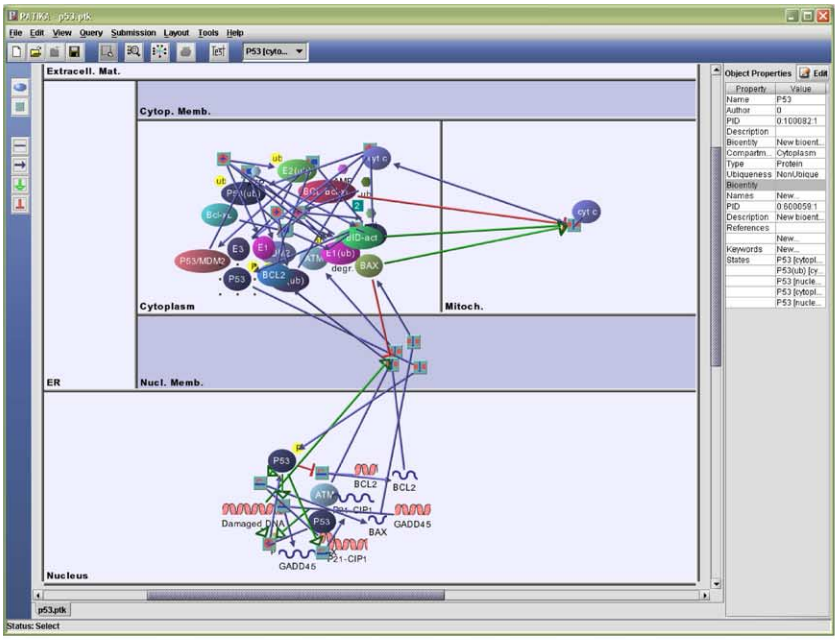
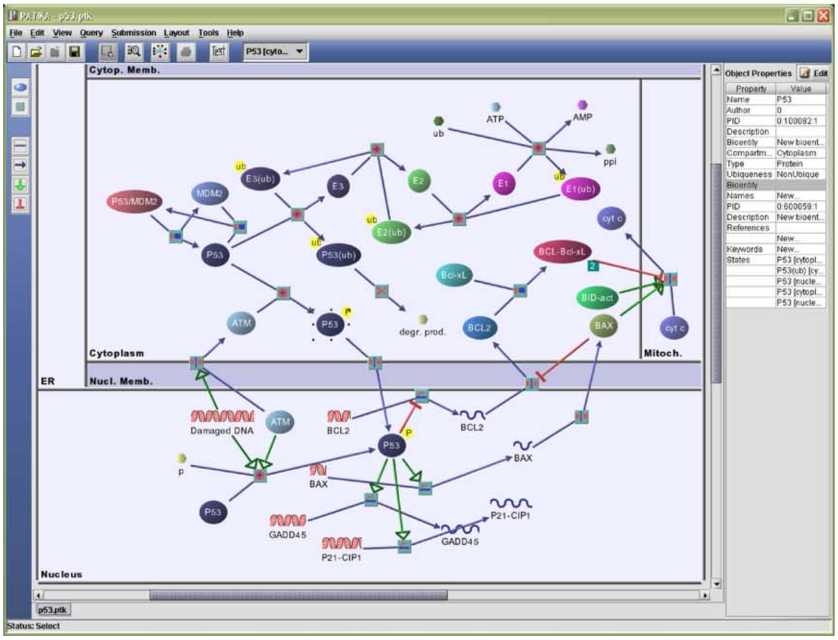


Fig. 4. Graph size vs. execution time. (For colour see online version.)



Property	Value
Name	P53
Author	0
PID	0:100082:1
Description	
Bioentity	New bioent...
Compartm...	Cytoplasm
Type	Protein
Ubiqueness	NonUbique
Bioentity	
Names	New...
PID	0:600059:1
Description	New bioent...
References	
	New...
Keywords	New...
States	P53 [cytopl...
	P53(ub) [cy...
	P53 [nucle...
	P53 [cytopl...
	P53 [nucle...



Property	Value
Name	P53
Author	0
PID	0:100082:1
Description	
Bioentity	New bioent...
Compartm...	Cytoplasm
Type	Protein
Ubiqueness	NonUbique
Bioentity	
Names	New...
PID	0:600059:1
Description	New bioent...
References	
	New...
Keywords	New...
States	P53 [cytopl...
	P53(ub) [cy...
	P53 [nucle...
	P53 [cytopl...
	P53 [nucle...

Fig. 5. A randomly laid out p53 pathway in PATIKA. (top); same pathway after our layout algorithm executes (bottom). (For colour see online version.)

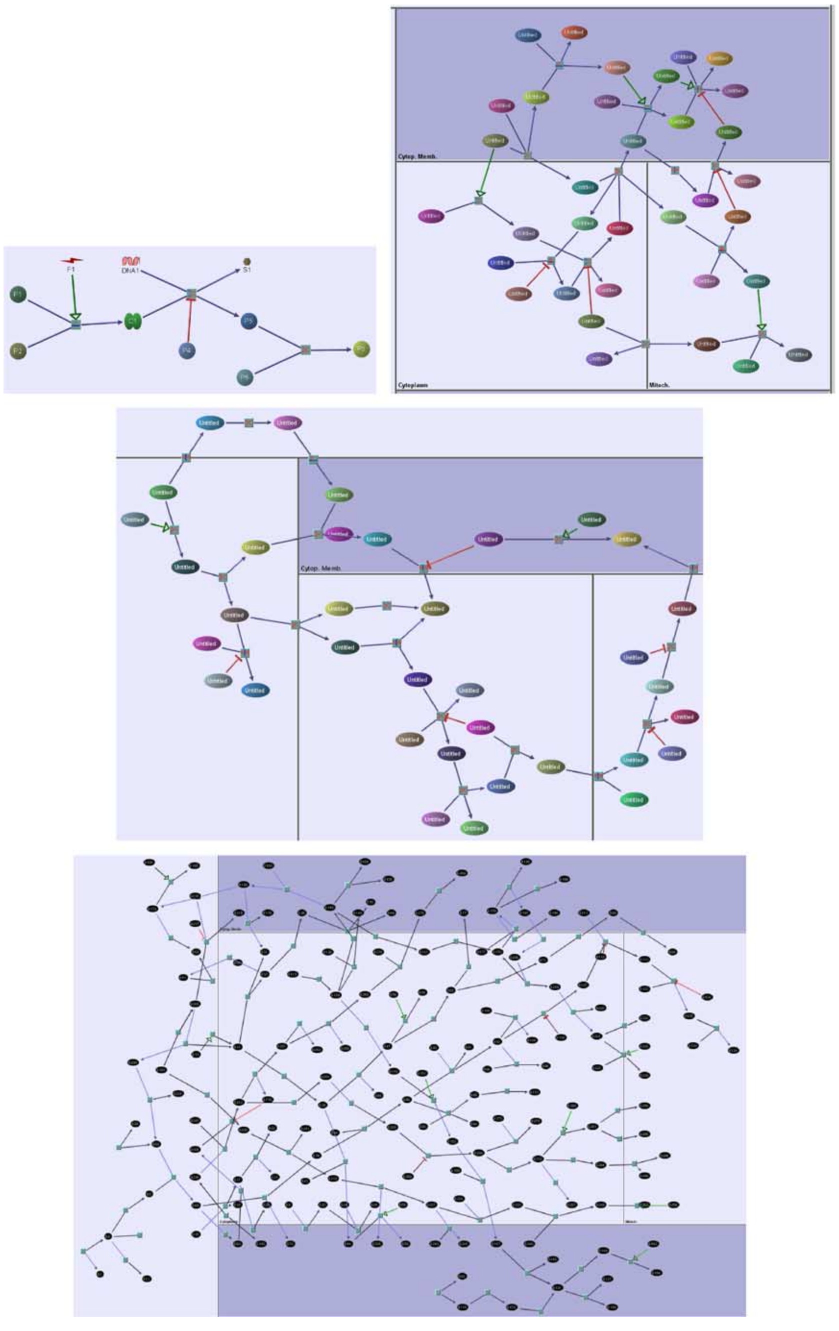


Fig. 6. Layouts of randomly generated pathway graphs of varying size and complexity. (For colour see online version.)

graph. The slight deviation from linearity observed is due to the fact that some graphs may have a larger number of nodes close to the compartment boundaries than the others, which results in more compartment resizing operations.

The total time complexity is seen to be quadratic as expected from theoretical analysis. Another important observation is that the compartment constraints are the major burden in the layout for small graphs, while as the graph size increases over a few hundred nodes, the calculation of repulsion forces dominate the execution time of the layout process. Overall, since most pathway analysis is done with small graphs of at most a hundred nodes or so, the performance of the algorithm is easily within acceptable range.

In addition, the quality of the layout is found to be acceptable in terms of general graph drawing criteria (e.g., discovering symmetries, generating plane drawings of planar graphs, and minimizing node-to-node overlaps and edge crossings) as well as pathway graph drawing conventions. Fig. 5 shows the layout of a p53 pathway within the PATIKA editor while Fig. 6 shows some randomly generated graphs laid out using our algorithm.

5. Conclusion

We have presented a new efficient algorithm for layout of biological signaling pathways, the underlying graph structure of which can be arbitrarily non-uniform and complex. Our algorithm uses a force-directed layout scheme with physical constraints due to cell geometry and constraints imposed by drawing conventions. In addition, it is inherently incremental as force-directed approaches normally are and keeps the relative positions of pathway objects when relatively minor changes are made in the topology and/or geometry of a pathway. The algorithm has been successfully implemented as part of a pathway integration and analysis toolkit named PATIKA. Finally, the algorithm may be easily adapted to be used in other applications with similar conventions and constraints as well.

References

[1] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Graph Drawing, Algorithms for the Visualization of Graphs, Prentice-Hall, 1999.

[2] U. Dogrusoz, Q. Feng, B. Madden, M. Doorley, A. Frick, Graph visualization toolkits, IEEE Computer Graphics and Applications 22 (1) (2002) 30–37.

[3] M. Arnone, E. Davidson, The hardwiring of development: organization and function of genomic regulatory systems, Development 124 (10) (1997) 1851–1864.

[4] G. Miklos, G. Rubin, The role of the genome project in determining gene function: insights from model organisms, Cell 86 (4) (1996) 521–529.

[5] H. Ogata, S. Goto, K. Sato, W. Fujibuchi, H. Bono, M. Kanehisa, KEGG: Kyoto Encyclopedia of Genes and Genomes, Nucleid Acids Research 27 (1999) 29–34, Available from <http://www.genome.ad.jp/kegg/>.

[6] P. Karp, M. Riley, M. Saier, I. Paulsen, J. Collado-Vides, S. Paley, A. Pellegrini-Toole, C. Bonavides, S. Gama-Castro, The EcoCyc database, Nucleic Acids Research 30 (1) (2002) 56–58.

[7] WIT, What is there? Interactive Metabolic Reconstruction on the Web, Available from <http://wit.mcs.anl.gov> 2001.

[8] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, M. Ozturk, PATIKA: An integrated visual environment for collaborative construction and analysis of cellular pathways, Bioinformatics 18 (7) (2002) 996–1003.

[9] E. Wingender, X. Chen, The TRANSFAC system on gene expression regulation, Nucleic Acids Research 29 (1) (2001) 281–283.

[10] T. Takai-Igarashi, T. Kaminuma, A pathway finding system for the cell signaling networks database, In Silico Biology 1 (1999) 129–146.

[11] T. Kamps, J. Kleinz, Constraint-based spring-model algorithm for graph layout, in: F. Brandenburg (Ed.), Graph Drawing (Proc. GD '95), Lecture Notes in Computer Science, vol. 1027, Springer-Verlag, 1995, pp. 349–360.

[12] T. Lin, P. Eades, Integration of declarative and algorithmic approaches for layout creation, in: R. Tamassia, I. Tollis (Eds.), Graph Drawing (Proc. GD '94), Lecture Notes in Computer Science, vol. 894, Springer-Verlag, 1995, pp. 376–387.

[13] X. Wang, I. Miyamoto, Generating customized layouts, in: F. Brandenburg (Ed.), Graph Drawing (Proc. GD '95), Lecture Notes in Computer Science, vol. 1027, Springer-Verlag, 1995, pp. 504–515.

[14] W. He, K. Marriott, Constrained graph layout, in: S. North (Ed.), Graph Drawing (Proc. GD '96), Lecture Notes in Computer Science, vol. 1190, Springer-Verlag, 1997, pp. 217–232.

[15] P.D. Karp, S. Paley, Automated drawing of metabolic pathways, in: Third International Conference on Bioinformatics and Genome Research, Tallahassee, FL, 1994, pp. 225–238.

[16] M.Y. Becker, I. Rojas, A graph layout algorithm for drawing metabolic pathways, Bioinformatics 17 (2001) 461–467.

[17] F. Schreiber, High quality visualization of biochemical pathways in BioPath, In Silico Biology 2 (2) (2002) 59–73.

[18] K. Sugiyama, S. Tagawa, M. Toda, Methods for visual understanding of hierarchical systems, IEEE Transactions on Systems, Man, and Cybernetics 21 (2) (1981) 109–125.

[19] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, A. Ayaz, G. Gulesir, G. Nisanci, R. Cetin-Atalay, An ontology for collaborative construction and analysis of cellular pathways, Bioinformatics 20 (3) (2004) 349–356.

[20] T.M.J. Fruchterman, E.M. Reingold, Graph drawing by force-directed placement, Software Practice and Experience 21 (11) (1991) 1129–1164.

[21] K. Sugiyama, K. Misue, A simle and unified method for drawing graphs: Magnetic-spring algorithm, in: R. Tamassia, I. Tollis (Eds.), Graph Drawing (Proc. GD '94), Lecture Notes in Computer Science, vol. 894, Springer-Verlag, 1995, pp. 364–375.