BIOINFORMATICS

Vol. 19 no. 15 2003, pages 1882–1888
DOI: 10.1093/bioinformatics/btg346

A fast layout algorithm for protein interaction networks

Kyungsook Han and Byong-Hyon Ju*

School of Computer Science & Engineering, Inha University, Inchon 402-751, Korea

Received on May 1, 2003; revised and accepted on July 15, 2003

ABSTRACT

Motivation: Graph drawing algorithms are often used for visualizing relational information, but a naive implementation of a graph drawing algorithm encounters real difficulties when drawing large-scale graphs such as protein interaction networks.

Results: We have developed a new, extremely fast layout algorithm for visualizing large-scale protein interaction networks in the three-dimensional space. The algorithm (1) first finds a layout of connected components of an entire network, (2) finds a global layout of nodes with respect to pivot nodes within a connected component and (3) refines the local layout of each connected component by first relocating midnodes with respect to their cutvertices and direct neighbors of the cutvertices and then by relocating all nodes with respect to their neighbors within distance 2. Advantages of this algorithm over classical graph drawing methods include: (1) it is an order of magnitude faster, (2) it can directly visualize data from protein interaction databases and (3) it provides several abstraction and comparison operations for effectively analyzing large-scale protein interaction networks.

Availability: http://wilab.inha.ac.kr/interviewer/

Contact: khan@inha.ac.kr

INTRODUCTION

While traditional biochemical experiments had generated a small set of data for individual protein–protein interactions, the last three years have seen a rapid expansion of protein interaction data due to the recent development of high-throughput interaction detection methods such as yeast two-hybrid (Ito *et al.*, 2000) and mass spectrometry techniques. The interaction data is available either in text files or in databases. However, due to the volume of data, a graphical representation of protein interactions has proven to be much easier to understand than a long list of interacting proteins. Furthermore, a network of protein interactions provides us with a clear notion of protein function by showing a context within which function can be interpreted.

Protein–protein interactions are typically visualized as an undirected graph $G = (V, E)$, where $x, y \in V$ represent proteins and $(x, y) \in E$ represents an interaction between proteins x and y. Visualization of a graph is straightforward when dealing with a small number of nodes and edges. In practice, protein–protein interaction networks often consist of thousands of nodes or more, which severely limit the usefulness of many graph drawing tools either because they produce cluttered drawings with many edge crossings or static drawings that are not easy to modify, they are too slow for interactive analysis with large data sets, or because they require input data to be in specific format rather than taking the data directly from protein–protein interaction databases. The ultimate usefulness of a protein interaction network depends on the readability of the network, and therefore, a protein interaction network should focus on conveying the interaction information quickly and clearly.

Force-directed layout algorithms have been the most popular methods for visualizing an undirected graph, which produce an optimal layout based on a force model. A simple implementation of a force-directed algorithm encounters real difficulties when drawing graphs of more than a few hundred nodes. These difficulties originate from two sources. First, layout adjustment involves computation of force between every pair of nodes at each step of the optimization process. Second, for large graphs the optimization process needs too many iterations for transforming the initial random layout into an optimal layout.

Previously we developed a force-directed layout program called InterViewer (Ju *et al.*, 2003). In this paper, we present a new program called InterViewer3 that efficiently produces a protein interaction network of good quality without computing force between every pair of nodes. InterViewer3 improves on InterViewer in many ways: (1) while InterViewer produces a drawing by computing force between every pair of nodes in each iteration of the optimization process, InterViewer3 produces a more pleasant drawing without computing force between every pair of nodes, (2) InterViewer3 is faster than InterViewer, (3) InterViewer3 provides several abstraction operations to reduce complex networks into simpler ones and (4) multiple protein interaction networks can be compared for common proteins and their interactions shared by all or part of the networks. The rest of this paper describes a set of algorithms of InterViewer3 and its results.

*To whom correspondence should be addressed

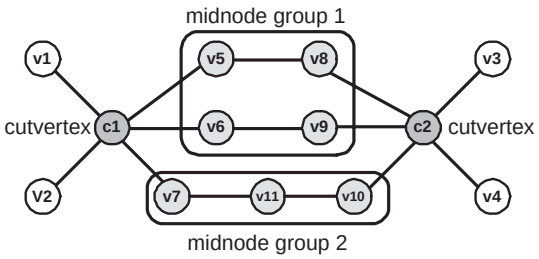


Fig. 1. Midnodes ($v5$–$v11$) on three paths between a pair of enclosing cutvertices ($c1$ and $c2$). Since the multiple paths have different lengths, midnodes on the paths are grouped into two groups: midnode group 1 = {$v5$, $v6$, $v8$, $v9$}, midnode group 2 = {$v7$, $v10$, $v11$}.

DEFINITIONS

The *degree* of a node v is the number of its edges and is denoted by $\deg(v)$. A *cutvertex* (also called an *articulation point*) in a graph G is a node whose removal disconnects G. A *path* in a graph G is a sequence $(v_1, v_2, \ldots, v_n)$ of distinct nodes of G, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq n-1$. A graph $G' = (V', E')$, such that $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$, is a *subgraph* of graph $G = (V, E)$.

When multiple paths exist between a pair of cutvertices, we call the nodes on the paths *midnodes*. In Figure 1, there are midnodes (shown in yellow) on three paths between a pair of enclosing cutvertices (shown in blue). If the multiple paths between a pair of cutvertices have different lengths, midnodes on the paths of same length are grouped together.

What we call *pivot nodes* are the key nodes in the layout of a graph. In order to produce a layout of high quality efficiently, we select pivot nodes that are almost uniformly distributed in each connected component (see Fig. 2 for examples). The number of pivot nodes and distance between them are determined based on the number of nodes and edges, and the diameter of a connected component (a diameter of a connected component is the maximum distance between two nodes in the component). In general, more pivot nodes are selected for a connected component with a large diameter compared to the number of nodes than for a connected component with a small diameter compared to the number of nodes. For a small connected component with 100 nodes or fewer, we select more pivot nodes so that the distance between them may be 3 or less. However, each connected component can have at most 100 pivot nodes in any case for the efficiency of the algorithm. A detailed method for selecting pivot nodes and for computing the distance between them is described in Algorithms 2 and 3 later.

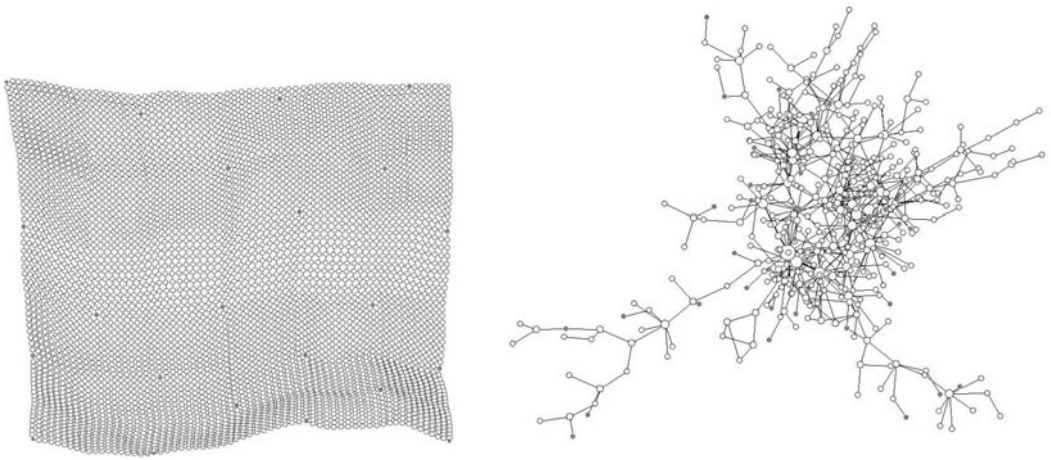
THE ALGORITHM

A common problem with many force-directed layout algorithms is that they become very slow when dealing with large graphs because layout adjustment at each step typically involves computation of force between every pair of nodes. Since a protein interaction network tends to be a disconnected graph with several connected components, we first compute a layout of connected components and then compute a layout of nodes within a connected component. Our experience is that this approach produces much better drawings in a shorter time than computing a layout of all nodes from the beginning.

Our algorithm uses a multilevel technique to draw a graph. It is composed of two steps at the top level: grouping and layout. In the grouping step, the algorithm first groups nodes of a disconnected graph into connected components, and finds midnodes and pivot nodes in each connected component. In the layout step, the coarsest graph is an initial layout of connected components based on their pivot nodes only. The layout of each connected component is then refined locally within the connected component based on its midnodes and neighbors of each node. Each step of the algorithm can be summarized as follows.

1. Grouping
   (a) Identify all connected components of an entire network.
   (b) For each connected component, determine its midnodes and pivot nodes.
   (c) Compute the distance of every node from the pivot nodes of the connected component to which the node belongs.
2. Layout
   (a) Find a layout of connected components of an entire network (layout between connected components).
   (b) For each connected component find a layout of nodes with respect to the pivot nodes of the connected component (global layout within a connected component).
   (c) Refine the layout of each connected component by relocating the midnodes adjacent to cutvertices with respect to the cutvertices and the cutvertices' direct neighbors (local layout of midnodes within a connected component).
   (d) Refine the layout of each connected component by relocating all nodes with respect to their neighbors within distance 2 (local layout of all nodes within a connected component).

Step 1(a) is straightforward, and Algorithm 1 describes step 1(b). In Algorithm 1, a *group* represents a connected component. Since step 1(a) and Algorithm 1 are performed on nodes with at least one edge, nodes with no edge are positioned after the connected components of size $\geq$2 are positioned in step 2(a). For a graph with $|V| = n$ nodes, the time complexity of step 1(a) is $O(n)$, and the time complexity of Algorithm 1 is $O(n \cdot |PvN|)$, where $|PvN|$ is the number of pivot nodes.



(a) (b)

Fig. 2. (**a**) Pivot nodes (shown in green) selected from a mesh. (**b**) Pivot nodes (shown in green) selected from a protein interaction network.

Algorithm 1 Distance(v, w)

```
1:  DLst.Add(v, 0)
                {Add v and its distance (= 0) from v to DLst}
2:  DLst.First                          {Get the first node of DLst}
3:  repeat
4:    DLst.GetCurrent(v', currentDist)
          {Get the current node v' and its distance from v}
5:    for all neighbor u of v' do
6:      if u ∉ DLst then
7:        if w = u then
8:          return currentDist+1
                                  {distance between v and u}
9:        end if
10:       DLst.Add(u, currentDist+1)
                    {Add u and its distance from v to DLst}
11:     end if
12:   end for
13:   DLst.Next                         {Get the next node of DLst}
14: until DLst.Eof          {until no more nodes exist in DLst}
```

Algorithm 2 SelectPivotNodes

```
1:  MaxDist ← 1
2:  PvN.Add(V[0], DistTable.Create(V[0], 0))
                                        {first node in a group}
3:  PvN.First                       {Get the first node of PvN}
4:  repeat
5:    DLst.Clear          {Initialize DLst as an empty list}
6:    DLst.Add(PvN.CurrentPivotNode, 0)
          {Add the current pivot node and its distance}
7:    DLst.First                    {Get the first node of DLst}
8:    repeat {distance from pivot nodes}
9:      ChkDistance(DLst, PvN.CurrentDistTable,
        MaxDist)
10:     DLst.Next                   {Get the next node of DLst}
11:   until DLst.Eof    {until no more nodes exist in DLst}
12:   PvN.Next                      {Get the next node of PvN}
13: until PvN.Eof       {until no more nodes exist in PvN}
```

Selecting pivot nodes from each connected component in step 1(c) is done by Algorithms 2 and 3. When selecting pivot nodes, distances of the pivot nodes from all other nodes are also computed. Algorithms 2 and 3 take $O(n)$ time for a single pivot node, and therefore, the total time complexity for selecting all pivot nodes is $O(|PvN| \cdot n)$. Algorithm 3 examines whether the current node v is already a pivot node; if not, it determines the possibility of including the node to the pivot node set PvN depending on the distance from existing pivot nodes, the structure of the connected component (i.e. diameter, number of nodes and edges of the connected component). The current node v can be selected as a pivot node if it satisfies the following rules (function ChkPvN(v)in step 16 of Algorithm 3).

1. In a connected component with <40 nodes, the distance of v from all existing pivot nodes should be at least 2.
2. In a connected component with ≥ 40 and <100 nodes, the distance of v from all existing pivot nodes should be at least 3.
3. In a connected component with ≥ 100 nodes,
   (a) if the diameter (d) of the connected component is <7, degree(v) should be ≥ 3.
   (b) if $7 \leq d < 15$, degree(v) should be ≥ 4.
   (c) if $15 \leq d < 20$, degree(v) should be ≥ 5.

Algorithm 3 ChkDistance(DLst, DistTable, MaxDist)

```
1:  DLst.GetCurrent(v, dist)
        {Get a node v and its distance from a pivot node}
2:  if (dist > MaxDist) then
3:     MaxDist ← dist      {Update the maximum distance}
4:  end if
5:  bAddPvN ← true                  {potential pivot node}
6:  for all neighbor w of v  do
7:     if w ∈ DLst then {distance of w from a pivot node has
       not been determined.}
8:        bAddPvN ← false      {w cannot be a pivot node}
9:        DLst.Add(w, dist+1)
                {Add w and its distance from v to DLst}
10:       DistTable(w) ← dist+1
              {Store the distance of w from v in DistTable}
11:    end if
12: end for
13: if MaxDist/3 = dist then {The node is at a distance of
    one third of the maximum distance}
14:    bAddPvN ← true               {potential pivot node}
15: end if
16: if bAddPvN and ChkPvN(v) then
17:    PvN.Add(v, DistTable'.Create(v, 0))
18: end if
```

Algorithm 4 Layout(v, V')

```
1: D ← 0      {Initialize the position displacement D to 0}
2: for all u ∈ V' do {V': subset of V}
3:    Δ ← pos[u] − pos[v]
                          {pos[u]: position of node u}
4:    D ← D + Δ(1 − Distance(u, v)/‖Δ‖)
                          {‖Δ‖: norm of a vector Δ}
5: end for
6: D ← D/|V'|              {|V'|: number of nodes in V'}
7: pos[v] ← pos[v] + D
                {Update the position of v by adding D.}
```

(d) else, let R be the ratio of the diameter of the connected component to the number of nodes of the connected component.

(i) if $R < 0.01$, the distance of v from all existing pivot nodes should be at least 40.

(ii) if $0.01 \leq R < 0.02$, the distance of v from all existing pivot nodes should be at least 17. If the total number of nodes >1000, adjust the distance to 30.

(iii) if $0.02 \leq R < 0.035$, the distance of v from all existing pivot nodes should be at least 13. If the total number of nodes > 1000, adjust the distance to 20.

(iv) if $0.035 \leq R < 0.07$, the distance of v from all existing pivot nodes should be at least 10.

(v) if $R \geq 0.07$, the distance of v from all existing pivot nodes should be at least 5.

Algorithm 4 provides a concise description of all layouts of step 2, including both global layout and local layout. The position of v is always determined with respect to a reference set V', which is a subset of V. In step 2(a), the reference set V' is a set of pivot nodes of *other* connected components, to which v does not belong. The maximum diameter of all connected components is used as the value of Distance(u, v) in step 4 of Algorithm 4, and therefore is constant for all nodes. In step 2(b), the reference set V' is a set of pivot nodes of the connected component to which v belongs. The value of Distance(u, v) is available in the distance table, which was already computed by Algorithm 3 for each pivot node. Steps 2(b) and 2(c) are repeated until the maximum edge length of the connected component $\leq$ a threshold value.

In step 2(c), the reference set V' of v is a set of its enclosing cutvertices and the cutvertices' direct neighbors, and v is a midnode that is directly adjacent to a cutvertex. The distance between a midnode and any node of its reference set is computed by simple arithmetic. Suppose that node $v5$ of Figure 1 is to be relocated in step 2(c) and that the path length between its enclosing cutvertices be p. The reference set V' of node $v5$ becomes $\{c1, c2, v1 - v10\}$. Then, the distance from $v5$ to its near cutvertex $c1$ is 1, and that to $c1's$ neighbors $v1$, $v2$, $v6$, $v7$) is 2. The distance from $v5$ to its far cutvertex $c2$ is $p-1$, that from $v5$ to any of $v3$, $v4$ or $v10$ is p, and that from $v5$ to any of $v8$ or $v9$ is $p-2$. Therefore, the distance from a midnode to any node in its reference set is either 1, 2, path length ($= p$) of its enclosing cutvertices, $p-1$, or $p-2$.

In step 2(d), the reference set V' of a node v is the neighbors of v within distance of 2, and v is any node in the network. A single execution of Algorithm 4 takes O($|V'|$) time, so the total time complexity of steps 2(a)–2(c) is O($n \cdot |PvN|$), where $|PvN|$ is the number of pivot nodes. The worst time complexity of step 2(d) is O(n^2) since the number of a node's neighbors within a distance of 2 can be as large as O(n).

ABSTRACTION OF PROTEIN INTERACTION NETWORKS

A large number of edges and nodes of a complex protein interaction network often reduces the readability of the network due to cluttered edges and nodes. In general there are two ways to analyze such a complex network. One is to extract smaller subnetworks from the entire network and to analyze each of the subnetworks one by one. Another is to abstract the entire network into a simpler one. InterViewer3 can extract a subnetwork in several ways. For example, it can extract a subnetwork of proteins within specified interacting distance from one or more target proteins or a subnetwork of proteins

shared by several protein interaction networks. For abstraction of a network, InterViewer3 provides the following operations. An abstract network can be expanded to a detailed network on demand.

1. *Collapse a clique into a star-shaped subgraph.* A clique in an undirected graph $G = (V, E)$ is a subset of V, each pair of which is connected by an edge in E. Each clique is replaced by a star-shaped subgraph centered at a dummy node, which is shown as a circle in the abstract graph.
2. *Collapse a group of nodes with the same interactions into a composite node.* A group of nodes with the identical interacting partners are collapsed into a single composite node, which is shown as a diamond in the abstract graph.

A clique with n nodes contains $n(n-1)/2$ edges, and a star-shaped graph for the clique contains exactly n edges. Therefore, replacing the cliques with star-shaped graphs substantially reduces the number of edges. Finding a clique with a maximum size in a graph is a NP-hard problem (Battiti and Protasi, 2001). We have developed an efficient, heuristic algorithm that identifies all *edge-disjoint* cliques (i.e. cliques that do not share an edge) (Ju and Han, 2003). While collapsing the cliques into a star-shaped graph only reduces the number of edges, collapsing the nodes with the same interactions into a composite node reduces the number of nodes as well as the number of edges.

RESULTS

The layout algorithms and abstraction operations were implemented in Borland Delphi 6.0, and databases of protein–protein interactions were constructed using Microsoft Data Access Components 2.7. InterViewer3 is executable on any PC with Windows 2000/XP/Me/98/NT 4.0 as its operating system, and available at http://wilab.inha.ac.kr/interviewer/.

InterViewer3 takes the input interaction data in several formats: (1) data from a Microsoft Access database, (2) graph modeling language (GML) format (Himsolt, 1997) http://www.uni-passau.de/Graphlet/GML, (3) a pair of interacting protein names, separated by a space or tab, in each line and (4) a pair of interacting protein indices, separated by a tab, in each line. As output, InterViewer3 produces two types of drawings (bitmap and GML file). The protein interaction data can also be saved in an ASCII file in any of the input formats described above. Figure 3 shows a network with 44387 interactions between 4242 human proteins. It appears to have edge crossings, but it actually contains no edge crossing when it is visualized as a three-dimensional drawing on a video monitor. The program allows the user to explore three-dimensional drawings by rotating or by zooming in or out of them.

For a graph with many cliques, the first collapsing operation alone is very effective in reducing the complexity of a graph. For a graph with few cliques, applying both collapsing operations is more effective in reducing the complexity. Collapsing a group of nodes with same interacting partners into a composite node is also effective to simplify dense subgraphs at the terminal (see supplementary figures in http://wilab.inha.ac.kr/intrviewer).

InterViewer3 provides two ways of comparing multiple protein interaction networks. One is to find a subnetwork shared by *all* networks being compared and the other is to find a subnetwork shared by *part* of networks by coloring the networks. Figure 4 shows an example of comparing three networks, each originally represented in cyan, yellow and magenta, by the second comparison method. Proteins shared by part of the networks are represented in mixed color of the corresponding networks.

Comparison of speed

For the purpose of comparing actual running times of our algorithm with others, we ran two other graph-drawing programs, Pajek (Batagelj and Mrvar, 2001) and Tulip (David, 2001). Table 1 shows the running times of five layout algorithms on a same set of test cases: the new algorithm of InterViewer3, Kamada and Kawai's layout (Kamada and Kawai, 1989) of Pajek, Frunchterman-Reingold's layout (Fruchterman and Reingold, 1991) of Pajek, GEM layout of Pajek, and Spring-Electric force layout of Tulip. Pajek with Kamada and Kawai's layout algorithm could not even visualize the human map 2 data due to 'out of memory' error. The new algorithm of InterViewer3 visualizes a network with thousands of nodes and edges in tens of seconds. It follows from this comparison that InterViewer3 is an order of magnitude faster than recent implementations of force-directed layout algorithms.

CONCLUSIONS

From the perspective of graph drawing, protein interaction data is a major bioinformatics challenge, because (1) it yields a large and complicated graph with excessive number of edge crossings, (2) it produces a disconnected graph with many connected components, (3) the graph contains nodes of wide range of degrees. Many force-directed graph drawing algorithms are too slow to be used in visualizing large-scale protein interactions and they often yield unclear drawings with many edge crossings. This paper presented a new algorithm and a program called InterViewer3 for drawing large-scale protein interaction networks in three-dimensional space. Unique features of InterViewer3 include: (1) its drawing algorithm is much faster than other forced-directed drawing algorithms (a network with thousands of nodes and edges is drawn in tens of seconds), (2) it can be used not only for visualizing protein interactions but also for finding and exploring individual connected components or subgraphs interactively, which we found very useful in studying protein–protein interactions on a large scale and (3) it provides an

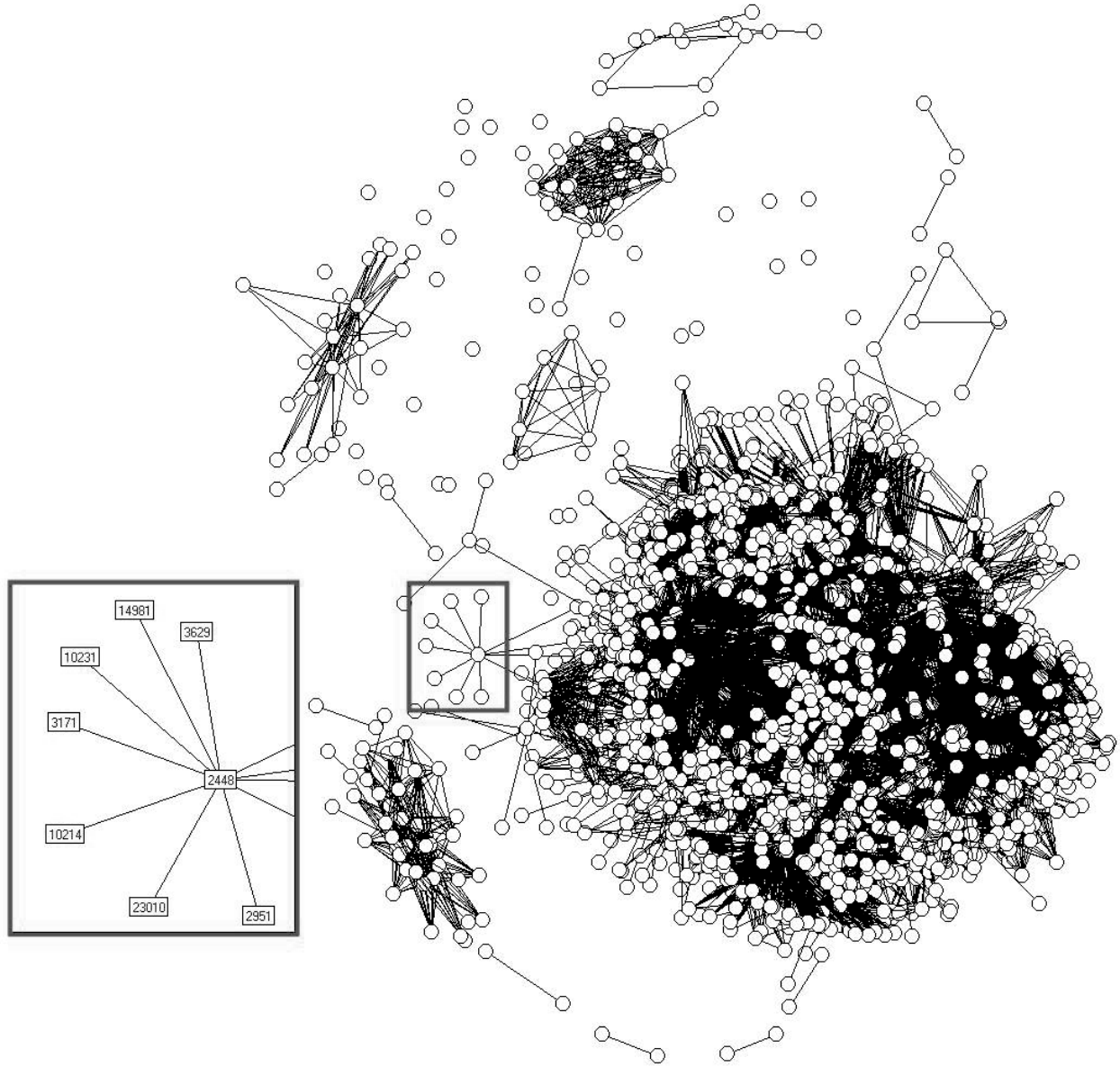


Fig. 3. Human protein interaction network with 44387 interactions between 4242 proteins. The network enclosed in a red box is a blowup of the marked subnetwork.

Table 1. Running times of the graph drawing programs on the five data sets on a Pentium IV 2.0 GHz processor with 1GB memory. K-K: Kamada–Kawai's layout, F-R: Frunchterman-Reingold's layout, S-E: Spring–Electric force layout, human map 1 & 2: human protein interaction data

Program (layout algorithm)	Y2H data (3751 nodes, 12917 edges)	BIND data (4048 nodes, 8286 edges)	DIP data (4690 nodes, 14460 edges)	Human map 1 (8654 nodes, 184407 edges)	Human map 2 (12056 nodes, 6989558 edges)
InterViewer3	7 s	6 s	7 s	19 s	8 min 20 s
Pajek (K-K)	2 min 31 s	1 min 37 s	3 min 04 s	56 min 38 s	Out of memory
Pajek (F-R)	28 min 23 s	20 min 02 s	42 min 45 s	2 h 28 min 40 s	5 h 43 min 32 s
Tulip (GEM)	2 min 10 s	4 min 40 s	18 min 40 s	9 h 19 min 10 s	$\gg$10 h
Tulip (S-E)	24 min 35 s	35 min 47 s	56 min 45 s	$\gg$10 h	$\gg$10 h

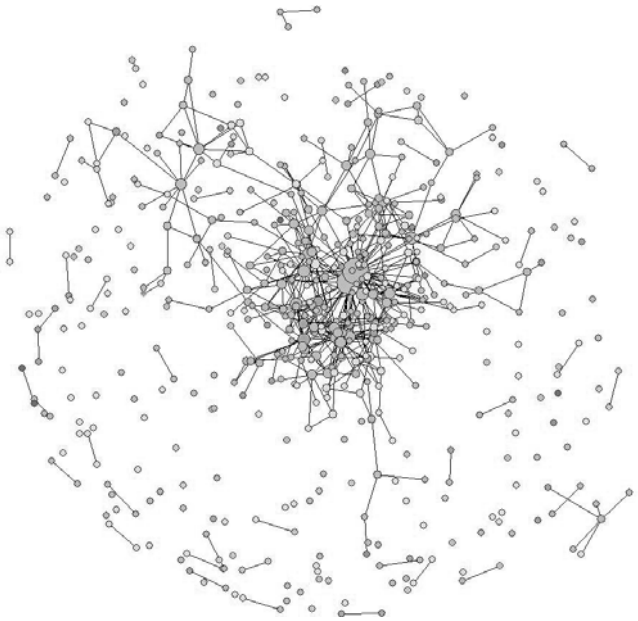


Fig. 4. Comparing three networks, each originally represented in cyan, yellow and magenta. Proteins shared by part of the networks are represented in mixed color of the corresponding networks. See the example section at http://wilab.inha.ac.kr/interviewer/ for a color figure.

integrated framework for querying protein–protein interaction databases and directly visualizing the query results, making the visualization and analysis of large amounts of updated data easy.

InterViewer3 is one of the most advanced and comprehensive visualization tools for studying protein interactions. Comparisons with other existing algorithms showed that InterViewer3 generates clear and aesthetically pleasing drawings of large-scale protein–protein interaction networks and that it is an order of magnitude faster than other force-directed algorithms. It is being extended to work as a web-based application program supporting both wired and wireless communication.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Information and Communication of Korea under grant number IMT2000-C3-4.

SUPPLEMENTARY DATA

For Supplementary data, please refer to *Bioinformatics* online.

REFERENCES

Batagelj,V. and Mrvar,A. (2001) Pajek – analysis and visualization of large networks. *Lecture Notes Comput. Sci.*, **2265**, 477–478.

Battiti,R. and Protasi,M. (2001) Reactive local search for the maximum clique problem. *Algorithmica*, **29**, 610–637.

David,A. (2001) Tulip. *Lecture Notes Comput. Sci.*, **2265**, 435–437.

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

Himsolt,M. (1997) GML: GraphModeling Language.

Ito,T., Tashiro,K., Muta,S., Ozawa,R., Chiba,T., Nishizawa,M., Yamamoto,K., Kuhara,S. and Sakaki,Y. (2000) Toward a protein–protein interaction map of the budding yeast: a comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins. *Proc. Natl Acad. Sci. USA*, **97**, 1143–1147.

Ju,-H.B. and Han,K. (2003) Complexity management in visualizing protein interaction networks. *Bioinformatics*, **19**, i177–i179.

Ju,-H.B., Park,B., Park,H.J. and Han,K. (2003) Visualization and analysis of protein interactions. *Bioinformatics*, **19**, 317–318.

Kamada,T. and Kawai,S. (1991) An algorithm for drawing general undirected graphs. *Inform. Process. Lett.*, **31**, 7–15.