Visualization of Protein-Protein Interaction Networks Using Force-Directed Layout*

Yanga Byun and Kyungsook Han**

School of Computer Science and Engineering
Inha University, Inchon 402-751, Korea
quaah@hanmail.net, khan@inha.ac.kr
http://wilab.inha.ac.kr/protein/

Abstract. Protein interactions, when visualized as an undirected graph, often yield a nonplanar, disconnected graph with nodes of wide range of degrees. Many graph-drawing programs are of limited use in visualizing protein interactions, either because they are too slow, or because they produce a cluttered drawing with many edge crossings or a static drawing that is not easy to modify to reflect changes in data. We have developed a new force-directed layout algorithm for drawing protein interactions in three-dimensional space. Our algorithm divides nodes into three groups based on their interacting properties: biconnected subgraph in the center, terminal nodes at the outermost region, and the rest in between them. Experimental results show that our algorithm efficiently generates a clear and aesthetically pleasing drawing of large-scale protein interaction networks and that it is much faster than other force-directed layouts.

1 Introduction

Recent improvements in high-throughput proteomics technology such as yeast two-hybrid [4, 11] have produced a rapidly expanding volume of protein interaction data of an unprecedented scale. The interaction data is available either in text files or in databases. However, due to the volume of data (e.g., thousands of interacting proteins), a graphical representation of protein-protein interactions has proven to be much easier to understand than a long list of interacting proteins, prompting visualization studies of protein-protein interaction networks.

A Java applet program [6] has been developed for drawing protein interactions based on a relaxation algorithm and tested on the yeast two-hybrid (Y2H) data [11]. This program requires all protein-protein interaction data to be provided as parameters of the applet program in html sources. There is no way to save a visualized graph except by capturing the window. An image captured from the window is a static image and is of a generally low quality. It cannot be refined or changed later to reflect an update in data. A user can move a node

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

** To whom correspondence should be addressed. email: khan@inha.ac.kr

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 190–199, 2003.
© Springer-Verlag Berlin Heidelberg 2003

but cannot select or save a connected component containing a specific protein for later use. Recent work on yeast proteome also utilized a relaxation algorithm to visualize a protein complex network [3].

Other visualization works on protein interactions do not have their own algorithms or programs developed for visualization, but use general-purpose drawing tools. PSIMAP [7], for example, displays interactions between protein families by comparing the Y2H data with the DIP data [12]. It was drawn by Tom Sawyer software (http://www.tomsawyer.com/) and then refined by significant amount of manual work to remove the edge crossings. From the perspective of graph drawing, PSIMAP is a static image and leaves several things to be improved. A research group of University of Washington [9, 10] has visualized the Y2H data using another general-purpose drawing tool called AGD (http://www.mpi-sb.mpg.de/AGD/). Although AGD is powerful, it is a general-purpose drawing tool and does not provide a function that we hold are necessary for studying protein-protein interactions.

Existing visualization studies in protein interactions suggest that the nature of protein interaction data require a new graph layout method for protein interaction networks. Protein interaction data can be characterized as follows:

1. The data yields a nonplanar graph with a large number of edge crossings that cannot be removed in a two-dimensional drawing.
2. Proteins have a very wide range of interacting proteins within the same set of data, so a graph visualizing the data contains nodes of very high degree as well as those of low degree.
3. When visualized as a graph, the data yields a disconnected graph with many connected components. The MIPS genetic interaction data (http://mips.gsf.de/proj/yeast/tables/interaction/), for example, contains 113 connected components.
4. The data often contains protein interactions corresponding to self-loops.

Considering these characteristics of protein interaction data, we have developed a new layout algorithm that divides nodes into three groups based on their interaction properties and layouts each group in three-dimensional space. The rest of this paper describes our algorithm and experimental results.

2 A Partitioned Approach to Graph Drawing

A common problem with many force-directed algorithms is that they become very slow when dealing with large graphs. We propose a new force-directed algorithm which divides nodes into three groups based on their interaction characteristics. Our layout is an extension of the algorithm by Kamada & Kawai [5] for drawing two-dimensional graphs. Their original algorithm has been modified not only for three-dimensional graph drawing but also for improvements in the efficiency and resulting drawings of the algorithm.

2.1 Finding Groups

Protein-protein interaction data can be visualized as an *undirected* graph $G = (V, E)$, where nodes V represent proteins and edges E represent protein-protein interactions. The *degree* of a node v_i is the number of its edges denoted by $deg(v_i)$. An edge $e = (v_i, v_j)$ with $v_i = v_j$ is a *self-loop*. A *cutvertex* 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 $G = (V, E)$. We divide nodes V into three exclusive and exhaustive groups: V_1, V_2, V_3. The three groups are defined as follows:

- Group V_1 is a set of terminal nodes, i.e., nodes with degree 1.
- Group V_2 is a set of nodes in V-V1, which are in the subgraphs separated by cutvertices of degree ≥ 3, except the nodes in the largest subgraph separated by the cutvertices.
- Group V_3 consists of nodes which are members of neither V_1 nor V_2.

Example 1 Consider a graph $G = (V, E)$ displayed in Fig. 1. The nodes in G are separated into three groups. Six nodes belong to V_1 and these are separated into three sub-groups, $V_1 = \{\{v_1\}, \{v_5, v_9, v_{10}\}, \{v_{31}, v_{32}\}\}$. Each of the three sub-groups shares a neighbor. □

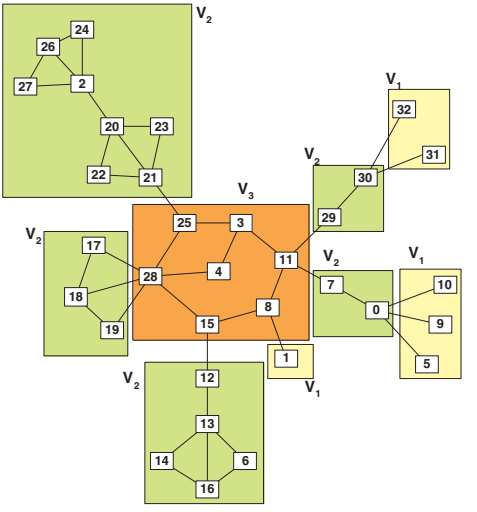

Fig. 1. Example of a partitioned graph. The nodes of V_1 are enclosed in yellow boxes, V_2 in green boxes, and V_3 in an orange box.

Example 2 In Fig. 1, two sub-groups $S_1 = \{v_0, v_7\}$ and $S_2 = \{v_{29}, v_{30}\}$ share a cutvertex v_{11}, so they are merged into one sub-group of V_2. Sub-groups $S_3 =$

$\{v_{24}, v_{26}, v_{27}\}$ and $S_4 = \{v_2, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{26}, v_{27}\}$ do not share a cutvertex since the cutvertex of S_3 is v_2 and that of S_4 is v_{25}. However, S_3 is not counted as a sub-group of V_2 since $S_3 \subset S_4$. □

Algorithm 1 FindCutvertex

```
1: for all v_i such that degree(v_i > 2) do
2:    inputSet = V − V_1 − v_i
3:    while P = IsCutvertex(inputSet) do
4:       inputSet = inputSet -P
5:    end while
6: end for
```

Algorithm 2 IsCutvertex(givenSet)

```
1: Randomly select a starting node v_s ∈ givenSet.
2: Insert v_s to a stack S.
3: while there is a top node v_t in S do
4:    Popup v_t and assign v_t to a current node v_c.
5:    if v_c is not marked then
6:       Mark v_c and insert v_c to P_i {P_i: a set of nodes in the path between v_s and v_i}
7:       Insert neighbors v_h of v_c to S, s.t. v_h are not marked.
8:    end if
9: end while
10: Insert nodes not in P_i to P'_i.
11: if (|P_i| > 0) and (|P'_i| > 0) then
12:    if |P_i| ≤ |P'_i| then
13:       Insert v ∈ P_i to V_2 and Return P_i.
14:    else
15:       Insert v ∈ P'_i to V_2 and Return P'_i.
16:    end if
17: end if
18: Return false
```

Nodes of each group are found in the order of V_1, V_2, and V_3. First, nodes with one neighbor are classified into V_1. Nodes of V_1 are further divided into sub-groups according to their shared neighbors. From $V - V_1$, nodes in V_2 are then found, and all remaining nodes constitute V_3.

After finding V_1, nodes of V_2 are determined by our heuristic algorithm *FindCutvertex* outlined in Algorithm 1. The initial input to the algorithm is nodes in $V - V_1$. For each input node v_i, the algorithm tests whether the node is a cutvertex (line 3 of Algorithm 1). Let P be the set of nodes in the path between v_i and the starting node and P' be the set of nodes not in the path. If neither of P and P' is empty, the node v_i is a cutvertex and the loop is repeated for the remaining nodes. The nodes in the smaller set between P and P' are included

in V_2 (lines 11-17 of Algorithm 2). The nodes of V_2 are further separated into sub-groups based on their cutvertex. These sub-groups are merged into one if they have the same cutvertex. All remaining nodes after determining both V_1 and V_2 constitute V_3. V_3 corresponds to a *biconnected* subgraph (a connected graph with no cutvertices) in protein interaction data.

2.2 Force-Directed Layout for Three-Dimensional Graph Drawing

The algorithm by Kamada & Kawai [5] searches for a drawing in which the energy is locally minimal. The aim of our algorithm is to find a drawing in which the actual distance between two nodes is approximately proportional to the desirable distance between them. The global energy of a spring system with n nodes is defined by the equation:

$$\begin{aligned} E &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} (|p_i - p_j| - l_{ij})^2 \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \Big\{ (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \\ &\quad + {l_{ij}}^2 - 2 l_{ij} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \Big\} \end{aligned} \tag{1}$$

where k_{ij} is the *stiffness* parameter of a spring, p_i is the position of a node v_i, and l_{ij} is the length at rest of the spring connecting v_i and v_j.

The algorithm seeks a position $p_m = (x_m, y_m, z_m)$ for each vertex v_m to minimize the potential energy in the spring system. Minima occur when the partial derivatives of E with respect to each variables x_m, y_m, and z_m are zero. This gives a set of $3|V| = 3n$ equations

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = \frac{\partial E}{\partial z_m} = 0, \ v_m \in V \tag{2}$$

In Kamada & Kawai's algorithm [5], a node is moved to a position that minimizes energy while all others remain fixed. The node to be moved is chosen as the one with the largest force acting on it, i.e., the one for which

$$\sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2 + \left(\frac{\partial E}{\partial z_m}\right)^2} \tag{3}$$

is maximized over all $v_m \in V$. However, this approach often yields unpleasant graphs and takes too much time for large-scale protein interactions. Thus, our algorithm moves all nodes to some level in each iteration until the difference between the current position and the previous position falls below a certain threshold value. For an initial layout we place nodes on the surface of a sphere instead of placing them randomly. Our algorithm yields more pleasant drawings than Kamada & Kawai's algorithm and is much faster for graphs with balanced groups.

Algorithm 3 ShortestPath

```
1: Compute adjacency matrix A[a, b] in G, for 1 ≤ a < b ≤ n.
2: Initialize shortest-path matrix S[a, b] in G, for 1 ≤ a < b ≤ n.
3: for all V_i, i = 3, 2, 1 do
4:    if V_3 then
5:       Call FindShortestPath(V_3,null)                          {in Algorithm 4}
6:    else if V_2 then
7:       for all each sub-group G_s do
8:          Call FindShortestPath(G_s ∪ v_c, null)        {v_c: shared cutvertex of G_s}
9:          Call FindShortestPath(G_s ∪ V_3, v_c)
                                  {compute S[p, q] not defined for p ≠ q ∈ G_s ∪ V_3}
10:      end for
11:   else if V_1 then
12:      for all each sub-group G_s do
13:         Call FindShortestPath(G_s ∪ v_h, null)         {v_h: shared neighbor of G_s}
14:         Call FindShortestPath(G_s ∪ V_3 ∪ V_2, v_h)
                               {compute S[p, q] not defined for p ≠ q ∈ G_s ∪ V_3 ∪ V_2}
15:      end for
16:   end if
17: end for
```

Algorithm 4 FindShortestPath (N, givenK)

```
1: for all k such that k ∈ N or givenK do
2:    for all l such that l ∈ N do
3:       S[k, l] = A[k, l] (S[k, l] = 2 if V_i = V_1)        {S[k, l]: shortest-path matrix}
4:       for all m such that m ∈ N do
5:          if S[l, m] > S[l, k] + S[k, m], for (m ≠ k)&&(m ≠ l) then
6:             S[l, m] = S[l, k] + S[k, m]
7:          end if
8:       end for
9:    end for
10: end for
```

2.3 Finding Shortest Paths in Groups

The shortest path between every pair of nodes is computed for each group $V_i, i = 1, \ldots, 3$. The algorithm for computing shortest paths is summarized in Algorithms 3 and 4. We first compute shortest paths between nodes in V_3. For V_2 and V_1, shortest paths are determined in each of their sub-groups. After computing shortest paths between nodes in each sub-group, shortest paths between nodes of V_2 and nodes of V_3 are computed using a shared cutvertex of each sub-group of V_2 (line 9 of Algorithm 3). Likewise, shortest paths between nodes of V_1 and nodes of V_2 and V_3 are computed using a shared neighbor of each sub-group of V_1 (line 14). For a sub-group of V_1, the initial shortest path between every pair of nodes is set to 2, since the distance between a node and its shared neighbor is 1 (line 3 of Algorithm 4).

3 Analytical and Empirical Evaluation

Here we briefly analyze the computational cost of our algorithm. Assuming that three groups are balanced, the total time for our algorithm is $(\frac{n}{3})^3 + (\frac{n}{3})^3 + (\frac{n}{3})^3 = \frac{n^3}{9}$ because we apply spring-embedder algorithm on each group. The asymptotic time complexity of our algorithm is the same as the time complexity $O(n^3)$ of Kamada & Kawai's algorithm [5]. But our algorithm is much faster Kamada & Kawai's algorithm in practice. Since the nodes in V_1 and V_2 are further divided into sub-groups, actual running times are reduced further for graphs with balanced groups. For a graph with unbalanced groups (for example, a graph in which the portion of V_3 is high due to few cutvertices or terminal nodes), the effect of dividing into three groups can be marginal, which is rare in protein interaction data. This has been supported by the experimental study to be discussed later.

Our algorithm was implemented in Microsoft C#. The program runs on any PC with Windows 2000/XP/Me/98/NT 4.0 as its operating system. We tested the program on five cases, the PSIMAP data [7], Helicobacter pylori data [8], Y2H data [11], and the data of protein interactions in yeast from the BIND (http://www.binddb.org/) and DIP (http://dip.doe-mbi.ucla.edu/) databases. In each of these data of protein interactions, the largest connected component was used for comparison. Table 1 shows the running times of our algorithm at each stage of partitioning nodes into three groups, finding shortest paths in all three groups, and layout and drawing.

Table 1. Running times of our algorithm at each stage. Hpylori: helicobacter pylori, P: partitioning nodes into V_1, V_2 and V_3, SP: finding shortest paths in all groups, LD: layout and drawing.

Data	Edges	Nodes			Running times			
		V_1	V_2	V_3	P (min: sec)	SP (min: sec)	LD (min: sec)	total (= $P + SP + LD$)
PSIMAP	661	187	83	171	00:00.06	00:00.04	00:02.02	00:02.12
Hpylori	1396	434	9	267	00:00.31	00:00.20	00:07.36	00:07.87
Y2H	12909	3066	28	642	00:26.78	00:34.38	09:12.25	10:13.41
BIND	8243	3039	45	893	00:15.67	00:26.02	08:21.82	09:03.51
DIP	14415	3278	126	1195	00:30.00	00:43.45	10:41.64	11:55.09

Fig. 2A shows an initial layout by our algorithm for the Y2H data with 12909 interactions between 3736 proteins. While we find groups in the order of V_1, V_2, and V_3, we layout them in reverse order; V_3 is first positioned in the center of the sphere, V_2 in the outer region of V_3, and V_1 in the outer region of V_2 and V_3. Groups for which node positions are fixed are shown in a rectangle. Nodes in remaining groups are relocated with modified polar coordinates in order to place them in the outer region of the groups that have been fixed. In Figs 2B

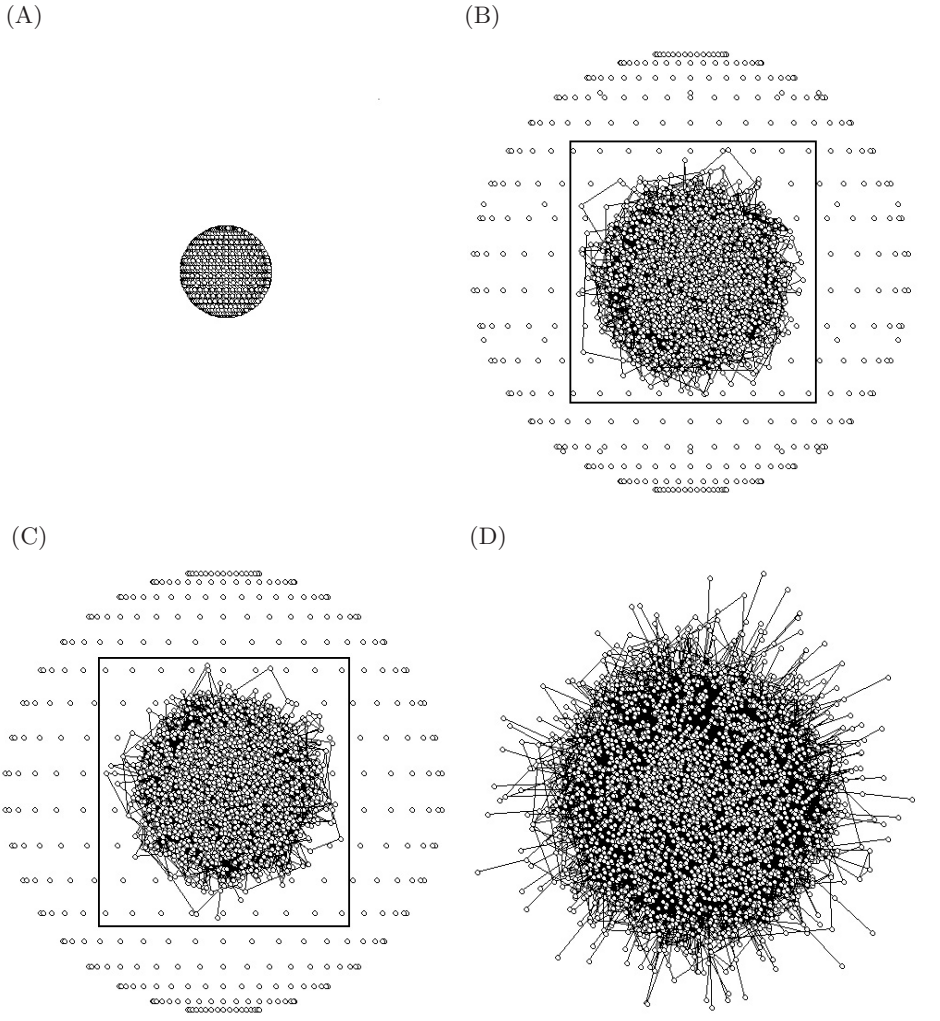

Fig. 2. Drawings of the Y2H data with 12909 interactions between 3736 proteins. (A) Initial layout, (B) After drawing the nodes in V_3 shown in the rectangle, (C) After drawing the nodes in V_3 and V_2 shown in the rectangle, (D) Final drawing.

and 2C, edges between nodes in the outer area are not shown for the clarity of the drawing.

For the purpose of experimental comparison with other algorithms, we also ran Pajek [1] with Fruchterman & Reingold's algorithm [2] and our own implementation of Kamada & Kawai's algorithm [5]. Since Kamada & Kawai's algorithm produces a two-dimensional drawing only, we extended their algorithm to a three-dimensional drawing. Table 2 shows the running times of our algorithm, Kamada & Kawai's algorithm extended to 3D, and Fruchterman & Reingold's algorithm on five test cases on a Pentium IV 1.5Ghz processor. With our partitioning method, the computation time can be significantly reduced. The running times of the three algorithms are also plotted in Fig. 3. It follows from this result that our algorithm is more effective for bigger graphs and for graphs with balanced groups.

Table 2. Running times of graph drawing algorithms on 5 test cases on a Pentium IV 1.5Ghz processor. K-K extended to 3D: Kamada & Kawai's algorithm extended for 3D drawing. Pajek (F-R): Pajek with Fruchterman & Reingold's layout.

Data	Nodes	Edges	Our algorithm	K-K extended to 3D	Pajek (F-R)
PSIMAP	441	661	00m 02.12s	00m 07.69s	00m 33s
Hpylori	710	1396	00m 07.87s	00m 22.14s	01m 14s
Y2H	3736	12909	10m 13.41s	13m 23.47s	32m 10s
BIND	3977	8243	09m 03.51s	14m 24.39s	41m 37s
DIP	4599	14415	11m 55.09s	19m 09.81s	55m 55s

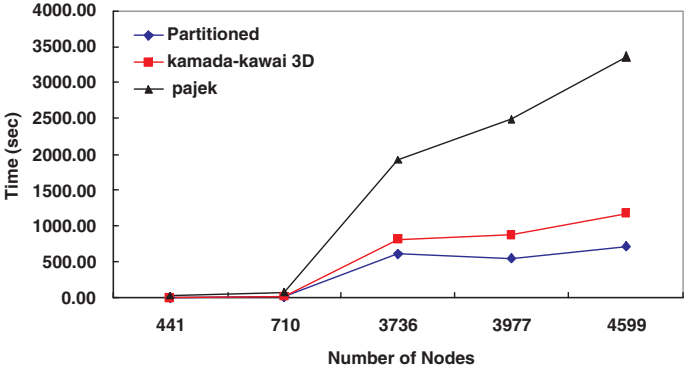

Fig. 3. Running times of three graph drawing algorithms.

4 Discussion and Conclusions

Many force-directed graph drawing algorithms are too slow to be used for visualizing large-scale protein interactions and they often yield unclear drawings with many edge crossings. This paper presented a new algorithm for drawing large-scale protein interaction networks in three-dimensional space. The algorithm divides nodes into three groups: a biconnected subgraph in the center of a graph (V_3), terminal nodes at the outermost region (V_1), and the rest in between them (V_2). These groups are identified from the outer group (V_1) to inner one (V_3) and are placed in reverse order. Experimental results demonstrate that the algorithm generates clear and aesthetically pleasing drawings of both large and small scale graphs and that for graphs with balanced groups it is significantly faster than other force-directed algorithms.

References

[1] V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. *Lecture Notes in Computer Science*, 2265:477–478, 2001.

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

[3] A.-C Gavin, M. Bosche, R. Krause, and et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415(6868):141–147, 2002.

[4] T. Ito, K. Tashiro, S. Muta, R. Ozawa, T. Chiba, M. Nishizawa, K. Yamamoto, S. Kuhara, and Y. Sakaki. Toward a protein-protein interaction network 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(3):1143–1147, 2000.

[5] T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Information Processing Letters*, 31:7–15, 1989.

[6] R. Mrowka. A java applet for visualizing protein-protein interaction. *Bioinformatics*, 17(7):669–670, 2001.

[7] J. Park, M. Lappe, and S.A. Teichmann. Mapping protein family interactions: intramolecular and intermolecular protein family interaction repertoires in the PDB and Yeast. *J. Mol. Biol.*, 307:929–938, 2001.

[8] J.-C. Rain, L. Selig, H.D. Reuse, V. Battaglia, C. Reverdy, S. Simon, G. Lenzen, F. Petel, J. Wojcik, V. Schachter, Y. Chemama, A. Labigne, and P. Legrain. The protein-protein interaction map of helicobacter pylori. *Nature*, 409:211–215, 2001.

[9] B. Schwikowski, P. Uetz, and S. Fields. A network of protein-protein interactions in yeast. *Nature Biotechnology*, 18(12):1257–1261, 2000.

[10] C.L. Tucker, J.F. Gera, and P. Uetz. Towards an understanding of complex protein networks. *Trends in Cell Biology*, 11(3):102–106, 2001.

[11] P. Uetz, L. Giot, G. Cagney, and et al. A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. *Nature*, 403:623–627, 2000.

[12] I. Xenarios, E. Fernandez, L. Salwinski, X.J. Duan, M.J. Thompson, E.M. Marcotte, and D. Eisenberg. DIP: the database of interacting proteins: 2001 update. *Nucleic Acids Res.*, 29(1):239–241, 2001.