Cerebral: Visualizing Multiple Experimental Conditions on a Graph with Biological Context

by

Aaron Barsky

B.Math., The University of Waterloo, 2000

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

(Vancouver)

June, 2008

Abstract

Systems biologists use interaction graphs to model the behaviour of biological systems at the molecular level. In an iterative process, such biologists observe the reactions of living cells under various experimental conditions, view the results in the context of the interaction graph, and then propose changes to the graph model. These graphs represent dynamic knowledge of the biological system being studied and evolve as new insight is gained from the experimental data. While numerous graph layout and drawing packages are available, these tools did not fully meet the needs of our immunologist collaborators. In this thesis, we describe the data display needs of these immunologists and translate these needs into visual encoding decisions.

These decisions led us to create Cerebral, a system that uses a biologically guided graph layout and incorporates experimental data directly into the graph display. Our graph layout algorithm uses simulated annealing with constraints, optimized with a uniform grid to have an expected runtime of $O(E\sqrt{V})$. Small multiple views of different experimental conditions and a measurement-driven parallel coordinates view enable correlations between experimental conditions to be analysed at the same time that the measurements are viewed in the graph context. This combination of coordinated views allows the biologist to view the data from many different perspectives simultaneously. To illustrate the typical analysis tasks performed, we analyse two datasets using Cerebral. Based on feedback from our collaborators, we conclude that Cerebral is a valuable tool for analysing experimental data in the context of an interaction graph model.

Contents

Appendices

List of Tables

List of Figures

Acknowledgements

I thank my supervisor Tamara Munzner, for finding a project that matched my passions and kept me enthused for years, for suggesting fruitful avenues of exploration, and quickly quashing awful ideas before I was buried in the mire. Thank you to all my colleagues at the Hancock Lab, particularly Jennifer Gardy, David Lynn, and Bob Hancock, for ideas, data sets, and enthusiastic excitement at the results, which made the whole project worthwhile. Thanks to Robert Kincaid for advice and for assembling the yeast cell cycle data set. Thanks to the whole InfoVis group, Stephen Ingram, Peter McLachlan, Dan Archambault, Heidi Lam, James Slack, and Ciaran Llachlan Leavitt, for the oft painful, but ultimately true and beneficial feedback.

Thank you to Agilent Technologies for a grant supporting this research.

And, most importantly, I would like to thank all my parents, Mom, Dad, and Donna, for their wonderful support.

Disclaimer

This thesis is based on material contained in the following papers

- Aaron Barsky, Tamara Munzner, Jennifer Gardy, and Robert Kincaid. Cerebral: Visualizing multiple condition gene expression data on a graph with biological context. Submitted for publication.
- Aaron Barsky, Jennifer Gardy, Robert Hancock, and Tamara Munzner. Cerebral: a Cytoscape plugin for layout of and interaction with biological networks using subcellular localization annotation. *Bioinformatics,* 23(8):1040-1042, 2007

Chapter 1

Introduction

Systems biology is a paradigm for biological experimentation in which researchers model biological systems by looking at the behaviour of the thousands of biological entities that influence each other, rather than single biomolecules or reactions. These interactions are modelled as a graph $G = (V, E)$, where the nodes V represent biomolecules such as proteins and genes, and the edges E join pairs of biomolecules that interact physically or chemically. This interaction graph model is used to interpret the results of experiments, and in turn experiments help biologists further refine the model.

Systems-level experimentation in cellular biology involves observing the response of cells to events by making large numbers of quantitative measurements. Examples of events are the introduction of a drug, the detection of a chemical signal from other cells, a change in environmental temperature, or the simple progression of time. A common way to observe the cell response is to measure the change in gene expression level, or abundance of proteins, in the cell across thousands of genes under a specific experimental condition. Interpreting these measurements in the context of an interaction model can help a biologist generate hypotheses about how the parts of the system influence each other.

We distinguish between two classes of interaction models available to biologists. Pathway diagrams are small directed graphs containing between ten and a few hundred nodes. These pathways show the interactions that comprise a specific biological event, such as a signalling pathway or a metabolic process. However, they provide a poor substrate for the hypothesis discovery process that drives systems biology. By limiting their representation to a small set of canonical biomolecules and interactions, the possibility of discovering new components of the process or interconnections with other

processes is eliminated. Thus, systems biologists often prefer to work with the second class of data: larger undirected graphs representing entire biological systems, which typically contain 500 to 3000 nodes and edges. Graph models such as these require more effort to interrogate, but ultimately yield more novel biological insights. By combining quantitative data and these large graph models, systems biologists have discovered new regulatory and signalling proteins, similarly-behaving cliques that are predictive for the metastasis of cancerous tumours, network modules that govern the aging process, and many other biological activities.

Information visualization can play an important role in the iterative model refinement process. In this domain, visualization is typically used for hypothesis generation, not hypothesis verification. Visually displaying the gathered quantitative measurements in the context of the graph model supports the hypothesis discovery process by allowing researchers to spot trends or subnetworks of interest. Testing a hypothesis about a biological process then takes months or years of slow and expensive lab work.

We began a collaboration with a group of systems biologists exploring the human immune response. The primary contribution of this thesis is the design of Cerebral, a new visualization tool that supports faster and richer hypothesis discovery for this group of immunologists. Further, Cerebral supports the more general system biology task of viewing data from multiple experimental conditions in the context of an interaction graph model. We embarked on an iterative design process to understand the visualization needs of these immunologists, which included interviews about their previous workflow and feedback from them on successive interactive prototypes.

While there are many graph layout and display systems that ably represent generic graphs [15, 18, 50], and even several aimed at biologists [47], none meet the targeted needs of our collaborators. We identified two visual requirements not met by existing exploration systems. First, the graph layout must use biological metadata to position nodes in biologically meaningful ways. A secondary contribution of this thesis is a new graph layout algorithm that incorporates biological metadata. The second requirement is that the system must be able to simultaneously represent data gathered

from multiple experiments, because the comparison between two or more experiments overlaid on a graph is a common analytical step.

In Chapter 2, we will discuss the workflow of our biologist collaborators, who used some existing visualization tools but undertook significant manual intervention to create the visual representations required for their tasks. Using these familiar data representations as a base, we describe our information design decisions and contrast alternative visual encodings in Chapter 3. After reviewing the related work in Chapter 4, we present Cerebral, a new system designed to meet our collaborators' need to interactively explore experimental data in the context of a graph model. Chapter 5 describes the Cerebral graph layout algorithm in detail, while Chapter 6 describes the interactive interface. In Chapter 7, we present the results of 2 sample analysis sessions with Cerebral. The first is an immunology-specific scenario to investigate the protective effects of a new therapeutic molecule, contrasting an earlier manual analysis of this data with the improved Cerebral analysis. We conclude with a more general example in which Cerebral is used to examine the passage of time in budding yeast, the first time that a visual approach has been used to explore this dataset.

Chapter 2

Immunology workflow

Our immunologist collaborators use a systems biology approach to investigate human responses to bacterial infection, as well as the effects of novel therapeutic compounds on these responses. Their ultimate goal is to understand and be able to predict host responses, as well as identify therapeutic compounds that modify the immune response. Therapeutic compounds could help to resolve bacterial infections while minimizing potentially harmful side effects of the immune response, such as inflammation and septic shock.

In a typical experiment, cells are divided into a control and a treatment group. The treatment group is given a candidate therapeutic compound and then both cell populations are exposed to a simulated bacterial infection. To see how each gene in the cell is responding to the infection, gene expression levels are measured using microarrays or other measurement technologies. Often a time series experiment is done, where an assay is performed at several time points to investigate how the immune response progresses.

The collected data is overlaid onto an interaction graph that models the immune response, derived from databases of known biomolecular interactions. Although tens of thousands of genes are typically measured, the immunologists do not usually work with an interaction graph model covering that entire dataset because of its overwhelming complexity. They instead consider simplified graphs of only the most interesting genes. The most simplified graphs contain only a few dozen genes directly involved in a specific biological process, while larger graphs show an entire biological system, such as the few thousand genes involved in immunity.

Undirected edges represent interacting proteins which propagate a signal from infection-detecting proteins at the surface to the nucleus of the cell. Di-

rected edges represent the binding of proteins to nuclear DNA, which either activates or represses the expression of genes in response to the infection signal. This ultimate response is of greatest interest to our collaborators, as it is the nature of the genes responding at this stage that determines whether the immune response will clear the infection without engaging any harmful inflammatory mechanisms.

By examining expression data in the context of the interaction graph, the immunologists can see how the immune system is responding to infection. The immunologists may develop a hypothesis about how the treatment is affecting the immune response by making comparisons between the control and the treatment conditions, and also between various time points. If the hypothesis is verified through more detailed lab work, the interaction graph will be modified to include interactions with the therapeutic compound.

As an example, we consider the results of an experiment previously published by our collaborators [37]. Human monocytes, a type of white blood cell associated with immunity, were stimulated with lipopolysaccharide (LPS), a molecule that can mimic the effect of bacterial infection. One batch of cells was treated with the candidate therapeutic compound LL-37, while one batch was left untreated. The gene expression level was measured using microarrays at four time points for each of these two conditions.

The figure showing this data in the published paper [37] is reprinted here as Figure 2.1. Although the biological graph viewer Cytoscape [45] was used to make Figure 2.1, creating this figure took several hours because significant manual intervention was required. Similar analyses using larger process graphs have taken days to construct.

In Figure 2.1, the left hand graph depicts the signalling cascade that begins with the detection of LPS by Toll-Like Receptor 4 (TLR4), proceeds through a series of intermediates, and ultimately ends in the regulation of several immune response genes. In order to create a layout reflecting the location within the cell of these biomolecules, as found in many textbooks and publications, the biologists positioned each node in the graph by hand. They thus used a very simple TLR4 graph model with only 66 nodes.

Figure 2.2 shows the same data displayed in our exploration tool Cere-

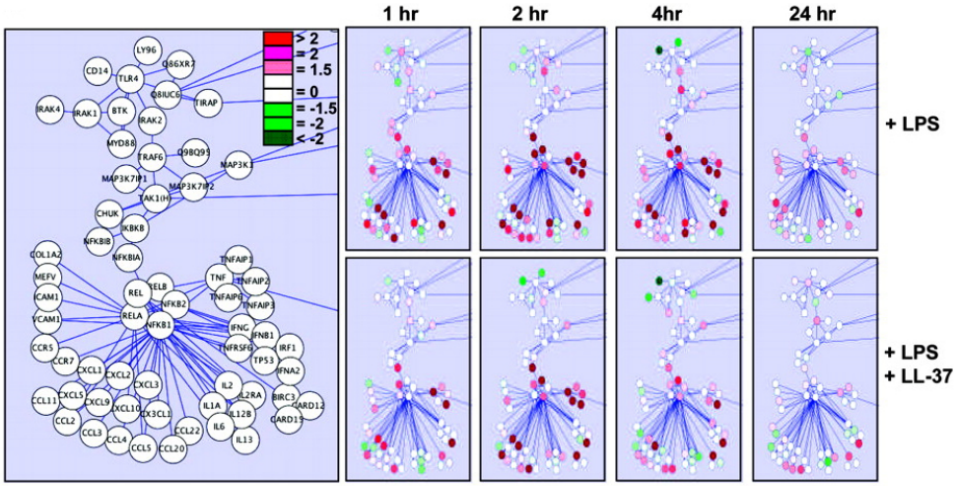


Figure 2.1: Original Cytoscape analysis with manually laid out graph and manually created small multiple views, from [37].

bral. Using a completely automatic algorithm, a larger TLR4 network model with 91 nodes is laid out in only a few seconds. Cerebral positions nodes in layers according to existing knowledge of where the biomolecules are found in a cell. The target nodes of directed protein-DNA edges are placed at the bottom of the diagram, in a layer representing immune response outcomes. The Cerebral layout groups these outcome nodes according to known biological function, enabling the biologists to easily categorize the nature of the immune response to the bacterial stimulus in the presence or absence of the therapeutic LL-37 compound.

Figure 2.1 also features small multiple [49] views, with each mini-graph coloured according to the expression level of a gene at a specific time point. According to biological tradition, genes whose expression was significantly increased are coloured red, while decreased expression levels are coloured green. As Cytoscape only loads a single experimental condition at a time, the multiple views were created one at a time by colouring the graph according to each of the eight experiments, taking a screenshot of each, and then assembling the results into a composite figure. In contrast, the small multiple views of Cerebral shown in Figure 2.2 fully support interactive exploration,

with linked navigation and brushing across all windows. The colouring in the main window was chosen by clicking in two of the small multiple windows to show a computed difference between those conditions.

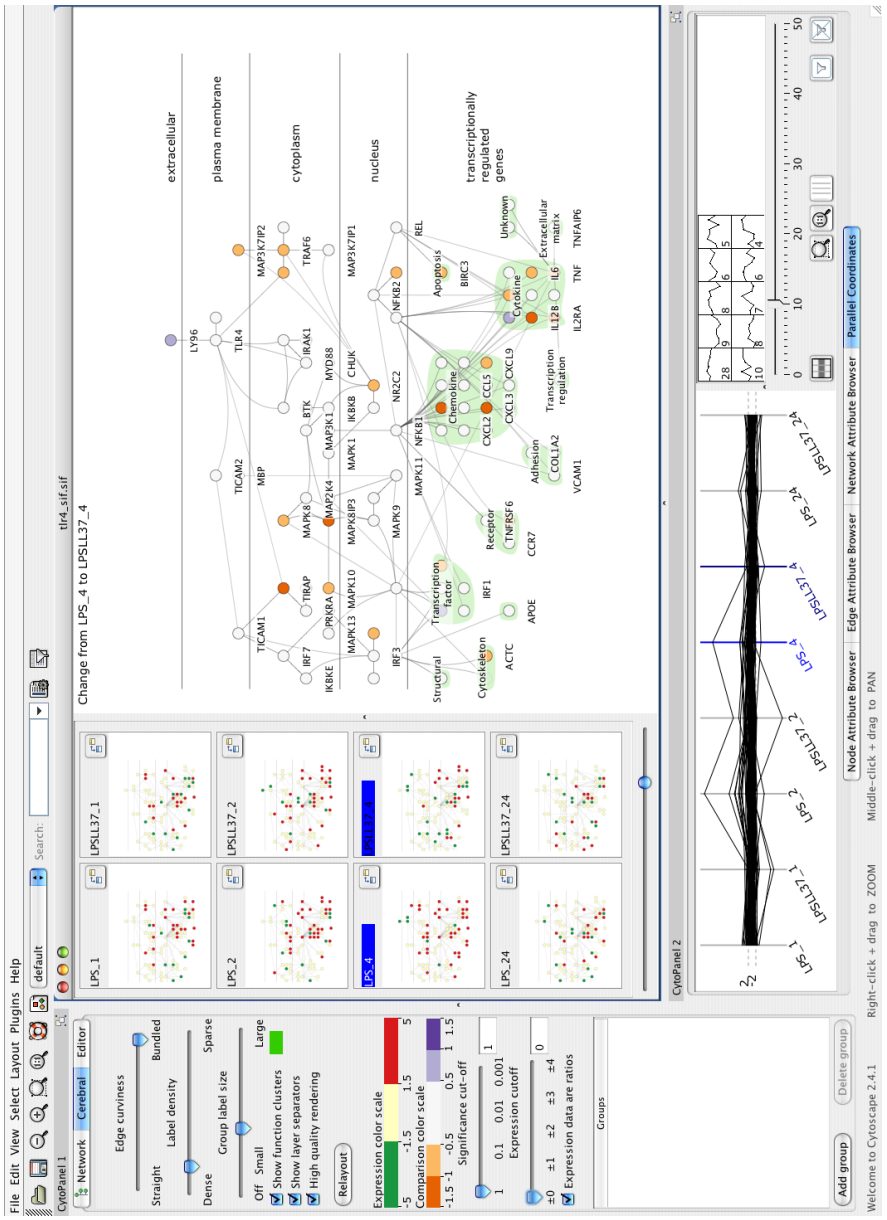


Figure 2.2: The Cerebral display of the TLR4 graph (V=91, E=124) with associated LPS experiments.

Chapter 3

Design decisions

When designing an automated system to display multiple experimental conditions in the context of an interaction model, we considered several alternate visual encodings. We now examine our design choices.

3.1 Graph layout with biological context

Graph layout quality has historically been evaluated with a number of heuristics such as edge length, crossings between nodes and edges, and uniformity of node distribution [40]. While many graph viewers [3, 7, 45] implement several layout algorithms that are favourably evaluated using these heuristics, none of these layout algorithms were acceptable to our immunologist collaborators when applied to their biological interaction graphs, as generic graph layout algorithms do not use biological metadata to guide the placement of nodes. Without biological metadata, layout algorithms often place connected nodes side-by-side in the layout, even though the corresponding proteins are physically distant in the cell. These layouts do not match the biology tradition of positioning proteins according to subcellular localization.

The nodes in biological graphs represent physical compounds in a cell that are separated by physical membranes, creating compartments defining their subcellular localization. Biologists typically depict biological processes with a stylised cross-sectional view of the cell according to this subcellular localization: nodes corresponding to the outermost membrane layer of a cell are placed at the top of the graph, nodes that are found in the innermost nucleus are placed at the bottom, and the remaining nodes are placed in the centre of the graph arranged neatly to show the step-wise series of in-

teractions that occur as a signal moves from membrane to nucleus. Layout algorithms that only use graph topology to position nodes would place some nodes near each other that are always positioned in separate compartments in hand drawn immunology diagrams, causing confusion and extra interpretation effort for the biologist. Cerebral restricts the placement of nodes to these subcellular localization layers, with each layer representing a distinct membrane-bound biological compartment in the cell. Subcellular localization is gathered from databases of protein properties and is usually known for the majority of nodes in a graph. Nodes with unknown localization are either placed in a separate layer, or are allowed to be placed in any layer, depending upon the preference of the user.

Furthermore, the biologists' assessment of what constitutes a good layout varies depending on the nature of the biomolecules involved. In the undirected portion of the graph, which comprises protein-protein interactions that propagates a signal from membrane to nucleus, they wish to see the network structure so that they can follow the signalling cascade. Thus for this section of the graph, it is important to minimize edge crossings, even if it places interacting nodes somewhat far apart. In contrast, for the directed portion of the graph, representing the genes whose expression was altered in response to the signalling cascade, the biologists want to see the nodes grouped tightly by function, even at the expense of not being able to clearly see the interactions between them. Translating these desires into automated graph layout requires an algorithm that uses metadata associated with the nodes, in addition to the direct graph structure, for node placement. We wrote a simple simulated annealing-based graph layout algorithm that uses node metadata to guide node placement.

3.2 Small multiple views for multiple conditions

Cerebral uses small multiples [49] to simultaneously display multiple experimental datasets. Each small multiple contains a complete copy of the interaction graph with the same spatial layout, but with different colouring according to the experimental data it is displaying. As each small multiple

view encodes a single experimental condition, interesting data can pop out using preattentive processing [25]. Preattentive processing allows the visual system to find and estimate the count of nodes with the same colour without engaging focussed mental attention. Our design target was to handle from two to a few dozen experimental conditions, and from 50 to 3000 nodes in the interaction graph.

One obvious alternative to multiple small views would be a single changeable or animated view, where the colour coding changes over time rather than being distributed over space [42, 45]. Comparing something visible with memories of what was seen before is more difficult than comparing things simultaneously visible side by side [39]. Thus, the limitations of human memory make our goal of comparing a few dozen conditions through animation quite difficult [51]. Although small multiples would not scale to hundreds of conditions, they handle the current usage of 8-10 easily and will certainly accommodate the projected usage of few dozen conditions.

A second alternative is to embed a glyph, such as a line graph or heat map, near or within the node itself [42, 52]. While embedded glyphs provide good detail when zoomed in for a local view, the glyphs become indistinguishable when zoomed out for a global view for graphs larger than a few dozen nodes, as shown in Figure 3.1. The biologists often need to see such a global view, as it more readily allows for the identification of interacting genes/proteins whose expression behaves similarly across several conditions, thus glyphs would not be appropriate in this domain.

3.3 Parallel coordinates and clusters for measurement-driven exploration

Cerebral's main views focus on the interaction graph model of the biological system or process of interest. We also provide a measurement-driven view and tools to help suggest areas for exploration. Each experimental condition corresponds to an axis in a parallel coordinates view. Each biomolecule, or node, in the graph is represented as a line that crosses each of these axes

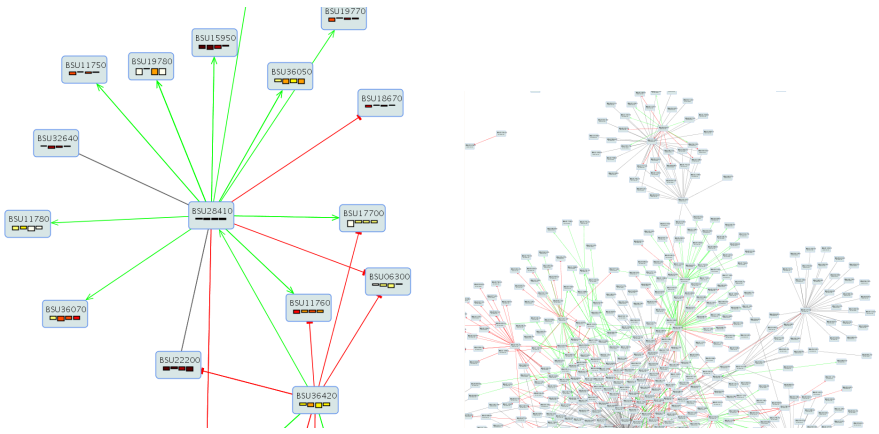


Figure 3.1: Though the glyphs in GENeVis [52] provide good detail when zoomed in for a local view (left), they quickly become indistinguishable when viewing a larger portion of the graph (right).

at point corresponding to that gene/protein's expression level under that condition, resulting in what the biologists refer to as an expression profile.

A frequent analysis of expression profile data involves clustering the profiles to identify genes/proteins that behave similarly across conditions, implying that they have related functions or are regulated by the same set of proteins. New and enhanced clustering algorithms are developed each year [53], but provide notably different results. Many of these methods are sensitive to noise, and the measurement technologies employed in systems biology often produce highly noisy results. Furthermore, the complex internals of a clustering algorithm – most often based on statistical rather than analytical decisions – make it hard for a biologist to understand and trust the clusters.

Despite these weaknesses, automatically detected clusters can complement the visualization of the interaction graph to direct the biologist towards interesting groups of proteins to explore. The potential for uncovering related or novel functions and regulatory mechanisms is valuable, as is the increased confidence in the statistically-generated cluster assignments if the clustered genes are found to biologically interact with each other.

Because our immunologist collaborators do not consider any current clus-

tering algorithm to be a clear winner, we implemented simple k-means clustering as a proof of concept. It would be straightforward to substitute a more sophisticated clustering algorithm into Cerebral. The focus in this thesis is on the design choices for the multiple coordinated views [4], and the layout algorithm in the graph view.

Chapter 4

Related work

We will review related work in the area of graph layout, followed by work that influenced our data interface, and finally compare against some other complete systems designed for the task of analysing expression data.

4.1 Graph layout

The general layout problem of assigning graph nodes to positions in a two dimensional plane has been addressed by many researchers [31]. We will restrict our discussion to force-directed layout, layout with constraints, and simulated annealing layout. We will discuss systems that have previously used the uniform grid to optimize layout. Finally we will review edge bundling, an alternative to drawing edges as straight lines.

4.1.1 Force-directed layout

Force-directed algorithms use a physical simulation model to lay out the graph, with edges modelled as springs and nodes as repulsive electric charges. Force-directed algorithms remain highly popular due to their simplicity and success at producing aesthetically pleasing layouts, and are implemented in several biological graph visualization packages [7] [13] [28] [45]. While unconstrained force-directed layouts cannot produce a layered subcellular localization view of the cell, we introduce them for comparison purposes.

We chose the GEM system [18] as an exemplar for force-directed algorithms. GEM builds on the original spring-embedder model proposed by Eades [17] and the improvments suggested by Fruchterman and Reingold [19], to produce a system that is fast, has good convergence, and detects

oscillations and rotations. While GEM can reveal the inherent complexity of a graph, we will see that relevant biological information can be obscured without a subcellular localization-based view of the cell. We compare Cerebral to GEM layouts in Section 5.5.

4.1.2 Force-directed layout with constraints

Several attempts have been made to extend force-directed placement to accommodate grouping. For example, Fruchterman and Reingold [19] use repulsive walls to contain nodes within an external boundary and Genc and Dogrusoz [20] use mobile internal walls to separate the graph into compartments. However, these methods suffer from fragility, requiring parameter adjustments to work across a range of datasets. As the graphs scale up in size, it becomes a problem to balance forces. If the repulsive forces of the walls are too large, the nodes are pushed strongly away from the walls and cluster in the middle. If the wall forces are too low, all of the nodes cluster against the walls. Moreover, they have only been shown to work on small datasets of less than 100 nodes.

4.1.3 Quadratic programming with constraints

He and Mariott [24] formulated graph layout with constraints as a quadratic programming problem. Their constraints enforced a minimum distance between nodes. Dwyer and Marriott [15] extended the class of constraints to linear separation constraints and created a specialised solver to quickly lay out large graphs in the IPSep-CoLa system. After satisfying the constraints, both of these techniques optimize the Kruskal [33] measure for graph layout. The Kruskal measure tries to match graph theoretic distance to layout distance between pairs of nodes. We wanted a more flexible measure that could evaluate other layout criteria, including intrinsic constraints such as edge crossings, and constraints based on extrinsic node attributes such as the biological function of nodes. Moreover, IPSep-CoLa also requires considerable parameter tweaking to work well in this application domain. Even after multiple rounds of personal communication with the authors, we did

not obtain competitive results. We compare Cerebral to IPSep-CoLa in Section 5.5.

4.1.4 Simulated annealing

Davidson and Harel [11] introduced simulated annealing to graph drawing. Although their framework was simple, flexible, and produced visually appealing results, their algorithm had cubic complexity and high constant factors, making it impractical for graphs larger than 50 nodes.

Li and Kurata [34] used simulated annealing to lay out biomolecular interaction graphs. Although they restricted node positions to an integer grid, their algorithm still ran in $O(V^3)$ time. Kato and Nagasaki [30] built on their work by introducing soft constraints to simulated annealing, adding edge crossings and node-edge crossings to the evaluation function. Their algorithm also had cubic complexity, and the largest graph shown in the paper contained 125 nodes. Further work [32] decreased layout time by a constant factor and changed the scoring function to align biologically related nodes. In contrast, we provide an $O(E\sqrt{V})$ algorithm for constrained graph layout with simulated annealing, and demonstrate it on much larger datasets.

4.1.5 Uniform grid

Akman *et al.*[2] introduced uniform grids as a data structure to optimize counting intersections of a large number of short line segments. The uniform grid partitions space by separating the plane into equally sized square cells. Unlike an adaptive grid, e.g. a quadtree, each cell in a uniform grid is the same size regardless of the number of objects in that region of space. Each cell stores a list of lines that pass through it. To count crossings, the algorithm follows a line through the grid, accumulating a set of possible intersecting lines, which are tested with the line intersection equation. A rasterization algorithm quickly identifies which cells a line passes through. Uniform grids are very efficient when there are numerous short edges, but the efficiency decreases as edges get longer and the number of expected intersections, I, increases. As an edge will intersect many grid cells, it

becomes more efficient with long edges to use a sweep line algorithm [6] that finds all intersections on a plane in $O(n \log n + I)$ time.

Tunkelang [50] presented an optimization-based approach to drawing undirected graphs that used a uniform grid to accelerate the evaluation of edge crossings. To keep edges short, the algorithm starts with an initial layout based on node centrality, then evaluates new configurations created by local adjustments to node positions. This local optimization method can get stuck in local minima. In our approach, we use a modified version of the uniform grid that remains efficient even as edge length and expected number of crossings increases, thereby allowing a more global search of the configuration space.

4.1.6 Edge bundling

Edges in graph layouts are typically displayed as straight lines connecting pairs of nodes. Edge bundling draws edges as curved lines grouped together into bundles around routing points. Holten [27] created bundles to reveal the structure of an associated tree hierarchy associated with the graph. We use Holten's edge bundling idea and his system of rendering bundles as splines with a parameter β that affects how tightly the edges are bundled, however we position our spline control points not from an associated hierarchy, but by the layout location of the nodes.

Deussen and Balzer [5] also bundle edges using the node layout location. Their system bundles edges between meta-nodes which contain sets of nodes grouped by a clustering function. As the bundled edges connect a set of nodes to another set of nodes, their edges establish a many-to-many relationship between the nodes. When looking at an edge bundle in the graph, the viewer only knows that the meta-nodes are connected, but has no knowledge of what individual nodes are connected. In contrast, our edge bundling algorithm forms a one-to-many relationship between nodes. The thick end of the bundle links to a single node. When looking at the branched end of the bundle the viewer can easily recall what node lies at the other end. When a node connects to many different bundles, the bundles are

merged together, like streams joining into a river, similar to the display of flow maps [38].

4.2 Data interface

Coordinated multiple view systems [41] have multiple windows showing different views of the data. Relationships between the multiple views are revealed to the user through interactive brushing and selection. Cerebral uses coordinated multiple views to examine expression data.

Dynamic query sliders [1] are graphical widgets used for querying a dataset. The user moves a slider to alter a parameter and the program interactively adjusts the displayed data to reflect the new setting. Dynamic sliders are used throughout Cerebral.

Saraiya *et al.* [42] evaluated four visualization approaches to integrating graph and time series data. They compared an animated series of graphs with the nodes coloured according to the time point against a static graph with embedded chart glyphs showing the node values across all time points. They also compared a graph view alone against a multiple view interface with the graph view linked to a parallel coordinates view. In almost all tasks they found single node colourings to be faster and more accurate than embedded charts, and that multiple views increased accuracy at the expense of slower task times. The one exceptional task was outlier detection for which embedded charts was found to be superior. We use single coloured nodes throughout our visualization, and attempt to improve the user interface further by using small multiples instead of animation to view multiple time points. We employ multiple views of the data as we deemed accuracy to be more important than task time. Finally, outlier detection is easily handled by a direct computational approach. We offer filters and clustering to find outliers, rather than relying on the biologist to visually search the dataset.

4.3 Biological expression analysis systems

Many visualization systems aimed at biological problems follow the well-established practice of using multiple linked coordinated views. For example, analysis tools such as SpotFire and HCE [44] provide a rich set of statistical tools including scatterplots, parallel coordinate views, and heat maps. However, they do not contain a biomolecular interaction graph view.

Several previous tools do allow visualization of gene expression data from a single experimental condition on a biomolecular interaction graph, including Cytoscape [45], Visant [28], GeneSpring [21], and GenMapp [10]. However, they are limited to visualizing data from a single experimental condition at a time and do not support automatically positioning nodes according to biological metadata. Although Cytoscape does have a graph layout algorithm that incorporates biological context, it requires manual intervention.

Chapter 5

Graph layout

We use simulated annealing to lay out the graph using biological annotations to guide the placement of the nodes. We begin with a review of using simulated annealing for graph drawing, and then describe how we add constraints to this approach. We present a discretization framework which speeds computation times and enhances the appearance of our layout. We analyse the complexity of the algorithm, revealing that the algorithm runs in expected time $O(E\sqrt{V})$ while using $O(V)$ memory. Finally we present examples of our layout and compare with other algorithms, and investigate the consequences of using approximations and randomisation.

5.1 Graph drawing with simulated annealing

Simulated annealing is an optimisation technique that has been applied to many NP-hard problems. The algorithm combines the simplicity of a greedy algorithm with the power of randomisation. Simulated annealing tries to minimize the total energy of a system by perturbing the state over a cooling period. At the start, when the system is hot, new configurations are freely accepted allowing escapes from local minima. As the system cools, the configuration becomes more stable. For graph layout, the energy function implements soft constraints, which are often aesthetic qualities such as edge length and overlaps. New configurations are generated by moving a single, randomly chosen node to a new location.

Figure 5.1 shows the pseudocode for simulated annealing. We start with an initial random layout, and set the temperature to the average energy over all the nodes in that layout. The initial temperature measure is the only time we compute the total energy over all nodes, after which we only compute the

```
SA-Graph-Layout(node[1..V])
 1  coolingSteps ← 30
 2  searchSteps ← 50
 3  coolingFactor ← 0.6
 4  for i ← 1 to V
 5      MoveNode(node[i])
 6  temperature ← 0
 7  for i ← 1 to V
 8      temperature += Energy(node[i])/V
 9
10  for i ← 1 to V
11      if !node[i].pinned
12              mobileNodes.add(node[i])
13
14  for i ← 1 to coolingSteps
15      for j ← 1 to searchSteps × V
16              n ← mobileNodes[Random(1, |mobileNodes|)]
17              oldEnergy ← Energy(n)
18              MoveNode(n)
19              improvement ← oldEnergy −Energy(n)
20              if improvement > 0
21                 then Accept change
                                                  improvement
22                 else Accept with probability 1 - e^(-----------)
                                                  temperature
23      temperature ← temperature ×coolingFactor
```

```
MoveNode(node)
1  uniformGrid.remove(node)
2  (y_min, y_max) = LayerConstraint(node.type)
3  repeat
4     newPos ← (Random(1, uniformGrid.width), Random(y_min, y_max))
5  until !grid.cellOccupied(newpos)
6  node.pos ← newPos
7  uniformGrid.add(node)
```

Figure 5.1: Pseudocode for simulated annealing.

energy change caused by a moved node. We perturb the state of the system by randomly picking a new position for a single node. The energy change is computed by subtracting the energy contribution of the node in its new position from its original position. We always keep the change if the energy is reduced, and also probabilistically accept changes that worsen our layout, allowing the algorithm to escape from local minima. Every $O(V)$ iterations of the search we decrease the probability that a worsening change will be accepted. The parameters of 30 cooling steps, $50 \times V$ search steps, and a geometric temperature decrease of 0.6 were chosen empirically to produce good layouts, based on typical values from previous simulated annealing algorithms [11].

5.2 Adding constraints

Hard constraints are realised in the SA-Graph-Layout function by assuring that the node position picked at random in MoveNode satisfies all the hard constraints. In Cerebral, our explicit hard constraint is to confine nodes to horizontal bands. Before the main loop begins, we divide the space on the y axis into regions sized proportionally to the number of nodes in each layer. We can look up the minimum and maximum allowable y values for the layer in constant time, and generate a position within this range. This framework could accommodate any other constraints that can be computed in constant time, for example bounding nodes in both the horizontal and vertical dimensions. In Section 5.3 we discuss the additional, implicit hard constraint that node-node overlaps are not allowed.

Soft constraints are handled though the Energy function, such that nodes violating the constraints have higher scores. In Cerebral, edge-edge and node-edge crossings are always penalised. To group nodes according to their biological function, the BioFunctionGrouping function sums the layout distance from a node to each functionally related node. The soft constraint scoring functions are shown in Figure 5.2.

We weight our constraints as follows: our unit weight is a separation distance of one grid cell. An edge-edge crossing has weight 3, a node-edge

```
ENERGY(node)
1  energy ← UNIFORMGRID.COUNTCROSSINGS(node)
2  energy += MANHATTANEDGELENGTH (node.edges)
3  energy += BIOFUNCTIONGROUPING(node)

BIOFUNCTIONGROUPING(node)
1  energy ← 0
2  group ← NodesWithFunction(node.role)
3  for each node g in group
4      energy += MANHATTANDISTANCE(node.XY, g.XY)

MANHATTANEDGELENGTH(edges[1..])
1  for i ← 1 to |edges|
2      ManhattanDistance(edges[i].source.XY, edges[i].dest.XY)

MANHATTANDISTANCE(a, b)
1  return abs(a.x - b.x) + abs(a.y-b.y)
```

Figure 5.2: Pseudocode for energy computations that handle soft constraints

crossing has weight 9. Biological function grouping has a weight of 90, which encourages grouping even at the expense of node and edge overlaps. These settings were chosen by visually inspecting several graphs and choosing parameters that produced nice layouts. Though the specific choice of parameters is somewhat arbitrary, they provide good visual results for a wide range of interaction graphs containing 50 to 10,000 nodes. Further, the algorithm is robust to these parameters, continuing to give good results when they are modified by 50% to 200%. They are thus fixed in Cerebral, and do not require user tweaking.

5.3 Discretization

The energy function is the heart of our algorithm, scoring the soft constraint violations of a single node and its incident edges. It is evaluated each time a node is considered at a new position: $1500 \times V$ times in total. Profiling a simple implementation showed that over 98% of the algorithm running time was spent testing for edge intersections. To evaluate our energy function efficiently, we partition our layout space using a uniform grid, and restrict node placement to the centres of unit length cells. We set the size of the grid to $3.2\sqrt{V}$ by $2.5\sqrt{V}$, which gives a 4:3 aspect ratio for the layout and provides $8V$ unit length cells in the grid leaving $7V$ empty cells for rearranging the nodes while keeping a compact layout. The seemingly simple idea of using a uniform grid has far-reaching consequences. First, it implicitly eliminates the possibility of node-node overlap. We can easily support the hard constraint that only one node can be located in a cell, and we draw nodes at a size smaller than the lattice cell size. Second, enforcing a limit on the lattice size also implicitly supports a compact layout as a hard constraint. Third, placing nodes at integer lattice points tends to increase the angular resolution at which edges cross, as shown in Figure 5.3. This aesthetic measure [40] is a soft constraint provided without any computational cost by our discretization approach.

Fourth, with a uniform grid, we can compute node distances cheaply. Instead of using Euclidean distance, we use Manhattan distance in all com-

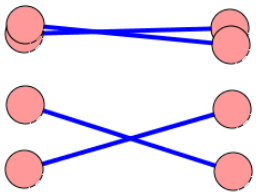


Figure 5.3: Edge crossings are unavoidable in non-planar graphs. When edges cross, it is easier to follow lines when they cross at high angular resolution (bottom), than when they intersect at glancing angles (top).

putations. We compute the L_1 distance measure using only integer arithmetic, as $d(a, b) = |a_x - b_x| + |a_y - b_y|$. We use this fast distance calculation in two of the soft constraints, edge length minimization and node proximity according to biological function.

Fifth, and most importantly, we use the grid to quickly compute approximate counts of edge-edge crossings and node-edge overlaps. Our approximation relies on storing the number of edges crossing each cell, updating the grid every time a node is moved.

We follow an edge through the grid using a variant of Bresenham's [8] line rasterization algorithm as shown in Figures 5.4 and 5.7. As the rasterizer visits each cell it can perform a VACATE, OCCUPY, or COUNT operation on each cell, described in Figure 5.5. As we move candidate nodes during simulated annealing, we maintain the grid state with functions shown in Figure 5.6.

The standard usage of a uniform grid [2] is to keep a list of edges that cross each cell, and then do an exact intersection test with each distinct edge during the traversal. We instead accumulate only the count of edges crossing each cell in a constant time lookup operation. Counting the edges is equivalent to checking for intersections at the granularity of the grid. The cost of scoring an edge is thus proportional to the number of cells it crosses; that is, the discretized edge length. Figure 5.8 gives an example of our scoring method.

Finally, the grid also allows us to compute node-edge overlaps quickly,

```
modBresenhamVisit(x0, y0, x1, y1, operation)
 1  steep = abs(y1-y0) > abs(x1-x0)
 2  if steep
 3      swap (x0,y0), swap (x1,y1)
 4  if x0 > x1
 5      swap (x0, x1), swap (y0, y1)
 6  deltax ← x1 - x0, deltay ← abs(y1-y0)
 7  deltaerror ← deltay/deltax
 8  y ← y0
 9  ystep ← (y0 < y1) ? 1 : -1
10  for x← x0 to x1
11      steep ? operation(y,x, error) : operation(x,y, error)
12      error ← error + deltaerror
13      if error ← 0.5 ▷ Standard Bresnham's until here.
14              ▷ Now the modifications. Also visit the cell we pass
15              ▷ through as we change y coordinate
16              if x < x1
17                 if error = 1.0
18                         ▷ Exiting at the exact corner
19                         y ← y + ystep;
20                 elseif (error > 1.0)
21                         ▷ Above the true line.
22                         y ← y + ystep;
23                         steep ? operation(y,x, error) : operation(x,y, error)
24                 else
25                         ▷ Below the true line
26                         steep ? operation(y,x+1, error) :
27                            operation(x+1,y, error)
28                         y ← y + ystep;
29              error ← error - 1.0;
```

Figure 5.4: Modified Bresenham's algorithm visits all cells a line passes through. See also Figure 5.7. The possible operation functions are shown in Figure 5.5.

```
VACATE(x, y, centerOffset)
1  if centerOffset < nodeWidth
2      edgesThroughCenterCount[x][y]–
3  edgesInCellCount[x][y] –


OCCUPY(x, y, centerOffset)
1  if centerOffset < nodeWidth
2      edgesThroughCenterCount[x][y]++
3  edgesInCellCount[x][y] ++


COUNT (x, y, centerOffset)
1  if centerOffset < nodeWidth
2      if cellOccupied[x][y]
3              energy += nodeEdgePenalty
4  energy += edgesInGridCount[x][y] * edgeEdgePenalty
5  return energy
```

Figure 5.5: The operators OCCUPY, VACATE, and COUNT can be executed on each cell visited by MODBRESENHAMVISIT

```
uniformGrid.remove(node)
1  for each edge e adjacent to node
2      modBresenhamVisit(e.source.XY, e.dest.XY,vacate)
3  uniformGrid.cellOccupied(node.x, node.y) ← false
```

```
uniformGrid.add(node)
1  for each edge e adjacent to node
2      modBresenhamVisit(e.source.XY, e.dest.XY,occupy)
3  uniformGrid.cellOccupied(node.x, node.y) ← true
```

```
uniformGrid.countCrossings(node)
1  ▷ Count the ways this node's edges cross other edges and nodes
2  for each edge e adjacent to node
3      energy ←modBresenhamVisit(e.source.XY, e.dest.XY, count)
4  ▷ Then check how placing a node in this cell causes intersections
5  ▷ with the edges already passing through this cell
6  energy ← edgesThroughCenterCount[n.x][n.y] *nodeEdgePenalty
```

Figure 5.6: Code for maintaining the uniform grid as we remove, add, and count crossings in the grid.

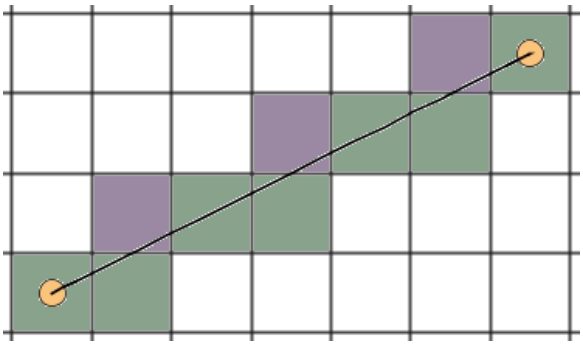


Figure 5.7: The cells of a uniform grid that a line crosses are found with a modified form of Bresenham's algorithm. The green cells are the standard pixels visited by Bresenham's algorithm. The purple cells are the additional corners visited in the modified form.

as an explicit constraint that is also handled during the traversal to check for edge-edge overlaps. We keep track of which grid cells are occupied by a node, and we maintain an error term during our line rasterization traversal indicating the current distance from the centre of the cell. If this error term is less than the radius of a node, we count a node-edge overlap in the cell.

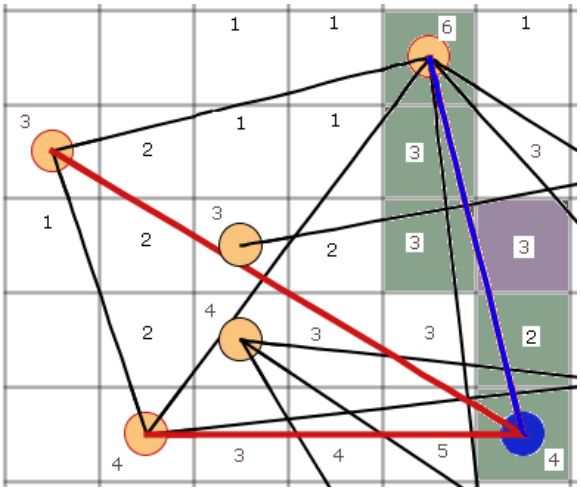


Figure 5.8: Scoring a node. We count crossings for the blue node. The blue line accumulates $4+2+3+3+3+6=21$. We also score the two red edges for a total of $21+24+20=65$.

5.4 Asymptotic analysis

We now analyse the computational complexity of our algorithm. Let V be the number of nodes in the graph, E be the number of edges in the graph, and $d(v)$ be the degree of node v, that is the number of edges adjacent to node v. We will also make the conservative assumption that biological functional groups that are to appear as a cluster contains no more than $\sqrt{V}$ members. This assumption was true for all of the biological graphs we encountered in this project.

The major work of SA-Graph-Layout happens within the two main **for** loops, lines 14 and 15. These loops combine to execute the loop body coolingSteps $*$ searchSteps $* V = 30 * 50 * V = 1500V = O(V)$ times.

Within the loop, ENERGY is evaluated twice, MOVENODE is evaluated once, and all other lines execute in constant time. Before evaluating the cost of ENERGY and MOVENODE let us first assess the cost of using the uniform grid.

MODBRESENHAMVISIT performs an operation, VACATE, OCCUPY, or COUNT, on each cell that an edge intersects in the uniform grid. The longest possible edge is the long diagonal of the grid, which has length

$$l = \sqrt{(3.2\sqrt{V})^2 + (2.5\sqrt{V})^2} = O(\sqrt{V})$$

so MODBRESENHAMVISIT performs an operation at most $O(\sqrt{V})$ times. The operations VACATE, OCCUPY, and COUNT all do a constant amount of work, so MODBRESENHAMVISIT takes at most $O(\sqrt{V})$ time.

UNIFORMGRID.REMOVE executes MODBRESENHAMVISIT $d(v)$ times for a total cost of $O(\sqrt{V}d(v))$. The same is true for UNIFORMGRID.ADD and UNIFORMGRID.COUNTCROSSINGS.

We return now to the cost of evaluating ENERGY. UNIFORMGRID.-COUNTCROSSINGS costs $O(\sqrt{V}d(v))$. MANHATTANEDGELENGTH costs $O(d(v))$. BIOFUNCTIONGROUPING costs $O(\sqrt{V})$ under our assumption that a biological group contains no more than $\sqrt{V}$ members. So the total cost of ENERGY is $O(\sqrt{V}d(v) + d(v) + \sqrt{V}) = O(\sqrt{V}d(v))$.

For the function MOVENODE, lines 1 and 7 each take $O(\sqrt{V}d(v))$ time. Line 2 executes in constant time as it is a simple matter to associate the layer constraint boundaries with each node. As the grid contains $8V$ cells by construction and there are V nodes in the grid, there will always be $\frac{7}{8}$ of the cells empty. Thus lines 3-5 will require an average of $\frac{8}{7}$ random trials to find an unoccupied grid cell. The total cost of MOVENODE is $O(\sqrt{V}d(v))$.

As the cost of both ENERGY and MOVENODE is $O(\sqrt{V}d(v))$ and the SA-GRAPH-LAYOUT loops execute $O(V)$ times, the total cost of SA-GRAPH-LAYOUT is $O(V\sqrt{V}d(v))$. As we are equally likely to pick any node in line 16, we take $d(V)$ to be the average degree of a node $= \frac{E}{V}$. This gives a total expected cost of $O(V\sqrt{V}\frac{E}{V}) = O(E\sqrt{V})$.

In terms of memory, each cell stores a constant amount of information;

the count of edges crossing it and whether the cell is occupied by a node. As there are $8V$ cells in the grid, we use $O(V)$ memory.

This analysis applies for graphs up to 10,000 nodes and edges. While we anticipate continuing good results as the graph sizes grow, it is possible that larger graphs would benefit from an increase in coolingSteps or a larger grid.

5.5 Graph results

We now present the results of running Cerebral on two sample graphs. We compare the layouts to competitive previous work in two categories: GEM [18] for force-directed placement, and IPSep-CoLa [15] for constraint-based layout. Our graphs, `TLR4` and `MAPK`, are networks involved in mammalian innate immunity. `TLR4` has 57 nodes and 74 edges, while `MAPK` has 760 nodes and 1263 edges. Cerebral supports the mechanism from Cytoscape for flexible assignment of visual cues such as colour and shape to nodes. Here we colour-code the nodes according to their subcellular location: pink for extracellular, blue for cell membrane, green for cytoplasm, orange for nucleus, and grey for unknown. All layouts are rendered with the Cerebral renderer for a consistent visual appearance, and all algorithms were run on a 3GHz Pentium running Linux with 2GB of RAM.

The Cerebral layout in Figure 5.9 has the standard layered arrangement used in cellular biology diagrams. Although we use the same colour coding as the other figures for reference, we note that because the layers are clearly shown with spatial position we could instead use colour for some other variable, for example protein abundance within the cell. We see that many of the cytokines are downstream from `REL`, `RELA`, and `NFKB1` suggesting that these three proteins might form a complex to activate cytokines.

Figure 5.10 shows the `TLR4` signalling pathway arranged using the GEM force-directed layout algorithm. The nodes are well placed and the topological structure of the network is very clear, yet it presents little in the way of biological context. Nodes are freely mixed throughout the layout, ignoring their subcellular location. The functional grouping is nowhere to

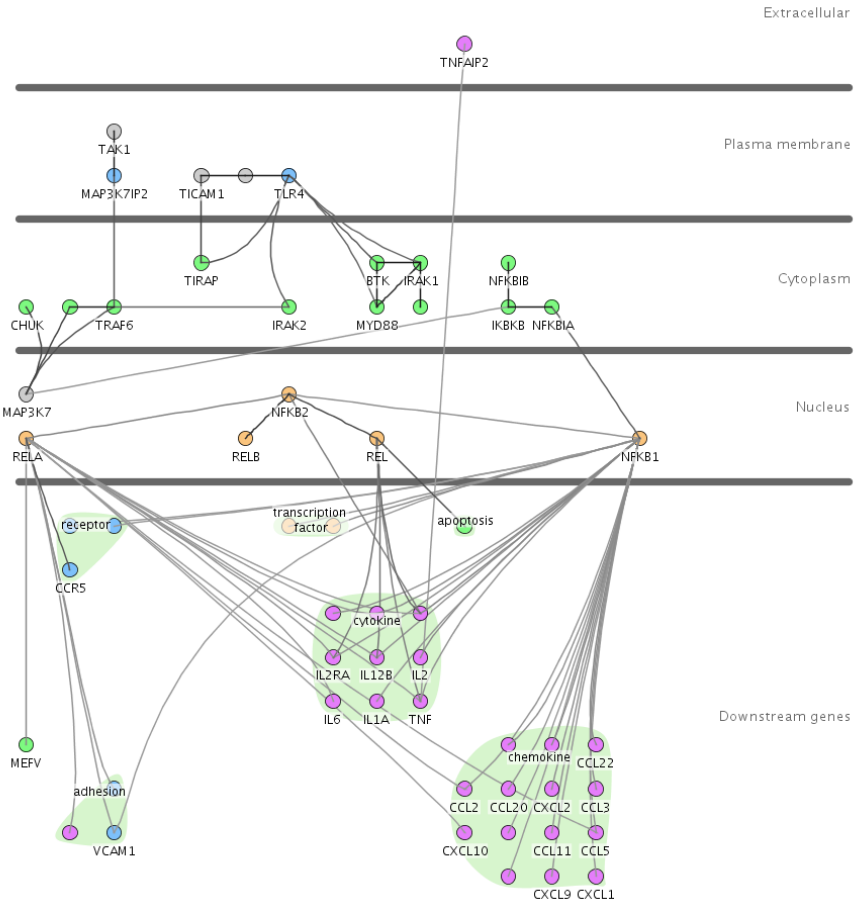


Figure 5.9: The TLR4 pathway using the Cerebral layout algorithm. Size: N=57, E=74. Time: 2.9 seconds. We can see how various proteins in the nucleus join together in pairs and triples to activate downstream genes. The long edge between `TNFAIP2` and `TNF` is an interesting feature that raises biological questions.

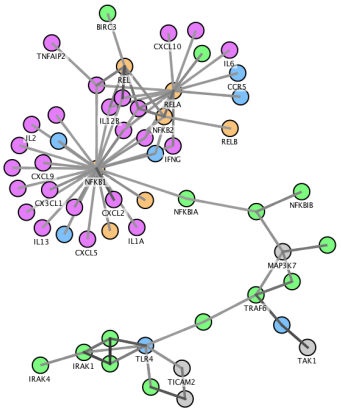


Figure 5.10: The TLR4 pathway using the GEM layout algorithm. Size: N=57, V=74. Time: $<$ 1 second. The graph topology is clear, but the biological context is hidden.

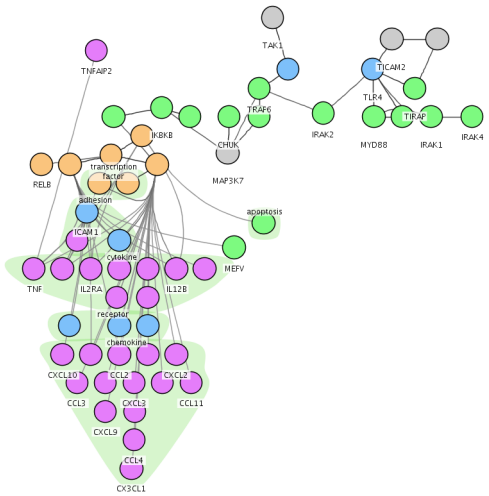


Figure 5.11: The TLR4 pathway using the IPSep-CoLa layout algorithm. Size: N=57, E=74. Time: 1.3 seconds. Using constraints we have been able to layer the graph and contain the downstream genes into functional groups. IPSep-CoLa does not attempt to optimize edge-edge crossings and node-edge crossings. There are numerous crossings in this diagram, hiding much of the topological structure.

be seen. The layout is unfamiliar to biologists as it bears no resemblance to the standard biology diagram style.

Most graph layout algorithms strive to avoid long edges. In Cerebral, the edge from the extracellular protein `TNFAIP2` to the downstream gene `TNF` in Figure 5.9 must be long because it spans many cell layers. This long edge, which is unremarkable in the other layouts, is an interesting feature rather than an indicator of a poor layout. It raises questions for our biologist target users. Is there an unusual interaction between the protein that resides outside the cell and its interacting neighbour generated within the cell? Does the long edge indicate an error in our data? Perhaps `TNFAIP2` is also a product of the `TLR4` pathway? Further investigations showed that despite being annotated as extracellular in several public databases, there was no literature support for this claim and the annotation was likely an input error.

Figure 5.11 shows the `TLR4` graph laid out with IPSep-CoLa. Using separation constraints between invisible boundary nodes and rectangular grouping for the downstream genes, we achieve a layered biological layout. IPSep-CoLa only optimises the Kruskal distance. It does not attempt to optimize edge-edge crossings or node-edge crossings. Figure 5.11 shows the disadvantage of this approach, because much of the topological structure is hidden under crossings.

We now consider the larger `MAPK` graph. We see from the unconstrained GEM layout in Figure 5.5 that this graph is inherently more complex than `TLR4`. Any unconstrained layout will necessarily have more crossings and longer edges. The Cerebral layout is shown in Figure 5.12. Though complex, in most regions of the graph we can follow links between edges. In particularly dense areas of the graph, mouseover highlighting helps to show connections. The Cerebral time of 62 seconds is slightly faster than the 66 seconds of GEM and considerably faster than the 296 seconds of IPSep-CoLa, shown in Figure 5.5.

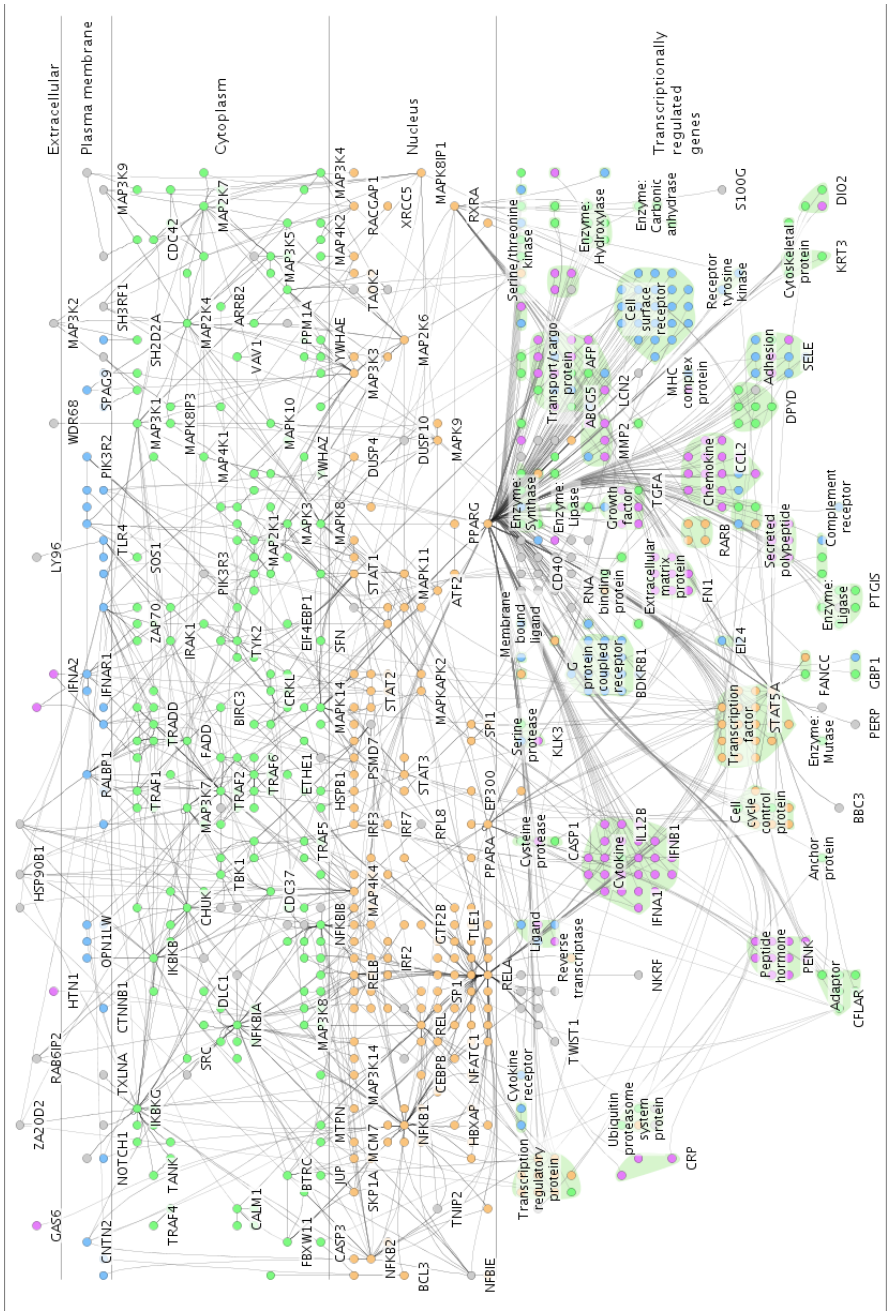


Figure 5.12: The MAPK pathway using the Cerebral layout algorithm. Size: N=760, E=1263. Time: 62 seconds.

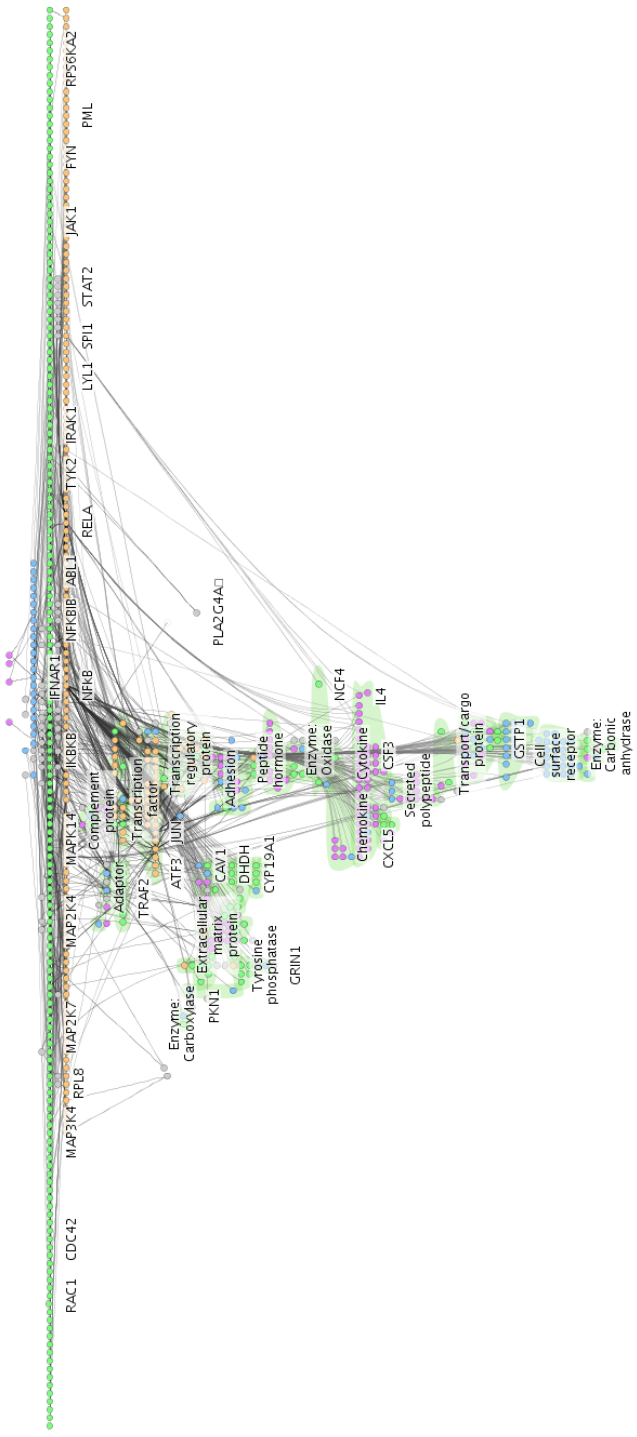


Figure 5.13: The MAPK pathway using the IPSep CoLa layout algorithm. Size: N=760, E=1263. Time: 296 seconds. In this view, the disadvantages of using only graph-theoretic vs. geometric distance for a layout constraint become clear because of the extreme visual clutter.

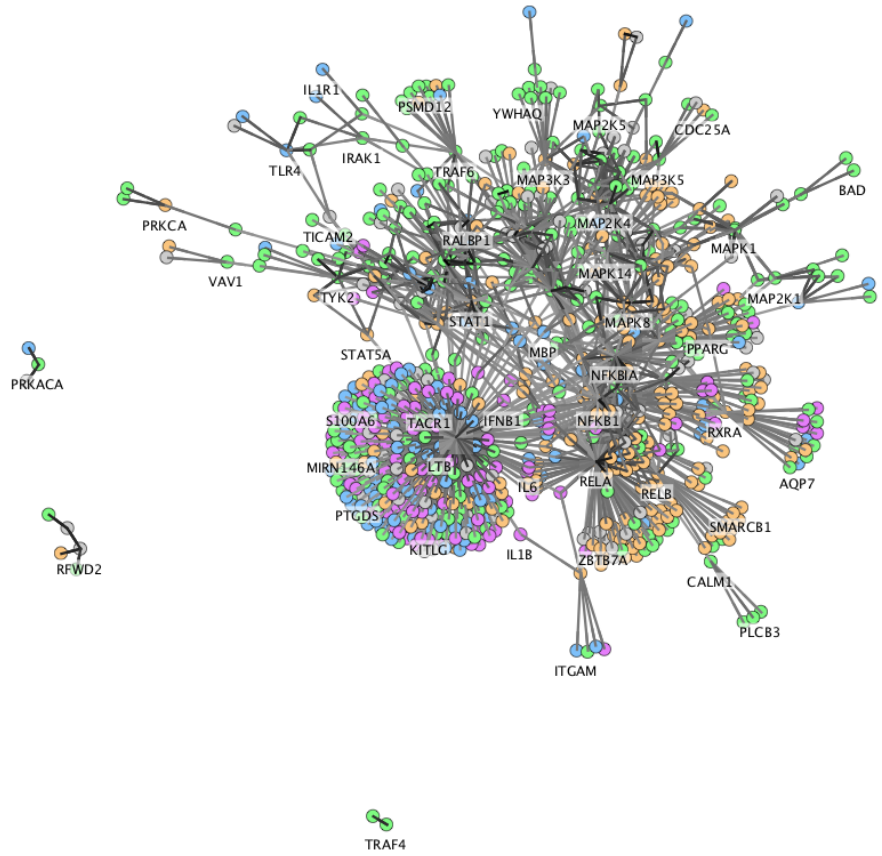
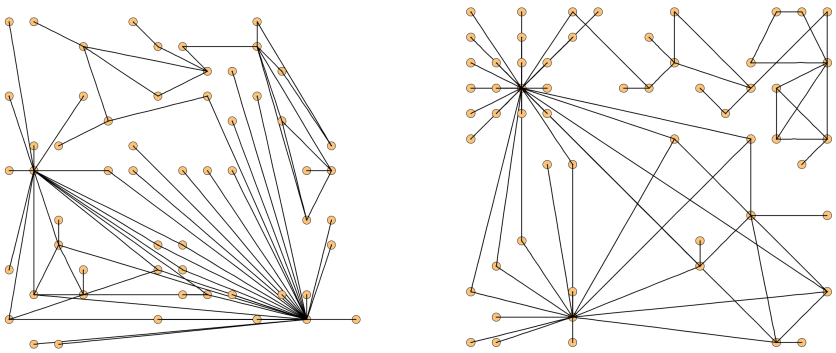


Figure 5.14: The `MAPK` pathway using the GEM layout algorithm. Size: N=760, E=1263. Time: 64 seconds. This force-directed rendering is unconstrained and shows the inherent complexity of the graph.

As expected, having no restrictions means that the GEM layouts better optimize aesthetic criteria such as short edges and crossings, but display less biological information to the user. Though IPSep-CoLa adds constraints to restrict the placement of nodes during layout according to biological function, the layouts contain many overlaps and crossings which hide much of the topological structure. Cerebral effectively arranges nodes in the style of classic biological diagrams while remaining competitive in optimizing classical graph layout criteria.

5.5.1 Effects of randomisation and approximation

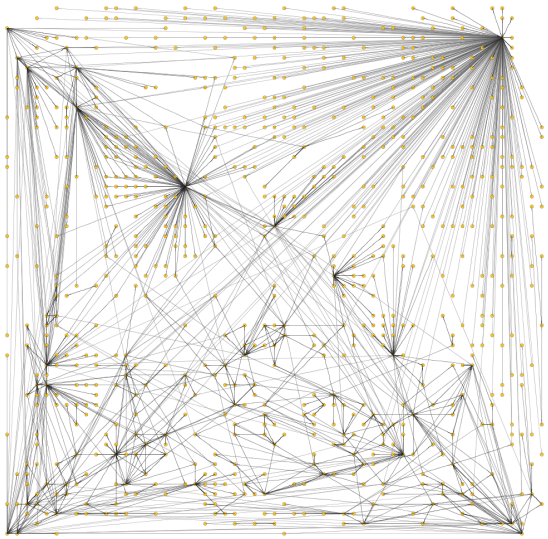
Our grid-resolution approximation is always an overestimation of the number of crossings, and is bounded below by the true count. Figures 5.15 and 5.16 compare the layouts produced by approximate versus exact edge intersection calculations, using a simple energy function where edge crossings are the only explicit constraint. In the smaller graph of 57 nodes and 74 edges, the approximate layout is more readable than the exact one, thanks in part to the improved angular resolution of the discretization approach. In the larger graph of 769 nodes and 1269 edges, it is no worse.



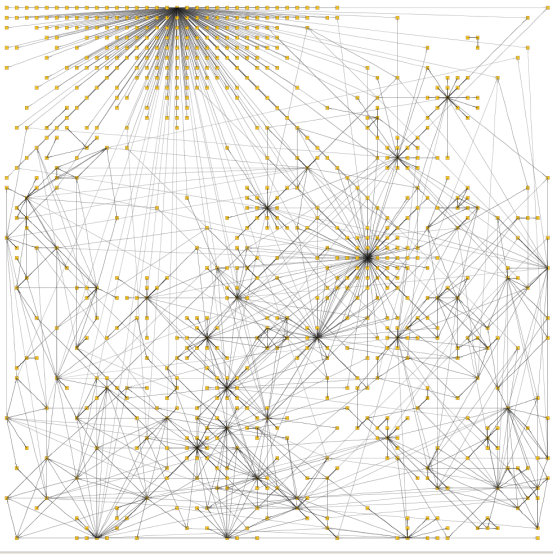
(a) (b)

Figure 5.15: Comparing approximate to exact edge crossings. In these layouts, the only constraint is edge crossings. **a)** Finding the layout of a small dataset (E=74, V=57) required 77 seconds when scoring edge crossings using an exact computation, and results in 3 crossings. **b)** The same dataset was laid out in only 12 seconds when scoring crossings approximately using our discretization framework. Although there are 12 rather than 3 crossings, the angular resolution between edges is better.

As the layout algorithm involves an element of randomness, a Cerebral layout will be different with each run. Figure 5.17 shows 6 consecutive different runs of the layout algorithm on the TLR4 dataset. The layouts finished with an average time of 4.0 seconds with a standard deviation of 0.0 seconds. Even though there is a large amount of variation between the absolute spatial positions of individual nodes between layouts, all of the



(a)



(b)

Figure 5.16: Comparing approximate to exact edge crossings. In these layouts, the only constraint is edge crossings. **a)** In a larger dataset (E=1269, V=769), the exact approach required 3100 seconds. **b)** The approximate approach required only 64 seconds to lay out the graph, and the quality is roughly equal to the exact case.

layouts show good quality with few edge crossings.

Table 5.1 shows the results of 10 consecutive layout runs on 5 different datasets. TLR4 is the original dataset discussed in Section 2, LL-37 Experiment is discussed in Section 7.1, the Yeast Cell Cycle Data is discussed in Section 7.2, IRAK4 contains the first and second degree neighbours of Interleukin-1 Receptor Associated Kinase 4, shown in Figure 5.18, MAPK is shown in Section 5.5, and finally Human Interactome shows a layout of all known human proteins and their interactions in Figure 5.19.

Dataset	Edges	Nodes	Avg. time(seconds)	σ
TLR4	74	57	4.0	0.0
LL-37 Experiment	123	91	6.1	0.2
Yeast Cell Cycle Data	417	104	14.2	0.1
IRAK4	617	513	29.6	0.2
MAPK	1269	769	61.9	0.7
Human interactome	53526	10344	6199	121

Table 5.1: Timings of our layout algorithm on human innate immunity graphs. We report the average and standard deviation (σ) of 10 consecutive runs.

Examining the table, we see that the standard deviation (σ) is very low in all cases, indicating that the randomness involved in the algorithm has a very small effect on the total running time. The average time to complete the graphs grows slowly as the edges increase as predicted by our theoretical runtime of $O(E\sqrt{(V)})$.

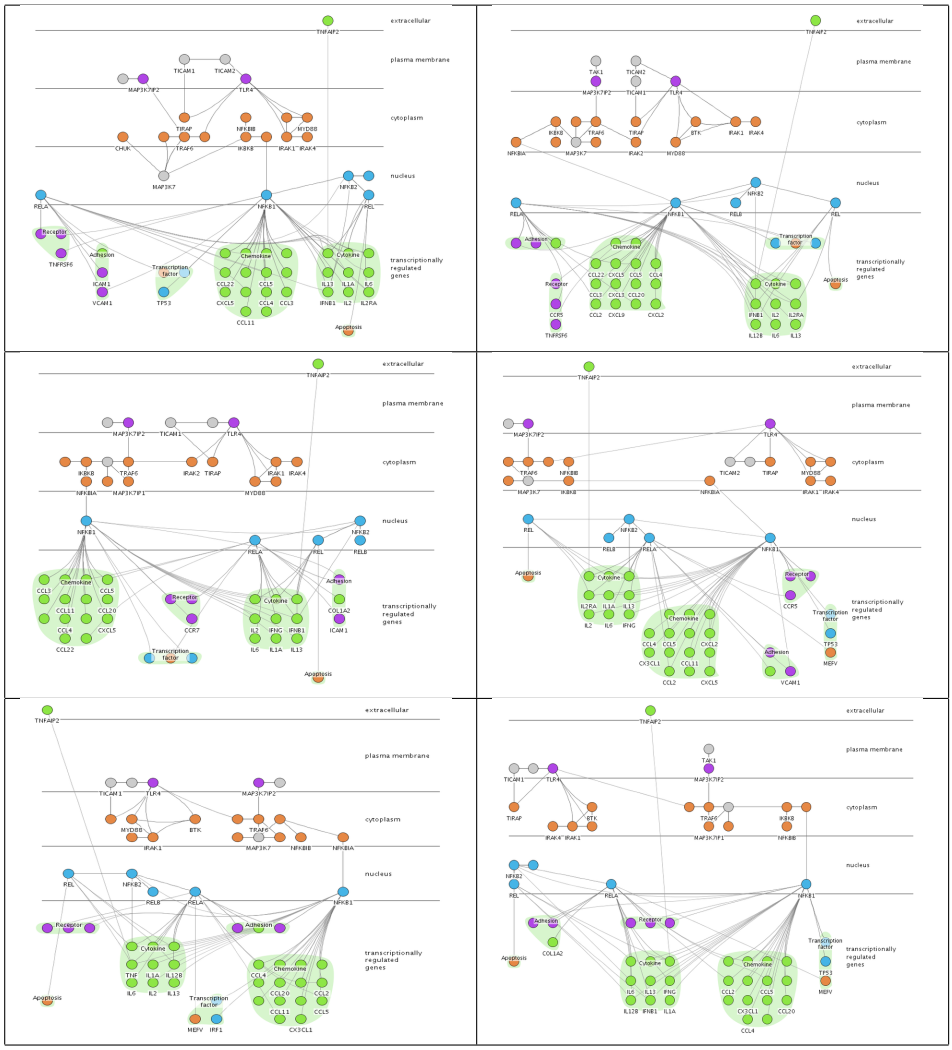


Figure 5.17: As our layout algorithm incorporate randomness, each run produces a different layout of the graph. This figure shows 6 consecutive layouts of the TLR4 (E=74, N=57) dataset. Though different, all runs produced a good layout for the dataset. The average runtime was 4.0 seconds with a standard deviation of 0.0 seconds.

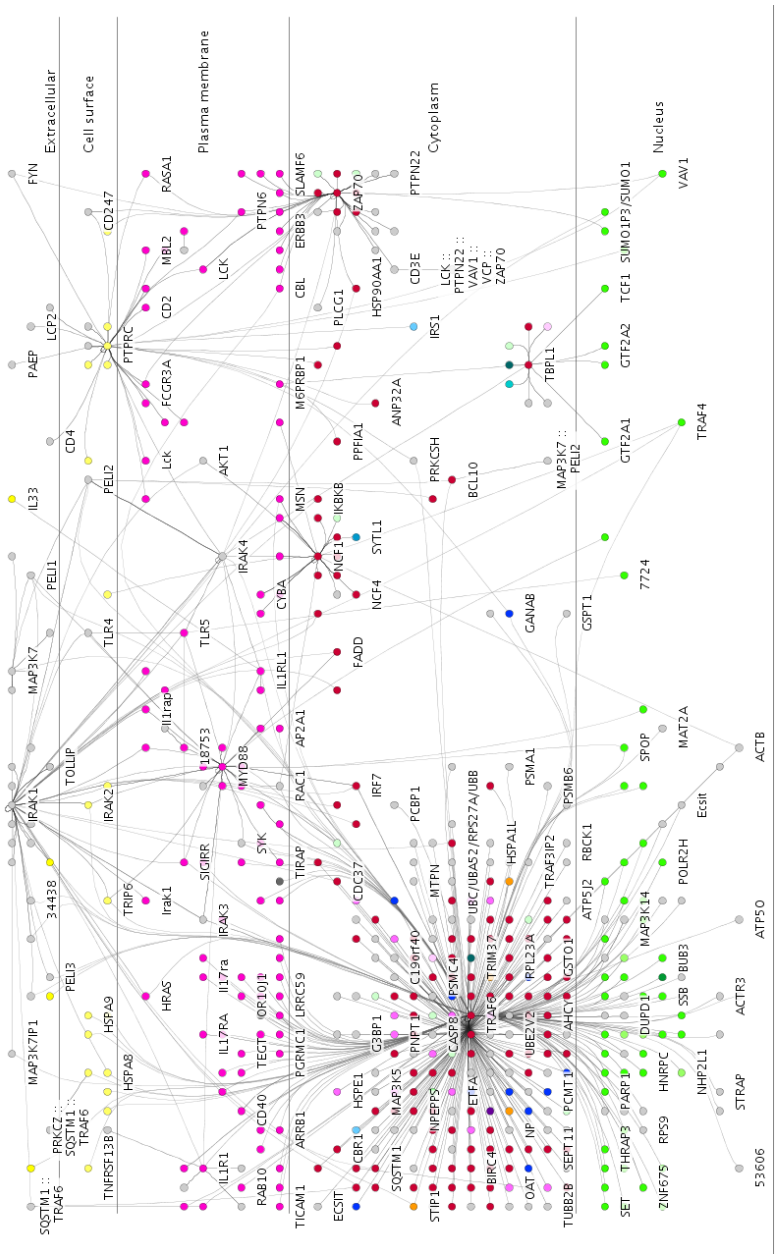


Figure 5.18: A midsized graph showing the first and second degree neighbours of Interleukin-1 Receptor Associated Kinase 4, Size: E=617, N=513. Time: 30 seconds.

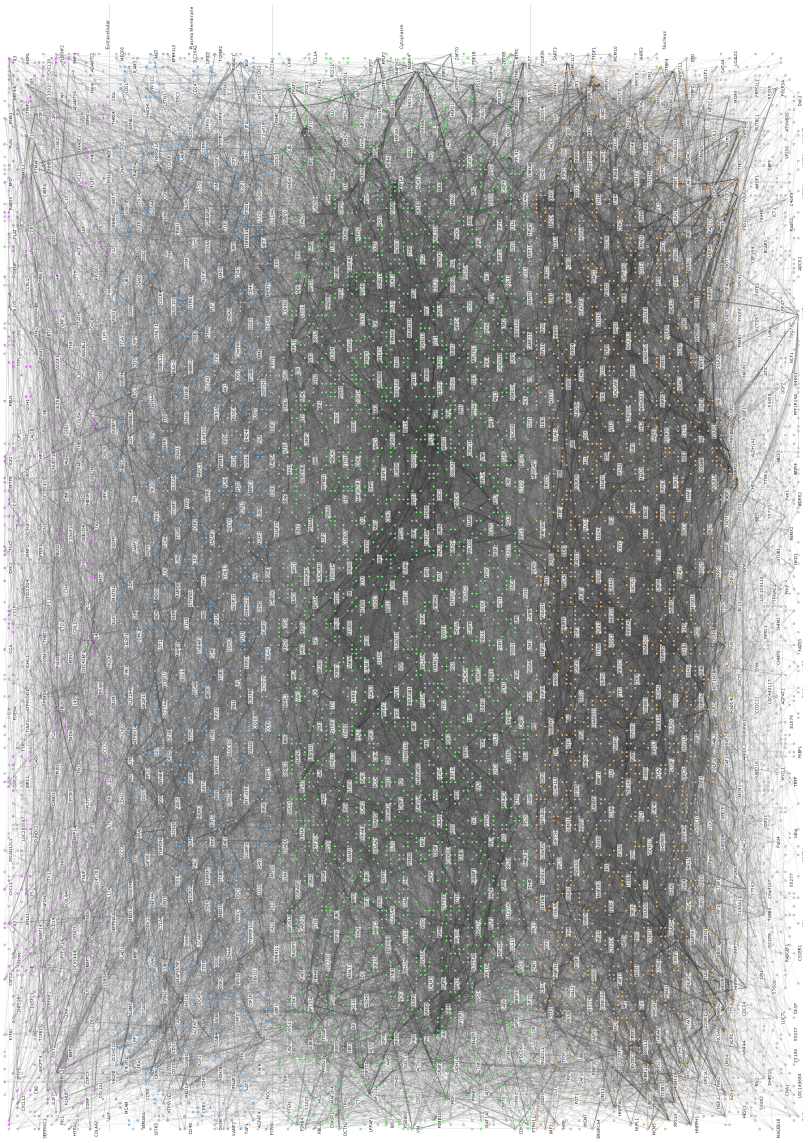


Figure 5.19: A very large graph showing all known human proteins and their interactions. While the resulting layout is too complex to be interpreted by biologists, it did not exceed the capabilities of the layout algorithm. Size: E=53526, N=10344. Time: 6282 seconds.

Chapter 6

Interactive interface

We will now describe the multiple coordinated views that make up the Cerebral system. We begin by describing the various panels that make up the Cerebral interface. We then describe means by which the biologist can interactively explore the datasets. A flexible colour scale system lets the user highlight data points both during direct examination of experiment data and in a calculated data comparison view. We describe the many means to manage large datasets with interactive filtering. We describe the visual simplification steps of edge bundling and intelligent labelling, which reduce visual clutter allowing the user to examine more data points simultaneously. Finally, we allow the biologist to manually position key nodes and let the layout algorithm optimally arrange the remaining nodes.

Cerebral is a highly interactive system. A video demonstrating the interactive features of Cerebral can be found online[1].

6.1 Interface components

Cerebral has three panels that display multiple coordinated views of the data and two panels which alter display attributes and filter data as shown in Figure 6.1

6.1.1 Multiple coordinated views

The biological graph model is laid out according to the algorithm presented in the previous chapter and displayed in the main view. The expression data is shown as two different presentations in the small multiples and parallel

[1] http://www.cs.ubc.ca/labs/imager/video/2008/BarskyMscThesis/cerebral.html

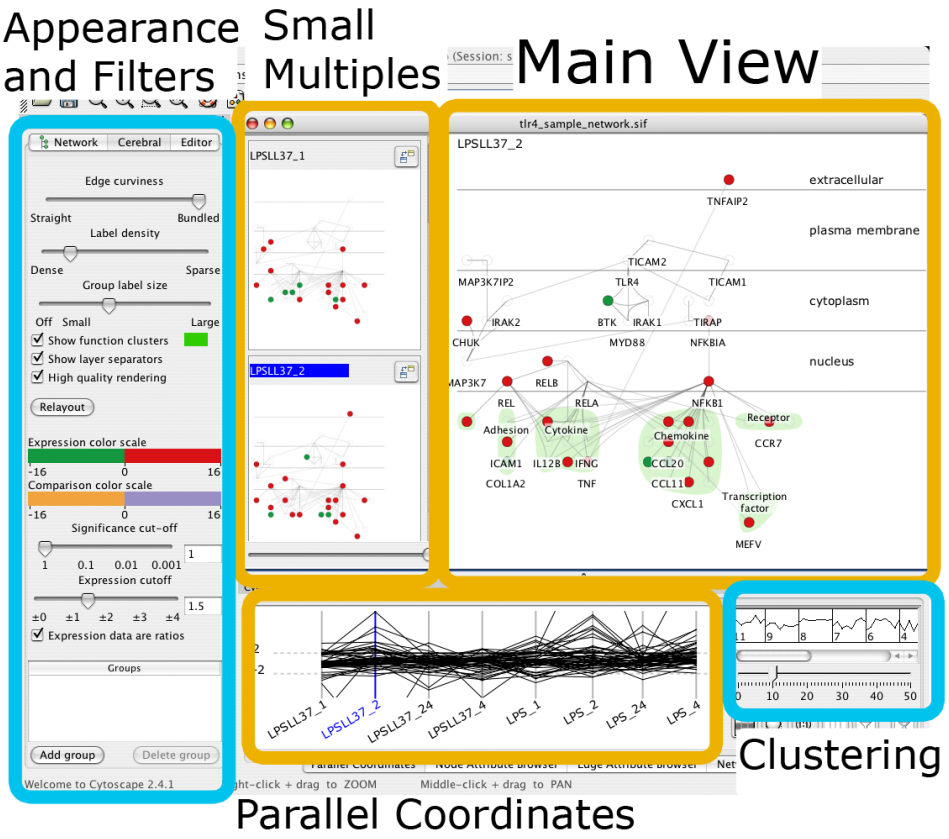


Figure 6.1: Cerebral has 3 coordinated data view panels (orange) and 2 data display and filter panels (blue). The Main View shows a detailed graph view with the nodes coloured by biological metadata, an experimental dataset, or a comparison scale between datasets. The Small Multiples view shows a graph in the same spatial arrangement as the Main view, with each graph coloured according to an experimental dataset. The Parallel Coordinates view shows one axis per dataset. Each line through the axes corresponds to all the measured values for a single node in the graph. The Appearance and Filter and Clustering panels have a variety of filters and controls that affect the appearance of the data.

coordinates views. Each experimental condition maps to a graph view in the small multiples panel and an axis in the parallel coordinates panel. Each node in the graph corresponds to a line in the parallel coordinates view. A node will have an expression value for each experimental condition, determining where the line will cross the axis in the parallel coordinates view, or the colour of the node in the small multiples view. Edge information in the graph is not shown in the parallel coordinates view.

The small multiple windows and parallel coordinate axes support linked reordering through dragging. When a window is dragged to another location the axes are reordered, and vice versa.

To maximize the ability to perceive colour, the nodes in the small multiples view are very large, but this leaves little room for edge information or labels. The main view has a larger screen area to show a graph view in high detail. Nodes are placed in the same relative positions in the main view and all small multiple views.

The nodes in the main view can be coloured in many different ways. The biologist can assign the colours of a small multiple view by clicking on the title of the small multiple view. Alternately, the nodes of the main view can be coloured to show the difference between two experimental conditions as described in Section 6.2.2. Finally, the nodes of the main view can be coloured according to non-numerical biological metadata such as protein function or chromosome location.

The user can navigate across the graph by panning and zooming. All graph views support linked navigation: panning and zooming in one also moves the viewpoint in all of the others. To prevent the user from being lost in the desert fog [29], Cerebral restricts the zoom depth and panning to always keep at least one data point visible in the display window. Cerebral also has shortcuts to frame the view window around selected items or all items. The small multiples are fixed at the same aspect ratio as the main view so that all graphs share the same viewing frame. A slider below the small multiples scales the views to let the user choose the number of views per row. Views that do not fit in the small multiple panel are accessible through a scrollbar.

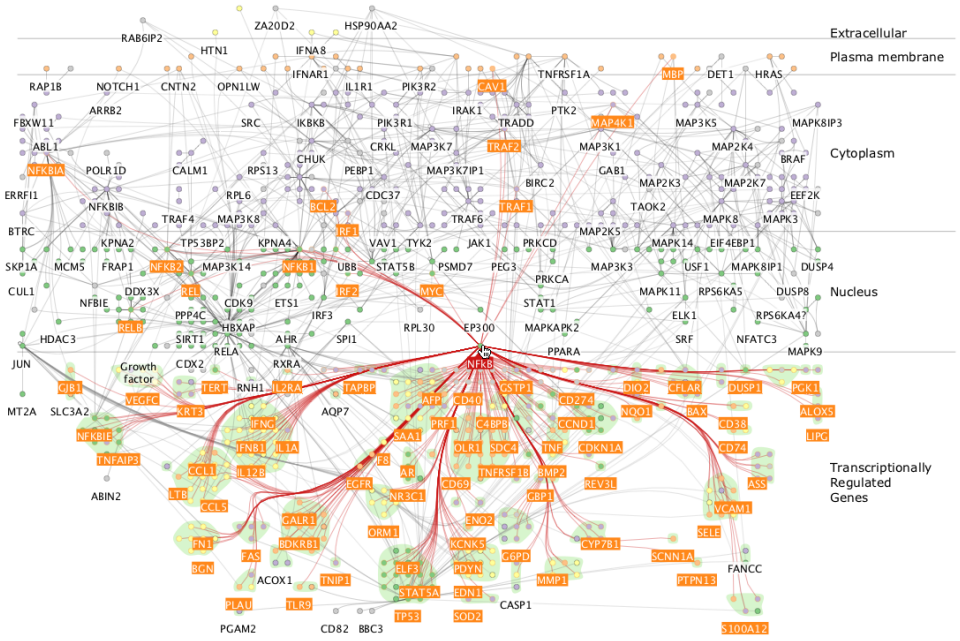


Figure 6.2: The node under the cursor and the node's edges are highlighted in red. First degree neighbours are highlighted in orange.

All views, include the parallel coordinates view, support linked selection and mouseover highlighting. When the user selects items, Cerebral dims the nonselected items so that they are perceived as a background layer. With mouseover highlighting, simply passing the mouse over an item highlights the item in red and all of the graph-theoretic neighbours one hop away are highlighted in orange as shown in Figure 6.2.

6.1.2 Appearance and filters

The control panels allow the user to alter the display of experimental data. Colour scales change the encoding of expression data as described in Sections 6.2.1 and 6.2.2. Dynamic query sliders filter the data as described in Section 6.2.3. Finally, the overall appearance of the graphs can be altered with dynamic sliders as described in Section 6.3.

6.2 Data investigation

Cerebral assists data investigation by visually mapping data values to node colour, computing direct comparison views, filtering subsets of data, and simplifying the display of large quantities of data.

6.2.1 Overlaying data with colour

We overlay expression data or other quantitative measurements on each node of the graph by colouring the node. The appropriate colour scale depends upon the reliability of the source data. Certain technologies provide accurate measurements for which a continuous gradient colour scale is appropriate. Other faster and cheaper technologies will have large error bars associated with each measurement, for which a binning colour scale is more appropriate. Other measurement processing pipelines will include a machine learning algorithm that outputs a simple binary value indicating whether protein level is significantly altered under the experimental condition, for which a simple two-colour scheme is appropriate. To allow maximum flexibility, our colour scale editor shown in Figure 6.3 allows the user to assign measurement value intervals to fixed or gradient colour mappings. To simplify colour scale creation, the editor includes a gallery of useful colour scales chosen using ColorBrewer [22]. We provide a red-green colour scheme traditionally used for gene expression data, a colour blind-friendly orange-purple colour scale, as well as linear and categorical colour scales.

The colour scale is individually applied to each experimental condition in the small multiple views. The user can interactively choose a single experimental condition to show in the larger main display.

6.2.2 Comparing conditions

A common task for biologists is to compare the data from two experimental conditions. For example, the immunologists wanted to compare the response of a cell to bacterial infection in the control case vs. a cell pretreated with an experimental drug. Rather than make comparisons by visually scan-

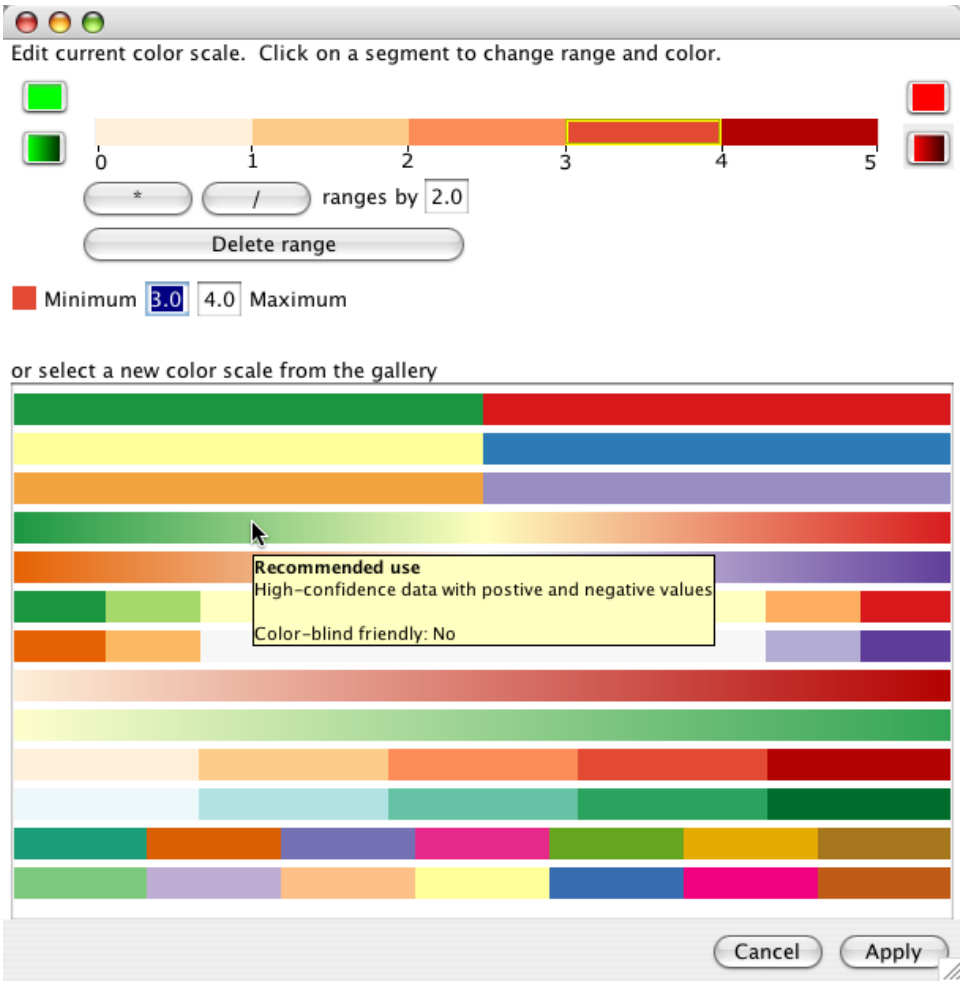


Figure 6.3: The colour scale editor allows the user to assign measurement value intervals to fixed or gradient colour mappings. To simplify colour scale creation, the editor includes a gallery of useful colour scales.

ning back and forth between two small multiple views, we provide a direct computed difference view.

After the user selects two conditions, A and B, the computed difference is shown in the main graph display. If the data measurements are ratio values, then the difference is also shown as a ratio with condition A chosen as the new baseline. The main view colours each node as the ratio of B

versus A computed as $C_x = (B_x - A_x)/|A_x|$. In the case where the data are absolute measurements, then the main view shows each node colour coded by the simple computed difference $C_x = B_x - A_x$. A separate user-defined colour scale is used for the comparison view.

6.2.3 Data filters

Cerebral has multiple methods for selecting a subset of the data. Selected items are brought visually to the foreground, by making all non-selected items translucent. A study of biological insights gained through the use of visualization tools [43] noted that selecting and grouping biomolecules into semantic groups was the most common interaction performed by users. We allow users to add selected biomolecules to user defined groups through a right-click menu. Members of the group can later be selected by clicking the group name in the Cerebral control panel.

Manual selection is the simplest selection mode. Shift-clicking individual nodes (or their corresponding lines in the parallel coordinates view) adds the items to the selection. Clicking the background blob of a biologically related functional group selects all members.

The biologist can apply two broad filters across all experimental conditions, the significance filter and the fold-change filter. The significance filter associates a statistical significance score with each measurement in each condition. The early stages of the data processing pipeline, where physical measurements are made in a wet lab, often generate these associated significance scores. Secondly, many biologists believe there is a minimum amount of chemical change that must occur inside a cell to have an observable effect. The fold-change filter removes data points that do not have a minimum expression level. These two filters are implemented as dynamic query sliders, immediately showing the results of setting the filter level. Other expression analysis systems force the biologist to apply these broad filters early in the analysis, removing all filtered nodes from further consideration. Common by convention, but ultimately arbitrary, values of 0.05 for significance and 1.5 for fold-change are commonly used. Cerebral allows the biologist to

interactively explore these values to quickly alter the balance between the false-positive and false-negative data filtering rate.

The biologist can apply more fine-grained filters by dragging a filter bar in the parallel coordinates display, defining ranges of values the data must satisfy to pass the filter. Figure 6.4 shows a pair of filters that select genes whose expression was highly upregulated in hour 2, but whose value returns to normal in hour 4.

Finally, we provide k-means clustering to find clusters of similar expression values across conditions. The clustering dynamically creates a set of buttons allowing the user to select or deselect all members of a cluster, each with a thumbnail glyph showing the expression profile for that cluster. Our k-clustering algorithm uses normalised expression values and Euclidean distance as the distance measure to find linearly correlated expression patterns. The k value is user selected by a dynamic slider. A standard desktop machine was able to cluster at interactive speeds, allowing the user to easily explore various values of k.

6.3 Visual simplification

Cerebral uses edge bundling and intelligent labelling to reduce the visual complexity of the displayed data. Edge bundling reduces the density of line drawings on the display without filtering, while intelligent labelling lets the user display more nodes simultaneously while keeping legible labels. Finally, we can ease the cognitive load by letting the biologist hand-position familiar nodes in the layout.

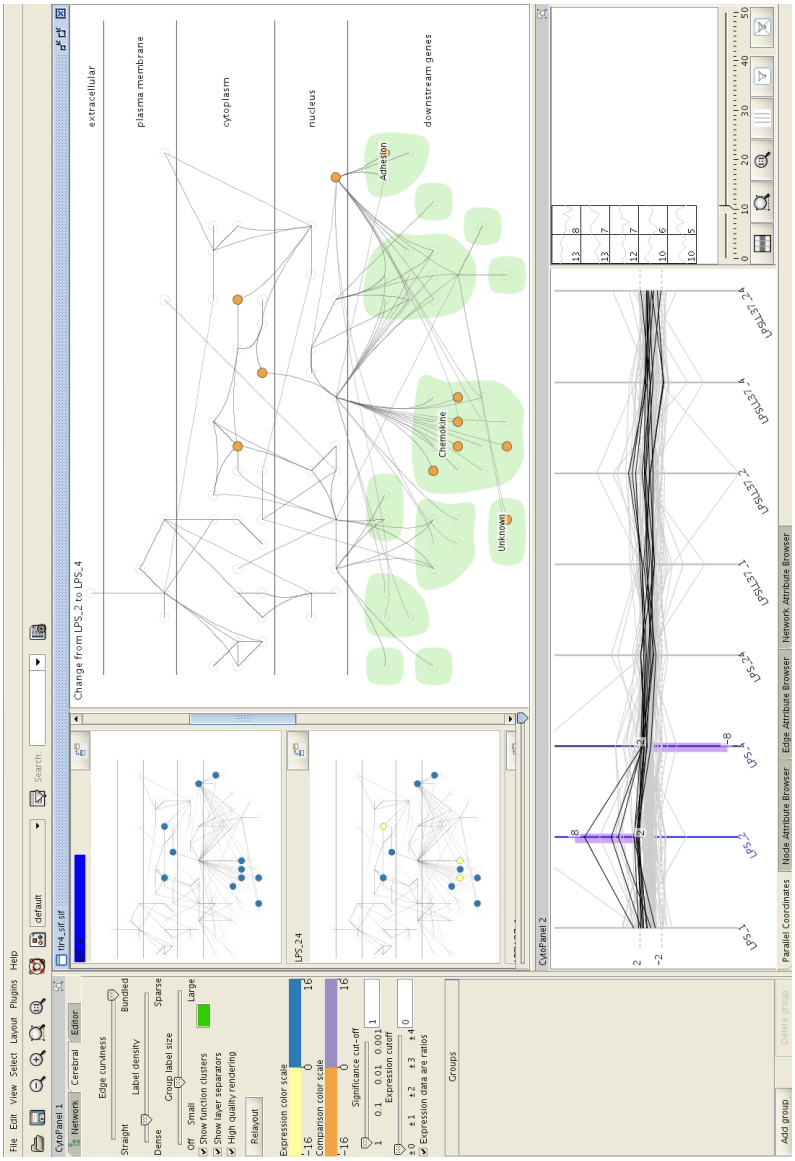


Figure 6.4: Applying bar filters to the LPS_2 and LPS_4 axes selects all biomolecules with a measured value between +2 and +8 in hour 2 and -8 in hour 4. Selection is linked across all views.

6.3.1 Edge bundling

Edge bundling can reduce the visual clutter in a graph by grouping together multiple edges from a source node to many different destination nodes. We were inspired by the work of Holten [27] on bundling edges using splines. In his approach, edges were bundled according to an auxiliary tree hierarchy accompanying the graph. We do not use bundling to reveal a secondary hierarchy, but instead propose an algorithm that uses the geometric layout of the graph itself to determine edge groupings and spline control point placement. Our approach is analogous to cable routing, where cables going to the same area are tied together into a single thick cable. All edges in a bundle share a common endpoint, so that the user does not have to mentally untangle the bundle. Our algorithm has two stages: first we form groups of nodes based on proximity in the grid, then we form higher-level connections between these groups based on the angle between them, creating branch-points whose appearance is reminiscent of a dividing river [38]. Figure 6.5 gives the pseudocode for our bundling algorithm, and Figure 6.6 shows a small example.

The first stage begins by sorting all the nodes by the number of incident edges, and works from the highest degree node down to the lowest degree. We attempt to assign all incident edges of a node to a bundle. After being bundled, an edge is never examined again. We make three passes. In the first, we merge into a group those nodes that are positioned directly next to each other in the layout. Next, we merge those that are two grid cells apart, and finally those three cells apart. After examining all nodes, each unassigned edge is assigned to a degenerate bundle containing the single edge.

In the second stage, we bundle groups whose edges travel in the same direction; specifically, those whose incident angle to the node are within $\frac{\pi}{8}$ radians of each other.

The bundle of edges is drawn with piecewise cube B-splines. Each spline has a control point at the start and end node of an edge. All edges within a bundle share the same intermediate control points. The control points

```
BUNDLE(nodes)
    sortedNodes ← Sort nodes by degree
    {completedEdges} ← ∅

    ▷ Stage 1
    for maxSeparation ← 1 to 3
        for each n ∈sortedNodes
                {proximityGroup} = ∅
                {edgesToExamine} = n.edges − {completedEdges}
                for each pair of edges (A, B) ∈ {edgesToExamine}
                   if Separation(A.target, B.target) < maxSeparation
                         then {proximityGroup} ∩ = A ∩ B
                                 {completedEdges} ∩ = A ∩ B
                n.bundles.add (proximityGroup)

    Assign each unbundled edge in allEdges - {completedEdges}
    to its own group

    ▷ Stage 2
    for each n ∈ sortedNodes
        for each pair of groups (G, H) ∈ n.bundles
                if Angle(G,H) < π/8
                   then MergeInteriorSplinePoints (G,H)
```

Figure 6.5: Pseudocode for edge bundling

are placed on a straight line between the high degree-node and the centre of mass of the bundle, just outside the bundle and immediately outside the high degree node. When spline control points are merged, the group that is furthest from the high degree node gains all the intermediate control points of the other group.

We note that the groups used behind the scenes for edge bundling are not necessarily identical to the functional groups that provide soft constraints on node proximity, visible in the display as coloured regions. However, they are often very similar, because the constraints guide the nodes to be positioned

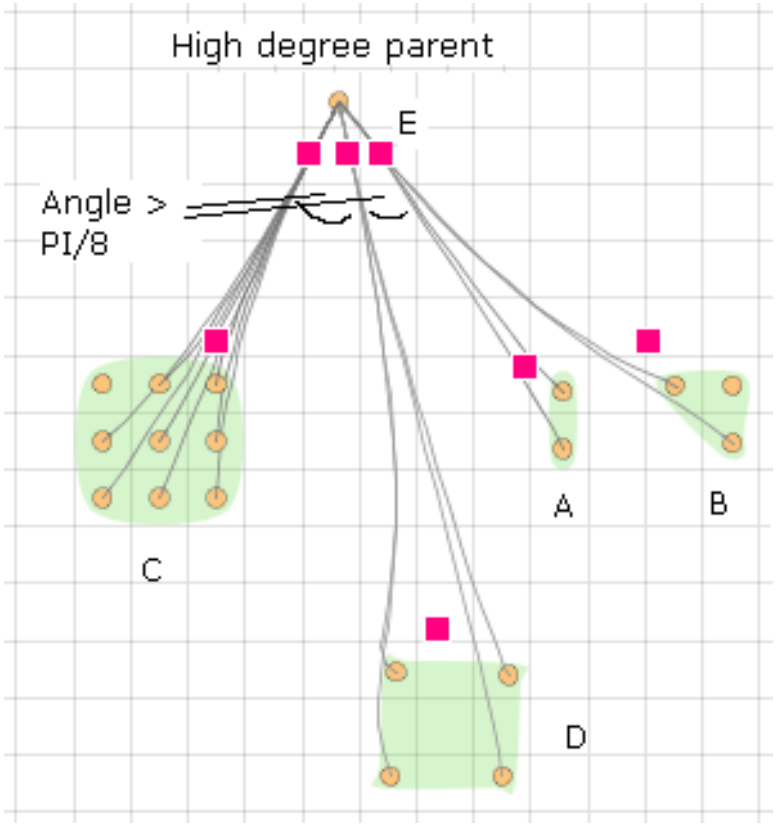


Figure 6.6: Edge bundling changes the shape of edges travelling in the same direction according to a spline. Here, the control points are shown explicitly as pink squares, although they are not visible in Cerebral itself. Edge bundling has a first stage where groups of nodes that are graph-theoretic neighbours are formed at successive grid distances from a high-degree base node. Groups A,B, and C were all found in the 1-cell pass. Group D is formed in the 2-cell distant pass. Groups A and B are bundled in the second stage of the algorithm, so they share the spline control point labelled E.

close enough together that their edges are bundled.

6.3.2 Intelligent labelling

Labelling is a problem in many biological graph exploration tools. Two common approaches are to draw all labels, resulting in numerous overlapping

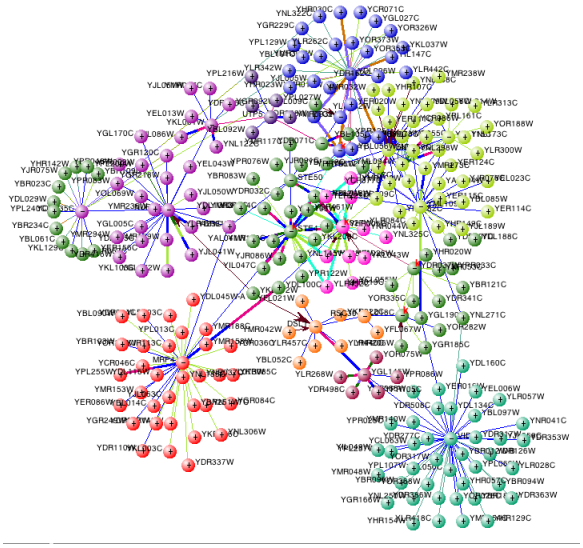


Figure 6.7: Drawing all labels on a graph, as is done in the VisANT [28] software, results in numerous overlapping illegible labels.

illegible labels, such as the VisANT [28] display shown in Figure 6.7, or to draw the labels in world space resulting in labels that are too small to read until the user zooms deeply into the graph, such as the default Cytoscape [45] renderer shown in Figure 6.8.

We draw labels with an algorithm that guarantees that labels do not overlap at any zoom level, and the density of labels is interactively controllable via a slider. Labels are always legible: they are drawn at a fixed size in screen space, so their size varies in world space in accordance with the zoom level. We use a greedy algorithm to draw labels, using a bitmap to keep track of occupied screen space, and only draw a label if its bounding box does not intersect any previously drawn one. We draw the label under the cursor first, then its graph-theoretic neighbours, then any nodes selected by the user, and then traverse the list of all nodes sorted by degree.

6.3.3 Pinning nodes

Our user interface allows users to interactively drag nodes to override the automatic layout. Once the nodes are positioned, the user can pin selected

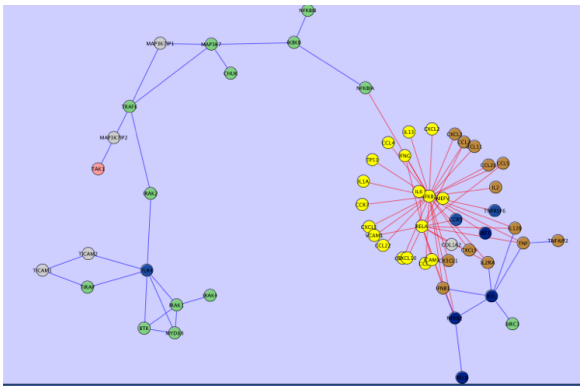


Figure 6.8: Drawing in world space, as is done by the standard Cytoscape [45] renderer, results in labels that are too small to read except at deep zooming levels.

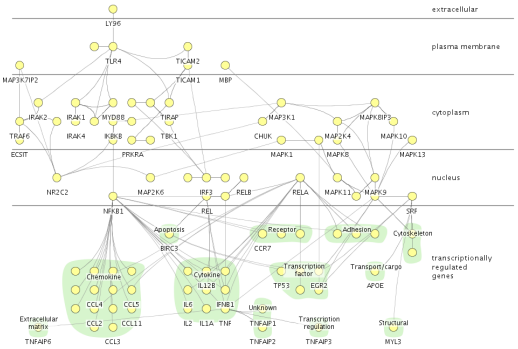
nodes to their current location. Pinning allows the biologist to create a hand-drawn skeleton of key nodes in a graph. Additionally, when more nodes are added to the graph when the model is refined, the user can pin the old nodes and apply the layout algorithm with only the new nodes mobile. Pinned nodes are added to the uniform grid and considered during the energy evaluation, but they are never selected as candidates to be moved by the perturbation function. Figure 6.9 shows the process of pinning key nodes and then using the layout algorithm to position the remaining free nodes.

6.4 Implementation

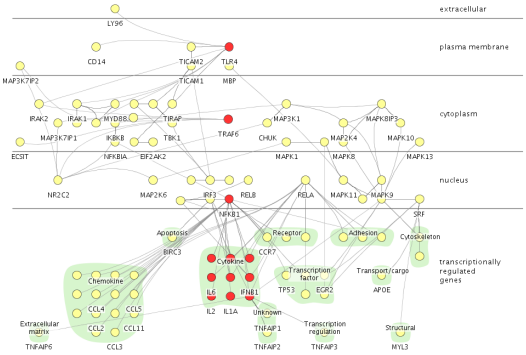
Cerebral is an interactive system implemented in Java 1.4.2 as a plugin for Cytoscape [45]. Cytoscape is a popular biomolecular graph editor in the biology community, which loads graphs and metadata from several standard biology file formats. The Cytoscape framework allows mapping node and edge metadata to visual appearance rules such as shape, size, and colour. We replaced the standard Cytoscape renderer with our own implemented using the Prefuse toolkit [26]. Prefuse provides a framework for managing nodes and edges, displaying nodes, and responding to user events. We use

its convex hull outlining capability to surround the functional groups in the bottom layer with coloured blobs. We wrote custom code for node layout, edge bundling, edge rendering, and label positioning.

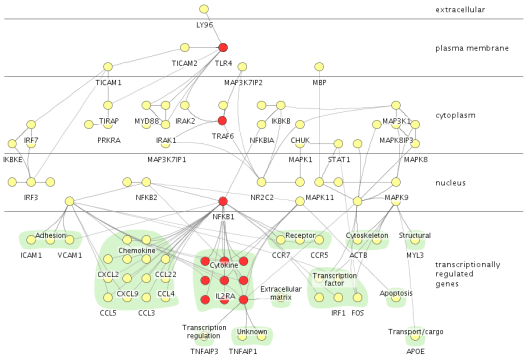
Integration with Cytoscape allows Cerebral users to take advantage of the large community of Cytoscape plugin developers. For example, the Enhanced Search plugin lets biologists select nodes with a simple query language on node metadata, and BinGO [35] creates clusters of nodes by testing for over represented gene ontology terms in the datasets.



(a)



(b)



(c)

Figure 6.9: **a)** The original layout is shown in yellow. **b)** The biologist hand-positions key molecules in the pathway (TLR4, TRAF6, NFKB1, and the cytokines) and pins them in place. **c)** The layout algorithm is rerun to optimize the remaining nodes, leaving the pinned red nodes in place.

Chapter 7

Sample sessions

We present two sample sessions using Cerebral to analyse microarray experimental data in the context of a biomolecular interaction graph. We begin with an immunology-specific dataset, and then move on to a more general cellular time series dataset.

7.1 Immune response and LL-37

We now return to the LL-37/TLR4 dataset previously published by our immunologist collaborators. Figure 7.1 shows the data displayed with Cerebral, which performs an automatic layout exploiting the cell localisation data, and displays an overview of all the datasets in the small multiple views. The outcomes of the cascade are shown at the bottom of the diagram grouped by previously known function and visually distinguished with a light green outline. As the data was gathered with microarrays, which have high error ranges associated with each measurement, a binned green-yellow-red colour scale was used for the small multiples, and a binned orange-grey-purple colour scale for the comparison view. The yellow and grey colours reduce the visual impact of genes whose expression level has changed by less than 50 percent.

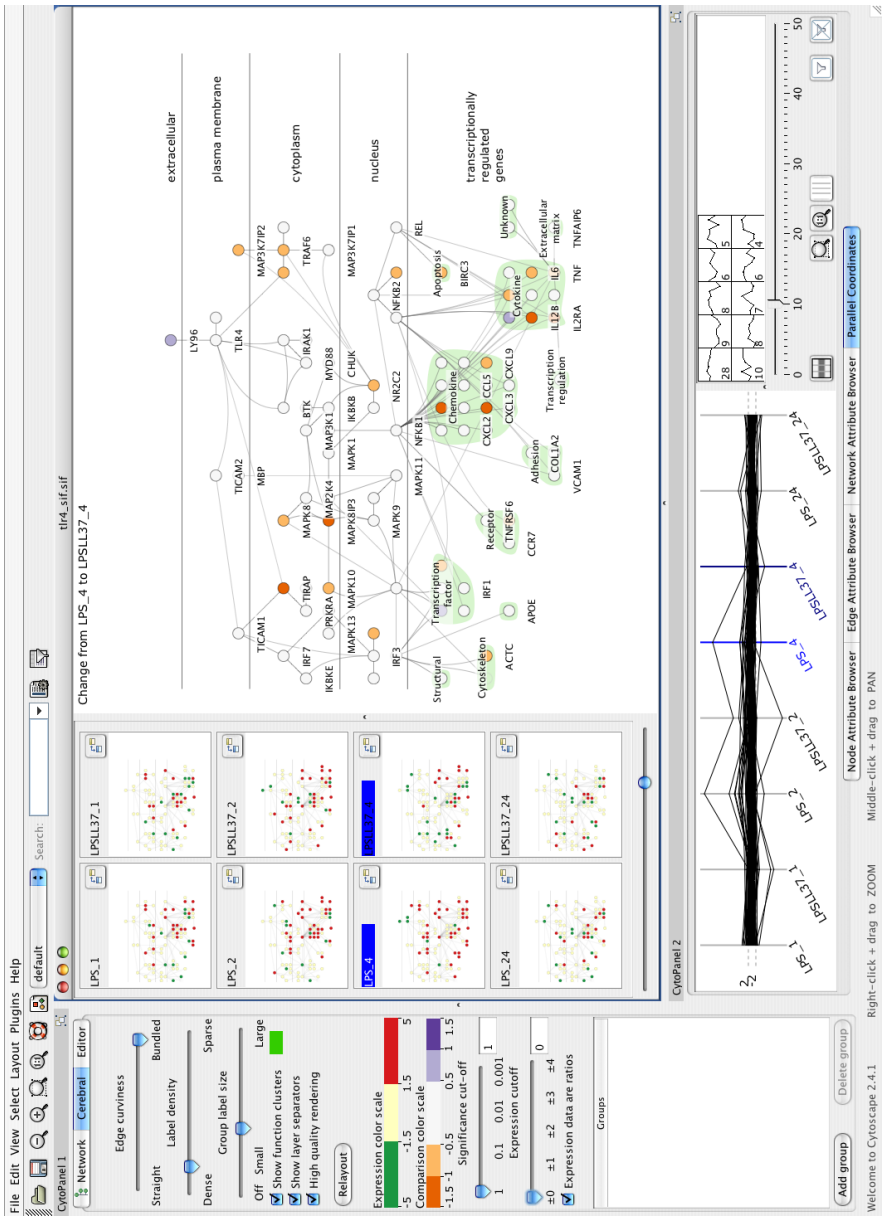


Figure 7.1: The Cerebral display of the TLR4 graph (V=91, E=124) with associated LPS experiments.

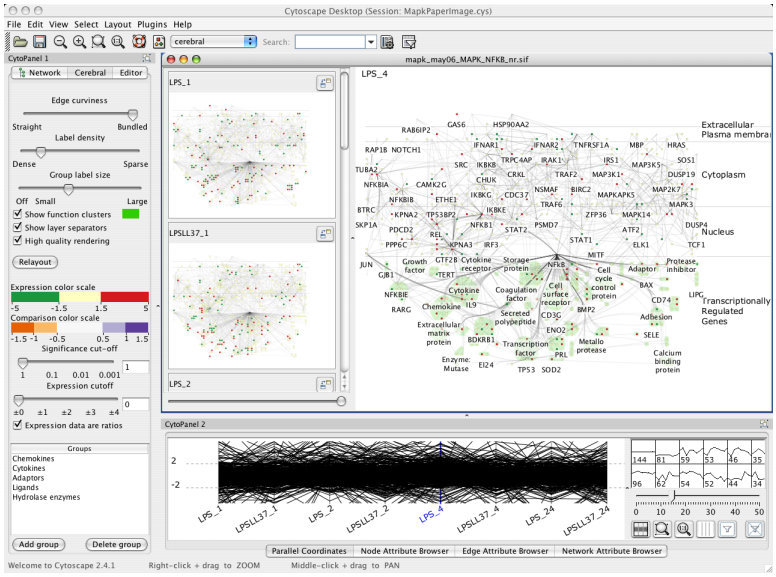
The biologist reorders the small multiples to place each LL-37-treated time point beside the untreated condition, allowing the user to quickly spot how the peptide affects the cell's response to bacterial infection. Scanning these pairs, we see that the differences in the two conditions are most pronounced in the third row at hour 4. We note there are many more nodes coloured solid green in the LPSLL-37_4 condition than the untreated LPS_4 condition. Rather than scanning back and forth between the two conditions to search for differences, we have Cerebral compute the direct difference between the two hour-4 conditions.

Most of the nodes in the difference view are coloured orange, indicating that the expression levels are reduced in the presence of LL-37. Furthermore, it is easy to see that the response proteins that show such changes primarily belong to the cytokine and chemokine functional categories. Many of these are implicated in the harmful inflammatory side effects of the immune response to bacterial infection, thus it appears that the protective LL-37 molecule is working to minimize these side effects.

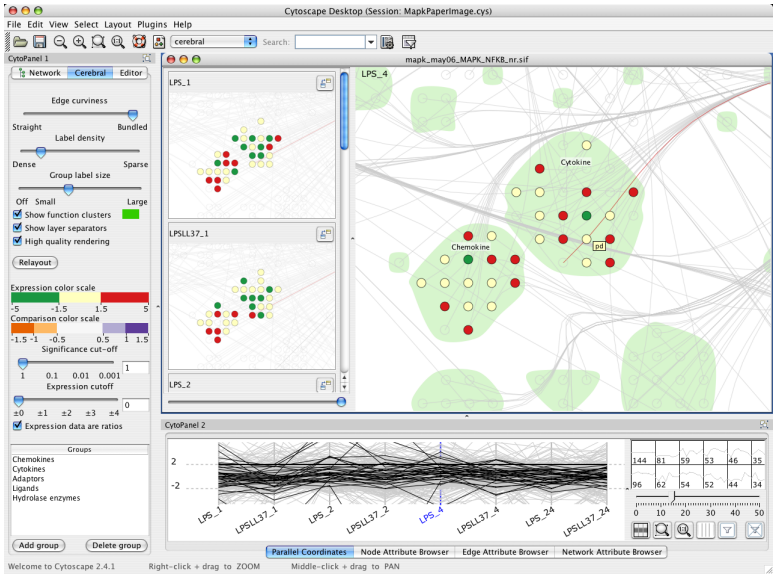
Recreating the same data views as the original paper took seconds in Cerebral, compared with many hours of manual manipulations. Moreover, the automatic layout and interactive exploration capabilities of Cerebral allows the immunologists to productively use more complex and complete graph models. Figure 7.2 shows the same LPS gene expression experiment data overlaid on the MAPK model graph, a superset of the TLR4 graph containing 1269 edges and 760 nodes. While the overview of the complete graph is quite complex, the interactive selection and navigation allow the immunologists to easily explore across multiple time points.

7.2 Yeast cell cycle: time series analysis

This example uses publicly available gene expression data from a time series study of the cell cycle of yeast published by Spellman et al. [46]. This expression data was combined with a yeast cell cycle protein interaction graph constructed by de Lichtenberg et al. [12] and available from the Cell Circuits database [36]. Cellular location information was obtained from the



(a)



(b)

Figure 7.2: **a)** The experimental LPS data shown in the context of the more complete and complex MAPK graph (V=760, E=1269). **b)** Interactive selection, panning, and zooming shows the expression of cytokines and chemokines across time points in the context of the larger MAPK graph.

Yeast Protein Localization Server [14]. Data from these sources were then combined to form an integrated dataset suitable for analysis by Cerebral. Though the Spellman study investigated how gene expression varied according to cell cycle phases, they did not examine the data in the context of a biological network. This example illustrates that the network context shown with Cerebral allows clear and fast detection of correlated expression changes.

Figure 7.3 shows the initial Cerebral display generated when we first load the data. We simultaneously view the entire set of expression data across all time points using the parallel coordinate display as well as the small multiples display. The automated Cerebral layout of protein interactions is arranged by cellular location as well as by interactions. We have clicked in the `cdc050` small multiple window, so the main view shows the colouring for that condition.

As we are not starting with a particular hypothesis, we begin exploration by using the data-driven parallel coordinates view. We interactively adjust the k-means slider until we see an interesting correlation pattern in the cluster glyphs. We click a sinusoidal pattern in the lower-right thumbnail cluster buttons, since this pattern is suggestive of cell cycle phases. In Figure 7.4 the parallel coordinate display clearly shows the sinusoidal nature of the selected cluster across the time series, indicative of its involvement in the phases of the cell cycle.

In fact, we see that six of the molecules in this cluster correspond to histone genes [48]. The graph view quickly and clearly conveys the fact that these molecules interact with one another and are active in the nucleus. Figure 7.5 shows the result of selecting this smaller subset for closer inspection by dragging out a box in the main view. We now see even more striking correlated behaviour in the parallel coordinates plot. Viewing the small multiples, we can scan the time series in the graph context and see how the cluster shifts as a unit from showing green under-expression to red over-expression. A trained biologist will immediately recognize this pattern as following the cell cycle phases. We note that while the parallel coordinate view can represent this correlated behaviour, it cannot show the relationship

between the proteins. The small multiple graph views show such correlation qualitatively, but they do not present as precise an analytical display as the parallel coordinate plot. By coordinating both views simultaneously, the user can visualize and compare views for more complete analysis.

This analysis session shows how the simple interaction graph could be extended to include temporal effects. We see how histone levels rise and fall in time with the progression of the yeast cell cycle.

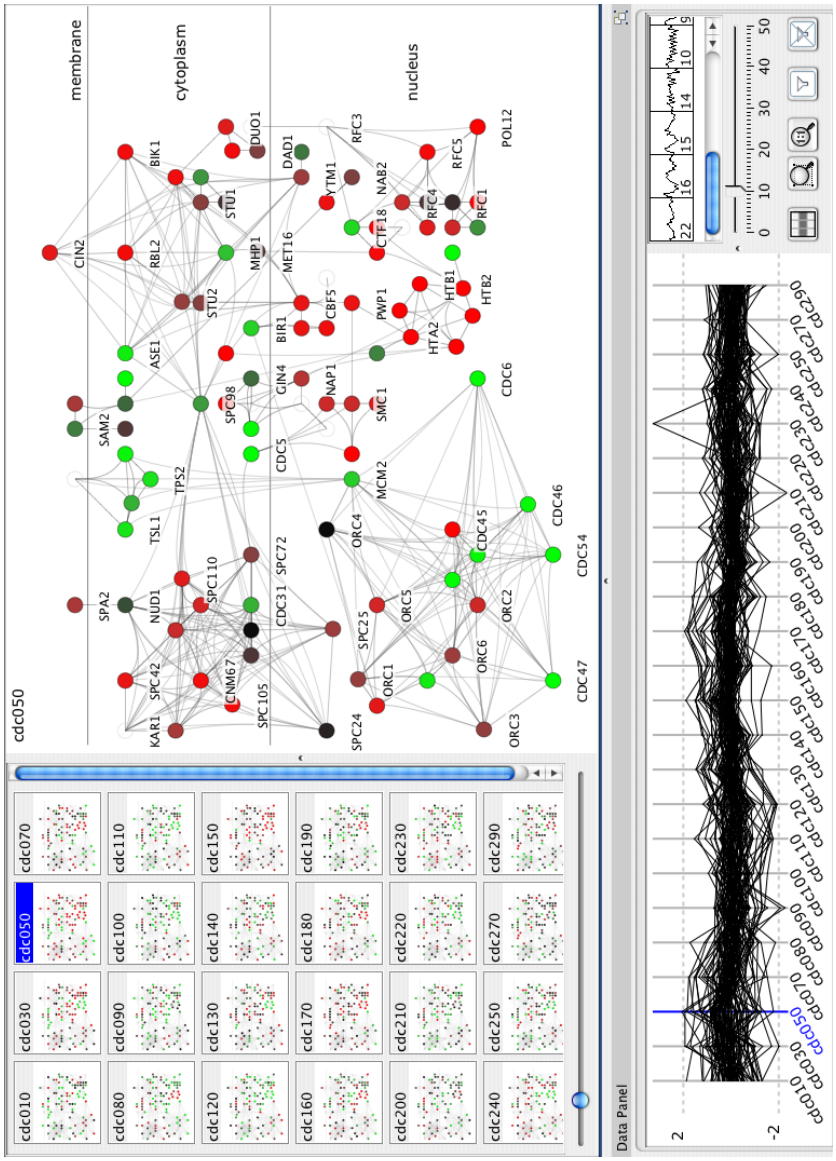


Figure 7.3: Initial Cerebral layout of the yeast cell cycle data (V=104, E=417) shows expression data in a subcellular localized graph context.

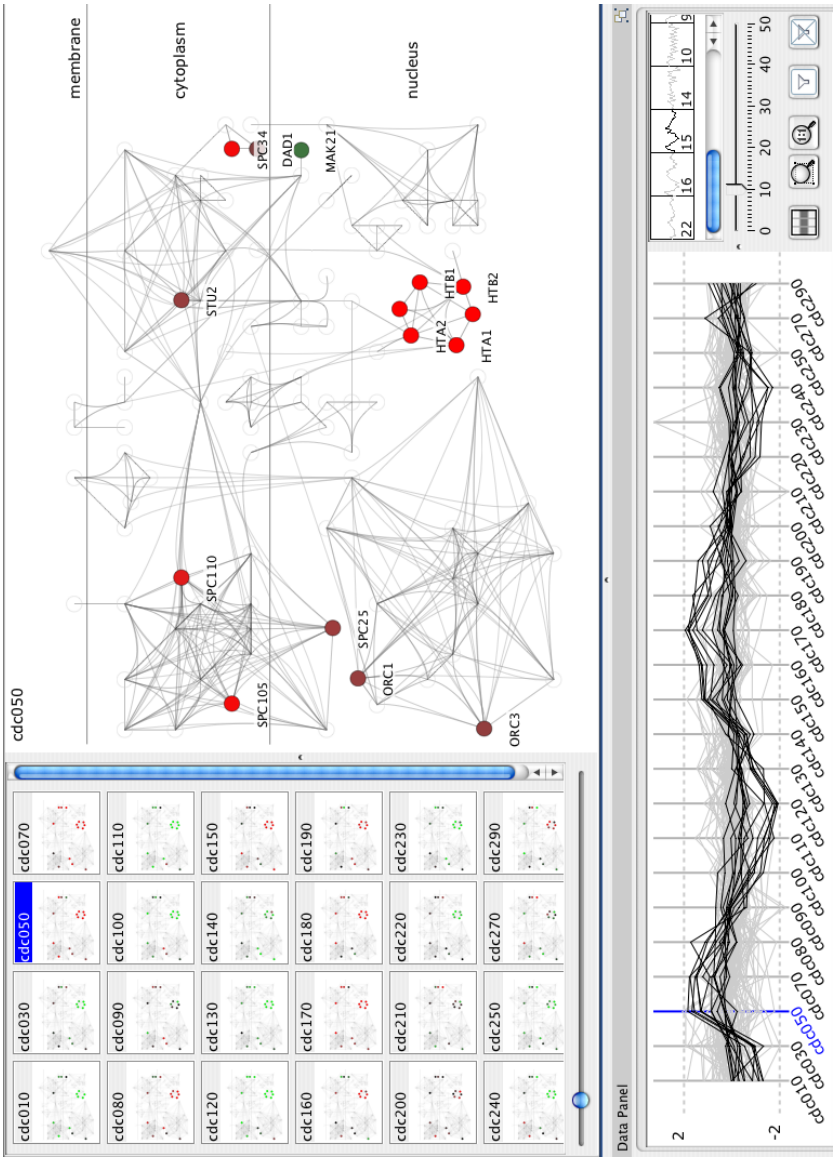


Figure 7.4: We set the k-means slider to 10 clusters, and select a cluster of 15 genes that show cyclic behaviour.

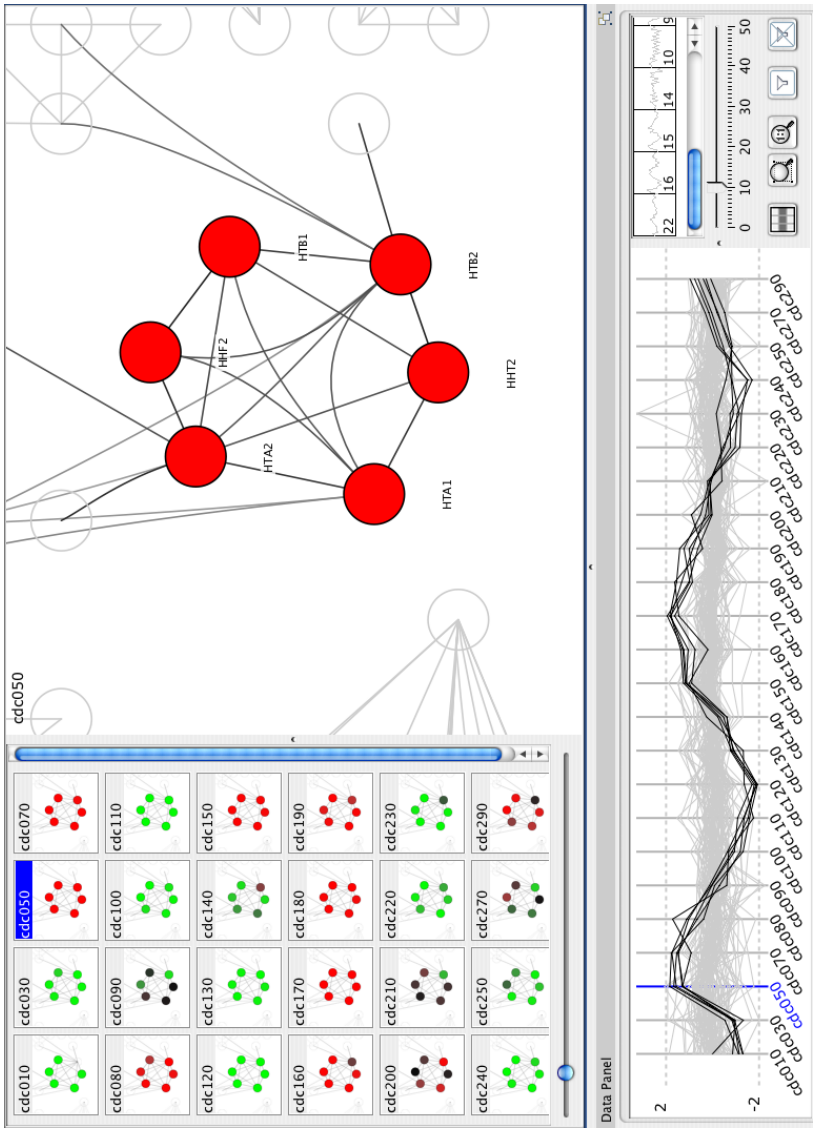


Figure 7.5: Selecting the interrelated histone group (HTA1, HTA2, HTB1, HTB2, HHF2, HHT2) by dragging in the main view shows a clearer pattern of early under-expression in green, then later over-expression in red, in the small multiple views.

7.3 User response

We gathered feedback from our immunologist collaborators on the design of Cerebral as they used a succession of interactive prototypes. We began by attacking the problem of biologically-guided layout, with a prototype that had only a single interactive graph view. After a period of refinement, we made this early single-view version of Cerebral publicly available in February 2007. The response from the bioinformatics community was very encouraging: we have heard from many groups who have used it, and three published biology papers include figures created with Cerebral. One is from our collaborators [9], and two are by other researchers [16, 23]. We note that one of the latter [23] explicitly mentions Cerebral in the methods section of the paper, showing that it was considered integral to their analysis methodology as opposed to simply being used for presentation.

We then continued on to the problem of handling multiple experimental conditions, again refining the design through feedback on prototypes. The final version of Cerebral tool documented in this thesis has been enthusiastically embraced by our immunology collaborators. They have integrated Cerebral as the visualization front end for InnateDB, their hand-curated immunology interaction database. This version was made publicly available as Cerebral v.2.0 in May 2008 at http://www.pathogenomics.ca/cerebral

Chapter 8

Conclusions and future work

We have shown that overlaying experimental data on an interaction graph with Cerebral provides systems biologists an opportunity to evaluate the current biological system model, generate hypotheses, and improve and refine the model. Cerebral has data displays customised for systems biology tasks, providing an environment where the data views are familiar and the graph nodes appear in biologically sensible locations. By creating biologically meaningful graph layouts automatically, systems biologists are now able to work with much larger and more complete graphs. Interactive simultaneous viewing of multiple experimental conditions allows for more in depth analysis of gathered data.

Some cell compartments do not follow a linear hierarchy; for instance, mitochondria and Golgi bodies are both found in the cytoplasm. Building an input system that supports subregions within layers for organelles, and circular regions for bacterial cells, should be a straightforward extension of our underlying layout algorithm.

The scoring for the layout algorithm currently chooses between revealing edge relations between nodes, which is done in most layers, or tightly grouping nodes related by biological function, as is done in the bottom layer. It would be interesting to look for a set of constraints and weightings that allow enough space and freedom between nodes to reveal edge relations, while keeping functionally related nodes within a convex hull.

While the expression data explored by the biologist is frequently created within the lab, the underlying graph model is usually gathered from databases summarizing many different experiments with varying levels of trustworthiness. It would help the biologist to include hyperlinks from Cerebral back to the original sources of the biological models and metadata.

Cerebral allows a biologist to search through experimental data overlaid on current models in the hopes of producing a new hypothesis about how biological systems function. As the biologist is examining the dataset without a hypothesis, the data mining bias may cause random data to appear interesting or significant. It may be helpful to provide a statistical test to indicate whether a discovered relationship in the graph and experimental data is statistically significant given the size of the graph and the number of measured data points.

Our k-clustering algorithm has been effective at finding ad-hoc patterns in suitable data sets such as time series experiments. However, this relatively simple technique could be replaced with more modern clustering and classification methods. Algorithms that incorporate biological information to inform the ordering or partitioning of the experimental conditions would a particularly interesting area to explore.

As high throughput biomolecule measurement technologies become more scalable and cost effective, biologists will include increasing numbers of experimental conditions in their study design. The small multiple views interactively display up to a few dozen conditions with a few thousand nodes. In order to support these larger study designs, we would need to improve both the visual scalability and the computational performance of the system.

Bibliography

[1] Christopher Ahlberg, Christopher Williamson, and Ben Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 619–626, 1992.

[2] Varol Akman, Wm Randolph Franklin, Mohan Kankanhalli, and Chandrasekhar Narayanaswami. Geometric computing and the uniform grid data technique. *Computer Aided Design*, 21(7):410–420, 1989.

[3] David Auber. Tulip : A huge graph visualization framework. In P. Mutzel and M. Jünger, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 105–126. Springer-Verlag, 2003.

[4] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings Advanced Visual Interface*, pages 110–119, 2000.

[5] Michael Balzer and Oliver Deussen. Level-of-detail visualization of clustered graph layouts. *International Asia-Pacific Symposium on Visualization*, pages 133–140, 2007.

[6] Jon Bentley and Thomas Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transcations on Computers*, C-28:643–647, 1979.

[7] Bobby-Joe Breitkreutz, Chris Stark, and Mike Tyers. Osprey: a network visualization system. *Genome Biology*, 4(3):R22, 2003.

[8] Jack E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[9] Kelly L. Brown, Céline Cosseau, Jennifer L. Gardy, and Robert E.W. Hancock. Complexities of targeting innate immunity to treat infection. *Trends in Immunology*, 28(6):260–266, 2007.

[10] Kam Dahlquist et al. GenMAPP, a new tool for viewing and analyzing microarray data on biological pathways. *Nature Genetics*, 31:19–20, 2002.

[11] Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.

[12] Ulrik de Lichtenberg, Lars Juhl Jensen, Soren Brunak, and Peer Bork. Dynamic complex formation during the yeast cell cycle. *Science*, 307(5710):724–727, 2005.

[13] Emik Demir et al. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, 2002.

[14] Amar Drawid and Mark Gerstein. A Bayesian system integrating expression data with sequence patterns for localizing proteins: comprehensive application to the yeast genome. *Journal of Molecular Biology*, 301:1059–1075, 2000.

[15] Tim Dwyer and Kim Marriott. IPSep-CoLa: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Information Visualization*, 12(5):821–828, 2006.

[16] Matthew D Dyer, T. M Murali, and Bruno W Sobral. The landscape of human proteins interacting with viruses and other pathogens. *PLoS Pathog*, 4(2):e32, 2008.

[17] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[18] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of the DIMACS International Workshop on Graph Drawing*, pages 388–403, 1994.

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

[20] Burkay Genc and Ugur Dogrusoz. A constrained, force-directed layout algorithm for biological pathways. In *Proceedings Graph Drawing*, pages 314–319, 2004.

[21] Silicon Genetics. GeneSpring. `www.silicongenetics.com`.

[22] Mark A. Harrower and Cynthia A. Brewer. ColorBrewer.org: An online tool for selecting color schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

[23] Liqun He et al. The glomerular transcriptome and a predicted protein-protein interaction network. *Journal of the American Society of Nephrology*, 19(2):260–268, 2008.

[24] Weiqing He and Kim Marriott. Constrained graph layout. *Constraints*, 3(4):289–314, 1998.

[25] Christopher G. Healey, Kellogg S. Booth, and James T. Enns. High-speed visual estimation using preattentive processing. *ACM Transactions on Human Computer Interaction*, 3(2):107–134, 1996.

[26] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 421–430, 2005.

[27] Danny Holten. Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[28] Zhenjun Hu, Joseph Mellor, Jie Wu, and Charles DeLisi. VisANT: an online visualization and analysis tool for biological interaction data. *BMC Bioinformatics*, 5(1):17, 2004.

[29] Susanne Jul and George W. Furnas. Critical zones in desert fog: aids to multiscale navigation. In *Proceedings of the ACM symposium on User Interface Software and Technology*, pages 97–106, New York, NY, USA, 1998.

[30] Mitsuru Kato, Masao Nagasaki, Atsushi Doi, and Satoru Miyano. Automatic drawing of biological networks using cross cost and subcomponent data. *Genome Informatics*, 16(2):22–31, 2005.

[31] Michael Kaufmann and Dorothea Wagner, editors. *Drawing graphs: methods and models*. Springer-Verlag, London, UK, 2001.

[32] Kaname Kojima, Masao Nagasaki, Euna Jeong, Mitsuru Kato, and Satoru Miyano. An efficient grid layout algorithm for biological networks utilizing various biological attributes. *BMC Bioinformatics*, 8:76–92, 2007.

[33] Joseph B. Kruskal and Judith Seery. Designing network diagrams. *Proceedings of the First General Conference on Social Graphics*, pages 22–50, 1980.

[34] Weijiang Li and Hiroyuki Kurata. A grid layout algorithm for automatic drawing of biochemical networks. *Bioinformatics*, 21(9):2036–2042, May 2005.

[35] Steven Maere, Karel Heymans, and Martin Kuiper. BiNGO: a Cytoscape plugin to assess overrepresentation of gene ontology categories in biological networks. *Bioinformatics*, 21:3448–3449, 2005.

[36] H. Craig Mak, Mike Daly, Bianca Gruebel, and Trey Ideker. CellCircuits: a database of protein network models. *Nucleic Acids Research*, 35:D538–D545, 2007. `http://www.cellcircuits.org/`.

[37] Neeloffer Mookherjee et al. Modulation of the Toll-like receptor-mediated inflammatory response by the endogenous human host defence peptide LL-37. *Journal of Immunology*, 176:2455–2464, 2006.

[38] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan, and Terry Winograd. Flow map layout. In *IEEE Symposium on Information Visualization*, pages 29–37, 2005.

[39] Matthew Plumlee and Colin Ware. Zooming versus multiple window interfaces: Cognitive costs of visual comparisons. *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 13(2):179–209, 2006.

[40] Helen C. Purchase, David Carrington, and Jo-Anne Alder. Empirical evaluation of aesthetics-based graph layout. *Empirical Software Engineering*, 7(3):255–255, Sep 2002.

[41] Jonathan C. Roberts. State of the art: Coordinated and multiple views in exploratory visualization. In *Proceedings of Coordinated and Multiple Views in Exploratory Visualization*, pages 61–71, 2007.

[42] Purvi Saraiya, Peter Lee, and Chris North. Visualization of graphs with associated timeseries data. In *IEEE Symposium on Information Visualization*, pages 225–232, 2005.

[43] Purvi Saraiya, Chris North, and Karen Duca. An insight-based methodology for evaluating bioinformatics visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):443–456, July 2005.

[44] Jinwook Seo and Ben Shneiderman. Interactively exploring hierarchical clustering results. *IEEE Computer*, 35(7):80–86, 2002.

[45] Paul Shannon et al. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13:2498–2504, 2003.

[46] Paul Spellman et al. Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.

[47] Matthew Suderman and Michael Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, 2007.

[48] Ann Sutton, Jean Bucaria, Mary Ann Osley, and Rolf Sternglanz. Yeast ASF1 protein is required for cell cycle regulation of histone gene transcription. *Genetics*, 158:587–596, 2001.

[49] Edward Tufte. *Envisioning information*. Graphics Press, 1990.

[50] Daniel Tunkelang. A practical approach to drawing undirected graphs. Technical Report CMU-CS-94-161, Carnegie Mellon University Department of Computer Science, June 1994.

[51] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: can it facilitate? *International Journal of Human-Computer Studies*, 57(4):247–262, 2002.

[52] Michel Westenberg et al. Interactive visualization of gene regulatory networks with associated gene expression time series data. In *Visualization in Medicine and Life Sciences, Visualization and Mathematics*, pages 293–312, 2008.

[53] Ka Yeung, Mario Medvedovic, and Roger Bumgarner. Clustering gene-expression data with repeated measurements. *Genome Biology*, 4(5):R34, 2003.

Appendix A

Raw timing data

Raw data from our layout timings.

Dataset	Time (seconds)									
TLR4	4.0	4.0	4.0	3.9	4.0	4.0	4..0	4.0	4.0	4.0
LL37 experiment	6.1	6.5	6.0	6.0	6.0	6.0	5.9	6.0	6.0	6.0
Yeast cell cycle data	14.5	14.2	14.1	14.1	14.2	14.1	14.1	14.1	14.1	14.2
IRAK4	29.5	29.3	29.4	30.0	29.5	29.5	29.5	29.3	30.0	29.8
MAPK	62.7	61.6	62.6	62.1	61.3	60.7	62.0	61.6	62.8	61.4
Human interactome	6282	6091	6128	6296	6183	6188	6360	6034	6384	6040

Table A.1: The raw timings for 10 consecutive runs as reported in the graph layout section of the thesis.