Graphite: A Suite of Hygraph Visualization Utilities

Emanuel G. Noik (noik@db.toronto.edu)
Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A1

Abstract

This document provides a high level overview of the functionality realized by a set of C++ programs that comprise the graphite utilities. The current package supports general and directed graphs and hygraphs (a nested graph formalism) and includes programs that: draw graphs, generate several types of fisheye views and other visual abstractions, convert graphs between several external formats, and perform simple graph manipulations. In addition, in citing nearly 40 publications in the area of graph visualization, this document also serves as a short annotated bibliography.

1 Introduction

The graph topovisual formalism [20] has been used in numerous systems to represent and visualize relational information. Well-known systems that employ graphs include: SemNet [15], GMB [23], Adapt [33], MView and LaSSIE [3, 9], 4Thought [7, 35], G+/Hy+ [6, 8], and others [24]. A common set of requirements for graph visualization systems has been outlined [31].

This document describes the programs that comprise the graphite utilities: a software package implemented by the author. The current implementation is realized by approximately 15000 lines of C++ [14] source code, and can be ported to most Unix environments. The package consists of three main programs, each a Unix filter: a program that obtains its input from the standard input stream (stdin) and writes its output to the standard output stream (stdout); errors are directed to the standard error stream (stderr). A graph layout filter, for example, reads a description of a graph from stdin, internally modifies the position and size information of various graph elements in accordance to particular layout criteria and writes the description of the modified graph to stdout.

2 Preliminaries

2.1 Graph layout

The standard 2D graph layout problem can be divided into roughly two subproblems: *vertex layout* and *arc layout*. Given a graph $\langle V, A\rangle$, where V is a set of vertices and $A \subseteq V \times V$ is a set of arcs, vertex layout assigns a Cartesian coordinate (x, y) to each $v \in V$. Arc layout is used to draw multi-segment arcs, by assigning a (possibly empty) set of middle points ((x, y) coordinates) for each $a \in A$; an arc can then be drawn as a set of straight line segments or as a spline curve with the middle points as control points. In general, the quality of a layout is directly related to the amount of time that an algorithm is given to compute it. Accordingly, most graph layout algorithms can be divided into two groups: algorithms that emphasize performance and algorithms that emphasize layout quality. Since graphite has been targeted primarily for use in interactive computer applications, it has embodied algorithms that predominantly belong to the first category. The current version of graphite supports vertex layout only – it does not perform arc layout.

In their survey paper, Eades and Tamassia discussed a number of graph drawing approaches; each approach can be characterized by the classes of graphs that it can be used to draw and the layout conventions that it implements [12]. Common classes of graphs that have been drawn automatically include: rooted and free trees, directed acyclic graphs, planar graphs, and general directed and undirected graphs.

With the advent of powerful graphics workstations and stereoscopic displays, 3D graph layout has been receiving increased attention. A number of systems that employ 3D graph layout have been documented in the literature [4, 15, 26, 34, 35].

2.2 Composite graph layout

Informally, composite graph layout refers to a graph layout technique in which the input graph is partitioned into several subgraphs which are laid-out independently (most often by different layout algorithms) and then composed to produce the final drawing. Some examples of composite layout approaches in the literature include: heuristic layout algorithms for network diagrams [25], the Compose algorithm [27], and, interactive graph layout [22].

2.3 Nested graphs

Informally, a nested graph is an extension of a digraph in which a vertex can contain other vertices. Examples of common nested graph formalisms include: higraphs [20, 21], compound digraphs [37, 10], and the graph formalism employed by EDGE [29].

A hygraph [5] is a triple $\langle V, A, B\rangle$, where V is a set of vertices, $A \subseteq V \times V$ is a set of arcs, and, $B : V \to 2^V$ is a set of blobs. While arcs specify one to one (binary) relations, blobs capture one to many relations.

2.4 Graph exchange format

Graph eXchange Format (GXF) [13] is a specification for a portable external representation for directed graphs and hygraphs. GXF allows graph processor programs to operate on data encoded in a common format which includes certain essential features of graphs. GXF is the standard external format employed by the current version of graphite.

GXF syntax is similar to that of the LISP programming language. A GXF file is a recursive structure of *lists* that contain *atoms* and *lists*. Figure 2 shows a skeleton of this structure (p. 7). Figure 3 shows an excerpt of a GXF file that encodes a small ‘flights’ database which serves as an example throughout this document (p. 7).

3 The graph layout filter

The grDraw filter can be used to draw hygraphs stored in GXF in both 2D and 3D. The filter handles both digraphs (hygraphs without blobs) as well full-fledged hygraphs, and can produce drawings that employ arbitrary combinations of simple layout methods (composite graph layout). Furthermore, a novel feature of the program allows the user to associate *ranks* or *weights* with a set of vertices that can then can be used to produce a variety of non-geometrical fisheye views [18, 32] during the process of computing the layout.

3.1 Layout architecture

In order to provide more flexible and powerful drawing facilities, all layout algorithms in graphite have been designed around a generic container model: a container may recursively contain (sub)containers; containers may optionally be linked by directed arcs. The hygraph formalism is mapped onto this model as shown in Figure 1.

All containers are treated uniformly by layout algorithms. Thus, for example, the same layout algorithm can be used to position a set of vertices in a blob’s vertex region, as well as the three components of a vertex.

3.2 Basic layout algorithms

The current version of grDraw is capable of producing a drawing using arbitrary combinations of the following simple parametric layout algorithms:[1]

1. circle: Elliptical layout. Containers are positioned along the perimeter of an ellipse; containers are ordered to reduce (but not necessarily minimize) the number of arc crossings.

[1] Each algorithm comes in both 2D and 3D versions.

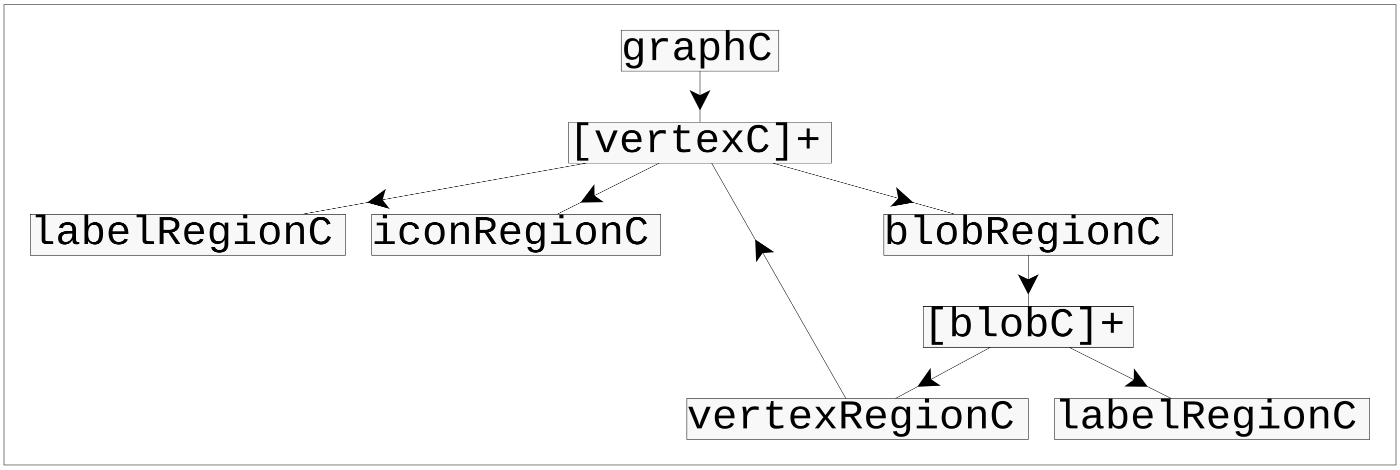

Figure 1: Mapping the hygraph formalism onto the generic container model.

2. grid: Grid layout. Containers are positioned at the intersections of an $m \times n$ grid. Containers are placed, one-connected component at a time, in top to bottom left to right order; containers are ordered to reduce (but not necessarily minimize) the number of arc crossings.

3. hier: Hierarchical layout. The algorithm computes a depth-first spanning forest F of the input graph, and places containers by following tree arcs with respect to F [2] (and ignoring forward, cross, back, and external arcs). Containers are assigned to layers by the *longest path layering* technique [11]. Trees can be grown along any axis. Graphs with multiple roots and cycles can be drawn, but no crossing minimization is performed.

4. nop: No layout. Containers' positions are not altered. This method can be used to provide a form of incremental layout capability. A container's size is changed to fully enclose its subcontainers.

5. overlap: Overlap elimination. Containers are repositioned to eliminate overlaps.

6. pack: Quadtree-based 2D bin packing. Containers are placed into a small enclosing region by an incremental quadtree growing approach. Note that bin packing in NP-complete [19], so this solution necessarily involves the use of heuristics.

7. random: Random layout. Containers are positioned randomly without overlaps.

8. spring: Force-based layout [17]. Containers are positioned according to the following analogy to a physical system: containers are treated as charged bodies that repel one another; arcs are treated as springs attached to pairs of containers that pull each member of the pair towards the other. The simulation is continued until a low-energy state is reached resulting in the final layout.

9. stack: Stack layout. Containers are positioned along the axis which results in the most square (cubical) or compact drawing.

10. 1d: One-dimensional layout. Containers are positioned along one of the specified axes.

Figures 4–7 (pp. 8–11) show a simple flights database graph drawn by grDraw. Note that the PostScript diagrams are obtained by grExport – a filter program described later in this paper.

3.3 Hygraph layout

Figures 9–11 show drawings of a nested version of the flights database (pp. 13–15). In this database, the *city* and *country* arc relations have been converted into blob relations using the grModify filter (Sec. 5).

3.4 Composite hygraph layout

The grDraw filter is capable of performing composite layout at several levels of representation. In addition to the top level graph container, a unique layout algorithm can be associated with each vertex, blob, vertex region, and blob region, by inserting layout directives into the GXF file.

Figure 12 shows a composite layout of the flights database (p. 16).

3.5 Vertex rank assignment

In addition to performing layout, the grDraw filter can also be used to produce non-geometric fisheye views of nested graphs [32]. This is achieved by associating a *rank* (or weight) with one or more vertices, which, for our purposes, serve as focal points. A larger rank associates greater importance with a vertex, which results in the magnification of the vertex and all of its subcontainers as well as the magnification of a subset of "nearby" vertices and their subcontainers.

Figure 13 shows a stack layout of the nested flights database with several fisheye focal points (p. 17).

4 The graph export filter

The grExport filter translates a GXF file into two external file formats:

1. ps: PostScript. This operation converts a GXF file into encapsulated PostScript [1] for subsequent inclusion in documents. Not surprisingly, the Figures shown in this document were produced by this program.

2. flat: Flat Ascii file. This operation creates a simple generic ascii file that could be imported by foreign programs.

5 The graph modification filter

The grModify filter modifies aspects of a graph's topology and visualization:

1. arcs to blobs: Convert an arc (binary) relation into a blob (one to many) relation.

2. blobs to arcs: Convert a blob relation into an arc relation.

3. fisheye: geometric fisheye view of a graph [18, 28, 36]. This operation distorts an existing layout by magnifying vertices near the focal point while simultaneously reducing distant vertices. Figure 8 (p. 12) shows a geometric fisheye view of the flights database.

4. hide: automatic detail hiding [16, 30]. This operation has two modes: (1) vertices and blobs whose size is less than a specified threshold are hidden; and, (2) blobs at a specified level of nesting and below are hidden. Display programs (including grExport) can use this information to suppress details in hidden areas. Figure 14 (p. 18) shows the nested flights graph with hidden elements.

5. link inherit: reduce the number of arcs by link inheritance [16]. Figure 15 (p. 19) shows the nested flights graph to which link inheritance operation has been applied.

6. reset: reset all layout information.

6 Internet archive site

This document and [13] are available in PostScript format from the internet host db.toronto.edu [128.100.2.100]. The compressed PostScript files can be obtained by anonymous ftp in /pub/gplus/gxf/doc/*.

7 Future enhancements

The following enhancements are anticipated in the near future:

1. Design and implementation of a persistent graph visualization server.
2. Design of an efficient fixed-record ascii command and data exchange protocol for above.
3. Design and implementation of a simple scripted demonstration program that illustrates the server's functionality.

Long range goals include:

1. Addition of crossing minimization in the hierarchical layout algorithm.
2. Addition of the arc layout capability.
3. Compatibility with future graph exchange standards.

References

[1] Adobe Systems Incorporated. *PostScript language: reference manual*. Addison-Wesley, Reading, MA, 1985.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[3] D.G. Belanger, R.J. Brachman, Y.F. Chen, P.T. Devanbu, and P.G. Selfridge. Toward a Software Information System. *AT&T Technical Journal*, pages 22–41, March/April 1990.

[4] M. Caplinger. Graphical Database Browsing. *ACM SIGOIS Bulletin*, 7(2–3):113–121, Summer-Fall 1986.

[5] M.P. Consens. Theory of Visual Manipulations of Database Visualizations. U. of Toronto, Ph.D. Research Proposal, Feb. 1992.

[6] M.P. Consens, I.F. Cruz, and A.O. Mendelzon. Visualizing Queries and Querying Visualizations. *ACM SIGMOD Record*, 21(1):39–46, Mar. 1992.

[7] M.P. Consens, A.O. Mendelzon, and A.G. Ryman. Visualizing and Querying Software Structures. In *CAS '91*, pages 17–35, Toronto, Canada, October 28–30 1991.

[8] I.F. Cruz. G+: A Graphical system for recursive queries. Master's thesis, Dept. of Comp. Sci., U. of Toronto, 1987.

[9] P.T. Devanbu, R.J. Brachman, P.G. Selfridge, and B.W. Ballard. LaSSIE: A Knowledge-based Software Information System. *Communications of the ACM*, 34(5):34–49, May. 1991.

[10] P. Eades, T. Lin, and X. Lin. Two Tree Drawing Conventions. *International Journal of Computational Geometry and Applications*, 1992.

[11] P. Eades and K. Sugiyama. How to Draw a Directed Graph. *Journal of Information Processing*, 13(4), 1990.

[12] P. Eades and R. Tamassia. Algorithms for Drawing Graphs: an Annotated Bibliography. Technical Report CS-89-09, Dept. of Comp. Sci., Brown U., 1989.

[13] Frank Ch. Eigler. GXF: A Graph Exchange Format. CORDS White Paper, Jan. 1993.

[14] Margaret A. Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley, Reading, MA, 1990.

[15] K.M. Fairchild, S.E. Poltrock, and G.W. Furnas. SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases. In R. Guindon, editor, *Cognitive Science and its Applications for Human-Computer Interaction*, pages 201–233. Lawrence Erlbaum Associates, 1988.

[16] S. Feiner. Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure. In *ACM OIS '88*, pages 205–212. ACM, March 23–25 1988.

[17] T. Fruchterman and E. Reingold. Graph Drawing By Force-Directed Placement. Technical Report UIUCDCS-R-90-1609, Dept. of Comp. Sci., U. of Illinois at Urbana-Champaign, Jun. 1990.

[18] G.W. Furnas. Generalized Fisheye Views. In *ACM CHI '86*, pages 16–23, Boston, MA, Apr. 1986. ACM.

[19] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, NY, 1979.

[20] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514–530, May. 1988.

[21] D. Harel and C.A. Kahana. On Statecharts with Overlapping. *ACM Transactions on Software Engineering and Methodologies*, 1(4):399–421, Oct. 1992.

[22] T.R. Henry and S.E. Hudson. Interactive Graph Layout. In *UIST '91*, pages 55–64. ACM, 1991.

[23] D. Jablonowski and Guarna A.Jr. GMB: A Tool for Manipulating and Animating Graph Data Structures. *Software–Practice and Experience*, 19(3):283–301, Mar. 1989.

[24] T. Kamada and S. Kawai. A General Framework for Visualizing Abstract Objects and Relations. *ACM Transactions on Graphics*, 10(1):1–39, Jan. 1991.

[25] G. Kar, B. Madden, and R.S. Gilbert. Heuristic Layout Algorithms for Network Management Presentation Services. *IEEE Network*, 29(11):29–36, Nov. 1988.

[26] J.A. Mariani and R. Lougher. TripleSpace: an experiment in a 3D graphical interface to a binary relational database. *Interacting with Computers: The Interdisciplinary Journal of Human-Computer Interaction*, 4(2):147–162, Aug. 1992.

[27] E.B. Messinger, L.A. Rowe, and R.R. Henry. A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(1):1–12, 1991.

[28] K. Misue and K. Sugiyama. Multi-viewpoint Perspective Display Methods: Formulation and Application to Compound Graphs. In *4th Intl. Conf. on Human-Computer Interaction*, volume 1, pages 834–838, Stuttgart, Germany, Sep. 1991. Elsevier.

[29] F. Newbery Paulisch and W.F. Tichy. EDGE: An Extensible Graph Editor. *Software–Practice and Experience*, 20(S1):63–88, Jun. 1990.

[30] J. Nielsen. *Hypertext and Hypermedia*. Academic Press, San Diego, CA, 92101, 1990.

[31] E.G. Noik. Challenges in Graph-Based Relational Data Visualization. In *CAS '92.*, pages 259–277, Toronto, Canada, Nov. 1992.

[32] E.G. Noik. Layout-independent Fisheye Views of Nested Graphs. In *VL '93: IEEE/CS Symp. on Visual Languages*, Bergen, Norway, Aug. 1993.

[33] J. Ramanathan and R.L. Hartung. A Generic Iconic Tool for Viewing Databases. *IEEE Software*, 6(9):50–57, Sep. 1989.

[34] G.G. Robertson, J.D. Mackinlay, and S.K. Card. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *ACM CHI '91*, pages 189–194. ACM, Apr. 1991.

[35] A.G. Ryman. Foundations of 4Thought. In *CAS '92*, pages 133–155, Toronto, Canada, Nov. 1992.

[36] M. Sarkar and M.H. Brown. Graphical Fisheye Views of Graphs. In *ACM CHI '92*, pages 83–91, Monterey, CA, May. 1992. ACM.

[37] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.

```
graph
    [graph-id]
    [graph-bounds]
    nodes*
        node*
            node-id
            node-label-ref | node-label
            [node-position]
            [node-bounds]
            [node-blobs-region]
            [node-icon-region]
            [node-label-region]
    edges*
        edge*
            from-node-id
            to-node-id
            edge-label
            [edge-points]
            [edge-distinguished]
    blobs*
        blob*
            blob-from-node-id
            blob-to-node-id
            blob-label
            [blob-bounds]
            [blob-nodes-region]
            [blob-label-region]
            [blob-distinguished]
```

Figure 2: A skeleton of GXF file structure.

```
( graph
    ( id "noik@mambo.db, Thu Feb 11 17:53:12 1993" )
    ( bounds
        ( rectangle
            ( xy 0 0 )
            ( xy 604.887 472.857 ) ) )
    ( nodes
        ( node
            ( id "6" )
            ( label "'Lisbon'" )
            ( point
                    ( xy 10 452.857 ) )
            ( bounds
                    ( rectangle
                        ( xy 10 452.857 )
                        ( xy 90 462.857 ) ) )
            ( label_region
                    ( rectangle
                        ( xy 10 452.857 )
                        ( xy 90 462.857 ) ) ) )
        …
    )
    ( edges
        ( edge
            ( from "6" )
            ( to "9" )
            ( label "'CP'" ) )
        …
    )
    ( blobs
        ( blob
            ( label "cities" )
            ( from "36" )
            ( to "20" )
            ( to "13" )
            ( to "29" )
            ( to "7" )
            ( point
                    ( xy 870.857 20.5714 ) )
            ( bounds
                    ( rectangle
                        ( xy 388.773 25 )
                        ( xy 687.446 105 ) ) ) )
        …
    )
)
```

Figure 3: An excerpt from the GXF file that encodes the *flights* database.

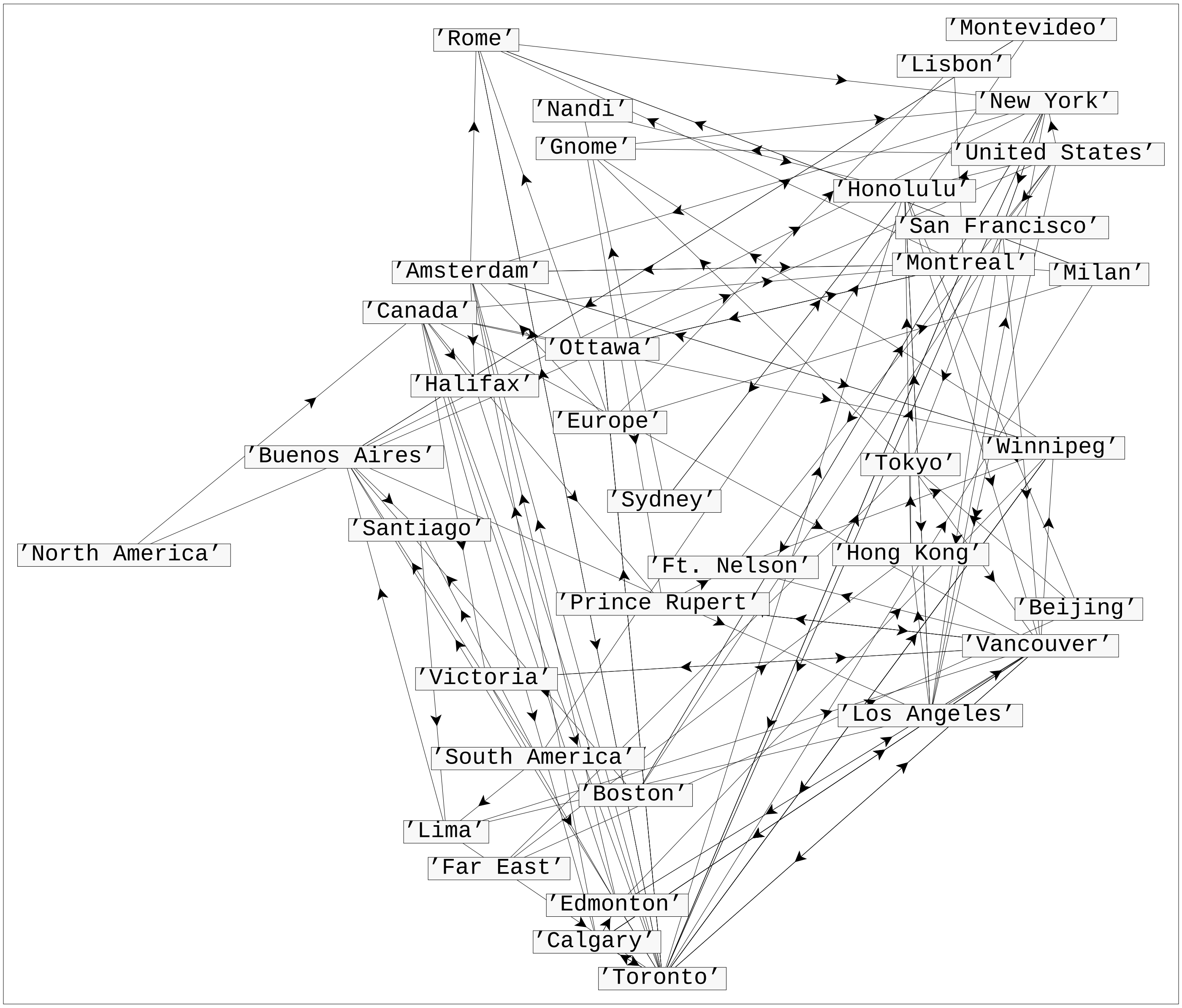

Figure 4: A *random* layout of the flights database.

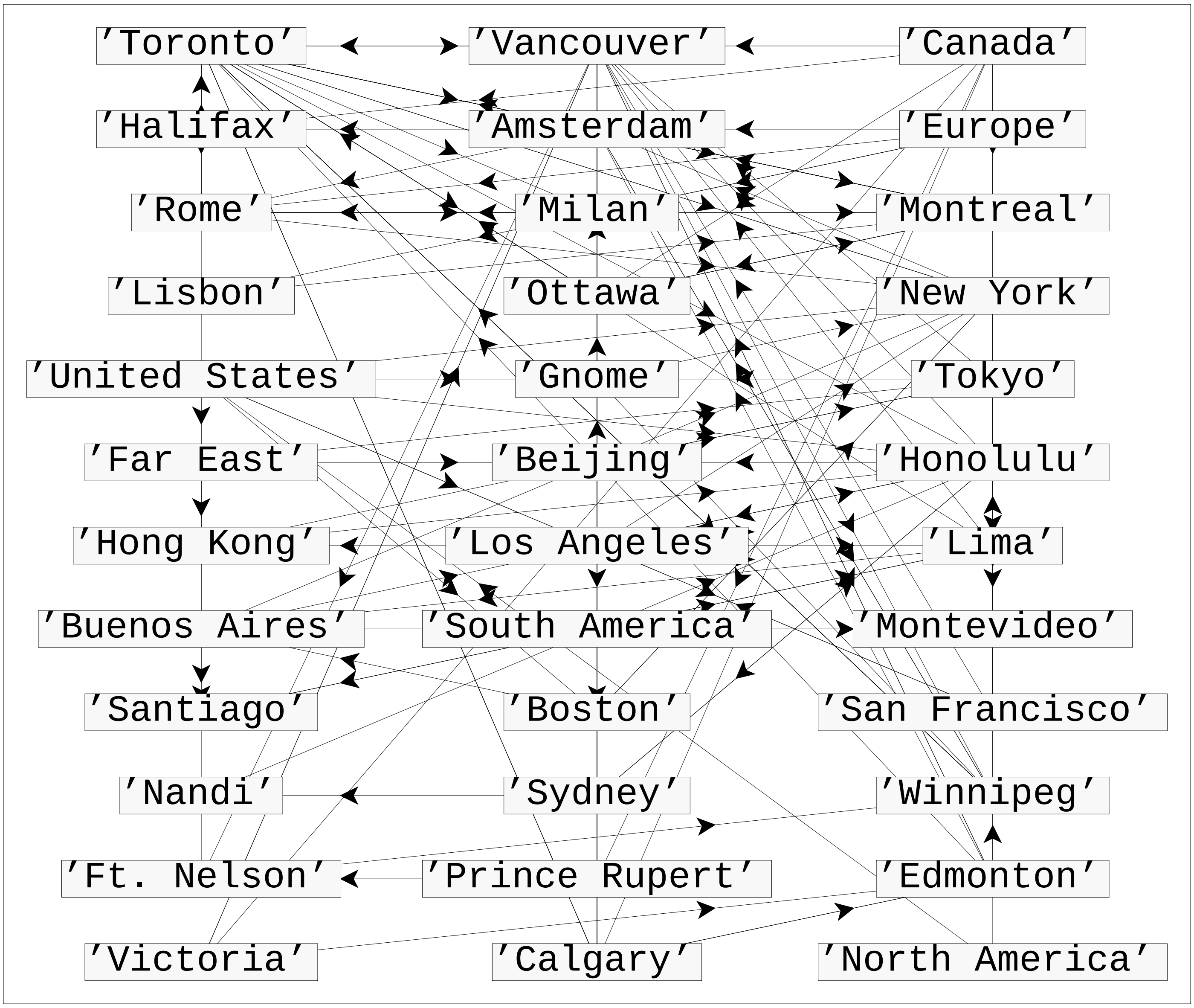

Figure 5: A *grid* layout of the flights database.

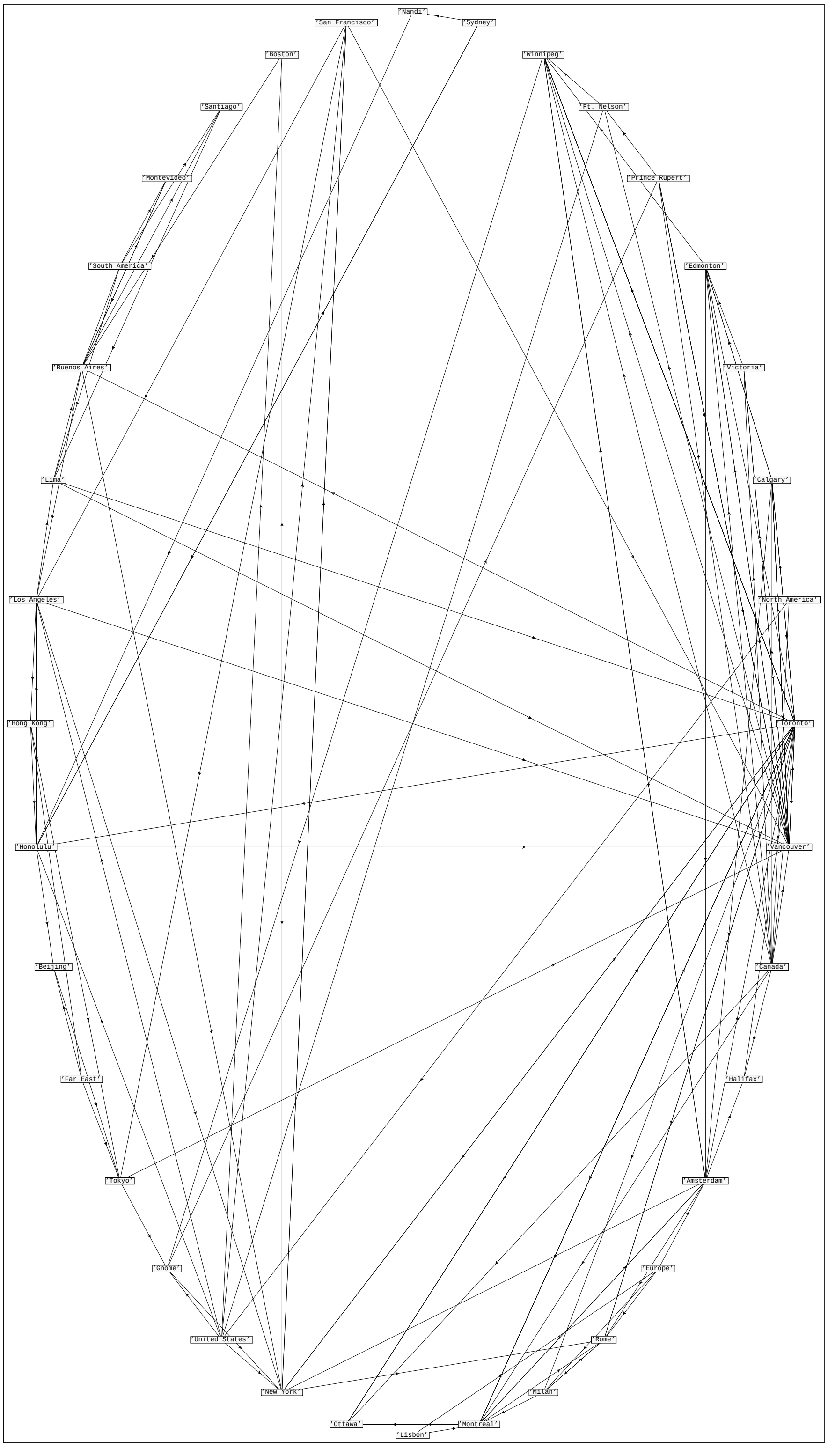

Figure 6: A *circle* layout of the flights database.

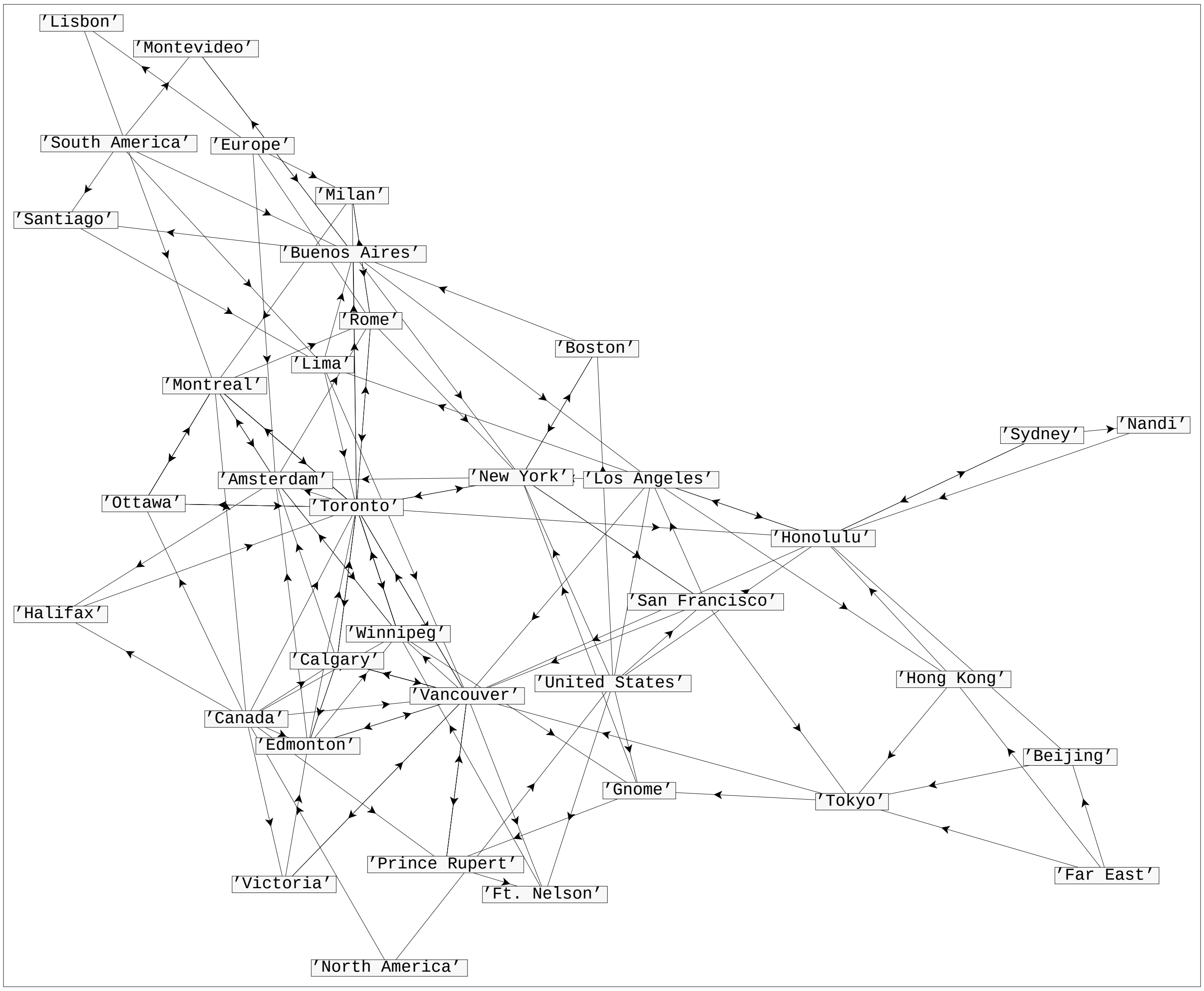

Figure 7: A *spring* layout of the flights database.

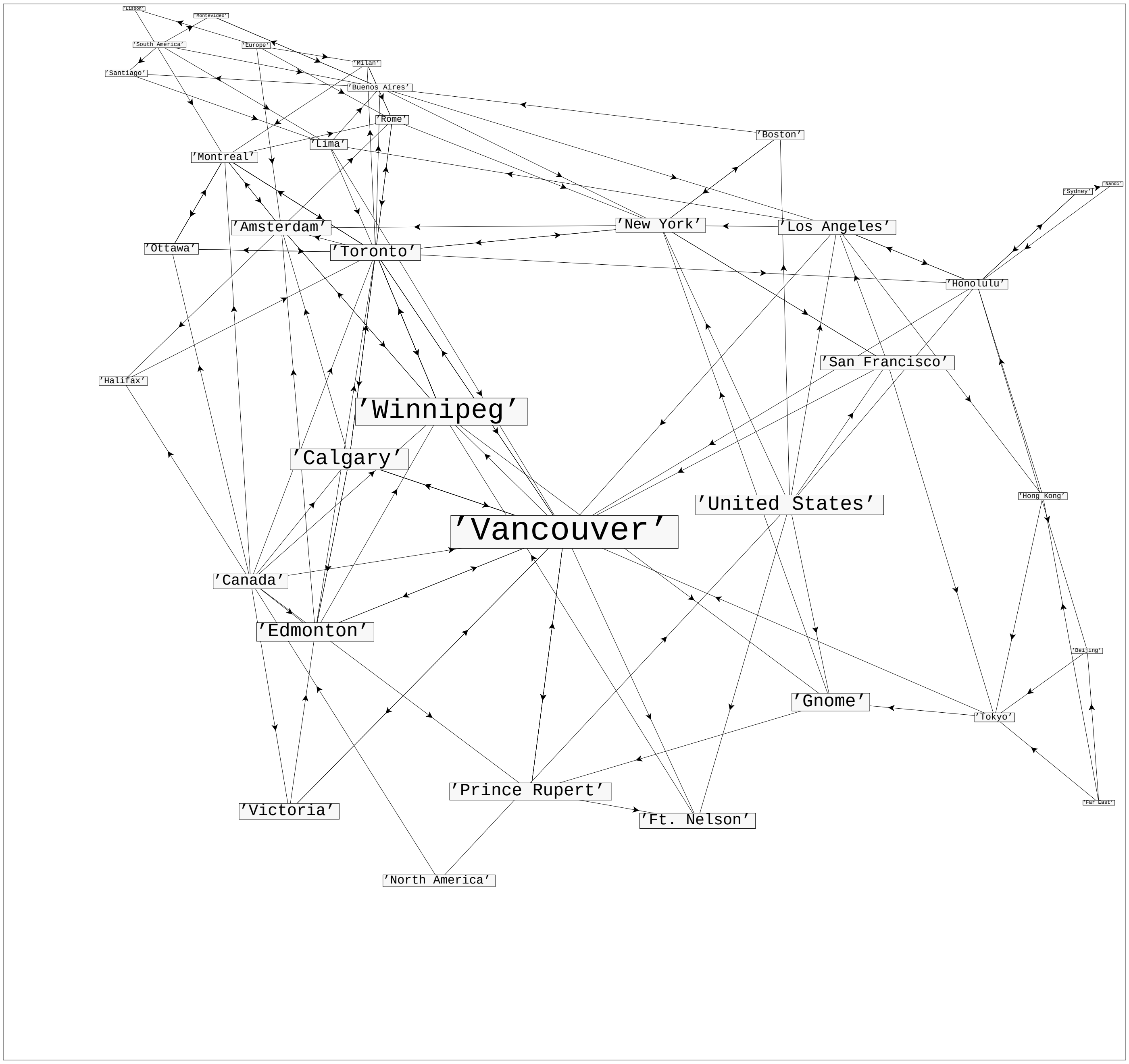

Figure 8: A geometric fisheye view of the *spring* layout of the flights database.

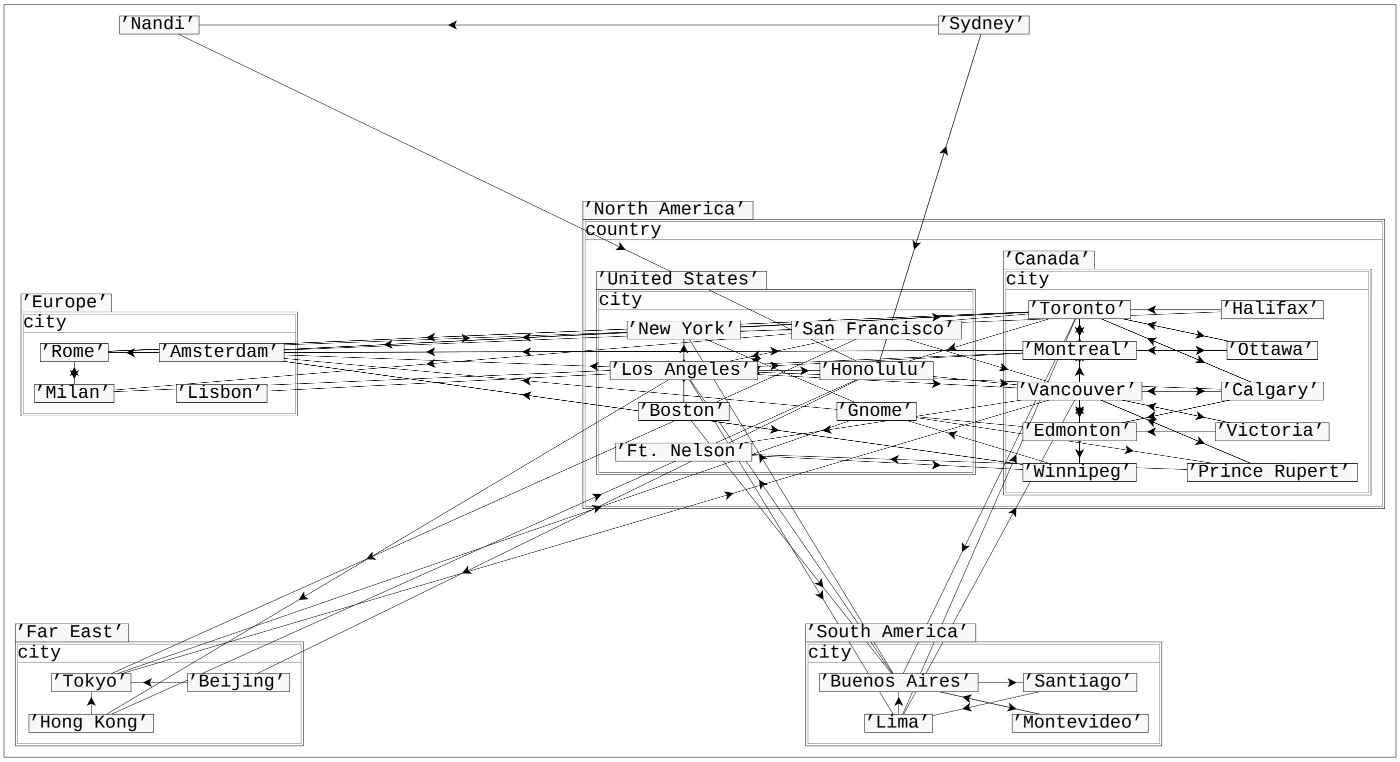

Figure 9: A *grid* layout of the nested representation of the flights database.

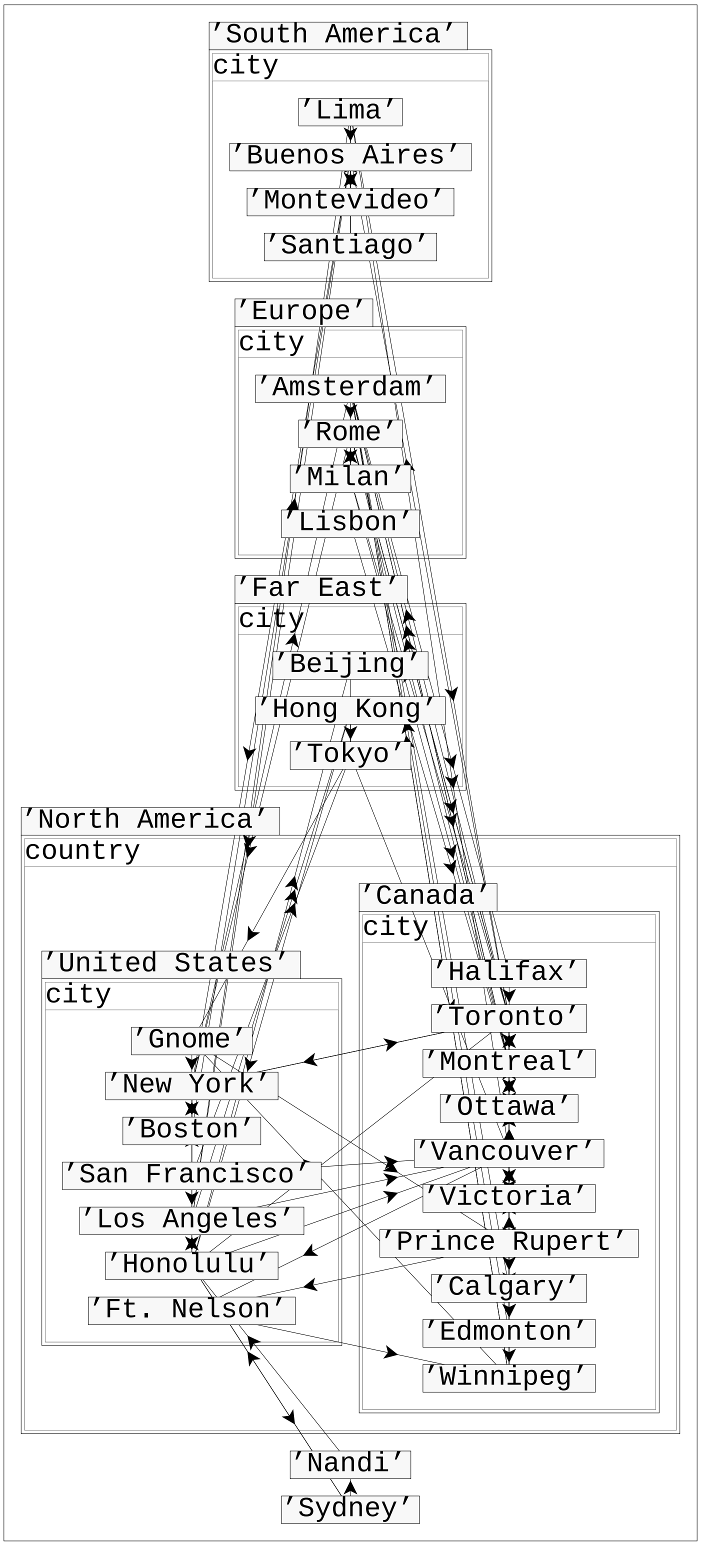

Figure 10: A *stack* layout of the nested representation of the flights database. This algorithm selects one of *horizontal* or *vertical* layout methods, depending on which one produces the most square-like result.

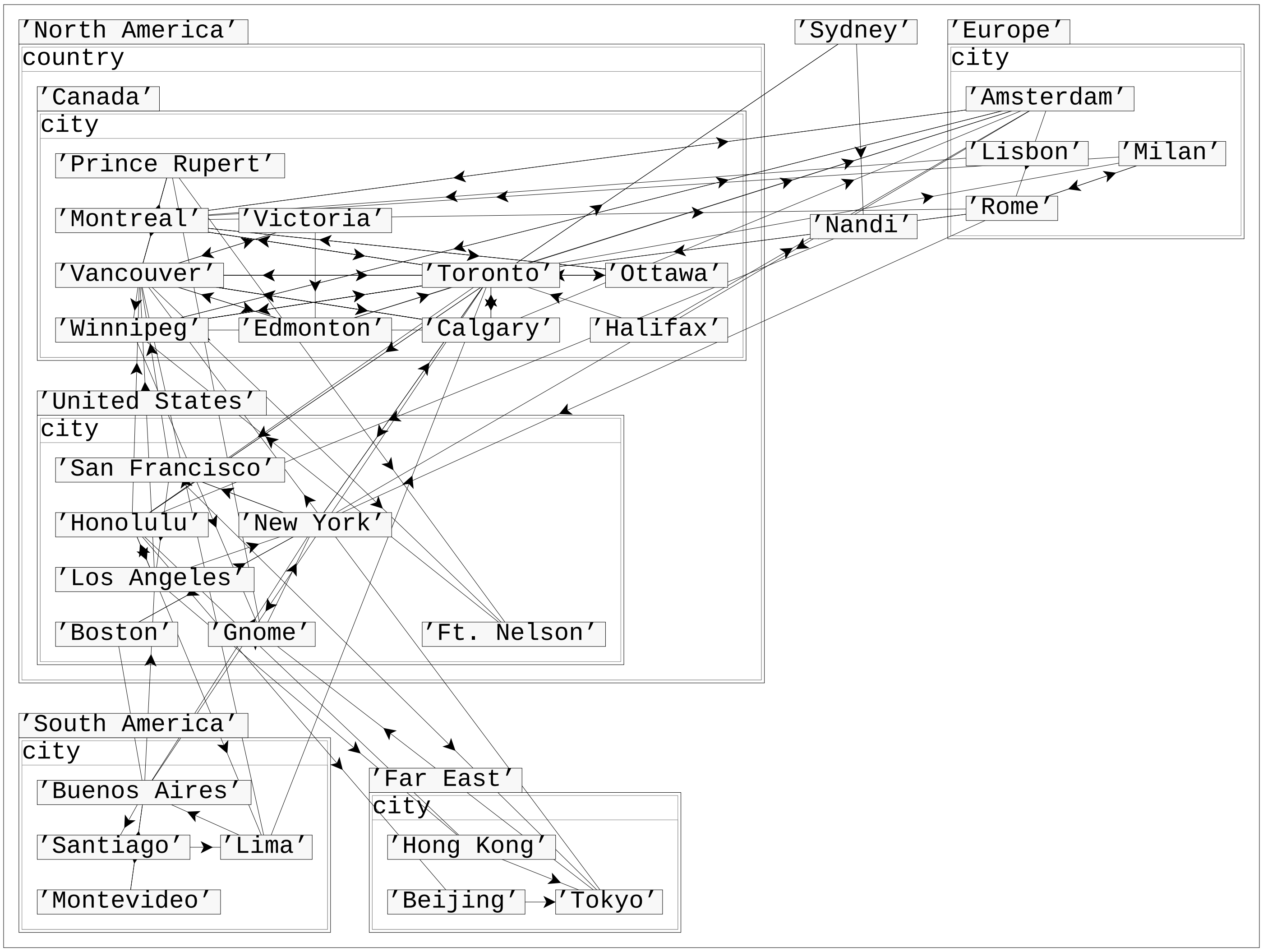

Figure 11: A *pack* layout of the nested representation of the flights database.

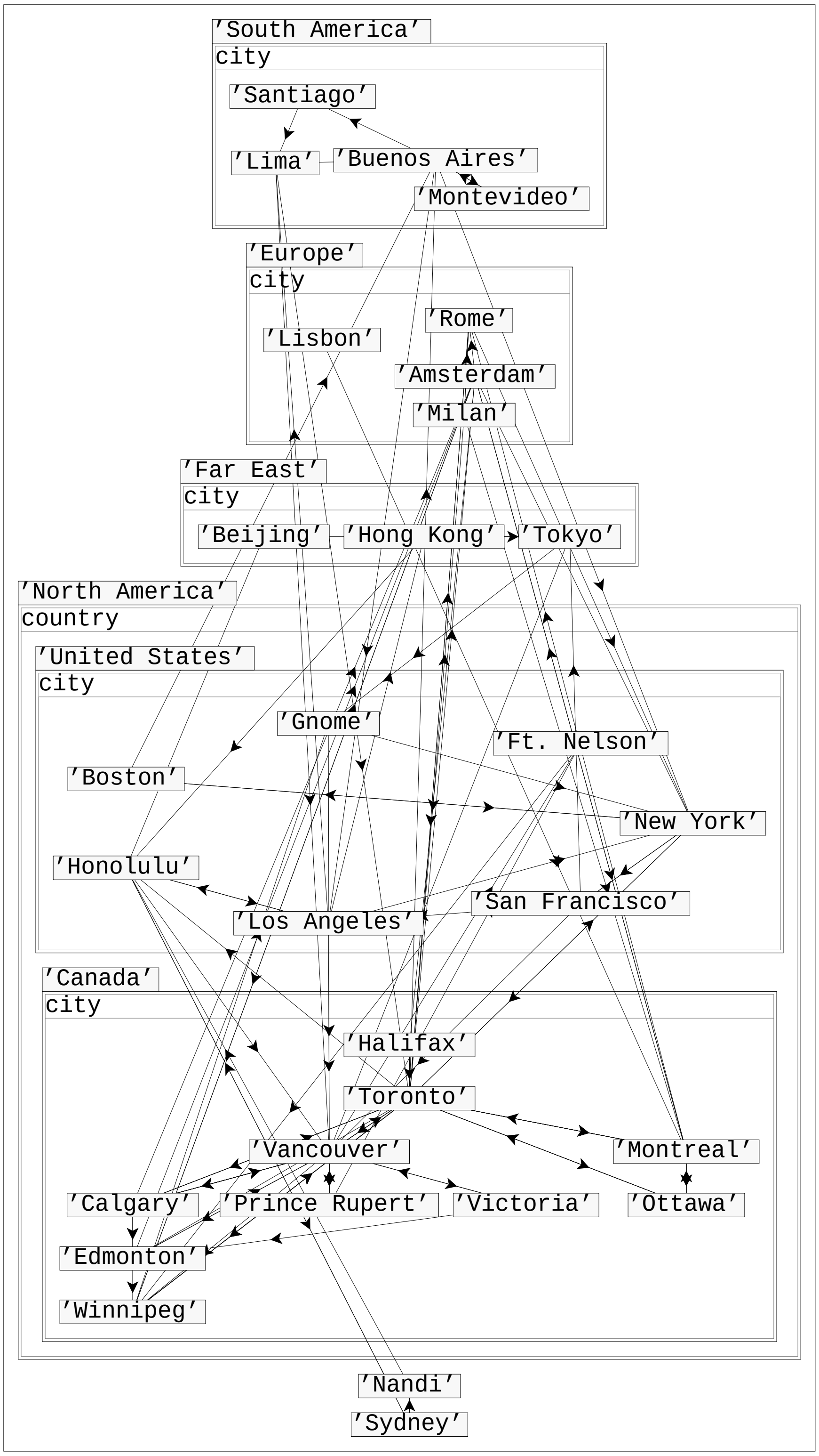

Figure 12: A composite layout of the nested flights database. The following algorithms have been composed: *South America:* force-based; *Europe:* random; *Far East:* horizontal; *United States:* ellipse; *Canada:* hierarchical; and, *North America* and 'world': vertical.

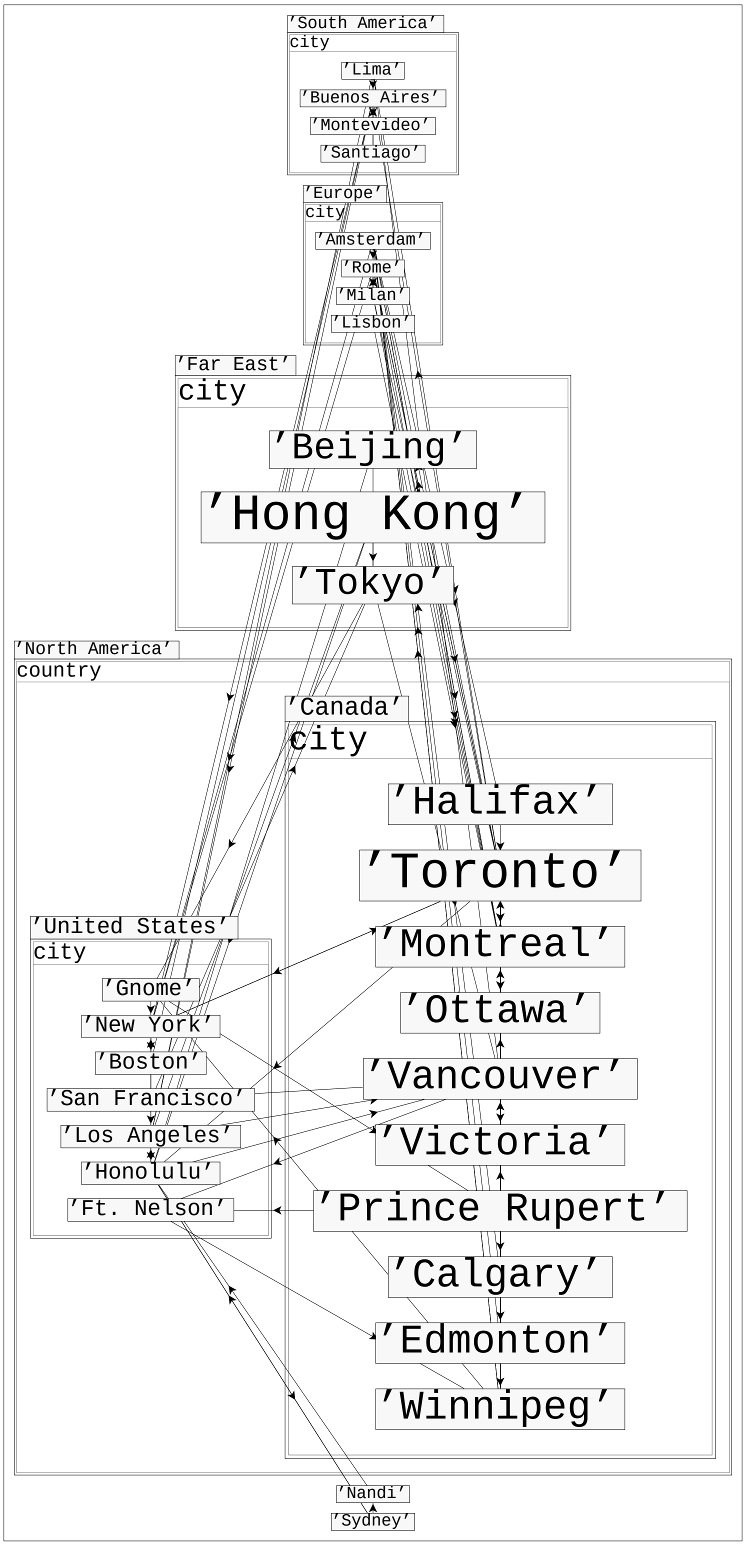

Figure 13: A non-geometric fisheye view of the *stack* layout of the nested representation of the flights database with focal points *Toronto* and *Hong Kong*.

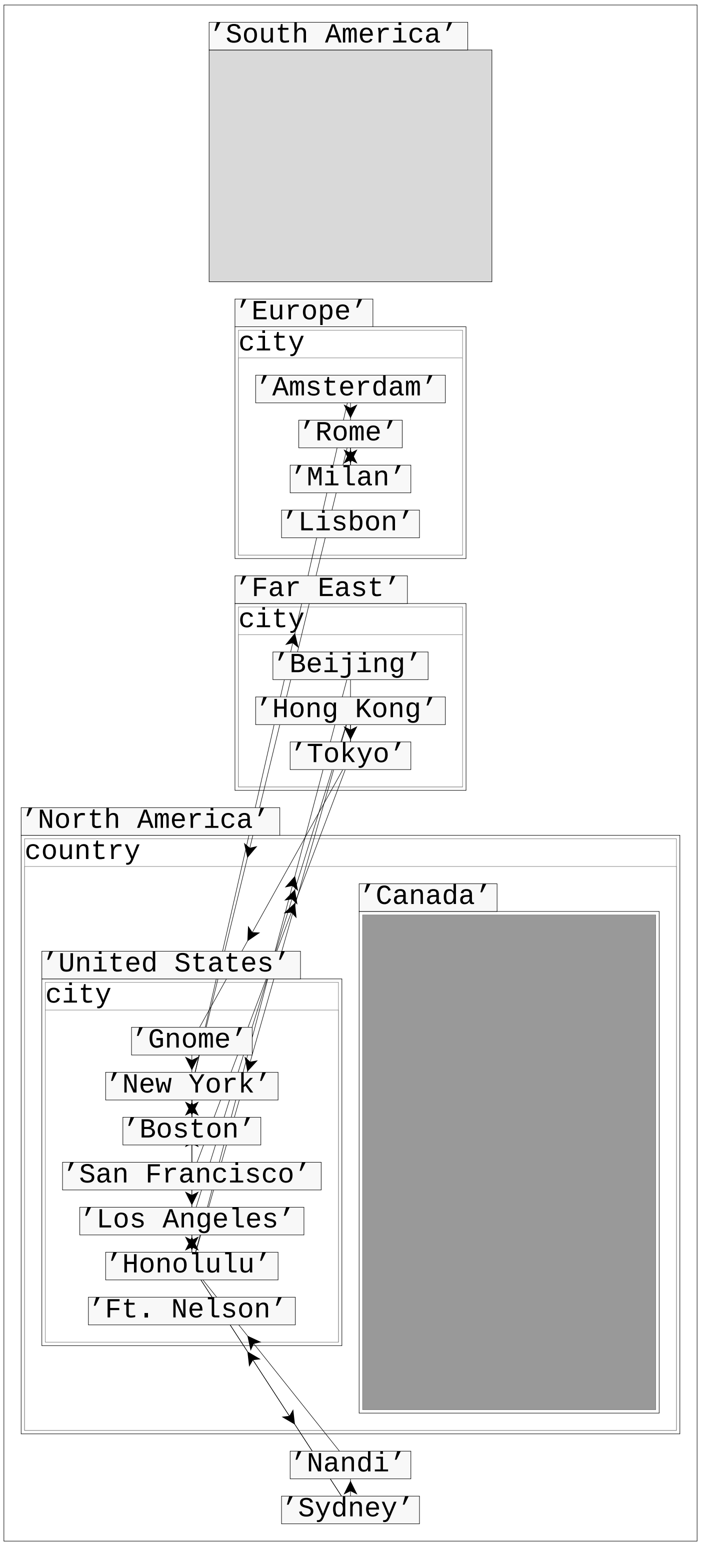

Figure 14: A *stack* layout of the nested representation of the flights database in which the contents of the *South America* vertex and the contents of the *Canada* cities relationship have been hidden. Both vertices that are contained in these regions and arcs that are adjacent to one or more of these vertices have been suppressed.

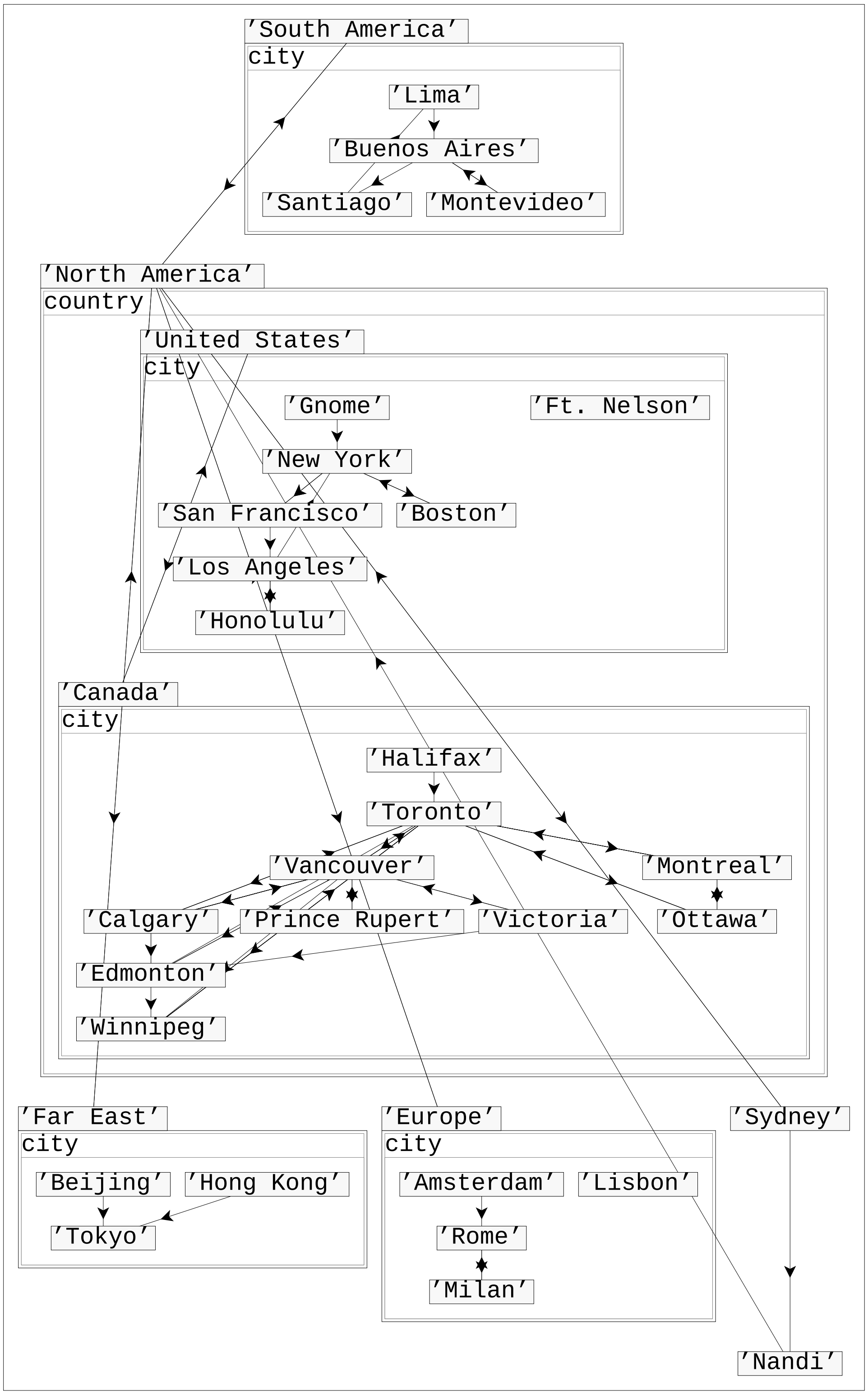

Figure 15: A *hierarchical* layout of the nested representation of the flights database in which the number of edges has been reduced by applying *link inheritance*.