Journal of Graph Algorithms and Applications
http://jgaa.info/
vol. 6, no. 3, pp. 281–311 (2002)

Visualization of the High Level Structure of the Internet with HERMES

Andrea Carmignani[1] *Giuseppe Di Battista*[1] *Walter Didimo*[2]
Francesco Matera[1] *Maurizio Pizzonia*[1]

[1]Dipartimento di Informatica e Automazione,
Università di Roma Tre, via della Vasca Navale 79,
00146 Roma, Italy.
{carmigna,gdb,matera,pizzonia}@dia.uniroma3.it
http://www.dia.uniroma3.it/

[2]Dipartimento di Ingegneria Elettronica e dell'Informazione,
Università di Perugia, Via G. Duranti 93, 06125 Perugia, Italy.
didimo@diei.unipg.it
http://www.diei.unipg.it/

Abstract

HERMES is a system for exploring and visualizing the Internet structure at the level of the Autonomous Systems and their interconnections. It relies on a three-tier architecture, on a large repository of routing information coming from heterogeneous sources, and on sophisticated graph drawing engine. Such an engine exploits static and dynamic graph drawing techniques, specifically devised for the visualization of large graphs with high density.

Communicated by Michael Kaufmann: submitted February 2001;
revised December 2001 and May 2002.

Research supported in part by the European Commission – Fet Open project COSIN IST-2001-33555.

1 Introduction and Overview

Computer networks are an endless source of problems and motivations for the Graph Drawing and for the Information Visualization communities. Several systems aim at giving a graphical representation of computer networks at different abstraction levels and for different types of users. To give only some examples (an interesting survey can be found in [17]):

1. Application level: Visualization of Web sites structures, Web maps [18, 16], and Web caches [10].
2. Network level: Visualization of multicast backbones [23], Internet traffic [24], routes, and interconnection of routers.
3. Data Link level: Interconnection of switches and repeaters in a local area network [2].

We deal with the problem of exploring and visualizing the interconnections between *Autonomous Systems*. An Autonomous System (in the following AS) is a group of networks under the same administrative authority. Roughly speaking, an AS can be seen as a portion of the Internet, and Internet can be seen as the totality of its ASes. To maintain the reachability of any portion of the Internet each AS exchanges routing information with a group of other ASes mainly selected on the basis of economic and social considerations. Several tools have been developed for analyzing and visualizing the Internet topology at the ASes level [9, 12, 24, 30]. However, in our opinion, such tools are still not completely satisfactory both in the interaction with the user and in the effectiveness of the drawings. Some of them have the goal of showing large portions of Internet, but the maps they produce can be difficult to read (see e.g. [9]). Other tools point the attention on a specific AS, only showing that AS and its direct connections (see e.g. [24]).

In this work we describe a new system, called HERMES, that provides a publicly accessible service over the Web[1]. It has a three-tier architecture which allows the user to visually explore the Internet topology by means of automatically computed maps. Such maps are computed by a graph drawing module based on the GDToolkit library [21] whose main features are the following:

- Its basic drawing convention is the podevsnef [20] model for orthogonal drawings having vertices of degree greater than four. However, since the handled graphs have often many vertices (ASes) of degree one connected with the same vertex, the podevsnef model is enriched with new features for representing such vertices.
- It is equipped with two different graph drawing algorithms. In fact, at each exploration step the map is enriched and hence it has to be redrawn. Depending on the situation, the system (or the user) might want to use a

[1] http://www.dia.uniroma3.it/~hermes

static or a dynamic algorithm. Of course, the dynamic and the static algorithms have advantages and drawbacks. The dynamic algorithm allows the user to preserve its *mental map* [19, 27] but can lead, after a certain number of exploration steps, to drawings that are less readable than those constructed with a static algorithm.

- The static algorithm is based on the topology-shape-metrics approach [14] and exploits recent compaction techniques that can draw vertices with any prescribed size [13]. The topology-shape-metrics approach has been shown to be very effective and reasonably efficient in practical applications [15].

- The dynamic algorithm is a new dynamic graph drawing algorithm that has been devised according to three main constraints. (1) It had to be completely integrated within the topology-shape-metrics approach in such a way to be possible to alternate its usage with the usage of the static algorithm. (2) It had to be consistent with the variation of the podevsnef model used by HERMES. (3) It had to allow vertices of arbitrary size. Several algorithms have been recently proposed in the literature on dynamic graph drawing algorithms. A linear time algorithm for orthogonal drawings is presented in [5]. In this algorithm the position of the vertices cannot be changed after the initial placement. Four different scenarios for interactive orthogonal graph drawings are studied in [29]. Each scenario defines the changes allowed in the common part of two consecutive drawings. An interactive version of GIOTTO [33] is described in [8]; it allows the user to incrementally add vertices and edges to an orthogonal drawing in such a way that the shape of the common part of two consecutive drawings is preserved and the number of bends is minimized under this constraint. A dynamic algorithm for orthogonal drawings that allows us to specify the relative importance of the number of bends vs. the number of changes between two consecutive drawings is given in [6]. Other algorithms for constructing drawings of graphs incrementally, while preserving the mental map of the user, are for example [26, 11, 28]. Also, a study on different metrics that can be used to evaluate the changes between drawings in an interactive scenario is presented in [7]. However, as far as we know, none of the cited dynamic algorithms enforces all the constraints (1), (2), and (3).

The paper is organized as follows. In Section 2 we give basic definitions about graph drawing and AS level networking which are needed to understand the rest of the work. The reader that is not interested in networking may skip Section 2.1. In Section 3 we explain how the user interacts with HERMES and provide a high level description of the functionalities of the system. In Section 4 we give some details about the three-tier architecture of HERMES. Section 5 shows the results of a study on the ASes interconnection graph. Such a study has been performed in order to design the drawing convention and the drawing algorithms of HERMES. In particular, we analyze the density of the graph and its distribution, and we give measures on the average degree of the vertices.

In Section 6 we describe in detail the drawing convention and the algorithms used by system. The techniques we adopt are not limited to the application domain described in this paper, but they are suitable for exploring and visualizing general large graphs with high density. In Section 7 some statistics about the effectiveness of Hermes as a Web service are provided. Conclusions and open problems are given in Section 8.

2 Background

2.1 Networking

Each AS groups a set of networks. An AS is identified over the Internet by an integer number while each network is identified by its IP address. A *route* is a (directed) path on the Internet that can be used to reach a specific set of (usually contiguous) IP addresses, representing a set of networks. A route is completely described by its destination IP addresses, its cost, and by the ordered set of ASes that it traverses (usually called *AS-path*). Routes can be seen as advertisements, from an AS to its adjacent ASes, meaning "through me you can reach a certain set of networks, with a certain cost and traversing a certain set of other ASes".

In order to exchange information about the routes, the ASes adopt a routing protocol called *BGP* (Border Gateway Protocol) [31]. Such a protocol is based on a distributed architecture where *border routers* that belong to distinct ASes exchange information about the routes they know. Two border routers that directly exchange information are said to perform a *peering session*, and the ASes they belong to are said to be *adjacent*. We define the *ASes interconnection graph* as the graph having a vertex for each AS and one edge between each pair of adjacent ASes. Note that, according to our definition, the ASes interconnection graph is not a multigraph.

Each route is incrementally built. A route is *originated* by an AS and initially it contains only such AS, then it is *propagated* to adjacent ASes which append their identifiers to the AS-path of the route and propagate it again. Hence, in the AS-path of a route two consecutive ASes are always adjacent.

2.2 Graph Drawing

We assume familiarity with elementary graph theory and graph connectivity [22].

A *plane drawing* Γ of a graph G maps each vertex of G to a point of the plane, and each edge of G to a Jordan curve between the two points associated with the end-vertices of the edge. A drawing Γ of G is *planar* if any two edges never intersect except at common end-vertices. A graph is *planar* if it admits a planar drawing. A planar drawing Γ of G induces for each vertex v of G a circular clockwise ordering of the edges incident on v. Also, Γ subdivides the plane into topologically connected regions, called *faces*. Exactly one of these

faces is unbounded; it is called *external face*. The other faces are said to be *internal*. Two planar drawings of G are said to be *equivalent* if (i) for each vertex v of G they induce the same ordering of the edges around v, and (ii) they have the same external face. Note that two equivalent drawings of G have the same set of faces. An *embedding* ϕ of G is a class of equivalent planar drawings of G. In other words, we can regard an embedding of G as the choice of a clockwise ordering of the edges around every vertex plus the choice of the external face. An *embedded graph* G_ϕ is a planar graph G with a given embedding ϕ.

An *orthogonal drawing* of G is a drawing of G such that all edges are represented as polygonal lines of horizontal and vertical segments. Clearly, an orthogonal drawing of G exists if and only if G is 4-planar, that is, each vertex of G has at most four incident edges. An *orthogonal representation* (or *shape*) of G is an equivalence class of planar orthogonal drawings such that the following hold:

1. For each edge (u, v) of G, all the drawings of the class have the same sequence of left and right turns (*bends*) along (u, v), while moving from u to v.

2. For each vertex v of G, and for each pair $\{e_1, e_2\}$ of clockwise consecutive edges incident on v, all the drawings of the class determine the same angle between e_1 and e_2.

Roughly speaking, an orthogonal representation defines a class of orthogonal drawings that may differ only for the length of the segments of the edges.

In order to orthogonally draw graphs of arbitrary vertex degree, different drawing conventions have been introduced in the literature. Here we recall the *podevsnef* (planar orthogonal drawing with equal vertex size and not empty faces) drawing convention, defined by Fößmeier and Kaufmann [20]. In a *podevsnef* drawing (see Figure 1 (a)):

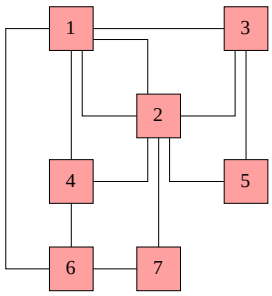
1. Vertices are points of an integer coordinate grid (but it is easier to think of them in terms of squares of half unit sides centered at grid points).

2. Two segments that are incident on the same vertex may overlap. Observe that the angle between such segments has zero degree.

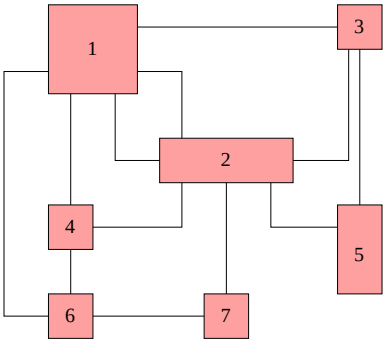
3. All the polygons representing the faces have area strictly greater than zero.

4. If two segments overlap they are presented to the user as two very near segments.

An algorithm that computes a *podevsnef* drawing of an embedded planar graph with the minimum number of bends is presented in [20] . Further, the authors conjecture that the drawing problem becomes NP-hard when Condition 3 is omitted. The *podevsnef* drawings generalize the concept of orthogonal



(a)



vertex	*width*	*height*
1	1	1
2	2	0
3	0	0
4	0	0
5	0	1
6	0	0
7	0	0

(b)

Figure 1: (a) A *podevsnef* drawing; (b) A *podavsnef* drawing with the same shape as the drawing in (a); the sizes of the vertices are specified in the table.

representation, allowing angles between two edges incident to the same vertex to have a zero degree value. The consequence of the assumption that the polygons representing the faces have area strictly greater than zero is that the angles have specific constraints. Namely, because of Conditions 2 and 3, each zero degree angle is in correspondence with exactly one bend [20]. An orthogonal representation corresponding to the above definition is a *podevsnef orthogonal representation*.

A drawing convention that allows the user to draw graphs in which each single vertex has a prescribed size (width and height) has been introduced in [13]. Such a drawing convention is referred to as *podavsnef* (planar orthogonal drawing with assigned vertex size and non-empty faces). A *podavsnef* drawing has the following properties (see also Figure 1 (b)):

1. Each vertex is a box with specific width and height (assigned to each single vertex by the user).

2. Two segments that are incident on the same vertex may overlap. Again, the angle between such segments has zero degree.

3. Consider any side of length $l \geq 0$ of a vertex v and consider the set I of arcs that are incident on such side.

   (a) If $l + 1 > |I|$ then the edges of I cannot overlap.

   (b) If $l + 1 \leq |I|$ then the edges of I are partitioned into $l + 1$ non-empty subsets such that all the edges of the same subset overlap.

4. The orthogonal representation constructed from a *podavsnef* drawing by contracting each vertex into a single point is a *podevsnef* orthogonal representation.

A polynomial time algorithm for computing *podavsnef* drawings of an embedded planar graph with the minimum number of bends over a wide class of *podavsnef* drawings is also described in [13].

3 Using HERMES

The user interacts with HERMES through a map (subgraph) of the ASes interconnection graph. A map is initially constructed using two possible starting primitives.

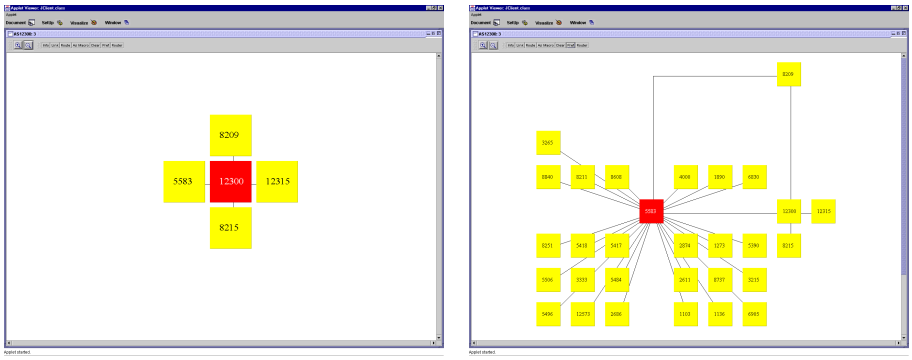
AS selection An AS is chosen. The obtained map consists of such an AS plus all the ASes that are connected to it. See Fig. 2(a).

Routes selection A set of routes is selected. The user has three possibilities: (1) selection of all the routes traversing a specific AS (see Fig. 3(a)); (2) selection of the routes starting from a specific AS (see Figs. 3(b)); (3) selection of all the routes traversing a specific pair of ASes. The obtained map consists of all the ASes and connections traversed by the selected routes.

The user can explore and enrich the map by using the following primitive:

AS exploration an AS u among those displayed in the current map is selected. The current map is augmented with all the ASes that are connected to u. Further, for each AS v connected to u an edge (u, v) is added. Observe that, according to this definition, a map is a subgraph of the ASes interconnection graph but in general it is not an induced subgraph.

Fig. 2 shows a sequence of exploration primitives applied to the map of Fig. 2(a). ASes 5583, 5484, and 6715 are explored in Fig's. 2(b), 2(c), and 2(d), respectively. Fig. 2 highlights several features of HERMES. HERMES can construct new drawings either using a static or a dynamic graph drawing algorithm. The drawing of Fig. 2(b) has been constructed with a dynamic algorithm starting



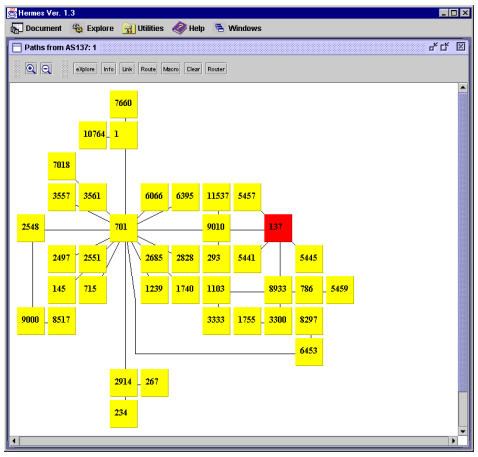
(a) Selection of AS 12300.

(b) Exploration of AS 5583.

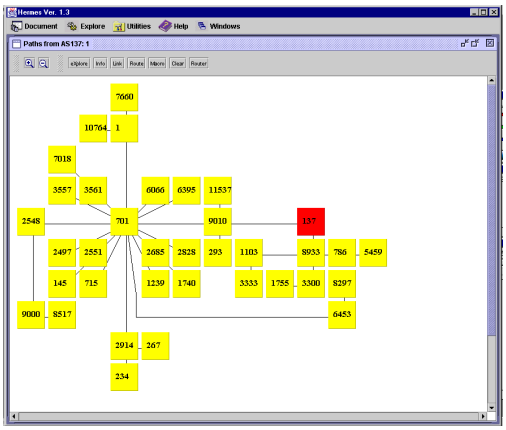
(c) Exploration of AS 5484.

(d) Exploration of AS 6715.

Figure 2: Exploration steps in the ASes graph. The selected AS is always drawn red.



(a) Map containing all the routes traversing AS 137.



(b) Map containing only routes starting from AS 137.

Figure 3: Selection of the routes traversing AS 137.

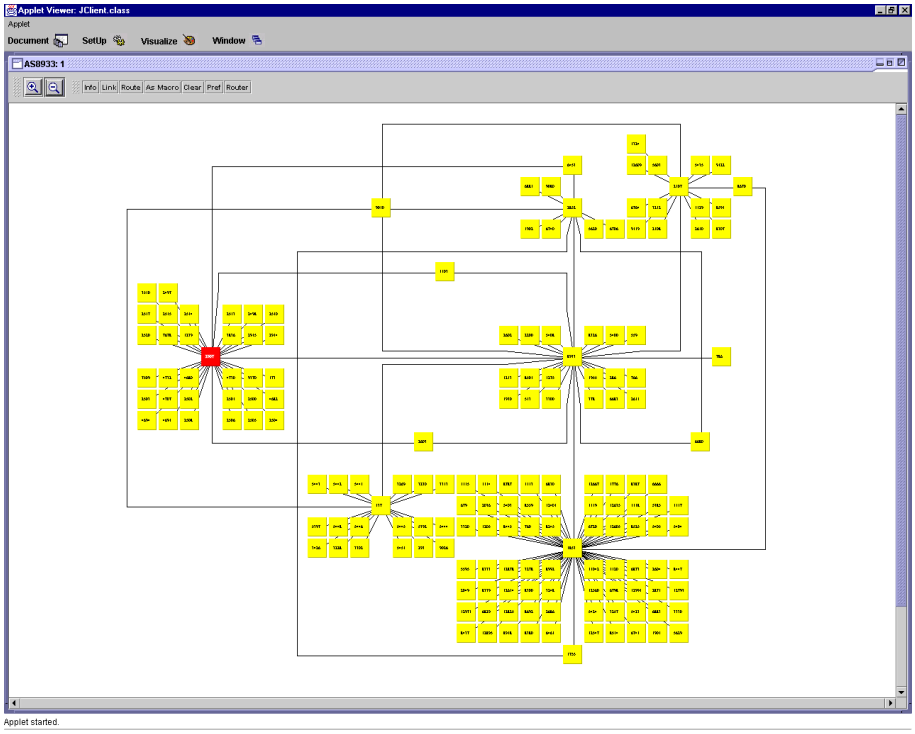


Figure 4: A map obtained with several exploration steps.

from the drawing of Fig. 2(a), and the drawing of Fig. 2(d) has been constructed with the same algorithm starting from the drawing of Fig. 2(c). Conversely, the drawing of Fig. 2(c) is obtained with a static algorithm. The choice of the algorithm to be applied can be done by the system (see Section 6) or forced by the user. Since the ASes degree can be large (see Section 5), in the project of the drawing algorithms of HERMES, special attention has been devoted to the representation of vertices of high degree. Fig. 2 shows how the vertices of degree one are placed around their adjacent vertices. A more complex map obtained with HERMES is depicted in Fig. 4. It contains more than 150 ASes.

Working on a map, independently on the way it has been obtained, the user can get several information on any AS:

General Info name, maintainers, and description of the AS. See Fig. 5.

Routing Policies For each connected AS, an expression describing the policy and its cost. See Fig. 5. This is possible both for in and for out policies. The default AS is also displayed.

Internal Routers List of the known border routers with the IP-numbers of the interfaces. Peering sessions with other routers are displayed.

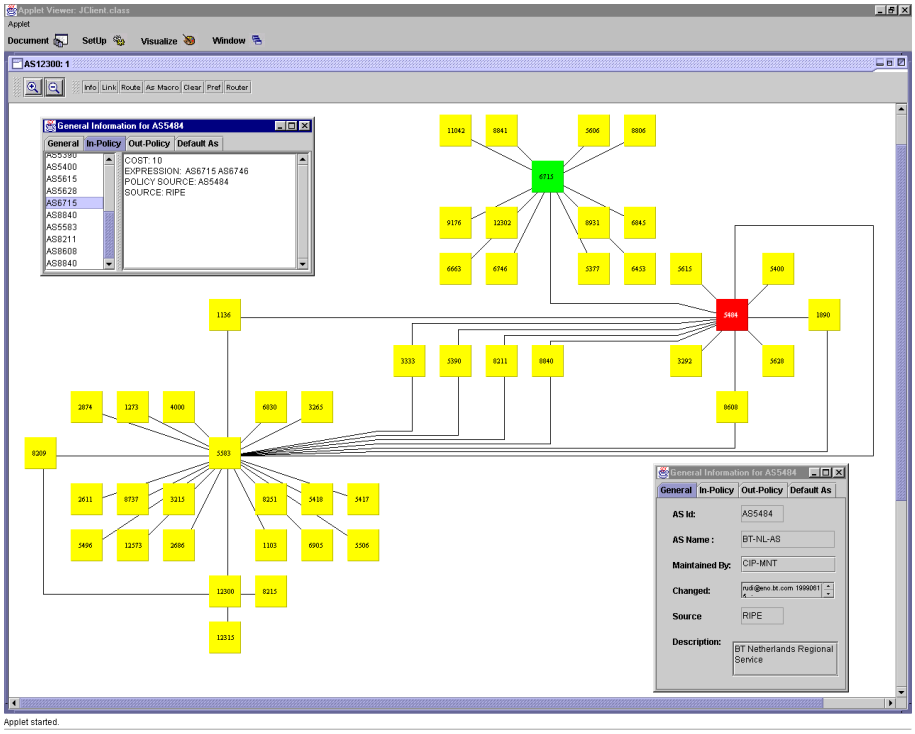


Figure 5: General Info and routing policies for AS 5484.

Routes List of the routes originated by the AS (see Fig. 6). It is also possible to visualize the propagation of a given route in the ASes composing the map.

AS Macros List of the macros [3] including the AS.

4 A Three-Tier Architecture

HERMES has a three-tier client/server architecture. The user interacts with a top-tier client which is in charge of collecting user requests and showing results. The requests are forwarded by the client to a middle-tier server which is in charge to process the raw data extracted from a repository (bottom-tier).

The client is a multi-document GUI-based application. It allows the user to carry-on multiple explorations of the ASes interconnection graph at the same time. The Java technology has been used to ensure good portability. Snapshots of the GUI have been shown in Section 3.

In HERMES the middle-tier server maintains the state of the session, that is the current map, for each connected user. The client communicates with the

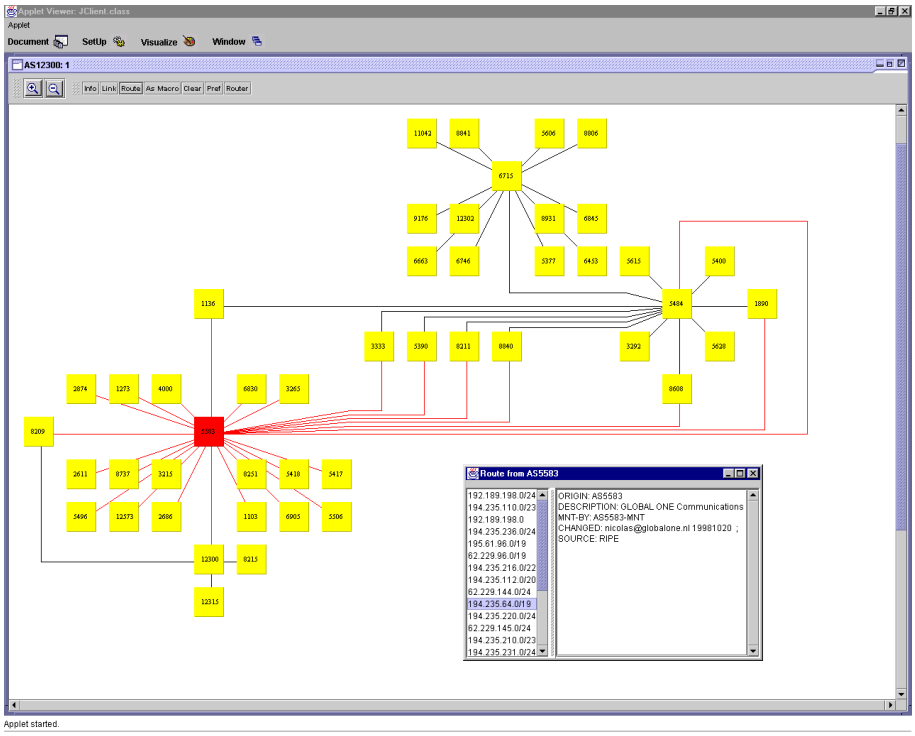


Figure 6: Routes through AS 5583.

server opening a permanent TCP connection for each user. This permits to amortize the inefficiency of the connection set up over all the requests of session. The protocol transported by such connections is specifically tailored for our application. In particular, the server sends its reply in the form of serialized software objects describing ASes, links, and related geometric information. The Java run-time environment transparently encodes objects into bytes on the middle-tier server and consistently decodes them on the client side.

The repository is updated off-line from a plurality of sources. At the moment we access the following databases adopting, for representing data, the RPSL language [1]: ANS, APNIC, ARIN, BELL, CABLE&WIRELESS, CANET, MCI, RADB, RIPE, VERIO. Further, we access the routing BGP data provided by the Route views project of the Oregon University [25]. However, the repository is easily extensible to other data sources.

Data are filtered so that only the information used by HERMES are stored in the database, but no consistency check or ambiguity removal is performed in this stage. The overall size of the repository is about 50 MB. The adopted DBMS technology is currently `mysql`.

The crucial part of the system is mainly located in the middle-tier. The top-tier requests two types of service to the middle-tier.

General info services the top-tier queries about ASes, routes, and path properties.

Topology services the top-tier queries for a new exploration and gets back a new map.

Info services requests are independent of each other and hence are independently handled by the middle-tier. On the contrary, topology services requests are always part of a *drawing session*. Each client may open one or more drawing sessions. Each drawing session is associated with a map that can be enriched by means of exploration requests.

Info services requests are directly dispatched to a *mediator*. The mediator module is in charge to retrieve the data from the repository and to remove ambiguities on-the-fly.

Topology services requests are handled by the kernel of the middle-tier. It gets information from the mediator and inserts new edges and vertices into the map. The drawing is computed by the *drawing engine* module (see Section 6). The drawing engine is based on the GDToolkit [21] library.

5 AS Interconnection Data from a Graph Drawing Perspective

In order to devise effective graph drawing facilities for HERMES, we have analyzed the ASes interconnection graph G. The data at our disposal[2] show the following structure for G.

[2]Observation of May 2000.

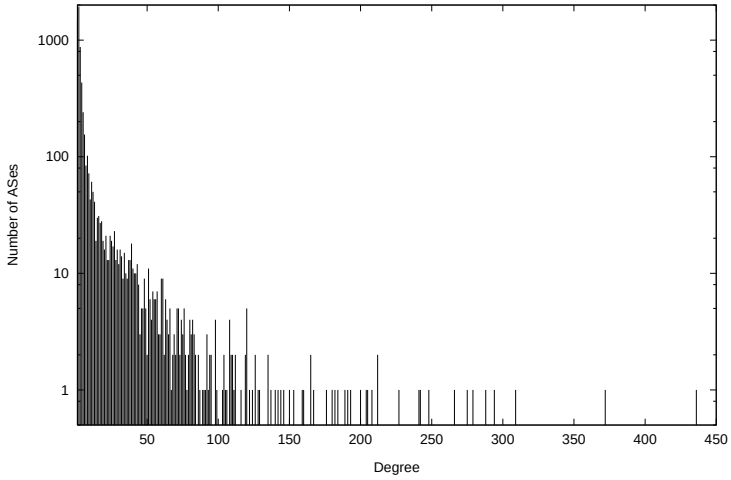


Figure 7: AS degree distribution (log. scale)

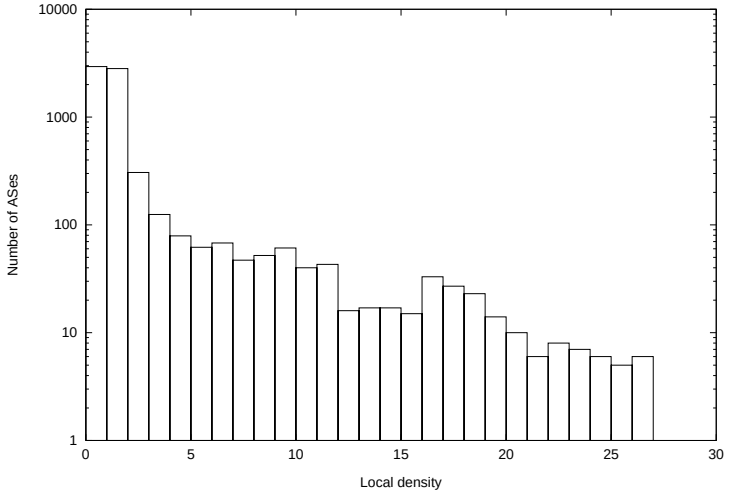


Figure 8: AS local density distribution (log. scale)

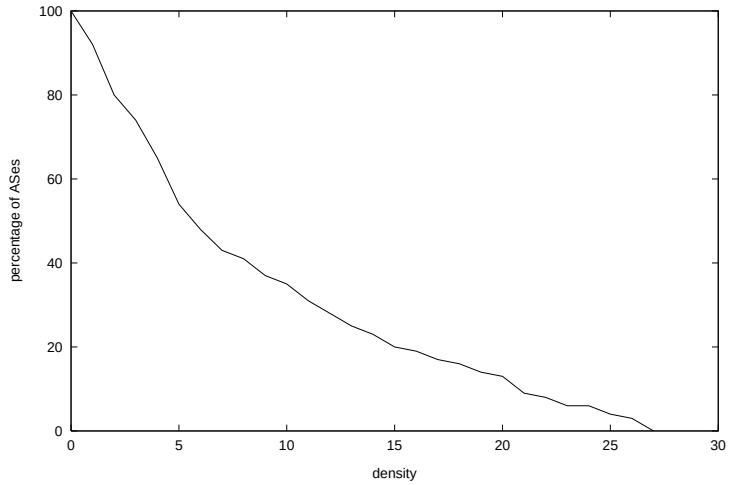


Figure 9: Percentage of ASes adjacent to an AS whose local graph has at least a given value of density

The number of vertices of G is $6,849$ and the number of edges is $27,686$. Fig. 7 illustrates the distribution of the degree of the vertices. The figure shows that while there are many vertices (about 75%) with degree less than or equal to 4, there are also several vertices whose degree is more than 100. For improving the readability of the chart, we have omitted two vertices with degree 862 and $1,044$, respectively. Further, consider that G contains 473 isolated vertices.

The density of G is 4.04. However, the "local" density can be much greater. In order to estimate such a local density, we have computed, for each vertex v, the density of the subgraph induced by the vertices adjacent to v. We call such graphs *local graphs*. Fig. 8 illustrates the distribution of the densities of the local graphs. From the figure it is possible to observe that about 5% of the local graphs have density greater than 10.

We have also tried to estimate the probability, for a user that explores G, to encounter a portion of G that is locally dense. Fig. 9 shows, for each value d of density, what is the percentage of vertices that are adjacent to a vertex whose local graph has density at least d. Note that more than 30% of the vertices are adjacent to a vertex whose local graph has density at least 10.

Concerning connectivity, the graph has 480 connected components, including the above mentioned 473 isolated vertices. One of them has $6,360$ vertices; each of the remaining 6 components has less than 6 vertices.

6 Drawing Conventions and Algorithms

We recall that the user interacts with the maps of the ASes interconnection graph G by means of the **AS exploration** primitive, described in Section 3. Namely, each time such a primitive is applied on a vertex u of the current map M, such a map is enriched with the vertices and the edges that are directly connected to u in G and that were not present in M. Then a drawing of the new current map is computed and displayed.

The choice of our exploration primitive for G is mainly motivated by the analysis performed on the structure of G (see Section 5). In particular, since G has many vertices that are adjacent to a vertex whose local graph has high density, we decided to discard a classical exploration approach based on induced subgraphs, because it is often lead to extremely dense maps.

In Section 6.1 we describe the drawing convention we adopt for visualizing the maps of G throughout a sequence of exploration steps performed by the user. In Section 6.2 we provide two different strategies for computing drawings of the maps within the defined drawing convention.

6.1 Drawing Convention

When the **AS exploration** primitive is applied on the current map M of G, the vertices of $G - M$ added to M will have degree one in the new current map. Also, these vertices of degree one are often numerous, due to the structure of G. Hence, for drawing a map we use a specific drawing convention that allows us to optimize the space occupied by the vertices of degree one.

Such a drawing convention is based on a variation of the podevsnef model for orthogonal drawings with high degree vertices (see Section 2.2). We modify the podevsnef model as follows. Each vertex of degree one adjacent to a vertex v is appropriately positioned on an integer coordinate grid around v and connected to v with a straight-line edge. Namely, as shown in Figure 10, vertex v is associated with a box partitioned into nine rectangles arranged into three rows and three columns. Denote these rectangles as B_{ij}, $(i, j \in \{1, 2, 3\})$. Rectangle B_{22} is used for drawing v centered on a grid point. Rectangles B_{11}, B_{13}, B_{31}, and B_{33} are used for drawing the degree-one vertices adjacent to v. Their incident edges are represented with straight-line segments, possibly overlapping other degree-one vertices. Actually, they are drawn on the back of the vertices. Rectangles B_{12}, B_{21}, B_{32}, and B_{23} are used for hosting the connections of v to the other vertices.

The height h_c of the center row is equal to one grid unit as well as the width w_c of the center column. Rectangles B_{11}, B_{13}, B_{31}, and B_{33} have all the same width w and height h. The values of h and w are expressed in terms of grid units and must guarantee enough room for placing all the degree-one vertices. How w and h are computed will be detailed in Section 6.3.

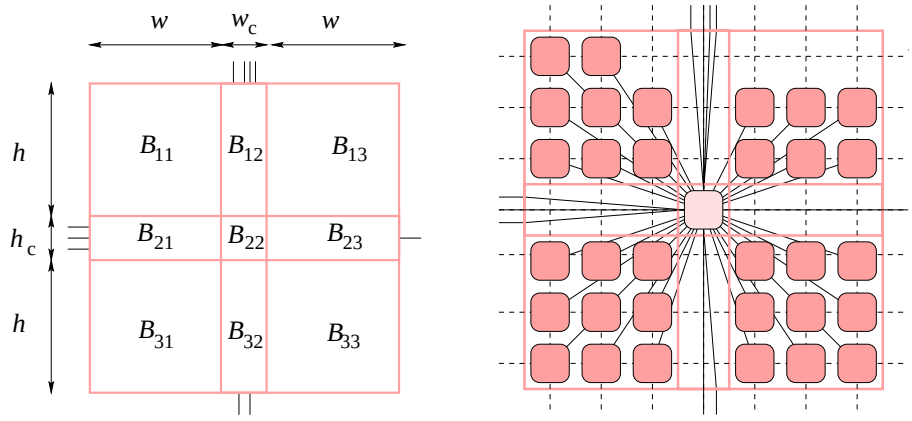


Figure 10: Using a box to make room around a vertex. (a) The nine rectangles that partition the box. (b) Using the nine rectangles to place the degree-one vertices. Each vertex in the box is centered on an integer grid point.

6.2 Algorithms Overview

In this section we describe two different strategies for computing drawings of the maps within the defined drawing convention, during a sequence of exploration steps. Namely, at each exploration step HERMES computes a new drawing of the current map, by applying one of the two following algorithms:

Static algorithm The current map is completely redrawn, after the new vertices and edges have been added.

Dynamic algorithm The new vertices and edges are added to the current drawing in such a way that the shape of the existing edges and the position of the existing vertices and bends are preserved "as much as possible".

Both the dynamic and the static algorithms have advantages and drawbacks. The dynamic algorithm allows the user to preserve its mental map but can lead, after a certain number of exploration steps, to drawings that are less readable than those constructed by the static algorithm. In fact, the dynamic algorithm makes use of local optimization strategies. The optimization strategies of the static algorithm are global and more effective. However, with the static algorithm, the new drawing can be quite different from the previous one and the user's mental map can be lost.

Because of the above motivations, HERMES automatically chooses between the static algorithm and the dynamic algorithm, according to the following criteria. Suppose v is the vertex that the user wants to explore. HERMES computes

a *dynamic exploration cost* associated with v. Such a cost represents an estimate of the efficiency and effectiveness of the dynamic algorithm with respect to the new exploration. In the current implementation of HERMES it simply depends on the kind and the number of drawing primitives (see Section 6.5) that would be needed for constructing the drawing of the new map with the dynamic algorithm and on how many consecutive times the dynamic algorithm has been invoked before the exploration of v. Once the exploration cost has been computed, HERMES compares it with a threshold value that can be set-up in a configuration menu of the system. If the exploration cost is lower than the threshold value, HERMES applies the dynamic algorithm, else it applies the static algorithm.

In the next two subsections we give details on how the static and the dynamic algorithms work in practice.

6.3 The Static Algorithm

Let G be the ASes interconnection graph and let M be the current map. We recall that M is a subgraph of G. Let v be the vertex of M explored by the user at the generic step. Map M is enriched with the vertices and the edges of G that are not present in M and are directly connected to v in G. The new map, which we still call M for simplicity, is fully redrawn according to the static algorithm, which consists of the following steps.

1. (*Degree-one Vertex Removal*) Vertices of degree one are temporarily removed from M. We call M' the new map. Each vertex u of M' is labeled by the number $\delta(u)$ of vertices of degree one that were attached to it.

2. (*Planarization*) A standard planarization [14] technique is applied to M'. In this phase a planar embedding of M' is computed and crossings among edges are represented by dummy vertices that will be removed later. We call such vertices *cross vertices*.

3. (*Orthogonalization*) A podevsnef representation of M' is constructed within the computed embedding.

4. (*Compaction*) A drawing for M' is computed from its orthogonal representation by assigning coordinates to vertices and bends. Since for each vertex u, our drawing convention requires that $\delta(u)$ vertices of degree one are placed around u, we must guarantee enough room for them (see Section 6.1). To do that we compute a podavsnef drawing by applying the algorithm described in [13]. Each vertex u is drawn as a box whose height h and width w depend on $\delta(u)$. We set $w = \lceil \sqrt{\delta(u)}/2 \rceil$. The height h is set equal to $= w - 1$ if this ensures enough room for all the vertices to be placed; else we set $h = w$. If $\delta(u) = 0$ then we set $h = w = 0$. We recall that w and h are expressed in terms of grid units. For example, referring to Figure 10, we have that $\delta(u) = 32$ (u is the vertex on the center of the box and has 32 vertices of degree one connected to it) and

then $w = 3$. Also, $h = 2$ is not sufficient for hosting all the 32 vertices into the four rectangles B_{11}, B_{13}, B_{31}, B_{33} (in fact, if $h = 2$ we can place at most $w \times h \times 4 = 3 \times 2 \times 4 = 24$ vertices inside the box). Hence h is set equal to 3 (in this way we can place up to 36 vertices inside the box; we use 32 positions for placing vertices and 4 positions will be not used). We also have to guarantee that the edges that connect u to non degree-one vertices are always incident on the middle points of the sides of the box. The basic version of the algorithm described in [13] allows the edges to freely shift along the side they are incident on. However, it is possible to easily adapt such an algorithm so that each edge is incident on a pre-assigned point. Finally, cross vertices are removed.

5. (*Degree-one Vertex Re-insertion*) The drawing is completed by replacing the box of each vertex u such that $\delta(u) > 0$ with a half unit square (edges that are incident on u are stretched). Also, the vertices of degree one that are incident on u are distributed in the rectangles B_{11}, B_{13}, B_{31}, and B_{33} and connected to u, according to the adopted drawing convention (see Figure 10(b)).

6.4 The Dynamic Algorithm

As in the case of the static algorithm, we denote by G the ASes interconnection graph and by M the current map. Also, denote by D the current drawing of M. Let v be the vertex of M explored by the user at the generic step.

The dynamic algorithm enriches M and D incrementally, by adding the vertices and the edges of G that are not present in M and are directly connected to v in G, in such a way that the user mental map is preserved as much as possible. Before describing how the dynamic algorithm works, we need to introduce some further notation.

Denote by ΔE_v and by ΔV_v the set of the edges and the set of the vertices to be inserted for exploring v. We recall that all the edges in ΔE_v are incident on v and that the vertices in ΔV_v will be degree-one vertices attached to v in the final drawing. For simplicity, we always use the notation M and D to denote the intermediate maps and drawings during the execution of the dynamic algorithm. In other words, we are assuming that M and D change dynamically throughout the algorithm execution. However, at the generic step of the dynamic algorithm, some of the vertices of M might be not explicitly represented in D. Namely, each vertex u of D might absorb all the degree-one vertices adjacent to it and implicitly represent the number of these vertices by a label $\delta(u)$, as in the case of the static algorithm. We call V_1 the set of degree-one vertices of M that are not explicitly represented in D, and V_2 all the vertices of M that are also explicitly represented in D. Note that, V_1 and V_2 partition the set of vertices of M, and that at the beginning of the dynamic algorithm V_1 is empty. Sets ΔV_v, V_1, and V_2 are modified during the algorithm execution.

A high level description of the dynamic algorithm is as follows (refer to Figure 11):

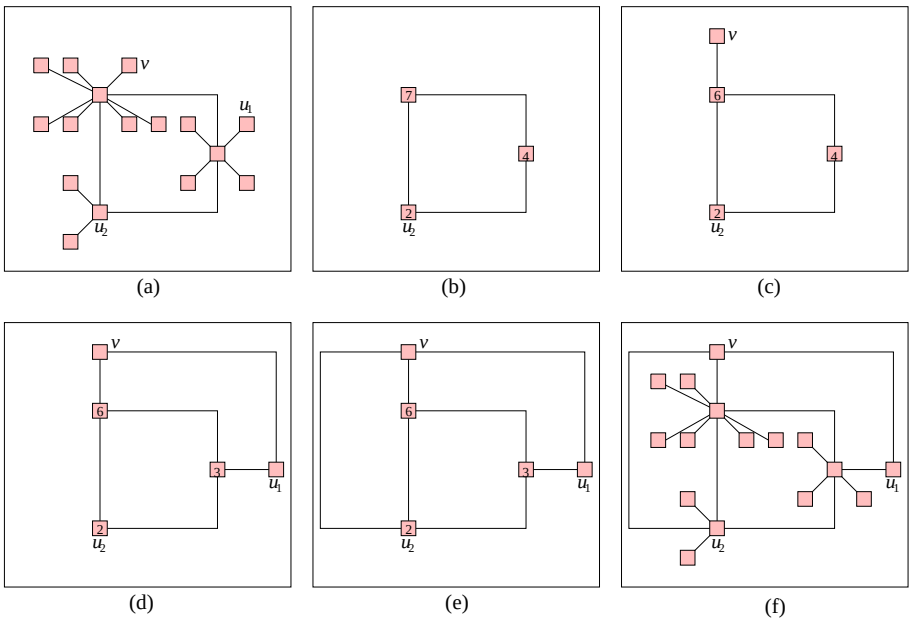


Figure 11: An example of the dynamic algorithm. (a) The initial drawing of a map; the user chooses to explore vertex v; suppose that v is directly connected to u_1 and u_2 in the ASes interconnection graph. (b) In the first step of the dynamic algorithm all the degree-one vertices of the drawing are temporarily removed; the remaining vertices are labeled with the number of vertices of degree one that were connected to them. (c) Vertex v is reinserted by using the Attach-Vertex primitive. (d) Vertex u_1 is reinserted by using the Attach-Vertex primitive and then edge (v, u_1) is inserted by using the New-Edge primitive. (e) Edge (v, u_2) is inserted by using the New-Edge primitive. (f) The compaction step is applied and the removed degree-one vertices are completely reinserted.

Step 1 We temporarily remove from D all the degree-one vertices. Each vertex u of D is labeled with the number $\delta(u)$ of vertices of degree one that were attached to it (see Figure 11 (b)). The deleted vertices are moved from V_2 to V_1. Note that, D is now a podevsnef drawing in the standard sense, where edge crossings are still replaced by cross vertices.

Step 2 We incrementally add to M the edges of ΔE_v and the vertices of ΔV_v. At the same time, specific subsets of edges and vertices of M are added to D, by applying on D a sequence of two primitives that modify the drawing within the podevsnef standard. The two primitives are as follows:

New-Edge(u,z) A new edge is added to the drawing between the two vertices u and z; vertices u and z must be already explicitly represented in D.

Attach-Vertex(u) A new vertex z is added to D and connected to u with a new edge (u, z); vertex u must be already explicitly represented in D.

How such primitives work in practice will be detailed in Section 6.5. The insertion of vertices and edges in M and D is performed according to the following ordered set of rules:

- If the explored vertex v belongs to V_1 (that is, if v is a one-degree vertex in M), we call w the vertex of V_2 that is connected to v in M. We reinsert v in D (that is, we explicitly represent v in D) by performing primitive Attach-Vertex(w) (see Figure 11 (c)). Consistently, we decrease $\delta(w)$ by a unit, set $\delta(v) = 0$, and move v from V_1 to V_2.
- For each edge $e = (v, u)$ of ΔE_v we insert in M vertex u, if it is not already present in M, and edge e. After that, drawing D is modified according to the following three cases:
  1. If u is in V_1, we reinsert (explicitly represent) u in D by performing primitive Attach-Vertex(z), where z is the vertex of V_2 connected to u in M; then we add e to D, by applying primitive New-Edge(v,u) (see Figure 11 (d)). Consistently, we decrease $\delta(z)$ by a unit, set $\delta(u) = 0$ and move u from V_1 to V_2 .
  2. If u is in V_2, we add edge e to D by performing primitive New-Edge(v,u)(see Figure 11 (e)).
  3. If u is in ΔV_v we just increase $\delta(v)$ by a unit, and move u from ΔV_v to V_1 (u will be implicitly represented in D).

  Once all edges in ΔE_v have been considered ΔV_v is empty, M is completely updated with the vertices and edges selected for insertion by the exploration of v, and the only vertices that remain to be added to D (those in V_1) are all the vertices (different from v) that have degree one in M.

Step 3 We perform on D the *Compaction* step and the *Degree-one Vertex Reinsertion* step described for the static algorithm. The degree-one vertices reinserted are those in V_1 (see Figure 11 (f)).

6.5 Primitives of the Dynamic Algorithm

In this section we conclude the description of the dynamic algorithm by explaining how primitive New-Edge and Attach-Vertex work.

We recall that these primitives modify a podevsnef drawing D preserving this drawing standard. Both the primitives compute the position of the new vertices and edges trying to optimize, at the same time, the following measures: number of crossings, number of bends, and edge length. Each of these measures has a prescribed cost that can be passed as a parameter to the primitives.

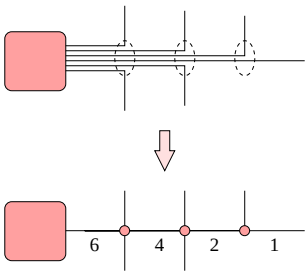


Figure 12: Collapsing the edges incident on the same side of a vertex. The labels represent the thickness of the edges. The small circles are the chain vertices.

In this way it is possible to decide the priority of each measure in the whole optimization. Once any of the primitives has been applied, the new drawing is guaranteed to have the same shape as the previous one, for the common parts.

The two primitives use two auxiliary data structures to perform their work. These data structures are a simplified orthogonal representation of the current drawing and a directed network associated with this orthogonal representation. We now describe these two data structures. After that, we shall describe how they are used by the primitives.

Let H be the orthogonal representation of D. The first data structure is an orthogonal representation H' obtained from H in the following way:

- H is simplified so that all vertices have degree less than or equal to four. This is done with a standard technique adopted in the podevsnef model [4], where all the edges incident on the same vertex from the same side are collapsed into a chain of edges (see Fig. 12). Each edge of the chain replaces a certain number of edges (possibly only one). We associate with each edge a *thickness* representing the number of edges replaced by it. We call the new vertices inserted by this operation *chain vertices*.

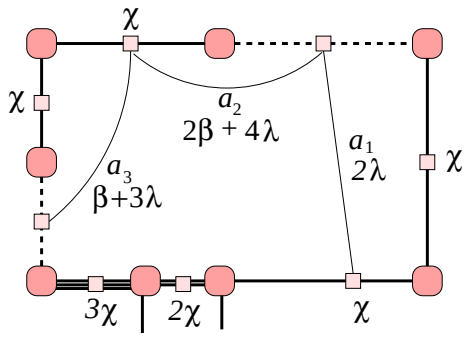
- Each face of H (including the external one) is decomposed into rectangles by adding a suitable number of dummy edges and vertices, with the linear time algorithm described in [32]. We call *dashed* the dummy edges and *solid* the edges of the original orthogonal representation.

The second data structure is a directed network N associated with H'. We call such a network the *incidence network* of H'. N is used to implicitly describe all the orthogonal paths that a new edge can follows in H'. Further, N is defined so that each path has an associated cost that reflects the cost of the new edge in terms of bends, edge crossings, and edge length. We denote by χ, β, and λ the costs of one crossing, one bend, and one edge length unit, respectively. As already mentioned at the beginning of this section, these costs can be set-up by the user in order to determine the priority level of the three different measures during the optimization.

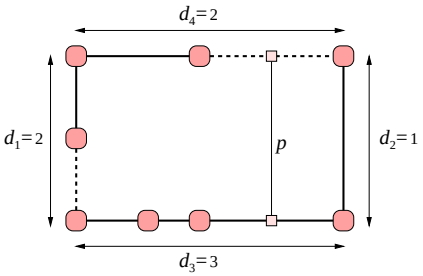
Network N is defined as follows (see Fig. 13(a)):

- N has a node v associated with each edge e (solid or dashed) of H'. Also, v has an associated cost that represents the cost of crossing e. Namely:
  - If e is a solid edge then the cost of v is set-up equal to the thickness of e multiplied by χ (see Fig. 13(a)). This reflects the fact that a path of N traversing v corresponds to a new edge of H' traversing edge e, and hence to a new edge of H that crosses a number of edges equal to the thickness of e.
  - If e is a dashed edge then the cost of v is set-up equal to zero. This is because dashed edges of H' are dummy and will be removed in the final drawing. Hence, they do not really originate edge crossings.

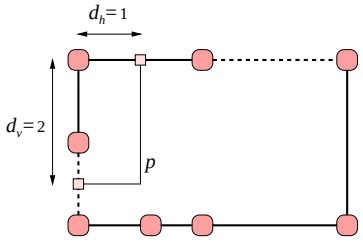
- N has an arc between every pair of nodes associated with two edges e_1 and e_2 in the same face f of H'. Such an arc represents an orthogonal path inside f, which can be used to reach e_2 from e_1 (or vice-versa) in H'. We distinguish three different kinds of arcs of N with respect to a face f of H' (refer to Fig. 13):
  - An arc a between nodes associated with two horizontal (vertical) edges that lie on different sides of f (see, for example, arc a_1 in Fig. 13(a)). In this case, we associate with a a straight-line path p (a path with no bends) inside f, because it suffices to move from a side of f to its opposite in the orthogonal representation (see Fig. 13(b)). Hence, denoted by d_1, d_2 (d_3, d_4) the lengths (in terms of number of edges) of the vertical (horizontal) sides of f, we assign cost $d\lambda$ to arc a, where $d = max\{d_1, d_2\}$ ($d = max\{d_3, d_4\}$). Such a cost is a lower bound on the length of p.
  - An arc a between a node associated with a horizontal edge e_h and a node associated with a vertical edge e_v of f (see, for example, arc a_3 in Fig. 13(a)). In this case, we associate with a an orthogonal path p with exactly one bend, as depicted in Fig. 13(c). Denoted by s_v the side of f on which e_v lies, let d_v be the number of edges of s_v that are necessarily spanned (completely or partially) by the projection of p on s_v. Analogously, denoted by s_h the side of f on which e_h lies, let d_h be the number of edges of s_h that are necessarily spanned by the projection of p on s_h. Hence, the cost we assign to a is equal to $\beta + (d_v + d_h)\lambda$. In particular, $(d_v + d_h)\lambda$ still represents a lower bound on the length of p, while β is the cost for one bend.
  - An arc a between the nodes associated with two edges e_1 and e_2 that lie on the same side of f (see, for example, arc a_2 in Fig. 13(a)). In this case, we associate with a an orthogonal path p with two bends, as shown in Fig. 13(d). Denoted by s the side of f in which lie the two edges, let d be the number of edges of s that are necessarily spanned (completely or partially) by p. Hence, the cost we assign to a is equal to $2\beta + (d+2)\lambda$. The two extra units for the length are due



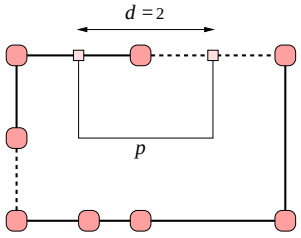
(a) Nodes (little squares) and three arcs of the incidence network for a face of a simplified orthogonal representation.



(b) The orthogonal path p associated with arc a_1.



(c) The orthogonal path p associated with arc a_3.



(d) The orthogonal path p associated with arc a_2.

Figure 13: Example of construction of an incidence network and related orthogonal paths.

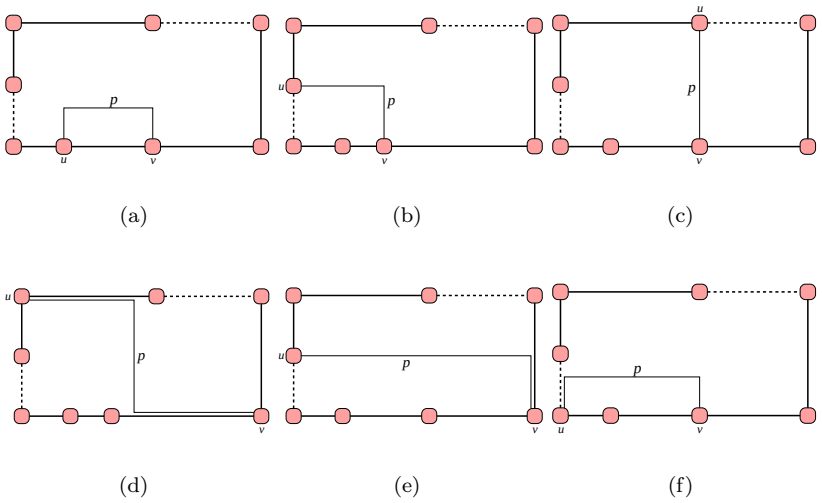


Figure 14: Possible cases of orthogonal paths between two vertices of the same face of an orthogonal representation.

to the fact that p consists also of two (unit-length) segments having a direction (vertical or horizontal) orthogonal to the direction of s.

We now explain how primitives New-Edge and Attach-Vertex perform on the current drawing D.

- Primitive New-Edge(u,v) computes from D the simplified orthogonal representation H' and the associated network N' above described. Clearly, from the point of view of the primitive interface, H' and N are transparent. It just needs to know u, v and D. After that, two different cases are possible:

  Case 1 Nodes u and v belong to two different faces of H'. In this case, the primitive completes network N by adding two extra nodes representing u and v. For simplicity we still refer to these extra nodes as u and v. Also, for each edge e of H' incident on u (resp. v) the primitive adds to N an arc between u (resp. v) and the node of N associated with e. All these extra arcs of N have zero cost. At this point, the primitive computes on N a shortest path between u and v. Such a path determines the route and the shape of the new edge (u, v) in H', according to the rules illustrated in the construction of N. Namely, the new edge is added to H' following the arcs of the shortest path. Note that, the route and the shape of the new edge in H' uniquely induces the route and the shape of the same edge in H. Hence, the primitive adds edge (u, v) to H and computes the new

drawing D by compacting H with a standard compaction algorithm for podevsnef orthogonal representations [20, 4].

Case 2 Nodes u and v belong to the same face. In this case the shape of edge (u, v) is simply chosen according to the set of cases shown in Figure 14.

- Primitive Attach-Vertex(u) works much simpler then New-Edge. It must add a new edge e that connects a new node to u. The primitive looks in H' for a side s of u such that either no edge is incident on s or a dashed edge is incident on s. If such a side s exists, the primitive add to H', and therefore to H, edge e as a straight edge incident on s. Otherwise, the primitive chooses an arbitrary side of u to insert edge e, and in this case edge e will have one bend in H. Finally, the new drawing D is computed by compacting H with a standard compaction algorithm for podevsnef orthogonal representations.

Note that, if a sequence of consecutive dynamic primitives have to be applied before D is displayed (as it often happens in a single HERMES's exploration step), the computation of D can be delayed until the end of the sequence. In this case, H' and N can be kept up to date after each primitive is executed instead of reconstructing them each time.

7 Statistics on the Hermes Service

We implemented Hermes as a service publicly available on the Web[3]. In this section we intend to show the effectiveness of the service. To this aim we provide several statistics obtained from data collected from September 2000 to November 2001. The total number of connections to the service during the considered period has been $4,063$. The number of distinct users (that is, distinct IP addresses) that accessed the service has been 860.

Figure 15(a) shows how many connections (y-axis) have been performed by a certain number of users (x-axis). Such a chart provides hints about the loyalty of the users to the Hermes service. About 70 users have accessed the service more than 10 times. For several of them we observed usage peaks of one or two days at distance of months.

Figure 15(b) shows the percentage of connections (y-axis) in which the user performed a certain number of exploration steps (x-axis). This chart provides information about the overall usability of the service which may be affected by drawing performances, reply speed, significance of the data, etc. A high number of exploration steps suggests that the quality of the drawing engine and its performance meet the user needs. However, consider that, due to the high local density of the ASes interconnection graph, after a certain number of exploration steps (usually 4 or 5) the visualized map often becomes quite large

[3] http://www.dia.uniroma3.it/~hermes/

(more than 200 vertices) and the information difficult to read. In these cases, the user usually prefers to interrupt the exploration of the current map and restarts the whole exploration from another AS. The chart shows that for about 20% of the connections, the number of exploration steps has been greater than or equal to 3. In some cases the number of steps has been greater than 10.

Figure 15(c) shows the distribution of the users within the Internet top level domains[4]. About half of the users belongs to domains *.net* and *.com* and are mainly Internet service providers and companies with interests in networking. About 20% of domains *.it* is due to system development and testing.

8 Conclusions

In this work we presented HERMES, a system for exploring and visualizing the interconnections between Autonomous Systems. In particular, we discussed the architecture of HERMES and we described efficient algorithms used by the system to compute pleasing drawings of ASes interconnection subgraphs. We explained how the adopted drawing convention and algorithms have been motivated by the structure of the ASes interconnection graph.

In the near future we plan to enrich HERMES with new functionalities for visualizing the internal structure of an AS, and with tools for evidencing inconsistency in the data sources.

From the point of view of the drawing algorithms, there are several open problems that can be considered. A limited list of such problems follows:

- Often, after many exploration steps the drawing tends to become very large, so increasing both the computational complexity of successive exploration steps and the difficulty for the user to understand and to interact with the map. Hence, it would be interesting to devise strategies for deleting "old vertices" in the maps throughout the exploration.
- To increase the effectiveness of the visualization, it could be useful to assign to the vertices different sizes, depending on the different "importance" of the corresponding ASes. To this aim, an extension of the used convention and algorithms to drawings with vertices of prescribed size is needed.
- It would be interesting to extend the applicability of our drawing algorithms to other domains, such as state diagrams, class diagrams, etc?

Acknowledgements

We are grateful to Sandra Follaro and Antonio Leonforte for their fundamental contribution in the implementation of the dynamic algorithm. We are also grateful to Andrea Cecchetti for useful discussion on the repository.

[4]The data are obtained from 60% of all users. The remaining part of the users is not associated to any domain.

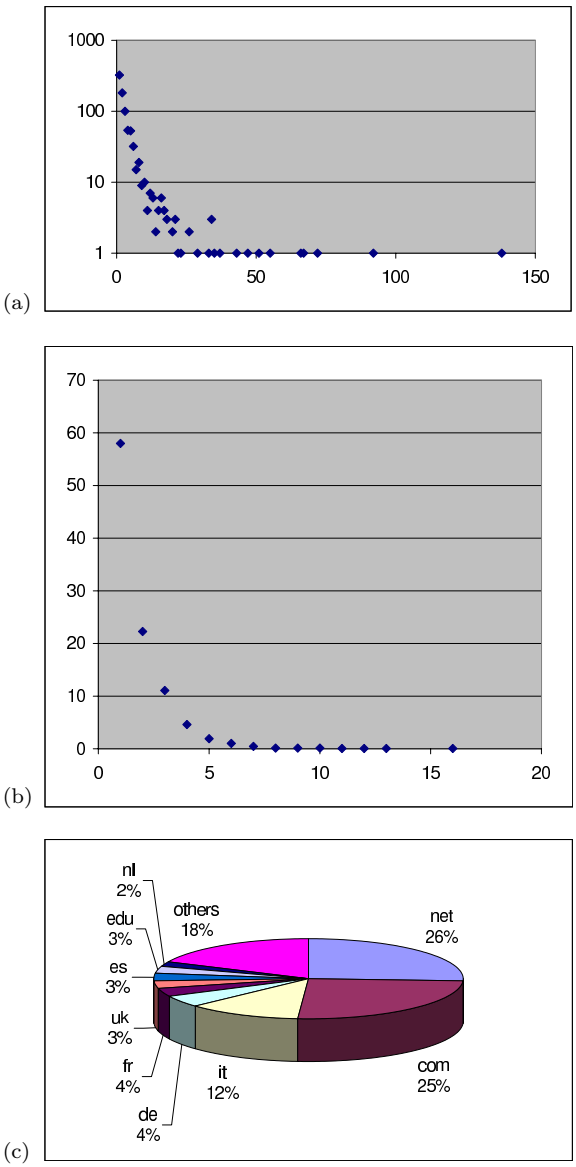


Figure 15: (a) Number of connections (y-axis) performed by a certain number of users (x-axis). (b) Percentage of connections (y-axis) in which the user performed a certain number of exploration steps (x-axis). (c) Distribution of the users within the Internet top level domains.

References

[1] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing policy specification language (rpsl). On line, 1999. rfc 2622.

[2] Aprisma. Spectrum. On line. http://www.aprisma.com.

[3] T. Bates, E. Gerich, L. Joncheray, J. M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of ip routing policies in a routing registry. On line, 1994. ripe-181, http://www.ripe.net, rfc 1786.

[4] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum numbr of bends. *IEEE Transactions on Computers*, 49(8), 2000.

[5] T. C. Biedl and M. Kaufmann. Area-efficient static and incremental graph darwings. In R. Burkard and G. Woeginger, editors, *Algorithms (Proc. ESA '97)*, volume 1284 of *Lecture Notes Comput. Sci.*, pages 37–52. Springer-Verlag, 1997.

[6] U. Brandes and D. Wagner. Dynamic grid embedding with few bends and changes. In K.-Y. Chwa and O. H. Ibarra, editors, *ISAAC'98*, volume 1533 of *Lecture Notes Comput. Sci.*, pages 89–98. Springer-Verlag, 1998.

[7] S. Bridgeman and R. Tamassia. Difference metrics for interactive orthogonal graph drawing algorithms. In S. H. Withesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 57–71. Springer-Verlag, 1998.

[8] S. S. Bridgeman, J. Fanto, A. Garg, R. Tamassia, and L. Vismara. InteractiveGiotto: An algorithm for interactive orthogonal graph drawing. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 303–308. Springer-Verlag, 1998.

[9] CAIDA. Otter: Tool for topology display. On line. http://www.caida.org.

[10] CAIDA. Plankton: Visualizing nlanr's web cache hierarchy. On line. http://www.caida.org.

[11] R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and planar ST-digraphs. *SIAM J. Comput.*, 24(5):970–1001, 1995.

[12] Cornell University. Argus. On line. http://www.cs.cornell.edu/cnrg/topology_aware/discovery/argus.html.

[13] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of prescribed sizes. In J. Kratochvil, editor, *Graph Drawing (Proc. GD '99)*, volume 1731 of *Lecture Notes Comput. Sci.*, pages 297–310. Springer-Verlag, 1999.

[14] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.

[15] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–325, 1997.

[16] G. Di Battista, R. Lillo, and F. Vernacotola. Ptolomaeus: The web cartographer. In S. H. Withesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 444–445. Springer-Verlag, 1998.

[17] M. Dodge. An atlas of cyberspaces. On line. http://www.cybergeography.com/atlas/atlas.html.

[18] P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 330–335. Springer-Verlag, 1997.

[19] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proceedings of Compugraphics 91*, pages 24–33, 1991.

[20] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 254–266. Springer-Verlag, 1996.

[21] GDToolkit:. Graph drawing toolkit. On line. http://www.gdtoolkit.com.

[22] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1972.

[23] B. Huffaker. Tools to visualize the internet multicast backbone. On line. http://www.caida.org.

[24] IPMA. Internet performance measurement and analysis project. On line. http://www.merit.edu/ipma.

[25] D. Meyer. University of oregon route views project. On line. http://www.antc.uoregon.edu/route-views.

[26] K. Miriyala, S. W. Hornick, and R. Tamassia. An incremental approach to aesthetic graph layout. In *Proc. Internat. Workshop on Computer-Aided Software Engineering*, 1993.

[27] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Lang. Comput.*, 6(2):183–210, 1995.

[28] S. North. Incremental layout in DynaDAG. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 409–418. Springer-Verlag, 1996.

[29] A. Papakostas and I. G. Tollis. Interactive orthogonal graph drawing. *IEEE Transactions on Computers*, 47(11):1297–1309, 1998.

[30] C. Rachit. Octopus: Backbone topology discovery. On line. http://www.cs.cornell.edu/cnrg/topology_aware/topology/Default.html.

[31] Y. Rekhter. A border gateway protocol 4 (bgp-4). IETF, rfc 1771.

[32] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[33] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.