Report

DEA in Bioinformatics
2001 - 2002

Lausanne and Geneva Universities

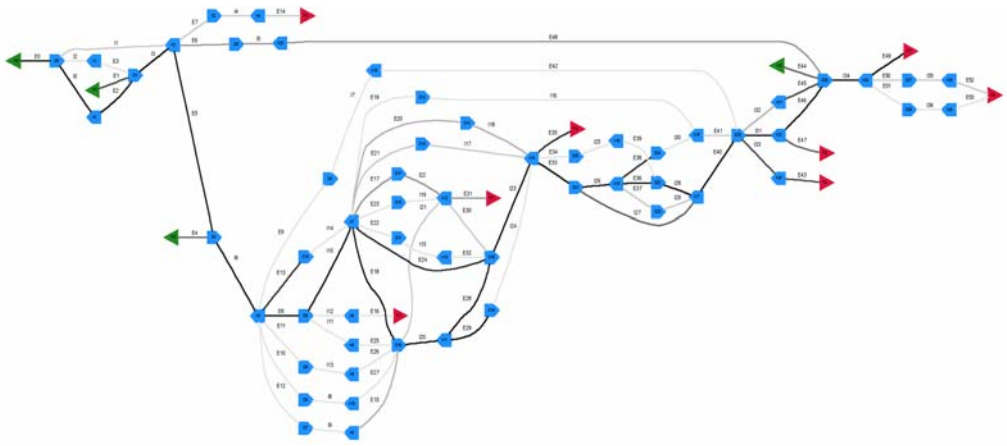
Swiss Institute for Bioinformatics

Representing human genes with graphs :

a graphic web interface.

Michel Crausaz

michel.crausaz@etu.unil.ch



Graph of the nucleophosmin gene (nuclear phosphoprotein B23 or numatrin, SwissProt Nbr. P06748).

11/12/2002

Table of Contents

1. Introduction.

The sequencing of the human genome, the documentation of its transcribed portion, called transcriptome, as well as the observation of mechanisms like alternative initiation of transcription, splicing and polyadenylation are producing a huge amount of data. The profusion of information increase the difficulty to convey them in a comprehensible manner. Tromer is a program producing computer-friendly graphs based on alignments of human genome with known RNA elements. The aim of this work is to transform the graph output of Tromer to a useful image map wrapped in a web interface.

After overview of what lies in transcriptome and general graph drawing, the DOT graph drawing algorithm is explain as well as the language going with. Then, the Tromer2Map concept is presented, first on the PERL script aspect and, secondly, on the graphic aspect. Finally, further possible developments are outlined.

2. Transcriptome.

The transcriptome can be simply defined as the transcribed portion of the genome. This short definition hides a mass of biological information particularly in the term "transcribed". A single gene can produce a lot of different transcripts. Most of them are still not characterized with regard to their function, their abundance on mRNA or protein level as well as their localization in tissues or developmental stages. The alternatively processed forms of human transcripts are derived from different transcription initiation sites, alternative exon splicing and multiple polyadenylation sites (*Strausberg and Riggins*). All these transformations magnify the number of transcripts toward the activation of a single gene.

Analysis of the human transcriptome originates from the expressed sequence tag (EST) technology (*Strausberg and Riggins*). This technique produced, by reverse transcription and amplification, nucleic acid fragments based on messenger RNA as template. EST sequences express a part of the transcribed genome and can be used as tag on the genome to fish genes. A modification of the EST strategy is called ORESTES

(ORF ESTs). In this approach, sequences are produced along the length of transcripts rather than just from clone extremities (*Camargo et al.*). Various sequences of complete RNA messenger, published gene list on human chromosome 21 or high throughput cDNA are also sources to determine presence or absence of transcripts on the human genome.

2.1. Data.

A human transcriptome database was produced using different data sources. On the genomic level, data used were the NCBI human chromosome contigs NT_*. Transcribed RNA data were extracted from several sources. These were the human EST section , the human HTC section and the human mRNA, all from EMBL, and, also, the ORESTES sequences, the human mRNA from the RefSeq database in NCBI as well as the published gene list of human chromosome 21 (*Iseli*). Another source of information is given by the 3' tags produced in silico as mentioned in *Iseli et al*. Rapidly, tags were extracted from trace files by looking at the 50 nuclotides that lay directly before the longest poly-A tract. After filtering and clustering, they obtained more than 95,000 trusted genome-matched tag clusters.

The transcript sequences were filtered to remove vector or E. coli sequences. Repetitive sequences and RNA elements were masked out. Megablast was used to identify pair-wise similarities between all transcript sequences and the genomic data. For each pair of matching, local alignments were generated by sim4. The output was then filtered to ensure that the whole alignment had an average good or acceptable quality. Finally, the graphical displays of AceDB were used to create an integrated view of the data (figure 1) (*Isely*).

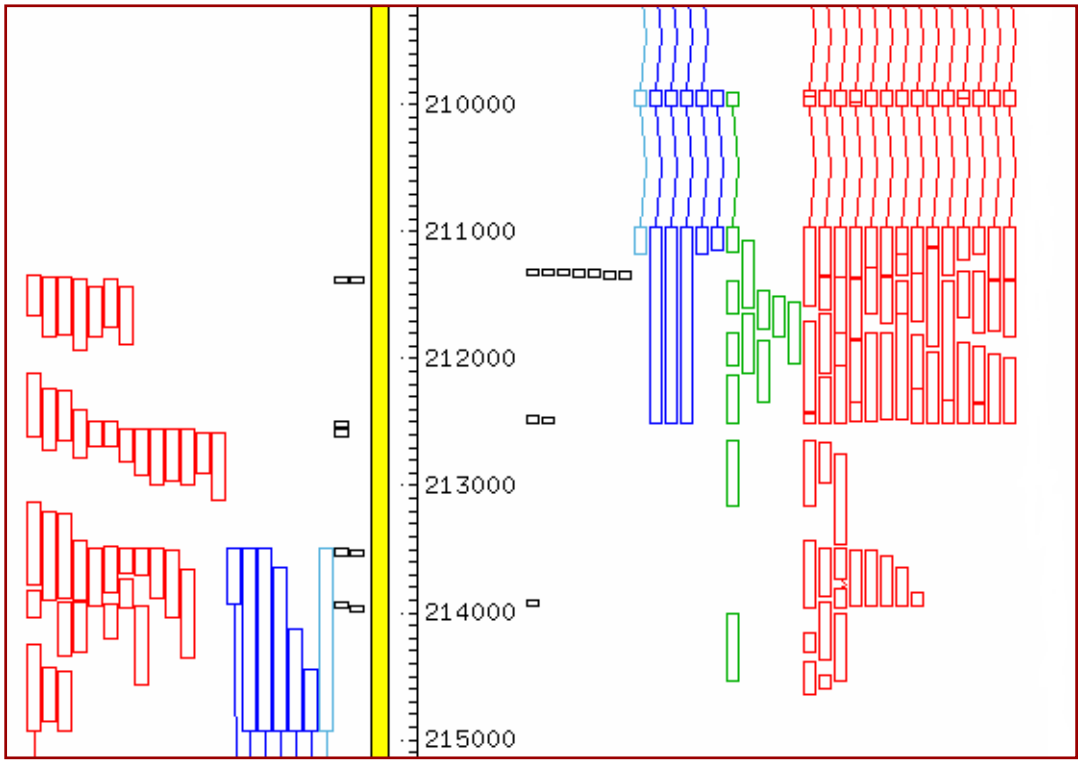


Fig. 1 : Alignments of transcripts to the genome (yellow bar) were visualized using AceDB. The direction of transcription is bottom to top on the left and top to the bottom on the right. Light-blue: RefSeq sequences; dark-blue: full-length cDNA sequences; green: ORESTES sequences; red: EST sequences. 3'tags are represented by blackboxes, with one box per cluster member.

2.2. Tromer.

The Tormer program (Transcriptome Analyzer) attempts to automate the reconstruction of transcripts from the mapping of RNA data mentioned above to genomic DNA contigs. The output of sim4 is used to construct a set of oriented exons. Virtual transcripts are reconstituted by following exon boundaries with known polarity When several paths can be followed, multiple transcripts can be generated. These multiple transcripts are evidence for alternative splicing or polyadenylation. At this point, the merging of these paths on a single genomic sequence can be considered as a gene and represented as a connected oriented graph where nodes are splice-donors or splice-acceptors and edges are exons or introns (*Iseli; Iseli et al.*).

2.3. Graph output.

The first line of the text file, which starts with ">", gives information on the graph itself with an accession number (NT_026281_10), a reference to the NCBI DNA contig with its version number (NT_026281.7), the chromosome where the sequence come from and the position on the DNA contig (347746..383506).

```
>map|NT_026281_10|NT_026281.7|- Chromosome 5 347746..383506
```

Then, the graph output consists of two parts : a description of the graph structure and a detailed list of the exons and introns composing the edges of the graph.

2.3.1. Graph description.

The description give the connectivity between each nodes as well as the names of these nodes and their associated edges. Nodes are represented by four possible capitals. H denote the start of the transcritp which is not necessarily the transcription start site. D are splice-donor sites and A are splice-acceptor site. T denote the end of a transcript which, again, is not necessarily the transcription stop site. Edges are symbolized by two possible capitals, E for exons and I for introns. Each capital gets an increasing index from 0 to infinity (if it is possible ?).

Let us have a look at the first node of the graph in figure 2. The node is labeled H0 and its position on the DNA contig is at 383,506 bps. Moreover, it has no incoming edge, but one outgoing edge with exon E0 going to node D0 and only one event shows or proves the existence of such an exon (see below). The last number (here −30) gives an idea about the confidence of this assertion.

To be sure about a total comprehension of everybody, let us make another example. The node A4 is at position 347,956 on DNA contig NT_026281 version 7. It has two incoming edges (I5 from D4, I4 from D3) and one outgoing edge (E6 to D5).

```
H0 [383506] 0,1              -I3-> A3 1,-10              H1 1,-21 =E5=>
 =E0=> D0 1,-30              -I2-> A2 1,-10               -I5-> A4 1,-10
D0 [383386] 1,1             A2 [362874] 1,1             A4 [347956] 2,1
 H0 1,-30 =E0=>              D2 1,-10 -I2->              D4 1,-10 -I5->
 -I0-> A0 1,-10              =E3=> T0 2,-30              D3 1,-10 -I4->
A0 [375227] 1,1             T0 [361968] 1,0              =E6=> D5 2,-51
 D0 1,-10 -I0->              A2 2,-30 =E3=>             D5 [347746] 1,1
 =E1=> D1 2,-60             A3 [355204] 1,1              A4 2,-51 =E6=>
D1 [374983] 1,1              D2 1,-10 -I3->              -I6-> A5 1,-10
 A0 2,-60 =E1=>              =E4=> D3 1,-30             A5 [336319] 1,1
 -I1-> A1 2,-20             D3 [355078] 1,1              D5 1,-10 -I6->
A1 [373531] 1,1              A3 1,-30 =E4=>              =E7=> T1 2,-21
 D1 2,-20 -I1->              -I4-> A4 1,-10             T1 [335999] 1,0
 =E2=> D2 2,-60             H1 [350898] 0,1              A5 2,-21 =E7=>
D2 [373393] 1,2              =E5=> D4 1,-21
 A1 2,-60 =E2=>             D4 [350705] 1,1
```

Fig. 2 : Description part of graph `NT_026281_10`. Here information is given in three part, normally it is continuous. See text for explanation.

2.3.2. Detailed list.

After graph description, each exon and intron are listed with its start and end positions on the DNA contig. For exons, positions are followed by the number of 3' tag falling within its bounds and by the number of associated RNA elements (what was called proof of existence above). For example, exon E6 starts at position 347,956 and ends at position 347,746 of the DNA contig NT_026281.7, contains none 3' tag, but two associated RNA elements. Potential alternative splicing are given as a list of conflicting introns after the exon description.

Each RNA elements is listed individually on a line. The first capital give the source of the element. This can be :

E for EST section of EMBL,	R for RefSeq sequences
O for ORESTES,	H for HTC from EMBL
M for human mRNA from EMBL,	U for other.

The next elements are the accession number in the defined section, the RNA nucleotides used in this exon and the corresponding nucleotides on the DNA.

For introns, the start and end positions are followed by the number of associated RNA elements and the part of the RNA element associated with each intron is then explicitly listed.

```
E0 383506..383386 0,1
 E:BG192901 1..121 (383506..383386)
E1 375227..374983 0,2
 E:AW938686 31..213 (375167..374983)
 E:BG192901 122..366 (375227..374983)
E2 373531..373393 0,2
 E:AW938686 214..352 (373531..373393)
 E:BG192901 367..505 (373531..373393)
E3 362874..361968 0,2 I3
 E:BG192359 680..11 (362635..361968)
 E:BG192901 506..774 (362874..362602)
E4 355204..355078 0,1
 E:AW938686 353..479 (355204..355078)
E5 350898..350705 0,1 I4
 E:BI829756 1..194 (350898..350705)
E6 347956..347746 0,2
 E:AW938686 480..676 (347956..347762)
 E:BI829756 195..405 (347956..347746)
E7 336319..335999 0,2
 E:BF329257 1..259 (336277..336020)
 E:BI829756 406..724 (336319..335999)
I0 383385..375228 1
 E:BG192901 121..122 GT/AG -10
I1 374982..373532 2
 E:AW938686 213..214 GT/AG -10
 E:BG192901 366..367 GT/AG -10
I2 373392..362875 1
 E:BG192901 505..506 GT/AG -10
I3 373392..355205 1
 E:AW938686 352..353 GT/AG -10
I4 355077..347957 1
 E:AW938686 479..480 GT/AG -10
I5 350704..347957 1
 E:BI829756 194..195 GT/AG -10
I6 347745..336320 1
 E:BI829756 405..406 GT/AG -10
```

Fig. 3 : Detailed list of introns and exons of the graph output `NT_026281_10`. See text for explanation.

3. Graph drawing.

3.1. **Graphs** (*Cormen et al*).

Graphs are data structure where an ensemble of nodes (vertices) are connected each other by an ensemble of edges. The importance of graphs are increasing in many various domains in computer science and its applications. There is a lot of computer problems how are defined by graphs (software engineering, database systems, knowledge representation, telecommunication, www, …).

The nomenclature on graphs are quite extensive. We give here some specific denominations that willl come soon in this work (see Fig. 1). Graphs could be directed or undirected. If they are directed one can go through an edge only in one given direction. The edges are represented by arrows and such a graph is called digraph. If they are undirected one can go through an edge in both directions. An undirected graph is connected or connexe if any couple of nodes are linked together by an edge. A directed graph is fully connected or fully connexe if each vertex is accessible from any other.

There are two principal ways to scan information in a graph : the Breadth-First Search (BFS) and the Depth-First Search (DFS). The complete explanation of these algorithms is beyond the scope of this work but just keep in mind that the second method is not only a "walk" in a graph but also a source of information on graph's structure like classification of edges (tree-edges, cross, forward and back edges). If the graph is directed and acyclic, one can perform a topological sorting (see Dot algorithm).

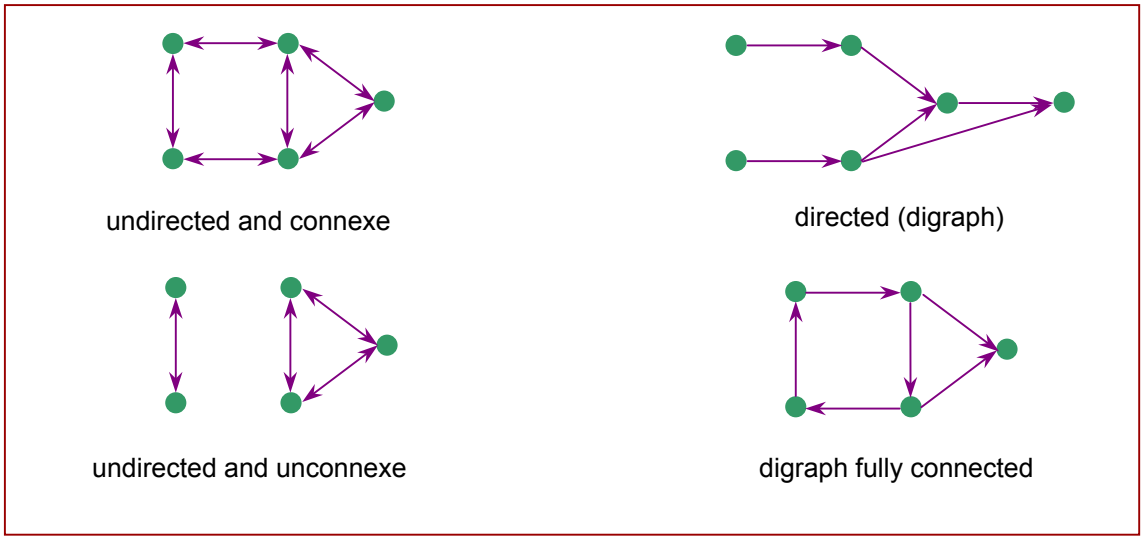


Fig. 4 : Small nomenclature on graphs.

3.2. **Graph drawing** (*Cruz and Tamassia*)**.**

Graph drawing is not a trivial task, as well by hand as by computer. People are working on this issue since years. The difficulty lies in the fact that it is not easy to transform an abstract data structure in a visual, bidimensional and comprehensible manner. The basic constraints in graph drawing are given by the data themself, these can be qualified as semantic criteria. For gene representation, digraphs seem to be suitable because of the linearity of the genome. Moreover, aesthetic criteria are the cornerstone of graph layout. For digraph and used in Dot algorithm (Gansner et al.), these principles could be the following :

- Expose hierarchical structure in the graph. In particular, aim edges in the same general direction if possible. This aids finding directed paths and highlights source and sink nodes.
- Avoid visual anomalies that do not convey information about the underlying graph (edge crossings, sharp bends).
- Keep edges short.
- Favor symmetry and balance.
- ...

Most of them are in polynomial time, NP-hard or NP-complete (Di Battista et al.). In fact, to solve such problems in a moderate time, we should have recourse to well-thought-out heuristics. In general, there is no way to optimize these aesthetics simultaneously and it is always a question of trade-offs between semantics and aesthetics as well as between aesthetics themself.

3.3. **Dot algorithm** (*Gansner and North*)**.**

This algorithm was developed in AT&T Bell Laboratories and the code is in open source. The input is a connected digraph and some attributes like minimum horizontal and vertical separation between nodes. The algorithm assigns each node to a rectangle in the plane with a center point and assigns each edge to a sequence of B-spline control points. A B-spline curve is an approximating curve which is defined by points (start – stop) and a gradient. It is like Bezier curves with a local control. The layout is guided by the aesthetic constraints mentioned above. The graph drawing algorithm has four passes.

The first pass places the nodes in discrete ranks. The second sets the order of nodes within ranks to avoid edge crossings. The third sets the actual layout coordinates of nodes. The final pass finds the spline control points for edges.

3.3.1. Optimal rank assignment.

To have a consistent rank assignment, a graph must be acyclic. A preprocessing step detects cycles and breaks them by reversing certain edges. At this point, a Depth-first search partitions edges into two sets : tree edges and non-tree edges. The tree defines a partial order on nodes. The non-tree edges are partitioned into three sets : cross edges (connect unrelated nodes in the partial order), forward edges (connect a node to some of its descendants) and back edges (connect a descendant to some of its ancestors). The presence of cross and forward edges does not create cycles. Actually, all cycles are broken by reversing back edges into forward ones. An aesthetic principle prescribes making short edges. It is desirable to find a node ranking for which the sum of all the weighted[1] edge lengths is minimal. Moreover, a spanning tree is feasible if it induces a feasible ranking. Each tree edge of a given feasible spanning tree recieve an integer cut value. A deleted tree edge divide a tree into a tail and a head components. The cut value is the sum of the weights of all edges from the tail component to the head component, including the tree edge, minus the sum of the weights of all edges from the head component to the tail component. Tree edges with negative cut value are replaced by appropriate non-tree edges. The process is stopped when all tree edges have non-negative cut values and the resulting spanning tree corresponds to an optimal ranking. This refers to the implementation of the so called network simplex algorithm.

[1] The weight signifies the edge's importance, which translates to keeping the edge short and veretical, usually 1.

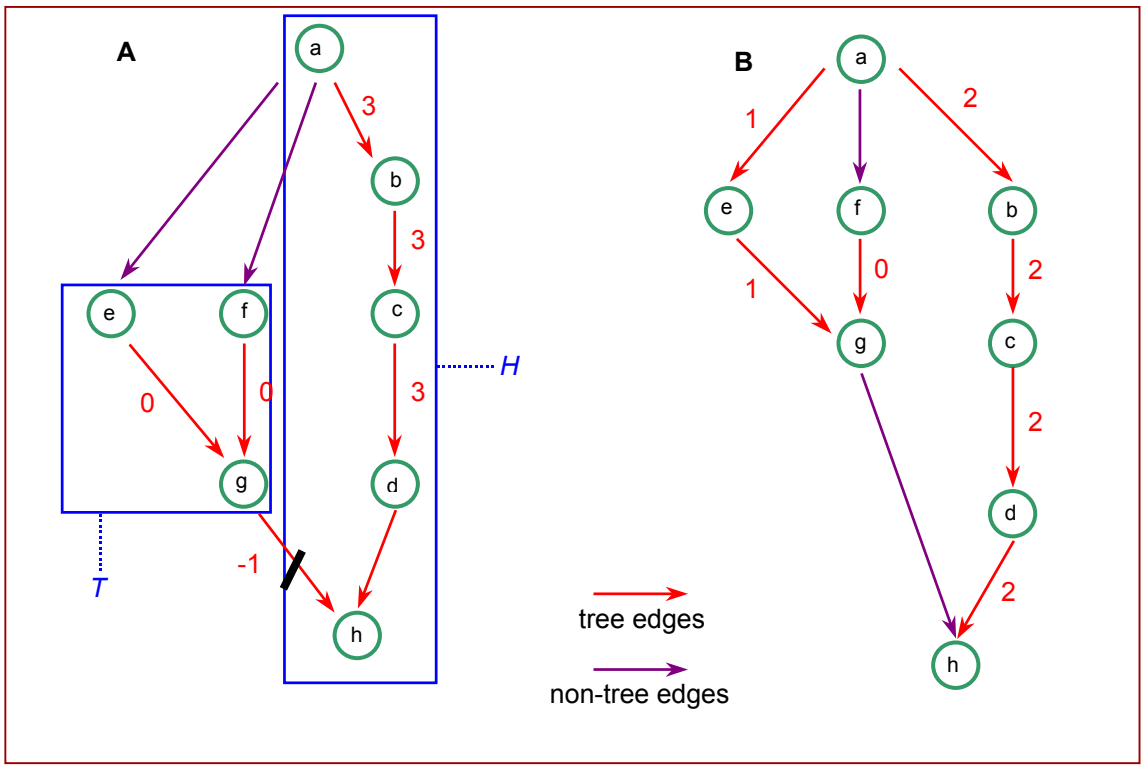


Fig. 5 : Optimal ranking. At the begining, all edges have weight 1. In A, the graph is shown after initial ranking, with cut values as indicated. The black bar represents a break. For instance, the cut value of edge (g,h) is –1 corresponding to the weight of edge (g,h) (1 edge from the tail component *T* to the head component *H*) minus the weights of edges(a,e) and (a,f) (2 edges from the head component *H* to the tail component *T*). In B, the edge (g,h) with a negative cut value has been replaced by the non-tree edge (a,e), with the new cut values shown. Because they are all non-negative, the solution is optimal and the algorithm terminates.

3.3.2. Vertex ordering within ranks.

At this point, edges between nodes more than one rank apart are replaced by edges of one unit length and "virtual nodes". The original graph is converted in one whose edges connect only nodes on adjacent ranks. The vertex ordering between rank determines the edge crossings. A good ordering minimizes crossings. This minimization is NP-complete, even for only two ranks and can be bypass by

using heuristics. Generally, an initial ordering within each rank is computed. Then a sequence of iterations is performed to try to improve the orderings. Each iteration traverses from the first rank to the last one, or vice versa. When visiting a rank, each of its vertices is assigned a weight based on the relative positions of its incident vertices on the preceding rank. Then the vertices in the rank are re-ordered by sorting on these weights. The algorithm is strongly influenced by the vertex weighting method used (barycenter function or median function). In DOT case, the used of the median function with small arrangements is preferred.

3.3.3. Node Coordinates.

X and Y coordinates are computed in two separate steps. The first step assigns X coordinates to all nodes, including virtual ones. The second step, assigns Y coordinates. This step is dependant of the minimum separation between nodes, parameter defined by the user. Because the Y coordinate step is straightforward, the challenge resides in X coordinates. The "traditional" method refers to a integer optimization problem that is dependant of node neighborhood and aesthetic principles as well as the nature of nodes (real or virtual). Unfortunately, graphs of more than few dozen nodes are drawn in to much time and, again, they should have recourse to heuristics. This heuristic finds a "good" initial placement, then tries to improve it by sweeping up and down the ranks similar to the vertex ordering algorithm described previously and called network simplex algorithm

3.3.4. Drawing edges.

Edges are drawn as spline curves. Although splines are more difficult to program, they yield better drawing and help to avoid visual anomalies. The spline routing algorithm is divided into a top half and a bottom half. The top half computes a polygonal region of the layout where the spline may be drawn. It calls the bottom half to compute the best spline within the region. As a final step, the top half resizes virtual nodes according to the bounding box of the spline, and splines and clips the spline to the boundaries of the endpoint node shapes.

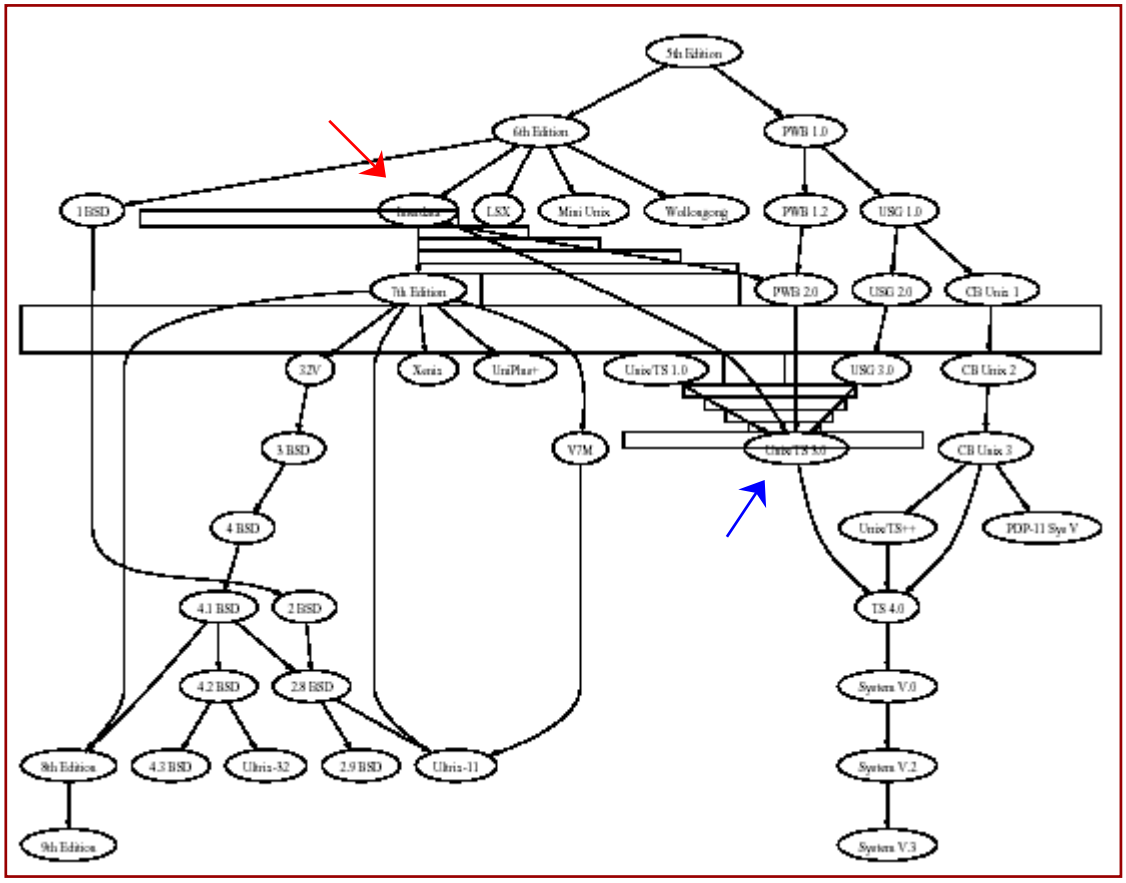


Fig. 6 : Region and its spline. The associated edges is from "Interdata" to "Unix/TS 3.0".

There are three kinds of edges in the drawing (edges between nodes on different ranks, flat edges between different nodes on the same rank, and self-edges or loops) and each edge requires a different drawing treatement. Finally, the last object to be managed is edge labels. They are, for inter-rank edges only, represented as off-center virtual nodes. Edge labels on self edges are easy to handle, but flat edges are more complicated.

We describe briefly the Dot algorithm. For more information, one can have a detailed and patient look at *Gansner and North*. In our opinion, what is the most interesting in this concept is the DOT language.

3.4. **Dot language** (*Koutsofios and North*).

The DOT language describe three kinds of objects : graphs, nodes and edges. Each object accept some attributes. For example, with graph attributes, one can manage the background color, the font family, the size and the orientation of the graph. With nodes attributes, one can handel the shape, the label or associate an URL with node. Finally, with edge attributes, one can manage the label, the style... . The best way to understand such a language is to take an example and to dissect it conscientiously.

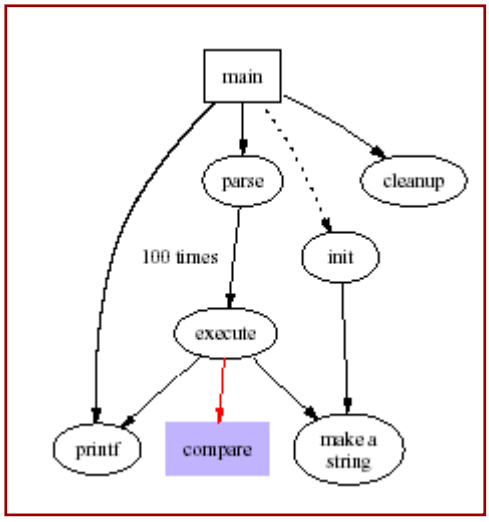


Fig. 7 : Drawing of example graph.

```
1:  digraph G {
2:    size = "4,4" ;
3:    main [ shape = box ] ;          /* this is a comment */
4:    main -> parse [ weight = 8 ] ;
5:    parse -> execute ;
6:    main -> init [ style = dotted ] ;
7:    main -> cleanup ;
8:    execute -> { make_string ; printf } ;
9:    init -> make_string ;
10:   edge [ color = red ];          // so is this
11:   main -> printf [ style = bold , label = "100 times" ] ;
12:   make_string [ label = "make a\nstring" ] ;
13:   node [ shape = box , style = filled , color = ".7 .3 1.0"] ;
14:   execute -> compare ;
15: }
```

Fig. 8 : The DOT code creating the example graph. (Color code = node name , attributes).

Line 1 : Give the graph name (*G*) and type (*digraph* for directed graph).

Line 2 : *Size* is a graph attribute that give the maximum drawing size in inches.

Line 3 : Node or edge attributes are set off in square brackets. The node *main* is assigned *shape boxe*. Look at one option to insert comments !

Line 4 : The node *main* is linked with the node *parse*. The arrow (->) indicates that we are in a directed graph. The edge is straightened by increasing its *weight* (the default is 1).

Line 5 : The node *parse* is linked with the node *execute*.

Line 6 : The node *main* is linked with the node *init*. The edge is drawn as a dotted line.

Line 7 : The node *main* is linked with the node *cleanup*.

Line 8 : The node *execute* is linked to the node *printf* and *make_string*.

Line 9 : The node *init* is linked to the node *make_string*.

Line 10 : The default edge *color* is set to *red*, this affects any edges created after this point in the file. Look at another option for comments.

Line 11 : The node *main* is linked to the node *printf*. The resulting edge is *bold* and labeled "*100 times*".

Line 12 : The node *make_string* is given a multi-line label.

Line 13 : The default node is a box filled with a shade of blue. A color value can bee a hue-saturation-brightness triple (this case), one of the colors names listed in *Koutsofios and North*, or a RGB triple.

Line 14 : The node *execute* is linked to the node *compare*. Note that only the node *compare* has attributes mentioned in line 13.

Line 15 : end of the graph.

Each line ends with a semicolon. The graph description starts with an opening curly bracket and ends with a closing one. This example gives only a synopsis of the DOT language. For more information, just have a careful look at Koutsofios and North.

Now, to have a graphic representation, DOT should be run such a text file as argument. Commands, on a command line, could have the following forms (-o specifies where the output should be redirected) :

```
$    dot -Tps graph1.dot -o graph1.ps        to create a PostScript output.
$    dot -Tpng graph1.dot -o graph1.png      to create a PNG output.
$    dot -Tjpg graph1.dot -o graph1.jpg      to create a jpg output.
$    dot -Tismap graph1.dot -o graph1.ismap  to create an output with map coordinates.
```

4. Tromer2Map concept[2].

4.1. Overview.

The concept lies in an intention to give a graphical web interface that allows to visualize a graph produced by the Tromer program. The input is mainly a graph file as describe above (chapter 2.3.) and a text file called "library" where all formatting information are given. The main output is a clickable picture representing the connection between nodes. The aim of the concept is to link introns and exons with different data sources like, in a first step, associated RNA elements or, in the future, data from a transcriptome database. This concept is also developed to accept easily further user's wishes.

4.2. Trome2dot.

This first PERL script grinds up data given by graph files and libraries to produce a text file in DOT language, file that can be used as argument for the DOT program. This script receive three arguments : a tromer graph file, a format file and an output file name (remark that all are text files).

The starting point is a partial parsing of the graph file. In a hash of hash, the script combines names of edges (introns-exons) with their respective start and stop nodes as well as the number of associated RNA elements. Then, the format file is processed and the data are divided in normals and specials. The normal data, which means all the attributes for graph, nodes or edges, are stored in a hash. The special ones, if exist, are also stored in a hash and will be used, for example, to highlight a single path in the graph. The number of associated RNA elements are used to assign each edge a colour gradation. More an edge is dark, more RNA elements prove its existence.

Tome2dot can produce two different text files, or DOT files, depending on the library used as argument. If the library is called "headerlib.txt", the script gives a base for a PNG picture used in the header of the result (see chapter 5.2.), otherwise it returns a base for the detailed part (see chapter 5.3.). This script is mainly called by another script named Tromap2.pl.

[2] All scripts can be download at http://ludwig-sun2.unil.ch/~mcrausaz/stage/script

4.3. Tromap2.pl.

Tromap2.pl is a PERL script based on the CGI technique. It gets data from a web submission form and generates a web page on the fly. It is calling the above Trome2dot script as well as the DOT program and itself. The resulting web page is composed of frames.

A way to produced frames on the fly is to call the same script several times, depending on number of frames, in changing a variable named "path_info". Because of the use of the process ID to named generated files, it is really important and vital that each call receive also this number. If path_info is empty, the script first checks data given through the form (see chapter 5.1.). If something is wrong Tromap2.pl returns an error warning and stops the process. Otherwise, it creates the frameset and calls itself two times.

If path_info is given as "tromhead_PID", the script generates the header part of the resulting web page. This process fills the first part of the framset in creating the first PNG picture. If path_info is given as "tromap_PID", Tromap2.pl produces the central part of the frameset composed of the clickable image map. Each recurrent call is stopped when every pictures are created. The third part of the framset is represented by a caption for the image map. This web page is static. Tromap2.pl is the script that make the link between user's query and result. it leads to a interactive web page with an image map that can be clicked to produced other information.

4.4. Info and Fetch_web.

Both are complementary scripts that are also using the CGI technique. Info is called by a click on the image map, exactly on an intron or an exon, and recovers the graph file name and the fragment name. Then, it creates a simple web page with information about contig, exon or intron and a list of associated RNA elements. All these elements are clickable and, via the script Fetch_web, it is possible to find the desired entry in raw text mode.

4.5. Libraries.

The use of such "formatting" libraries allows user to specify own wishes. The file is a text that can be easily changed to highlight different aspects of a graph. In fact, and with low implementation costs, it is possible to

increase the information content of a graph. For example, if one want to emphasize a path in the graph, one uses the "special" part of the library.

After the name of the file (line starting with ">"), a list of attributes is given. The first character defines graph (g), node (n) or edge (e) attributes (for more information, see figure 10). The special zone mentions edges with different color. In our case, the label of edge E8 should be red instead of black.

```
>default lib                    library name
g_bgcolor = white               graph background color
g_nodesep = 0.4                 node separation
g_rankdir = LR                  landscape orientation
n_fontname = Arial              node font name
n_fontsize = 12                 node font size
n_style = filled                node shape style
n_A = dodgerblue                acceptor color
n_D = dodgerblue                donor color
n_H = forestgreen               start color
n_T = crimson                   stop color
e_fontname = Arial              edge font name
e_fontsize = 16                 edge font name
e_style = bold                  edge style
e_arrowsize = 1.0               size of the arrowa
e_E = black                     exon label color
e_I = grey                      intron label color
special
E8 = red
I8 = red
E21 = green
I14 = blue
E23 = green
```

Fig. 9 : Example of a library used to format graph image map. For more information on the value of attributes refer to the Dot user's guide (*Koutsofios and North*).

The special field can be used to connect the Trome2Map form with a transcriptome database. Suppose that one want to see if a path of a graph is more involved in cancer cell line than in normal cell. With a simple query on a well-done database, the script with help of the library, can emphasize each path. Moreover, if one want to see difference in expression depending on organs, a color code can be implemented and labels of path can be highlighted.

It is also possible to play around with "normal" attributes. For example, it is possible to create several libraries that users can choose in the starting form. These libraries can be designed to allow one to get a black-and-white printable version or to give more importance to others aspects in a graph.

5. Web interface.

In this chapter, we analyze scripts output. Each part of the frameset returns by Tromap2.pl as well as the complementary script output is observed and explained.

5.1. Form.

The web submission form is classic. One can give a graph accession number, copy-paste a Tromer output file in text mode or upload a file with a *.graph* extension. The submission button sends the http query to the Tromap2.pl script.

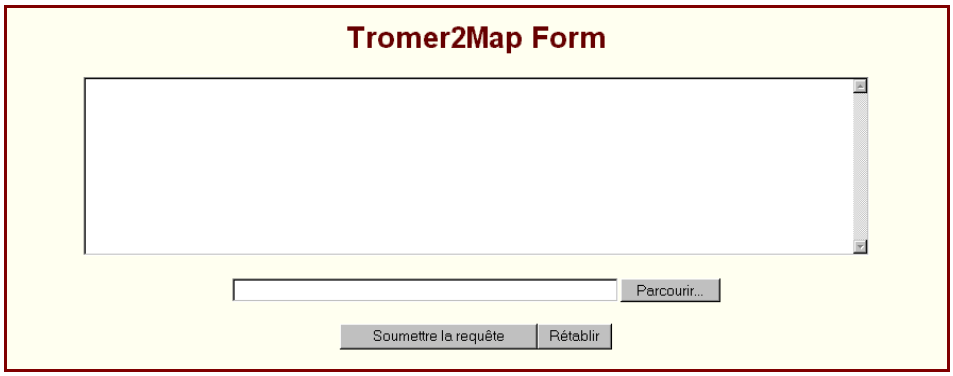


Fig. 10 : Web submission form.

5.2. Header.

As we will see in chapter 5.4., the clickable graph can be very big. To make visualization easier, the option was taken to create a small header picture. So users can quickly have an overview on the complexity and the global arrangement of the gene. Moreover, this header being always on screen, navigation in the map is more flowing.

The left part of this frame is used to give information about the gene. Briefly, one can find the graph accession number, the contig accession number and the chromosome number which are link to the NCBI database, the start and stop positions on the contig, the number of introns and exons and the total size of the gene.

In graph itself, transcript starts are green and stops are red like introns. Exons as well as donors and acceptors are colored in blue. All the header is optimized to be best viewed on a 1280 x 1024 pixels screen. This resolution allows to show a fair picture as well as some data about it.

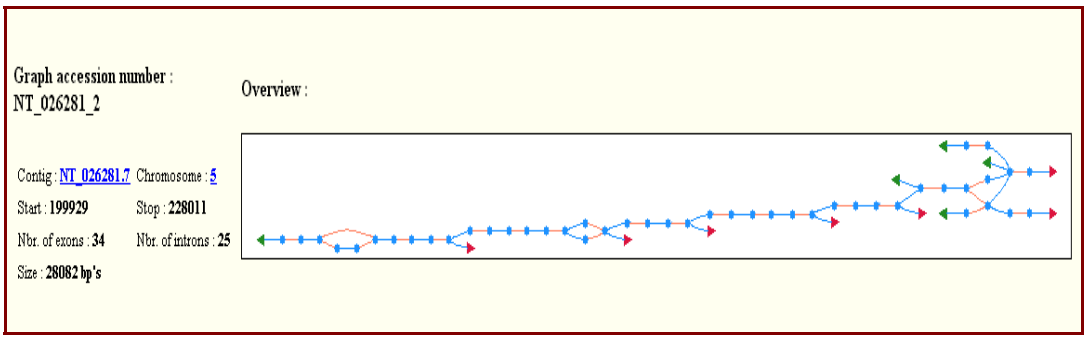


Fig. 11 : Header of the resulting frameset. The represented graph is the human threonyl-tRNA synthetase gene (TARS, SwissProt Nbr. P26639).

5.3. Caption.

This caption or legend is, actually, a static web page. It show a grey gradation link to the number of associated RNA elements per introns/exons. This gradation is based on a logarithmic progression with nine different classes. Moreover, it give a color code for nodes. It is also possible to customize such a frame.

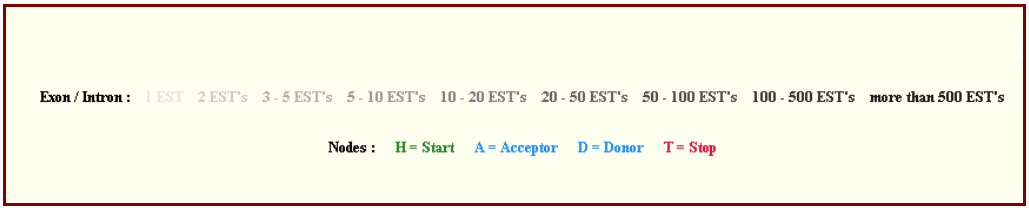


Fig. 12 : Caption (or legend) gives for each graph, it is the frameset footer.

5.4. Total frameset.

The total frameset, as in figure 13, is composed of the header, the caption and the map itself. The last one is an image map representing a graph with all edges clickable. Because of its size, the use of a scrolling bar is necessary to display a visually comfortable and well-explorable picture.

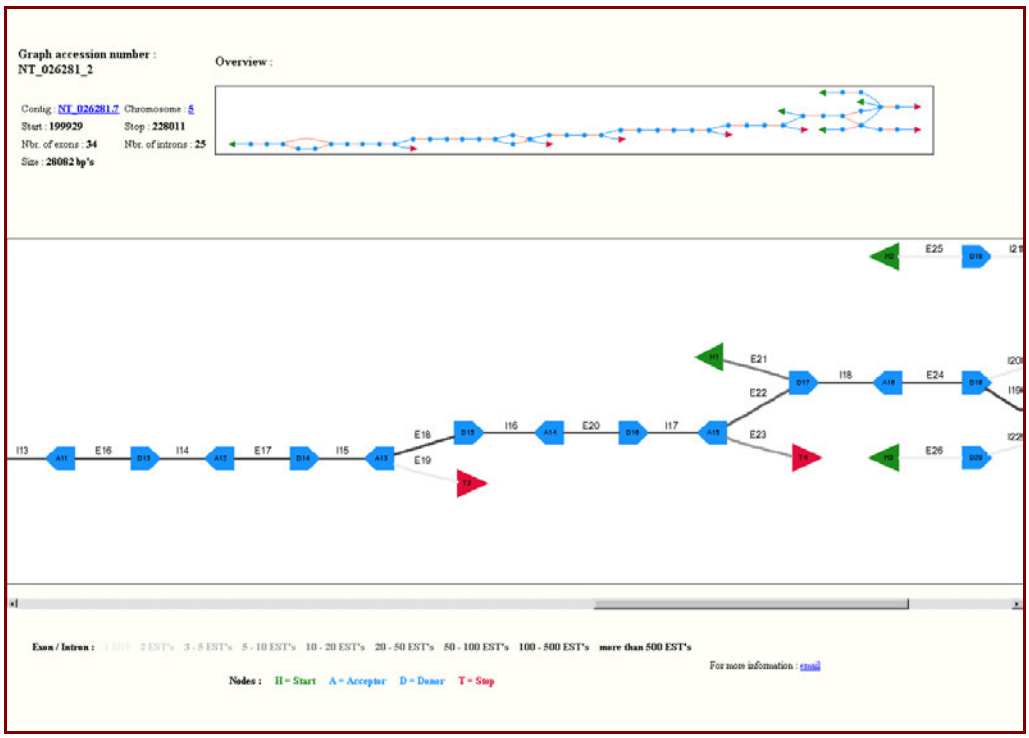
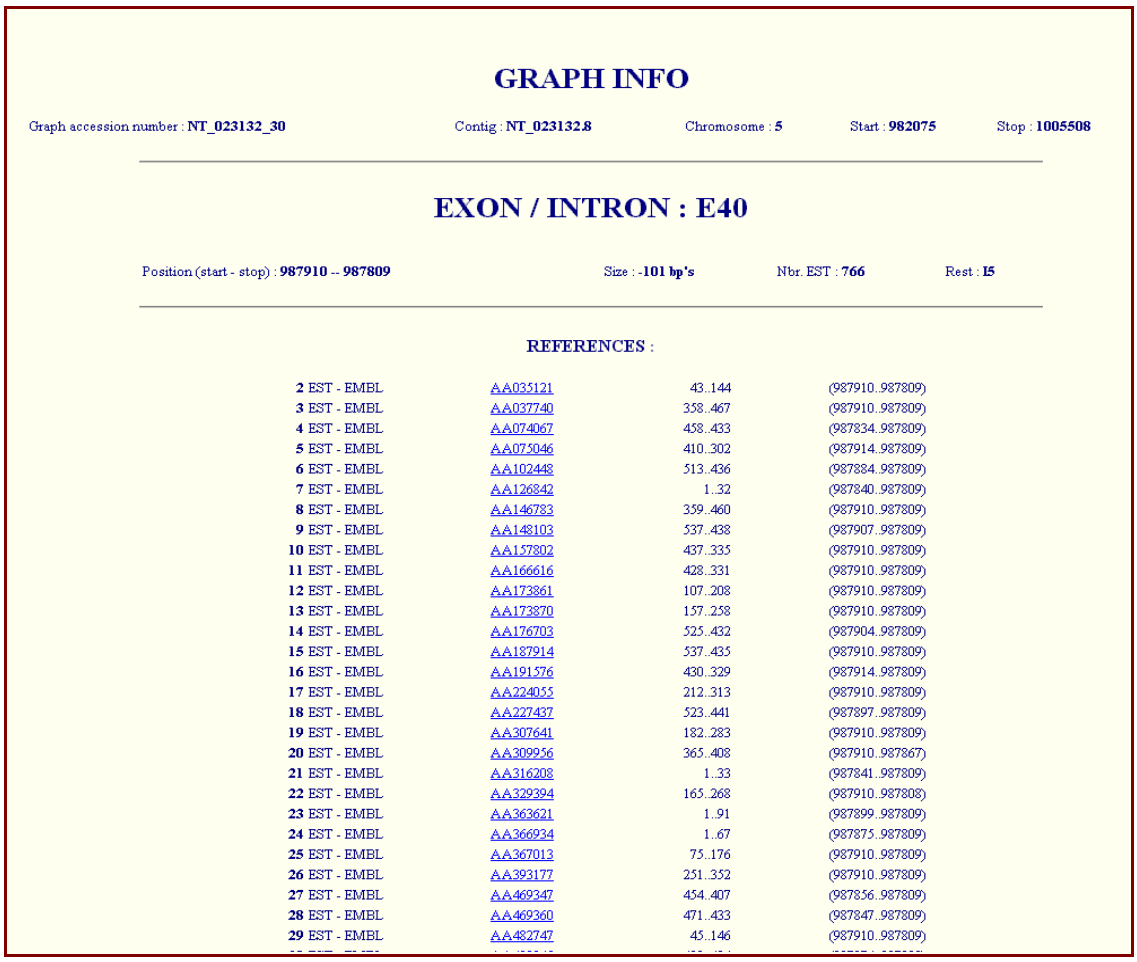


Fig. 13 : Complete result of the threonyl-tRNA synthase gene (TARS). Only a part of the map is visible.

5.5. Other information.

The fact to click on an edge in the image map calls the Info script with graph file and exon/intron name as arguments. This script will display a picture like figure 14, where one can find information about graph and the chosen intron/exon as well as a clickable list of associated RNA elements.

The graph data are more or less the same as the ones on the header part before. Intron/exon data are made up of the name (E or I followed by a number), the start and stop positions on the contig, the size in bps, the number of EST or associated RNA elements and a section called "rest" where one can find, for exons only, a list of conflicting introns. If one clicks on a link under References, the script Fetch_web finds the corresponding entry in raw text mode.



GRAPH INFO

Graph accession number : **NT_023132_30** Contig : **NT_023132.8** Chromosome : **5** Start : **982075** Stop : **1005508**

EXON / INTRON : E40

Position (start - stop) : **987910 -- 987809** Size : **-101 bp's** Nbr. EST : **766** Rest : **I5**

REFERENCES :

2 EST - EMBL	AA035121	43..144	(987910..987809)
3 EST - EMBL	AA037740	358..467	(987910..987809)
4 EST - EMBL	AA074067	458..433	(987834..987809)
5 EST - EMBL	AA075046	410..302	(987914..987809)
6 EST - EMBL	AA102448	513..436	(987884..987809)
7 EST - EMBL	AA126842	1..32	(987840..987809)
8 EST - EMBL	AA146783	359..460	(987910..987809)
9 EST - EMBL	AA148103	537..438	(987907..987809)
10 EST - EMBL	AA157802	437..335	(987910..987809)
11 EST - EMBL	AA166616	428..331	(987910..987809)
12 EST - EMBL	AA173861	107..208	(987910..987809)
13 EST - EMBL	AA173870	157..258	(987910..987809)
14 EST - EMBL	AA176703	525..432	(987904..987809)
15 EST - EMBL	AA187914	537..435	(987910..987809)
16 EST - EMBL	AA191576	430..329	(987914..987809)
17 EST - EMBL	AA224055	212..313	(987910..987809)
18 EST - EMBL	AA227437	523..441	(987897..987809)
19 EST - EMBL	AA307641	182..283	(987910..987809)
20 EST - EMBL	AA309956	365..408	(987910..987867)
21 EST - EMBL	AA316208	1..33	(987841..987809)
22 EST - EMBL	AA329394	165..268	(987910..987808)
23 EST - EMBL	AA363621	1..91	(987899..987809)
24 EST - EMBL	AA366934	1..67	(987875..987809)
25 EST - EMBL	AA367013	75..176	(987910..987809)
26 EST - EMBL	AA393177	251..352	(987910..987809)
27 EST - EMBL	AA469347	454..407	(987856..987809)
28 EST - EMBL	AA469360	471..433	(987847..987809)
29 EST - EMBL	AA482747	45..146	(987910..987809)

Fig. 14 : Example of the information recieved after clicking on an intron/exon in an image map.

By adding special information in the default library, it is possible to highlight a path across the graph. Figure 16 shows such a way while figure 15 show the added part. Only labels are highlighted what allows to keep informations about associated RNA elements.

```
special
E8 = red
I8 = red
E21 = red
I14 = red
E23 = red
```

Fig. 15 : Added part in the library that will highlight a desired path in the graph.

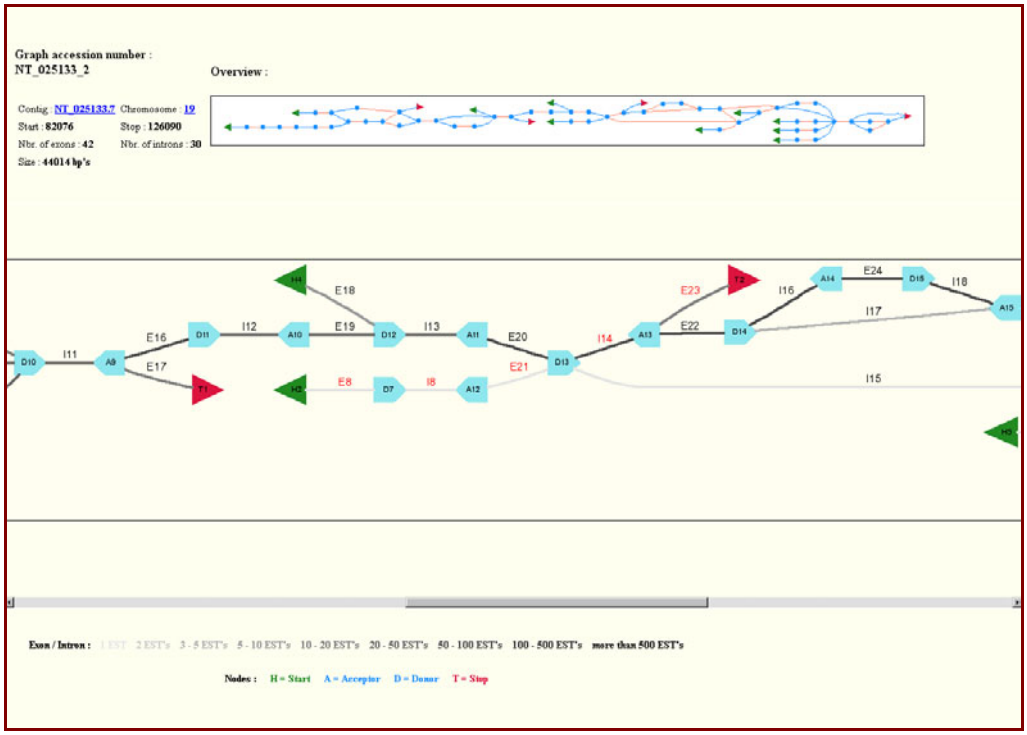


Fig. 16 : Highlighting a special path. This graph represents the heterogeneous nuclear ribonucleoprotein M gene (HNRPM, SwissProt Nbr. P52272) with a arbitrary path highlighted.

6. Conclusions.

In this report, we suggest a graphic representation of Tromer graph outputs. The concept is based on the DOT algorithm which allows to compose graphs from a text file with a straightforward language. The result is given under a web interface constituted of three frames. The header is made with an overview and some data about contig, gene and graph. The central one is a clickable map. The footer give a caption explaining the image map. Queries are submit through the web by a simple submission form. Each script use the CGI technique to manage data exchange.

The Trome2Map concept give a user-friendly graph representation while the text mode is more computer-friendly. It is also easy to use. One just have to give a graph accession number and, after a relative short time, depending on graph size, a comprehensive graphic representation is shown on the screen. The three frames are a good manner to dispense more information. Because one of it can be scrolled, the map size is increased what improve visibility and comprehensibility. Representing edges with colour gradations allows to emphasize the most represented transcript in vivo, at least the most sequenced by EST technology.

This concept was thought to allow representation of the higher number of graphs. One should be aware that genes or graphs have different level of complexity. Some have only one exon and others have much more. For example, the titin gene is made of more than 700 introns/exons shared between few transcripts. Note that Tromer2Map can produced such a map but web browsers can not edit it.

The Tromer2Map process is open and evolutive. Without high development costs and working with libraries, it is possible to express more biological data in an image map. It will be easy to make a link with a transcriptome database through mySQL queries. A PERL script can manage such a problem in reading user wishes, making queries on database and returning data to a Tromer2Map library.

The principal weak point of such graphic representations is the lack of connection with genomic DNA. The idea is to position exons on its relative genomic contig which amounts to aligning exons together. The difficulty lies in the fact that size of exons and introns are sometimes very different. Exons alignment should be at one basepair level while truncated introns should be at 100 to 1000 basepaires level. A possible answer could be the use of subgraph. A pre-processing step sorts overlapping and non-overlapping regions. The overlapping parts are then aligned and a subgraph is made. Finally, all subgraphs are linked together by introns of same length. The precise positioning of 3' tags as well as a 5'$\rightarrow$3' orientation of the gene encounter same technical problems as above. This developement is not trivial. According to principles of universality and readability, it becomes hard to imagine what could give a graph like the one of figure 17 that have a high level of complexity.

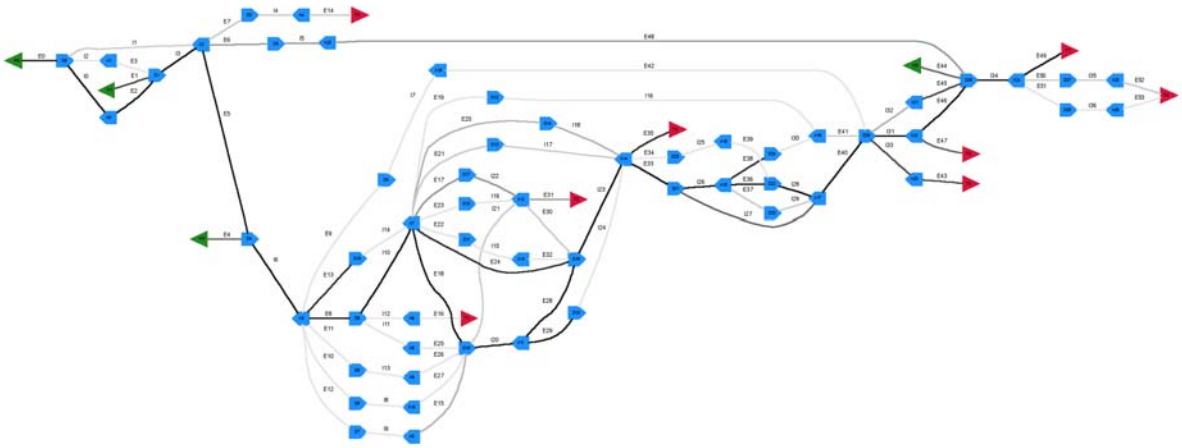


Fig. 17 : Graph of the nucleophosmin gene (nuclear phosphoprotein B23 or numatrin, SwissProt Nbr. P06748).

Other further developments can be envisaged. For example and briefly, it will be useful to work on default libraries to give to users different formatting possibilities like emphasize exons or minimize nodes. In header, it will be helpful to think about a cursor which shows where we are in the image map. Finally, why not trying to transform pixel picture in vectorial mode to allow zoom in and out ?

The concept presents in this report give a nice, user-friendly interface to the graph output of the Tromer program. It is a young web interface which needs to be tuned. The major aspect of this report is the use of a simple, rapid and free graph drawing program which can be easily adapted to other applications.

7. Acknowledgements.

I will like to thank warmly people for the SIB in Epalinges, particularly, in alphabetical order, Brian, Christian, Laurent, Marco, Victor, Viviane and Volker, for professional and technical supports.

I also thank my coreligionist (DEA students 2001 – 2002, Marc, Vassilios, Yvan) for freindly but also technical supports.

Finally, I thank affectionately my nearest and dearest (Roselyne, Salomé and Bastien) for psychological, sentimental, temporal ... supports.

8. References.

Camargo A.A. et al. : The contribution of 700,000 ORF sequence tags to the definition of the human transcriptome. *PNAS* **98**(21) 12103-12108 (2001).

Cormen T., Leiserson C. and Rivest R. : Introduction to Algorithms, The MIT Press, Cambridge, Massachusets (1990).

Cruz F. and Tamassia R. : Graph drawing tutorial, www.cs.brown.edu/people/rt/gd-tutorial.html.

Di Battista G., Eades P., Tamassia R. and Tollis I.G. : Algorithms for drawing graphs : an annotated bibliography, www.cs.brown.edu/people/rt/papers/gdbiblio.pdf, June 1994.

Gansner E.R., Koutsofios E., North S.C. and Vo K.-P. : A technique for drawing directed graphs, *IEEE Trans. Software Engineering*, **19**(3), 214-230 (1993).

Gansner E.R. and North S.C. : An open graph visualization system and its applications to software engineering, *Softw. Pract. Exper.*, **00**(S1). 1-5 (1999).

Iseli C., Stevenson B.J., et al. : Long-Range Heterogeneity at the 3' ends of human mRNAs; *Genome Research* **12** 1068-1074 (2002).

Iseli C : Transcriptome database roadmap, "*internal publication"* march 28, 2002.

Koutsofios E. and North S.C. : Drawing graphs with dot, February 4, 2002, www.research.att.com/sw/tools/graphviz/dotguide.pdf.

Strausberg R.L. and Riggins G.J. : Navigating the human transcriptome, *PNAS* **98**(21) 11837-11838 (2001).