A Tailorable and Extensible Automatic Layout Facility,

Massimiliano Beccaria*, Paola Bertolazzi*, Giuseppe Di Battista#, and Giuseppe Liotta#

# Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
via Salaria 113 00198 Roma Italia

* IASI - CNR,
viale Manzoni, 30 00185 Roma Italia

Abstract

An automatic layout facility is a tool that receives as input a graph-like structure and is able to produce a diagram that nicely represents such a structure. Due to the increasing number of systems that manage diagrams, the automatic layout facilities and the algorithms for graphs layout have been extensively studied in the last years. We present a new approach in building an automatic layout facility. Our approach is based on a modular management of a collection of algorithms and on a tool that is able to "synthesize" the best algorithm for a given application. Such approach has been used for devising the automatic layout facility of Diagram Server, a network server that offers to its clients an effective set of facilities for managing diagrams.

1. Introduction

Diagrams are effective communication tools for people in several fields. Examples include computer aided software engineering, information systems analysis and design, and project planning. As a consequence, an increasing number of systems use diagrams as an interaction language with the users. A very limited list of examples include Mast-ER [Ma87], IEW [IE89] and GELO [DuMeRe89] in computer-aided software engineering, G+ [CMWoWo88] and Qbio* [AnCaSa9O] as database query systems, Tango [St88] and Brown’s works [Br87] on algorithm animation, X-PERT [DiPaTaTo89] to support project management.

An increasing number of systems that manage diagrams are provided with an automatic layout facility [EsTa89, TaBb88]. An automatic layout facility is a tool that receives as input a graph-like structure and is able to produce a diagram that nicely represents such a structure (users may wish to avoid edge crossings, illustrate symmetries, emphasize logical separations, etc.).

An effective automatic layout facility should be able to:
- accept a set of requirements that have to be satisfied in the diagram;
- accept a priority level between requirements;
- automatically propose the layout algorithm or algorithms that can produce diagrams satisfying such requirements (the user doesn’t need to know details about layout algorithms).

Several automatic layout tools are proposed in the literature that attempt at solving the above problems. A very limited set of such tools includes SPREMB [Ea87], TYGES [HyEa85], GRAB [KoDaMeSpTu87] GraphView [BiSh89], and EDGE [Ne88].

However, in our opinion, existing tools and algorithms are not enough flexible and extensible to solve the aforementioned problems. To give some examples: GRAB has an effective behaviour for producing diagrams with a monotonic fashion (so its main target is to represent directed acyclic graphs) but it is not completely satisfactory for representing flat structures; SPREMB attempts to solve the problem of using in an integrated framework several existing layout algorithms (the result is a very flexible paradigm), in any case, the choice of the algorithm to apply is under the responsibility of the user; moreover, it is not completely clear how to use SPREMB for representing complex layout algorithms like Giotto [BaNaTa86].

We present a new approach in building an automatic layout facility. Our approach is based on a modular management of a collection of algorithms and on a tool that is able to "synthesize" the best algorithm for a given application. Such approach has been used for devising the automatic layout facility (ALF) of Diagram Server (DS), a network server that offers to its clients an effective set of facilities for managing diagrams [DiGiSaTa90].

The paper is organized as follows: in Section 2 we show the architecture of ALF; in Section 3 we present a new approach to the problem of classifying layout algorithms from disparate areas; in Section 4 we define the concept of diagram model, a concept that represents the requirements of the diagrams of a certain class of applications and we describe a tool that manages a collection of diagram models; in Section 5 we describe a tool that is able to select the algorithms that fit the given requirements.

2. General description of ALF.

ALF has been conceived to take into account several classes of requirements from different classes of applications. It manages a collection of algorithms; when a graph-like structure has to be represented with a diagram, it selects and applies the best suitable algorithm for that particular application.

The realization of ALF required a solution to two main problems:
(1) a classification problem: ALF has to manage a collection of layout algorithms, where both theoretical and application oriented algorithms are stored; due to this fact it has to be provided with a taxonomy of the existing layout algorithms; that taxonomy should be easily extensible to classify new algorithms, proposed in this field;
(2) a selection problem: ALF has to be able to select, among the available algorithms the best suited for a certain application.
Problem (1) required the analysis of a large number of automatic layout algorithms. This work led to the following observations:

a. each automatic layout algorithm that has been presented in the literature can be decomposed into a sequence of some main functional steps; usually, each step is built to take into account certain layout requirements; for instance, in figure (1) the algorithm by Sugiyama et al. [SuTaTo81] modified by Rowe et al. [RoDaMeMeSpTu87] is decomposed into five steps. The first step receives as input a digraph and produces as output an acyclic digraph by reversing one edge for each cycle; in the second step the algorithm builds a layered network: where nodes are assigned to horizontal levels; in the third step the algorithm builds a proper k-layered network; dummy nodes are introduced so that each node is on a level below its ancestors; in the fourth step the nodes are sorted on each level, in order to minimize the number of crossings; in the last step the algorithm changes the positions of the nodes on each level (without changing their order) to minimize the number of bends and assigns final coordinates to vertices.

b. for a given set of requirements there often exist several possible layout algorithms;

c. it is often possible to "synthesize" new layout algorithms by composing functional steps from different algorithms; an example on how functional steps from different algorithms can be assembled to form new algorithms is shown later in the paper.

The selection problem (Problem (2)) led to the exact definition of all the possible requirements that may affect the choice of the algorithm. The set of requirements for a given graph is called diagram model (DMModel in the following). Moreover, a tool that is able to find the set of algorithms (in the following a solution space) corresponding to the given DMModel is needed.

The inputs and outputs of ALF are shown in figure (2). The choice of the algorithm (or algorithms) to apply in a certain situation is made by ALF in four steps:

First step: a DMModel is selected by the user from those that fulfill the requirements specified in the CBase by the Client System.

Second step: ALF receives from the user a first description of the diagram to produce, manually composed by means of the User Interface diagram editor.

Third step: the user is enabled to specify his requests for that particular application. These requests typically regard aesthetic features (e.g. even distribution of symbols, crossing minimization, convex faces), semantic features (e.g. high degree vertices positioned in the center, placement of a group of vertices close together) and system response time (computational complexity). According to these requests the ALF performs an evaluation (in what follows labelling) of the algorithms belonging to the solution space. If the user doesn't specify his requests, a default evaluation of the solution space is performed. The default evaluation is based on the previous user's requests for that solution space.

Fourth step: ALF proposes a set of algorithms which give the best graphical results by the user's point of view. The user, acting on icons that synthetically represent the representations, can choose one of them. This activity can be successively repeated to have at disposal several representations.

The architecture that underlies this description includes the following blocks (see figure (3):

- an automatic layout algorithms-base (in the following ABase) including algorithms for trees ([ReTi81], [Va81]), planar graphs ([TaDi88], [Ta87], [TaTo89], [Tu63]), general undirected graphs ([BaNaTa86]), acyclic digraphs ([DiTa88], [DiTaTo91]), hierarchies ([SuTaTo81], [Ca80]), visibility representations ([TaTo86]); upward drawable digraphs ([BaDi91]), compound digraphs ([SuMi89]) and others.
- a tool that manages the ABase (in the following AManager). The AManager fetches from the ABase a solution space according to a certain DMModel. Moreover the AManager is capable to store new proposed algorithms in the ABase.
- a DMModels base (in the following DMBase). The DMBase is a collection of constraints and requirements describing a family of diagrams; to each DMModel is univocally associated a solution space.

- a tool that manages the DMBase (in the following DMDBase). The DMDBase is a client that selects the current DMModel according to Client System's customization (through the CBase) and user's requirements (through the User Interface); it sends to the AManager all the information about current DMModel that it needs to select the right sequences of algorithms.

In the next sections we describe in detail each block of the above architecture.

3. The ABase.

The ABase is a collection of modules, mostly taken from existing layout algorithms. As already observed, these modules are independent functional steps of the above algorithms that can be assembled together through suitable interfaces in order to obtain the algorithms satisfying user's requirements. In this section we describe the ABase of ALF. The section is organized as follows: in 4.1 we present a survey on the general problems involved with the ABase; in 4.2 we show the ABase structure.

3.1 General problems involved with the ABase.

The main problem involved with the ABase is the characterization of the independent functional steps of existing algorithms. To solve this problem, we started from the bibliographic survey by Tamassia and Eades [TaTa89]. They give a classification of the layout algorithms with respect to the classes of graphs they are developed for.

From an analysis of several existing algorithms, we derived:

- a hierarchical structure of the classes of graphs of interest;
- a set of modules (in the following methods) with the following characteristics:
- each method is associated to a class and maps an object of that class into an object of another class;
- a method developed for a class A of graphs can be used by all the classes descending from A in the hierarchy (inheritance);
- two or more independent methods can be sequentially assembled to have a new layout algorithm.

3.2 The ABase structure.

For defining the hierarchical structure and the collection of methods of the ABase, the main problem is to identify "good" classes of graphs; we say that a class is a good class if it includes significant layout algorithms or if the data structure associated to that class is a generalization of more specific classes.

In the following we present some of the classes of the hierarchy. To do that we need to introduce some terminology:

All the ABase algorithms work with graph: a graph is the abstract structure that underlies a diagram.

The most general class of graphs managed by the ABase algorithms is the class of 

a multigraph if, for each pair of adjacent vertices \((u,v)\) there is at least one edge that connects \(u\) and \(v\).

In particular, we consider multigraphs \(G(V,E,D)\) such that \(V\) is the set of vertices, \(E\) is the set of undirected edges which connect pairs of vertices of \(V\) and \(D\) is the set of directed edges which connect pairs of vertices of \(V\).

We assume that every multigraph managed by the ALF is topologically defined, which means that it is given a collection of lists, one for each vertex, where each list specifies the cyclic order (say clockwise) of the edges at the vertex.

To realize the ABase structure we have chosen five main classification coordinates, namely, connectivity, planarity, orientation of the edges, presence of layers, and presence of a shape. The five classification coordinates originated five starting classes:

- \(k\)-connected multigraphs: a multigraph is said to be \(k\)-connected if, for each pair \((u,v)\) of vertices there exist at least \(k\) paths whose start vertex is \(u\) and whose end vertex is \(v\) (we consider all edges as they were undirected).

- Directed multigraphs: a multigraph is said to be directed if \(E\) is empty.

- Planar multigraphs: a multigraph is said to be planar if it can be drawn on the plane without edge crossings.

- Layered multigraphs: a multigraph is said to be layered if it is possible to order the vertices of a multigraph such that all edges go from layers \(L_i\) to \(L_{i+1}\) and there are no edges among vertices in the same layer.

- Shaped multigraphs: a multigraph is said to be shaped if the shape of a subset of its edges is fixed. For example, in the orthogonal graphs all edges are polylines with angles multiple of 90 degree.

Each other class of our classification is a subset of one or more of the above five classes. In figure (4) we show all the classes and their relationships. A directed edge from class \(A\) to class \(B\) means that \(A\) is a subset of \(B\). A precise description of all the classes of the taxonomy could result in a tedious list; we prefer to give an informal description of some of them by means of examples.

Each automatic layout algorithm we have taken into account can be mapped on a path that visits the nodes of the above figure. Moreover, we think that our ABase structure is modular enough to easily embody new classes of multigraphs.

A layout algorithm can be seen as a pipeline process, where each functional step executes a single task and provides output to the next one. Then, the set of all the pipeline processes is a directed graph whose vertices are the ABase classes and whose edges represent every intermediate transformation process of the output.

In figure (5) some of the layout algorithms of the ABase are represented.

We already mentioned the problem of synthesizing new algorithms by using a sequence of methods from different layout algorithms.

From the above point of view the ABase structure can be seen as an algorithm development support tool: when we want to find an algorithm that maps an object of a class into an object of another class, we can use the methods embodied in the taxonomy, adding, possibly, only the methods we need to complete the pipeline process.

For example, we can obtain a polyline representation of a biconnected planar graph with the following three steps: first, we use an \(st\)-numbering to obtain a planar \(st\)-digraph; second, we add extra vertices to obtain a reduced planar \(st\)-digraph (i.e., a planar \(st\)-digraph without transitive edges); finally, we apply a method ([DiTaTo89]) that maps such graph into the polyline class. The synthesized algorithm is described by figure [Synthesis]. Observe that the above algorithm was not previously present in the literature.

4. The DModel

4.1 Preliminaries: the selection problem.

As stated before, ALF automatically selects different algorithms that fit user's application area requirements and sense of aesthetically pleasing drawing.

Two different aspects of the selection problem are:

- ALF needs a flexible and well-defined standard to represent a set of constraints and requirements that the diagram representation has to satisfy;
- ALF must correctly use the above standard both to select a solution space and to establish a hierarchy among algorithms according to how they fit the requirements. The solution space of a layout problem is a subgraph of the aforementioned taxonomy; its vertices are the nodes of the taxonomy; its edges are the methods of the taxonomy. Thus, the solution space contains all the methods suitable for a layout problem.

In this section we describe our approach to the first subproblem. The section is organized as follows: in paragraph 4.2 we give a definition of DModel; in paragraph 4.3 we describe a tool that manages a collection of DModels.

4.2 Definition of DModel.

A DModel is a collection of requirements and constraints on the diagram.

Niceness requirements can be classified as in the following:

- Aesthetic criteria: these are requirements that concern graphic aspects of the diagram representation. Examples of allowed aesthetic criteria are:
  - minimization of the area occupied by the drawing;
  - balance of the diagram with respect to the vertical or horizontal axis;
  - minimization of the number of bends along the edges;
  - minimization of the length of the longest edge;
  - symmetry of sons in hierarchies;
  - uniform density of vertices in the drawing;
  - verticality of hierarchic structure;
- Semantic criteria: these are requirements that deal with features that require a knowledge about the meaning of the diagram representation. For example, if there is a concept of utmost importance, the representation is usually arranged so that the corresponding symbol is placed in the center of the diagram. Examples of allowed semantic criteria are:
  - place a set of given vertices in the center of the drawing;
  - place specified vertices on the external boundary;
  - place close together a group of vertices;
  - draw a subgraph with a specified shape;
  - place a sequence of vertices along a straight line.
A requirement allows the ALF to perform an evaluation (or labelling) of the algorithms belonging to the solution space. A constraint forces the graph representing the solution space to contain a specified set of nodes or a specified set of edges. When the constraint forces the solution space graph to contain a given vertex, it is said to be a node constraint; when it forces the solution space to have a given edge, it is said to be an edge constraint.

For example, the constraint "planar" is a node constraint which forces the solution space to contain only algorithms passing through the vertex "planar." The admitted node constraints can be easily deduced from figure (classes); the set of node constraints corresponds to the set of the ABase classes.

An edge constraint allows the user to specify a method he wants in the solution space; for example, the constraint "idy orthodoxization" forces the solution space to contain the method [Tab87] by Tamassia.

Another type of constraints that can be specified regards the upper bound time complexity of the algorithms of the solution space. This kind of constraint is said to be a time constraint. For example the time constraint $O(n^2)$ forces the ALF to select only algorithms whose time complexity is less or equal than $O(n^2)$.

4.3 The DMBManager.

A Dmodel is said to be consistent if the associated solution space is not empty. The DMBManager is a tool that manages a collection of consistent Dmodels (DMBase in the ALF architecture). It receives from the CBASE and from the user's interface a set of constraints and requirements and selects the current DModel among the available ones.

More precisely, its main features are:

- it allows the user to specify only a set of possible requirements, according to the given constraints; for instance, if one edge constraint is "straight-line" (that means that all the paths of the space end in the "straight-line" node), the requirement "minimum number of bends" has to be allowed;
- it provides a default value for any not specified requirement; the default value is the same that the requirement had in the last application involving the current DModel;
- it is able to define a new DModel in the DMBase according to a new group of requirements; the new DModel definition is made by ALF checking if there is at least one algorithm able to satisfy the new group of constraints and requirements; if this consistency test gives a positive result, then the new DModel is stored in the DMBase;
- it is able to restore a previously DModel associated to a diagram representation stored in the diagram base by the user;
- it provides a default DModel if the set of constraint and requirements received as input is empty.

The DMBModel is still under improvement and implementation.

5. The ABManager.

The ABManager is a tool that receives the current DModel as input and produces as output a set of algorithms that satisfy the given constraint and requirements.

ABManager performs two main phases: in the cutting phase the algorithms of interest are assembled; in the labelling phase a set of them are chosen on the basis of user's requirements.

In what follows we describe each phase with some more detail.

- Cutting phase: the ABManager selects a solution space according to current DModel's constraints. Each assembled algorithm contains the edges specified by the edge constraints, visits the nodes specified by the node constraints and doesn't have a computational complexity greater than the time constraint.

- Labelling phase: in this phase the ABManager first evaluates how the algorithms selected in the cutting phase fit the requirements of the current DModel; then it establishes a priority among them and proposes as output the best suited ones.

The evaluation is performed using a fitting function. This function is computed for each method of the taxonomy; its domain are integers that represent the relative interest of each requirement in the current DModel and its codomain are integers that represent the method fitting degree (i.e. how the algorithm fits user's requirements). This function is a linear combination whose coefficients are numeric parameters which evaluate how the method is able to satisfy the above requirements.

The fitting value of an algorithm assembled by several methods is computed by summing the fitting values of each method.

Acknowledgements

This work is partially supported by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR, Sottoprogetto VI, LRC Infokit. We thank Marco Abbate for his invaluable help in the implementation of a prototype of ALF.

References

[ChPa89] Chrobak, T.H. Payne, A linear-time algorithm for drawing a planar graph on a grid , Manuscript, University of California at Riverside.
figure (1): the main functional steps of the algorithm by Sugiyama et al.

figure (2): ALF as a black box.

figure (3): Architectural description.