How to Visualize a Graph: Specification and Algorithms

Part II: Declarative Approaches

Isabel F. Cruz, Tufts University

Roberto Tamassia, Brown University

© 1994, Isabel F. Cruz and Roberto Tamassia
Outline of Part II

• Introduction and Motivation
• Tightly-coupled vs. Loosely-coupled Approaches.
• Languages for Specifying Layout Constraints
• Drawing of Graphs with Constraints
• Visual Graph Drawing
• Challenges and Open Problems
• Acknowledgements
Introduction and Motivation

• some readability aspects require knowledge about the semantics of the specific graph (e.g., place “most important” vertex in the middle)

• constraints are provided as additional user-defined input to a graph drawing algorithm

Examples

• place a given vertex in the “middle” of the drawing
• place a given vertex on the external boundary of the drawing
• draw a subgraph with a prescribed “shape”
• keep a group of vertices “close” together
Declarative Approach

- Layout of the graph specified by a user-defined set of constraints
- Layout generated by the solution of a system of constraints

Advantages

- Expressive power

Disadvantages

- Some natural aesthetics (e.g., planarity) need complicated constraints to be expressed
- General constraint-solving systems are computationally inefficient
- Lack of a powerful language for the specification of constraints (currently done with a detailed enumeration of facts, or with a set notation)
Declarative Approach

• These approaches cover a broad range of possibilities:

  • **Tightly-coupled**: specification and algorithms cannot be separated from each other.
  
  • **Loosely coupled**: the specification language is a separate module from the algorithms module.
  
  • Most of the approaches are somewhere in between ...

---

Tightly-coupled approaches

**Advantages:**

• The algorithms can be optimized for the particular specification.

• The problem is well-defined.

**Disadvantages:**

• Takes an expert to modify the code (difficult extensibility).

• User has less flexibility.
Loosely-coupled approaches

Advantages:

• Flexible: the user specifies the drawing using constraints, and the graph drawing module executes it.
• Extensible: progressive changes can be made to the specification module and to the algorithms module.

Disadvantages:

• Potential "impedance mismatch" between the two modules.
• Efficiency: more difficult to guarantee.
Languages for Specifying Constraints

- Languages for display specification
  - ThingLab [Borning 81]
  - IDEAL [Van Wyk 82]
  - Trip [Kamada 89]
  - GVL [Graham & Cordy 90]
- Grammars
  - Visual Grammars [Lakin 87]
  - Picture Grammars [Golin and Reiss 90]
  - Attribute Grammars [Zinßmeister 93]
  - Layout Graph Grammars [Brandenburg94] [Hickl94]
  - Relational Grammars [Weitzman & Wittenburg 94]
- Visual Constraints
  - U-term language [Cruz 93]
  - Sketching [Gleicher 93] [Gross94]

**Visual**
- Used in GD
- Used in GD and Visual
**ThingLab** [Borning 81]

- Graphical objects are defined by example, and have a **typical** part and a **default** part.
- Constraints are associated with the classes (methods specify constraint satisfaction).
- Object-oriented (message passing, inheritance).
- Visual programming language.

**Ideal** [Van Wyk 82]

- Textual specification of constraints.
- Graphical objects are obtained by instantiating abstract data types, and adding constraints.
- Uses complex numbers to specify coordinates.

**GVL** [Graham & Cordy 90]

- Visual language to specify the display of program data structures.
- Pictures can be specified **recursively** (the display of a linked list is the display of the first element of the list, followed by the display of the rest of the list).
Constraint specification in TRIP
[Kamada 89]

- A **graphical object** consists of:
  - a set of variables
  - a set of constraints
  - a set of drawing instructions.

- **Geometric relations** among graphical objects are expressed as extra constraints such as:
  - alignment
  - ordering of x- or y- coordinates.

  (Geometric relations should be generic.)

- The layout specification and the constraints are specified textually using a Prolog-like syntax.
Layout Graph Grammars
[Brandenburg 94] [Hickl 94]

• Grammatical (or rule-based methods) for drawing graphs

• Extension of a context-free string grammar
  • underlying context-free graph grammar
  • layout specification for its productions

• By repeated applications of its productions, a graph grammar generates labeled graphs, which define its graph language

• For appropriate layout graph grammars, optimal graph drawings can be constructed in polynomial time:
  • H-tree layouts of complete binary trees
  • hv-drawings of binary trees
  • series-parallel graphs
  • NFA state transition diagrams from regular expressions
Attribute Grammars

[Zinßmeister 93]

• Generalization of attribute string grammars.
• Production rules have a visual notation.
• So far are limited to trees, but are expressive. For example they can express the Reingold Tilford algorithm.
Picture Grammars
[Golin & Reiss 90, Golin 91]

• Production rules use constraints.

• Terminals are:
  - shapes (e.g., rectangle, circle, text)
  - lines (e.g., arrow)

• spatial relationships between objects are operators in the grammar (e.g., over, left_of)

FIGURE $\rightarrow$ over (rectangle$_1$, rectangle$_2$)
Where
rectangle$_1$.lx == rectangle$_2$.lx
rectangle$_1$.rx == rectangle$_2$.rx
rectangle$_1$.by == rectangle$_2$.ty

* More expressive relationships: tiling.

* Complexity of parsing has been studied.
Relational Grammars  
[Weitzman & Wittenburg 93, 94]

• Generalization of attribute string grammars that allow for the specification of geometric positions in 2D and 3D, topological connectivity, arbitrary semantic relations holding among information objects.

\[
\text{Article} \rightarrow \text{Text Text Text Number Image}
\]

\[
(\text{Defrule (Make-Article The-Grammar)} \\
\hspace{1em} (0 \text{ Article}) \\
\hspace{1em} (1 \text{ Text}) \\
\hspace{1em} (2 \text{ Text (Author-Of 2 1)}) \\
\hspace{1em} \ldots \\
\hspace{1em} :\text{OUT} \\
\hspace{1em} ( \\
\hspace{2em} \ldots \\
\hspace{2em} (\text{spaced-below 2 1}) \\
\hspace{2em} (\text{spaced-below 3 1}) \\
\hspace{2em} (\text{set-font 1 10pt :bold}) \\
\hspace{2em} (\text{set-font 1 8pt :italic}) \\
\hspace{2em} \ldots \\
) \\
)\]

• Constraints are solved with DeltaBlue (U. of Washington) for non-cyclic constraints.
Visual Grammars
[Lakin 87]

- **Context-free grammar.**
- **Symbols are visual, and are visually annotated.**

The interpretation of the visual symbols is left to the implementation.
U-term Language
[Cruz 93, 94]

- Visual constraints.
- Simplicity and genericity of the basic constructs.
- Ability to specify a variety of displays: graphs, higraphs, bar charts, pie charts, plot charts, . . .
- Compatibility with the framework of an object-oriented database language, DOODLE.
- Recursive visual specification.
Expressing Constraints by Sketching

• **Briar** [Gleicher 93]

**Constraint-based drawing program:**
- Direct manipulation drawing techniques.
- Makes relationships between graphical objects persistent
- Performance concerns in solving constraints.

• **Spatial Relation Predicates** [Gross 94]

![Diagram showing spatial relations between objects](image)

- (CONTAINS BOX CIRCLE)
- (CONTAINS BOX TRIANGLE)
- (IMMEDIATELY-RIGHT-OF CIRCLE TRIANGLE)
- (SAME-SIZE CIRCLE TRIANGLE)

- Applications include retrieval of buildings from an architecture database.
Other Declarative Graph Drawing Approaches

- Marks
- Kamada
- Eades and Lin
- Cruz, Garg, Tamassia, Van Hentenryck
ANDD
[Marks et al.]

• Layout-aesthetic concerns subordinated to perceptual-organizational concerns

• Notation for describing the visual organization of a network diagram

• Layout task as a constrained optimization problem:
  • constraints derived from a visual-organization specification
  • optimality criteria derived from layout-aesthetic considerations

• Two heuristic algorithms:
  • Rule-based strategy
  • Massive parallel genetic algorithm
TRIP
[Kamada 89]

- **Framework for visualizing abstract objects and relations.**

- **Constraint-based object layout system (COOL):**
  - rigid constraints
  - pliable constraints
  - conflicting constraints can be solved approximately.

```
original textual representation

Analyzer

relational structure representation

Visual Mapping

visual structure representation

COOL

layout library

target pictorial representation
```
Visual Graph Drawing
[Cruz, Tamassia & Van Hentenryck 93]
[Cruz & Garg 94]

• A new approach to graph drawing

Visual: reconciles expressiveness with efficiency

Goals

• Visual specification of layout constraints: the user should not have to type a long list of textual specifications
• Visual specification of aesthetic criteria associated with optimization problems
• Extensibility: the user should not be limited to a prespecified set of visual representations.
• Flexibility: the user should not have to give precise geometric specifications.
Visual Graph Drawing (Tree Layout)

\[ WL + 1 \ [h] \]
\[ 1 \ [v] \]
\[ 1 \ [v] \]
\[ W_L \ [h] \]
\[ H_L \ [v] \]
\[ H_R \ [v] \]
\[ R \]
\[ max(H_L, H_R) \ [v] \]

\[ T:binTree[root\rightarrow N:node; \text{left}\rightarrow L:binTree; \text{right}\rightarrow R:binTree] \]
Characteristics of the Previous Tree Drawings

- **Level Drawings**
  - Upward
  - Planar
    - Nodes at the same distance from the root are horizontally aligned.

- **Display of symmetries.**

- **Display of isomorphic subtrees.**
Change a few things . . .

Declarative Approaches to GraphVisualization
A Visual Graph Drawing System

- We envision a system consisting of:
  - a **declarative component** based on a **DOODLE**.
  - an **algorithmic component** based on an algorithms database (e.g., Diagram Server).
  - a **compiler** that translates visual specifications into a drawing algorithm synthesized from the database (e.g., extension of parametric graph drawing [Bertolazzi, Di Battista & Liotta 91]).
Efficient Visual Graph Drawing
[Cruz & Garg 94]

• Recognize classes of graphs and drawings that can be expressed with DOODLE and evaluated efficiently.

• Devise algorithms and data structures for performing drawings in linear time (optimal time):
  • Trees (upward drawing, box inclusion drawing).
  • Series-parallel digraphs (delta drawing).
  • Planar acyclic digraphs (visibility drawing, upward planar polyline drawing).

• Next:
  • Extend above results to other classes of graphs and drawings.
  • Constraint viewpoint: framework for evaluating constraints efficiently.
  • Incorporate these algorithms into a declarative graph drawing system that uses DOODLE.
More examples

• Series-parallel graphs / delta-drawings [Bertolazzi, Cohen, Di Battista, Tamassia & Tollis, 92]
Declarative Approaches to Graph Visualization

**deltaGraph**

- **connects** $(x,y)$
- **series** $(x,y)$
- **parallel** $(x,y)$
- **sp-digraph** $(G_1)$
Challenges and Open Problems (Declarative Approach):

- New approach, therefore much left to explore, in particular:
  - New specification languages.
  - Reducing the “impedance mismatch.”
  - Design of user interfaces, and evaluation in different environments/applications.
  - Identification of levels of complexity in drawing graphs (e.g., with graph grammars, constraint languages).
  - Expressiveness of the specification languages, in particular of declarative and visual languages.
  - Refinement of the diagram server hierarchy, so that we can have a true “tool box” for the declarative, loosely-coupled approach.
Acknowledgements

Franz Brandenburg, University of Passau

Pino Di Battista, University of Rome

Peter Eades, University of Newcastle

Ashim Garg, Brown University

Timo Hickl, University of Passau

Tomihisa Kamada, ACCESS Co., Ltd

Tao Lin, CSIRO Canberra

Joe Marks, MERL

Steve Reiss, Brown University

Yanni Tollis, UT Dallas

Louis Weitzman, MIT Media Lab

Kent Wittenburg, Bellcore