Hamburg University of Technology

Automatic Orthogonal Graph Layout

Jianing Sun

Supervised by
Prof. Dr. Ralf Möller
M. Sc. Miguel Garcia

June 2007

Declaration

Hereby, I declare that:

This student project, with the subject “Automatic Orthogonal Diagram Layout”, has been prepared by me. All literal and content related quotations from other sources are clearly pointed out, and no other sources or aids than the declared ones have been used.

Jianing Sun
Hamburg, June 2007

Acknowledgements

Hereby, I would like to thank Prof. Dr. R. Möller for finding this student project topic and supervising it.

Furthermore, M. Sc. Miguel Garcia was very patient and helpful for answering my questions and providing programming tools in order to develop the applications. His advices directed the project progress for a better implementation.

Special thank to Dr. Markus Eiglsperger, Universität Tübingen, since this work chiefly based on his doctoral thesis “Automatic Layout of UML Class Diagrams: A Topology-Shape-Metrics Approach”.

Contents

Chapter 1

Introduction

In this student work we define the automatic layout problem for EMF diagrams and propose an algorithm based on the Topology-Shape-Metrics approach to solve it. The Topology-Shape-Metrics approach has been intensively studied in the last years in the area of graph drawing and has been used successfully in application areas like visualization of UML class diagram or database schema. We will see how this graph drawing paradigm can be promisingly used for the visualization of EMF diagrams.

1.1 Motivation

The Eclipse Modeling Framework *EMF* is a powerful open-source JAVA/XML framework and code-generation facility, which already widely adopted, for generating tools and building Java applications based on a structured data model. EMF helps rapidly turn models into efficient, correct, and easily customizable Java code. It is intended to provide the benefits of formal modeling, but with a very low cost of entry. In addition to code generation, it provides the ability to save objects as XML documents for interchange with other tools and applications. EMF provides its own meta-model, called Ecore model for describing application data models, which called core models; an XMI serializer for persisting models; tools for transforming model forms like UML, XML Schema and simple annotated Java interfaces into Ecore; and powerful code generator tools, which are used to produce high-quality Java code from Ecore model descriptions.

Emfatic language for EMF Development is a language for creating, editing and representing EMF Ecore models in textual form. Emfatic comprises several Eclipse plug-ins, which provide a parser for the language and a basic text editor based on the Eclipse development platform, which also add two actions to allow Emfatic source code to be complied into an Ecore model and allow Ecore models to be decompiled back to Emfatic source code. The advantages of Emfatic in comparison with other EMF tools including annotated Java, XML Schema, Unified Modeling Language (UML) tools and the EMF model editor are that Emfatic represents the entire EMF Ecore model in a single source file, uses a Java-Like syntax familiar to many programmers, and it closely combined with the Eclipse IDE.

Unfortunately there is no visualization tool for EMF, which representing EMF Ecore models in visual form, which we called *EMF diagrams*. In this case a visualization tool embedding in

Emfatic for EMF diagram drawing is desirable and the placement of the diagram elements is determined automatically, because the diagram elements are generated automatically. Figure 1.1 shows an Example for a manually drawn EMF diagram.

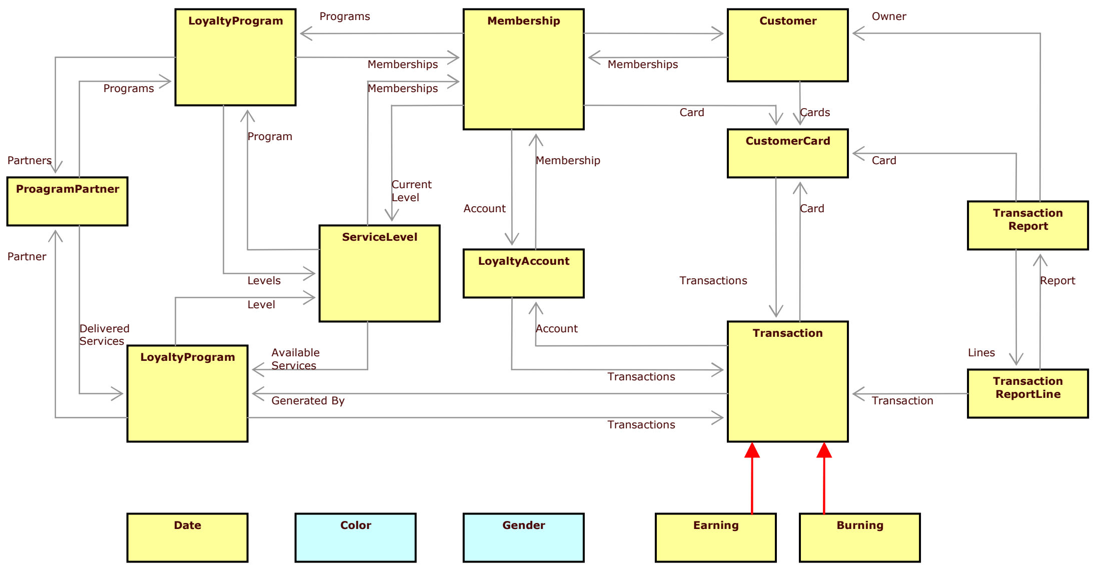


Figure 1.1: An EMF Diagram

There is no mathematical definition of aesthetics for graph drawing. It can be defined informally that a drawing of graph is more aesthetic than another if it looks “nicer” or it is “more readable”. Thus the concept *aesthetic criterion* is used to mathematically describe aesthetics of graph drawing **[Sec. 2.2]**.

Current automatic orthogonal layout algorithms for diagram drawing are based on the hierarchic graph drawing paradigm and focus on the direction of **FLOW** aesthetic criterion. Applied to diagram drawing this aesthetic criterion says that all edges of some type should point in a common direction, i.e., all edges representing generalizations point upward. These algorithms produce good results with large and deep structural information, i.e., diagrams with a large and deep inheritance hierarchy. However, they do not perform satisfactorily in absence of this information. We propose in this work a new algorithm for automatic layout of diagram drawing which is based on the topology-shape-metrics approach. The algorithm is an adaptation of sophisticated graph drawing algorithms which have proven their effectiveness in many applications. The algorithm works as well for diagrams with rich structural information as for diagrams with few or no structural information. It improves therefore the existing algorithms significantly.

1.2 Overview

This document is organized in to five chapters:

In Chapter 2 we will review the main mathematical concepts that we will use in the remainder of this work: graph and graph drawing, planarity, planar representation, embedding, orthogonal shape, aesthetic criterion and the Min-Cost-Flow problem.

In Chapter 3 we discuss the EMF diagram model in detail and present a graph based model for it.

In Chapter 4 we present the Topology-Shape-Metrics approach for automatic orthogonal diagram drawing. The three phases, planarization, orthogonalization and compaction will be discussed deeply.

We finish with Chapter 5, which contains a conclusion of the presented work and show same sample diagram layout using the Topology-Shape-Metrics approach.

Chapter 2

Preliminaries

In this chapter we will review the main mathematical concepts that we will use in the remainder of this work: graph and graph drawing, planarity, planar representation, embedding, orthogonal shape, aesthetic criterion and the Min-Cost-Flow problem.

2.1 Graph Theory

2.1.1 Graphs

In this section we introduce basic concepts from graph theory, mainly based on **[5]**. For a comprehensive overview of graph theory we refer to **[20] [21]**.

A *graph* G is denoted by a pair $G=(V,E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. We denote with $adj(v)$ the set of edges adjacent a vertex $v \in V$. The degree $\delta_G(v)$ of a vertex $v \in V$ is the number of edges in E adjacent to v. A graph is called *4-graph* if each vertex has degree maximal possible 4. We say that $G' = (V',E')$ is a subgraph of $G = (V,E)$ if $V' \subseteq V$ and $E' \subseteq E$. In this case we write $G' \subseteq G$. A graph isomorphism f: $V(G) \rightarrow V(H)$ is a bijection between the vertices of two graphs G and H with the property that any two vertices u and v from G are adjacent if and only if $f(u)$ and $f(v)$ are adjacent in H. If an isomorphism can be constructed between two graphs, then we say those graphs are isomorphic.

We call a graph *directed* if all pairs in E are ordered and *undirected* if all pairs in E are unordered. We call the first entry in a directed edge the *source* and the second *target*. Ignoring for every directed edge the order of its vertices, we get an undirected graph, which is called the *underlying graph*. For a vertex $v \in V$, we denote with $in(v)$ the set of edges in E which have target v, and with $out(v)$ the set of edges with source v. The in-degree $\delta_G^-(v)$ denotes the number of edges in $in(v)$, and the out-degree $\delta_G^+(v)$ the number of edges in $out(v)$. We call a vertex with in-degree 0 a *source*, and a vertex with out-degree 0 a *sink*. A directed acyclic graph is called a *st-graph* if it has exactly one sink and one source.

If a graph contains both, directed and undirected edges, we call it a *mixed graph*. In this case we denote the set of directed edges with $E_d(G)$ and the set of undirected edges with $E_u(G)$. Often the shorter terms *digraph*, resp. *migraph*, are employed instead of the terms directed graph, resp. mixed graph.

2.1.2 Graph Drawing

A *point drawing* Γ of a graph $G = (V,E)$ maps each vertex $\upsilon \in V$ to a point $p(\upsilon)$ in the plane and each edge $e = (\upsilon,\omega) \in E$ to an open Jordan curve $c(e)$ such that $c(e)$ connects $p(\upsilon)$ with $p(\omega)$. A *rectangle drawing* Γ of a graph $G = (V,E)$ maps each vertex $\upsilon \in V$ to a rectangle r(v) in the plane and each edge $e = (\upsilon,\omega) \in E$ to an open Jordan curve $c(e)$ such that c(e) connects $r(\upsilon)$ with $r(\omega)$. An *orthogonal drawing* of a graph is a point drawing in which the curve for each edge is a sequence of horizontal and vertical segments. Note that a graph admits an orthogonal drawing if and only if it is a 4-graph. An *orthogonal rectangle drawing* of a graph is a rectangle drawing in which the curve for each edge is a sequence of horizontal and vertical segments. For an illustration see Figure 2.1.

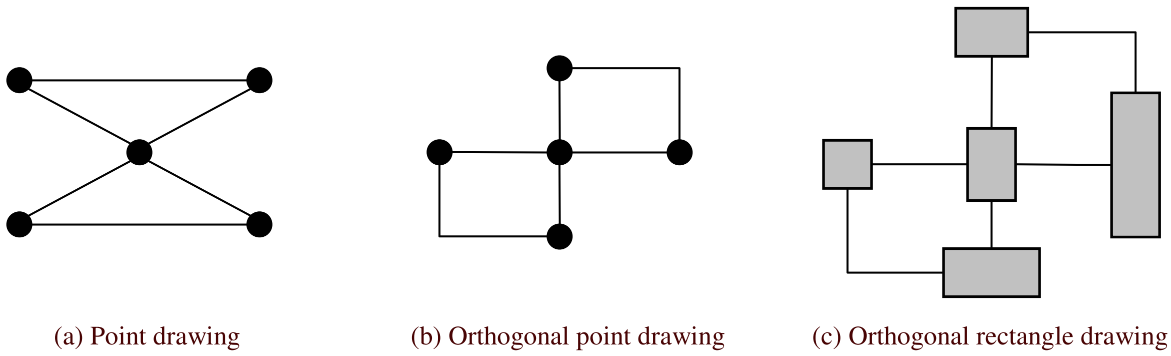

(a) Point drawing
(b) Orthogonal point drawing
(c) Orthogonal rectangle drawing

Figure 2.1: Example for different types of graph drawings.

2.1.3 Planar

A point drawing Γ of a graph $G = (V,E)$ is *planar* if no two edges in the drawing intersect except at common points. A graph is *planar* if it has a planar point drawing. A combinatorial description of planar graphs will be defined:

> *"A graph G is planar if and only if it contains no subgraph that is isomorphic to or is a subdivision of K_5 (the complete graph with 5 vertices)* or *$K_{3,3}$ (the complete bipartite graph with 3 vertices in each side)."*

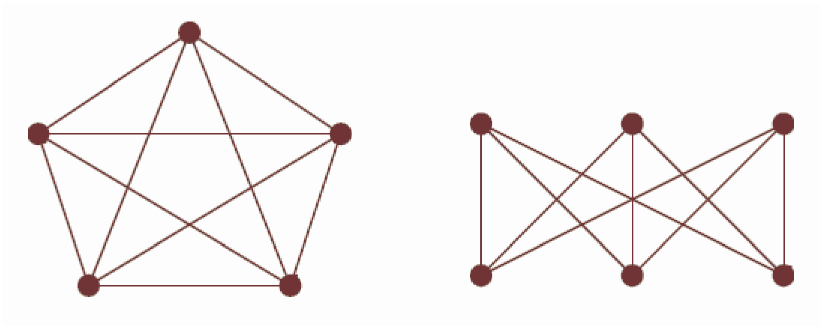


Figure 2.2: The two minimal non-planar graphs K_5 and $K_{3,3}$**[5]**.

2.1.4 Upward Planar

An *upward drawing* of a directed graph is a drawing in which each edge is represented by a curve monotonically increasing in the vertical direction. A drawing of a directed graph is *upward planar* if it has an upward planar drawing.

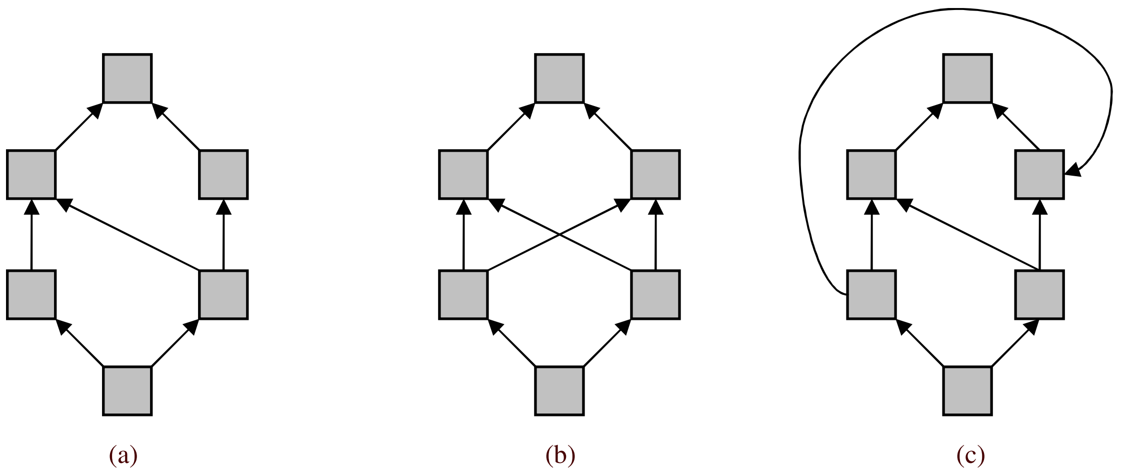


Figure 2.3: Upward planar drawing (a). Upward non-planar drawing (b). Non-upward planar drawing (c).

2.1.5 Mixed Upward Planar

A *mixed upward drawing* of a mixed graph G is a drawing in which each directed edge of G is represented by a curve monotonically increasing in the vertical direction. A mixed graph G is called *mixed upward planar* if it has a planar mixed upward drawing.

2.1.6 Face, Planar Representation and Embedding

If Γ is a planar drawing, the set $\mathrm{IR}^2 \setminus \Gamma$ is open and its regions are called the *faces* of Γ. Since Γ is bounded, exactly one of the faces is unbounded. This face is called the *outer face* of Γ. The boundary of each face is a cycle in the graph.

A convenient encoding of a planar drawing is a *planar representation*. A planar representation F of a planar graph $G = (V,E)$ defines for each edge $(\upsilon,\omega) \in E$, which might be directed or undirected, two darts $e = (\upsilon,\omega)$ and $e^R = (\upsilon,\omega)$. We say that e^R is the reverse of e and vice versa. We denote with $\overline{e}$ the reverse of a dart e. We denote with $\overline{E}$ the set of darts defined by E. The planar representation F contains one cyclic list for each face, which contains the darts encountered by walking in clockwise ordering around the face. The first face in F is by convention the outer face and is denoted by f_{out}. When we use the term face in the remainder of this work, we refer to the list of darts describing the face. For a dart e we denote with *face*(e) the face which contains e.

An *embedding* ε of a graph is defined as the counter-clockwise cyclic ordering ε (υ) of the adjacent edges of each vertex v of the graph. Each edge $e = (\upsilon,\omega) \in E$ appears twice in ε, namely as (υ,ω) in ε (υ) and as (υ,ω) in ε (ω). An embedding is planar if there is a planar drawing of the graph which preserves this ordering.

It is easy to obtain the planar representation from an embedding and vice versa, since planar representation and embedding are dual problems in discrete mathematics. Given an embedding ε we denote the planar representation induced by ε with $F\varepsilon$, and given a planar representation F we denote the embedding induced by F with ε_F . A graph with a given planar representation F is called a *plane graph* and is denoted with $G = (V,E,F)$. We will omit the index F in ε_F and write just ε if it is clear to which planar representation we refer.

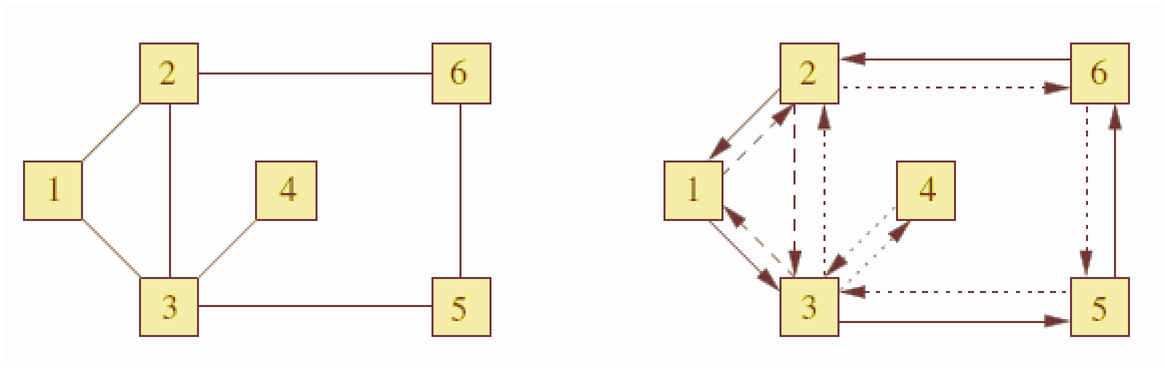


Figure 2.4: An example of planar embedding, taken from **[6]**.

Figure 2.4 illustrates the definition of planar representation on an example. The plane graph is defined by the planar representation $G = (V,E,F)$ and $F = (f_0, f_1, f_2)$ where

$f_0 = \{(1, 3), (3, 5), (5, 6), (6, 2), (2, 1)\}$,
$f_1 = \{(1, 2), (2, 3), (3, 1)\}$,
$f_2 = \{(5, 3), (3, 4), (4, 3), (3, 2), (2, 6), (6, 5)\}$.

In Figure 2.4 (b) the face f_0 is denoted by the solid darts, the face f_1 by the dashed darts and the face f_2 by the pointed darts. The corresponding embedding ε is:

$\varepsilon(1) = \{(1, 2), (1, 3)\}$,
$\varepsilon(2) = \{(2, 1), (2, 3), (2, 6)\}$,
$\varepsilon(3) = \{(3, 2), (3, 1), (3, 5) , (3, 4)\}$,
$\varepsilon(4) = \{(4, 3)\}$,
$\varepsilon(5) = \{(5, 6), (5, 3)\}$,
$\varepsilon(6) = \{(6, 2), (6, 5)\}$.

2.1.7 Orthogonal Shape

An *orthogonal shape H* for *G* is an extension of planar representation and describes, in addition to the topology, the *shape* of a drawing for *G* by specifying the bends in the edges and angles inside the faces. Let $G = (V,E,F)$ be a place 4-graph. An *orthogonal shape H* is a mapping from the set of faces in *F* to clockwise ordered lists of tuples (e,b,a). The first entry in the tuple corresponds to the dart in the face. The second entry is a bit string denoting the bends of the dart. A 0 in the bit string denotes a *convex* bend (90°), while a 1 denotes a *concave* bend (270°), with ε denotes no bends in the dart. The third entry is the angle formed with the preceding dart in the face.

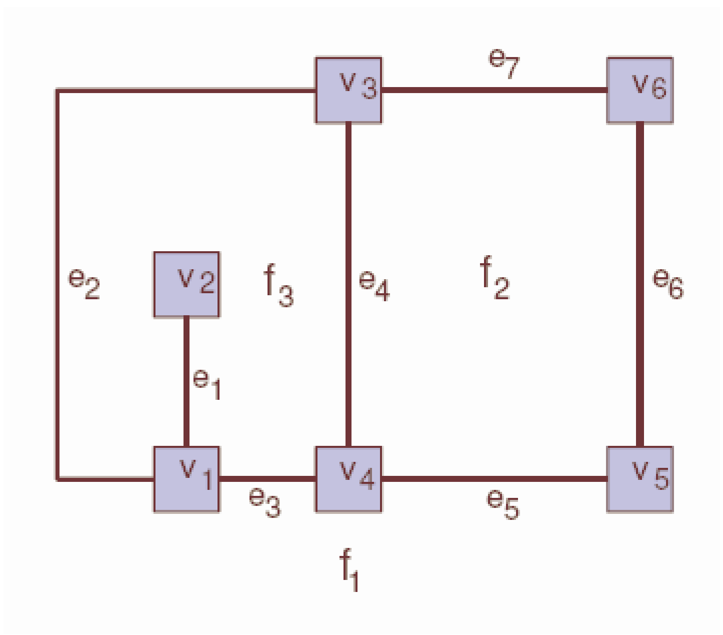


Figure 2.5: An example of orthogonal shape, taken from **[13]**.

Orthogonal shape of the graph depicted in Figure 2.5:
$H(f_1) = \{(e_2, 11, 180°), (e_3, \varepsilon, 0°), (e_5, \varepsilon, 0°), (e_6, \varepsilon, 90°), (e_7, \varepsilon, 90°)\}$ outer face,
$H(f_2) = \{(e_6, \varepsilon, 90°), (e_5, \varepsilon, 90°), (e_4, \varepsilon, 90°), (e_7, \varepsilon, 90°)\}$,
$H(f_3) = \{(e_1, \varepsilon, 90°), (e_1, \varepsilon, 180°), (e_2, 00, 270°), (e_4, \varepsilon, 90°), (e_3, \varepsilon, 90°)\}$.

2.2 Aesthetic Criterion

There is no mathematical definition of aesthetics for graph drawing. It can be defined informally that a drawing of graph is more aesthetic than another if it looks "nicer" or it is "more readable". Thus the concept *aesthetic criterion* is used to mathematically describe aesthetics of graph drawing. One aesthetic criterion measures one isolated mathematically defined property of the drawing and defines rules for the values of this property **[5]**.

The important aesthetic criteria for diagram drawing are:

- Minimize number of edge crossings **[CROSSING]**
- Minimize number bends **[BEND]**
- Minimize number of node and edge overlap **[OVERLAP]**
- Maximize number of orthogonal edges **[ORTHOGONAL]**
- Minimize edge length **[EDGELENGTH]**
- Minimize area **[AREA]**
- Maximize number of edges respecting flow **[FLOW]**

These aesthetic criteria will guide the design of the automatic layout algorithm.

Some of the above criteria can be contradicting, e.g., **CROSSING** and **AREA**. Therefore finding an aesthetic for graph drawing can be seen as solving a multi-objective problem, the objective function is the set of aesthetic criteria **[12]**.
The on Topology-Shape-Metrics approach based orthogonal layout algorithm presents in this work tries to optimize all the aesthetic criteria above.

2.3 The Min-Cost-Flow Problem

Graph orthogonalization will be treated as solving a minimum cost maximum flow problem; we present here the Min-Cost-Flow problem and its variation.

The Min-Cost-Flow problem is defined as following:

Given a flow network G=(V,E) *with source* $s \in V$ *and sink* $t \in V$*, where edge* $(u,v) \in E$ *hast capacity* c(u,v)*, flow* f(u,v)*, and the cost of sending this flow is* $f(u,v) \cdot a(u,v)$*. An amount of flow* d *is required.*

The definition of the problem is to minimize the **total cost** of the flow:

Min: $\sum f(u,v) \cdot a(u,v)$
Where:

Capacity constraints: $f(u,v) \leq a(u,v)$.
Skew symmetry: $f(u,v) = -f(v,u)$.
Flow conservation: $\sum f(u,w) = 0$, for all $u \neq s, t$.
Required flow: $\sum f(s,w) = \mathrm{d}$.

A variation of this problem is to find a flow which is maximum, but has the lowest cost among the maximums. This could be called a **minimum cost maximum flow problem**. In this work, the algorithm to find an orthogonal shape of an input embedded graph is to solve a minimum cost maximum flow problem **[sec 4.3]**.

Chapter 3

Graph Based Model

In this chapter we discuss the EMF diagram model in detail and present a graph based model for it.

3.1 A Graph Based Model for EMF Diagram

There are no formal specifications for EMF diagram; there is also no representing of EMF Ecore models in visual form. Since EMF diagrams are similar to UML class diagrams in the visualization, we discuss the EMF models, give a definition of EMF diagram and present a graph based model to form the problem as in **[5]**.

EMF diagrams are graphs containing nodes connected by edges. The information is mostly in topology, not in the size or placement of the symbols.

The EMF diagram graph is defined as follows:

- *A mixed graph $G=(V,E)$*
- *A vertex mapping $V \rightarrow \{EClass, EDataType, EEnum\}$*
- *An edge mapping $E \rightarrow \{ESuperType, EReference\}$*
- *A size mapping $V \rightarrow IN^2$*

A drawing of EMF diagram graph defines a drawing for EMF diagram.

The EMF diagram layout is defined as follows:

A layout of a EMF diagram graph *G* is defined as a mapping *$\Gamma(G)$* of the vertices to rectangles of size as defined by the mapping size and the edges to open jordan curves.

In the following we will discuss the visual notation of EMF diagrams and define the mapping of the diagram elements to graph elements in the EMF diagram graph.

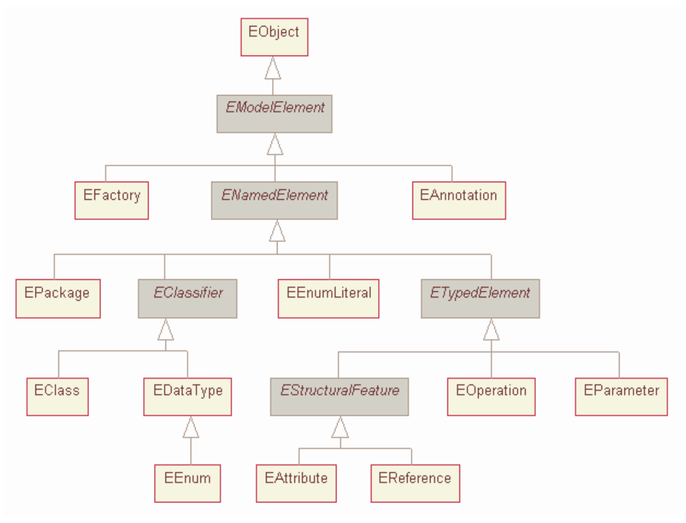


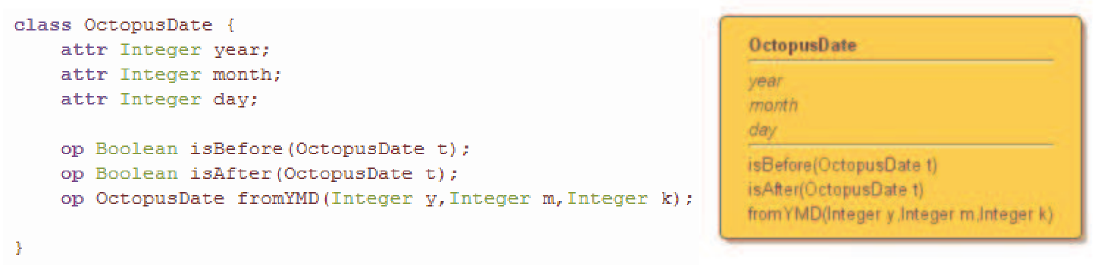
Figure 3.1: complete class hierarchy of the Ecore model **[16]**.

3.1.1 Semantic Entities Mapping to a Vertex

The diagram elements EClass, EDataType, and EEnum correspond directly to a vertex in the graph. Figure 3.1 is the complete class hierarchy of the EMF Ecore model, in the EMF diagram only the partial top-levels leaves will be represented.

EClass

EClass is represented by round rectangle consisting of multiple compartments. Each compartment contains different features of the EClass including EAttribute, EOperation.

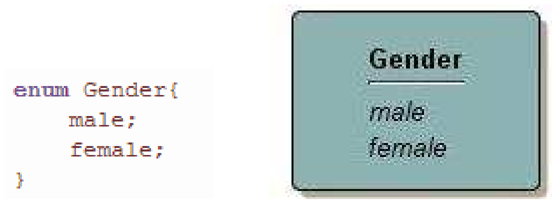


```
class OctopusDate {
    attr Integer year;
    attr Integer month;
    attr Integer day;

    op Boolean isBefore(OctopusDate t);
    op Boolean isAfter(OctopusDate t);
    op OctopusDate fromYMD(Integer y,Integer m,Integer k);

}
```

Figure 3.2: An EClass presents in Emfatic editor and the representation of it in EMF diagram.

EEnum

EEnum is also represented by round rectangle consisting of only one compartment, namely EEnumLiteral.



```
enum Gender{
    male;
    female;
}
```

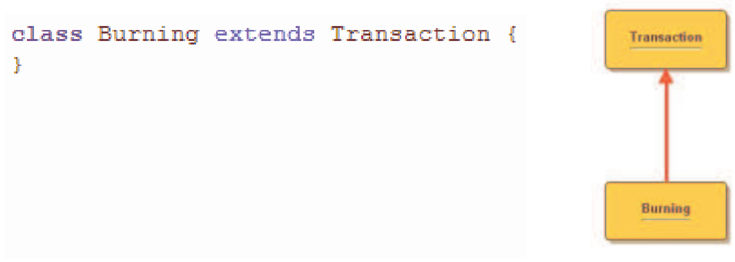
Figure 3.3: An EEnum presents in Emfatic editor and the representation of it in EMF diagram.

The same for EDataType etc.

3.1.2 Semantic Entities Mapping to an Edge

The diagram elements ESuperType and EReference correspond directly to an edge in the graph. The type of the edge is the type of the diagram element. See also **[Figure 3.1]**.

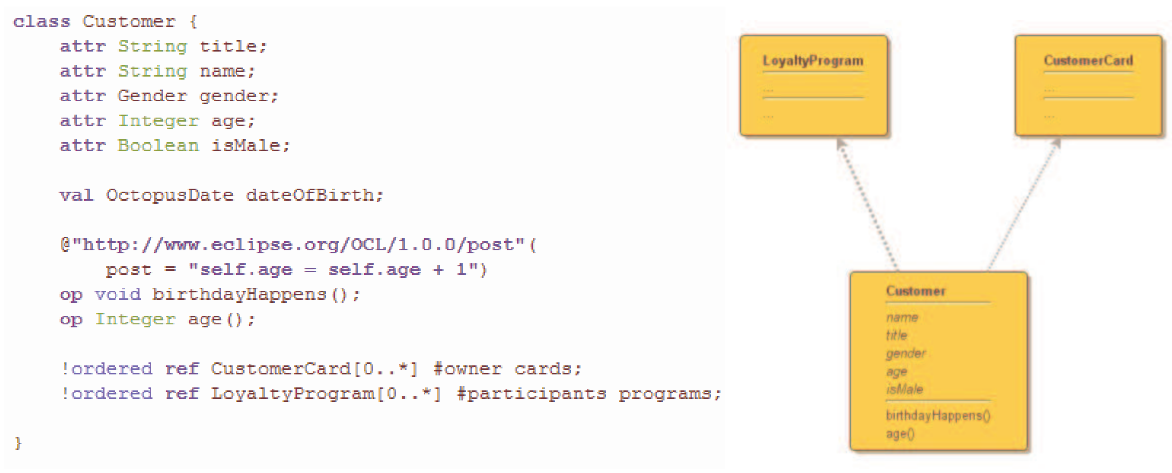
ESuperType
ESuperType is rendered as a solid line with an arrow pointing to the target.



```
class Burning extends Transaction {
}
```

Figure 3.4: An ESuperType presents in Emfatic editor and the representation of it in EMF diagram.

EReference
EReference is rendered also as dashed line with an arrow pointing to the target.



```
class Customer {
    attr String title;
    attr String name;
    attr Gender gender;
    attr Integer age;
    attr Boolean isMale;

    val OctopusDate dateOfBirth;

    @"http://www.eclipse.org/OCL/1.0.0/post"(
        post = "self.age = self.age + 1")
    op void birthdayHappens();
    op Integer age();

    !ordered ref CustomerCard[0..*] #owner cards;
    !ordered ref LoyaltyProgram[0..*] #participants programs;

}
```

Figure 3.5: An EReference presents in Emfatic editor and the representation of it in EMF diagram.

Interestingly there are two-way EReference between two EClass, this is already a circle between two adjacent vertices in the EMF diagram graph. In this case, we draw only one edge which with two arrows in both direction between the vertices and denote the edge as an undirected edge for the orthogonal layout algorithm presents later in this work.

Label Placement
Labelling of edges, such as association multiplicities, role names, is treated separately as additional nodes in **[5]**, in this work it will not be coved.

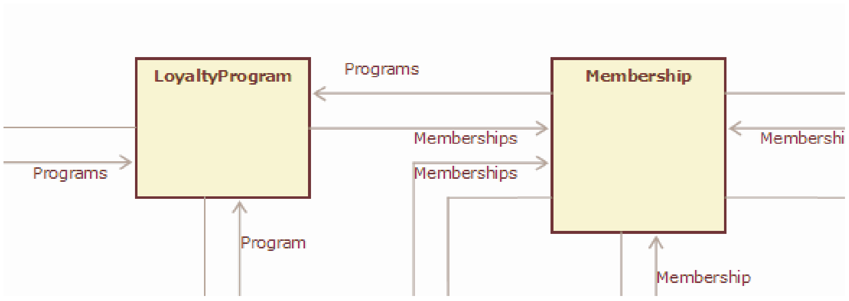


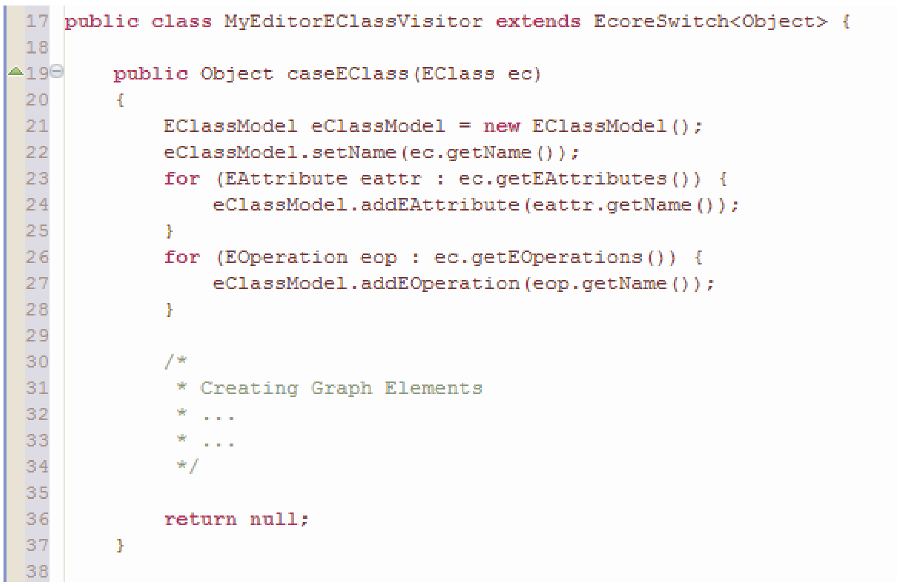
Figure 3.6: Labelling of edges (A part of Figure 1.1).

3.2 The Interface to the Algorithm

Thus we have the input for the orthogonal layout algorithm. The algorithm assumes as input an EMF diagram graph which consists of a graph $G = (V,E)$, a mapping S: $V \rightarrow IN^2$ denoting the size of the nodes in the drawing. To each node, edge a type is assigned denoting its semantics in the EMF diagram:

Node types	**EClass, EDataType, EEnum**
Edge types	**ESuperType, EReference**

The graph based model for EMF diagrams will be obtained by using visitors, which parse a textual EMF expression to evaluating expression on object population.



```
public class MyEditorEClassVisitor extends EcoreSwitch<Object> {

    public Object caseEClass(EClass ec)
    {
        EClassModel eClassModel = new EClassModel();
        eClassModel.setName(ec.getName());
        for (EAttribute eattr : ec.getEAttributes()) {
            eClassModel.addEAttribute(eattr.getName());
        }
        for (EOperation eop : ec.getEOperations()) {
            eClassModel.addEOperation(eop.getName());
        }

        /*
         * Creating Graph Elements
         * ...
         * ...
         */

        return null;
    }
```

Figure 3.7: A sample EClass visitor class.

Chapter 4

The Topology-Shape-Metrics Approach

In this Chapter we present the Topology-Shape-Metrics approach for automatic orthogonal diagram drawing. The three significant phases, planarization, orthogonalization and compaction will be discussed. For more detailed information on the algorithms here described refer to **[5][6][10][12][13]**.

4.1 The Algorithmic Framework Overview

The Topology-Shape-Metrics approach origins from the seminal paper of Tamassia (1987). The Topology-Shape-Metrics approach is one of the most popular graph drawing methods, it has been applied successfully to application domains like the visualization of data flow diagrams, database schemas and industrial plant schemas. In a comparison of four graph drawing algorithms for orthogonal drawings, the one following the Topology-Shape-Metrics approach was the clear winner **[22]**. We now outline the complete algorithm.

We assume that the input graph is connected, if this assumption is violated, the input graph should be divided into its connected components and each connected component will be processed separately. The whole diagram of the connected components can then be arranged by a floor planning algorithm **[14]**. We assume furthermore that the directed subgraph of the input graph is acyclic. If the directed subgraph D contains cycles, only a subset of the edges will be drawn, which induces an acyclic upward subgraph, the remainder directed edges will be handled as undirected edges.

Like other former orthogonal layout algorithms such as the hierarchical approach the Topology-Shape-Metrics approach is divided into several steps **[5]**:

Preprocessing:

(a) Divide the graph into its connected components. Each connected component will be processed separately by the algorithm. Thus we can assume that the input graph is connected.

(b) Remove edges from D until the directed edges induce an acyclic subgraph of G.

(c) If the edges in *D* do not induce a connected subgraph some edges are added temporarily to *D* to make this subgraph connected by using a minimum spanning tree algorithm.

Planarization: This step determines the topology of the drawing, which is described by a planar embedding. If a graph has a drawing in the plane without any edge crossings, it is planar. Such a drawing divides the graph plane into faces. A planar embedding is a combinatorial description of the faces and contains for each face the sequence of edges contouring it. For graphs which are not planar dummy vertices are introduced which represent crossings to make the resulting graph planar. The algorithm tries to minimize the number of crossings **[FLOW + CROSSING]**.

Orthogonalization: This step determines the angles and the bends in the drawing. Only multiples of 90° are assigned as angles which ensure that the drawing is orthogonal. The algorithm tries to minimize the number of bends in this step. **[ORTHOGONAL + BEND]**

Compaction: In this step the final coordinates are assigned to the nodes and to the edge bends. The dummy vertices introduced in the planarization phase are removed. In this phase the main goal is to minimize the sum of the length of all edges and/or the area of the drawing **[EDGELENGTH + AREA]**.

Postprocessing:

(a) All dummy vertices are removed from the graph such as crossings.
(b) The connected components of the graph after layout process are arranged by a floor planning algorithm.

4.2 Mixed Upward Planarization

In this section we consider the problem of finding a planarization of a mixed graph for which a drawing exists in which all directed edges are represented by monotonically increasing curves and which has a low number of crossings at the same time **[FLOW + CROSSING]**.

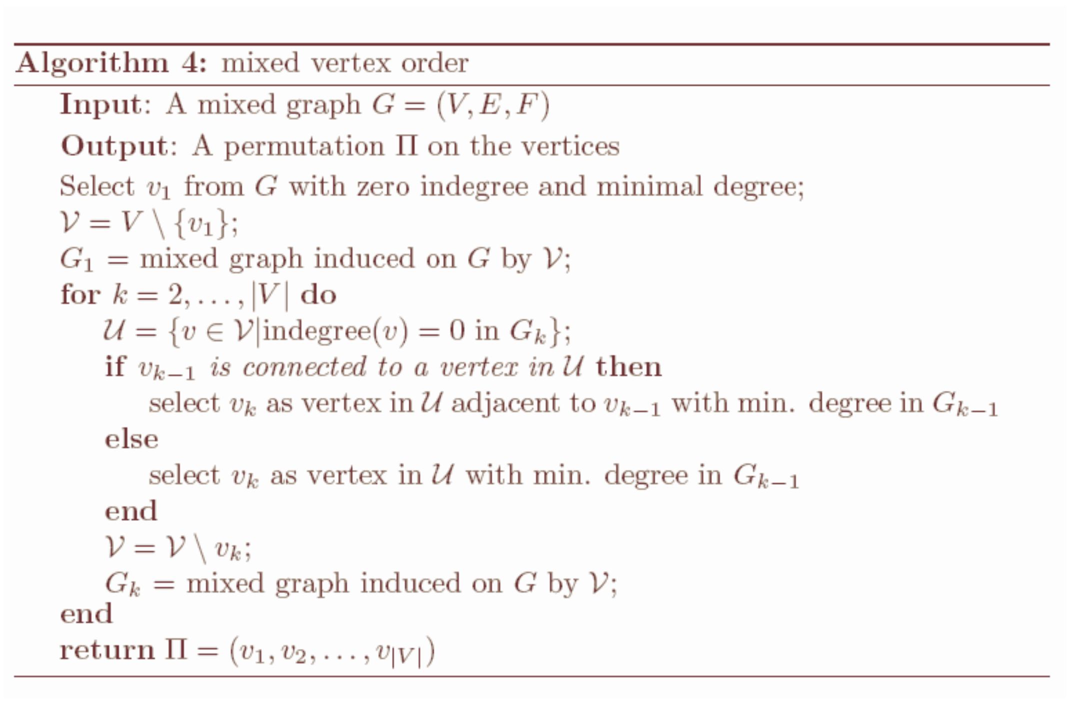
The algorithm is based on a heuristic which is a popular technique for the planarization of undirected graphs:

1. Construct an upward planar subgraph.
2. Determine an upward embedding of this subgraph.
3. Insert the edges not contained in the subgraph, one by one.

4.2.1 Maximum Upward Planar Subgraph

The maximum upward planar subgraph problem can be stated as follows: Given a directed graph $G=(V,E)$, find $E' \subseteq E$ such that the directed graph $G=(V,E)$ is upward planar with maximum number of edges.

The Mixed Vertex Ordering algorithm is a variant for mixed graph of the **Goldschmidt/Takvorian [6]** algorithm, which makes an undirected graph planar and be divided into two phases. The first phase of GT consists of devising an ordering Π of the set of vertices *V* of the input graph *G*, such that as many edges as possible between adjacent vertices can also be placed on the line. The second phase of GT partitions the edge set *E* of *G* into subsets *L* (left of the line), *R* (right of the line) and *B* (the remainder) in such a way that $|L+R|$ is large as maximal possible an that no two edges both in *L* or both in *R* cross with respect to the sequence Π devised in the first phase.



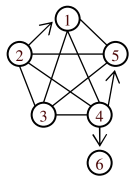
```
Algorithm 4: mixed vertex order
  Input: A mixed graph $G = (V, E, F)$
  Output: A permutation $\Pi$ on the vertices
  Select $v_1$ from $G$ with zero indegree and minimal degree;
  $\mathcal{V} = V \setminus \{v_1\}$;
  $G_1$ = mixed graph induced on $G$ by $\mathcal{V}$;
  for $k = 2, \ldots, |V|$ do
      $\mathcal{U} = \{v \in \mathcal{V} | \text{indegree}(v) = 0 \text{ in } G_k\}$;
      if $v_{k-1}$ is connected to a vertex in $\mathcal{U}$ then
          select $v_k$ as vertex in $\mathcal{U}$ adjacent to $v_{k-1}$ with min. degree in $G_{k-1}$
      else
          select $v_k$ as vertex in $\mathcal{U}$ with min. degree in $G_{k-1}$
      end
      $\mathcal{V} = \mathcal{V} \setminus v_k$;
      $G_k$ = mixed graph induced on $G$ by $\mathcal{V}$;
  end
  return $\Pi = (v_1, v_2, \ldots, v_{|V|})$
```

Figure 4.1: Mixed Vertex Ordering Algorithm, taken from **[6]**.

With the algorithm of Asano, Imai and Mukaiyama **[5]** a maximum independent set of an overlap graph can be calculated in time *O(NM)*, where *N* is the number of different interval endpoints and *M* is the number of edges in the overlap graph. In our setting, $N \leq n$ and $M \leq m$, which leads to a running time of *O(nm)*. Since this algorithm computed the maximum independent set, the algorithm proposed in this section can compute planar subgraph.

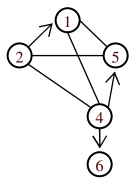
In Figure 4.2 we show a run of the mixed vertex order algorithm. As input graph we take the minimal non-planar graph K_5, in order to make the problem trivial we add a new vertex 6 and a directed edge from vertex 4 point to the new vertex 6.

$G =$



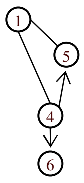
$v_1 = \{3\}$
$\mathcal{V} = \{1, 2, 4, 5, 6\}$

$G_1 =$



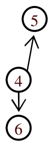
$\mathcal{U} = \{2, 4\}$
$v_2 = \{2\}$
$\mathcal{V} = \{1, 4, 5, 6\}$

$G_2 =$



$\mathcal{U} = \{1, 4\}$
$v_3 = \{1\}$
$\mathcal{V} = \{4, 5, 6\}$

$G_3 =$



$\mathcal{U} = \{4\}$
$v_4 = \{4\}$
$\mathcal{V} = \{5, 6\}$

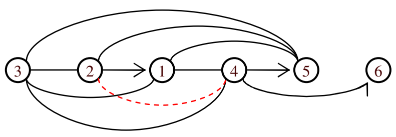
$G_4 =$

$\mathcal{U} = \{5, 6\}$
$v_5 = \{5\}$
$\mathcal{V} = \{6\}$

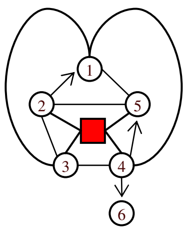
$G_5 =$

$\mathcal{U} = \{6\}$
$v_6 = \{6\}$
$\mathcal{V} = \varnothing$

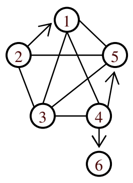
Ordering Π = (3, 2, 1, 4, 5, 6)



L= {(3, 5), (2, 5), (1, 5), (3, 2), (2, 1), (1, 4), (4, 5)}
R= {(3, 1), (3, 4), (4, 6)}
B= {(2, 4)}



A planar drawing of the input graph *G* by introducing a dummy vertex red depicted. See section Edge Insertion.



Maximum Upward Planar Subgraph

Figure 4.2: A run of the mixed vertex ordering algorithm.

From the set L and R and the ordering Π, we can now easily obtain the mixed upward embedding: For each vertex $v \in V$ we sort the edges with source v in R decreasing according to Π and the edges with source v in L increasing according to Π, and concatenate these two ordered lists to one. For the incoming edges, we first sort the edges with target v in L increasing according to Π and the edges with source v in R decreasing according to Π and append the result to the list of outgoing edges.

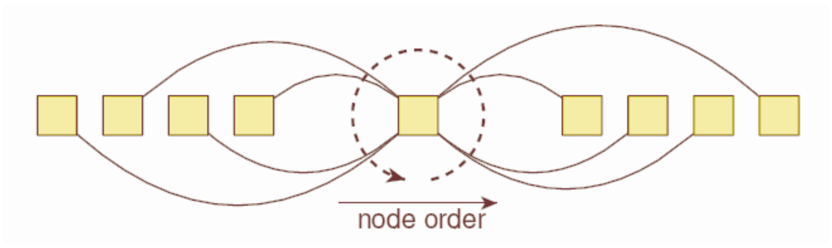
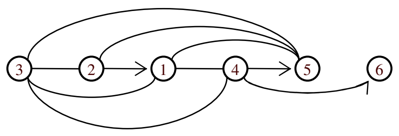


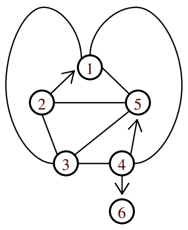
Figure 4.3: The order of the edges at a vertex can be derived directly from Π and the sets L and R **[5]**.

The embedding ε of the example runs in figure 4.2:



L= {(3, 5), (2, 5), (1, 5), (3, 2), (2, 1), (1, 4), (4, 5)}
R= {(3, 1), (3, 4), (4, 6)}

ε (3) = {(3, 4), (3, 1), (3, 2), (3, 5)}
ε (2) = {(2, 1), (2, 5), (2, 3)}
ε (1) = {(1, 4), (1, 5), (1, 2), (1, 3)}
ε (4) = {(4, 6), (4, 5), (4, 1), (4, 3)}
ε (5) = {(5, 3), (5, 2), (5, 1), (5, 4)}
ε (6) = {(6, 4)}



Maximum Upward Planar Subgraph

4.2.2 Edge Insertion

Insertion of Undirected Edges

In this section we consider the problem of inserting an undirected edge into an embedded graph. An algorithm for insertion of undirected edges is based on the *dual graph* of the plane graph. The dual graph G^* of plane graph $G = (V,E,F)$ has a vertex for each face of G, and an edge $d(e) = (f,g)$ between two faces f and g for each edge e that is shared by f and g. To insert an edge (a,b), the *extended dual graph* $G_{(a,b)}^*$ will be constructed from G^* by adding two vertices a' and b' to G^* and inserting an edge from (a', f), resp. (b', f), into G^* for each f which contains an edge adjacent to a', resp. b' according to a, b. From each path $e_0, \ . \ . \ . \ , e_k$ from a' to b' in $G_{(a,b)}^*$ we can obtain a planarization of G by subdividing the edges in G corresponding to $e_1, \ . \ . \ . , e_{k-1}$ and adding a path from a to b which uses the vertices introduced by the subdivision. A shortest path from *a'* to *b'* induces therefore a planarization of $G \cup (a, b)$ with minimal crossing number without changing the embedding of *G*. The algorithm is illustrated in Figure 4.4.

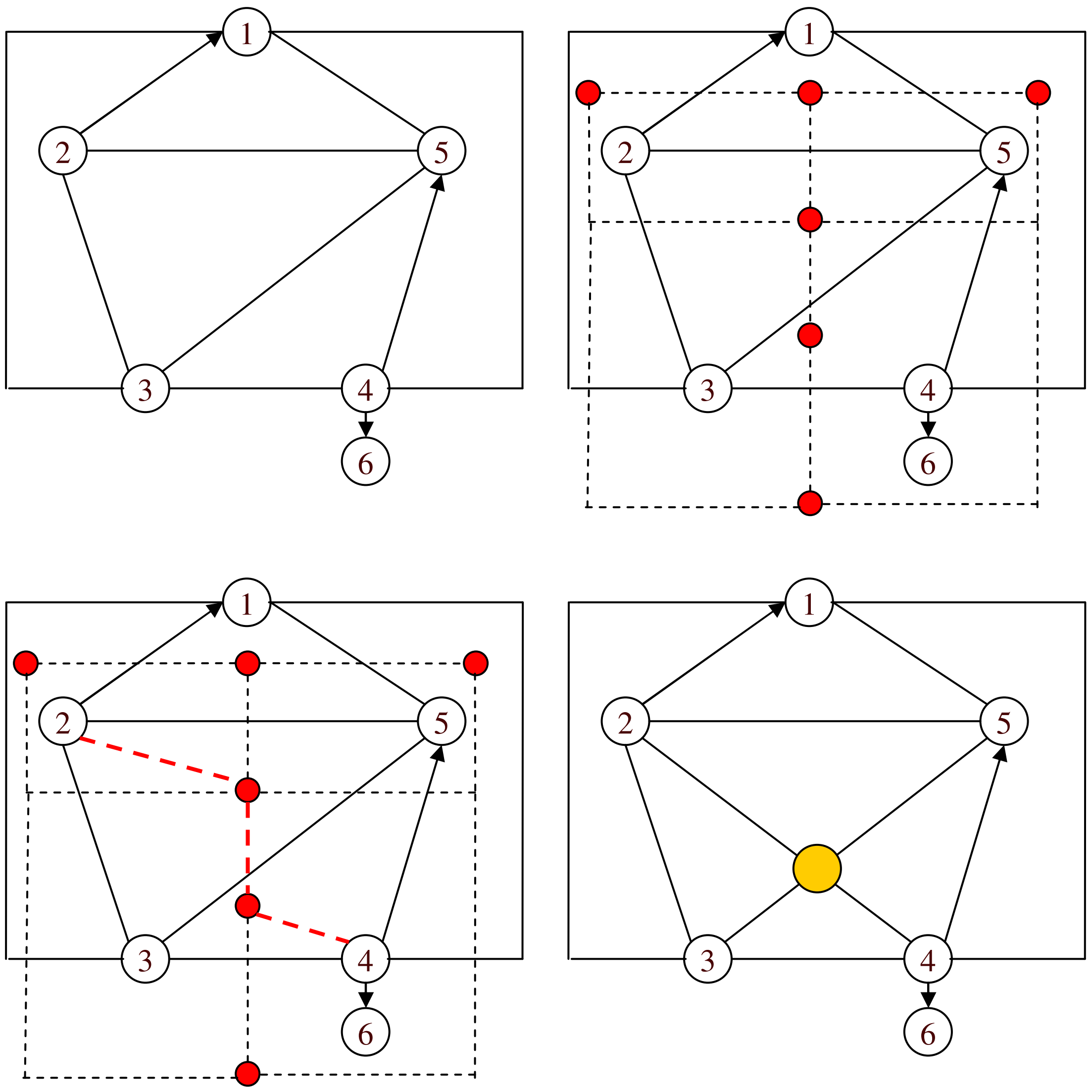


Figure 4.4: Insertion of edge *(1,3)* into the plane graph shown in upper left. The dual graph *G** is depicted in upper right and the extended dual graph $G_{(1,3)}^{*}$ in under left. The shortest path from vertex 1 to vertex 3 illustrated by the thick dashed red edges. The resulting planarization is shown in under right.

Insertion of directed edges

Inserting directed edges into an upward embedded directed graph can be much more complex than inserting undirected edges in a embedded graph. Since we cannot easily insert edge into the drawing without considering the remaining edges which will be inserted later. The reason for this is that the dummy vertices added to the graph in a planarization step introduce changes in the ordering of the vertices of the graph. Although the graph remains acyclic, this may introduce a directed cycle with an edge which will be added later. Then this edge can no longer be inserted. We show here an example to illustrate this critical configuration.

In Figure 4.5(a) the dashed edges will be inserted into the drawing, and we first insert edge (5, 6). If we insert edge (5, 6) as in Figure 4.5(b), we produce a dummy vertex in edge (1,3).

Then, it is no longer possible to introduce edge (3, 2) without destroying the upward property because of the new directed cycle (5, Dummy, 3, 2, 5).

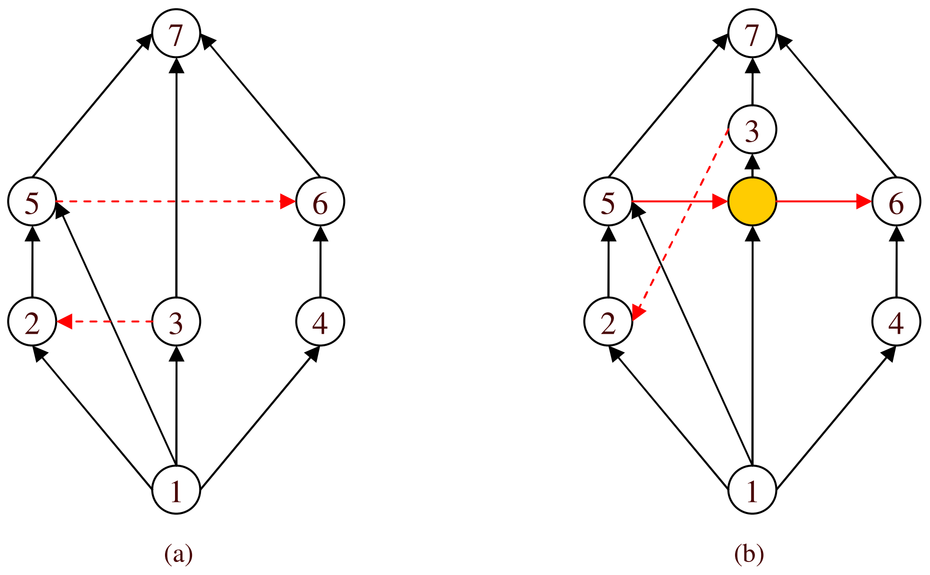


Figure 4.5: A critical configuration.

As shown above cycles must be avoided when inserting directed edges. This will be achieved by *layering* the graph. A valid layering *l* is defined as a mapping of *V* to integers such that $l(v) > l(u)$ for each edge $(u,v) \in E$. Computing a layering for a directed acyclic graph has been studied extensively, for more details refer to **[5]**. An longest path layering method, which calculates a layering of minimal height in time $O(n+m)$ is used.

From the layered graph we construct the routing graph $R_{(a,b)}$ for the insertion of a directed edge (a, b). The routing graph contains, for each face f and for each layer i that f spans, a vertex $\upsilon(f, i)$. Two vertices lying in neighbouring layers and representing the same face are connected by a directed edge of weight 0 in increasing layer order. Additionally, two vertices at the same layer i of adjacent faces are connected by an edge if the source vertex of an edge separating these two faces is less than or equal to i and the layer of the target vertex is greater than i. We assign to this edge the product of the weight of the edge separating the faces and the weight of edge (a, b).

In this graph, there are no edges in decreasing layer order, in other words for each edge $(\upsilon(f, i), \upsilon(g, j))$ holds $i \leq j$. Each edge of positive weight represents one crossing. A shortest path in the routing graph represents, therefore, an insertion of an edge with minimal weighted crossing number with respect to the given layering.

We add a vertex $\upsilon(a)$ to the routing graph and connect it to faces which are adjacent to outgoing edges of a. Analogously we add a vertex $\upsilon(b)$ to the routing graph and connect it to faces which are adjacent to incoming edges of b. A path from $\upsilon(a)$ to $\upsilon(b)$ corresponds to a valid routing of the edge (a, b). Figure 4.6(b) shows an example for a routing graph.

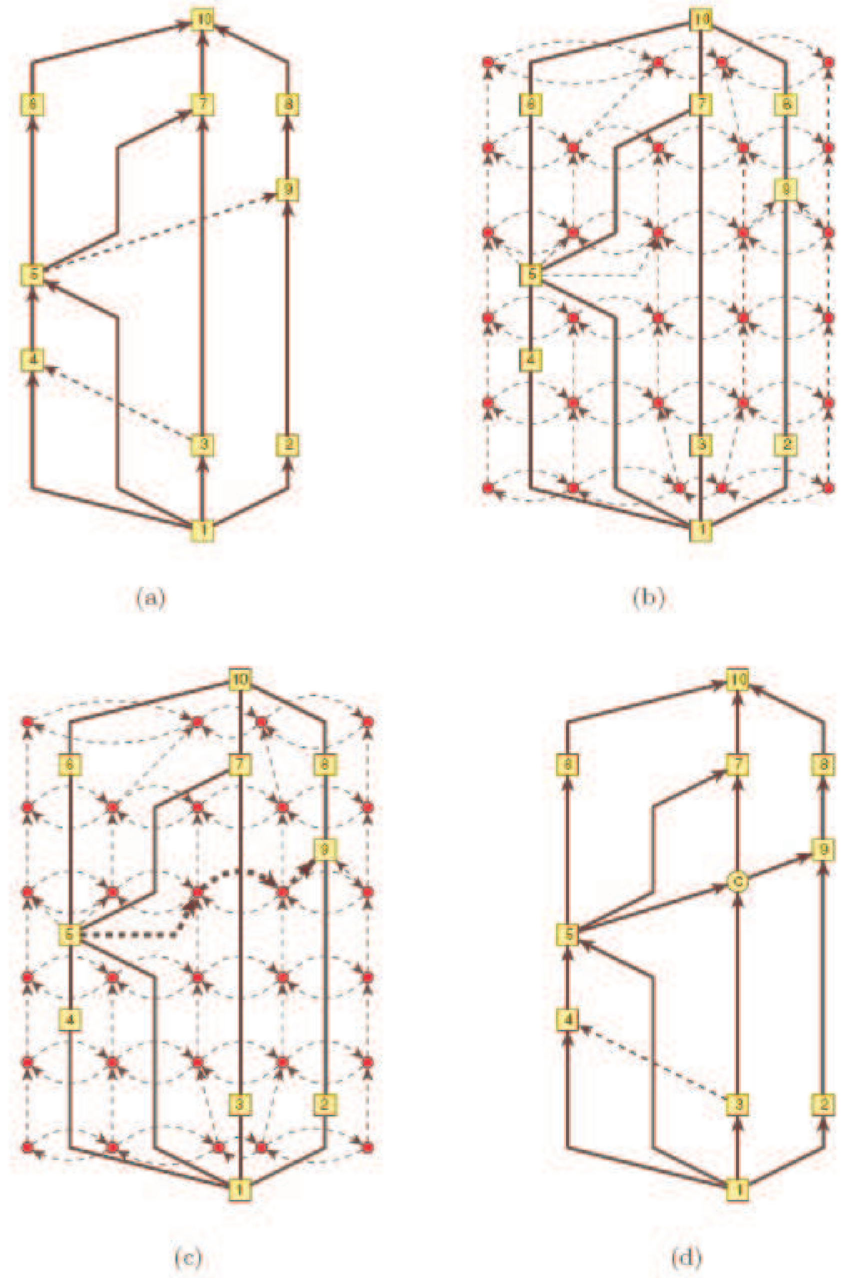


Figure 4.6: Example for algorithm directed edge insertion. Figure (b) shows a valid layering of the input graph (a), taken from **[5]**.

4.3 Orthogonalization

This section describes orthogonalization algorithms which try to minimize the number of bends, notably which vertices have prescribed size and directed edges point upward. The result of the algorithm is an orthogonal shape which is computed from a mixed upward planarization. The **KANDINSKY** algorithm is an extension of **Tamassia**'s algorithm that computes bend-minimal point drawings of plane 4-graphs. The **KANDINSKY** algorithm overcomes the severe restriction of **Tamassia**'s algorithm that it is restricted to point drawing which means available only for 4-graphs. The **KANDINSKY** algorithm is also the only one that guarantees prescribed vertex sizes. We review the **Tamassia's** Algorithm in detail, for further reading of the **KANDINSKY** algorithm, we refer to **[6]**.

4.3.1 Tamassia's Algorithm

Tamassia's algorithm computes an orthogonal shape of a plane 4-graph with respect to an input embedding with a minimal number of bends. Calculating a drawing from an orthogonal shape is called compaction and it will be discussed in **[sec 4.4]**.

This problem can be solved using Min-Cost-Flow **[sec 2.3]** network algorithm:

For each plane graph *G* there is a minimum cost flow network N_G in which there is a one-to-one correspondence between valid flows in the network and valid orthogonal shapes of *G*. The cost of a network flow corresponds to the number of bends in the induced orthogonal shape, and there fore a bend-minimal orthogonal shape can be computed with a Min-Cost-Flow solving algorithm.

The minimum cost flow network $N_G=(N,A)$ for a plane 4-graph $G=(V,E,F)$ is a minimum cost flow network and is defined as follows:

- The set of nodes *N* is defined as $N=N_V \cup N_F$ with
  1. The set N_V contains a node for each vertex in *V*.
  2. The set N_F contains a node for each face in *F*.

- The set of arcs *A* is defined as:
  1. The set (s, n_v), where $n_v \in N_V$, connects source *s* with every *v*; this set of arcs have cost zero and capacity 4-δ (n_v). (Compare **Figure 4.7(b)**)
  2. The set (s, n_f), where $n_f \in N_F$, *f* is an inner face with δ $(n_f) \leq 3$; this set of arcs have cost zero and capacity 4-δ (f). (Compare **Figure 4.7(c)**)
  3. The set (n_f, t), where $n_f \in N_F$, f is either the outer face or an inner face with δ $(n_f) \geq$ 5; this set of arcs have cost zero and capacity δ (f)+4 for outer face or δ (f)-4 else. (Compare **Figure 4.7(d)**)
  4. The set (n_v, n_f), where $n_v \in N_V$, $n_f \in N_F$; this set of arcs have cost zero and capacity ∞. (Compare **Figure 4.7(e)**)
  5. The set (n_f, n_g), where $n_f, n_g \in N_F$, and the face *f* and *g* have at least one common edge; this set of arcs have cost one and capacity ∞. (Compare **Figure 4.7(e)**)

$$\text{Flow } |f| = \sum cap(s,n_v) = \sum cap(n_t,t).$$

A proof of the above equation is the Euler's formula, see **[21]**.

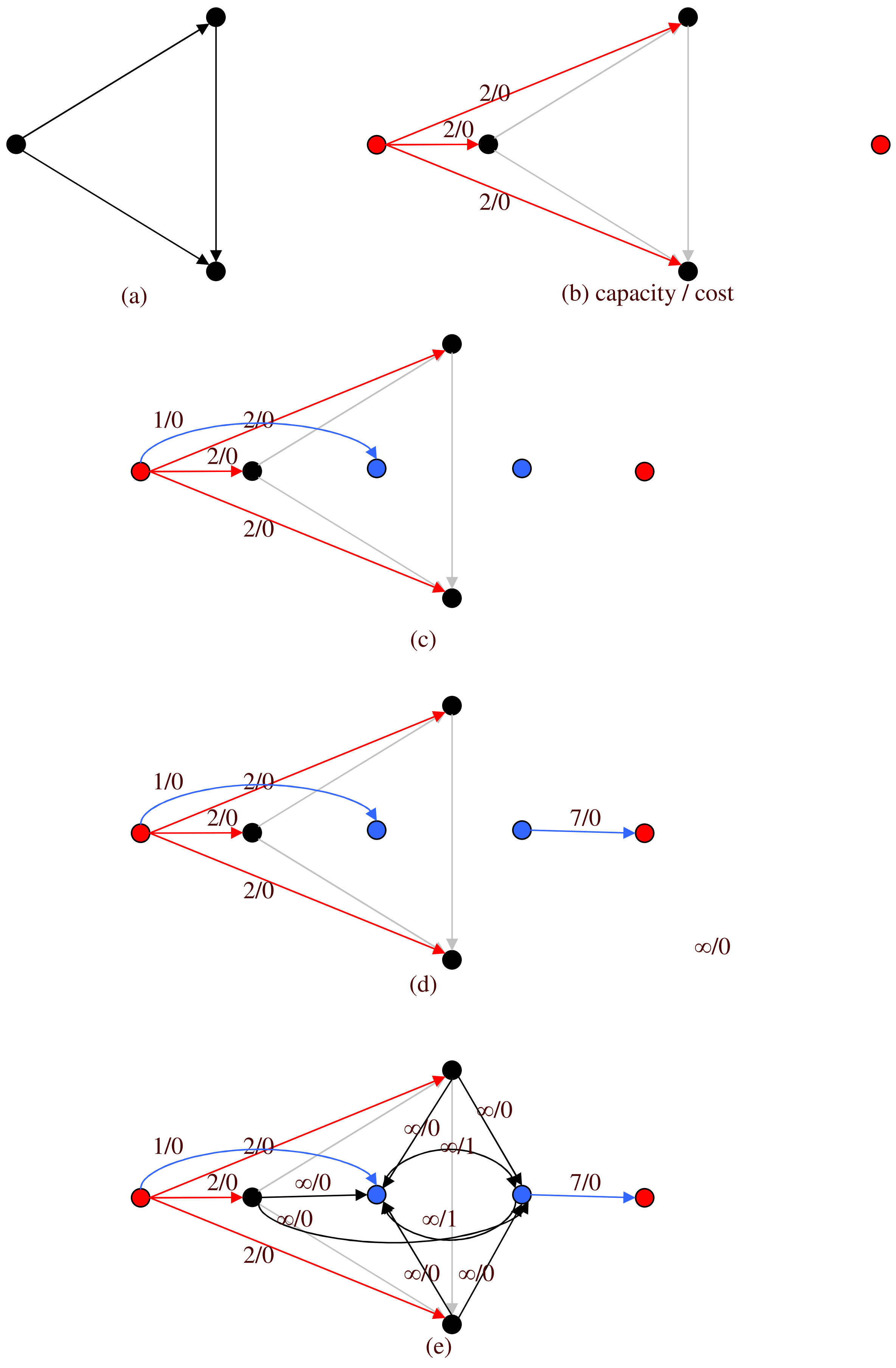


Figure 4.7: Min-Cost-Flow st-settings. (a) is the input graph.

We can solve this simple Min-Cost-Flow problem with inspection:

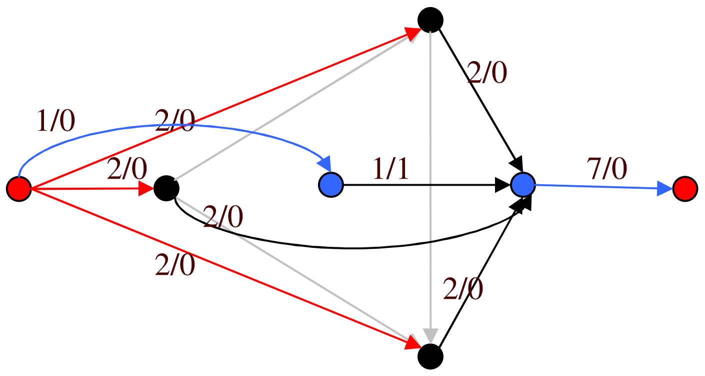
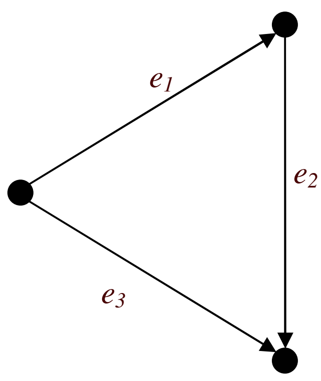


Figure 4.8: Min-Cost- Flow of the input graph.

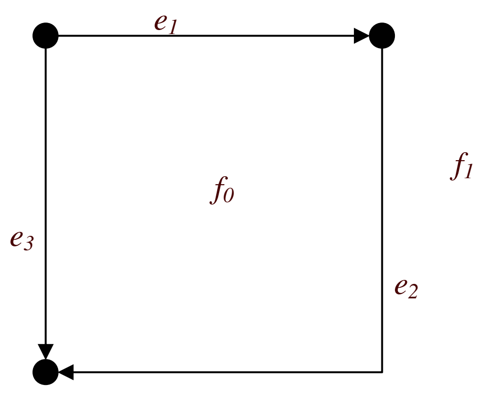
A feasible flow f = 7 and cost = 1 are foreseeable. For more complex Min-Cost-Flow problems there are different algorithms with best running time $O(n^2 \log n)$ **[13].**

The flow f and the cost can be interpreted as:

- Each flow value over an edge (n_f, n_g) represent a bend in an arbitrary edge, which connects the two faces *f* and *g*.
- A flow value *x* over an edge (n_v, n_f) defined an angle with $(x+1) \cdot 90°$ between the both edges incident by node *v*.



(a)



(b)

$H(f_0) = \{(e_1, \varepsilon, 270°), (e_2, 0, 180°), (e_3, \varepsilon, 270°)\}$,
$H(f_1) = \{(e_3, \varepsilon, 270°), (e_2, 1, 180°), (e_1, \varepsilon, 270°)\}$.

Figure 4.9: The input graph (a) and its orthogonal shape (b)

We summarize the above discussion as follows: Given a plane graph *G* and a feasible flow *x* in the bend-minimization network, we can define an orthogonal shape *H*(*G*) for *G*. In this phrase, we calculate an orthogonal shape for an input planar graph.

4.3.2 Generalizations of Tamassia's Algorithm Using Reduction

One possibility to generalize Tamassia's Algorithm to orthogonal rectangle drawing of graphs with arbitrarily degree is the *reduction approach*. The idea behind this approach is not to change the algorithm, rather change the input to accommodate the restriction of Tamassia's Algorithm with 4-graph. The reduction approach is based on the observation, that we can often overcome the limitations of an algorithm which requires a special type of input by transforming the input, so that it fulfils these requirements, and then perform the algorithm on the transformed input. We can then obtain a solution for the original input by interpreting the algorithms result. In an object-oriented sense this is captured by the *Algorithmic Reduction* design pattern.

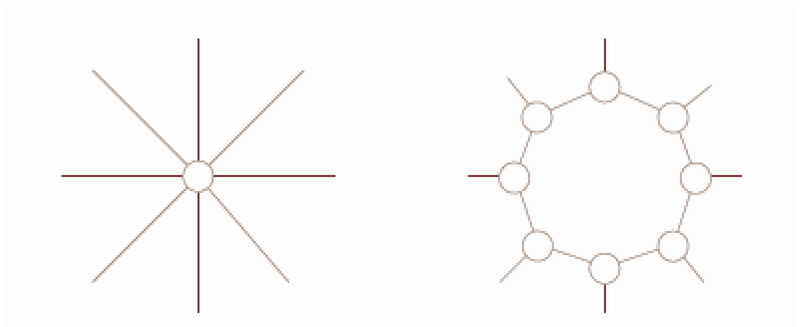


Figure 4.10: Transformation of a high-degree vertex into a face, taken from **[6]**.

4.4 Compaction

In this section we briefly introduce the compaction algorithm. We first describe the constructive techniques which produce a drawing for a given orthogonal shape *H*. The key idea behind these methods is to transform *H* into an auxiliary representation $H^{'}$ by introducing artificial edges and vertices and to find a drawing for $H^{'}$ in polynomial time. Removing the artificial vertices and edges in the auxiliary drawing leads to a drawing for *H*.

By the computation of edge length the following points must be considered:

- Length of all line segments must be positive integer.
- Each cycle of the graph will be mapped as a polygon.
- There are no segments, which overlaps with themselves, unless they have common end points.

For more details on compaction of orthogonal drawing see **[6] [22]**.

4.4.1 Constructive Heuristic

Tamassia mentions the first and still most common method to produce an auxiliary representation $H^{'}$. He introduces the *dissection method* which consists of decomposing each internal face of the given simple orthogonal shape *H* into to a set of faces each of which has rectangular shape by introducing artificial vertices and edges. Figure 4.11 illustrates this method with an example. Please note that the method works at representation level, the coordinates have not yet been assigned. This process can be done in O(*n*) time where *n* denotes the number of vertices in *H*. In the resulting orthogonal shape $H^{'}$, all interior faces have rectangular shape. Of course, the artificial vertices and edges impose additional constraints on the geometry which may lead to suboptimal total edge length and are in the resulting drawing.

With the help of decomposing the faces into rectangles we can describe the set of orthogonal shapes, for which optimal compactions can be calculated in polynomial time.

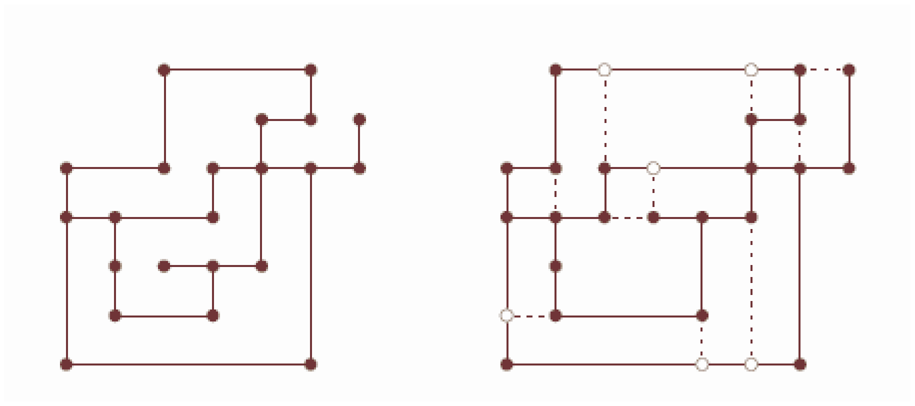


Figure 4.11: The dissection method. On the left is the original orthogonal shape H. On the right is the transformed auxiliary representation $H^{'}$. Dashed lines and empty vertices represent artificial edges and vertices. Taken from **[22]**.

There are also other effective heuristics for construct an auxiliary representation, for further reading we refer to **[22]**.

4.4.2 Optimal Compaction

An ILP-based approach is used to solve instances of the two-dimensional compaction problem to optimality. It is based on a characterisation of the set of feasible solutions in terms of paths in the pair of constraint graphs. Given a pair of constraint graphs in which only the relative positions known from the shape H are present, the compaction problem can be seen as optimising over the set of certain extensions of these graphs.

The complete compaction algorithms for prescribed vertex-size **KANDINSKY** shape see also **[6][22]**.

Chapter 5

Conclusion

In this last Chapter we draw a conclusion of the presented work and show same sample diagram layout using the Topology-Shape-Metrics approach.

5.1 Conclusions

In this work we presented state-of-the-art techniques for the automatic layout of graphs using the topology-shape-metrics approach which yield automatic layout for EMF diagrams.

Orthogonal graph drawing is getting increasing attention from industry because of its numerous applications, *e.g.*, in database design, software engineering and many more. For many of these applications, the Topology-Shape-Metrics approach leads to the best results. In this approach, a first phase (planarization) determines the topology of the drawing. The aim is to generate a drawing with a small number of edge crossings, *e.g.*, by computing a large planar subgraph and carefully reinserting the removed edges. Then, the crossings are replaced by artificial vertices resulting in a planar graph. A second phase determines the shape of the final layout. This phase is often restricted to planar orthogonal drawings (orthogonalization). A widely accepted optimization criterion is to minimize the number of bends without changing the topology. Finally, the third phase (compaction) deals with computing the metrics of the layout. Here, the optimization problem is to compute a layout of minimum area with minimum total edge length.

Although the Topology-Shape-Metrics approach was suggested already in 1984, it took some years until the approach was getting popular. The reason for this is that it is time consuming to implement the approach, since many different kinds of algorithms are needed. Recently, a lot of research has been done to solve many of the upcoming problems. Major improvements have been made concerning various aspects of the three phases of the Topology-Shape-Metrics approach. Moreover, today there exist some software libraries containing the Topology-Shape-Metrics approach **[18]**.

5.2 Sample Diagram Layout Using Topology-Shape-Metrics Approach

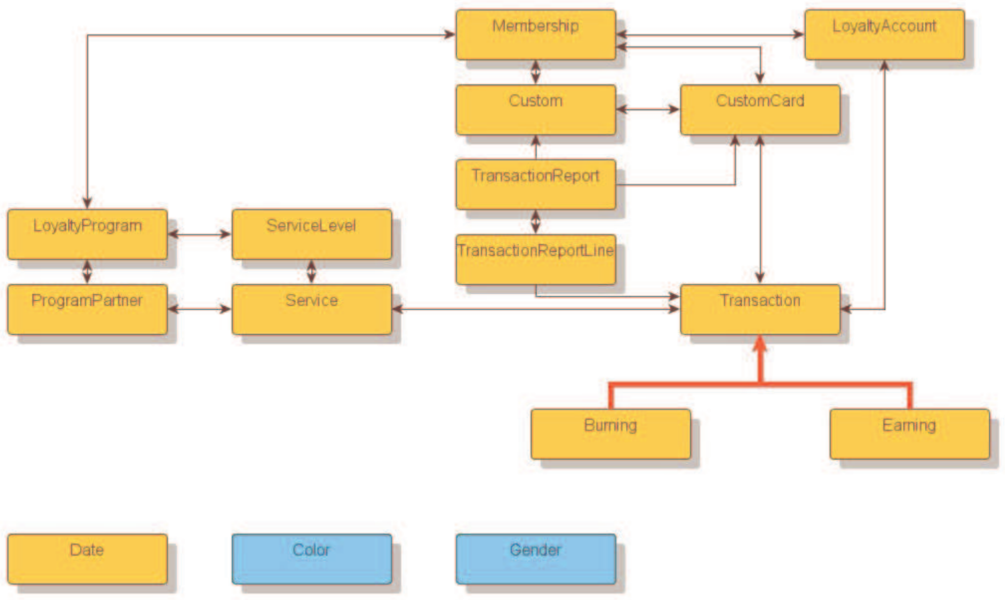


Figure 5.1: The Royal and Loyal EMF Diagram see also **[Figure 1.1]**.

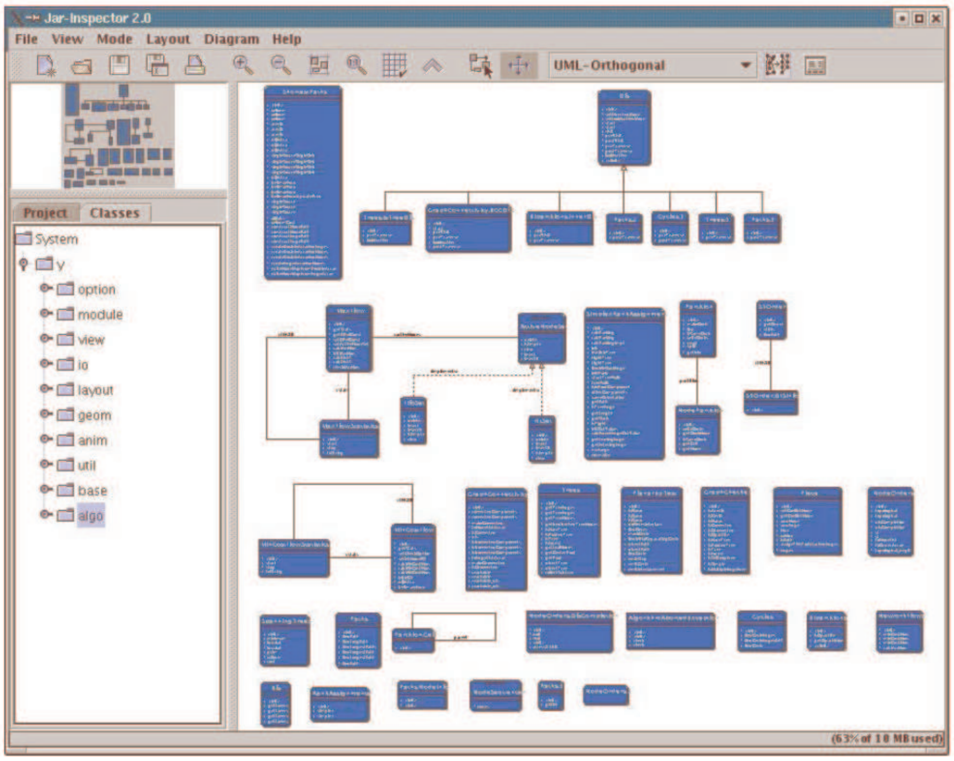


Figure 5.2: An UML class diagram depicted using *JarInspector*, which also based on the Topology-Shape-Metrics Approach, screenshot taken from **[12]**.

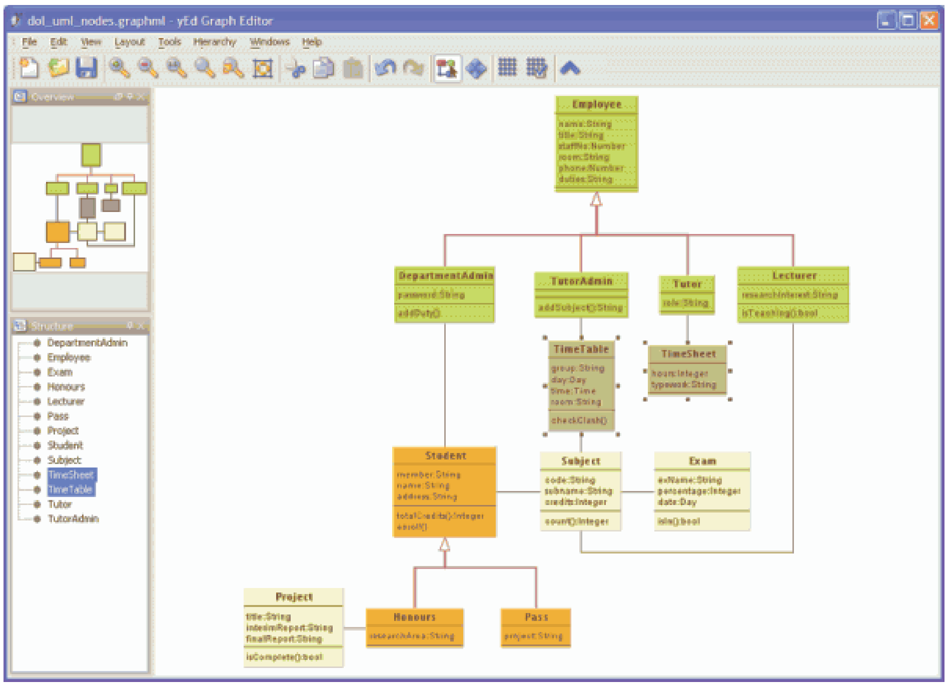


Figure 5.3: "yEd", a Java graph editor (http://www.yworks.com).

5.3 Issues Not Concerned in This work

Some themes were not coved in this work, such as prescribed size orthogonalization, hyperedges, label placement, *etc*. For detailed information about these issues we recommend **[5][6][12][13]**.

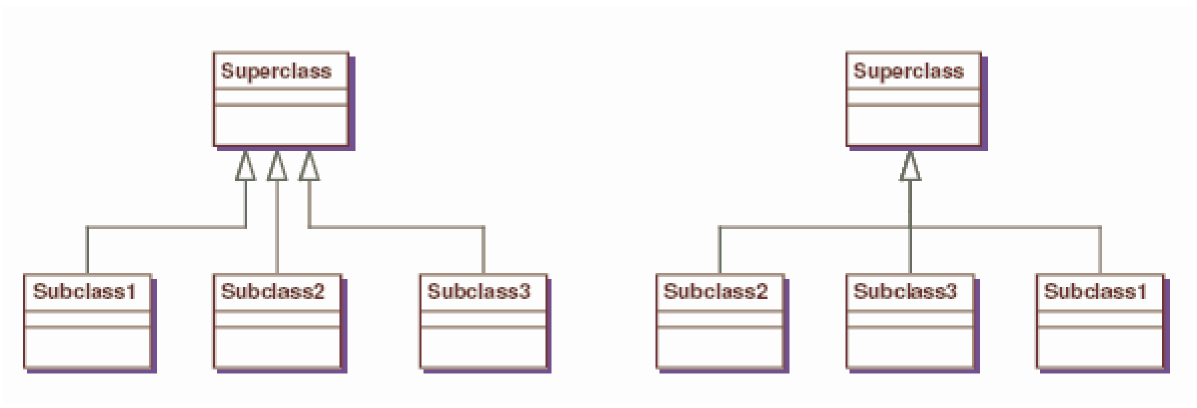


Figure 5.1: Generalization with distinct paths and hyperedge notation, taken from **[6]**.

Bibliography

[1] Sherry Shavor, Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, Pat McCarthy (2003). *The Java™ Developer's Guide to Eclipse*, Addision-Wesley.

[2] Jeff McAffer, Jean-Michel Lemieux (2005). *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications,* Addision Wesley Professional.

[3] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, Timothy J. Grose (2003). *Eclipse Modeling Framework: A Developer's Guide*, Addision-Wesley.

[4] Randy Hudson (2006). *Create an Eclipse-based application using the Graphical Editing Framework*, IBM developerWorks article.

[5] Markus Eiglsperger (2003). *Automatic Layout of UML Class Diagrams A Topoly-Shape-Metrics Approach.*

[6] Markus Eiglsperger, Frank Eppinger, Michael Kaufmann (2003), *An Approach for Mixed Upward Planarization*.

[7] Robert Cimikowske, *An Analysis of Some Heuristics for the Maximum Planar Subgraph Problem*.

[8] Holger Eichelberger, *Demostration of Advanced Layout of UML Class Diagrams by SugiBib*.

[9] Bernhard Tatzmann, *Dynamic Exploration of Large Graphs*.

[10] Carsten Gutwenger, Michael Juenger, Karsten Klein, Joachim Kupke, Sebastian Leipert, Petra Mutzel, *GoVisual A New Approach for Visualizing UML Class Diagrams*.

[11] Denis Dube (2006), *Graph Layout for Domain-Specific Modeling*.

[12] Markus Eiglsperger, Michael Kaufmann, Martin Seibenhaller, *A Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams*.

[13] Martin Siebenhaller (2003), *Automatisches Layout von UML-Klassendiagrammen*.

[14] Karlis Freivalds, Ugur Dogrusoz, Paulis Kikusts (2001), *Disconnected graph layout and the polyomino packing approach*.

[15] Ulrik Brandes, Markus Eiglsperger, Michael Kaufmann, Dorothea Wagner, *Sketch-Driven Orthogonal Graph Drawing*.

[16] Eclipse Modeling Framework project, http://www.eclipse.org/modeling/emf/.

[17] Emfatic Language for EMF Development, http://www.alphaworks.ibm.com /tech/emfatic.

[18] AGD Library User Manuel, http://www.ads.tuwien.ac.at/AGD/MANUAL/ MANUAL.html.

[19] Methods for Visual Understanding of Hierarchical System Structures, http://plg.uwaterloo.ca/~itbowman/CS746G/Notes/Sugiyama1981_MVU/.

[20] Geir Agnarsson, Raymond Greenlaw, *Graph theory: modeling, applications, and algorithms*.

[21] Reinhard Diestel (2006), *Graphentheorie*.

[22] Gunnar W. Klau, Karsten Klein, Petra Mutzel, *An Experimental Comparison of Orthogonal Compaction Algorithms*.